UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Meta-Learning applied to Neural Architecture Search. Towards new interactive learning approaches for indexing and analyzing images from expert domains

Gean Trindade Pereira

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Gean Trindade Pereira

Meta-Learning applied to Neural Architecture Search. Towards new interactive learning approaches for indexing and analyzing images from expert domains

Thesis submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP / La Rochelle Université – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisors: Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho, Prof. Dr. Muriel Visani (Cotutelle)

USP – São Carlos / La Rochelle Université – La Rochelle
April 2024

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

T833m

Trindade Pereira, Gean
Meta-Learning applied to Neural Architecture Search. Towards new interactive learning approaches for indexing and analyzing images from expert domains / Gean Trindade Pereira; orientador André Carlos Ponce de Leon Ferreira de Carvalho; coorientadora Muriel Visani. -- São Carlos, 2024.
218 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. Neural Networks. 2. Neural Architecture Search. 3. Meta-Learning. 4. Convolutional Neural Networks. 5. Computer Vision. I. Ponce de Leon Ferreira de Carvalho, André Carlos, orient. II. Visani, Muriel, coorient. III. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Gean Trindade Pereira

Meta-Aprendizado aplicado à Busca de Arquitetura Neural. Rumo à novas abordagens de aprendizagem interativa para indexação e análise de imagens de domínios especializados

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP / La Rochelle Université, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadores: Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho, Prof. Dr. Muriel Visani (Cotutelle)

USP – São Carlos / La Rochelle Université – La Rochelle
Abril de 2024

Gean Trindade Pereira

Méta-Apprentissage appliqué à la Recherche D'Architecture Neuronale – Vers de nouvelles approches d'apprentissage interactif pour faciliter l'indexation et l'analyse d'images de domaines experts

Thèse soumis à la Institut de mathématiques et d'informatique – ICMC-USP / La Rochelle Université – conformément aux exigences de la Programme d'études supérieures en informatique et en sciences mathématiques, pour le diplôme de docteur en science. *VERSION FINALE*

Zone de concentration: Informatique et mathématiques computationnelles

Conseillers: Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho, Prof. Dr. Muriel Visani (Cotutelle)

USP – São Carlos / La Rochelle Université – La Rochelle
April 2024

Dedico esta tese à minha família, tanto à de sangue quanto à escolhida ao longo da vida. Sem vocês, este trabalho e eu não estaríamos aqui.

ACKNOWLEDGEMENTS

A geração dessa tese só foi possível graças à colaboração e suporte de muitas pessoas queridas. Primeiramente, gostaria de agradecer aos meus pais, Tânia e Glênio, e minhas irmãs, Gabriele e Géssica, não podendo esquecer também das novas integrantes da família, os bebêzinhos Mione e Melzinha. Vocês foram e sempre serão o suporte estrutural de tudo pra mim.

Esse trabalho também não poderia ter sido realizado sem o apoio dos meus orientadores. Agradeço ao professor André Carlos Ponce de Leon Ferreira de Carvalho por ter me dado a oportunidade de ingressar no doutorado sob sua orientação, e a Muriel Visani e Thierry Urruty pelo convite ao doutorado de duplo diploma. Em paralelo, agradeço ao ICMC/USP, seus professores e demais funcionários, e toda a estrutura da universidade que forneceram não só um local de trabalho como uma verdadeira casa durante os anos que desenvolvi minha pesquisa. Não sei o que seria de mim sem o famoso bandejão, as quadras de volêi, e o jardim secreto. Também, sou muito grato a Poitiers Université e o laboratório XLIM que me acolheu durante minha estadia na França, bem como a La Rochelle Université e o laboratório L3i.

Agradeço também aos queridos colegas do Biocom que viraram amigos e uma segunda família: Zé Pedro, Iury, Moisés, Saulo, Barella, Ângelo, Padula, Luís Paulo, Angêlica, e Marília. Também não poderia esquecer dos amigos que fiz do outro lado do Atlântico, pessoas que me acolheram em um lugar onde eu não conhecia nem a cultura, nem as pessoas. Meus queridos amigos Thibault, Clément, Damien, Adrien, Maugan, Elza, André, Safaa, Inês, Alex, Ruqin, Charles, Asali, David, Marouane, Lucas, Sylvan, Guillaume, e Arthur.

Também agradeço a Juliana, e os bebêzinhos Emizinha, Shoyu, e Alicinha. Sinto saudades e nunca esqueço de vocês.

Aos meus queridos amigos de mestrado também presto agradecimentos, Thiago manauara, Eduardinho, e Gerson. Vocês vem me salvando desde o começo dessa caminhada arriscada em ML e permanecem os mesmos, mesmo depois de muito tempo. Obrigado.

Agradeço também os meus amigos de infância mas que permanecem até hoje, nas jogatinas de final de semana e encontros anuais, Luciano (gugu) e Gustavo (coto).

Por último mas não menos importante, gostaria de agradecer o apoio das agências de fomento pelo financiamento total ou parcial desta tese, em especial a Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo 2019/19994-2, e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Life is suffering. It is hard. The world is cursed. But still, you find reasons to keep living
— Hayao Miyazaki

ABSTRACT

PEREIRA, G. T. **Meta-Learning applied to Neural Architecture Search. Towards new interactive learning approaches for indexing and analyzing images from expert domains** . 2024. 218 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

A critical factor for the Deep Learning progress over the years was the proposal of novel architectures that enabled considerable advancements in the learning capabilities of Neural Networks. However, experts still mainly define neural architectures in a time-consuming trial-and-error process. As a result, the need for optimizing this process led to the emergence of Neural Architecture Search (NAS), which has two main advantages over the status quo: It can optimize practitioners' time by automating architecture design, and enables the discovery of novel architectures. The NAS framework has three main components: (i) Search Space, which defines the space of candidate architectures; (ii) Search Strategy, which specifies how the Search Space is explored; and the (iii) Performance Estimation Strategy that defines how an architecture's performance is estimated. While the Cell-based Search Space has dominated popular NAS solutions, the same is not true for Search and Performance Estimation Strategies where no dominant approach is used. Many NAS methods explore architectures' space using Reinforcement Learning, Evolutionary Computation, and Gradient-based Optimization. As a Performance Estimation Strategy, the so-called One-Shot models and the more recent Training-Free and Prediction-based methods have also gained notoriety. Despite presenting good predictive performance and reduced costs, existing NAS methods using such approaches still suffer from model complexity, requiring many powerful GPUs and long training times. Furthermore, several popular solutions require large amounts of data to converge, involve inefficient and complex procedures, and lack interpretability. In this context, a potential solution is the use of Meta-Learning (MtL). MtL methods have the advantage of being faster and cheaper than mainstream solutions by using previous experience to build new knowledge. Among MtL approaches, three stand out: (i) Learning from Task Properties; (ii) Learning from Model Evaluations; and (iii) Learning from Prior Models. This thesis proposes two methods that use prior knowledge to optimize the NAS framework: Model-based Meta-Learning for Neural Architecture Search (MbML-NAS) and Active Differentiable Network Topology Search (Active-DiNTS). MbML-NAS learns from both task characteristics encoded by architectural meta-features and performances from pre-trained architectures to predict and select ConvNets for Image Classification. Active-DiNTS learns from model evaluations, prior models, and task properties in the form of an Active Learning framework that takes information from model outputs, uncertainty estimations, and newly labeled examples in an iterative process. Experiments with MbML-NAS showed that the method was able to generalize to different search spaces and datasets using a minimum set of six interpretable meta-features. Using a simple approach with

traditional regressors, MbML-NAS reported comparable predictive performances with the state-of-the-art using at least 172 examples or just 0.04% and 1.1% from the NAS-Bench-101 and NAS-Bench-201 search spaces. Active-DiNTS obtained state-of-the-art results in segmenting images in the Brain dataset from the MSD challenge, surpassing the main baseline DiNTS by up to 15%. In terms of efficiency, alternative configurations achieved comparable results to DiNTS using less than 20% of the original data. Furthermore, Active-DiNTS is computationally efficient as it generates models with fewer parameters and better memory allocation using one GPU.

Keywords: Neural Networks, Neural Architecture Search, Meta-Learning, Convolutional Neural Networks, Computer Vision.

RESUMO

PEREIRA, G. T. **Meta-Aprendizado aplicado à Busca de Arquitetura Neural. Rumo à novas abordagens de aprendizagem interativa para indexação e análise de imagens de domínios especializados**. 2024. 218 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

Um fator crítico para o progresso de Deep Learning ao longo dos anos foi a proposta de novas arquiteturas que permitiram avanços consideráveis nas capacidades de aprendizagem de Redes Neurais. No entanto, especialistas ainda majoritariamente definem arquiteturas neurais em um processo demorado de tentativa e erro. Como resultado, a necessidade de otimização deste processo levou ao surgimento da Busca de Arquitetura Neural (NAS), que apresenta duas vantagens principais sobre o status quo: Pode otimizar o tempo de profissionais ao automatizar o projeto das arquiteturas, e permite a descoberta de novas arquiteturas. A estrutura de NAS tem três componentes principais: (i) Espaço de Busca, que define o espaço das arquiteturas candidatas; (ii) Estratégia de Busca, que especifica como o Espaço de Busca é explorado; e (iii) Estratégia de Estimativa de Performance, que define como o desempenho de uma arquitetura é estimado. Embora o Espaço de Buca baseado em célula tenha dominado soluções NAS populares, o mesmo não acontece com as Estratégias de Busca e Estimativa de Performance, onde nenhuma abordagem dominante é usada. Muitos métodos de NAS exploram o espaço das arquiteturas usando Aprendizado por Reforço, Computação Evolucionária e Otimização Baseada em Gradiente. Como Estratégia de Estimativa de Performance, os chamados modelos One-Shot e os mais recentes métodos Training-Free e Prediction-based também ganharam notoriedade. Apesar de apresentar bom desempenho preditivo e custos reduzidos, os métodos de NAS existentes que utilizam tais abordagens ainda sofrem com complexidade de modelo, exigindo muitas GPUs poderosas e longos tempos de treinamento. Além disso, diversas soluções populares exigem grandes quantidades de dados para convergir, envolvem procedimentos ineficientes e complexos, e carecem de interpretabilidade. Neste contexto, uma solução potencial é a utilização de Meta-Aprendizado (MtL). Os métodos de MtL têm a vantagem de serem mais rápidos e baratos que soluções convencionais, pois utilizam experiência prévia para construir novos conhecimentos. Dentre as abordagens MtL, três se destacam: (i) Aprendizado a partir de Propriedades de Tarefa; (ii) Aprendizado a partir de Avaliações de Modelos; e (iii) Aprendizado a partir de Modelos Anteriores. Esta tese propõe dois métodos que utilizam conhecimento prévio para otimizar o framework NAS: Model-based Meta-Learning for Neural Architecture Search (MbML-NAS) e Active Differentiable Network Topology Search (Active-DiNTS). O MbML-NAS aprende tanto com características de tarefas codificadas por meta-atributos arquitetônicos quanto com desempenhos de arquiteturas pré-treinadas para prever e selecionar ConvNets para Classificação de Imagens. O Active-DiNTS aprende com avaliações de modelos, modelos anteriores e propriedades de tarefas na forma de uma estrutura de Aprendizado Ativo que obtém

informações de resultados de modelos, estimativas de incerteza e novos exemplos rotulados em um processo iterativo. Experimentos com o MbML-NAS mostraram que o método foi capaz de generalizar para diferentes espaços de busca e conjuntos de dados usando um conjunto mínimo de seis meta-atributos interpretáveis. Usando uma abordagem simples com regressores tradicionais, o MbML-NAS relatou desempenhos preditivos comparáveis com o estado-da-arte usando pelo menos 172 exemplos ou apenas $0,04\%$ e $1,1\%$ dos espaços de busca do NAS-Bench-101 e NAS- Bench-201. O Active-DiNTS obteve resultados estado-da-arte na segmentação de imagens do conjunto de dados Brain do desafio MSD, superando a linha de base principal DiNTS em até 15%. Em termos de eficiência, configurações alternativas alcançaram resultados comparáveis ao DiNTS usando menos de 20% dos dados originais. Além disso, o Active-DiNTS é computacionalmente eficiente pois gera modelos com menos parâmetros e melhor alocação de memória usando uma GPU.

Palavras-chave: Redes Neurais, Busca de Arquitetura Neural, Meta-Aprendizado, Redes Neurais Convolucionais, Visão Computacional.

RÉSUMÉ

PEREIRA, G. T. **Méta-Apprentissage appliqué à la Recherche D'Architecture Neuronale – Vers de nouvelles approches d'apprentissage interactif pour faciliter l'indexation et l'analyse d'images de domaines experts** . 2024. 218 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

Un facteur essentiel dans les progrès de l'apprentissage profond au fil des années a été la proposition de nouvelles architectures qui ont permis des avancées considérables dans les capacités d'apprentissage des réseaux neuronaux. Cependant, les experts définissent encore principalement les architectures neuronales au cours d'un processus d'essais et d'erreurs chronophage. Par conséquent, la nécessité d'optimiser ce processus a conduit à l'émergence de la Recherche d'Architecture Neuronale (NAS), qui présente deux avantages principaux par rapport au statu quo: Il peut optimiser le temps des praticiens en automatisant la conception d'architectures et permet la découverte de nouvelles structures. Le cadre NAS comporte trois composants principaux: (i) L'Espace de Recherche, qui définit l'espace des architectures candidates; (ii) La Stratégie de Recherche, qui spécifie comment l'Espace de Recherche est exploré; et (iii) La Stratégie d'Estimation des Performances qui définit comment les performances d'une architecture sont estimées. Alors que l'Espace de Recherche basé sur les cellules a dominé les solutions NAS populaires, il en va autrement pour les Stratégies de Recherche et d'Estimation des Performances, où aucune approche dominante n'est utilisée. De nombreuses méthodes NAS explorent l'espace des architectures en utilisant l'Apprentissage par Renforcement, le Calcul Évolutif et l'Optimisation basée sur les Gradients. En tant que Stratégie d'Estimation des Performances, les modèles dits One-Shot et les méthodes plus récentes Training-Free et Predicion-based ont également gagné en notoriété. Malgré de bonnes performances prédictives et des coûts réduits, les méthodes NAS existantes utilisant de telles approches souffrent toujours de la complexité des modèles, nécessitant de nombreux GPU puissants et de longs temps d'entraînement. De plus, plusieurs solutions populaires nécessitent de grandes quantités de données pour parvenir à la convergence des modèles, impliquent des procédures inefficaces et complexes et un manque d'interprétabilité. Dans ce contexte, une solution potentielle est le recours au Méta-Apprentissage (MtL). Les méthodes MtL ont l'avantage d'être plus rapides et moins coûteuses que les solutions traditionnelles en utilisant l'expérience antérieure pour acquérir de nouvelles connaissances. Parmi les approches MtL, trois se démarquent: (i) Apprendre à partir des Propriétés de la Tâche; (ii) Apprendre des Évaluations Modèles; et (iii) Apprendre des Modèles Antérieurs. Cette thèse propose deux méthodes qui utilisent des connaissances antérieures pour optimiser le cadre NAS: Le Model-based Meta-Learning for Neural Architecture Search (MbML-NAS) et Active Differentiable Network Topology Search (Active-DiNTS). MbML-NAS apprend à la fois des caractéristiques des tâches encodées par des méta-fonctionnalités architecturales

et des performances de modèles pré-entraînés pour prédire et sélectionner des ConvNets pour la Classification d'Images. Active-DiNTS apprend des évaluations de modèles, des modèles antérieurs et des propriétés de tâche sous la forme d'un cadre d'Apprentissage Actif qui utilise des informations issues des sorties du modèle, des estimations d'incertitude et des exemples nouvellement étiquetés dans un processus itératif. Les expériences avec MbML-NAS ont montré que la méthode était capable de se généraliser à différents espaces de recherche et ensembles de données en utilisant un ensemble minimum de six méta-fonctionnalités interprétables. En utilisant une approche simple avec des régresseurs traditionnels, MbML-NAS a rapporté des performances prédictives comparables à celles de l'état de l'art en utilisant au moins 172 exemples ou seulement 0,04% et 1,1% des espaces de recherche NAS-Bench-101 et NAS-Bench-201. Active-DiNTS a obtenu des résultats de pointe dans la segmentation d'images dans l'ensemble de données cérébral du défi MSD, surpassant le principal modèle de référence DiNTS jusqu'à 15%. En termes d'efficacité, des configurations alternatives ont obtenu des résultats comparables à ceux de DiNTS en utilisant moins de 20% des données d'origine. De plus, Active-DiNTS est informatiquement efficace car il génère des modèles avec moins de paramètres et une meilleure allocation de mémoire en utilisant un seul GPU.

Mots clés: Réseaux de neurones, Recherche d'architecture neuronale, Méta-apprentissage, Réseaux de neurones convolutifs, Vision par ordinateur.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AB	AdaBoost
Acc	Accuracy
Active-DiNTS	Active Differentiable Network Topology Search
AL	Active Learning
ANOVA	Analysis of Variance
AR	Algorithm Recommendation
BO	Bayesian Optimization
BR	Bayesian Ridge
ConvNet	Convolutional Neural Network
CSP	Constraint Satisfaction Problem
CT	Computerized Tomography
CV	Computer Vision
DiNTS	Differentiable Network Topology Search
DL	Deep Learning
DNNs	Deep Neural Networks
DR	Dimensionality Reduction
DT	Decision Trees
EAs	Evolutionary Algorithms
FE	Feature Extraction
FFNN	Feed-Forward Neural Networks
FLOPs	Floating Point Operations
FS	Feature Selection
GAs	Genetic Algorithms
GB	Gradient Boosting
GCN	Graph Convolutional Network
GD	Gradient Descent
KNN	K-Nearest Neighbors
LR	Logistic Regression
MAE	Mean Absolute Error
MbML-NAS	Model-based Meta-Learning for Neural Architecture Search
MDI	Mean Decrease in Impurity

ML	Machine Learning
MLP	Multi-Layer Perceptron
MRI	Magnetic Resonance Imaging
MSE	Mean Squared Error
MtL	Meta-Learning
NAS	Neural Architecture Search
NB	Naive Bayes
NN	Neural Network
PC	Principal Components
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
TL	Transfer Learning

CONTENTS

LA ROCHELLE UNIVERSITÉ

ÉCOLE DOCTORALE EUCLIDE

Laboratoire Informatique, Image et Interaction

THÈSE présentée par :

Gean T. Pereira

soutenue le : **18 mars 2024**

pour obtenir le grade de : **Docteur de La Rochelle Université**

Discipline : **Informatique et applications**

Méta-Apprentissage appliqué à la Recherche D'Architecture Neuronale – Vers de nouvelles approches d'apprentissage interactif pour faciliter l'indexation et l'analyse d'images de domaines experts

JURY :

Heloisa ARAÚJO	Professeur, Universidade Federal de São Carlos, Examinatrice, Présidente du jury
Akka ZEMMARI	Professeur, Université de Bordeaux, Rapporteur
David HELBERT	Professeur, Université de Poitiers, Rapporteur
Aurora POZO	Professeur, Universidade Federal do Paraná, Examinatrice
André CARVALHO	Professeur, Universidade de São Paulo, Co-Directeur de thèse
Muriel VISANI	Maîtresse de conférences HDR, La Rochelle Université, Co-directrice de thèse

INVITÉS :

Thierry URRUTY	Maître de conférences HDR, Université de Poitiers
Luis NONATO	Professeur, Universidade de São Paulo

CHAPTER

1

INTRODUCTION

Research on Machine Learning (ML) has shown considerable progress over the years in a series of complex tasks, such as in Image Recognition, Speech Translation, and Text Generation (LECUN; BENGIO; HINTON, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016; DONG; WANG; ABBAS, 2021). Especially on Deep Learning (DL), such progress was possible due to three major technological advances (HUTTER; KOTTHOFF; VANSCHOREN, 2019): (*i*) the larger amount of data currently available and the quality improvement of such data, which made it possible to train a Neural Network (NN) with enough examples to extract useful knowledge; (*ii*) the necessary hardware to process the data, in particular, the availability of more powerful Graphics Processing Units and their adoption for training NNs, allowing greater optimization in terms of data usage and training time; and (*iii*) the increasing complexity of neural architectures, changing the status quo from shallow NNs to Deep Neural Networks (DNNs) containing hundreds of layers with different types and configurations and, quite often, composing architectures with a variety of topology patterns (HE *et al.*, 2016).

Especially when considering the latter technological advances in modern architectures, it becomes clear that a key factor for their success is the automatic feature extraction embedded in these models (HUANG *et al.*, 2017). A classic example of this mechanism is seen in Convolutional Neural Network (ConvNet), where features automatically extracted from images or grid-like data in the format of simple patterns, such as contours and lines, are identified in the first layers to be later combined in deeper layers, forming more complex patterns (LECUN *et al.*, 1989). These procedures, combined with the high intrinsic parallelism of ConvNets, not only give these models the ability to learn from larger amounts of data but often generate improved performance when more data is available (ZEILER; FERGUS, 2014). Consequently, DNNs have a considerable advantage over other ML algorithms as they can manage to learn complex relationships from larger datasets without having to do manual feature engineering, which can be a long and costly process for learning systems (HAYKIN, 1998).

With the growing popularity of NNs, an interest in automating their entire design has arisen (ZOPH; LE, 2016). Neural architectures are traditionally defined in an iterative process often guided by expert knowledge via trial-and-error or based on heuristics, which presents several challenges (BAKER *et al.*, 2016). Manual architectural design can be time-consuming and inefficient, requiring specialists to adjust various architectural parameters iteratively (WANG *et al.*, 2020). Furthermore, this process tends to be highly subjective and dependent on the expert's intuition, making it difficult to find optimal solutions (REAL *et al.*, 2017). This manual design also leads to limited or sub-optimal exploration of the architectural space, which tends to be quite broad, thus potentially missing out on more effective configurations (LIU *et al.*, 2018). Additionally, manual approaches can struggle to capture complex relationships and dependencies in the data, leading to lower performance (WHITE *et al.*, 2021). Motivated by these problems and the impracticality of the process, automated methods have emerged to address these limitations and efficiently discover architectures that overcome the ones designed by humans, leading to a research topic known as Neural Architecture Search (NAS).

NAS methods aim to automate the entire process of architecture design, which often starts with a pre-defined search space of candidate models and search algorithms such as Genetic Algorithms and Reinforcement Learning to explore the spaces (REAL *et al.*, 2019). In addition, these search strategies are usually used alongside strategies to accurately estimate the performance of sampled architectures, thus speeding up the process and discovering optimized architectures for the task at hand (DENG; YAN; LIN, 2017). In recent years, various NAS methods have been proposed and promising results across various tasks have been reported, including Image Classification, Medical Image Segmentation, and Natural Language Processing (SIEMS *et al.*, 2020; ZHU *et al.*, 2019). Initially, most NAS methods relied on Bayesian Optimization, Reinforcement Learning, and Evolutionary Algorithms (ELSKEN; METZEN; HUTTER, 2018; KANDASAMY *et al.*, 2018; BAKER *et al.*, 2016). However, Gradient-based search methods have gained attention over the years (LUO *et al.*, 2018). Additionally, several performance estimation strategies have emerged, which may or may not be used along with the aforementioned search algorithms. These performance estimation approaches include the so-called One-shot models, Prediction-based NAS, and Training-free NAS, which are increasingly being adopted in the field (BENDER *et al.*, 2018; CHEN; GONG; WANG, 2021; SUN *et al.*, 2019).

Although recent solutions have alleviated some of the common problems in NAS, existing methods still suffer from complex issues that must be addressed (WEI *et al.*, 2022; MELLOR *et al.*, 2021). One of the most crucial is the high training complexity, which often involves dozens or hundreds of GPU hours to generate suitable architectures even when using small datasets (ZOPH *et al.*, 2018). Combined with the need for powerful hardware, this limits NAS accessibility to researchers or practitioners with limited computing resources. Another challenge lies in the complexity associated with pipeline and model design. Simplifying the NAS pipeline is vital for enhancing its usability and scalability, where intricate procedures and complex models are

common, making it difficult to understand, reproduce, and interpret the outcomes (MOLNAR, 2020). Furthermore, another problem that affects reproducibility and reliability is the lack of interpretability of these models (RU *et al.*, 2020). Both NAS methods and generated architectures are often difficult to interpret, which limits the insights into the decision-making process of the design choices. Another limiting factor of mainstream NAS is that it is task-specific, thus lacking the ability to generalize knowledge across different tasks (LI *et al.*, 2021). Consequently, for each new task, it is necessary to conduct a new search from scratch without any basic knowledge to assist in the current task. In this context, and considering the aforementioned problems, an approach called Meta-Learning (MtL) or Learning to Learn can be used (BRAZDIL *et al.*, 2008). This type of learning consists of using experiences acquired in previous tasks and, with such prior knowledge, building a more general knowledge to learn new tasks more quickly using fewer training examples and computational resources (THRUN; PRATT, 1998).

There are at least three well-known MtL approaches: Learning from Tasks, Learning from Model Evaluations, and Learning from Prior Models (ELSKEN; METZEN; HUTTER, 2019). Examples of such approaches can be found by the name of popular ML techniques, such as Transfer Learning (TL) (DONAHUE *et al.*, 2014), a closely related topic to MtL. TL is widely employed to initialize DNNs' weights for new tasks after training on large datasets such as ImageNet [1]. Thus, TL can be seen as a specific instance of MtL, where knowledge acquired by an NN's initial layers works as a meta-model of knowledge (PERRONE *et al.*, 2018). Therefore, this meta-model is leveraged to facilitate learning on new tasks by transferring the learned representations from the pre-trained layers. Another powerful technique related to MtL is Active Learning (AL) (COHN; ATLAS; LADNER, 1994), which also uses prior knowledge to enhance learning. However, this knowledge comes in the form of an Oracle or Expert figure to address the challenges of labeled data scarcity and model data inefficiency. By selecting and labeling the most informative samples from a large unlabeled dataset, an AL strategy can considerably reduce the amount of labeled data required for training and can often reduce related costs such as training time and computational resource consumption. Additionally, AL can save experts' labeling efforts, which in domains such as the medical field is an important issue to consider. Thus, by actively querying the most informative instances, AL effectively maximizes model learning efficiency and can even lead to increased predictive performance (DASGUPTA, 2011).

To validate the proposal of novel NAS methods, many studies have been using Computer Vision (CV) problems as proof-of-concept and benchmarks (LIU; SIMONYAN; YANG, 2018; WISTUBA; RAWAT; PEDAPATI, 2019; WHITE *et al.*, 2023). Likewise, DNNs have a long history of being validated on image datasets, as were the cases of popular architectures such as AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), VGG (SIMONYAN; ZISSERMAN, 2014), and GoogLeNet (SZEGEDY *et al.*, 2015), originally introduced in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (RUSSAKOVSKY *et al.*, 2015). Image

[1] http://www.image-net.org/

Classification, in particular, remains one of the most popular tasks in CV, and this popularity stems from the relative simplicity of classifying images when compared to other tasks, such as Object Detection and Segmentation (LIU *et al.*, 2019; REN *et al.*, 2021). Nonetheless, Image Classification still holds relevance to practical applications and is challenging enough, enabling reliable state-of-the-art assessments for both NAS and DNNs (GOODFELLOW; BENGIO; COURVILLE, 2016; DONG; WANG; ABBAS, 2021).

Despite the significant advancements made in these fields, the validation of new proposals continues to rely heavily on classification tasks (WHITE *et al.*, 2021). However, it is crucial to consider other tasks, particularly those that fall outside the mainstream, to determine the actual robustness of these new proposals. Among alternative tasks, Image Segmentation possesses considerable scientific and practical appeal, although it receives far less attention from the research community (WHITE *et al.*, 2023). Especially for Medical Image Segmentation, a sub-field of Image Segmentation that remains relatively unexplored, with only a handful of recent NAS works making progress in this matter (ANTONELLI *et al.*, 2022). Nevertheless, even these works are limited in number and lack certain essential aspects, such as the absence of more realistic methods that can be readily applied in practical scenarios (LIU *et al.*, 2019). Many proposals prioritize performance while disregarding other crucial factors such as labeled data scarcity, the necessity for lightweight models with fast training, less complex pipelines, and interpretable models (ISENSEE *et al.*, 2019). Hence, there is a need to address these aspects and develop comprehensive solutions that encompass such challenges.

This introductory chapter has provided a brief context and established the domain in which this Ph.D. thesis operates. The subsequent sections of this chapter are structured as follows: Section 1.1 introduces the underlying motivations behind this doctoral research; Section 1.2 outlines the central research question that served as the guide for the development of this thesis; Section 1.3 presents the hypothesis formulated to answer the research question throughout the course of this Ph.D.; Section 1.4 elucidate the general and specific objectives of this thesis; Lastly, Section 1.6 provides a comprehensive overview of this document's organizational structure.

1.1 Motivation

A few motivations guide this thesis, which mainly covers the subjects of NAS and MtL. The first motivation lies in the problems related to the manual process of designing NN architectures. Neural architectures are commonly defined by experts in a trial-and-error process, often guided by empirical evaluations and heuristics. Although several hand-designed architectures have been successful over the years, this practice tends to limit the quality of the generated architectures since the process is mediated by the experts' knowledge. In addition, such a process is frequently not data-oriented, thereby missing the opportunity to generate customized or optimized architectures for the task at hand. Another related issue is the amount of precious time spent by specialists in defining and fine-tuning neural hyper-parameters. Especially in real-world scenarios where resources are scarce, spending long and tedious periods of time could be crucial. Overcoming these limitations is essential to explore automated approaches, such as NAS, which can harness the power of ML algorithms to find and optimize NN architectures.

The second motivation concerns the investigation and optimization of NAS. In recent years, numerous NAS methods have been proposed and promising results reported. However, existing methods still suffer from data inefficiency, high training times, complex pipelines, and a lack of interpretability. Another common problem is reproducing such state-of-the-art methods. For many popular NAS solutions, reproducing experiments requires thousands of GPU hours worth of training with high-end GPUs. Moreover, certain NAS methods exhibit significant instability during architecture search and lack flexibility when applied to different tasks. Consequently, each new task requires the search process to be started from scratch. Given the exorbitant costs involved, which can include days of search time and multiple GPUs, this entire process poses a significant predicament for ML practitioners and researchers who lack such infrastructure. Therefore, a NAS method needs to be optimized to be used on a large scale, and a possible solution for this problem is the usage of MtL.

This thesis's third and final motivation centers around the comprehensive study and investigation of MtL approaches. MtL provides the opportunity to leverage prior knowledge acquired from different and/or diverse tasks in order to enhance learning. Consequently, it facilitates faster and optimal generalization to new tasks, often using fewer examples and reducing the required computation power when compared to mainstream solutions. Additionally, the possibilities offered by the MtL approaches, along with their various forms and applications, introduce a complex decision-making challenge that requires careful exploration. Nevertheless, before delving into the application of MtL on NAS, it is imperative to explore the different forms of MtL to establish its feasibility. In addition, it is crucial to carefully analyze and assess the gains brought by the MtL approaches to determine its adoption justification. Furthermore, it is also important to identify the most effective way of applying the MtL methodology to address the specific challenges inherent to the NAS problem.

1.2 Research Question

In view of what was presented, the following research question is formulated:
How to automatically find neural architectures for Image Recognition with comparable predictive performances using fewer data and less complex procedures than popular NAS methods?

1.3 Hypothesis

To answer the research question, the following hypothesis is formulated:
A Meta-Learning approach can leverage prior knowledge to reduce the computational burden and simplify the search for Image Recognition architectures with good predictive performance using fewer data samples than NAS state-of-the-art methods.

1.4 Objectives

In addition to seeking evidence that validates the hypothesis and, consequently, answers the research question that guides the development of this research, this thesis aims to investigate and propose MtL methods to find neural architectures for Image Recognition and related MtL tasks. More precisely, the tasks of Image Classification and Image Segmentation are explored, in addition to the co-related task of Algorithm Recommendation. Such developed methods should be able to generate results with comparable predictive performance to the NAS state-of-the-art using fewer training examples, shorter training times, and more simplified pipelines with a limited amount of hardware. The specific objectives of this thesis are:

- Investigate the applicability of MtL approaches in the NAS framework;
- Investigate the impact on predictive performance, data efficiency, and pipeline complexity caused by MtL approaches on NAS;
- Propose and maintain MtL methods to support ML researchers and practitioners on decisions related to NNs' design, training, and tuning;
- Propose new meta-datasets for NAS suitable to standard and modern ML methods;
- Create and maintain an updated literature review with a focus on NAS and MtL applied to Image Recognition and Algorithm Recommendation tasks.

1.5 Main Results

The main contributions, and therefore, the principal results of this thesis revolve around the proposal and experimental validation of two novel methods: Model-based Meta-Learning for Neural Architecture Search (MbML-NAS) and Active Differentiable Network Topology Search (Active-DiNTS). These new solutions address distinct challenges within NAS, each providing unique insights and advancements in the field, but both focus on efficiency and simplicity.

MbML-NAS, introduced in Chapter 5, is a novel Prediction-based NAS method that leverages meta-characteristics from neural architectures to select the most promising models for Image Classification. The primary objective of MbML-NAS is to overcome challenges related to data efficiency, model complexity, and interpretability in NAS. Through the incorporation of interpretable meta-features and simplified meta-predictors, MbML-NAS efficiently selects optimal neural architectures. The results obtained through extensive experimentation demonstrate MbML-NAS's capability to achieve comparable predictive performance to intricate state-of-the-art models while utilizing only a minimal fraction of the search space. In its most cost-effective configuration, MbML-NAS employs a mere 0.04% and 1.1% of the search spaces of NAS-Bench-101 and NAS-Bench-201, respectively. Moreover, the interpretability analysis sheds light on how meta-information extracted from cell-based spaces influences the learning of meta-predictors. The comprehensive assessment, including meta-error analysis, standalone performances, and meta-feature correlations, underscores MbML-NAS as a well-balanced solution, offering a favorable compromise between predictive performance, data efficiency, and interpretability.

Active-DiNTS, introduced in Chapter 6, is a One-shot model that focuses on the challenging task of Medical Image Segmentation. The method combines the power of NAS with an Active Learning (AL) framework that allows for iterative labeling of the most impactful examples to improve learning and address the scarcity of labeled data in medical imaging. On top of that, Active-DiNTS explores a flexible network topology search space using gradient-based optimization, which makes it possible to discover architectures with many topology patterns. The experimental results reveal the effectiveness of Active-DiNTS in generating state-of-the-art architectures for segmenting medical images. Through a Pool-based Sampling strategy using Uncertainty functions, Active-DiNTS was able to significantly reduce search time by up to 27 times and training data by up to 5 times compared to a non-active version and other baselines, showcasing its potential for resource-efficient architecture discovery. Additionally, the analysis of the Uncertainty functions further contributed to understanding example importance in the active labeling process. The comparisons with various segmentation methods, including NAS models and hand-crafted methods, highlight the superiority of Active-DiNTS in terms of predictive performance, computational efficiency, and resource utilization.

1.6 Outline

The rest of this document is organized as follows: Chapter 2 introduces the main concepts for the understanding of this thesis in addition to a literature review of NAS and MtL. Chapter 3 presents the first experiment formulated to explore some capabilities of the MtL framework. Chapter 4 is a continuity where the characteristics of the meta-datasets and their generalization power were explored. Chapter 5 presents the novel MbML-NAS method, where some of the previous findings are applied in order to advance towards the validation of the research hypothesis. Chapter 6 presents the novel Active-DiNTS method, a proposal that goes beyond the mainstream Image Classification task in NAS by addressing Medical Image Segmentation, in addition to exploring other forms of MtL. Lastly, Chapter 7 concludes this thesis by presenting the main contributions, limitations, and prospective work.

CHAPTER

2

FOUNDATIONS

This chapter introduces the fundamental concepts necessary for a better understanding of the proposals and discussions presented in this thesis. The formal definitions and assumptions of ML and ML tasks involved in this thesis are presented in Section 2.1. An introduction to NNs is presented in Section 2.2, describing a brief history of the area and pointing out some important types of NNs, such as the Multi-Layer Perceptron, Convolution Neural Networks, and U-Nets. In Section 2.3, the concept of NAS is presented, a recent research field dedicated to the discovery and optimization of novel neural architectures. In Section 2.4, the introduction of a different type of learning, known as Meta-Learning (MtL) or Learning to Learn, is presented together with three MtL approaches. In Section 2.5 and Section 2.6, the concepts of Transfer Learning and Active Learning, related topics to MtL, are introduced along with their respective approaches and details. Lastly, the final considerations for the chapter are presented in Section 2.7.

2.1 Machine Learning

Being one of the most popular sub-areas of Artificial Intelligence, Machine Learning focuses on developing algorithms that can learn how to make decisions from data without being explicitly programmed (SAMUEL, 1967). Through an iterative learning process, such learning algorithms can extract patterns from input information and adjust their decision functions over time. According to Mitchell (1997), an algorithm can learn from experience E concerning a class of tasks T and regarding a performance measure P, if its performance at T as measured by P improves with E. For instance, consider an algorithm that learns to classify cats in images. In this case, the following components of the learning problem can be defined: (i) Task T: Classifying cats in images; (ii) Performance measure P: Accuracy of correctly identifying cats; (iii) Experience E: Training the learning algorithm with a dataset of labeled images containing cats and non-cat objects. Thus, by providing the algorithm with a training dataset of labeled

images, where each image is categorized as either a cat or a non-cat object, the learning algorithm can learn from this experience and improve its performance in classifying cats in images. After being trained, the learning algorithm has its final performance measured in a test dataset, which would be the percentage of accurately identified cat images in this specific case.

The previously described scenario corresponds to the learning paradigm of Supervised Learning and the Classification task. However, this is not the only possible case, as ML encompasses various learning paradigms and associated tasks. Each paradigm plays a crucial role in the continued advancement of ML, where the most popular paradigms are Supervised, Unsupervised, and Reinforcement Learning, in addition to Semi-Supervised Learning (GAMA *et al.*, 2011). In Supervised Learning, models learn from labeled examples, with Classification and Regression being the most popular tasks in this paradigm (DONG; WANG; ABBAS, 2021). On the other hand, Unsupervised Learning involves discovering patterns and structures in unlabeled data, with Clustering and Anomaly Detection being typical examples (JORDAN; MITCHELL, 2015). Reinforcement Learning involves training agents to make sequential decisions using rewards and punishments, and it is typically applied to robotics or simulations (RUSSELL; NORVIG, 2009). Additionally, there is Semi-Supervised Learning, which combines labeled and unlabeled data and aims to make the most of supervised feedback, presenting great flexibility since it can be used with Classification, Regression, and Clustering, among others (MITCHELL, 1997).

Among various ML paradigms and tasks, Supervised Learning, along with the tasks of Classification and Regression, is the most elementary and popular subject in the field even to this day (HASTIE *et al.*, 2009). Classification is a supervised task that involves assigning input data to pre-defined categories or classes (BISHOP, 2006). Formally, it can be defined in terms of a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathcal{X}$ represents the feature vector of the i-th instance, and $y_i \in \mathcal{Y}$ is its corresponding class label from a set of classes $C = \{c_1, c_2, ..., c_k\}$, being k the finite number of classes. Thus, the goal is to learn a function $c : \mathcal{X} \rightarrow \mathcal{Y}$ that maps the input space $\mathcal{X}$ to the corresponding output space. Regression aims to estimate or predict a continuous target variable based on the patterns observed in the input data (JORDAN; MITCHELL, 2015). This target variable can represent a wide range of real-valued quantities, such as housing prices, stock market prices, temperature, or any other continuous quantity of interest (MICHIE *et al.*, 1995). Similar to Classification, Regression can be formally defined by a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathcal{X}$ represents the feature vector of the i-th instance, and $y_i \in \mathcal{Y}$, which in this case is the corresponding continuous target variable. The goal is then to learn a function $r : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ represents the input space and $\mathcal{Y}$ represents the output space.

Another relevant factor when talking about learning paradigms and tasks is the choice of ML models and evaluation metrics. For Classification, popular algorithms include Decision Trees (DT), Logistic Regression (LR), Support Vector Machines (SVM), Naive Bayes (NB), K-Nearest Neighbors (KNN), and Feed-Forward Neural Networks (FFNN) (GAMA *et al.*, 2011). For Regression, several of these same algorithms can be used with minor modifications, such as DTs,

SVMs, Naive Bayes, KNN, and FFNNs (MCLACHLAN, 2005). As for the evaluation metrics, the performance of a classification model is typically evaluated using Accuracy, Precision, Recall, and/or F1-score (RUSSELL; NORVIG, 2009). For Regression, however, performance is commonly measured using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2) (HASTIE *et al.*, 2009).

2.1.1 Algorithm Recomendation

It is common in ML that traditional and well-known tasks such as Classification and Regression present several variations and often being called by different names for handling very specific problems and priors. One example is the so-called Algorithm Recommendation (AR) task, which can be treated either as Classification or Regression and consists of learning to select the best algorithm among a range of possible candidates for a set of tasks or datasets considering a specific performance metric (VILALTA; DRISSI, 2002). The first known formulation of the AR problem is seen in Rice (1976), sometimes referred to as the Algorithm Selection problem. Given a set of problem instances P from a distribution D, a set A of algorithms, and a performance measure m: $P \times A \rightarrow \mathbb{R}$, the AR problem is concerned with finding a mapping f: $P \rightarrow A$ that optimizes the expected performance measure m for instances P with a distribution D.

While it may seem intuitive that similar datasets might be solved by the same algorithm, the question of whether a single algorithm can perform optimally across all problem domains remains open. This question was addressed by Wolpert (1996) in the famous "no free lunch" theorems, which demonstrates that given a set of algorithms applied to all possible domains, the average performance of these algorithms remains the same. This implies that a single learning algorithm with significantly superior performance across all domains does not exist for supervised ML. To cope with this limitation, AR emerges as an approximate solution to intelligently select the most appropriate algorithm for each specific task or dataset given its inherent characteristics (VILALTA; DRISSI, 2002). By learning from past experiences, AR is much closer to MtL since it benefits from leveraging prior data to optimize model performance and efficiency for different scenarios. Even though AR is a well-known problem, using MtL to explore prior knowledge and accelerate inference is relatively recent and is a potential trend that still requires exploration (ALCOBAÇA *et al.*, 2018). Through AR, ML systems can make data-driven decisions on algorithm selection, tailoring their approach to each unique problem and ultimately improving overall predictive accuracy and generalization (LEMKE; BUDKA; GABRYS, 2015). This approach proves valuable in handling various challenges posed by real-world data, making ML more robust and adaptable in diverse applications.

The Algorithm Recommendation problem is significantly similar to the Performance Estimation facet of NAS, where the ultimate objective lies in the automatic discovery of optimal neural architectures tailored for specific tasks (THORNTON *et al.*, 2013). In the context of the Performance Estimation component, its core principle revolves around the utilization of approximations to determine the performance of different neural architectures before embarking on their actual end-to-end training (GIRAUD-CARRIER; PROVOST, 2005). This strategic approach serves to speed up the exploration of the usually large and complex NAS search spaces, effectively curbing resource-intensive costs. Therefore, akin to AR, the principal aim within the NAS domain is to achieve cost reduction by acquiring a deep understanding of the most fitting algorithms or neural architectures that align with the specific problem at hand (LIU *et al.*, 2018).

2.1.2 Image Classification

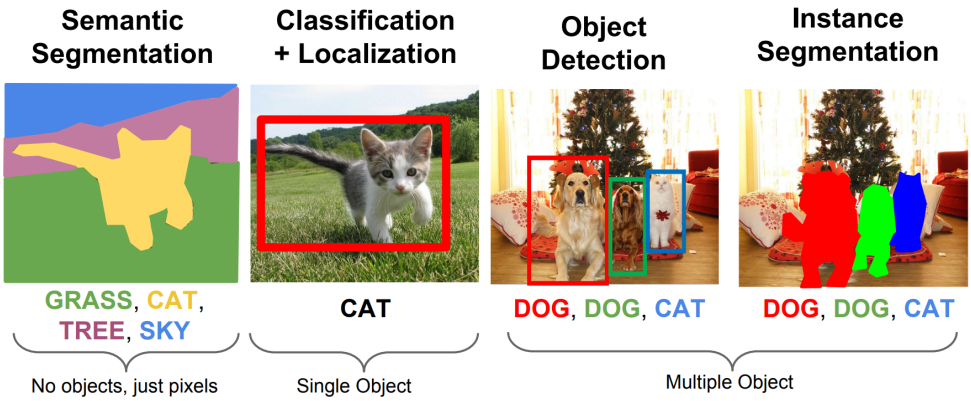


Figure 1 – Illustration of popular Computer Vision Tasks. Adapted from the Convolutional Neural Networks for Visual Recognition Stanford Course, 2020 [2].

Image Classification is perhaps the most popular task in the field of Deep Learning (RUSSAKOVSKY *et al.*, 2015). Originally from the interdisciplinary area of Computer Vision (CV), this task involves Image and Signal Processing, Physics, Biology, and ML. However, CV has been seen as a sub-area of ML in recent years, in which the main objective is to make computers capture and understand the content of images (PRINCE, 2012). Although CV problems seem simple at first glance, since even children can extract knowledge from a single image and generalize reasonably well to unseen examples, they are challenging tasks for algorithms (DENG *et al.*, 2009). As seen in Figure 1, the broad problem of understanding images can be divided into several sub-problems that aim to answer specific questions, some of which are (FORSYTH; PONCE, 2002): (i) Image Classification: What category does the object or objects in this image belong to? (ii) Object Detection: What broad category does the object or objects in this image belong to, and where are they located? (iii) Instance Segmentation: Which pixels belong to the individual instance object from a broad category in the image? (iv) Semantic Segmentation: Which pixels belong to a broad category in the image?

Image Classification is arguably the most fundamental among the different CV tasks, as it involves assigning a class label to an input image based on its visual content (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Formally, given an input image I, the objective is to learn a mapping function $f : I \rightarrow C$, being C a label class from a predefined set of classes. These classes represent different objects, scenes, or concepts that the model needs to recognize (WITTEN *et al.*, 2016). An input image I is often represented as a 4D tensor[1] $I \in \mathbb{R}^{N \times H \times W \times C}$, where N is the number of input images in a dataset, H and W are the height and width of the images, and C is the number of channels (e.g., 3 for RGB images). The output for this task is a probability distribution over L classes denoted as $P(Y|I)$, where Y represents the class label. To obtain $P(Y|I)$, a classification model M consisting of learnable parameters θ is used, which takes I and produces a class probability distribution $P(Y|I,\theta)$. To train M, a loss function L is defined to quantify the difference between the predicted probability distribution and the ground truth labels. The loss function measures the model's performance on a given image-label pair. Typically, the Cross-Entropy loss is used for Image Classification (DUDA; HART; STORK, 2000; DONG; WANG; ABBAS, 2021), as seen in Equation 2.1,

$$L(I,Y,\theta) = -\sum_{i=1}^{N} Y_i \log P(Y_i|I,\theta) \tag{2.1}$$

where L is the total number of class labels, Y_i is the ground truth label, and $P(Y_i|I,\theta)$ is the predicted probability of the i-th class. During training, the model's parameters θ are learned by minimizing the overall loss across a training dataset $\mathcal{D}$, as seen in Equation 2.2,

$$\theta^* = \arg\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(I,Y)\in\mathcal{D}} L(I,Y,\theta) \tag{2.2}$$

where θ^* represents the optimized parameters, and $|\mathcal{D}|$ is the number of samples in the training dataset. After training, the model M can be used for inference by taking an unseen image I_{test} and predicting its class label using Equation 2.3,

$$\hat{Y} = \arg\max_{Y} P(Y|I_{\text{test}},\theta^*) \tag{2.3}$$

where $\hat{Y}$ represents the predicted class label for the input test image I_{test}.

In addition to its great popularity and wide range of comparison baselines, Image Classification is a good and versatile task, serving as a valuable validation platform for new methods due to its simplicity and yet challenging nature towards the current state of ML algorithms (LECUN; BENGIO; HINTON, 2015). Despite being a relatively simple problem compared to other CV tasks, such as Object Detection and Image Segmentation, its moderate level of difficulty makes it an excellent reference for evaluating cutting-edge solutions and assessing the current state-of-the-art (HE *et al.*, 2016). In fact, many traditional and even recent NAS

[1] A mathematical object that generalizes scalars, vectors, and matrices to higher-dimensional arrays with support for autograd operations

proposals use Image Classification as a benchmark (WHITE *et al.*, 2023). As a widely studied problem with reference datasets such as ImageNet, this task provides a controlled environment for evaluating the performance of new methods. Furthermore, due to its relative simplicity, it allows researchers to focus on optimizing new NAS architectures without the need for excessive complexity to comprehend essential image features and patterns, making it a significant test for advancing the state-of-the-art in CV research (WISTUBA; RAWAT; PEDAPATI, 2019).

2.1.3 Medical Image Segmentation

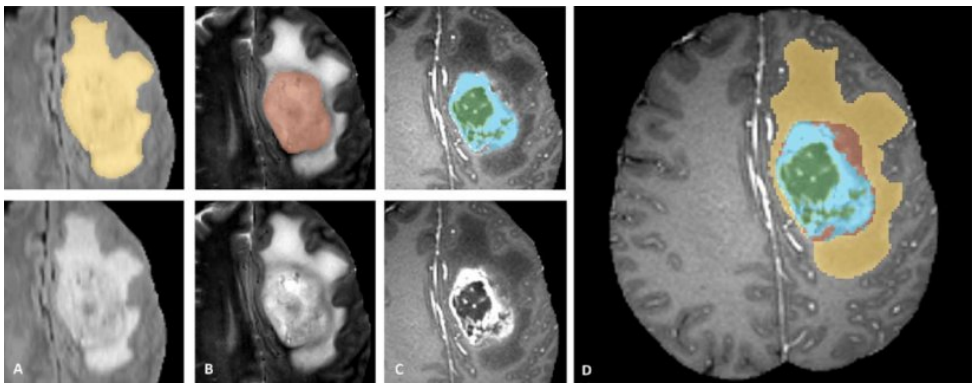


Figure 2 – Glioma sub-regions from BraTS dataset. Image patches show tumor annotations across different modalities (Top left) and final dataset labels (Right). From left to right: (A) the whole tumor in FLAIR; (B) the tumor core in T2; (C) the enhancing tumor structures in T1Gd (Blue) around the core's cystic/necrotic components (Green); and (D) Where the combined segmentations produce the final labels: Edema (Yellow), Non-enhancing solid core (Red), Necrotic/Cystic core (Green), and the Enhancing core (Blue). Adapted from Menze *et al.* (2014).

Another widely popular, and perhaps one of the most important tasks in CV is Image Segmentation, which consists of partitioning an input image into one or more regions of interest based on visual characteristics (PRINCE, 2012). In practical terms, this task can be seen as an extension of the Image Classification task since each pixel is assigned to a class of interest (FORSYTH; PONCE, 2002). As output, these pixels are combined to generate the so-called Segmentation Masks, which are then compared to the real class labels (YU *et al.*, 2020). Besides, within the general task of Image Segmentation, there are sub-categories, the most popular being called Instance Segmentation and Semantic Segmentation, both involving classifying each image pixel into specific object categories (WENG *et al.*, 2019). However, the key difference between the two lies in their level of granularity. Instance Segmentation goes beyond Semantic Segmentation by identifying object categories and providing precise pixel-level masks for each individual object instance (ISENSEE *et al.*, 2019). In contrast, Semantic Segmentation focuses solely on partitioning the image into different regions corresponding to distinct object categories without distinguishing between multiple instances of the same category (KIM *et al.*, 2019). In summary, Instance Segmentation offers fine-grained object instance delineation, while Semantic Segmentation provides a coarser image-level segmentation.

In the Image Semantic Segmentation task, given a 2D or 3D image input, the aim is to assign a semantic label to each pixel or voxel in the image, classifying them into a specific category or class (ZHU *et al.*, 2019). Mathematically, this problem can be defined in terms of an input image I represented as a 2D matrix for $I \in \mathbb{R}^{H\times W\times C}$ for 2D segmentation, where H is the height, W is the width, and C is the number of channels (e.g., 3 for RGB images). The output of the Semantic Segmentation task is a pixel-wise label map represented as $S \in \{0,1,\ldots,N-1\}^{H\times W}$. To obtain the pixel-wise label map, a segmentation model M fully defined by its learnable parameters θ is used. Thus, the model M takes the input image I and produces the pixel-wise label map S as output, denoted as $S = M(I,\theta)$. Figure 2 exemplifies the Image Semantic Segmentation scenario where a brain has its tumor regions identified at the pixel/voxel level, thus forming the segmentation masks for each class of the problem.

To train a model M, a loss function L is defined to quantify the difference between the predicted pixel-wise label map and the ground truth segmentation S_{gt}. Various loss functions such as Cross-Entropy Loss, Dice Loss, or Intersection over Union (IoU) can be used (DUDA; HART; STORK, 2000; PRINCE, 2012). As seen in Equation 2.4,

$$L(I,S_{\text{gt}},\theta) = \text{Loss}(S,S_{\text{gt}}) \tag{2.4}$$

$\text{Loss}(S,S_{\text{gt}})$ computes the dissimilarity between the predicted segmentation S and the ground truth S_{gt}. In turn, the model's parameters θ are learned by minimizing the overall loss across a training dataset $\mathcal{D}$. As seen Equation 2.5,

$$\theta^* = \arg\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(I,S_{\text{gt}})\in\mathcal{D}} L(I,S_{\text{gt}},\theta) \tag{2.5}$$

θ^* represents the optimized parameters, and $|\mathcal{D}|$ is the number of samples in the training dataset. After training, the model M can be used for inference by taking an unseen input image I_{test} and predicting its pixel-wise label map S_{pred} using $S_{\text{pred}} = M(I_{\text{test}},\theta^*)$.

One of the most popular Image Segmentation applications is in the Medical field (ANTONELLI *et al.*, 2022; MENZE *et al.*, 2014). Seen as of great importance both in industry and academia, Medical Image Segmentation plays a crucial role in clinical applications and healthcare practices where precise segmentation capabilities are required (RONNEBERGER; FISCHER; BROX, 2015; YAN *et al.*, 2020b). Exams such as Computerized Tomography (CT) and Magnetic Resonance Imaging (MRI) are used in such applications, helping professionals to accurately identify and delineate anatomical structures (OLSON *et al.*, 2016). Such information is vital for early disease detection, treatment planning, surgical interventions, and post-treatment evaluations (ÇIÇEK *et al.*, 2016). In the context of segmenting medical images, especially MRIs, it is common to deal with volumetric data, often referred to as voxels, where the goal is to assign a semantic label to every pixel in the 3D volume (PERSLEV *et al.*, 2019). Mathematically, it can be defined in terms of an input image represented as a 3D voxel volume $V \in \mathbb{R}^{H\times W\times D\times C}$,

where H is the height, W is the width, D is the depth (number of slices), and C is the number of channels representing different imaging modalities. As generated output, a 3D segmentation voxel label mask is represented as $L \in \{0,1,\ldots,N-1\}^{H\times W\times D}$, where N is the total number of semantic classes, and $L_{i,j,k}$ denotes the class label of the voxel at position (i,j,k).

2.2 Neural Networks

Artificial Neural Networks or simply *Neural Networks* are learning models slightly inspired by the human brain (GOODFELLOW; BENGIO; COURVILLE, 2016). Like their inspiration, these models are composed of processing units called *Neurons*, but in the case of artificial neurons, these are much simpler abstractions of their biological counterparts. Such artificial neurons are also linked, where the signal strength is represented through numerical weights. In addition, these neurons can be stacked in layers, and within each layer, neurons can be connected to each other or to neurons from other layers, with auto-connections also being possible. These multiple information flow possibilities allow Neural Networks to represent both linear and non-linear types of functions, which is one of the main aspects that make neural architectures such powerful and flexible learning models for a wide range of problems (NIELSEN, 2015). A representation of what an artificial neuron looks like is seen in Figure 3.

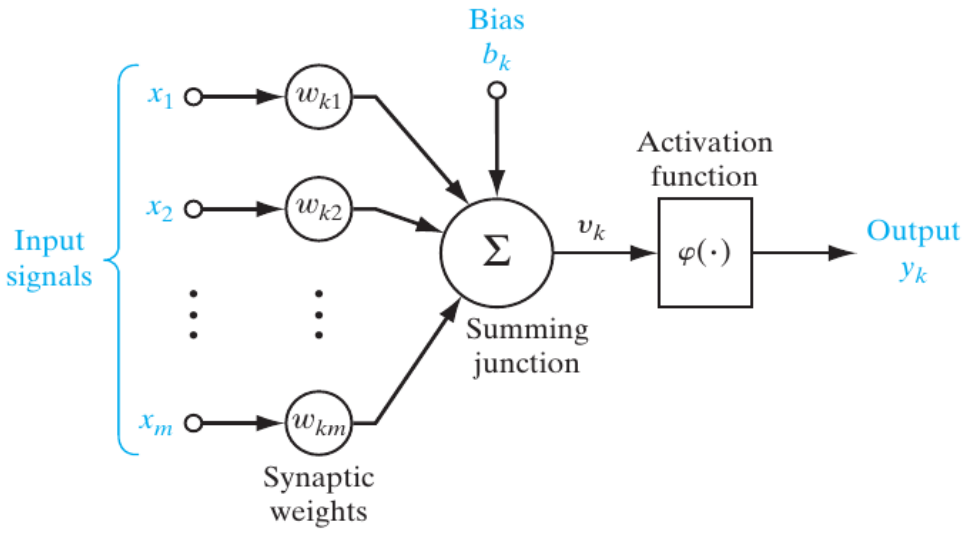


Figure 3 – Basic structure of an artificial neuron. Adapted from Haykin. (2010).

As seen in Figure 3, a neuron k receives a set of inputs $x_m \in X$ associated with weights $w_k m \in W$, where each input $x_1, x_2, ..., x_m$ will have a weight $w_{k1}, w_{k2}, ..., w_{km}$ linked to it, being these weights the free parameters of the Neural Network which are learned during training (HAYKIN., 2010). These inputs and their respective weights are then linearly combined by a sum function, in the form of $\upsilon_k = \sum_{j=1}^{m} w_{kj}\, x_j$. After such combination, the generated υ_k is transformed by an φ activation function that limits the neuron's activation potential, which is generally a non-linear function (HAYKIN, 1998). Many activation functions are used in modern

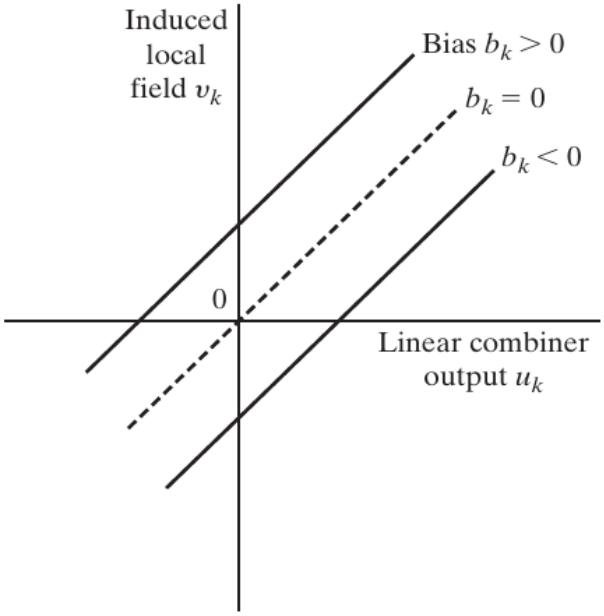


Figure 4 – Dynamics of bias in a neuron over a Cartesian plane. Adapted from Haykin. (2010).

Neural Networks, with Sigmoid, Hyperbolic Tangent, and Rectified Linear Unit (ReLU) being some of the most popular (LECUN; BENGIO; HINTON, 2015). In addition, these models use an external parameter to the input data called bias b_k. The bias applies an affine transformation to the sum function's input values, increasing or decreasing the activation potential υ depending on whether its value is negative or positive (HAYKIN, 1998). In practical terms, the effect of bias is to modify the origin of the neuron output, as shown in Figure 4.

In the early stages of Neural Networks research, the foundational work of McCulloch and Pitts (1943) in the 1940s led to the creation of the first mathematical model to simulate the behavior of biological neurons, capturing their binary firing mechanism. The McCulloch-Pitts neuron model, represented by Figure 3, laid the groundwork for the subsequent development of the Perceptron by Rosenblatt (1958) in the late 1950s, introducing weighted inputs and an activation function. Although modern networks are easily composed of millions or billions of neurons, this was not the case with the Perceptron, known as the first Neural Network described algorithmically and the first neural model to learn via supervised learning (FORSYTH; PONCE, 2002). The Perceptron is often referred to as a "single-layer network", as it consists of a single neuron with adjustable weights and a bias, similar to the one shown in Figure 3. The main difference between this neuron model and the Perceptron is that instead of an arbitrary activation function, the Perceptron uses a hardbound function known as the Heaviside step function (HAYKIN, 1998), as shown in Equation 2.6.

$$f(x) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \geq 0, \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The Perceptron holds a special place in the history of Neural Networks not only for its pioneering work but because of its limitations (HAYKIN., 2010). Designed to learn linearly separable patterns with two classes only, the Perceptron is the simplest example of a Neural Network (ROSENBLATT, 1958). In fact, Rosenblatt (1958) proves via the Perceptron Convergence Theorem that if the patterns used to train the Perceptron are drawn from two linearly separable classes, the model converges and positions the decision boundary in the form of a hyperplane between the two classes. However, it was proved years later that Perceptron did not generalize toward the notion of binary parity. Among many other mathematical proofs and experiments presented in their book *Perceptrons* (MINSKY; PAPERT, 1969), Minsky and Selfridge demonstrate the Perceptron's limitation via the XOR problem. This classic problem consists of predicting the outputs of XOR logic gates given two binary inputs. An XOR function returns True if the two inputs are not equal and False if they are equal. Although this problem seems simple, Minsky and Papert (1969) showed that it was a big problem for the Perceptron architecture, given its limitations of linear nature. Since XOR is a non-linear separable problem, Perceptron could not draw a single hyperplane that separates the XOR classes. Such discovery discouraged the ML community from working with Neural Networks, which stagnated the research field for years (GOODFELLOW; BENGIO; COURVILLE, 2016). However, this would change with the emergence of Multi-layer Networks and the adoption of gradient descent applied to the backpropagation algorithm to train such networks (LECUN *et al.*, 1998).

2.2.1 Feed-Forward Neural Networks

Multi-Layer Neural Networks or *Feed-Forward Neural Networks (FFNN)* are fundamental architectures in modern ML, comprising interconnected neurons with non-linear activation functions in multiple layers (GAMA *et al.*, 2011). In addition to the input and output layers, these networks have what are called hidden layers, as they are hidden from the input and output signals (HAYKIN., 2010). FFNNs usually have at least two hidden layers, which allows them to capture complex patterns and deal with non-linearly separable data. In addition, this type of architecture often has a high degree of connectivity, as each neuron is connected to all neurons in the previous layer (NIELSEN, 2015). An example of a fully connected FFNN architecture with two hidden layers can be seen in Figure 5.

In the NNs literature, it is common to see a misunderstanding with the terms FFNN and *Multi-Layer Perceptron (MLP)*. Although MLP is an example of an FFNN, the original MLP used Sigmoidal neurons (CYBENKO, 1989). More precisely, and despite not using the same activation function, the MLP is defined as an extension of the Perceptron where there is an arbitrary number of neurons arranged in multiple layers instead of a single neuron (HASTIE *et al.*, 2009). The addition of hidden layers and the activation function non-linearity allowed MLPs to overcome some of the known limitations of the base model, such as the binary parity problem (NIELSEN, 2015). Another difference between Perceptron and MLP/FFNNs is how these models are trained.

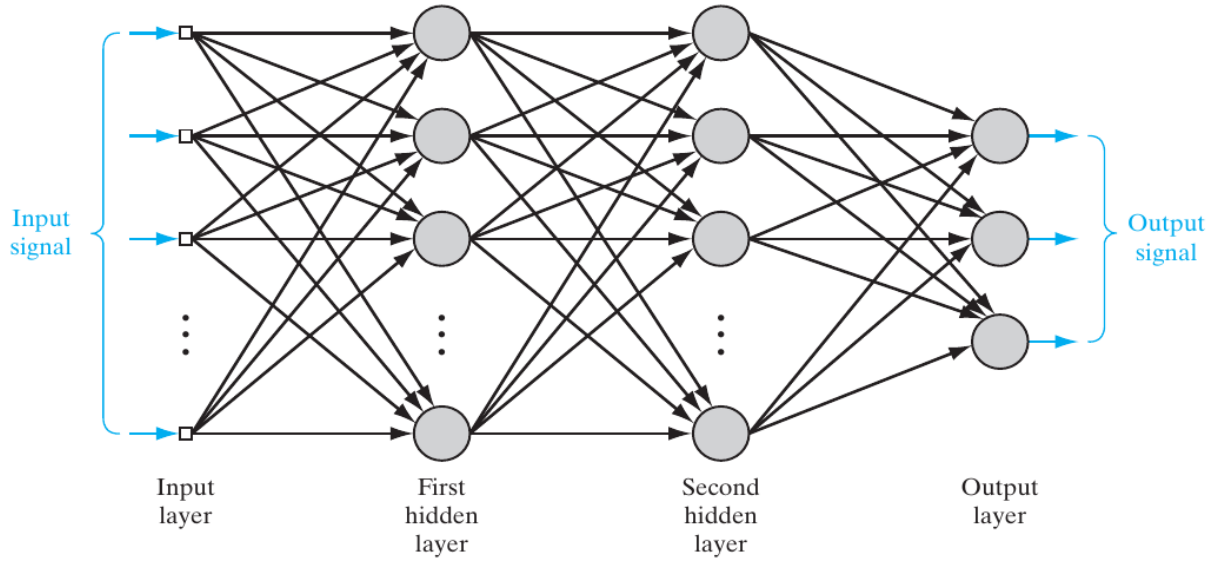


Figure 5 – Feed-Forward Neural Network with two hidden layers. Adapted from Haykin. (2010).

While Perceptron is trained by calculating the prediction error directly, as it has a single neuron, MLP/FFNNs cannot have their errors calculated directly for the entire network due to the existence of multiple hidden layers (ROSENBLATT *et al.*, 1962; HASTIE *et al.*, 2009). Therefore, these networks are commonly trained with an algorithm called Backpropagation (RUMELHART; HINTON; WILLIAMS, 1985). Another important distinction is that the activation functions used by MLP/FFNNs are differentiable, which allows the calculation of gradients and enables the use of Backpropagation to learn the weights (BENGIO; DUCHARME; VINCENT, 2000).

The development and successful application of Backpropagation and Stochastic Gradient Descent (SGD) are of vital relevance to the success of Deep Learning and Deep Neural Networks (GOODFELLOW; BENGIO; COURVILLE, 2016). SGD is an optimization algorithm used to minimize a Neural Network's loss function by iteratively updating its parameters (weights and biases) based on the function's gradient with respect to the parameters (AMARI, 1967). The "stochastic" aspect refers to the random mini-batches used for computing the gradients, as it does not use the entire training dataset at once (WERBOS, 2005). On the other hand, Backpropagation is the algorithm used to compute the loss function's gradients that SGD then uses to update the neural network parameters to improve its performance (LINNAINMAA, 1970). Backpropagation involves two main steps: (i) the forward pass, where input data is passed through the network to compute predictions and calculate the output error; and (ii) the backward pass, where the chain rule of calculus is used to calculate the gradients of the output error layer by layer and then propagated back from the output to the input layer (LEIBNIZ, 2012). In essence, SGD and Backpropagation, as well as FFNNs, are the cornerstones for almost any type of neural network, and both their composition and how they are applied nowadays are the fruit of an extended list of works over the years (LECUN; BENGIO; HINTON, 2015).

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (ConvNets) or CNNs are a type of neural architecture specialized in processing data with a grid-like topology (LECUN *et al.*, 1998). The most common use of this type of network is for pattern recognition in images, as they can be viewed as a 2D, 3D, or higher dimensional grid-like structure of pixels. However, the application cases for ConvNets are quite broad, and it is not uncommon to see them being applied to other types of data, such as Audio and Text (ELSKEN; METZEN; HUTTER, 2019). The name "Convolutional" in ConvNets refers to the mathematical operation called convolution, a specialized type of linear operation used by convolutional layers (LECUN *et al.*, 1989). In essence, Convolutional Networks are Neural Networks that use convolution in at least one of their layers and are not limited to general matrix multiplication, as is the case with FFNNs (KRIZHEVSKY; HINTON *et al.*, 2009).

An example of what a ConvNet architecture looks like can be seen in Figure 6. This specific network, called LeNet-5 (LECUN *et al.*, 1998), was a pioneering architecture and one of the first examples of what became the modern convolutional architectures known today. As seen in Figure 6, the LeNet-5 is composed of two convolutional layers interspersed with pooling layers that subsample the feature maps generated by the convolution operations. At the end of the network, fully connected layers, similar to FFNNs, are trained with the outputs of the feature maps reduced by pooling, feeding the output layer that classifies the examples according to a probability distribution function. Although some authors claim that fully connected layers are not part of ConvNets but an add-in, it is quite common to see works nowadays that consider them as any other part of the ConvNet architecture.

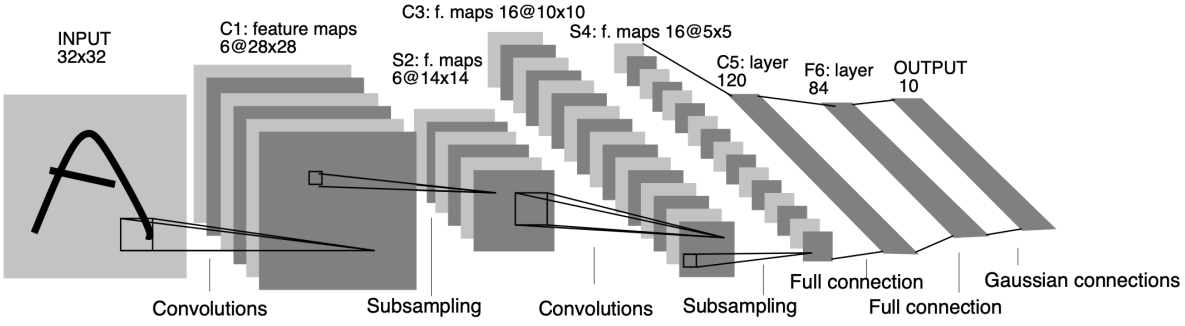


Figure 6 – LeNet-5 architecture. Adapted from LeCun *et al.* (1998).

Convolutional architectures employ three essential components with specific roles: *local receptive fields*, *shared weights*, and *subsampling* (NIELSEN, 2015). As in an FFNN, the layers of a ConvNet also contain neurons. However, unlike the neurons of a regular NN connected to all the input signals, neurons in a ConvNet are only connected to small regions of the input data. These regions are called local receptive fields and determine the spatial extent of the input that the neurons will process. The *filters* or *kernels* are the learnable parameters of those neurons that extract specific features from the local regions. As shown in Figure 7, when a filter is convolved

through the input data, it slides over the input by computing dot products between the filter's weights and values in the receptive field. This process results in a *feature map*, or *activation map*, that highlights the presence of the learned features. In addition, as the input data is passed through successive layers, the receptive fields of higher-level neurons capture more complex patterns by combining information from lower-level receptive fields. This hierarchical processing enables the ConvNet to learn increasingly abstract and meaningful features (REN *et al.*, 2021).

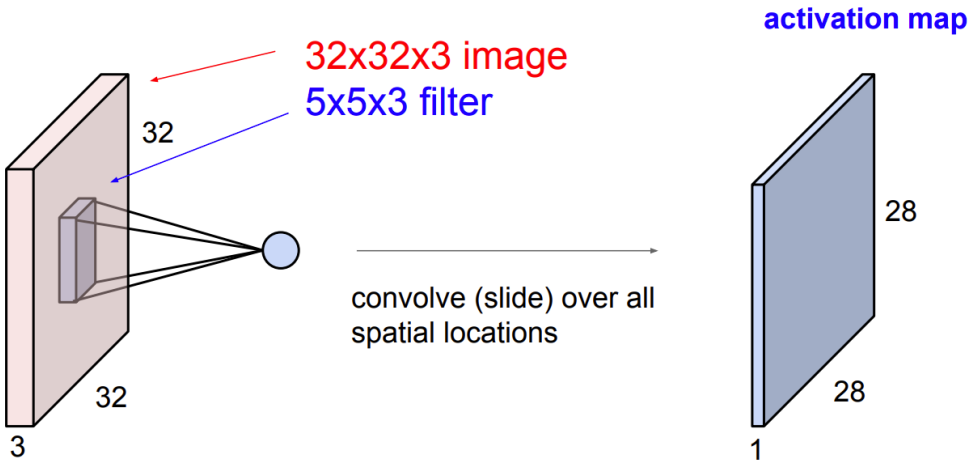


Figure 7 – Convolution operation with a 5x5x3 kernel over a 32x32x3 image. Adapted from the Convolutional Neural Networks for Visual Recognition Stanford Course, 2020 [2].

Opposite to what happens with FFNNs, where each neuron has its own weights in a grid of dense connections, the neurons of a convolutional layer have their learned weights (kernels) shared among themselves (HAYKIN., 2010). The weight sharing used by ConvNets makes it necessary to learn only one set of weights per layer instead of learning a separate set of weights for each input location. This causes the number of weights to be drastically reduced when compared to a traditional FFNN, making it less dense and, consequently, lighter and with more local features. Therefore, the convolution operation is dramatically more efficient than dense matrix multiplication in terms of memory requirements and statistical efficiency (GOODFELLOW; BENGIO; COURVILLE, 2016). In a traditional neural network, each element of the weight matrix is used precisely once, and after calculating the output of a layer, it is never revisited. In ConvNets, however, they share kernels and local receptive fields, which creates certain redundancy between neurons and, often, feature maps (NIELSEN, 2015).

The last key component of ConvNets is the subsampling or, as it is commonly called, the Pooling operation (WITTEN *et al.*, 2016). The Pooling is responsible for reducing the input dimensions of previous layers, and this operation is usually applied after a convolution, condensing the previously extracted information. In short, the most common pattern to observe is to have a sequence of convolutional layers followed by pooling layers, reaching the fully connected layers at the end of the feature extraction part of the network. As a convolutional layer usually generates multiple feature maps and its dimensions grow quickly, pooling is extremely important to keep the training of Convnets feasible, which is thus applied to each

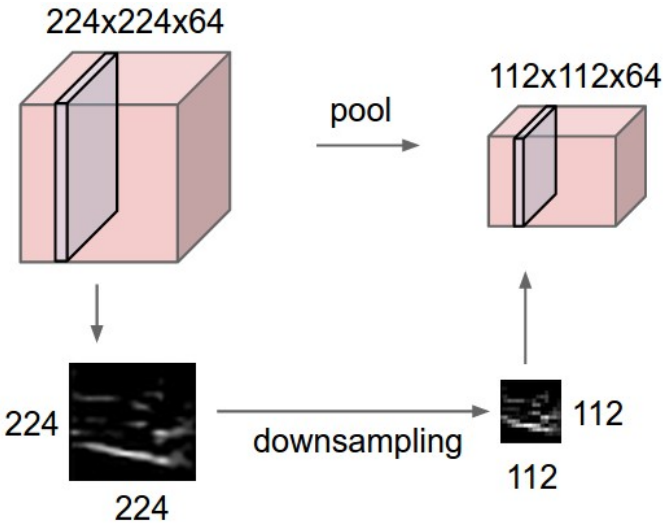


Figure 8 – General effect of the Pooling/Subsampling operation. Adapted from the Convolutional Neural Networks for Visual Recognition Stanford Course, 2020 [2].

map separately. Figure 8 illustrates how pooling works in practice. There are different ways to perform pooling, and two popular operators are Max Pooling and Average Pooling (LECUN *et al.*, 1998). Max Pooling captures the maximum activations in a specific region according to the size of the pooling kernel. In turn, Average Pooling extracts an average of the activations in a kernel. Figure 9 illustrates how these two operators work in practice. Given an activation map (Left), Max Pooling extracts the maximum activations using a 2x2 kernel with stride 2, moving two positions horizontally and two vertically. On the other hand, the Average pooling (Right) takes an average of activations over a 2x2 kernel with stride 2.

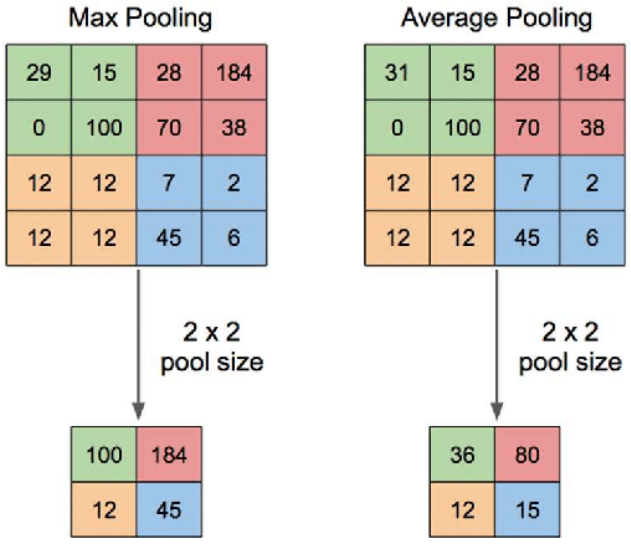


Figure 9 – Max and Average Pooling operations. Adapted from Yani *et al.* (2019).

[2] http://cs231n.stanford.edu/2020/

2.2.3 U-Net

The U-Net architecture proposed by Ronneberger, Fischer and Brox (2015) has emerged as a groundbreaking ConvNet-like model developed for biomedical image segmentation tasks. This architecture was specifically designed to excel in per-pixel segmentation, exhibiting exceptional optimization and performance (RONNEBERGER; FISCHER; BROX, 2015). In fact, its impact and application extend beyond the field of medical imaging, reshaping the landscape of semantic segmentation in general (YAN *et al.*, 2020b). The U-Net's effectiveness lies in its fusion of a contracting path for capturing context and a symmetrical expanding path for precise localization. Consequently, the expansive path exhibits a symmetrical relationship with the contracting counterpart, yielding a distinct U-shaped architectural configuration which is similar to the idea of an autoencoder, as can be observed in Figure 10. This synergy between the two paths enables U-Net to deal with tasks like image segmentation and generation by capturing contextual information and producing precise results. The reason why encoder-decoders are relevant is that they can capture the abstract representations from the input and preserve the original dimensions. While the encoder extracts relevant features from images, the decoder takes those features and reconstructs them in segmentation masks.

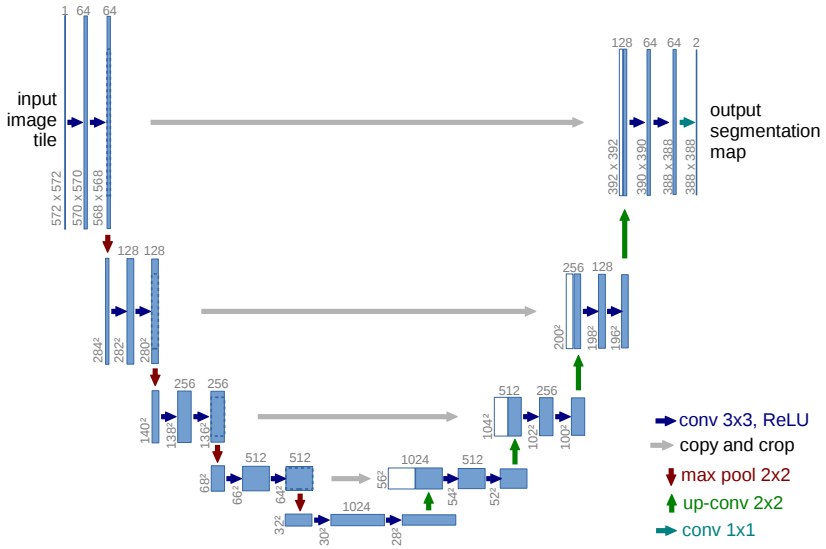


Figure 10 – U-Net architecture. Adapted from Ronneberger, Fischer and Brox (2015).

Another inspiration for the U-Net can be traced back to the Fully Convolutional Network introduced by Long, Shelhamer and Darrell (2015). This Neural Network effectively enhanced a standard contracting ConvNet by adding sequential layers with upsampling operators instead of pooling operations to incrementally boost output resolution. Another distinctive feature of U-Net is the incorporation of a significant number of feature channels in the upsampling segment path (WENG *et al.*, 2019). This enhancement facilitates the propagation of context information to higher-resolution layers, which results in an expansively symmetric architecture that mimics the contracting counterpart. In particular, this neural network exclusively uses convolutional

layers to generate segmentation outputs in the form of segmentation masks, thus avoiding the usage of fully connected or dense layers. For pixel-accurate prediction along the border region of the images, contextual information gaps are filled by extrapolating the missing context by mirroring the inputs. This strategy allows training the network with larger images and helps to overcome resolution size limitations linked to GPU memory consumption.

2.3 Neural Architecture Search

Neural Architecture Search or NAS is a recent research topic that arose from works that explored the automated design of Neural Networks (ZOPH; LE, 2016). This research field covers the automatic definition of various hyper-parameters related to neural architectures, such as layer operations, activation functions, and how layers are connected, among others (ZOPH *et al.*, 2018). Thus, NAS methods aim to automate the entire engineering process of neural architectures, and their functioning can be seen as the result of three principal components (HUTTER; KOTTHOFF; VANSCHOREN, 2019): Search Space, Search Strategy, and Performance Estimation. Figure 11 illustrates these components and how they usually interact with each other. In the following sections, each of them is explored in more detail.

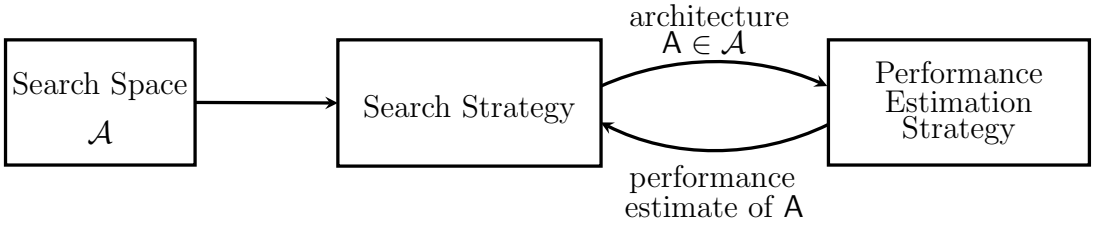


Figure 11 – The three basic components of the NAS framework. From a Search Space $\mathcal{A}$, a Search Strategy selects an architecture A which has its performance estimated by a Performance Estimation Strategy. Adapted from Hutter, Kotthoff and Vanschoren (2019).

2.3.1 Search Space

The Search Space in the context of NAS defines which architectures can be found or learned in principle (LIU *et al.*, 2017). Depending on how the architecture space is defined, the search can be simplified, and the computational cost associated with finding good architectures can be reduced. However, this is not a trivial task since oversimplifying the search space can compromise the quality of candidate architectures. On the other hand, finding suitable architectures in a large and complex space can be computationally infeasible. Therefore, there is a trade-off between architecture's quality and the feasibility of searching in the space of solutions (BAKER *et al.*, 2017). In Figure 12, two search spaces commonly used in the literature are illustrated: a relatively simple one called Chain-structured space (REAL *et al.*, 2019); and a more complex one called Multi-branch space (LIU *et al.*, 2018).

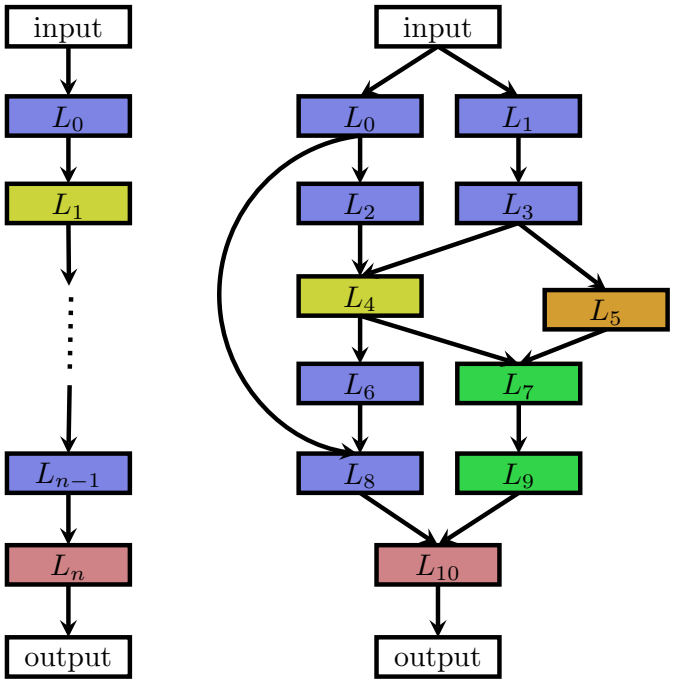


Figure 12 – Types of Search Space: (Left) Chain-structured space; and (Right) Multi-branch space. Colored nodes represent hidden layers, edges represent layers' inputs and outputs, and each color represents a layer type. Adapted from Hutter, Kotthoff and Vanschoren (2019).

A Chain-structured space consists of a sequence of n layers, where a layer i receives the output of a layer $i-1$ as input and, consequently, its output is used as input for a layer $i+1$. This space can be customized by: (1) maximum and minimum number of hidden layers; (2) type of layers; and (3) specific hyper-parameters of each layer. Since the layers' hyper-parameters (3) are dependent on their type (2), the search space changes to a variable size, making the search more difficult (REAL *et al.*, 2019). In turn, a Multi-branch space is made of conditional connections that allow the building of more complex neural networks. In the current NAS literature, the Multi-branch space is frequently adopted for CV tasks (LIU *et al.*, 2018). Moreover, there is a third search space called Cell-based (LIU *et al.*, 2019), which is illustrated in Figure 13.

Motivated by the recurrence of certain mechanisms observed in popular architectures, such as convolutional layers followed by pooling operations in ConvNets, the Cell-based Search Space emerged as a novel approach that consisted of searching for such motifs (LIU *et al.*, 2020). In works such as (ZOPH *et al.*, 2018) and (ZHONG *et al.*, 2018), the focus has shifted from optimizing entire architectures to the optimization of individual mechanisms, referred to as "blocks" or "cells." This optimization strategy often involves two types of cells: A normal cell that preserves input dimensionality, and a reduction cell that applies a transformation that reduces the input's dimensions (LUO *et al.*, 2018). After being searched and optimized, these cells are systematically stacked in a predetermined manner to compose the final architectures. Overall, the Cell-based space has two main advantages over the Chain-structured and Multi-branch search spaces (ZHONG *et al.*, 2018): (*i*) the search space is considerably smaller, leading to searching

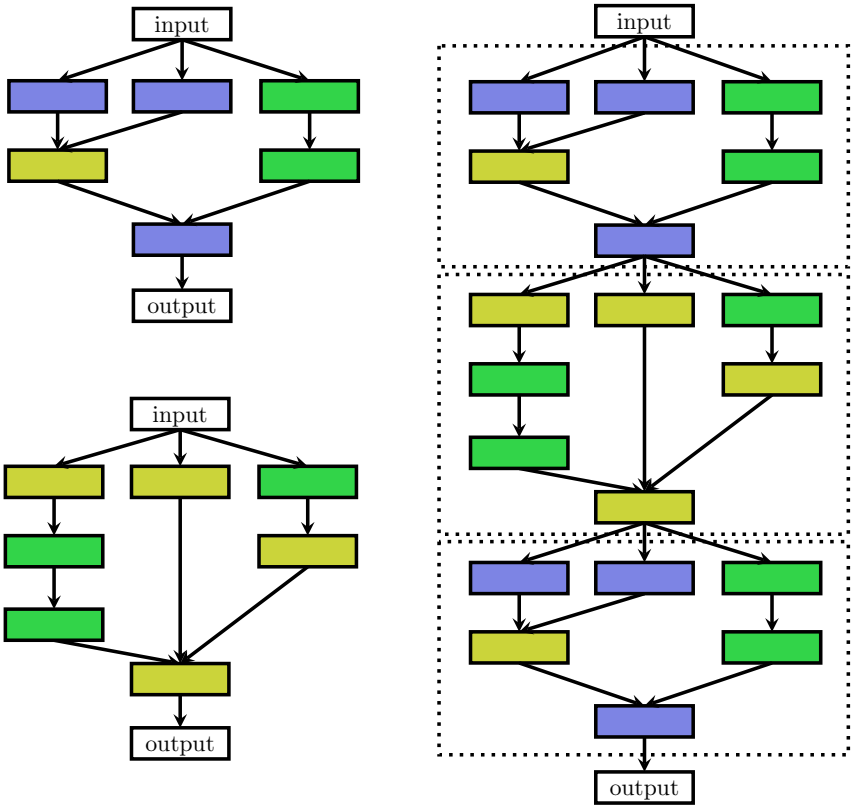


Figure 13 – Cell-based Space: (Top Left) normal cell; (Bottom Left) reduction cell; (Right) an architecture built by stacking the cells.

speed-up; And (*ii*) it offers greater flexibility for transferring knowledge across similar tasks once cells can be easily added or removed. However, the Cell-based space introduces new meta-architecture hyper-parameters, including possible cell types, the minimum and maximum number of cells in an architecture, and how cells are connected. Defining these new parameters adds extra difficulty to the problem and requires a solution by itself. According to Hutter, Kotthoff and Vanschoren (2019), to avoid search oversimplification and ensure automation feasibility, it is necessary to adjust the meta-parameters along with the NAS process.

2.3.2 Search Strategy

Once a Search Space is defined is necessary to establish how searching for architectures in such space will be performed (LUO *et al.*, 2018). The Search Strategy thus defines how architectures will be sampled in the space of possible configurations, and, in some cases, even determines how this space is modified during search and optimization (ELSKEN; METZEN; HUTTER, 2019). Algorithms with different learning paradigms have been used to explore the space of candidate architectures, the most common being Reinforcement Learning (RL), Evolutionary Algorithms (EAs), Bayesian Optimization (BO), and Gradient Descent (GD) (WISTUBA; RAWAT; PEDAPATI, 2019). Although a considerable amount of work has been reporting advances in NAS for various tasks, it seems that a unanimous superiority of one family of

algorithms over others has not yet been proven (WHITE *et al.*, 2023). This may be the case because the NAS framework involves different components and associated hyperparameters, making it difficult to conduct reliable ablation studies (LI; TALWALKAR, 2020).

The first Search Strategy to become widely used in NAS was Reinforcement Learning (ELSKEN; METZEN; HUTTER, 2019), where its algorithms and paradigm were used as a framework to generate models and guide the search for optimal architectures, as seen in Figure 14. In the pioneering work of Zoph and Le (2016), in which the term "Neural Architecture Search" was coined, Reinforcement Learning was applied to find novel and competitive architectures to rival the best hand-crafted architectures designed for CIFAR-10. Recurrent Neural Networks (RNN) were used to generate model descriptions of neural architectures, and these same RNNs were trained with the REINFORCE algorithm to maximize the expected accuracy of such generated architectures. However, the proposed method showed efficiency problems, requiring about 800 GPUs running for three to four weeks to completely train the models. After this work, many other methods were proposed with the concern of reducing the computational cost of NAS. In Zoph *et al.* (2018), an improvement of Zoph and Le (2016) was proposed using a different RL policy called Proximal Policy Optimization (PPO), which generated better predictions for CIFAR-10 using 450 GPUs and less than half of the parameters. Another example of RL application is seen in Pham *et al.* (2018), where a controller LSTM is trained with a policy gradient to select a subgraph from a large graph of neural architectures that maximizes the expected reward on a validation set. Experimental results on CIFAR-10 showed a 1000x speed-up compared to Zoph and Le (2016) using one GPU only.

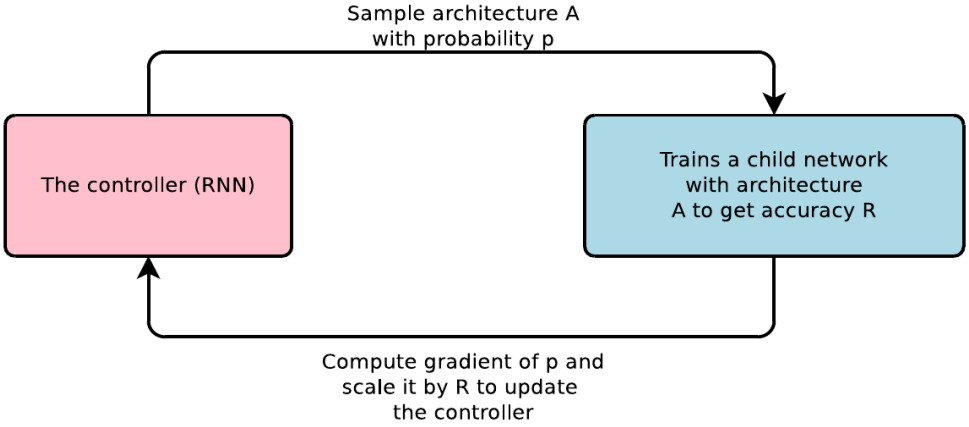


Figure 14 – NAS from a Reinforcement Learning perspective. Adapted from Zoph and Le (2016).

Evolutionary Algorithms used as Search Strategies to generate neural architectures have been used since the mid-1980s (THRUN; PRATT, 1998). One of the first attempts to evolve architectures through Genetic Algorithms (GAs) is seen in Miller, Todd and Hegde (1989), in which new architectures are discovered via evolutionary process and trained using Backpropagation (LECUN *et al.*, 1989). More recent works have used GAs to optimize neural topologies and weights, differing in how individuals are selected, which crossovers are used, and how mutations are performed (WEI *et al.*, 2022). In Real *et al.* (2017), Liu *et al.* (2017), and Real

et al. (2019), Tournament Selection is employed (WETZEL, 1983), while in Elsken, Metzen and Hutter (2018), architectures are selected according to the Pareto Front (BANDARU; NG; DEB, 2017). In Real *et al.* (2017), the worst individual in a population of candidate architectures is removed at the end of each generation, whereas in Real *et al.* (2019), only the oldest individual is removed, and in Liu *et al.* (2017), there are no removals. In Elsken, Metzen and Hutter (2018), the creation of a new offspring or child models occurs through Morphisms (WEI *et al.*, 2016), where weights from parent architectures are transferred to the offspring architectures. Morphism is also applied in Real *et al.* (2017), but offspring architectures only receive parameters that were not affected by mutation in the parent models.

Bayesian Optimization (BO) has a long history of successful applications in ML, particularly in the context of fine-tuning learning algorithms' hyperparameters (BRAZDIL *et al.*, 2008). In the field of NAS, BO solutions have also shown promising results, as evidenced by the pioneering works of Bergstra, Yamins and Cox (2013) and Domhan, Springenberg and Hutter (2015), where state-of-the-art performance on well-known image datasets was demonstrated. However, it is worth noting that BO methods face fierce competition from Evolutionary and Reinforcement Learning approaches. An example is the work of Kandasamy *et al.* (2018), who made notable strides by deriving kernel functions to adapt Bayesian methods for architectural search spaces. Nevertheless, achieving new state-of-the-art in NAS remains a challenge for BO-based methods. In this dynamic landscape, BO continues to hold promise, and further advancements may enhance its efficacy in NAS optimization, ultimately contributing to the field.

Gradient Descent plays a pivotal role in NAS by enabling the efficient discovery of optimal neural network architectures (XIE *et al.*, 2018). Recent advancements have introduced innovative approaches like combining the weight-sharing paradigm with continuous relaxation of the search space, thus allowing for the use of gradient-based optimization techniques (XIE *et al.*, 2018). This approach, collectively known as differentiable NAS, has proven to be highly efficient in exploring the vast neural architecture search spaces. DARTS is one of the popular methods that stands out as one of the most prominent algorithms among the gradient-based NAS solutions (LIU; SIMONYAN; YANG, 2018). Nevertheless, DARTS faces challenges such as performance collapse due to skip connection aggregation and limited generalization (XIE *et al.*, 2018). Fortunately, numerous subsequent algorithms have addressed these issues (XIE *et al.*, 2018). For instance, some focus on enhancing DARTS' robustness and smoothing the landscape of validation accuracy through Hessian norm-based regularization and random smoothing/adversarial attacks (XIE *et al.*, 2018). Further investigation into the performance degradation aspect is undertaken from the architecture selection perspective (XIE *et al.*, 2018). Differentiable NAS has demonstrated its ability to achieve competitive results with a fraction of the search time required by traditional reinforcement learning-based search methods. Additionally, other gradient-based NAS techniques, such as SNAS (XIE *et al.*, 2018), ProxylessNAS (DONG; YANG, 2019), and GDAS (CAI; ZHU; HAN, 2018), have emerged as promising alternatives, offering diverse strategies to streamline architecture search and deliver efficient neural models.

2.3.3 Performance Estimation

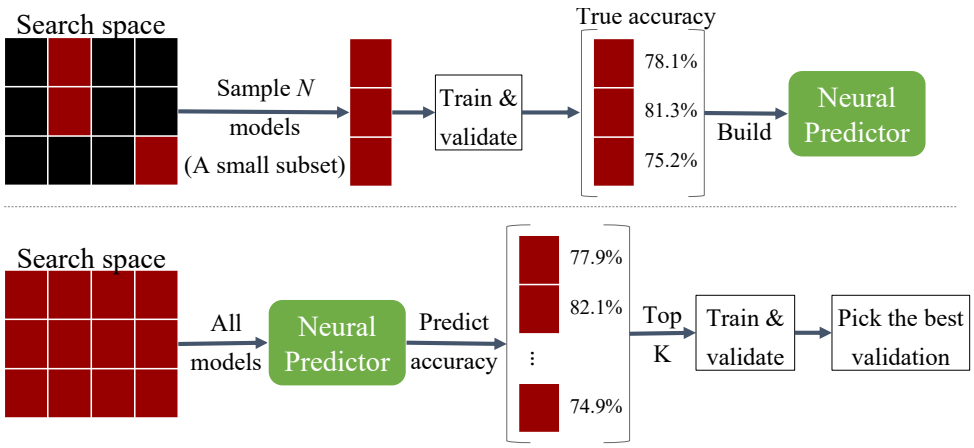


Figure 15 – Performance Prediction example on NAS. Adapted from Wen *et al.* (2020).

NAS methods aim to automatically find good neural architectures in a limited time, and since most search spaces can easily contain thousands or millions of architectures, evaluating each candidate's actual performance becomes a complex and sometimes computationally unfeasible process (BAKER *et al.*, 2017). Therefore, one important component of the NAS framework is the Performance Estimation Strategy, which determines how to estimate one architecture candidate's performance without actually training it enterily (WHITE *et al.*, 2021). Figure 15 exemplifies one possible configuration for this component, which consists of collecting data from sampled architectures and training a model that predicts the real performance of unseen models. In that regard, recent research has been looking for alternatives to reduce the computational burden and improve architectural performance estimation quality. In the current literature, some strategies have stood out (ELSKEN; METZEN; HUTTER, 2019): (i) Partial estimations; (ii) Learning curve exploration; (iii) One-shot models; (iv) Prediction-based NAS; and (v) Training-free NAS. In the following, each one of these strategies is presented in more detail.

Partial estimates or *Low fidelities* are the most basic performance approximators, including everything from reduced training times to training on subsets of data or training on data with reduced dimensions (ELSKEN; METZEN; HUTTER, 2019). In Zela *et al.* (2018), both the neural architecture and its weights are optimized jointly, where different training epochs were used to verify the performance correlation of adjusting the weights or not. In Klein *et al.* (2016a), optimization is accelerated through training on subsets from the original dataset. First, preliminary explorations were made in the subsets as a form of validation, and, in a second moment, if promising results were generated, extrapolations in the complete dataset were performed. In turn, in Chrabaszcz, Loshchilov and Hutter (2017), the performance estimation strategy is based on simplified versions of the ImageNet dataset (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), in which the image resolutions were decreased to speed up the search and validation process.

Learning curve exploration is another way to estimate the performance of candidate models, in which, as well as Partial estimates, it is based on partial information from the real output signals related to performances (RIJN *et al.*, 2015). Learning curves are basic graphical representations that show how a metric changes over time. In the context of ML and NAS, these curves are often the progress of a performance metric, such as accuracy or error, throughout the training or search (WISTUBA; PEDAPATI, 2020). Thus, the Learning curve exploration strategy consists of analyzing the history values of a metric (curve) from candidate architectures to select the most promising (KLEIN *et al.*, 2016b). In Domhan, Springenberg and Hutter (2015), a method speeds up deep neural network hyperparameter search with a probabilistic learning curve model. It predicts network performance early, improving hyperparameter optimization on benchmarks like CIFAR-10. "F-Hyperband" by Baker *et al.* (2017) predicts neural network performance using architecture features, hyperparameters, and data, accelerating optimization without disruption. Rawal and Miikkulainen (2018) enhances neural architecture via evolutionary optimization on LSTM nodes, outperforming manual designs in language modeling and music prediction. Finally, Liu *et al.* (2018) introduces PNAS, efficiently designing CNNs with progressive search and a performance predictor, achieving results comparable to existing methods with fewer resources.

One-Shot Architecture Search stands out as another estimation strategy that aims to speed up the search for high-performance neural architectures (BENDER *et al.*, 2018). In this approach, candidate architectures are treated as different sub-graphs within a super-graph known as the One-shot model, where weights are shared between sub-graphs with common edges (WHITE *et al.*, 2023). Notably, since this technique requires only the One-shot models' weights to be learned, this significantly accelerates the performance estimation while eliminating the need for isolated training of each architecture in the super-graph (SIEMS *et al.*, 2020). Two pioneering works in this field that use this approach are the Efficient Neural Architecture Search (ENAS) (PHAM *et al.*, 2018) and Differentiable Architecture Search (DARTS) (LIU; SIMONYAN; YANG, 2018). ENAS is the pivotal work to achieve efficient training by weight-sharing among child models in the super-graph during architecture search, reducing computational complexity and enhancing stability and architectural diversity through the combination with Reinforcement Learning (PHAM *et al.*, 2018). On the other hand, DARTS employs a continuous relaxation of the architecture space, enabling Gradient-based optimization. This approach delivers competitive performance compared to non-differentiable methods, particularly in Image Classification and Language Modeling tasks (LIU; SIMONYAN; YANG, 2018). DARTS' simplicity, scalability, and efficiency enhancements solidify its position as a significant contribution to automated architecture search, making it a mainstream paradigm and a frequent baseline in the field.

Prediction-based NAS emerged as an alternative strategy to tackle the complex challenges seen in other NAS performance estimation methods (KLEIN *et al.*, 2016b; DUDZIAK *et al.*, 2020). By leveraging ML models to predict the performances of candidate architectures, this strategy offers a compelling solution to reduce computational costs and accelerate architectural search while using a small number of sampled architectures(LI *et al.*, 2021). Although

Prediction-based NAS shows promising results in minimizing computational burden by making informative predictions on a limited number of sampled architectures, it comes with an additional set of associated challenges (WHITE *et al.*, 2021). Accurately estimating neural architecture performance is a complex task that requires well-designed and tuned models able to capture the intricate relationships between architecture designs and their performances (NING *et al.*, 2020). Moreover, the quality and diversity of the sampled architectures significantly influence the reliability of these predictions (BAKER *et al.*, 2016). Furthermore, the Prediction-based strategy may have limitations in exploring architectures beyond the sampled pool, potentially restricting its ability to uncover innovative designs outside the sampled domain (LIU; TANG; SUN, 2021). However, with just a few data points, Prediction-based NAS can achieve better or competitive results with the most complex techniques in the area, such as One-shot models, and often in less training time and fewer computational resources.

Training-free NAS is a recent paradigm that aims to speed up architecture selection based on surrogate metrics of real performances instead of extensive training (WHITE *et al.*, 2023). Mellor *et al.* (2021) introduced this pioneering concept by extracting linear mappings of untrained networks at initialization to create a distinctive kernel matrix. Therefore, a ranking of the candidate architectures is determined by scores derived from the kernel matrix, with higher values indicating the potential for improved accuracy post-training. While Training-free NAS promise to reduce computational costs while accelerating the architectural search process, they often rely on heuristics, which limits their ability to capture complex relationships within network components and potentially leads to sub-optimal solutions. To address these challenges, Chen, Gong and Wang (2021) presents an innovative approach to create optimized architectures for trainability and expressiveness. Leveraging two key metrics $\kappa_{\mathcal{N}}$ and $\hat{R}_{\mathcal{N}}$, this proposal evaluates and enhances architectures by assessing their trainability and capacity to capture complex data relationships. $\kappa_{\mathcal{N}}$ evaluates a network's capability to undergo stable and efficient training by examining the spectral characteristics of the Jacobian matrix, where lower values of $\kappa_{\mathcal{N}}$ represent architectures that facilitate smooth training processes. In contrast, $\hat{R}_{\mathcal{N}}$ measures a network's expressiveness by quantifying its aptitude for capturing intricate data relationships, where higher values of $\hat{R}_{\mathcal{N}}$ reflect the effectiveness of the architecture in modeling complex patterns.

2.4 Meta-Learning

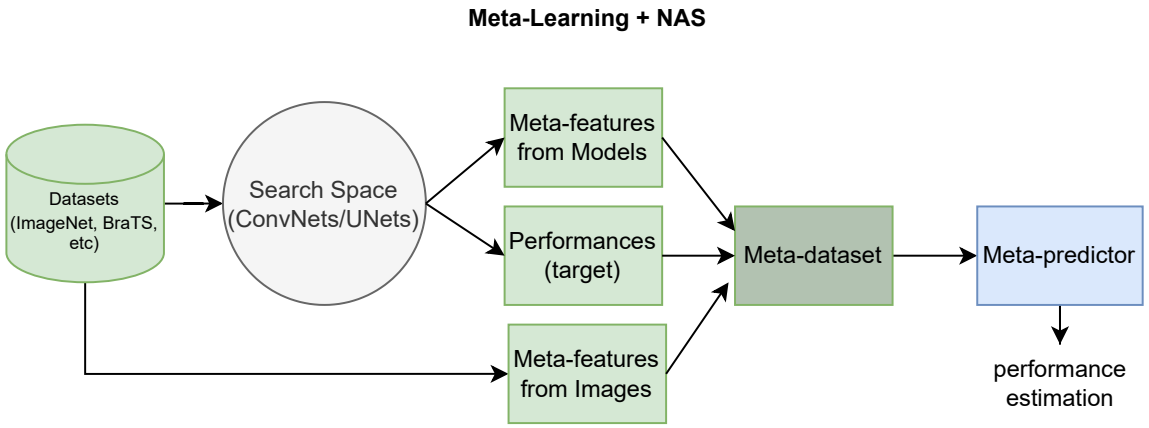


Figure 16 – Meta-Learning applied to NAS.

Human beings hardly start from scratch or without any prior knowledge when solving an unknown problem (THRUN; PRATT, 1998). Human behavior is inherently predisposed to reuse previous and successful strategies, either to solve tasks quickly or to acquire new skills (LAKE *et al.*, 2017). Even when confronted with unfamiliar territories, the ability to generalize behaviors across diverse tasks assists the speeding up of problem-solving and skill acquisition (FINN; ABBEEL; LEVINE, 2017). According to Thrun and Pratt (1998), these factors are regarded as the pivotal reason why humans can efficiently adapt to novel challenges and adversities. An approach that allows using prior knowledge to optimally solve new tasks is known as Meta-Learning. Figure 16 exemplifies the Meta-Learning framework applied to the NAS problem, with a particular emphasis on the Prediction-based approach. Starting from a Dataset, a Search Space of possible models is meticulously defined for the task at hand. With such a Dataset and defined Search Space, an array of meta-information is systematically gathered from these sources. This includes extracting meta-features from the task, such as image-based characteristics when dealing with images, architectural meta-features when using Neural Networks, and performance-related statistics from the models, to name a few examples. Therefore, a meta-dataset can be built by combining all this meta-information, which can be used to induce a meta-predictor capable of estimating the performance and selecting unseen ConvNets/UNet.

In contrast, traditional learning algorithms find it difficult to reuse acquired knowledge, requiring the arduous process of training the models from scratch for every single task (VILALTA; DRISSI, 2002). Another common difficulty is learning from fewer examples (RAVI; LAROCHELLE, 2016). For many algorithms, large volumes of data are needed to effectively learn something and the gained knowledge can only be transferred to very closely resembling tasks. This data inefficiency and lack of transferability make it challenging to apply these models in applied scenarios where data is scarce or expansive to collect, such as in medical and biomed-

ical tasks in which the data volume is usually low (BRAZDIL *et al.*, 2008). Meta-Learning, sometimes referred to as "Learning to Learn" (FINN; LEVINE, 2017), on the other hand, distinguishes itself from the traditional learning paradigms, or Base-Learning, by taking advantage of previous experiences in tasks, algorithms, and performance metrics to learn how to solve new tasks faster, using fewer examples, and less computational resources. In addition, Meta-Learning has a higher abstraction level and knowledge generalization power, not being conditioned to a specific learning process, task, or dataset (PAVEL; SOARES, 2002).

A task in Meta-Learning is fundamentally defined in the same way as in Base-Learning, and it comprehends a diverse range of domains such as Classification, Regression, Segmentation, among others (BRAZDIL *et al.*, 2008). Thus, what distinguishes these two learning paradigms is the level at which the learning process takes place. While Base-Learning involves learning from data features and knowledge is built from scratch, Meta-Learning leverages prior learning experiences represented by meta-features. In fact, Meta-Learning starts from the idea that once some knowledge is acquired, building new knowledge becomes easier (LAKE *et al.*, 2017). Thus, it aims to smooth empirical processes like trial-and-error and encourage decision-making grounded on the knowledge extracted from known data. To train a meta-model, it is first necessary to collect meta-data and build a meta-dataset, and there are at least three sources of meta-knowledge (HUTTER; KOTTHOFF; VANSCHOREN, 2019): (*i*) Learning from Tasks; (*ii*) Learning from Model Evaluations; and (*iii*) Learning from Prior Models. In the following sections, each Meta-Learning approach is presented in detail.

2.4.1 Learning from Tasks

This first source of meta-knowledge consists of learning from meta-characteristics directly extracted from the tasks or, more precisely, from the datasets. Each learning task $t_j \in T$ can be characterized by a meta-features vector $\mathbf{m}_j = \{m_1, ..., m_k\}$ where k represents a single meta-feature in the K set of all known meta-features (HUTTER; KOTTHOFF; VANSCHOREN, 2019). From this meta-feature vector $\mathbf{m}$, it is possible to train a meta-model to make predictions based on these meta-features instead of relying on the original data, thereby often enhancing the model's generalization capacity. As the process of extracting meta-features is typically fast and computationally cheap, applying such a Meta-Learning approach is not only feasible but often beneficial in terms of efficiency and performance (BRAZDIL *et al.*, 2008). Furthermore, even a dimensionality reduction while maintaining or improving predictive performance can be expected to happen in some cases (ALCOBAÇA *et al.*, 2018).

Meta-features describe more general characteristics inherent to the data. These characteristics can range from elementary attributes, like the enumeration of classes within a dataset, to more complex descriptors, such as the Skewness and Kurtosis statistical metrics of a sample distribution (RIVOLLI *et al.*, 2018). As a matter of fact, the realm of meta-features includes a

broad and diverse range of categories, encompassing metrics from Clustering, Data Complexity, Information Theory, Model-based, and Landmarking [3], to name a few (ALCOBAÇA *et al.*, 2020). These categories encapsulate a rich tapestry of meta-features that enable comprehensive data characterization (RIVOLLI. *et al.*, 2022). Table 1 presents several of these meta-features commonly employed in the Meta-Learning literature along with their categories.

Table 1 – Types of meta-features often used in the literature.

Description	Type	Reference
No. of Examples	General	Michie *et al.* (1995)
No. of Features	General	Kalousis and Theoharis (1999)
No. of Numerical Features	General	Engels and Theusinger (1998)
Correlation Coef.	Statistical	Castiello, Castellano and Fanelli (2005)
Covariance	Statistical	Castiello, Castellano and Fanelli (2005)
Kurtosis	Statistical	Michie *et al.* (1995)
Shannon's Entropy	Info. Theory	Michie *et al.* (1995)
Features Noisiness	Info. Theory	Michie *et al.* (1995)
Mutual Information	Info. Theory	Michie *et al.* (1995)
Imbalance Ratio	Complexity	Lorena *et al.* (2019)
Collective Feature Efficiency	Complexity	Lorena *et al.* (2019)
Overlapping Region Volume	Complexity	Lorena *et al.* (2019)
Calinski and Harabasz index	Clustering	Caliński and Harabasz (1974)
Mean Silhouette Value	Clustering	Rousseeuw (1987)
Dunn index	Clustering	Dunn (1974)
Linear Discriminant Performance	Landmarking	Bensusan and Giraud-Carrier (2000)
Naive Bayes Performance	Landmarking	Bensusan and Giraud-Carrier (2000)
1-Nearest Neighbor Performance	Landmarking	Bensusan and Giraud-Carrier (2000)
No. of Leaves	Model-based	Pavel and Soares (2002)
No. of Branches	Model-based	Pavel and Soares (2002)
Tree Depth	Model-based	Pavel and Soares (2002)

2.4.2 Learning from Model Evaluations

The second source of meta-knowledge lies in the Learning from Model Evaluations or, more precisely, from the intricate mapping between the attributes of tasks and the corresponding model performances on those tasks. This approach can be defined in terms of tasks $t_j \in T$, algorithms $a_i \in A$, their associated hyper-parameters $\theta_i \in \Theta$, and a set of real-valued evaluations $p_{i,j} = P(\theta_i, t_j)$, according to previously defined performance measures. Hence, it becomes feasible to induce a meta-model capable of leveraging the attributes of an incoming task drawing from the repository of prior experiences encapsulated in T to offer recommendations encompassing an algorithm a_i which is fully defined by θ_i (HUTTER; KOTTHOFF; VANSCHOREN, 2019). Although this Meta-Learning approach primarily revolves around leveraging past model performances and is typically employed to make model recommendations for specific tasks, its methodology shares similarities with both the Learning from Tasks and Learning from Prior Models approach, as it can use information from these two sources.

[3] Quick estimates obtained by simplified versions of the algorithms or reduced datasets

When it comes to recommending models, a broad spectrum of Machine Learning models can be effectively employed as meta-models (PFAHRINGER; BENSUSAN; GIRAUD-CARRIER, 2000). This repertoire spans everything from traditional algorithms like Support Vector Machines, Random Forests, and Decision Trees to contemporary models such as UNets and Transformers. Augmenting these meta-models, repositories composed of rich meta-data, including the popular Aslib (BISCHL *et al.*, 2016) and OpenML (VANSCHOREN *et al.*, 2014), emerge as invaluable troves of information. Meta-models can thus leverage the extensive data on optimization problems, such as the Algorithm Recommendation (AR) Problem, the Traveling Salesman Problem (TSP), and the Satisfiability Problem (SAT), among other records regarding tasks, algorithms, and performances (RICE, 1976; KANDA *et al.*, 2011; DAVIS; LOGEMANN; LOVELAND, 1962). Beyond the recommendation of algorithms with fixed hyper-parameters, it is also possible to recommend the hyper-parameters of learning algorithms.

Typically, recommending hyper-parameters entails the initial learning of an optimal standard set of values followed by a subsequent assessment of each hyper-parameter importance (PROBST; BISCHL; BOULESTEIX, 2018). Such importance is often based on the performance gain when adjusting a specific hyper-parameter instead of leaving it with the default value. Usually, default values for the algorithms' hyper-parameters are learned for each task before being assessed. Therefore, the resulting configurations are sampled and the setting that minimizes the average risk across all tasks is recommended as default (MANTOVANI *et al.*, 2015). Another possibility is to independently learn hyper-parameters' default values and the necessity of tuning them (WEERTS; MEULLER; VANSCHOREN, 2018). Through a ranking approach, default values can be established according to the frequency of appearance at the top-K positions for each task. Subsequently, statistical tests can determine whether to tune or not to tune the default value of a hyper-parameter based on the decrease in performance observed when all but the hyper-parameter under analysis is adjusted (MANTOVANI *et al.*, 2015).

2.4.3 Learning from Prior Models

The third and final source of meta-knowledge explored in this thesis is the Learning from Prior Models or, more specifically, leveraging pre-trained models to enhance the learning process (HUTTER; KOTTHOFF; VANSCHOREN, 2019). Unlike previous approaches relying on pre-defined meta-data, this Meta-Learning approach primarily focuses on unsupervised learning of meta-features from existing knowledge. Transfer Learning is a prominent technique example within this category, which has garnered considerable attention across diverse domains (WONG *et al.*, 2018). Transfer Learning often involves the usage of pre-trained models on a source task as a warm-start for training on novel and related tasks, or it is used as a feature extractor. Previous works have explored Transfer Learning and Meta-Learning in Bayesian models (BAKKER; HESKES, 2003), Kernel-based methods (EVGENIOU; MICCHELLI; PONTIL, 2005), and even Clustering algorithms (THRUN; PRATT, 1998), showcasing its versatility.

Few-Shot Learning (RAVI; LAROCHELLE, 2016) is another paradigm that harnesses prior models' knowledge to boost learning efficiency. When prior knowledge is available, Few-Shot Learning can leverage such information to facilitate rapid adaptation to unseen similar tasks. For example, it can tackle the Classification K-shot N-way problem, where the objective is to classify N new classes with only K instances of each. Other examples are the few-shot models that learn how to rapidly adapt to novel tasks with minimal labeled examples, ultimately mitigating the data scarcity challenge prevalent in Machine Learning (FINN; LEVINE, 2017). Techniques such as Siamese Neural Networks (KOCH *et al.*, 2015), Matching Networks (VINYALS *et al.*, 2016), and Relation Networks (SUNG *et al.*, 2018) have demonstrated impressive performance in Few-Shot Learning tasks. By learning effective representations from prior models, these models facilitate generalization to new classes or tasks with only a few instances per class.

Some meta-learning models, like Model-Agnostic Meta-Learning (MAML) (FINN; ABBEEL; LEVINE, 2017) and Reptile (NICHOL; ACHIAM; SCHULMAN, 2018), aim to learn how to learn an entirely base model from prior models' knowledge. MAML centers on learning effective weight initializations for neural networks. It recommends weight initializations that generalize well to similar tasks while allowing quick adaptation to new similar tasks with minimal adjustments. This model iteratively refines these weight initializations by training on a subset of tasks, enhancing the adaptability of base-model weights to novel tasks. Furthermore, MAML recommends initializations that exhibit robustness against overfitting in small datasets and possess broad generalization capabilities (FINN; ABBEEL; LEVINE, 2017). On the other hand, Reptile adopts a different approach by repeatedly fine-tuning a base model across a sequence of tasks. During each iteration, the base model's parameters are updated to minimize the loss on a novel task, fostering increased adaptability. Over multiple iterations, the base model accumulates knowledge from these task-specific adaptations, enhancing its ability to generalize to novel tasks efficiently (NICHOL; ACHIAM; SCHULMAN, 2018).

2.5 Transfer Learning

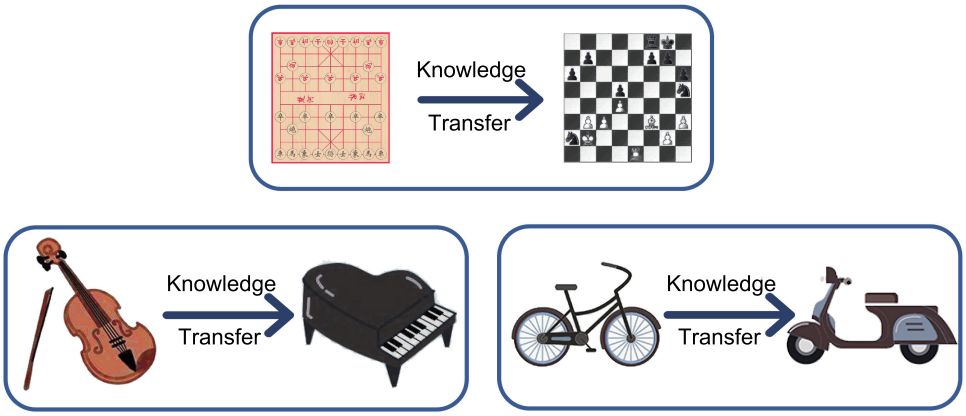


Figure 17 – Transfer Learning practical example. Adapted from Zhuang *et al.* (2020).

Transfer Learning (TL) is a powerful and related technique to MtL that has gained substantial popularity over the years, especially in the CV domain (PAN; YANG, 2009). This learning technique aims to leverage knowledge acquired from one or more source tasks to enhance or accelerate the learning on a related target task (KUMAGAI, 2016). TL gained significant notoriety in Image Classification, where knowledge obtained from large databases such as ImageNet is effectively transferred to recognize visual patterns in smaller datasets like CIFAR (ZHUANG *et al.*, 2020). In scenarios where data is scarce or the learning models are resource-intensive, TL proves to be especially advantageous since it can both speed up and cut off the costs of the learning process (DU *et al.*, 2017). Figure 17 provides a visual representation of the Transfer Learning framework, illustrating knowledge transfer from source tasks to closely related target tasks and highlighting its adaptability across diverse domains and contexts.

Given a source domain D_s, a target domain D_t, learning tasks T_s and T_t, TL can be formally defined in terms of the learning enhancing on T_t with knowledge obtained from a different but similar target domain D_s and learning task T_s, where $D_s \neq D_t$ and $T_s \neq T_t$ (HUTTER; KOTTHOFF; VANSCHOREN, 2019). This definition allows for versatility in TL's application across various domains and contexts, providing a framework for efficient knowledge transferring and adaption to new tasks (WONG *et al.*, 2018). However, it is important to take notice that TL tends to be most effective when the source and target tasks exhibit a close similarity (KUMAGAI, 2016). Thus, this approach may not yield the desired results when transferring between entirely dissimilar concepts, such as using the learning patterns of animal identification to vehicle recognition, for example (DU *et al.*, 2017). Nevertheless, several works in the field explore the transferring of fundamental knowledge even when considering quite different target datasets. Therefore, such methods focus on simple and more general pattern representations like lines, contours, and basic geometric shapes in the context of images (SCOTT; RIDGEWAY; MOZER,

2018). The practice of transferring simple patterns was shown to help the learning of following layers in a network or sequential models, thus often speeding up recognition by providing initial insights and avoiding the need to start the learning process from scratch (PERRONE *et al.*, 2018).

It is also worth highlighting that the application of TL is broad and flexible, with different ways of doing TL and not limited to applications with Neural Networks. Previous works have been using traditional ML models like Support Vector Machines, Bayesian Networks, and Ensembles to transfer lower or higher dimensional representations (ZHUANG *et al.*, 2020). Also, TL is not limited to only serving as a warm-start for model training. In some cases, TL is harnessed to build fixed feature extractors from pre-trained models, preserving the initial learning representations while fine-tuning only the final layers responsible for task-specific operations (PERRONE *et al.*, 2018). Others adopt a mixed approach, preserving the initial layers' knowledge while utilizing TL to initialize subsequent layers during training, offering flexibility and adaptability in ML applications (YANI *et al.*, 2019). Some works even use TL with different data formats and patterns instead of transferring across different concepts, like transferring knowledge between CT and MRI data (ANTONELLI *et al.*, 2022; AZIZI *et al.*, 2017). This cross-modal property of TL further highlights its versatility and ability to accommodate different data modalities, highlighting its role as a powerful tool in the ML scenario and showing its potential to improve performance across multiple domains, thus contributing to the advancement of fields like NAS and MtL (SCOTT; RIDGEWAY; MOZER, 2018; PAN; YANG, 2009).

2.6 Active Learning

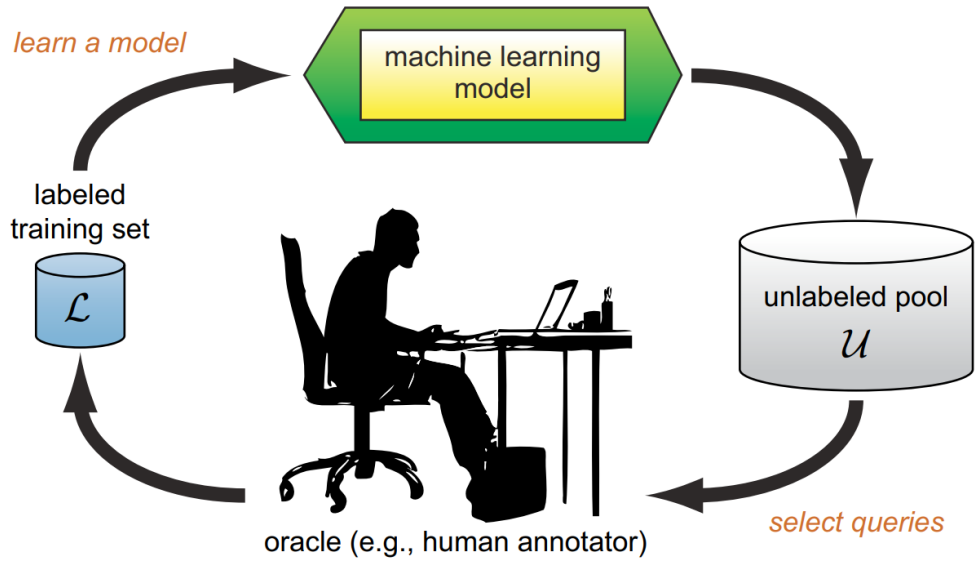


Figure 18 – Typical Active Learning loop. Adapted from Settles (2009).

Active Learning (AL) offers a pragmatic solution for tasks marked by the abundance of unlabeled data and prohibitive cost associated with manual labeling (COHN; ATLAS; LADNER, 1994). It consists of an iterative paradigm of Supervised Learning where algorithms learn in an active way by querying samples and consulting expert knowledge to label data from unlabeled datasets (SETTLES, 2009). Figure 18 illustrates a typical Active Learning loop. Starting with an ML model that fits samples from an unlabeled pool dataset, uncertainty scores are calculated based on model outputs and ranked. Thus, the examples that exhibit the highest levels of uncertainty are presented to an expert entity commonly referred to as the "Oracle". The Oracle's task is to label these uncertain examples with the expectation that they will provide the most valuable insights for enhancing the model's capacity to fit the input data (COHN; GHAHRAMANI; JORDAN, 1996). Subsequently, these queried and labeled samples are incorporated into a labeled training dataset that is used to train the initial learning model. This cyclical process thus continues until the unlabeled pool is empty or another predetermined criterion is met.

A notable advantage of AL is that it allows the learner to strategically choose which examples to query for labeling (LEITE; BRAZDIL, 2010). Consequently, this significantly reduces the number of examples required to fit particular data points when compared to traditional Supervised Learning methods, as learning is directly related to the model learned parameters. Therefore, the capacity to actively select examples to label enables AL algorithms to accelerate model convergence while minimizing the amount of labeled data needed. Even though this is a secondary objective, even an increase in predictive performance can be expected in some cases. To achieve this, AL considers a range of scenarios to select examples for labeling, including Membership

Query Synthesis, Pool-Based Sampling, and Stream-Based Selective Sampling (DASGUPTA, 2011). These scenarios are designed to maximize learning performance considering different ML task requirements. However, all of them share the goal of identifying informative examples while minimizing the dependence on labeled data (COHN; GHAHRAMANI; JORDAN, 1996). Therefore, the selection of examples directly depends on multiple factors, including the AL scenario, the sampling strategy, the query function, and the representation learned by the model.

Of the various AL scenarios, Pool-based Sampling is perhaps the most popular given its simple framework and flexibility (SETTLES, 2009). As seen in Figure 18, it consists of submitting samples from an unlabeled data pool to the learner and, based on its knowledge, choosing the examples for labeling. This example selection usually occurs through a query function that will assign confidence scores. Thus, the examples with the lowest confidence scores are chosen for labeling, and by focusing on examples that the learner is least confident about, one can expect to maximize the potential for information gain which will enhance the learning process (DASGUPTA, 2011). Therefore, through a series of queries to an Oracle, the learner gradually reduces uncertainty regarding the dataset (LEITE; BRAZDIL, 2010).

The Pool-based Sampling scenario of AL can be mathematically defined as follows: Let X represent the input space, and Y denote the output space. Given a training set $D = (x_i, y_i)i = 1^n$, where the pair $x_i \in X$ and $y_i \in Y$ are input features and labels for each example, and an unlabeled pool set $U = x_j j = 1^m$, where $m > n$, the goal is to iteratively select a subset $S \subseteq U$ of size k, with $k \ll m$, to query for their labels and integrate them into the training set D. At each iteration t, a query function $q_t : U \rightarrow S_t$ is utilized to select the most informative instances from U based on the current model's knowledge. The labeled examples (x, y) obtained through queries are then appended to D, and the learning algorithm can use D to update its model parameters θ_t. In addition to choosing the AL scenario, it is necessary to define the query function that will determine which data points should be submitted to the Oracle to be labeled. In turn, this query function depends on the specific AL scenario employed and the nature of the sampling, such as Uncertainty Sampling, Committee query, or Expected model change (SETTLES, 2009).

2.7 Chapter Remarks

This chapter presented a comprehensive theoretical foundation necessary to promote a deeper understanding of subsequent research contributions and the discussions articulated throughout this thesis. Section 2.1 focused on establishing some of the fundamental concepts of Machine Learning, such as the learning paradigms, ML tasks, learning algorithms, and various evaluation metrics used in the proposals. Additionally, the problems addressed by this thesis' proposals are also discussed thoroughly, those being the Algorithm Recommendation problem and the Computer Vision problems of Image Classification, Image Segmentation, and Medical Imaging Segmentation. Building on this solid foundation, Section 2.2 focused on Neural Networks, offering a succinct historical record of some crucial milestones and advances that have shaped the field. In addition to tracing the trajectory of NNs, highlights were given to key models such as Feed-Forward Neural Networks, Convolutional Neural Networks, and U-Nets, which all serve as building blocks for several experiments presented later in this document. Section 2.3 introduced the growing field of Neural Architecture Search, one of the main focuses of this thesis. Besides introducing the premises, concepts, motivations, and challenges of the field, the three basic components that comprise most NAS methods, Search Space, Search Strategy, and Performance Estimation, along with several related baselines, were presented in detail.

Coming from the context of NAS and its challenges, Section 2.4 addressed one of the possible solutions to such problems in the concept of Meta-Learning. In addition to presenting details of the Meta-Learning framework, this section elucidated its unique place in the ML landscape by differentiating it from conventional ML and presenting three popular and diverse MtL approaches, each leveraging prior knowledge to streamline the learning process. The introduction of these approaches, called Learning from Tasks, Learning from Model Evaluations, and Learning from Prior Models, is essential for understanding the subsequent research chapters shaping the basis for innovative solutions at the intersection between MtL and NAS. In the following, Section 2.5 and Section 2.6 presented the concepts of Transfer Learning and Active Learning, respectively. These topics, closely related to MtL, enrich the understanding of the broader ML ecosystem and prepare for the proposals in subsequent chapters that use such concepts. While in Section 2.5, the concept, advantages, disadvantages, and applications of Transfer learning, such as its use as a warm-start and as a fixed feature extractor for similar tasks is presented, Section 2.6 introduces some of the Active Learning scenarios, in particular the Pool-based Sampling, and the Query Strategies such as the Uncertainty sampling-based. In the next chapters, the main research contributions of this thesis are presented in detail.

CHAPTER

3

TRANSFER LEARNING AT THE META-LEARNING LEVEL

This chapter shows the initial experiment of this thesis. Before diving into the application of Meta-Learning (MtL) to search for neural architectures, a proper exploratory study of its applicability was necessary. Consequently, this first experiment aims to assess meta-knowledge effectiveness through different meta-datasets using Transfer Learning (TL), a closely related concept to MtL. Thus, meta-datasets related to the popular Algorithm Recommendation (AR) problem were adopted for the usage of various TL configurations. By leveraging prior knowledge from multiple ML algorithms trained on various meta-datasets, the goal is to determine the appropriate algorithm for unseen meta-datasets.

In summary, this chapter presents the following contributions:

- A proof-of-concept TL experiment demonstrating that meta-knowledge encoded as meta-features associated with the task of Algorithm Recommendation exhibits generalization capabilities across various datasets and models;
- An analysis comparing different configurations within a TL setup establishing that freezing all layers often produces the most favorable outcomes;
- A comprehensive performance comparison analysis reveals that TL incorporating meta-data not only demonstrates generalization across multiple datasets but also enhances accuracy and minimizes loss.

The rest of the chapter is organized as follows: Section 3.1 describes the research contribution. Section 3.2 details the experimental setup for reproducibility purposes. Section 3.3 discusses the main results. The final considerations for the chapter are presented in Section 3.4.

3.1 Transfer Learning for Algorithm Recommendation

This section introduces an experiment with different approaches of TL to solve the MtL problem of AR. For this purpose, meta-datasets related to classical optimization problems are employed in the training of a shallow Neural Network used as a meta-classifier along with three TL setup variations. Furthermore, this experiment also includes standalone training and knowledge transfer between all meta-datasets. In the following, an illustrative overview of the TL procedure using MtL datasets is shown in Figure 19, followed by a comprehensive elucidation of the TL framework procedure in Algorithm 1. Moreover, essential details regarding the meta-datasets and the FFNN meta-classifier are presented in Table 2 and Table 3, respectively, along with a thorough explanation of the underlying concepts.

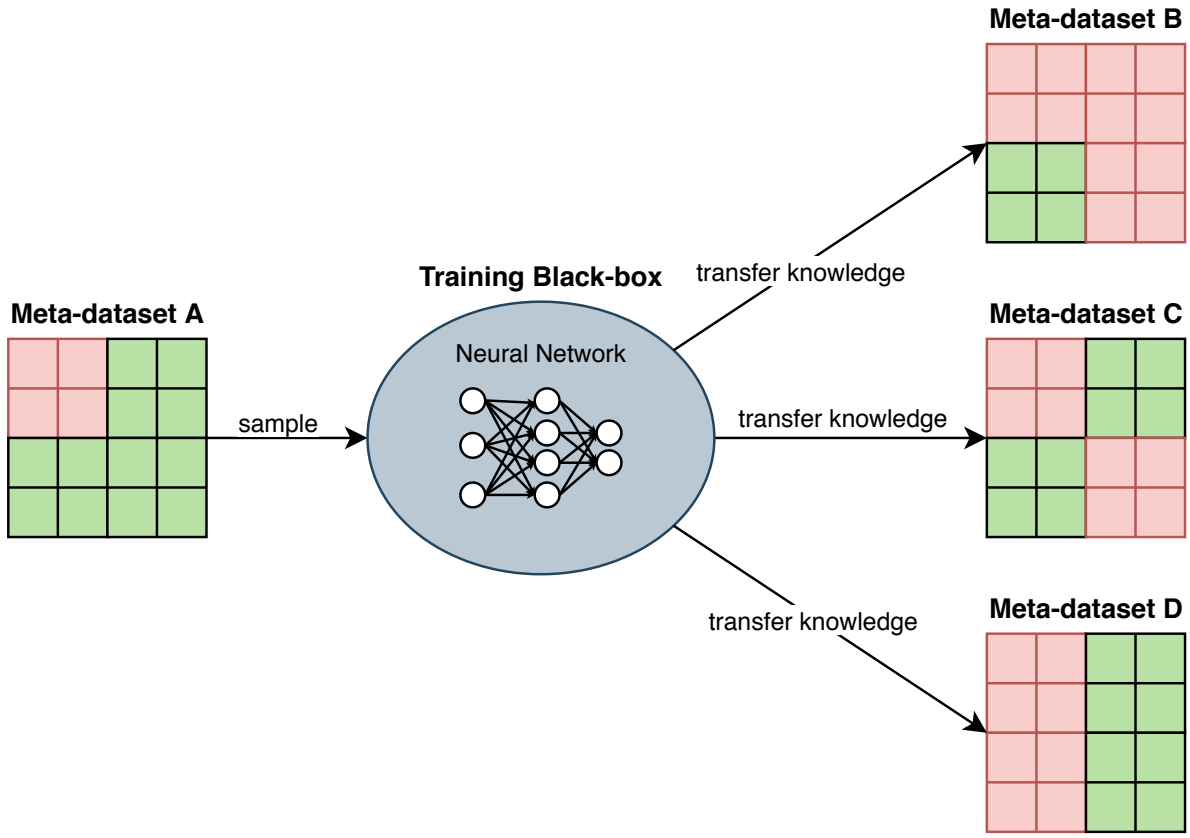


Figure 19 – Illustration of the Transfer Learning process considering similar Meta-datasets.

As seen in Figure 19, the TL at the MtL level follows the same workflow as in the traditional or Base-Learning level. The process starts with a Meta-dataset A that serves as the central source dataset used to train a meta-classifier in the form of a Neural Network. This model thus learns patterns and gains knowledge from Meta-dataset A that will be transferred to enhance the learning on other meta-datasets, such as Meta-datasets B, C, and D. Hence, such meta-datasets symbolize distinct target datasets that will benefit from knowledge transfer, sharing some common patterns with Meta-dataset A. In fact, this is a requirement for the successful

application of a TL approach since the more similar a dataset or meta-dataset is, the better will be the observed impacts on learning. Therefore, by fine-tuning its prior knowledge based on the target meta-datasets' unique characteristics while retaining common patterns from Meta-dataset A, the meta-classifier can speed up the learning or improve performance on the target data. In the following, the specific steps of the TL experiment with meta-data are shown in Algorithm 1.

The specific steps of the TL experiment shown in Algorithm 1 begin by defining the three primary inputs S as the source meta-dataset, T as the target meta-dataset, and $Meta$ as the FFNN, where the goal is to recommend a set of algorithms A for T as an output. The meta-classifier $Meta$ is initially trained on the source meta-dataset S, enabling the acquisition of meta-knowledge from optimization algorithms and their performances, in addition to statistics of the optimization problem to which these algorithms were applied. Subsequently, the TL setup loop starts and three TL configurations are assessed: (i) freezing the first hidden layer, thus preserving prior knowledge on low-level features while adjusting deeper layers; (ii) freezing the two hidden layers, thus using the learned weights as a feature extractor for the target task at hand; and (iii) freezing no hidden layers, thus using the learned weights as a warm-start for the end-to-end fine-tuning of $Meta$ on T. Once the layer freezing procedure is terminated, $Meta$ undergoes end-to-end training on the target meta-dataset T, updating its weights and biases with the target data's characteristics. Finally, having trained $Meta$ on the target meta-dataset, the model is employed to predict a set of algorithms A based on its acquired knowledge from the source meta-dataset S and fine-tuning performed on the target meta-dataset T.

Algorithm 1: Algorithm Recommendation with Transfer Learning.

Input : Source meta-dataset S, Target meta-dataset T, Neural Network $Meta$
Output : Selection of Algorithms A

```
1  A ← ∅
2  train(Meta, S)
3  foreach Transfer Learning setup do
4      if First layer frozen then
5          freeze(Meta, 1HL)
6      else if Both layers frozen then
7          freeze(Meta, 2HL)
8      else
9          Use Meta as warm-start
10     end
11     train(Meta, T)
12     A ← predict(Meta, T)
13 end
```

A successful TL application is dependent on the degree of shared knowledge between source and target tasks, which can even dictate the setup definition. The practice of freezing neural layers during TL, in particular, is a common procedure when layers contain general or task-agnostic representations, allowing an efficient adaption to target tasks and making the choice of which layers to freeze extremely important (GOODFELLOW; BENGIO; COURVILLE, 2016). Freezing the first layers might be suitable when low-level features from the source are likely to be transferable to a target dataset. On the other hand, freezing the deeper layer allows a model to adapt at a slightly higher level since such layers tend to learn more complex patterns as they concatenate knowledge learned by initial layers. The decision to not freeze any layers, however, provides a complete fine-tuning opportunity by enabling a model to adapt to more dataset-specific features. In this setup, a prior model used as a warm start is justified when there is limited prior knowledge of the source and target dataset's relationship.

3.2 Experimental Setup

This section presents the complete experimental setup involved in the TL procedures and training of meta-classifiers. To guarantee the transparency of experiments and facilitate their reproducibility, Section 3.2.1 introduces the employed meta-datasets, while Section 3.2.2 presents the evaluation methodology that includes data pre-processing, tuning procedure, implementation details, software, and hardware specifications used to train and validate the models.

3.2.1 Meta-Datasets

This section presents four meta-datasets adopted for training the meta-learners and assessing the various TL configurations employed. Table 2 highlights the meta-datasets along with their statistics. Such meta-data were collected from the Aslib benchmark (BISCHL *et al.*, 2016), a repository associated with optimization problems such as the Traveling Salesman Problem (TSP), Quantified Boolean Formula (QBF), and Propositional Satisfiability Problem (SAT). All employed meta-datasets are related to the AR task, where the same optimization problem but with different priors is addressed. Furthermore, these meta-data were selected not only due to their wide usage but for their various characteristics and complexities, making them a simple but powerful tool for validating TL configurations at the MtL level.

Table 2 – Meta-datasets' statistics.

Meta-Dataset	Examples	Meta-Features	Classes
CSP-2010	2024	67	2
CSP-Minizinc-Obj-2016	100	123	3
CSP-Minizinc-Time-2016	100	117	4
CSP-MZN-2013	4636	117	10

The meta-datasets presented in Table 2 are closely tied with the Constraint Satisfaction Problem (CSP), a class of problems renowned for its NP-completeness (LECOUTRE, 2013). CSPs revolve around finding solutions that adhere to predefined rules and constraints, playing a crucial role in tasks such as resource allocation, scheduling, and decision-making. The problem consists of three key components: variables, domains, and constraints. To tackle CSPs, algorithms like backtracking and forward checking are employed to systematically explore solutions by assigning variable values and backtracking when necessary, ensuring rule adherence. Additionally, constraint propagation algorithms efficiently reduce the search space to find valid CSP solutions. Therefore, CSP was chosen as a use case for its relevance and being particularly difficult among other Aslib problems (MISIR; SEBAG, 2017).

Furthermore, several factors influenced the selection of these meta-datasets in particular. CSP-2010, with 2024 examples, 67 meta-features, and 2 classes, offers an ideal starting point for exploratory analysis given its moderate size and simplified binary classification. In contrast, CSP-Minizinc-Obj-2016, featuring 100 examples, 123 meta-features, and 3 classes, elevates complexity with a multi-class classification challenge and higher feature dimensionality. Similarly, CSP-Minizinc-Time-2016 further amplifies the multi-class classification comprising 100 examples, 4 classes, and 117 meta-features, thus testing the model's capacity to handle higher-dimensional data. In particular, CSP-Minizinc-Obj-2016 and CSP-Minizinc-Time-2016 present a severe issue regarding their dimensionality with an *example : feature* ratio of 0.81 and 0.85, respectively. Given the curse of dimensionality (BELLMAN, 1966), this can be really troublesome for ML models in general. Lastly, CSP-MZN-2013, the largest meta-dataset with 4636 examples, 117 meta-features, and 10 classes, serves as a robust benchmark for assessing model performance in a more realistic application. This progressive complexity ensures a comprehensive evaluation of the meta-learner's adaptability to a variety of meta-dataset characteristics, enhancing external validity and applicability to real-world scenarios.

3.2.2 Evaluation Methodology

This section briefly presents the evaluation methodology adopted to conduct experiments with different TL setups. Given that the employed meta-classifiers consist of FFNNs, and Neural Networks frequently benefit from normalized features (HAYKIN., 2010), all numerical meta-features from all four meta-datasets were normalized to the [0, 1] interval, and meta-targets were transformed into discrete numerical values. In addition, as TL requires certain standardization and not all meta-datasets present the same amount of predictive features, a Feature Selection procedure was employed. The Select *K* Best method (PEDREGOSA *et al.*, 2011) was used to filter the *K* meta-features with the highest score for a specific metric. The metric used was the F-ANOVA value (FISHER; MACKENZIE, 1923), and *K* was the number of features from the meta-dataset with fewer meta-features, in this case, CSP-2010 composed of 67 meta-features. Another problem observed in those meta-datasets was the class imbalance. To handle this, a Stratified Hold-out validation method was used (PEDREGOSA *et al.*, 2011), where 80% of the original data was partitioned for training and 20% for testing.

As highlighted in Algorithm 1, three configurations of TL were adopted in this experiment: (i) Freeze [1] two hidden layers, thus setting them to be not trainable and just using their weights as a feature extractor; (ii) Freeze only the first hidden layer, thus setting only the second hidden layer to be trained and re-using the hyper-planes of the first layer; and (iii) Freeze no hidden layer, thus using the learned weights to warm-start the training on the target meta-datasets. Therefore, each meta-dataset was used as both a source meta-dataset and as a target meta-dataset at a given moment, and each of these TL approaches was used and compared.

As for the meta-learner, an FFNN classifier composed of two hidden layers was applied for recommending algorithms using the CSP meta-datasets. Table 3 introduces the related hyperparameter configurations for this model optimized through Grid search during the TL process. More specifically, the tuning of the meta-classifiers was done using 20% of the training split. The choice for using a simplified and shallow FFNN architecture was defined in order to minimize the costs for this initial exploratory experiment. Thus, a minimum budget necessary to answer the research questions and thus advance toward the general objective of the thesis was defined. Despite these deliberately streamlined configurations, extensive and rigorous experimentation was conducted. This entailed training the meta-classifiers across 30 iterations for each TL setup, where averages and standard deviations were calculated for accuracy and loss, ensuring comprehensive and robust validation results.

[1] Layers' weights will not change; they will just be used

Table 3 – Hyperparameter tuning space.

Hyperparameter	Range
Neurons 1st HLayer	(Input features, Input features*2, 100)
Neurons 2nd HLayer	(Input features, Input features*2, 100)
Dropout Input Layer	(0, 0.5, 0.9)
Dropout 1st HLayer	(0, 0.5, 0.9)
Dropout 2nd HLayer	(0, 0.5, 0.9)
Activation	(ReLU, Leaky ReLU, Sigmoid)
Initialization	(Uniform, Normal, He et al, Xavier)
Optimizer	(SGD, Adam, RMSProp)
Batch size	(16, 32, 64)
Epochs	(25, 50, 100)

Table 3 hyperparameters encompass essential factors for the trainability of the FFNN meta-learners. Each hyperparameter choice has been meticulously made to ensure model adaptability and robustness across diverse scenarios. Notably, the number of neurons in the first and second hidden layers have a well-considered range, striking a balance between model capacity and overfitting avoidance. Dropout was adopted to collaborate with the prune to overfitting, where rates for different layers were systematically varied. A diverse set of activation functions, including the popular ReLU, Leaky ReLU, and Sigmoid, helped to examine the balance between linearity and non-linearity. Multiple weight initialization methods were also considered, including the Uniform, Normal, He et al., and Xavier, aiming to explore the model's sensitivity at initialization and generalization. Various optimization algorithms, including the standard SGD and other popular choices such as Adam and RMSProp, were included in the Grid search to find the most suitable for computational efficiency and convergence. Lastly, Batch size and Training epoch ranges were defined based on a balance between memory efficiency and gradient accuracy, whereby their impact on the learning curve and final model performance were closely examined.

When it comes to Software, a few details should be mentioned. The four meta-datasets related to CSP were collected from the Aslib repository (BISCHL *et al.*, 2016), and all preprocessing procedures, including Feature Selection, were done by using the popular Scikit-learn (PEDREGOSA *et al.*, 2011) and Pandas (TEAM, 2020) libraries. For implementing and training the FFNNs, the high-level Keras library (CHOLLET *et al.*, 2015) was used with a TensorFlow backend (ABADI *et al.*, 2015). In addition, it is worth mentioning that the entire codebase was constructed using Python language and relied only on open-source libraries. The complete experimental setup encompassing method implementation and associated documentation has been made openly available to the public through a GitHub repository [2]. In terms of Hardware, all training was conducted on a single Linux server with an Intel Xeon CPU E5-2620 v2 2.10GHz, 128GB of RAM, and no GPU, showcasing a lite and accessible approach.

[2] https://github.com/geantrindade/TL_MtL

3.3 Results and Discussion

The experiments presented in this section were designed to answer the following research questions: (i) How does applying TL impact MtL performance across meta-datasets? (ii) Which TL configuration yields the best MtL performance gain, and what are the differences in effectiveness? (iii) To what extent can TL generalize across meta-datasets and provide insights into compatibility for different problem characteristics? The following sections address these questions through analysis and comparisons of the three TL configurations applied to an FFNN meta-classifier with two hidden layers trained in four meta-datasets.

Accuracy (Acc) and Loss are displayed for every meta-dataset in Tables 4 to 7. The pre-trained models that were later submitted to TL were induced using the meta-datasets listed in the headers of these tables. Table 4, for example, displays results obtained with the CSP-2010 meta-dataset. The final test accuracy and loss obtained via end-to-end training on CSP-2010 without TL are indicated in the top row's column labeled "Normal". However, results using pre-trained models on each of those source meta-datasets are displayed in columns CSP-MZN-2013, CSP-Minizinc-Obj, and CSP-Minizinc-Time. Such pre-trained models were then refined on the target meta-dataset CSP-2010 using the three TL setups, designated 0HL, 1HL, and 2HL.

Table 4 – CSP-2010 results. Column *Normal* stands for no Transfer Learning, and *2HL*, *1HL*, and *0HL* stands for Two, One, and Zero hidden layers frozen, respectively (PEREIRA *et al.*, 2019).

	Normal	**CSP-MZN-2013**			**CSP-Minizinc-Obj**			**CSP-Minizinc-Time**		
	-	0HL	1HL	2HL	0HL	1HL	2HL	0HL	1HL	2HL
Acc	0.87 ± 0.01	$\mathbf{0.88 \pm 0.01}$	0.87 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.86 ± 0.01	0.87 ± 0.01	0.87 ± 0.01	0.86 ± 0.01	0.85 ± 0.01
Loss	0.64 ± 0.15	$\mathbf{0.56 \pm 0.12}$	0.84 ± 0.08	1.01 ± 0.06	0.65 ± 0.15	0.90 ± 0.04	1.01 ± 0.09	0.65 ± 0.17	1.00 ± 0.06	1.07 ± 0.09

As mentioned, Table 4 shows the outcomes of three TL setups applied to CSP-2010 meta-dataset: freeze two, one, and zero hidden layers. After pre-training on CSP-MZN-2013, 0HL setup slightly surpassed the baseline with an 0.88 accuracy, whilst the Normal setup with no TL attained an accuracy of 0.87. The accuracies of other TL combinations, which ranged from 0.85 to 0.87, were comparable or somewhat worse. The Normal setup yielded a loss of 0.64, whereas the best-performing TL variation, the 0HL setup from CSP-MZN-2013, obtained a reduced loss of 0.56. With the exception of 0HL from CSP-Minzinc-Obj which had a slightly higher loss of 0.65, other TL settings had comparable loss values. Notably, some setups exhibited considerably higher losses, reaching up to 1.07 as seen in the 2HL configuration from CSP-Minizinc-Time. These results demonstrate that TL can improve accuracy and decrease loss in comparison to the Normal baseline, especially when zero hidden layers are frozen after pre-training on the CSP-MZN-2013 meta-dataset. Nevertheless, freezing two hidden layers or only the first hidden layer did not consistently lead to improvements in accuracy or loss. These results thus emphasize the impact of pre-trained models and TL setups on the learning of meta-classifiers.

Table 5 – CSP-Minizinc-Obj results. Column *Normal* stands for no Transfer Learning, and *2HL*, *1HL*, and *0HL* stands for Two, One, and Zero hidden layers frozen, respectively (PEREIRA *et al.*, 2019).

	Normal	**CSP-2010**			**CSP-MZN-2013**			**CSP-Minizinc-Time**		
	-	0HL	1HL	2HL	0HL	1HL	2HL	0HL	1HL	2HL
Acc	**0.91 ± 0.02**	0.87 ± 0.04	0.90 ± 0.00	0.90 ± 0.00	0.66 ± 0.02	0.70 ± 0.00	0.70 ± 0.00	0.90 ± 0.00	0.90 ± 0.00	0.90 ± 0.00
Loss	**0.77 ± 0.06**	1.01 ± 0.16	1.37 ± 0.11	1.55 ± 0.06	3.26 ± 0.06	3.30 ± 0.01	3.27 ± 0.01	0.79 ± 0.11	1.01 ± 0.10	1.37 ± 0.15

Table 5 presents the results of TL setups on the CSP-Minizinc-Obj meta-dataset. While the Normal baseline yielded an 0.91 accuracy, the best TL runs from both CSP-2010 using 1HL and 2HL configurations, and all CSP-Minizinc-Time TL setups obtained an accuracy of 90%. Interestingly, these results from applying TL to CSP-Minizinc-Obj are the only instance in this entire experiment where TL did not lead to improved performances. Notably, the worst performances were obtained when utilizing CSP-MZN-2013 as the source dataset, where accuracy results spanned from 0.66 to 0.70 across various TL configurations. Concerning the loss metric, the Normal baseline registered a value of 0.77, while the best configurations of TL using 0HL and 1HL with CSP-Minizinc-Time achieved lower loss values of 0.79 and 1.01. Remarkably, when the source meta-dataset was CSP-MZN-2013, loss values ranged from 3.26 to 3.30 across different TL setups, the worst performing results among all. In summary, these findings indicate that while TL did not significantly enhance the learning for the target task CSP-Minizinc-Obj, the usage of CSP-Minizinc-Time as a source with zero hidden layers frozen has produced the best results among TL variations, closely approaching the performance of the Normal baseline. These observations align with the results analyzed in Table 4 concerning the CSP-2010 meta-dataset.

Table 6 – CSP-Minizinc-Time results. Column *Normal* stands for no Transfer Learning, and *2HL*, *1HL*, and *0HL* for Two, One, and Zero hidden layers frozen, respectively (PEREIRA *et al.*, 2019).

	Normal	**CSP-2010**			**CSP-MZN-2013**			**CSP-Minizinc-Obj**		
	-	0HL	1HL	2HL	0HL	1HL	2HL	0HL	1HL	2HL
Acc	0.65 ± 0.00	0.65 ± 0.01	0.65 ± 0.00	0.65 ± 0.00	0.67 ± 0.03	0.70 ± 0.00	**0.73 ± 0.02**	0.70 ± 0.00	0.70 ± 0.00	0.70 ± 0.00
Loss	4.11 ± 0.60	3.78 ± 0.92	5.24 ± 0.18	5.59 ± 0.04	**3.27 ± 0.30**	3.81 ± 0.10	4.04 ± 0.03	3.87 ± 0.74	4.54 ± 0.07	4.60 ± 0.02

Table 6 outlines results from TL setups submitted to the CSP-Minizinc-Time meta-dataset. The Normal configuration recorded an accuracy of 0.65, in pair with TL setups of CSP-2010 pre-trained models. Remarkably, these CSP-2010 results were the worst among TL runs. When TL was applied with the 2HL configuration of CSP-MZN-2013 pre-trained model, it resulted in the highest accuracy of 0.73, surpassing the Normal setup by the largest margin in this entire experiment. In terms of loss, the Normal configuration recorded a high value of 4.11, being comparable with the rest of the TL models. Conversely, the best result from TL with CSP-MZN-2013 pre-training using the 0HL setup led to a substantial reduction in loss of 3.27, indicating the potential for enhancing model performance. The configuration that generated the best accuracy, however, 2HL from CSP-MZN-2013, resulted in a higher loss of 4.04. Interestingly, even though both metrics are inverted correlated, this is the first time in the whole experiment that the model with the highest accuracy did not generate the lowest loss. At the same time, TL generated the largest margin from the normal baseline, obtaining an 8% improvement in performance. In alignment with past observations, these results highlight CSP-MZN-2013 as the most suitable source of meta-data for the current AR problem, yielding higher performance compared to the Normal baseline.

Table 7 – CSP-MZN-2013 results. Column *Normal* stands for no Transfer Learning, and *2HL*, *1HL*, and *0HL* stands for Two, One, and Zero hidden layers frozen, respectively (PEREIRA *et al.*, 2019).

	Normal	**CSP-2010**			**CSP-Minizinc-Obj**			**CSP-Minizinc-Time**		
	-	0HL	1HL	2HL	0HL	1HL	2HL	0HL	1HL	2HL
Acc	0.71 ± 0.01	0.71 ± 0.01	0.70 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	0.71 ± 0.01	$\mathbf{0.72 \pm 0.01}$	0.71 ± 0.01	$\mathbf{0.72 \pm 0.01}$	0.71 ± 0.01
Loss	1.23 ± 0.19	1.27 ± 0.22	1.71 ± 0.07	1.95 ± 0.07	1.25 ± 0.21	1.71 ± 0.08	1.98 ± 0.08	$\mathbf{1.18 \pm 0.18}$	1.63 ± 0.11	1.92 ± 0.06

Table 7 provides results for the three employed TL setups on CSP-MZN-2013 meta-dataset. The differences in performances for CSP-MZN-2013 were similar to the ones on CSP-2010, where TL setups varied around 1%-2% accuracy. Being more specifically, the pre-training on CSP-Minizinc-Obj using 2HL and on CSP-Minizinc-Time using 1HL yielded the best accuracies of 0.72, overcoming the Normal configuration baseline with a 0.71 accuracy. Regarding loss, the gap differences were larger, with Normal obtaining a 0.64 loss while the best-performing TL variation CSP-Minizinc-Time yielded a loss of 1.18. An interesting fact of these results is the recurrence of a low correlation between accuracy and loss regarding the best models. As seen in Table 6, models with the best accuracies did not obtain the lowest losses, and vice-versa. As already mentioned, the tendency is for these measures to have a strong negative correlation. However, the phenomenon observed here can still occur, and some explanations are viable. While accuracy measures how accurate the meta-model is in relation to the original data, and higher values indicate better performance, the loss measures how far the predictions are from the real signal, thus quantifying the errors made by the model according to a specific criterion. For this experimental setup, the loss used was the cross-entropy loss, which measures the dissimilarity between predicted and actual class probability distributions. Therefore, in cases

where the distribution of one class is denser than another, or if there is an overlapping decision boundary, it is possible to minimize the cross-entropy loss and not necessarily increase the accuracy, as the loss is a smooth function whilst the accuracy is not.

3.4 Chapter Remarks

This chapter presented an exploratory experiment on the Meta-Learning applicability in the context of NAS. Before applying Meta-learning to the search for neural architectures, it was essential to check how representative and generalist meta-features and meta-learners are in recommending learning models. To this end, employed meta-datasets from a specific problem known as Algorithm Recommendation were used, which, instead of neural architectures, deals with the problem of recommending optimization algorithms or ML algorithms in general. Together with a Transfer Learning approach, a technique that, like Meta-Learning, uses prior knowledge to accelerate or enhance learning in unseen tasks, this experiment served as a proof of concept of the applicability and effectiveness of MtL for NAS in several scenarios.

The experiments carried out in this chapter were specific to answering the research questions that guided this study. Three research questions were defined, "How does applying TL impact MtL performance across meta-datasets?", "Which TL configuration yields the best MtL performance gain, and what are the differences in effectiveness?", and "To what extent can TL generalize across meta-datasets and provide insights into compatibility for different problem characteristics?", addressed by the analyses shown in Tables 4, 5, 6 and 7. In these analyses, three variations of the TL setup were applied to four meta-datasets in order to mitigate results variations when evaluating the effectiveness of meta-knowledge and its generalization in the model recommendation task. Such variations include the knowledge transfer from a pre-trained model used as a warm-start on the target dataset (zero hidden layers), the reuse of hyperplanes learned in the first layer only and adjusting the remaining (1HL), and the reuse of hyperplanes of the entire architecture, thus adjusting only the output classification layer (2HL).

The results demonstrated that TL at the meta-level is not only feasible but is significantly valuable, consistently matching or surpassing the performance achieved through the original data training. Among the three variations tested, the optimal TL configuration involved reusing the learned internal representations and adjusting the final classification layer to the task at hand, which led to increased accuracy and reduced losses. It was also found that meta-knowledge from meta-datasets of the same domain or problem tends to generalize more, as expected. However, the effectiveness of TL configurations is context-dependent, influenced by variables such as the choice of pre-trained models, frozen layers, and specific datasets, requiring careful examination and experimentation to avoid misleading interpretations of the results. The following chapter investigates a detailed subsequent exploratory experiment of this thesis, advancing the understanding of Meta-Learning and the properties of metadata.

CHAPTER

4

DIMENSIONALITY REDUCTION OF META-INFORMATION

This chapter presents the follow-up exploratory experiment on the analysis of meta-data but now focusing on its dimensionality. Once it was verified that simple models such as Feed-Forward Neural Networks (FFNNs) could leverage Meta-Learning (MtL) data to recommend algorithms, the next logical step was to study the properties of such meta-data. This experiment was then designed to assess the representativeness and redundancy of meta-features from various sources. The AR problem is again addressed for this purpose, where Dimensionality Reduction (DR) methods are applied to meta-datasets and their impact is validated on three criteria: predictive performance, dimensions reduction, and pipeline runtime.

In summary, this chapter presents the following contributions:

- A proof-of-concept experiment using DR demonstrates the redundancy within meta-datasets and indicates that approximately 80% of the original data can be discarded;
- An investigation of Feature Selection (FS) and Feature Extraction (FE) methods demonstrates that with DR, it is possible to use only about 20% of the data and achieve performance comparable to those using the original dataset while reducing pipeline runtimes;
- Comparison of FS methods to a more intricate FE baseline revealed that FS could achieve slightly superior dimension reduction and predictive performance but statistically significant evidence of better runtime attributed to their simplicity.

The rest of the chapter is organized as follows: Section 4.1 describes the research contribution. Section 4.2 details the experimental setup for reproducibility purposes. Section 4.3 discusses the main results. Lastly, the final considerations are presented in Section 4.4.

4.1 Meta-Feature Selection for the AR problem

This section introduces an experiment with different DR techniques, where the goal is two-fold: to analyze how representative or redundant the meta-datasets can be and to assess the impact on meta-model learning when reducing feature dimensions. Therefore, four DR methods applied to twenty-eight meta-datasets used in the training of four meta-classifiers are employed. Such meta-data comes from combinatorial optimization problems related to the MtL problem of AR. The following sections present each DR method in detail. Section 4.1.1 introduces the first and simplest Variane Threshold method for DR. Section 4.1.2 describes Univariate Feature Selection, the broader category to which Chi-squared and Analysis of Variance (ANOVA) belong. Finally, Section 4.1.3 presents the popular Feature Extraction method, Principal Component Analysis, adopted as the main baseline for the more simple DR techniques.

There has been a long history of research in the field of MtL that explores meta-level problems of different natures and the development of meta-models (FILCHENKOV; PENDRYAK, 2015; PARMEZAN; LEE; WU, 2017; BOMMERT *et al.*, 2020). Nevertheless, there is a noticeable gap in works that focuses on the discriminative potential of meta-datasets and the identification of critical meta-features that could contribute to good performance gains in meta-models (ALCOBAÇA *et al.*, 2018; PEREIRA *et al.*, 2019). One of the possible solutions for these concerns is centered around the extraction of the most representative sets of meta-features within problematic meta-datasets. For this reason, it is applicable to use Feature Selection (FS), a DR technique known for its ability to reduce the number of feature dimensions to a manageable size for effective processing and analysis (LAZAR *et al.*, 2012).

When considering a DR method, it is essential to understand the differences between FS and other DR methods, particularly when concerning the so-called Feature Extraction (FE) perspective, which fundamentally changes the composition of the input data. FE techniques transform the original data to enhance specific characteristics and reduce feature dimensions (GUYON *et al.*, 2008). FS, on the other hand, meticulously selects a sub-group of relevant features while systematically eliminating those features whose presence does not positively affect the learning model (LAZAR *et al.*, 2012). Another difference between FS and FE is that the first is interpretable-oriented, as it can be used to inspect the relevant and irrelevant features, something that, in most cases, cannot be done with FE (MOLNAR, 2020).

The use of DR becomes particularly important when a dataset is overloaded with noisy features, unfavorable example-feature ratios, and redundancy, factors that greatly hinder model learning (GAMA *et al.*, 2011). Therefore, finding an optimal feature subset that faithfully represents the original dataset is not a trivial task. Selecting too many features increases time and space complexity, overloading the learning model and decreasing performance (GUYON; ELISSEEFF, 2003). On the other hand, opting for few features has the risk of losing valuable information, which also leads to model performance reduction (FLACH, 2012). Another important point to

consider is the DR methods' nature. Within DR techniques, in particular, for FS, two broad categories are model-based and filter-based FS, which include Recursive Feature Elimination and Chi-squared, respectively (FLACH, 2012). In this present study, the focus is on filter-based FS methods due to their lower computational overhead. Subsequent sections provide a concise overview of specific techniques within this filter-based category and FE methods.

4.1.1 Variance Threshold

The Variance Threshold is a simple FS method that systematically removes features characterized by a low variance (GUYON; ELISSEEFF, 2003). It achieves this by first computing each individual feature variance and subsequently discarding the features whose variances fall below a defined threshold. Another important quirk of this process is the method's ability to ensure a consistent scale among the selected features, which can boost model learning (BOMMERT *et al.*, 2020). While the method itself is straightforward in its application, its motivation is notably compelling. Its fundamental premise is rooted in the idea that features displaying low variance contain no valuable information or relevance (PEDREGOSA *et al.*, 2011). Furthermore, the Variance Threshold method is inherently unsupervised, as it solely assesses feature values without consideration for the target output. This intrinsic unsupervised quality renders it highly versatile and well-suited for scenarios where external labels or supervisory information is either limited or absent, making it a valuable tool in various data analysis and pre-processing contexts.

Given a meta-dataset matrix X, where columns are meta-features and rows are data points, the goal is to select a subset of meta-features x_i from X using the Variance Threshold $VarTh$ method seen in Equation 4.2. Firstly, the variance Var of a meta-feature x_i have to be calculated according to Equation 4.1,

$$\mathrm{Var}(x_i) = \frac{1}{n}\sum_{j=1}^{n}(x_{ij} - \bar{x}_i)^2 \tag{4.1}$$

$$\mathrm{VarTh}(X) = p(1-p) \tag{4.2}$$

where $\mathrm{Var}(x_i)$ is the Variance of x_i, n is the number of data points, x_{ij} is the value of x_i for the j-th data point, and $\bar{x}_i$ is the mean value of meta-feature x_i. Since the Variance Threshold method works by removing features whose variance falls below a threshold, if $\mathrm{Var}(x_i)$ is less than the threshold p, the meta-feature x_i is considered to have low variance and is removed from meta-dataset X. For this current experiment, Variance Threshold was employed to remove meta-features where values do not change in more than 80%, 85%, 90%, or 95% of the observations. All these configurations were tested and the one which generated the best results was reported.

4.1.2 Univariate Analysis

The so-called Univariate Feature Selection is a broad category of methods that encompasses the process of strategically choosing the most pertinent features through the application of rigorous statistical tests (KOHAVI *et al.*, 1995). Its operational principle hinges on deploying a battery of statistical tests to evaluate the intrinsic relationships between each individual input feature and the output target (PARMEZAN; LEE; WU, 2017). Thus, features that exhibit robust statistical associations with the output variable are selectively retained for subsequent analysis, while the rest is discarded. This Univariate Analysis offers a diverse array of statistical tests for consideration, including but not limited to the Chi-squared test and the ANOVA F-value test. These versatile tests empower the ability to tailor FS according to the unique dataset characteristics, thereby optimizing model performance while reducing computational overhead. The following describes two of these statistical tests in detail.

4.1.2.1 Chi-squared

The Chi-squared is a statistical test that can assess the dependency between two variables. Although it has common similarities with the coefficient of determination, this test is primarily used to examine whether two categorical variables have some kind of association. The Chi-squared test finds application in various domains, with a notable use being the identification of features that exhibit strong relationships with a target variable, contributing to better-informed FS. Initially, the test involves calculating the Chi-square statistic that quantifies the difference between observed and expected frequencies under the null hypothesis of no association. Subsequently, features are assessed based on their chi-square statistic values, with higher values meaning greater relevance to the target variable, making them essential for further analysis or modeling. At the same time, features that exhibit weaker dependencies with the target are systematically excluded from further consideration. The process then starts by constructing a contingency table for each feature, where each cell represents the observed frequency for a specific combination of feature values and classes. These observed frequencies are compared to the expected frequencies under the independence assumption. The chi-square statistic is computed by summing the squared differences between observed and expected frequencies and normalizing them by the expected frequencies. Features with low p-values are considered strongly associated with the target variable and are chosen for further analysis. In short, Chi-squared is simple, computationally efficient, and widely used for FS, making it an advantageous tool for handling complex datasets and identifying key features. Equation 4.3 shows its computation,

$$\chi^2 = \sum_i \sum_j \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{4.3}$$

where χ^2 is the Chi-squared statistic, O_{ij} is the observed frequency (i, j), and E_{ij} is the expected frequency in (i, j) under the assumption of independence between the feature and target variable.

The expected frequency E_{ij} for each cell is calculated according to Equation 4.4,

$$E_{ij} = \frac{(\text{row total}) \times (\text{column total})}{\text{grand total}} \tag{4.4}$$

where (row total) is the total number of observations in the i-th row (feature category), (column total) is the total number of observations in the j-th column (target category), and grand total is the total number of observations in the entire dataset. The Chi-squared statistic quantifies how much the observed frequencies deviate from what would be expected under this independence assumption, where larger chi-squared values indicate a stronger association between features and targets.

4.1.2.2 ANOVA

The statistical method known as ANOVA uses the F test to evaluate the linear dependence between input variables and targets, being designed mainly to evaluate quantitative data (LOMAX; HAHS-VAUGHN, 2013). Thus, the ANOVA metric assigns higher scores to features that exhibit a robust correlation with a target variable (JR, 1997). In turn, the F value (F) generated by the F test plays a key role in examining whether different groups, formed by categorizing numerical features based on the target vector, demonstrate statistically significant differences in their means. Essentially, the ANOVA F test evaluates the significance of variance between groups and, consequently, the relevance of individual characteristics to the target variable. These tests are commonly used in FS, facilitating the identification of the most informative characteristics and thus improving model performance and even interpretability. In short, F ANOVA or ANOVA F-test is an essential test statistic used to determine equality of means across multiple groups, typically corresponding to multiple features relative to a categorical target variable.

The F-value can be calculated as in Equation 4.5,

$$F = \frac{\text{Between-group variability}}{\text{Within-group variability}} \tag{4.5}$$

where F is the ANOVA F-value, Between-group variability measures the variation between different feature groups (classes), and Within-group variability measures the variation within each feature group (class). This calculation involves comparing the variance among group means to the variance within each group. A larger F-value indicates a greater difference between the group means relative to the variability within each group. In its most basic form, ANOVA extends the scope of the t-test beyond two means by offering a statistical test to determine whether two or more population means are equal. The ANOVA F-value is computed by comparing the variance among the means of different feature groups to the variance within each group. In the context of FS, one can use the F-value to rank or select features based on their ability to discriminate between classes or groups represented by the target variable, proving to be a valuable tool for enhancing the FS process in ML applications. For this current experiment, both Chi-squared and F ANOVA were used to select meta-features according to a percentile with the highest score for each FS method. The percentiles considered were 80%, 85%, 90%, and 95%.

4.1.3 Principal Component Analysis

The Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction and data transformation (PEARSON, 1901). Given a dataset with a set of data points, each represented by a vector of multiple features, PCA aims to find a new set of orthogonal axes called the Principal Components (PC) in which the variance of the data is maximized (JOLLIFFE; CADIMA, 2016). Thus, PCA does so by transforming the original data into a new coordinate system, where the PCs are orthogonal and ordered by the amount of variance they capture. These PCs are linear combinations of the original features, and PCA ensures that the first PC captures the maximum variance in the data. Consequently, each subsequent component captures the maximum remaining variance while remaining orthogonal to the previous components (WOLD; ESBENSEN; GELADI, 1987).

PCA can be defined in terms of a data matrix X of N data points with D features, where it seeks to find a set of orthogonal vectors, *PCs*, that can best explain the variance in the data. The first *PC* corresponds to the direction of maximum variance, and subsequent *PCs* capture decreasing amounts of variance. The first *PC* can be found by identifying the eigenvector corresponding to the largest eigenvalue of the covariance matrix of X, as seen in Equation 4.6,

$$\Sigma \mathbf{v} = \lambda \mathbf{v} \tag{4.6}$$

where, Σ is the covariance matrix of $\mathbf{X}$, $\mathbf{v}$ represents the eigenvector associated with *PC*, indicating the direction, and λ is the eigenvalue corresponding to *PC*, indicating the amount of variance it captures. The projection of the original data $\mathbf{X}$ onto the *PCs* results in a reduced-dimensional representation $\mathbf{Z}$, as seen in Equation 4.7,

$$\mathbf{Z} = \mathbf{X} \cdot \mathbf{V} \tag{4.7}$$

where $\mathbf{V}$ is the matrix of *PCs*. Subsequent principal components can be obtained in a similar manner, with the constraint that they are orthogonal to the previously found components.

Although known for reducing data dimensionality while preserving most of the original variance and sometimes even boosting ML performance, PCA can also be employed for tasks such as Data Visualization and Noise Reduction (FILCHENKOV; PENDRYAK, 2015). When dealing with data visualization, PCA can project data into a lower-dimensional space, making it easier to visualize and interpret the data (WOLD; ESBENSEN; GELADI, 1987). At the same time, by focusing on the top *PCs*, one can effectively remove noise from data. In the experiments of this chapter, PCA, as an FE technique, is adopted as the principal baseline to the more simple FS methods for being a more complex DR technique and the most popularly used. Thus, the selected components correspond to that maintain 80%, 85%, 90%, or 95% of the original data points, following the standards of (ALCOBAÇA *et al.*, 2018). Each of these configurations was tested for the present experiments and the one which yielded the best results was reported.

4.2 Experimental Setup

This section presents the complete experimental setup for the experiments with four dimensionality reduction techniques applied to twenty-eight meta-datasets related to the AR problem. To guarantee transparency across experiments and comparisons and to facilitate reproducibility, details are given in the following. Section 4.2.1 introduces the employed meta-datasets related to many classical optimization problems. Section 4.2.2 describes the baseline methods used as meta-classifiers in the comparative analyses. Lastly, Section 4.2.3 presents the evaluation methodology that includes data pre-processing, evaluation metrics, implementation details, software, and hardware specifications used to train the meta-models.

4.2.1 Meta-Datasets

This section briefly presents the twenty-eight meta-datasets collected from Aslib (BISCHL *et al.*, 2016), a benchmark library used in the experiments of this chapter. This benchmark provides data on the Algorithm Selection/Recommendation problem considering a range of classical optimization problems, such as the Constraint Satisfaction Problem (CSP), Propositional Satisfiability Problem (SAT), Travelling Salesman Problem (TSP), among others. Furthermore, Aslib contains information concerning various algorithms and their performances when applied to solving these problems, thus culminating in a rich source of meta-data. Such meta-datasets were chosen due to their widespread usage in the current literature, in addition to presenting interesting and diverse characteristics. This includes a wide range of optimization problems, different numbers of samples, meta-features, and classes, ranging from simple binary problems to complex multi-class problems. Table 8 shows statistics from these meta-datasets.

Table 8 provides a comprehensive overview of the AR meta-datasets, showing the diverse nature of the employed data across a broad spectrum of real-world scenarios. These meta-datasets exhibit notable heterogeneity, spanning a wide range of attributes. Notably, the number of examples within these datasets varies significantly, with examples ranging from a minimum of 100 in CSP-Minizinc-Obj-2016 and CSP-Minizinc-Time-2016 to a maximum of 9714 in TTP-2016. Additionally, the meta-features that capture high-level characteristics display substantial variability, ranging from as few as 22 in CPMP-2015 to a substantial 157 in OPENML-WEKA-2017. These meta-datasets also demonstrate diversity in the number of classes. The CSP-2010, for instance, is the only meta-dataset that encompasses a binary classification problem, while others handle a broader array of multi-class problems, as seen in SAT12-ALL and PROTEUS-2014 with 29 and 22 distinct classes, respectively. The selection of such meta-datasets for evaluating dimensionality reduction techniques was supported by several factors. Their high diversity allowed for a comprehensive evaluation of DR methods across varying scenarios. The representation of real-world datasets ensures the relevance and applicability of

Table 8 – AR Meta-datasets' statistics (PEREIRA; SANTOS; CARVALHO, 2021).

Meta-dataset	Examples	Meta-Features	Classes
ASP-POTASSCO	1294	139	11
BNSL-2016	1179	93	8
CPMP-2015	527	22	4
CSP-2010	2024	67	2
CSP-Minizinc-Obj-2016	100	123	3
CSP-Minizinc-Time-2016	100	117	4
CSP-MZN-2013	4636	117	10
GRAPHS-2015	5723	36	6
MAXSAT-PMS-2016	596	44	12
MAXSAT-WPMS-2016	630	53	10
MAXSAT12-PMS	876	31	6
MAXSAT15-PMS-INDU	601	57	16
MIP-2016	214	120	3
OPENML-WEKA-2017	105	157	3
PROTEUS-2014	4021	32	22
QBF-2011	1368	46	5
QBF-2014	1248	46	13
QBF-2016	825	67	14
SAT03-16_INDU	2000	139	10
SAT11-HAND	296	66	9
SAT11-INDU	300	67	11
SAT11-RAND	592	53	8
SAT12-ALL	1614	58	29
SAT12-HAND	767	70	22
SAT12-INDU	1167	74	21
SAT12-RAND	1362	98	9
SAT15-INDU	300	87	10
TTP-2016	9714	58	18

the study's findings. These meta-datasets also have served as benchmarks in prior AR research, providing greater reliability (ALCOBAÇA *et al.*, 2018; BENGIO; LODI; PROUVOST, 2021). Therefore, their differing levels of dimensionality, volume, and complex class structures provide a comprehensive assessment of DR techniques under diverse data conditions.

4.2.2 Baseline Methods

This section presents a concise overview of the foundation meta-classifiers employed in experimental comparisons with all four DR techniques. Such models were selected for their distinct learning paradigms, simplicity, data efficiency, and fast convergence. Among them, there is K-Nearest Neighbors (KNN), an instance-based and lazy learning algorithm, the tree-based models, C4.5 Decision Tree (DT) and Random Forest (RF), the latter being an ensemble model, and the Support Vector Machine (SVM), a kernel-based model. Table 9 lists these models used as meta-classifiers with their respective hyperparameters and tuning space ranges.

Table 9 – Meta-classifiers and their hyperparameters (PEREIRA; SANTOS; CARVALHO, 2021).

Meta-Classifier	Hyperparameter	Range
K-Nearest Neighbors	Nº of nearest neighbors	(1, 51)
C4.5 Decision Tree	Min samples split	(2, 51)
	Min samples leaf	(2, 51)
	Max depth	(2, 31)
Support Vector Machine	C	(1, 32769)
	Gamma	(1, 32769)
Random Forest	Nº of estimators	(1, 1025)
	Max depth	(1, 21)

In addition to what was previously mentioned, the selection of meta-classifiers seen in Table 9 is substantiated by their prevalent usage in AR and their role as established models for the problem. This, combined with the diversity of paradigms and learning biases, offers a practical relevance for validation concerning real-case AR scenarios, contributing to a more robust assessment of which models and configurations obtain the most benefits from DR. And regarding such models, some details should be mentioned.

For SVM, it was employed the Radial Basis Function (RBF) kernel, a versatile and popular choice that allows to modeling complex, non-linear relationships in the data, especially for classification tasks (VAPNIK, 1999). The RBF kernel, also known as the Gaussian kernel, is defined as in Equation 4.8,

$$\mathrm{K(x,\, x')} = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{4.8}$$

where x and x' are data points in the feature space, and σ is a hyperparameter that controls the kernel's width and influences the smoothness of the decision boundary. Assigning high similarity to data points that are close in the feature space, and lower similarity to points that are farther apart, the RBF function allows SVMs to model intricate decision boundaries. The RBF kernel is also flexible, thus being applicable to a wide range of applications. Its ability to map data into

higher-dimensional spaces without explicitly computing the transformations makes it suitable for a variety of datasets. For this, the σ hyperparameter plays a crucial role in determining the smoothness of the decision boundary. A small σ results in a more complex and wiggly boundary, while a large σ leads to a smoother decision boundary, thus providing control over the trade-off between overfitting and underfitting. In addition, the RBF kernel is highly effective when there are no strong assumptions about the data distribution, as it can adapt to a wide range of data types and is particularly useful when the decision boundary is not a simple geometric shape.

For both decision C4.5 DT and RF tree-based models, the Gini criterion is used to assess the quality of a split when constructing the DTs (OLSON; MOORE, 2016). Gini is a measure of impurity that quantifies how often a randomly chosen element would be incorrectly classified. This Gini impurity, denoted as $Gini(D)$ for a dataset D with K unique classes can be calculated using Equation 4.9,

$$\text{Gini(D)} = 1 - \sum_{i=1}^{K} (p_i)^2 \tag{4.9}$$

where p_i is the probability of randomly selecting an element of class i from D. In turn, the Gini criterion for a split in a DT is calculated as in Equation 4.10,

$$\text{Gini Criterion} = \sum_{j} \frac{|D_j|}{|D|} \cdot Gini(D_j) \tag{4.10}$$

where D_j is the subset of D after the split, $|D_j|$ is the number of elements in D_j, and $|D|$ is the total number of elements in D. Therefore, the Gini criterion quantifies the impurity reduction achieved by a split, as it measures the difference between Gini impurity from a dataset and the weighted sum of Gini impurities for the subsets after splits. A lower Gini criterion indicates a purer split and is desirable when constructing DTs, helping to find the best splits during construction by minimizing impurity and maximizing classification purity.

To calculate the distances between data points in the instance-based KNN, the Minkowski distance was adopted and adjusted, a metric considered to be a generalization of both Euclidean and Manhattan distances (BISHOP, 2006). The distance is given by Equation 4.11,

$$\text{Minkowski}(P,Q) = \left(\sum_{i=1}^{n} |P_i - Q_i|^p \right)^{1/p} \tag{4.11}$$

where P and Q are two points in a n-dimensional space, n is the number of dimensions in that space, P_i and Q_i are the values of the i-th dimension, and p is a parameter that determines the order of the Minkowski distance. When $p = 1$, it corresponds to the Manhattan distance, and when $p = 2$, it corresponds to the Euclidean distance. In essence, the Minkowski distance calculates the distance between two points by taking the absolute differences of each dimension, raising them to the power of p, then summing them and taking the $1/p$-th root of the result. This allows for a customization of the distance metric based on p, making it a flexible choice for measuring dissimilarity between points in different dimensions.

4.2.3 Evaluation Methodology

This section briefly presents the evaluation methodology defined to conduct the experiments in this chapter. Given the aim of comparing and assessing the impact of different DR techniques on meta-data, and since the employed meta-datasets varied in terms of feature cardinality, all features were scaled to the [0, 1] interval using the Minimum-Maximum scaler (HAN; PEI; KAMBER, 2011). This re-scaled also helped to facilitate the meta-classifiers fitting. Concerning the meta-models, their hyperparameter tuning process was conducted using Random Search with K-Fold Cross-Validation, in which data is split into K subsets of equal size for training and tuning (BERGSTRA; BENGIO, 2012). The standard split size of $K = 10$ was employed (RUSSELL; NORVIG, 2009), where training and tuning were done on the $K = 9$ folds, and performance metrics were calculated on the left testing fold. Given a hyperparameter space of a specific ML model, this process is repeated thirteen times until all folds have been evaluated, each time with a random sampling of the hyperparameter space. Thus, the hyperparameters set with the best performance is selected to be applied to test data.

As previously mentioned, every DR technique was evaluated according to its impact on three criteria: Predictive Performance, Dimensionality Reduction, and Pipeline Runtime. For measuring the performances, the Balanced Accuracy metric was adopted (BRODERSEN *et al.*, 2010). Balanced accuracy is a fair version of the original accuracy metric for multi-class problems, used to mitigate the class imbalance problem observed in the employed meta-datasets. Regarding dimensionality reduction, this reduction is simply the difference between the number of dimensions in the original meta-dataset and the resulting feature dimensions after applying a feature selection or feature extraction technique. Lastly, the pipeline execution time corresponds to the average time to perform the entire end-to-end training setup, which includes the application of a DR method and the induction of the meta-classifier.

To assess if performance differences are statistically reliable when using DR methods, appropriate statistical tests are essential. Given the scenario involving multiple meta-classifiers, meta-datasets, and DR techniques, the Friedman test is employed as it accommodates dependencies among samples and does not assume any particular distribution (FRIEDMAN, 1937). The Friedman test is a non-parametric statistical test used to determine if there are significant differences between multiple related groups. It assesses whether there are differences in the distributions of several paired samples and is particularly useful when dealing with non-independent data or when data distribution is unknown. To complement it, the Nemenyi test is also applied. The Nemenyi test is a post-hoc test commonly used alongside the Friedman test, performing pairwise comparisons between groups to identify specific differences in performance (NEMENYI, 1962). When the Friedman test indicates that there are significant differences among groups, the Nemenyi test helps pinpoint which groups differ from each other. This combined approach provides a comprehensive understanding of the relationships and differences between groups in a multi-group comparison, making it a powerful tool for statistical analysis.

Bar and box plots were used to analyze result distributions and their properties, while Critical Difference (CD) diagrams were employed to illustrate the statistical comparisons between distributions. CD diagrams are powerful graphical tools to compare two or more distribution populations after a statistical test (DEMŠAR, 2006). In this chapter, a Friedman-Nemenyi test with 95% confidence level is applied and the generated result distributions for Balanced Accuracy, Dimensionality Reduction, and Pipeline Runtime are plotted using CD diagrams. A higher position toward 1 on the diagrams indicates superior method performance, implying that methods on the left outperform the remaining. It is also worth mentioning that CD intervals that intersect with method lines indicate statistically equivalent results. Consequently, no method exhibits significant superiority over the others when connected, being statistically indistinguishable.

The entire experimental setup for this chapter, encompassing meta-classifiers and DR methods implementation alongside the associated documentation, is available at GitHub [1]. All code was written in Python, and only open-source libraries were used. The popular Scikit-learn (PEDREGOSA *et al.*, 2011) was used for implementing both meta-classifiers and DR methods, including Feature Selection techniques and PCA. In addition, meta-data extracted from the benchmark library Aslib (BISCHL *et al.*, 2016) was manipulated and pre-processed using Pandas (TEAM, 2020). In terms of hardware resources, all training and validation was conducted on a single Linux server with an Intel Xeon CPU E5-2620 v2 2.10GHz 24 cores processor and 128GB of RAM. It is noteworthy that experiments were conducted without any GPU usage, emphasizing the efficiency and computational accessibility of this experiment.

[1] https://github.com/geantrindade/DR_MtL

4.3 Results and Discussion

The experiments presented in this section were designed to answer the following research questions: (i) Does applying dimension reduction methods on meta-data associated with the AR problem yield any predictive performance enhancements? (ii) What is the efficiency of dimension reduction methods in reducing meta-data dimensionality? (iii) Are there any observable impacts on pipeline runtime when employing dimensionality reduction methods? The following sections address these questions through analysis and comparisons from 4.480 end-to-end training runs considering the full setup of four meta-learners, twenty-eight meta-datasets, four DR techniques, plus the original setup without DR, everything executed ten times through K-Fold Cross-Validation. The four meta-learners employed were SVM, KNN, DT, and RF, all trained with meta-datasets from the Aslib benchmark. Throughout the process of training, these meta-datasets were submitted to three feature selection methods, Variance threshold, Chi-squared, and F-ANOVA, in addition to one Feature Extraction method, the PCA.

4.3.1 Predictive Performance

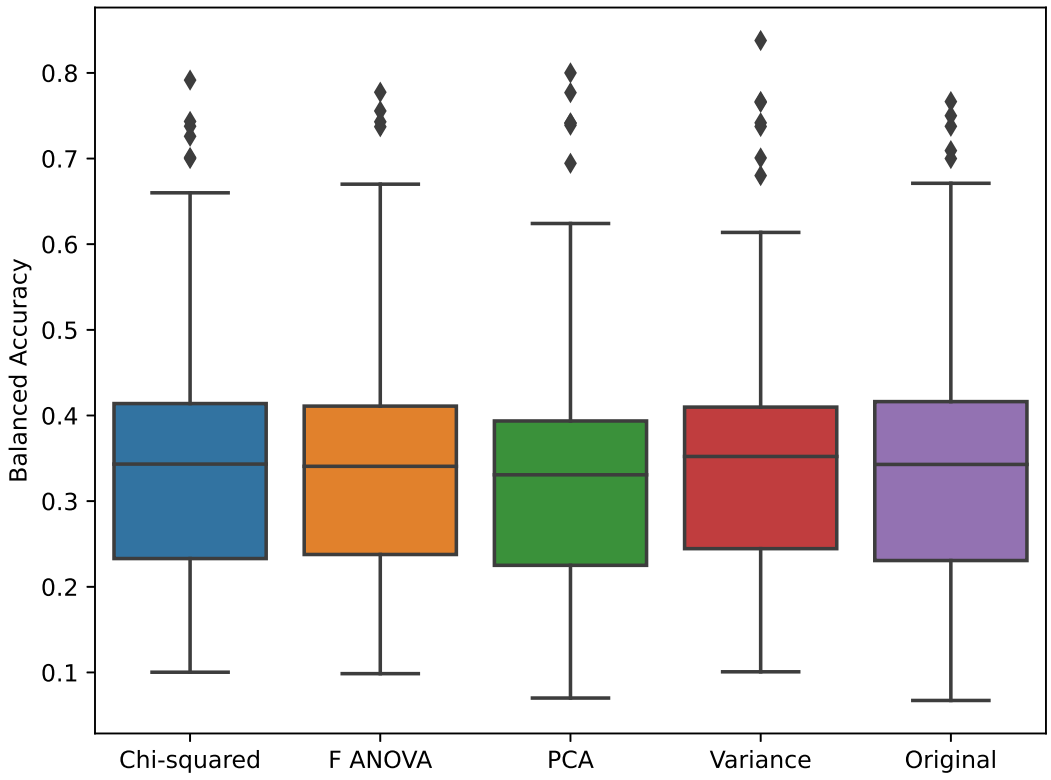


Figure 20 – Balanced Accuracy comparison (PEREIRA; SANTOS; CARVALHO, 2021).

In order to verify whether DR methods yield any improvements in terms of predictive performance, an analysis was conducted on the resulting balanced accuracy distributions from the application of DR techniques to every meta-dataset used in the training of meta-learners. As illustrated by the box plots of Figure 20, some findings indicate that all DR methods demonstrated similar performance levels, even in comparison to the original meta-datasets. Notably, PCA and F-ANOVA showed slightly better median results. Another interesting and expected observation is that the Variance threshold exhibited slight variance reductions in performances across meta-datasets and meta-classifiers, thereby presenting a narrower range of values. It is worth mentioning, however, that this method produced a higher number of sparse maximum outliers. Furthermore, with regard to outliers, a noteworthy observation is that their prevalence remained relatively consistent across the different DR methods applied, and such outliers were consistently localized within the range of maximum values only.

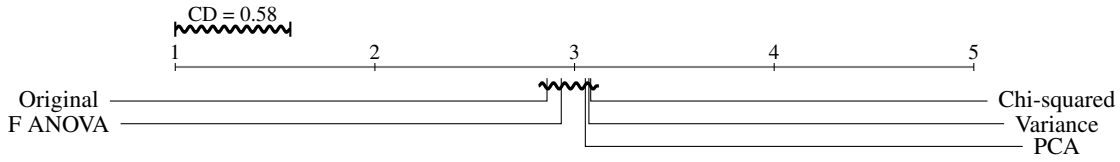


Figure 21 – Balanced Accuracy's Critical Difference (PEREIRA; SANTOS; CARVALHO, 2021).

For complementing the distribution analysis of the predictive performances, Figure 21 presents the Critical Diagram results from the Nemenyi post-hoc statistical test. By examining these results, it is revealed that despite the observed stability of the Variance method, with the exception of its outliers, the statistical evaluation establishes that F-ANOVA presented slightly superior performance among DR techniques, as seen in the previous case. However, a CD interval analysis indicates that none of the DR methods produced statistically significant improvements in overall predictive performance for the AR problem. Nevertheless, it is worth highlighting that the use of DR techniques demonstrated performance comparable to that achieved with the original, unaltered data. Nevertheless, this outcome can be viewed as positive feedback, as it aligns with the expectations of achieving comparable results to the original data while simultaneously reducing the input data's dimensionality in this experiment.

4.3.2 Dimensionality Reduction

The following-up experiment consists of a comprehensive analysis of the Feature Selection methods' effectiveness in reducing meta-dataset dimensions. This encompassed the percentage reduction examination across all pipelines involving each meta-dataset and all meta-learners. As seen in Figure 22, the distribution outcomes indicate that the DR methods consistently achieved substantial reductions, with all methods, except for the Variance Threshold, surpassing an impressive 80% reduction rate on average for all meta-datasets. In particular, Chi-squared, PCA, and F-ANOVA exhibited similar and stable reduction averages, with the latter showing

slightly better results among all. On the other hand, the Variance Threshold demonstrated the worst performance on average in addition to having greater variability, as evidenced by the higher standard deviation. Still, the amount of reduction was considerably relevant.

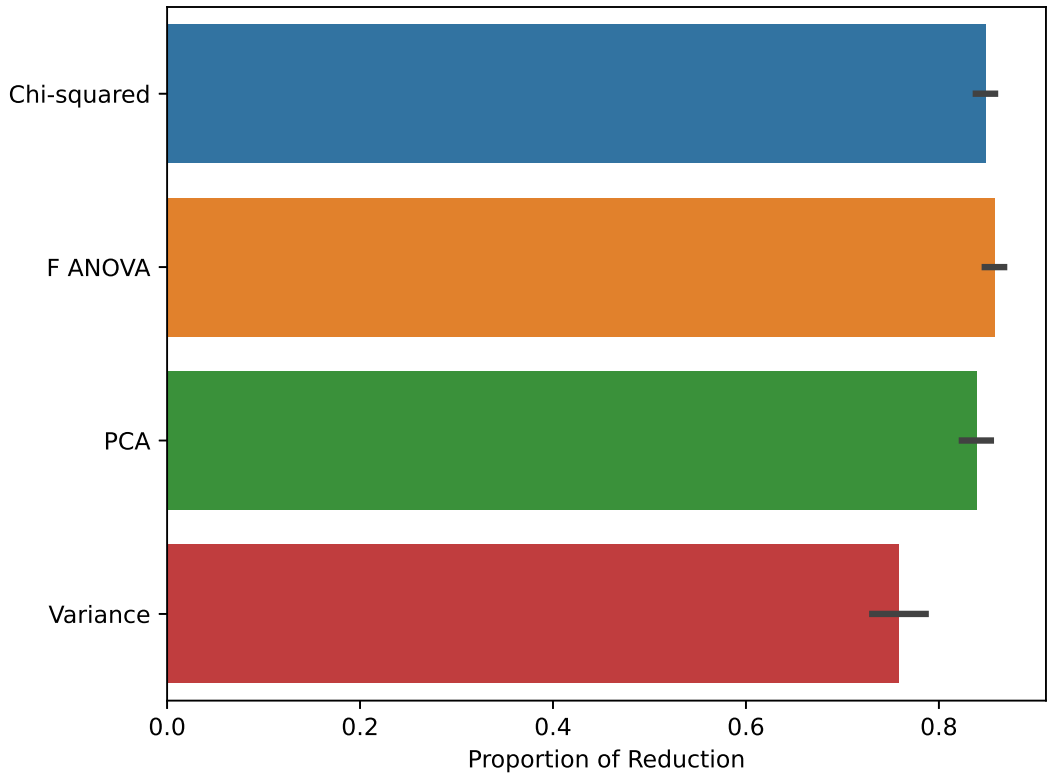


Figure 22 – DR percentage comparison (PEREIRA; SANTOS; CARVALHO, 2021).

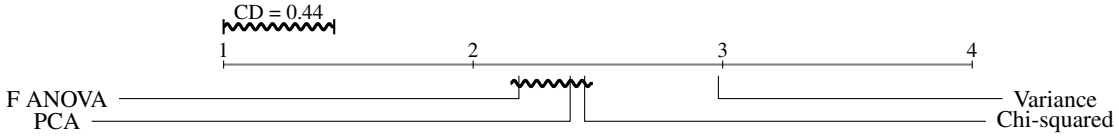


Figure 23 – DR's Critical Difference (PEREIRA; SANTOS; CARVALHO, 2021).

Upon a meticulous statistical analysis comparison using the CD diagrams shown in Figure 23, a close examination reaffirms the excellence of F-ANOVA and the limitations of Variance Threshold as the top-performing and underperforming models, respectively. Furthermore, it becomes evident that F-ANOVA, PCA, and Chi-squared methods demonstrate statistically similar performance, with F-ANOVA maintaining a slight edge. Notably, it is worth noting that the lighter-weight Feature Selection methods, namely F-ANOVA and Chi-squared, yield comparable statistical outcomes to the more complex Feature Extraction method, PCA. This underscores the effectiveness of Feature Selection in reducing the original data dimensions while incurring minimal computational costs. In addition, Variance Threshold emerges as a statistically inferior choice for dimensionality reduction when compared to its counterparts. These findings, complemented by the insights from Section 4.3.1, also point to a significant redundancy within

these meta-datasets, as it is possible to yield remarkably similar outcomes to those using the original meta-datasets while utilizing only 20% of the meta-features.

4.3.3 Pipeline Runtime

This final analysis aims to assess the potential impacts on pipeline runtimes associated with the application of dimensional reduction techniques. Thus, runtimes from all pipelines executed within our experiments were collected. This includes the time required for both meta-feature sub-set extraction, in the case of Feature Selection methods, and the original meta-feature space transformation when using a Feature Extraction method. Additionally, the runtime for data fitting was also considered since it is a dependent variable in this whole process. Figure 24 displays the runtime distributions for each DR method along with the original setup. The findings indicate that pipeline runtimes were relatively short in general, with maximum values approximately approaching 2 seconds. The median values across these distributions exhibited similar values and were in close proximity to the first quartile (Q1) and the minimum values. An intriguing observation is that the variations primarily occurred within the range of maximum values, while minimum values, Q1, and the interquartile range remained notably consistent.

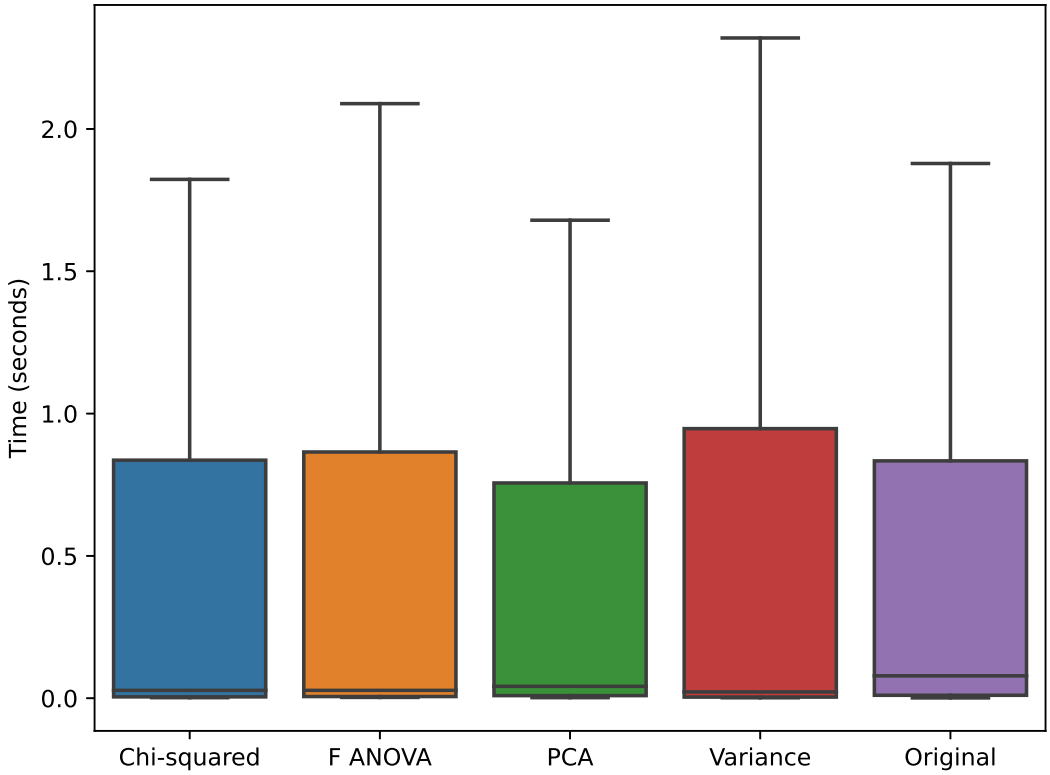


Figure 24 – Pipeline Runtime comparison (PEREIRA; SANTOS; CARVALHO, 2021).

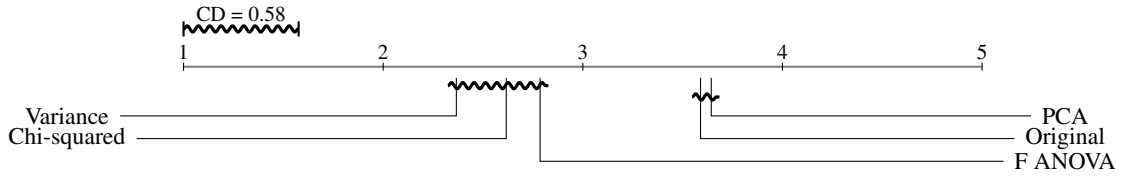


Figure 25 – Pipeline Runtime's Critical Difference (PEREIRA; SANTOS; CARVALHO, 2021).

When analyzing the statistical comparisons through the CD diagram shown in Figure 25, a closer look at the pipeline execution times reveals notable insights. Specifically, the runtimes for Variance, Chi-Square, and F-ANOVA have shown to be statistically similar and notably shorter than the pipelines using the original data and PCA. Adopting feature selection methods, in particular, emerges as a superior strategy to reduce overall pipeline runtime compared to not adopting the approach. These results suggest that despite the additional time required for dimensionality reduction, the total execution time covering the training process remains shorter. On the other hand, the presence or absence of PCA in pipelines does not confer statistically significant changes in execution time. However, it is important to highlight that although these statistical analyses prove that it is beneficial to use DR methods from the perspective of pipeline execution time, the runtime using the original data is already low. Therefore, future analyses with larger meta-datasets and with longer processing times are necessary in order to validate whether the observed differences are scalable and behave similarly.

4.4 Chapter Remarks

This chapter presented the second exploratory experiment with Meta-Learning as a foundation step to optimize NAS. More specifically, it focused on the study of meta-data dimensionality through the usage of several dimensionality reduction techniques applied to the Algorithm Recommendation problem used again as proof of concept. Once it was verified that meta-knowledge is effective for recommending models and that it can generalize across algorithms and meta-datasets, analyzing the properties of such meta-data was necessary. To this end, this subsequent study relied on an expanded set of twenty-eight meta-datasets related to AR encompassing a wider range of optimization problems. As it is common for the problem of the curse of dimensionality to occur in meta-bases, and a high dimensionality was observed in the AR meta-data, in addition to a high ratio in relation to the number of examples, it was necessary to study the relevance of meta-features in such databases. To check how representative and relevant the meta-features are, Feature Selection and Feature Extraction methods were used. Such Feature Selection methods were chosen based on their simplicity and fast processing, namely Variance Threshold, F-ANOVA, and Chi-squared. As the objective was not the study of DR techniques per se, Feature Selection methods were prioritized for being simpler and having interpretability, as they do not transform the feature space like Feature Extraction methods. For this reason, only the Principal Component Analysis (PCA) was adopted as a baseline, given its wide usage and to counter Feature Selection methods.

The performed experiments in this chapter were designed to answer the following research questions: "Does applying dimension reduction methods on meta-data associated with the AR problem yield any predictive performance enhancements?", "What is the efficiency of dimension reduction methods in reducing meta-data dimensionality?", and "Are there any observable impacts on pipeline runtime when employing dimensionality reduction methods?". These questions were addressed in analysis and discussions encompassing three evaluation criteria, respectively: (i) Predictive performance, where through Figures 20 and 21 is analyzed the impact of dimensionality reduction on model performance; (ii) Dimensionality reduction, where Figures 22 and 23 compare the percentages of dimensionality reduction in relation to the original data; and (iii) Pipeline runtime, where through Figures 24 and 25 is analyzed the reductions in pipeline execution times caused by the application of DR techniques.

In a comprehensive empirical investigation, the presented chapter sheds light on the effectiveness of Dimensionality Reduction techniques. As a result, it was shown that DR methods may not notably enhance predictive performance, but their effectiveness shines through in reducing meta-feature or feature dimensionality as well as overall pipeline runtimes while achieving comparable performance with the original data. This is especially the case for meta-datasets with a high number of non-informative features. Feature Selection methods F-ANOVA and Chi-squared demonstrated comparable performance to the more complex Feature Extraction

method PCA in predictive performance and dimensionality reduction efficiency but were superior in reducing pipeline runtime. This highlights the feasibility of adopting lightweight techniques for dimensionality reduction in Algorithm Recommendation, emphasizing the trade-offs between various methods. The study also showcases that, despite not universally improving predictive performance, applying dimensionality reduction reduces around 80% of meta-features, achieving comparable performance with shorter runtimes. Therefore, this concludes that various meta-datasets have many non-informative meta-features and that it is possible to obtain high predictive performance using around 20% of the original meta-features. Therefore, due to their natural trend for high dimensionality, DR methods should be used for Meta-Feature Selection and Meta-Feature Extraction. In conclusion, the insights from this chapter contribute to the discourse on ML techniques in Algorithm Recommendation, laying a foundation for further research and the development of more efficient Meta-Learning approaches, NAS included. The following chapter presents in detail a new method resulting from the combination of knowledge generated from the previous and current chapters called MbML-NAS.

CHAPTER

5

MODEL-BASED META-LEARNING FOR NAS

This chapter presents a novel method called Model-based Meta-Learning for Neural Architecture Search (MbML-NAS) to address the challenges of data efficiency, model complexity, and interpretability in NAS. After examining the applicability of Meta-Learning (MtL) and the dimensionality issues of meta-datasets, this chapter advances the study of MtL applicability to NAS by introducing a simplified MtL method that uses few representative meta-features to predict ConvNets' performances and select candidate models. To this end, MbML-NAS leverages prior knowledge from neural architectures and task-specific information through interpretable meta-features and simple meta-predictors to efficiently select suitable architectures.

In summary, this chapter presents the following contributions:

- A novel and simple Prediction-based NAS method that uses few interpretable meta-features with traditional regression models to attain comparable predictive performance to more intricate state-of-the-art models using only 0.04% and 1.1% of the search spaces;
- Novel meta-datasets comprising interpretable meta-features that characterize the general attributes of architectures from NAS-Bench-101 and NAS-Bench-201;
- A comprehensive assessment encompassing an examination of predictors' errors, standalone performances, and comparative analyses with traditional regression models, RL, EA, BO, Gradient/One-shot, Training-free, and Prediction-based models;
- A thorough interpretability analysis providing insights into how meta-information extracted from NAS-Bench-101 and NAS-Bench-201 influences the learning of meta-predictors.

The remaining content of this chapter is organized as follows: Section 5.1 presents the related literature. Section 5.2 describes the research contribution in detail. Section 5.3 details the experimental setup for reproducibility purposes. Section 5.4 discusses the main results. Lastly, the final chapter considerations are presented in Section 5.5.

5.1 Related Work

Research on NAS emerged from the urging necessity to automate NN design (HUTTER; KOTTHOFF; VANSCHOREN, 2019). This field revolves around the automated specification of various hyperparameters within neural architectures, encompassing layer operations, activation functions, layer interconnections, among others (ZOPH; LE, 2016). NAS methods, therefore, aim to automate the complete process of engineering NNs. Many learning strategies have been employed to discover promising neural architectures, including approaches such as Reinforcement Learning (RL), Evolutionary Algorithms (EAs), Bayesian Optimization (BO), Gradient-based Optimization, Training-Free NAS, and Prediction-based NAS. In Zoph and Le (2016), RL is explored with the REINFORCE algorithm to train candidate architectures generated by a controller LSTM network. In turn, Falkner, Klein and Hutter (2018) proposed to use a Bandit-based strategy for hyperparameter optimization using BO, the HyperBand (BOHB), a combination for efficient NAS. Luo *et al.* (2018) proposed a One-shot model named Neural Architecture Optimization (NAO) that uses gradient-based optimization to approximate a continuous decision space for finding good architectures. Liu, Simonyan and Yang (2018) also developed a One-shot model called Differentiable Architecture Search (DARTS) that uses gradient descent and continuous relaxation but with a different search space than NAO, which leads to distinct performance and scalability trade-offs. Real *et al.* (2019) go back to classical EAs by using Regularized Evolution (RE) to evolve a neural architecture population regularized by a limited search space, in which the fitness function is their validation performance. Lastly, the pioneering work of Mellor *et al.* (2021) introduced the concept of Neural Architecture Search without Training (NASWOT), where linear maps of untrained networks are extracted to derive a score that, when high at initialization, indicates improved final accuracy after training.

Another NAS avenue involves the use of Performance Prediction models, an emerging research area known as Prediction-based NAS (DUDZIAK *et al.*, 2020; LI *et al.*, 2021). In Tang *et al.* (2020) is introduced SSANA, a semi-supervised method that uses an auto-encoder and a Graph Convolutional Network (GCN) to predict performances based on learned architecture representations. Lukasik *et al.* (2021) present SVGe, a comprehensive two-sided graph Variational AutoEncoder (VAE) used to reconstruct architectures from different search spaces and predict architectures beyond those seen during training. They also adapt the generative model DGMG (LI *et al.*, 2018) as a VAE with a single decoder for evaluating their encoder architecture. Deng, Yan and Lin (2017) propose Peephole, a method employing a dense layer and an LSTM with a block-based generation scheme to predict network performance based on architecture and past performances. Zhang *et al.* (2019) introduce D-VAE, a deep generative model leveraging Graph Neural Networks to encode DAGs into a continuous latent space, facilitating the search for DAGs with improved performance through Bayesian Optimization. Sun *et al.* (2019) introduce E2EPP, an integrated performance predictor based on a Random Forest and an evolutionary

DNN to forecast the performance of ConvNets. Wen *et al.* (2020) propose a GCN-based Neural Predictor that learns from one-hot encodings representing architectural layers and adjacency matrices denoting layer connections. This two-stage model includes a classifier to filter out unstable and underperforming architectures and a GCN regressor for performance predictions. Additionally, Li *et al.* (2021) put forth GenNAS-N, a generic NAS framework designed to address the limitations of task-specific methods. This approach uses regression on synthetic signal bases for architecture evaluation, enabling a self-supervised task focused on capturing the inherent capabilities of architectures in modeling and transforming input signal patterns.

Although popular, these approaches entail several drawbacks. RL, for instance, can be resource-intensive and data-inefficient, demanding multiple iterations for an in-depth search space exploration and optimal policy acquisition so that it can yield suitable architectures (WILLIAMS, 1992). Similarly, EAs may be computationally expensive and prone to premature convergence, especially when facing a fitness landscape abundant in local optima, a prevalent challenge in several NAS search spaces (REAL *et al.*, 2019). In turn, BO shows sensitivity to prior and acquisition function configurations. Often, BO incurs high computational costs when dealing with high-dimensional search spaces due to the numerous iterations requirement for updating its belief in promising architectures (FALKNER; KLEIN; HUTTER, 2018). Despite being fast, One-shot models present high computational costs as they require training a super-graph composing all architectures in a search space (LI; TALWALKAR, 2020). Besides, One-shot models may struggle to effectively generalize beyond the super-graph, thus demanding careful regularization and sampling strategies for diverse and high-quality architecture generation (BENDER *et al.*, 2018). Conversely, Training-free models rely on heuristics for architecture generation, limiting their ability to capture network component relationships and resulting in sub-optimal solutions and premature convergence, as they cannot incorporate feedback from the training data (CHEN; GONG; WANG, 2021). The Prediction-based approach, although promising in reducing the computational cost of identifying optimal architectures through performance predictions from a limited set of sampled architectures, comes with its own set of challenges (DUDZIAK *et al.*, 2020). Predicting architecture performance is a complex task that requires well-designed prediction models to capture the intricate relationship between architectural configurations and performances (NING *et al.*, 2020). Besides, such predictions' quality heavily depends on the representativeness and diversity of the sampled architectures. Additionally, this approach may face limitations in discovering architectures beyond the scope of the sampled pool, especially when the target range significantly deviates from the training data (LIU; TANG; SUN, 2021).

A possible solution to these issues lies in the use of Meta-Learning (MtL), which can reduce computational burden and accelerate the search process by leveraging prior experiences to build new knowledge faster using a small number of sampled architectures. Wang *et al.* (2020) propose an MtL method featuring a task-aware architecture controller to generate task-specific architectures, where MtL is used to learn meta-weights that efficiently adapt to new

tasks on corresponding architectures with only a few gradient descent steps. Lee, Hyung and Hwang (2021) address the limitations of conventional NAS on generalization across multiple tasks by proposing a framework trained on a database of datasets and pre-trained networks, where by using a cross-modal latent space learned through amortized MtL, it can generate architectures for novel datasets. Li *et al.* (2021) introduce GenNAS-N, another MtL method that aims to be task-agnostic and, for this, adopts a self-supervised regression task that relies on synthetic signal bases for architecture evaluation. Although these methods use MtL in various ways and share similarities with MbML-NAS, they address distinct issues and pose different research questions, hypotheses, and objectives than our proposal. Wang *et al.* (2020) deal with a toy regression example and few-shot classification problems, Lee, Hyung and Hwang (2021) consider priori models to generalize to unseen target datasets, and Li *et al.* (2021) centers around a semi-supervised task with synthetically generated data. In contrast, MbML-NAS focuses on generalizing to different search spaces and datasets by learning from general sets of interpretable meta-features and straightforward meta-predictors through prior performance evaluations.

5.2 Interpretable Meta-NAS with Fast Predictors

This section introduces a novel MtL-based method named Model-based Meta-Learning for Neural Architecture Search (MbML-NAS), designed to find promising ConvNets in Cell-based Search Spaces such as the popular NAS-Bench-101 and NAS-Bench-201 benchmarks. To predict architecture performances, MbML-NAS employs simplified yet effective linear models, such as Linear Regression and Stochastic Gradient Descent, and non-linear models, such as Tree-based models and Ensembles, to name a few. Thus, as a Prediction-based approach, MbML-NAS can serve a dual purpose as a Search Strategy and a Performance Estimation tool to speed up NAS. MbML-NAS learns from neural architecture performances and interpretable meta-features extracted directly from the architectures to predict the most promising candidates within the vast space of possible models. To the best of our knowledge, this is the first attempt to show that by using simple and interpretable meta-features and classical regression models, it is possible to find accurate neural architectures. In the following, an illustrative overview of MbML-NAS is shown in Figure 26, followed by a comprehensive elucidation of its framework procedure in Algorithm 2. Furthermore, essential details regarding the meta-features and meta-predictors employed in the proposal are presented in Tables 10 and 25, along with a thorough explanation of underlying concepts used in this new approach.

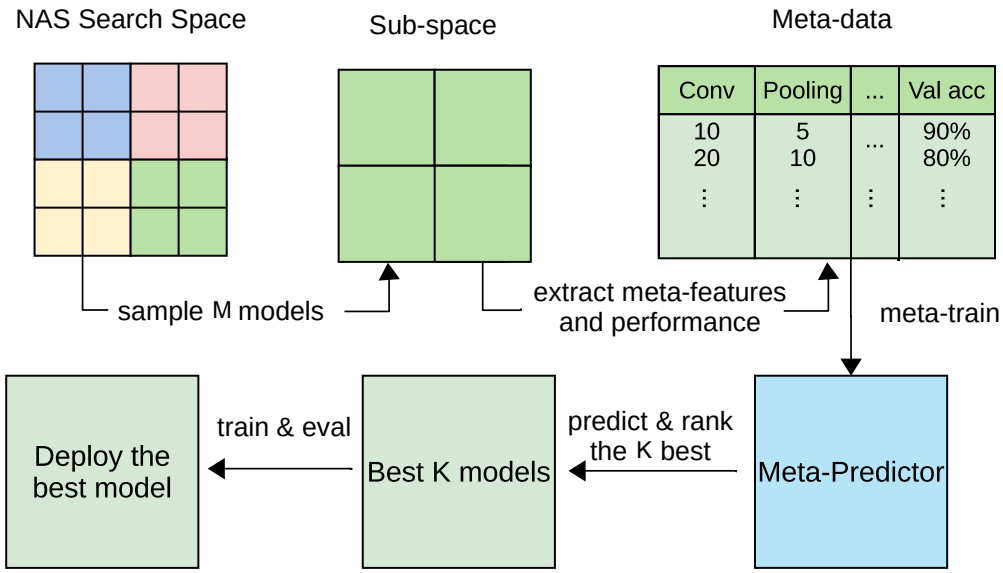


Figure 26 – MbML-NAS overview (PEREIRA *et al.*, 2023).

The MbML-NAS workflow starts by sampling a pre-defined number M of neural architectures at uniformly random, drawn from an expansive and intricate search space encompassing a wide variety of ConvNets configurations. Various simple statistics are then extracted from each sampled architecture within the compact sub-space, including but not limited to the number of convolutional layers, maximum kernel size, and validation accuracy. This systematic procedure effectively results in a meta-dataset, wherein each row corresponds to a meta-example encoded by a set of meta-features and a meta-target, encapsulating the extracted statistics and the predictive performance, respectively. The meta-dataset serves as the foundation for training a meta-predictor, which exhibits the flexibility to employ any regression model. Once a meta-predictor is fully trained, it undertakes the pivotal task of predicting and ranking the incoming and unseen ConvNets, and from the ranked list, the top-performing K architectures are selectively chosen for final validation. Each K chosen architecture is then trained and validated from scratch on the designated target dataset. Consequently, the architecture that achieves the highest test accuracy is selected, and its test performance is reported. In the following, more detailed insights into the intricate workings of MbML-NAS are shown in Algorithm 2.

Algorithm 2: MbML-NAS Procedure (PEREIRA *et al.*, 2023).

Input : Search Space S, Nº of architectures to train M, Training dataset Tr, Validation dataset Va, Test dataset Te, Meta-model $Meta$, Meta-features to extract Mf, Nº of architectures to predict P, Nº of architectures to select K

Output : highest test accuracy architecture from K_best_archs

```
1  K_best_archs ← ∅
2  Meta_dataset ← ∅
3  Train_archs ← sample(S, M)
4  for each arch in Train_archs do
5  |   Meta_features ← meta_extraction(arch, Mf)
6  |   while arch is untrained do
7  |   |   train(arch, Tr)
8  |   end
9  |   meta_target ← val(arch, Va)
10 |   Meta_dataset ← Meta_dataset + (Meta_features, meta_target)
11 end
12 while Meta is untrained do
13 |   train(Meta, Meta_dataset)
14 end
15 Predict_archs ← sample(Train_archs, P)
16 K_best_archs ← predict(Meta, Predict_archs, K)
17 Acc_test ← ∅
18 for each arch in K_best_archs do
19 |   while arch is untrained do
20 |   |   train(arch, Tr)
21 |   end
22 |   Acc_test ← Acc_test + test(arch, Te)
23 end
24 K_best_archs ← sort_by(K_best_archs, Acc_test)
```

Algorithm 2 comprises five main phases that characterize MbML-NAS workflow: (i) Meta-dataset Creation (Lines 4-11): The process begins by selecting M architectures at uniformly random from a Search Space S (line 3). Each architecture has its meta-features Mf (line 5) and meta-target (line 9) extracted, which are then combined to form a comprehensive $Meta\_dataset$ (line 10); (ii) Meta-model Training (Lines 12-14): With a built meta-dataset, the meta-predictor can be trained to approximate the validation accuracy of unseen ConvNets, where a diverse set of linear and non-linear regression models can be used; (iii) Selection of K Best Architectures (Lines 15-16): Built on the predictions of trained meta-predictors, the top K architectures are selected; (iv) End-to-End Training of K Architectures (Lines 18-23): The K selected architectures are then trained from scratch on the target dataset; and (v) Ranking and Reporting (Line 24): The final phase involves the ranking of K best architectures based on their test accuracy, where

the one with the highest test performance is selected and its performance reported (Line 24). It is essential to mention that this is the generic and abstract procedure of MbML-NAS, which does not change for other search spaces or datasets. However, it is worth highlighting that neural architectures are not trained from scratch in this study. Instead, performance metrics are queried from NAS-Bench-101 and NAS-Bench-201 pre-computed tabular benchmarks, which already contain such information. Furthermore, Table 10 provides detailed information on the interpretable meta-features extracted and used by MbML-NAS.

Table 10 – Meta-Features extracted from NAS benchmarks. Columns *NAS-Bench-101* and *NAS-Bench-201* show the range values [min, max] for each meta-feature, *parameter* refers to the whole architecture, while the remaining are meta-information concerning the neural cells. All values were normalized before training (PEREIRA *et al.*, 2023).

Meta-Feature	**Description**	**NAS-Bench-101**	**NAS-Bench-201**
parameters	Architecture's parameters	[227274, 49979272]	[0.0805, 1.5387]
conv_num_layers	Number of convolutional layers	[0, 5]	[0, 6]
conv_kernel_min	Minimum kernel size	[0, 3]	[0, 3]
conv_kernel_max	Maximum kernel size	[0, 3]	[0, 3]
conv_kernel_mode	Most common kernel size	[0, 3]	[0, 3]
pool_num_layers	Number of pooling layers	[0, 5]	[0, 6]

Table 10 showcase the six meta-features extracted from NAS-Bench-101 and NAS-Bench-201 neural cells that encode fundamental ConvNets characteristics: Notably, *parameters* exhibits different scales across benchmarks. In NAS-Bench-101, it is the trainable parameters in a complete architecture, while it is the total architecture size in megabytes in NAS-Bench-201. In practice, they are comparable since the more trainable parameters, the larger the model size. This discrepancy is mitigated through Min-Max Normalization that re-scales all values to the [0, 1] interval. Additionally, cells from these benchmarks may lack convolutional or pooling layers, as seen in the meta-features range. Furthermore, distinctions between benchmarks exist regarding pooling layers (*pool_num_layers*). While NAS-Bench-101 employs max pooling (*maxpool_num_layers*), NAS-Bench-201 uses average pooling (*avg_pool_num_layers*). However, they are the same for MbML-NAS, being analogous to the *parameters* interpretation. Each cell is also characterized by its validation accuracy, which serves as a meta-target for MbML-NAS. The NAS-Bench-10's meta-target is named $final\_validation\_accuracy$, while NAS-Bench-201's is acc_valid. Despite the distinct nomenclature, the targets are functionally equivalent, as meta-datasets are standardized across benchmarks and datasets. For simplicity, original benchmark names are retained for meta-features and meta-targets, with alternative references in the text.

MbML-NAS employs traditional regression models as meta-predictors whose choosing was based on several factors such as their distinct learning paradigms, data efficiency, rapid convergence, simplicity, and interpretability. These models fall into two main categories: regularized linear models, and non-linear models. Table 25 in Appendix A shows all models used by MbML-NAS and their respective hyperparameters tuned via Random Search. Considering the linear models, MbML-NAS uses Linear Regression (LR), Bayesian Ridge (BR), and Stochastic Gradient Descent (SGD). Linear Regression is a foundation model that learns linear relationships between input features and target variables, offering simplicity, computational efficiency, and interpretable feature coefficients (HASTIE *et al.*, 2009). In turn, Bayesian Ridge introduces Bayesian inference for parameter estimation of a linear regression model, providing a probabilistic perspective and regularization to prevent overfitting (BISHOP, 2006). As for Stochastic Gradient Descent, it extends Linear Regression by employing stochastic optimization to efficiently handle larger datasets (RUSSELL; NORVIG, 2009).

On the other hand, the tree-based models used in MbML-NAS capture non-linear relationships. These models include C4.5 Decision Tree (DT), Random Forest (RF), Gradient Boosting (GB), and AdaBoost (AB). C4.5 constructs decision rule hierarchies based on input feature that ensures straightforward interpretability (DUDA; HART; STORK, 2000). Random Forest combines multiple decision trees and, by using bootstrapping and random feature subsampling, it can improve generalization (MITCHELL, 1997). As the Random Forest, Gradient Boosting is an ensemble method, but it iteratively trains weak decision tree learners to create a strong predictive model (GAMA *et al.*, 2011). AdaBoost is another ensemble but with higher flexibility since it combines various weak classifiers, such as decision trees or even linear models, with adaptive boosting for enhanced performance (WITTEN *et al.*, 2016). Therefore, the choice for these regression models reflects a careful balance between several criteria that include model complexity, predictive performance, and suitability for the target task.

5.3 Experimental Setup

This section presents the entire experimental setup involved in the training and validation of MbML-NAS meta-predictors. To guarantee the transparency of experiments and facilitate their reproducibility, details are given in the following. Section 5.3.1 introduces the employed image datasets and NAS benchmarks. Section 5.3.2 describes the baseline methods used in the comparative analyses. Lastly, Section 5.3.3 presents the evaluation methodology that includes data pre-processing, evaluation metrics, implementation details, software and hardware specifications used to train all MbML-NAS meta-predictors, and baselines.

5.3.1 Datasets and Benchmarks

This section briefly presents the three datasets and two NAS benchmarks used for the creation of meta-datasets, training of MbML-NAS meta-predictors, and comparisons with state-of-the-art NAS methods. All datasets are related to the Image Classification task, chosen due to their widespread usage in the current literature. In addition, they exhibit simplicity but are shown to be powerful for validating new proposals both in NAS and in ML as a whole. The NAS benchmarks were also chosen based on the same criteria.

CIFAR-10 [1]: Proposed by Krizhevsky, Hinton *et al.* (2009) as a labeled subset of the Tiny Images dataset (TORRALBA; FERGUS; FREEMAN, 2008), CIFAR-10 is a widely popular dataset used for training CV methods and as a proof-of-concept for ML models in general. It contains 60.000 colorful images of 32x32 pixels partitioned into 50.000 and 10.000 instances for training and testing, respectively. These images are categorized into 10 classes comprehending objects and animals, whose variety of shapes and concepts aligned with simplicity makes it one of the most valuable resources for practical applications.

CIFAR-100: Seen as a CIFAR-10 extension, it has the same standardization and number of instances as its predecessor (KRIZHEVSKY; HINTON *et al.*, 2009). However, it presents increased complexity with a broader range of 100 classes, 500 training images, and 100 test images per class. The real richness of CIFAR-100 lies in how its 100 classes are grouped into 20 superclasses. Each image has two labels, a "fine" label specifying its precise class, and a "broad" label denoting its superclass. These superclasses provide a higher-level categorization, facilitating a more comprehensive understanding of the data.

ImageNet16-120 [2]: ImageNet16-120 is built from the down-sampled ImageNet16×16 dataset (CHRABASZCZ; LOSHCHILOV; HUTTER, 2017), a variant of the original ImageNet. Proposed by Deng *et al.* (2009) as a substantial resource for training learning

[1] https://www.cs.toronto.edu/ kriz/cifar.html
[2] http://image-net.org/download

algorithms, the ImageNet dataset comprises over 20.000 classes and 14.000.000 colorful images of various sizes. Its popularity dramatically increased with the ILSVRC competition, fostering the development of CV methods to this day. Therefore, ImageNet16-120 was built by selecting a data subset of ImageNet16×16, thus resulting in a dataset with 151.007 training images, 3.000 validation images, 3.000 test images, and 120 classes.

NAS-Bench-101 [3]: Introduced by Ying *et al.* (2019) as a tabular benchmark to facilitate research on NAS, this dataset stands as a valuable resource to enhance the reproducibility of NAS experiments while minimizing computational overhead. The dataset efficiency lies in the possibility of querying the pre-computed information, which empowers researchers to evaluate a diverse range of models within milliseconds, speeding up the NAS experimentation process dramatically. NAS-Bench-101 is composed of 423.000 unique Convolutional architectures trained and evaluated multiple times on the CIFAR-10 dataset, culminating in an expressive collection of over 5.000.000 pre-trained models.

NAS-Bench-201 [4]: Proposed by Dong and Yang. (2020) as an extension to NAS-Bench-101, it extends the pioneering work with a different and larger Search Space, performance records on CIFAR-10, CIFAR-100, and ImageNet16-120, on top of supporting more recent NAS methods such as NASNet, ENAS, and DARTS. NAS-Bench-201's search space is inspired by the popular Cell-based space, and it contains a total of 15.625 neural cell candidates. Notably, it offers a comprehensive array of logged metrics for each dataset, including training loss, validation loss, test loss, training accuracy, validation accuracy, and test accuracy. All this extended data helps researchers delve into a rich performance data repository across various neural architecture experiments.

Table 11 provides statistics from the two NAS benchmarks. NAS-Bench-101 composes 423.000 models trained only on CIFAR-10, while NAS-Bench-201 encompasses 15.625 models trained on CIFAR-10, CIFAR-100, and ImageNet16-120. From NAS-Bench-101 is possible to query statistics from the training of all models after 4, 12, 36, and 108 epochs. Such data were collected to construct the four meta-datasets referred to as the number of each epoch subset (e.g., 4 epochs subset) in the presented analyses. From NAS-Bench-201 is possible to query statistics from 1 to 200 epochs, but not all epochs have statistics on the test split. To standardize the experimental setup, the same number of subsets built from NAS-Bench-101 with CIFAR-10 was used for NAS-Bench-201 with CIFAR-10, these being the subsets of all models trained for 4, 12, 36, and 108 epochs. In addition, subsets using the maximum epochs available were also built, which are usually the ones used by the community, totaling seven meta-datasets from NAS-Bench-201 data and four from NAS-Bench-101.

[3] https://github.com/google-research/nasbench
[4] https://github.com/D-X-Y/NAS-Bench-201

Table 11 – NAS benchmarks/datasets statistics.

Benchmark	Models	Datasets	Subsets	Meta-Features	Epochs
NAS-Bench-101	423k	1	4	6	{4, 12, 36, 108}
NAS-Bench-201	15k	3	7	6	[1, 200]

Table 12 presents more data processing and experimental setup specifications. Such details include the two NAS benchmarks used to create the meta-datasets employed in the training of all meta-predictors, the target Datasets present in these benchmarks, the Subsets (from different epochs in the benchmarks), the Train Size for each sampled subset, and the used random Seeds adopted to run experiments multiple times. As previously mentioned, the subsets 4, 12, 36, and 108 were also created from NAS-Bench-201 with CIFAR-10 to maintain consistency with NAS-bench-10. Furthermore, for the other datasets in NAS-Bench-201, subsets were created with the final performances from epoch 200. Regarding the different sample sizes, the standards of the strong baseline Neural Predictor were followed throughout all experiments. Finally, the seeds used for the entire NAS process of each repeated experiment, from sampling through tuning and end-to-end training of the models, are also specified.

Table 12 – General setup details for all the experiments with NAS-Bench-101 and NAS-Bench-201.

Benchmark	Dataset	Subset	Train Size	Seed
NAS-Bench-101	CIFAR-10	4	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
		12	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
		36	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
		108	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
NAS-Bench-201	CIFAR-10	4	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
		12	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
		36	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
		108	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
		200	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
	CIFAR-100	200	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}
	ImageNet16-120	200	{43, 86, 129, 172, 344, 860}	{0, 1, 10, 42, 100, 123, 1000, 1234, 12345}

5.3.2 Baseline Methods

This section provides a concise overview of foundation models employed in the experimental comparisons within and with MbML-NAS. Within the suite of meta-predictors used by MbML-NAS, one encounters traditional regression models, a lower bound, and an upper baseline. As mentioned in Section 5.2, MbML-NAS uses regularized linear models and tree-based models to predict the performances of ConvNets. Such algorithms were selected for their different learning paradigms, simplicity, data efficiency, fast convergence, and interpretability. The linear models are Linear Regression (LR), Stochastic Gradient Descent (SGD), and Bayesian Ridge (BR). From tree-based models, there are C4.5 Decision Tree (DT), Random Forest (RF), AdaBoost (AB), and Gradient Boosting (GB). Table 25 lists these models with their respective hyperparameters, with the addition of Neural Predictor reproduction from the original study. All model's hyperparameters were adjusted using Random Search multiple times, where different random seeds were used, as specified in Table 12.

In order to establish robust baselines for evaluation, a lower bound known as the Dummy regressor [5] and an upper bound baseline known as the Oracle were also adopted. The Dummy regressor adopts a straightforward approach by applying simple rules to fit the training dataset and predict outputs. There are three key strategies: (i) Mean: This rule consistently predicts the target's mean value, providing a direct and minimally informative prediction strategy; (ii) Median: Similarly, this strategy adheres to a constant prediction, consistently forecasting the median value of the target distribution; and (iii) Quantile: This approach takes a specified quantile from the target distribution and consistently predicts it, offering a more flexible yet static prediction method. Given that Random Search was employed to fine-tune the hyperparameters of MbML-NAS's meta-predictors, this regressor configures the most effective among these predefined rule-based strategies. In short, this lower bound baseline serves as a critical benchmark to assess the performance and effectiveness of MbML-NAS's predictive capabilities.

Following the evaluation methodology outlined in Wen *et al.* (2020), an upper bound baseline commonly referred to as Oracle was established. This baseline has the distinct advantage of having free access to the true validation accuracy signals of each architecture in the dataset. With such privileged information, the Oracle can then select the architecture with the highest conceivable validation accuracy value without training. However, it is essential to recognize that even the Oracle can potentially make sub-optimal choices. This is due to the fact that the architecture with the highest validation accuracy is not always guaranteed to have the highest test accuracy, a phenomenon observed in both NAS-Bench-101 and NAS-Bench-201 benchmarks. This discrepancy arises due to the inherent differences between validation and test dataset target distributions. Consequently, a meta-predictor can potentially outperform the Oracle if it accurately infers the architectures with the highest test accuracies. Such instances highlight

[5] https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyRegressor

the need for NAS methods that can effectively bridge the gap between validation and test performances, contributing to more informed and robust architecture selections.

The GCN-based Neural Predictor proposed by Wen *et al.* (2020) serves as a prominent baseline for MbML-NAS, as both methods present similarities in predicting ConvNets performances based on neural hyperparameters information. Such a Neural Predictor is a two-stage model consisting of a classifier and a predictor that shares the same GCN architecture but with different output layers. The classifier's role is to filter unstable and poorly performed architectures that cannot surpass a predefined minimum accuracy threshold. Meanwhile, the predictor estimates the validation accuracies for these pre-selected architectures. In summary, the method operates in three key steps: (i) it trains a set of N random architectures to establish pairs of (architecture, validation accuracy); (ii) it predicts the accuracy of the N architectures and identifies the top K most promising candidates; and (iii) it trains the leading K architectures and deploys the one with the highest validation accuracy. For such, the Neural Predictor relies on input arrays representing architecture layers and an adjacency matrix denoting connections between layers to encode and learn the architectures. To be able to reproduce its results, the Neural Predictor was replicated using a publicly available third-party repository [6] which was modified accordantly since the authors released no official code.

Following Wen *et al.* (2020), the Neural Predictor was trained on NAS-Bench-101 only and with and without a classification stage, as it is the benchmark for which the proposal was initially designed and tuned. The training was performed using the same set of N, D, and K used in the original study. For each N, models were randomly sampled from NAS-Bench-101 using different random seeds. Besides, additional experiments were run with other NAS-Bench-101 subsets (besides 108 epochs used initially), train splits, and seeds, as described in Section 5.3.3. A threshold of 0.91 was used for the classifier's training, which is compatible with the median of validation accuracy on the 108 epochs subset. Although not mentioned by the authors, it was assumed that using the median was the adopted strategy. Consequently, the median of validation accuracies was used as the minimum threshold for the remaining subsets of NAS-Bench-101, resulting in the values of 0.23, 0.54, 0.86, and 0.91 for subsets 4, 12, 36, and 108 epochs, respectively. Despite the authors mentioning the adoption of Mean Squared Error (MSE) loss during the predictor's training, it is unclear if the same is applied to the classifier's training. Although it is not prohibitive to use MSE for classification, given that maximizing entropy in binary classification is more appropriate, the Binary Cross-Entropy (BCE) loss was adopted to train the classifier. Furthermore, the Sigmoid on the predictor's output layer was shifted to output values ranging from 0.1 (10%) to 1.0 (100%).

[6] https://github.com/ultmaster/neuralpredictor.pytorch

The selection of remaining baselines spans a range of NAS paradigms, each focusing on the automated exploration and generation of ConvNet architectures. These NAS techniques are categorized into distinct groups, including Reinforcement Learning, Evolutionary Algorithms, Bayesian Optimization, One-shot models, and the more recent Training-free and Prediction-based strategies. Therefore, the selected third-party methods are the final class of baselines used to assess MbML-NAS performance. These adopted methods were introduced in section 5.1, namely REINFORCE (WILLIAMS, 1992), RE (REAL *et al.*, 2019), BOHB (FALKNER; KLEIN; HUTTER, 2018), NAO (LUO *et al.*, 2018), DARTS (LIU; SIMONYAN; YANG, 2018), NAS-WOT (MELLOR *et al.*, 2021), GenNAS-N (LI *et al.*, 2021), SSANA (TANG *et al.*, 2020), E2EPP (SUN *et al.*, 2019), Peephole (DENG; YAN; LIN, 2017), D-VAE (ZHANG *et al.*, 2019), DGMG (LUKASIK *et al.*, 2021), and, finally, SVGe (LUKASIK *et al.*, 2021). It must be mentioned that such baselines are well-recognized for their strong performance on the two adopted NAS benchmarks. Thus, comparing MbML-NAS against these well-established baselines allows for a robust evaluation of the method's search efficiency and predictive capabilities.

5.3.3 Evaluation Methodology

This section briefly describes the evaluation methodology used in experiments with MbML-NAS and baselines. In alignment with Wen *et al.* (2020), this study follows a consistent approach for data splitting and hyperparameter tuning. A $\frac{1}{3}$-fold Cross-validation was employed on a random subset of size M, where Random Search was used for hyperparameters tuning. Upon completing the tuning process, the set of M examples was used to train the final meta-predictors where M is the entire training split. Since the aim was to ensure a less noisy performance estimation, all the remaining architectures (i.e., ALL - N) from the respective search spaces were employed for testing. Ten iterations of training/testing were conducted, each time with different random seeds using both NAS-Bench-101 and NAS-Bench-201 to enhance results reliability. Furthermore, six distinct training split sizes (43, 86, 129, 172, 344, and 860) were explored.

In order to comprehensively assess the performance prediction capabilities of MbML-NAS, an evaluation framework guided by prior Prediction-based NAS methodologies and widely used state-of-the-art NAS models was established. In line with the standard practices for Prediction-based techniques, MbML-NAS predictive performance was assessed by computing the Mean Squared Error (MSE) for each of its meta-predictors. Additionally, conventions prevalent in the NAS literature were adhered to by comparing the individual or standalone performances of MbML-NAS meta-predictors against other state-of-the-art NAS baselines. Thus, this involved evaluating the performance of architectures generated by various NAS methods in an end-to-end training setting. To provide a broader perspective, independent assessments of MbML-NAS standalone performances were also conducted, in addition to comparisons with the ones from the prominent baseline, Neural Predictor, across varying values of K in Figure 27.

A methodology based on the nature of tree-based and linear meta-predictors, alongside the numerical meta-features in use, was also defined to evaluate MbML-NAS interpretability. The first tool is the Feature Importance based on the Mean Decrease in Impurity (MDI) (LOUPPE, 2014). This impurity is quantified by the splitting criterion of tree-based models, being the MSE in this setup. Although the MDI can be calculated on either training or test split, the latter was used since it has more data samples and, thus, enhances reliability. In the case of linear models, the coefficient values assigned to the meta-features were extracted and treated as indicators of Feature Importance (RIBEIRO; SINGH; GUESTRIN, 2016; LIPTON, 2018). Since all meta-features were normalized, it is safe to compare the magnitudes of different coefficients, contributing to a thorough and straightforward assessment of interpretability.

To address the limitations inherent in Feature Importance methods, such as their bias towards high-cardinality features and the assumption of linear relationships between independent and dependent variables, the Permutation Importance method (BREIMAN, 2001) have been incorporated in this interpretability analysis. Such a tool assesses the reduction in a model's performance when feature values are randomly shuffled, unveiling a non-linear relationship between input features and targets. To enhance results reliability, the Permutation Importance of all the models and each meta-feature were calculated and averaged from ten repetitions, effectively reducing variance and producing more robust findings. Furthermore, Spearman correlations (SPEARMAN, 1961) were also computed to gain insights into the informativeness of meta-features and their relation with meta-targets, thus providing a more complete evaluation.

The complete experimental framework encompassing method implementation and associated documentation has been made openly available to the public through a GitHub repository [7]. The entire codebase was constructed using the Python language and relied on open-source libraries. Specifically, the popular Scikit-learn library (PEDREGOSA *et al.*, 2011) was employed for implementing the meta-regression models, while PyTorch (PASZKE *et al.*, 2019) was adopted to reproduce the Neural Predictor baseline (WEN *et al.*, 2020). In terms of hardware infrastructure, all the training and validations were conducted on a single Linux server equipped with an Intel Xeon CPU E5-2620 v2 2.10GHz processor with 24 cores and 128GB of RAM. It is noteworthy that the performed experiments were conducted without any GPU utilization, emphasizing the proposal efficiency and computational accessibility.

[7] https://github.com/geantrindade/MbML-NAS

5.4 Results and Discussion

Experiments presented in this section were designed to answer the following research questions: (i) Can simple meta-features effectively encode neural architectures so that traditional regression models can accurately predict their performances? (ii) Is it feasible to discover high-performing architectures through Meta-Learning with limited training examples? (iii) Can simple meta-features lead to generalization across diverse search spaces and datasets? (iv) Is it possible to effectively use interpretable meta-features and meta-predictors to get insights into the intrinsic characteristics of neural architecture search spaces? The next sections address these questions through analysis and comparisons between MbML-NAS and state-of-the-art NAS methods on both NAS-Bench-101 and NAS-Bench-201. As for the methods included in these comparisons, only the GCN-based Neural Predictor was reproduced, whilst the remaining baselines were cited for reference. Additional results, including more standalone comparisons, MSE analysis, data exploratory studies with the NAS benchmarks, among others, can be seen in the Appendix A.

5.4.1 Standalone Predictive Performance Analysis

This section presents a comparative analysis of standalone performances from the top-K architectures selected meta-predictors and baseline methods. Figure 27 provides insights into these comparisons by displaying the average top-1 test accuracy along with standard deviations across ten experimental runs. To ensure clarity and readability, results from some of the best linear and tree-based models have been incorporated alongside the lower and upper-bound baselines, which are consistently included in all plots. For a fair assessment, the Neural Predictor (GCN) baseline was exclusively trained on NAS-Bench-101, the benchmark for which it was originally designed and tuned. Additionally, the training set size of $M = 172$ was maintained, as outlined in Wen *et al.* (2020), as the best trade-off between computational budget and predictive performance. Regarding K values, $K = 10$ was used instead of the $K = 5000$ considered by the Neural Predictor. This choice is driven by practical considerations, reflecting a more feasible and cost-effective scenario for NAS, especially when computational resources are limited.

Figure 27 shows that the performances of MbML-NAS meta-predictors consistently exhibited a remarkable closeness to the Oracle across various scenarios. Especially on the ImageNet16-120 dataset, all meta-predictors showed competitive performances with the Oracle even when using smaller values of K such as $K = 3$ and $K = 4$. When compared to the lower bound Dummy, all MbML-NAS meta-predictors exhibited significantly superior performance. Even when Dummy's performance improves with increasing the value of K, it still lags behind all the compared meta-predictors. This inferiority of Dummy can be attributed to its simplistic approach of relying on simple rules, such as predicting the mean or median performance from a distribution, which fails to capture the intricate relationship between architectures and their

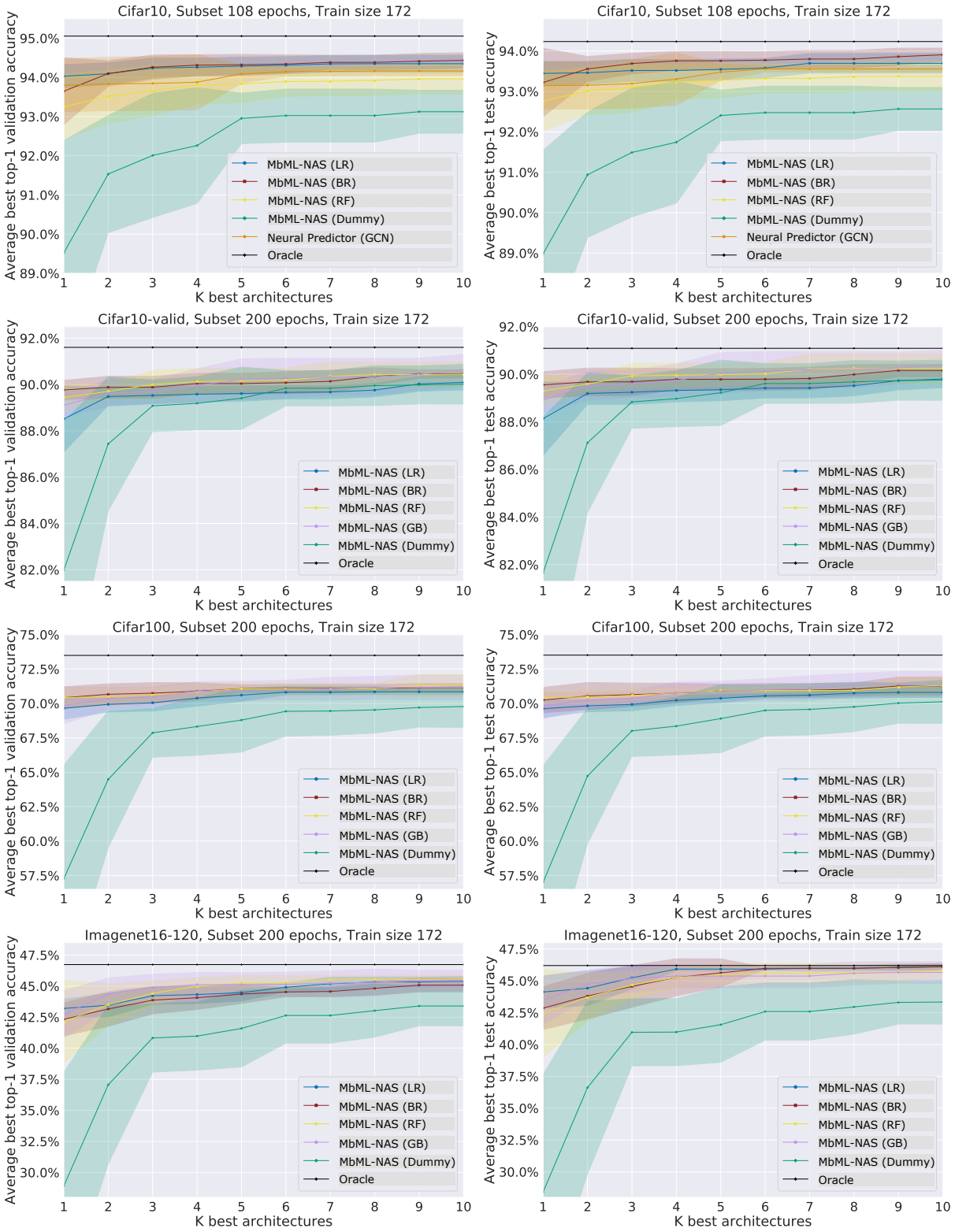


Figure 27 – Test Accuracy Averages and Standard Deviations from the K best architectures selected by meta-predictors on NAS-Bench-101 (Top left), NAS-Bench-201 with CIFAR-10 (Top right), CIFAR-100 (Bottom left), and ImageNet16-120 (Bottom right) (PEREIRA *et al.*, 2023).

performances. Furthermore, it can be seen that regression models are less sensitive to changes in K, showing only a marginal increase in performance. This is not the case with Dummy, which displays greater sensitivity to changes in K. When using larger values of K, the likelihood of identifying superior architectures becomes higher. In the extreme scenario of K being equal to the number of architectures in the test sample, the standalone performance equals that of the Oracle. Consequently, when a model's predictions are less accurate, increasing the number of top architectures K tends to lead to a significant enhancement in its standalone performance. This trend is observable in the case of Dummy but not for the other predictors.

Except for Dummy, MbML-NAS meta-predictors exhibit comparable performances overall with consistent improvements as K increased. BR and GB achieved slightly superior results in the resource-intensive setup ($K = 10$), which were closely followed by RF and LR. Conversely, LR emerged as the top performer in NAS-Bench-101 and NAS-Bench-201 with ImageNet16-120 in the cost-effective scenario ($K = 1$) but had the worst performance among the leading meta-predictors on both CIFAR-10 and CIFAR-100 from NAS-Bench-201, suggesting a higher overall variance. This discrepancy may indicate that these datasets possess greater non-linearity that LR could not effectively capture for being a linear model. Another possibility might be that the meta-examples used for LR were not sufficiently representative for generalization. Additionally, when comparing the meta-predictors' performances with the Neural Predictor (GCN), the latter outperformed RF but presented slightly inferior results on NAS-Bench-101 than LR and BR. Given that both MbML-NAS and Neural Predictor were trained with the same $M = 172$ and standardized experimental details, it can be inferred that MbML-NAS boasts a more effective feature representation than the GCN-based method that relies on topological information. Furthermore, MbML-NAS demonstrates superior usage of limited examples for selecting the best architectures. In the following Tables 13 and 14, detailed performance comparisons of the top-$K = 1$ architectures identified by all MbML-NAS meta-predictors and paired with state-of-the-art baselines using the most expensive setup ($M = 860$).

Table 13 shows results on NAS-Bench-101 and demonstrates that all MbML-NAS predictors, excluding Dummy, achieved performance levels akin to those state-of-the-art NAS methods. While the MbML-NAS primary focus lies not in predictive performance but in simplicity, fast convergence, data efficiency, and interpretability, it still achieved comparable results with several well-established NAS baselines that prioritize high performance. The top-performing method overall was GenNAS-N with a test accuracy of 93.92% using 500 training samples. Following closely, MbML-NAS delivered commendable results of 93.26% test accuracy with GB using 860 samples. In addition, MbML-NAS demonstrated comparable or superior performance to the recent NAO and NASWOT, even though they used 1000 training samples. In the case of NAO, it was expected to achieve superior results since it utilizes a super-graph encompassing all potential architectures from the search spaces and undergoing lengthy training. GenNAS-N also employs

Table 13 – State-of-the-art methods on NAS-Bench-101. Values are standalone test accuracy averages and standard deviations from MbML-NAS and baselines, *Optimal* is the best test accuracy possible to achieve in the benchmark, and *Time* is the training time in seconds (PEREIRA *et al.*, 2023).

Method	**CIFAR-10**	**Time**	**Strategy**
REINFORCE (WILLIAMS, 1992; YAN *et al.*, 2020a)	93.72±0.00	1000000	reinforcement
BOHB (FALKNER; KLEIN; HUTTER, 2018; YAN *et al.*, 2020a)	93.72±0.00	1000000	bayesian
RE (REAL *et al.*, 2019; MELLOR *et al.*, 2021)	93.87±0.22	12000	evolution
NAO (LUO *et al.*, 2018; WHITE *et al.*, 2021)	93.74±0.00	1000000	gradient
NASWOT (MELLOR *et al.*, 2021)	91.77±0.05	23	training-free
GenNAS-N (LI *et al.*, 2021)	93.92±0.01	20700	prediction-based
Neural Predictor (GCN) (WEN *et al.*, 2020)	93.15±0.01	55	prediction-based
Ours			
MbML-NAS (Dummy)	88.99±0.03	0.01	random
MbML-NAS (LR)	93.19±0.01	0.12	prediction-based
MbML-NAS (SGD)	93.03±0.01	1.82	prediction-based
MbML-NAS (BR)	93.11±0.01	0.63	prediction-based
MbML-NAS (DT)	92.55±0.01	0.12	prediction-based
MbML-NAS (RF)	93.16±0.01	6.65	prediction-based
MbML-NAS (AB)	92.60±0.01	3.44	prediction-based
MbML-NAS (GB)	93.26±0.01	5.89	prediction-based
Optimal	94.32		

a more intricate approach of generating synthetic data from the original benchmark to learn representations and then transferring the knowledge to the target task, incorporating GPU training in the process. In contrast, MbML-NAS follows a more straightforward procedure with GPU-free usage to achieve competitive outcomes. As for the individual MbML-NAS meta-predictors, no clear superiority of one model family was evident, which may suggest that this benchmark exhibits a more linear nature since even linear models achieved reasonable results.

Table 14 – State-of-the-art methods on NAS-Bench-201. Values are standalone test accuracy averages and standard deviations from MbML-NAS and baselines, *Optimal* is the best test accuracy possible to achieve in the benchmark, and *Time* is the training time in seconds (PEREIRA *et al.*, 2023).

Method	**CIFAR-10**	**CIFAR-100**	**ImageNet16**	**Time**	**Strategy**
REINFORCE (WILLIAMS, 1992; MELLOR *et al.*, 2021)	93.85±0.37	71.71±1.09	45.24±1.18	12000	reinforcement
RE (REAL *et al.*, 2019; MELLOR *et al.*, 2021)	93.92±0.30	71.84±0.99	45.54±1.03	12000	evolution
BOHB (FALKNER; KLEIN; HUTTER, 2018; MELLOR *et al.*, 2021)	93.61±0.52	70.85±1.28	44.42±1.49	12000	bayesian
DARTS (MELLOR *et al.*, 2021; LIU; SIMONYAN; YANG, 2018)	54.30±0.00	15.61±0.00	16.32±0.00	10890	gradient
NASWOT (MELLOR *et al.*, 2021)	92.96±0.81	69.98±1.22	44.44±2.10	306	training-free
GenNAS-N (LI *et al.*, 2021)	94.18±0.10	72.56±0.74	45.59±0.54	1080	prediction-based
Ours					
MbML-NAS (Dummy)	85.41±5.96	57.01±8.46	28.41±9.24	0.01	random
MbML-NAS (LR)	91.68±1.54	69.36±0.69	44.47±1.01	0.01	prediction-based
MbML-NAS (SGD)	91.95±1.45	69.66±0.66	43.46±1.27	0.31	prediction-based
MbML-NAS (BR)	91.88±1.41	69.77±0.47	43.66±0.62	0.01	prediction-based
MbML-NAS (DT)	92.88±0.71	68.88±1.69	43.49±1.73	0.01	prediction-based
MbML-NAS (RF)	93.36±0.20	70.33±0.85	42.99±4.21	1.31	prediction-based
MbML-NAS (AB)	92.60±1.61	70.04±1.40	43.40±2.79	0.25	prediction-based
MbML-NAS (GB)	93.03±0.52	70.02±1.17	44.28±1.42	1.41	prediction-based
Optimal	94.37	73.51	47.31		

Table 14 presents some findings concerning the NAS-Bench-201. These results demonstrate the superior performance of MbML-NAS across all datasets against two robust baseline methods, namely DARTS and NASWOT. Notably, within the various meta-predictors utilized by MbML-NAS, RF and GB emerge as the top-performing choices, where RF achieved 93.36% test accuracy on CIFAR-10 and 70.02% on CIFAR-100, while GB attained 44.28% on ImageNet16-120. These performance results can be attributed to the distinctive non-linear characteristics exhibited in NAS-Bench-201, in contrast to NAS-Bench-101, where all tree-based models surpass their linear counterparts, except for Logistic Regression (LR) on ImageNet16-120. In terms of remaining baselines, GenNAS-N was the lead performing with a test accuracy of 94.18% on CIFAR-10, 72.56% on CIFAR-100, and 45.59% on ImageNet16-120, closely followed by RE which demonstrates the comparable results of 93.92% on CIFAR-10, 71.84% on CIFAR-100, and 45.54% on ImageNet16-120. It is essential to note that these baseline approaches are non-interpretable and use more complex models that prioritize predictive performance, in addition to utilizing approximately 1000 examples. At the same time, MbML-NAS achieves competitive results with a more modest dataset of 860 examples.

A training time comparative of MbML-NAS meta-predictors in Tables 13 and 14 reveals intriguing insights. On NAS-Bench-101, Dummy and LR emerge as the fastest models, boasting training times of 0.01 and 0.12 seconds, respectively. In contrast, RF lags significantly, with a training time of 6.65 seconds. However, a notable shift occurs on NAS-Bench-201, where Dummy, LR, BR, and DT uniformly exhibit a swift time of 0.01 seconds, establishing themselves as the benchmark's fastest meta-predictors. Contrastively, RF and GB demonstrate relatively extended training of 1.31 and 1.41 seconds, marking them as the slowest models on NAS-Bench-201. Generally, training times vary across different meta-predictors, benchmarks, and datasets, but it is important to note that these variations are consistent. The models with the worst predictive performances exhibit lower training times, with the inverse occurring with the best-performing methods. We believe these differences can be attributed to each meta-predictor's underlying algorithms and computational requirements. Meta-predictors such as Dummy, LR, and DT adopt simpler algorithms with minimal computational overhead, resulting in shorter training times. Inversely, meta-predictors such as RF and GB leverage more complex algorithms and require additional computational resources, which leads to a longer training process. Furthermore, it is worth highlighting that the present approach generates lighter models than the primary baseline, Neural Predictor, which has a training time of 55 seconds on NAS-Bench-101.

Furthermore, upon examining the training times of MbML-NAS as seen in Tables 13 and 14, it becomes apparent that they rank as the most time-efficient among all baselines, even surpassing the Training-free approach. However, it is imperative to note that despite the lower training times, MbML-NAS relies on meta-data from the prior training of NAS-Bench-101 and NAS-Bench-201 architectures. Despite that, it is worth mentioning that such incorporation of pre-existing knowledge is a standard premise of all Meta-Learning (MtL) approaches (DUDZIAK *et al.*, 2020; WANG *et al.*, 2020; MUÑOZ *et al.*, 2018). Furthermore, in alignment with prior

research in the domains of NAS and MtL (LI *et al.*, 2021; DUDZIAK *et al.*, 2020; LEE; HYUNG; HWANG, 2021), it is assumed that the performance metrics of existing models serving as learning source material are readily available during the search process. In an analogous situation, the training times for GenNAS-N, markedly shorter both in NAS-Bench-101 and NAS-Bench-201 when compared to evolutionary, reinforcement, and gradient-based approaches, are derived from prior knowledge. More specifically, the training times on NAS-Bench-201 stem from a task search initially conducted on NAS-Bench-101, wherein the search model was transferred. Additionally, it is noticed that the longest training times are attributable to exhaustive and frequently time-consuming exploration approaches, exemplified by evolutionary and reinforcement techniques, in addition to the One-shot/Gradient models.

5.4.2 MSE Analysis

This section presents a comparative analysis involving all MbML-NAS meta-predictors and various Prediction-based NAS methods from the current literature. This comprehensive analysis is focused not on the standalone performances of selected architectures as in Figure 27 and Tables 13 and 14, but on the prediction accuracy of the NAS methods themselves. As each predictor tries to approximate the actual performances of candidate architectures, the accuracies of such predictors are measured by the Mean Squared Error (MSE) along with its corresponding standard deviations. Table 15 and 16 then present the MSE averages and standard deviations from ten runs of MbML-NAS and state-of-the-art NAS baselines.

Table 15 – MSE averages and standard deviations of NAS methods on NAS-Bench-101. *Train/Test* is the train and test sample size used for training and testing the models (PEREIRA *et al.*, 2023).

Model	**Train/Test**	**CIFAR-10**
SSANA (TANG *et al.*, 2020)	1000/422625	0.0031±0.0003
E2EPP (SUN *et al.*, 2019)	1000/422625	0.0042±0.0003
Peephole (DENG; YAN; LIN, 2017)	1000/422625	0.0071±0.0005
D-VAE (ZHANG *et al.*, 2019)	1000/-	0.0039±0.0003
DGMG (LUKASIK *et al.*, 2021)	1000/-	0.0037±0.0001
SVGe (LUKASIK *et al.*, 2021)	1000/-	0.0028±0.0001
Neural Predictor (GCN) (WEN *et al.*, 2020)	860/422765	0.0047±0.0001
Ours		
MbML-NAS (Dummy)	860/422765	0.0034±0.0001
MbML-NAS (LR)	860/422765	0.0031±0.0001
MbML-NAS (SGD)	860/422765	0.0032±0.0001
MbML-NAS (BR)	860/422765	0.0031±0.0001
MbML-NAS (DT)	860/422765	0.0030±0.0001
MbML-NAS (RF)	860/422765	0.0029±0.0001
MbML-NAS (AB)	860/422765	0.0030±0.0001
MbML-NAS (GB)	860/422765	0.0030±0.0002

Results in Table 15 showcase MbML-NAS outperforming all baseline methods except for SVGe, which yielded comparable results. Notably, the Random Forest (RF) meta-predictor achieved the lowest MSE of 0.0029, closely followed by SVGe with an MSE of 0.0028, a remarkable feat considering that RF used a smaller training dataset. Unfortunately, SVGe and the other baselines do not report the test sample sizes for comparisons, which negatively impacted the reproducibility of these methods. MbML-NAS also demonstrated superior efficiency when compared to the more complex Neural Predictor (GCN) since it employs the same number of architectures for training but achieves a lower MSE. Furthermore, an additional and interesting pattern similar to what was observed in the standalone performances on NAS-Bench-201 was noted in the results of 15, where total dominance of tree-based models occurs over linear models in terms of achieving lower MSE values. However, patterns are not seen for NAS-Bench-101 regarding the standalone performances of selected architectures, where mixed dominance is observed. This disparity suggests that while some meta-predictors excel in minimizing the errors, occasional discrepancies arise in selecting the optimal architectures that elucidate the models' context-dependent performance. In any case, Gradient Boosting (GB) and Random Forest (RF) still appear as the best-performing meta-predictor for NAS-Bench-101, both regarding standalone performances and MSE, with Dummy retaining its status as the least effective model. Subsequently, Table 16 presents a comprehensive MSE comparison for NAS-Bench-201.

Table 16 – MSE averages and standard deviations of NAS methods on NAS-Bench-201. *Train/Test* is the train and test sample size used for training and testing the models (PEREIRA *et al.*, 2023).

Model	**Train/Test**	**CIFAR-10**	**CIFAR-100**	**ImageNet16-120**
HAAP (LIU; TANG; SUN, 2021)	1000/-	0.0003±0.0001	-	-
MbML-NAS (Dummy)	860/14765	1.6459±0.0156	1.4786±0.0100	0.8560±0.0031
MbML-NAS (LR)	860/14765	1.3213±0.0122	0.9914±0.0077	0.4674±0.0025
MbML-NAS (SGD)	860/14765	1.3287±0.0175	0.9965±0.0126	0.4691±0.0036
MbML-NAS (BR)	860/14765	1.3235±0.0128	0.9923±0.0078	0.4676±0.0025
MbML-NAS (DT)	860/14765	1.3083±0.0296	0.9761±0.0161	0.4708±0.0049
MbML-NAS (RF)	860/14765	1.2931±0.0169	0.9594±0.0098	0.4640±0.0041
MbML-NAS (AB)	860/14765	1.3216±0.0487	0.9640±0.0228	0.4706±0.0061
MbML-NAS (GB)	860/14765	1.3223±0.0301	0.9757±0.0282	0.4657±0.0048

As showcased by Table 16, results on NAS-Bench-201 include only one baseline method named HAAP (LIU; TANG; SUN, 2021), which, to the best of our knowledge, it is the one single method that reports MSE values for this benchmark. However, the authors only report performances on the CIFAR-10 dataset, ignoring CIFAR-100 and ImageNet16-120. When compared to the best-performing MbML-NAS (RF), HAAP exhibits a lower MSE compared. However, this advantage comes at the cost of using more training examples. In addition, HAAP's paper does not disclose the test set's sample size, which can lead to misleading results. Among the different MbML-NAS variations, the lowest MSE of 1.2931 on CIFAR-10, 0.9594 on CIFAR-100, and 0.4640 on ImageNet16-120 were achieved by the Random Forest meta-predictor. It is worth emphasizing that, in line with previous observations on NAS-Bench-201, tree-based

meta-predictors consistently outperform linear models. This reinforces the non-linearity of NAS-Bench-201 and highlights the robustness of these predictors in capturing the complex relationship between meta-representations and predictive performances of architectures.

Upon an extensive examination of MbML-NAS meta-predictors across multiple scenarios and datasets, it became clear that two models have distinctly stood out from the remaining: Gradient Boosting (GB), which demonstrated the best standalone performances on NAS-Bench-101 with CIFAR-10 and NAS-Bench-201 with ImageNet16-120, and Random Forest (RF), which achieved the top overall results on NAS-Bench-201 with CIFAR-10 and CIFAR-100 datasets. While GB exhibited commendable performances, RF managed to achieve a lower MSE, showcasing greater robustness to more substantial prediction errors. Consequently, it can be confidently concluded that RF stands as the best overall model for the selection of high-quality neural architectures within the search spaces of NAS-Bench-101 and NAS-Bench-201.

In light of these promising findings, it is crucial to clarify that MbML-NAS, as a MtL method, fundamentally relies on using prior outcomes from pre-trained ConvNets to construct the meta-dataset that guides NAS. This reliance on prior data aligns with the principles of Meta-Learning, where leveraging prior knowledge facilitates the ability to generalize to novel tasks and datasets (BRAZDIL *et al.*, 2008). Similar to other NAS methods, such as GenNAS-N, MbML-NAS embraces this approach to navigate diverse search spaces and datasets effectively. While achieving superior performance is certainly a desired outcome, the foremost objective of this proposal lies not solely in surpassing existing models but in prioritizing simplicity and interpretability. By opting for a smaller number of input neural architectures for training, the aim is to furnish a pragmatic, efficient, and easy-to-implement method. This emphasis on simplicity ensures that both researchers and practitioners can readily embrace and comprehend the proposal. Furthermore, it is noteworthy that MbML-NAS demonstrates the advantage of employing fewer training samples compared to other NAS baselines. For instance, GenNAS-N relies on a set of 500 neural architectures from NAS-Bench-101 for its pre-training phase. In contrast, MbML-NAS requires only 172 neural architectures for training in its most resource-efficient configuration. This reduction in sample utilization not only streamlines computational requirements for training but also alleviates the burden associated with data collection and processing.

5.4.3 Interpretability Analysis

This section presents an interpretability experiment concerting meta-predictors and meta-features used by MbML-NAS in order to gain deeper insights into the influence of meta-information on the learning models. The adopted tools for inspecting and analyzing interpretability were chosen based on their fast computation and for being as simple and effective as possible. These selected methods, including the Mean Decrease in Impurity (MDI) and Feature Coefficient Importance, are readily obtainable from the trained tree-based and linear meta-predictors, respectively. Thus, they offer valuable insights into the models' interpretability without requiring additional procedures or reliance on third-party tools. Moreover, it is worth noting that these methods extend beyond the scope of MbML-NAS and can be applied to a diverse range of models. The analysis of Feature Importance is adequate for any tree-based or linear model, and, at the same time, the Permutation Importance, as a model-agnostic tool, can be employed with virtually any ML model. This flexibility underscores the versatility and broader applicability of the interpretability assessment framework employed in this experiment.

Starting from Figure 28, bar plots and box plots representations of Feature and Permutation Importances are shown for the Random Forest, the best overall meta-predictor in MbML-NAS. In order to consolidate the findings regarding the impact of all meta-features on the performance of meta-predictors, a summarization based on the Permutation Importance analysis is shown in Table 20. Additionally, Spearman correlations between meta-features and meta-targets are seen in Table 18, which also contributes to understanding the real impact of those meta-features on the prediction and selection process of architectures. Nonetheless, before entering into the specific details of interpretable regression models and meta-features, it is essential to clarify and establish the concept of interpretability adopted in this proposal.

Interpretability in the field of ML often revolves around the capacity to comprehend and elucidate a model's decision-making process (MOLNAR, 2020). Nevertheless, it is noteworthy that there is no universal agreement on the definition of interpretability, and its significance may vary across different domains and scenarios (RIBEIRO; SINGH; GUESTRIN, 2016; BAEHRENS *et al.*, 2010). Some scholars assert that interpretability offers a qualitative understanding of the relationship between input features and outputs (LIPTON, 2018). Others underscore the importance of aligning interpretability with the user's cognitive limitations, emphasizing the need for ease of understanding (MOLNAR, 2020). Consequently, whether a linear model or decision tree is deemed interpretable hinges on both the number of influential features contributing to predictions and the complexity of those features in terms of human comprehension (BAEHRENS *et al.*, 2010). This proposal considers meta-features to be interpretable because they hold clear and discernible meanings within the context of DNNs and NAS, making them accessible to individuals with at least a basic understanding of neural architectures. Additionally, we consider our models interpretable because their decision-making processes can

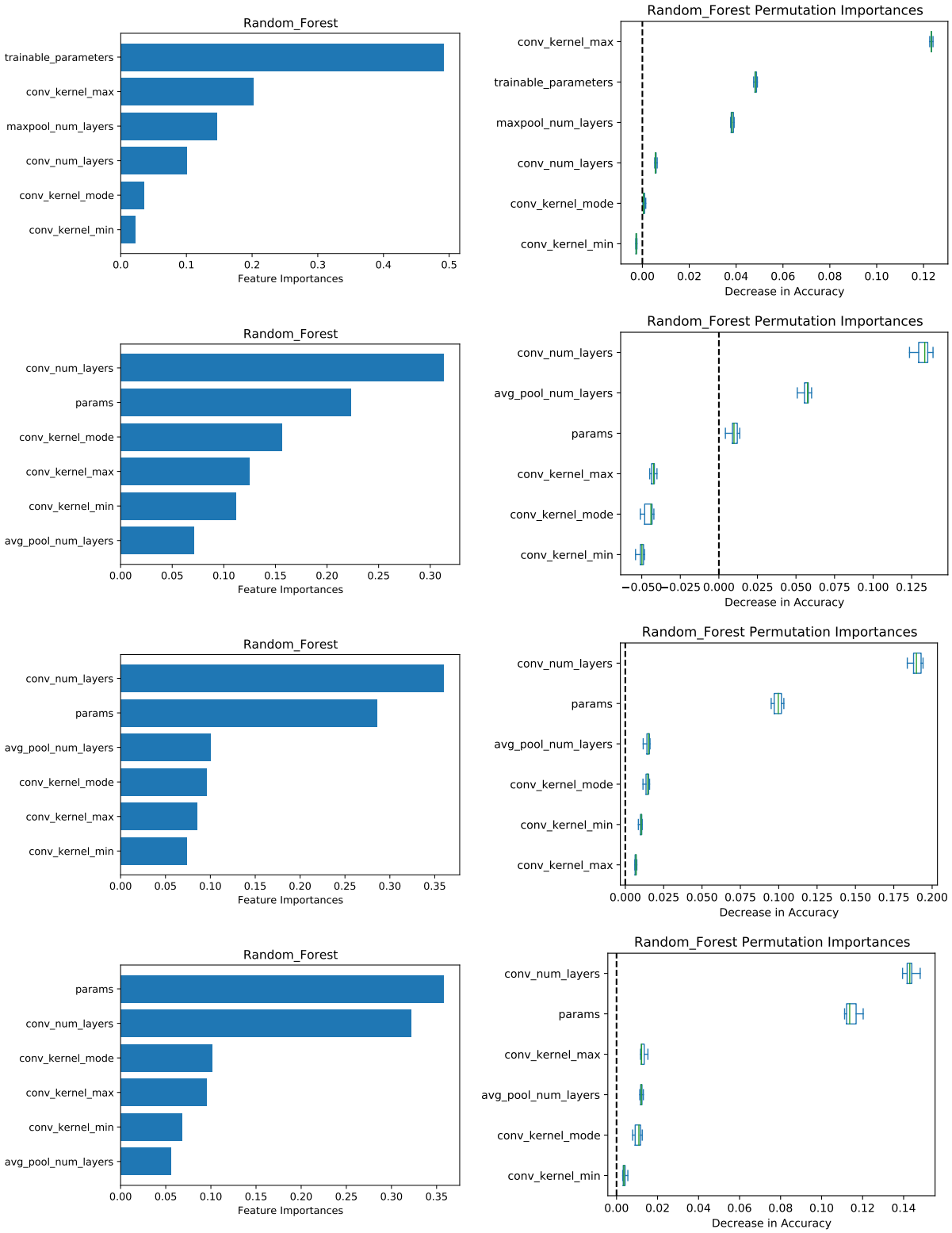


Figure 28 – Feature and Permutation Importance of the Random Forest on NAS-Bench-101 (CIFAR-10) (Top row), NAS-Bench-201 (CIFAR-10) (Second row), NAS-Bench-201 (CIFAR-100) (Third row), and NAS-Bench-201 (ImageNet16-120) (Bottom row) (PEREIRA *et al.*, 2023).

be inspected by examining coefficient values in the case of linear models, and one can trace the hierarchical decision flow of tree-based models. Furthermore, such models do not compromise human comprehension since a minimal input consisting of six meta-features is used.

Figure 28 presents box plot distributions based on ten runs of the Permutation Importance analysis using the overall best model, Random Forest. These plots illustrate the resulting percentage reduction in accuracy from the permutation of individual meta-features. The vertical demarcation lines within these plots serve the purpose of discerning the impact of each meta-feature on the model's performance, with those positioned to the left indicating no observable impact. It is important to emphasize that the influence of a meta-feature is intricately linked to the specific model and its learning process, and the insignificance of a particular meta-feature in one context should not be hastily extrapolated to other tasks or models. In the context of NAS-Bench-101 with CIFAR-10, the Impurity-based Feature Importance ranked *trainable_parameters* as the most influential meta-feature, closely followed by *conv_kernel_max* while designating *conv_kernel_min* as the least important. Nevertheless, the Permutation Importance analysis revealed a contrasting ranking for these two vital meta-features, with *conv_kernel_max* emerging at the top spot and inducing an approximate 12% reduction in accuracy. On the other hand, *conv_kernel_min* was found to be irrelevant in the Random Forest decision-making process.

On the NAS-Bench-201 benchmark using the CIFAR-10 dataset, both Feature Importance and Permutation Importance analyses consistently identified *conv_num_layers* as the most relevant meta-feature, leading to an accuracy reduction of approximately 12.5%. Conversely, three other meta-features, named *conv_kernel_min*, *conv_kernel_mode*, and *conv_kernel_max*, were deemed irrelevant in terms of performance impact. Similar patterns emerged when considering the NAS-Bench-201 benchmark with CIFAR-100, where *conv_num_layers* emerged as the most important meta-feature, inducing a nearly 20% accuracy decline. Except for these two standout meta-features, the remaining exhibited minimal impact. When considering NAS-Bench-201 with ImageNet16-120, *conv_num_layers* was the most impactful, closely followed by *params*, resulting in an approximately 14% and 12% of performance decrease, respectively. Therefore, these findings highlighted a consistent and slight variance in meta-feature importance across diverse datasets and benchmarks. In most cases, the impact of *conv_num_layers* and *params* remained prominent, while *conv_kernel_min* was consistently ranked as the least influential meta-feature. To better understand the impact of all meta-features on the performance of MbML-NAS meta-predictors, a comprehensive summary is shown in Table 20.

Results shown in Table 20 reveal significant disparities between meta-features, with some consistently emerging as the most important and never ranking as the least influential. Notably, *parameters* (comprising *trainable_parameters* and *params*) appeared as the top relevant twice, while *conv_num_layers* took the lead a remarkable seventeen times, firmly establishing itself as the predominant meta-feature. In contrast, *conv_kernel_min* was consistently found to be the

Table 17 – Meta-features relevance in twenty-eight use cases from the Permutation Importance analysis. Columns represent the number of times a meta-feature was found to have the most or least impact on model performance, respectively (PEREIRA *et al.*, 2023).

Meta-Feature	Most relevant	Least relevant
parameters	2	0
conv_num_layers	17	0
conv_kernel_min	0	13
conv_kernel_max	6	3
conv_kernel_mode	1	6
pool_num_layers	2	6

least impactful, featuring in this role thirteen times but never rising to the most relevant. Moreover, some meta-features exhibited mixed performance outcomes. For instance, *conv_kernel_max* was at the top spot six times but occupied three times the least influential position. Similarly, *conv_kernel_mode* was deemed the most important on one occasion but considered six times the less relevant. Meanwhile, *pool_num_layers* (encompassing *maxpool_num_layers* or *avg_pool_num_layers*) showcased dual roles by emerging as the most impactful twice and on six occasions ranking as the least influential. Therefore, these findings provide valuable information for future model design by indicating the differing impact of various meta-features on predictive performance. In the following, a Spearman correlation between meta-features and the meta-target is presented in Table 18 to complement the interpretability analysis.

Table 18 – Spearman correlations between meta-features and meta-target (architectures' validation accuracy) on NAS-Bench-101 and NAS-Bench-201 datasets. † is the CIFAR-10 from NAS-Bench-101, while ‡ represents the CIFAR-10 from NAS-Bench-201 (PEREIRA *et al.*, 2023).

Meta-Feature	CIFAR-10†	CIFAR-10‡	CIFAR-100	ImageNet16-120
parameters	0.44	0.71	0.73	0.68
conv_num_layers	0.42	0.70	0.74	0.71
conv_kernel_min	0.02	0.06	0.04	0.03
conv_kernel_max	0.52	0.53	0.52	0.51
conv_kernel_mode	0.24	0.28	0.27	0.23
pool_num_layers	-0.46	-0.49	-0.44	-0.43

Table 18 provides an overview of Spearman correlations among extracted meta-features, encapsulating neural cells' statistics, and the meta-target representing the validation accuracy of candidate architectures. It is essential to emphasize that this is a post-hoc analysis, meaning that no information derived from such correlations influenced model training or the inference/selection process. Notably, *parameters*, *conv_num_layers*, and *conv_kernel_max* consistently exhibit the strongest correlations with the meta-target across various datasets and benchmarks. On the other end of the spectrum, *conv_kernel_min* and *conv_kernel_mode* consistently emerged as the

least strongly correlated meta-features with validation accuracy. Another intriguing observation is noted when analyzing the correlation relationship of $pool_num_layers$, which showed a unique negative correlation with validation accuracy. Consequently, this suggests that architectures with fewer pooling layers tend to produce higher predictive performance, offering a distinctive perspective on the relationship between this type of layer and model outcomes.

In a broader context, such correlation analyses unveiled a discernible non-linear monotonic relationship between meta-features and validation accuracies. The most influential meta-features exhibited moderate to high correlations with the target performance, where the top-ranking meta-feature *conv_kernel_max* attained a correlation coefficient of 0.52 in NAS-Bench-101, while *parameters* achieved a substantial 0.71 correlation in NAS-Bench-201 with CIFAR-10. Additionally, *conv_num_layers* showed the highest correlation strength of 0.74 in NAS-Bench-201 with CIFAR-100, as well as 0.71 in NAS-Bench-201 with ImageNet16-120. These findings offer compelling evidence that these meta-features hold the potential to empower meta-predictors to capture the intricate non-linearity inherent in the data. Additionally, it can facilitate more accurate predictions of architectural performance, a conclusion consistently substantiated by other experiments within this study. Furthermore, such correlations align with empirical observations in the community (HUTTER; KOTTHOFF; VANSCHOREN, 2019), where architectures characterized by an abundance of convolutional layers and trainable parameters, fewer pooling layers, and smaller kernel sizes have often demonstrated superior performance.

5.5 Chapter Remarks

This chapter presented a novel and simplified Prediction-based method called Model-Based Meta-Learning for Neural Architecture Search (MbML-NAS). By leveraging meta-knowledge from a small set of training examples, MbML-NAS can effectively select near-optimal ConvNets using traditional regression models and only six interpretable meta-features directly extracted from neural architectures. Among these regressors employed as meta-predictors are the linear models, Linear Regression, Stochastic Gradient Descent, and Bayesian Ridge, and the non-linear models, C4.5 Decision Tree, Random Forest, AdaBoost, and Gradient Boosting. Of the statistical measures used by meta-predictors to approximate the actual performance of candidate architectures, they were simple statistics collected from the neural layers, including the number of convolutional and pooling layers, minimum, maximum, and most common kernel size, in addition to the total number of trainable parameters.

The proposal of MbML-NAS was well concerned with standardization and reproducibility. For this reason, extensive experimentation was conducted to ensure rigorous evaluation, which included standalone performance analysis, meta-predictive error comparisons, and interpretability inspections, with each experiment being repeated multiple times to increase reliability. All relevant details were also reported to guarantee reproducibility, including data splitting procedures, hyperparameter tuning, implementation specifications, evaluation metrics, hardware specs, and the official code. Consequently, the presented results inspire strong confidence in the robustness and high-quality evaluation of MbML-NAS.

As mentioned in Section 5.4, each experiment was designed to answer one of this study's fourth research questions. The first question, "Can simple meta-features effectively encode neural architectures so that traditional regression models can accurately predict their performances?" and the second, "Is it feasible to discover high-performing architectures through Meta-Learning with limited training examples?", were addressed individually through the analysis of predictive performance comparisons seen in Figure 27 and in Tables 13, 14, 15, and 16. As discussed, MbML-NAS generated results with predictive performance comparable to the state-of-the-art using at least 172 examples in its cheapest configuration, representing 0.04% and 1.1% of NAS-Bench-101 and NAS-Bench-201 search spaces. In the most expansive configuration, using 860 examples and representing only 0.2% and 5.5% of the original search spaces, MbML-NAS generated even better results, showing that it can take advantage of more meta-data, if available.

By using the same meta-features for the two most popular NAS search spaces and the three most used image classification datasets, it is possible to safely state that MbML-NAS provides a reasonable generalization across search spaces and datasets, thus answering the third research question: "Can simple meta-features lead to generalization across diverse search spaces and datasets?". Finally, we were able to answer the fourth and final research question, "Is it possible to effectively use interpretable meta-features and meta-predictors to get insights into

the intrinsic characteristics of neural architecture search spaces?", through the analysis of meta-feature correlations in Tables 18 and 20, and Feature Importances and Permutation Importances in Figure 28. These analyses provided insights into how meta-features helped meta-predictors accomplish the tasks, also highlighting the most important meta-features to consider when trying to achieve the best-performing architectures. Such analysis is valid not only for this study but also for future work that may use meta-learning to predict or recommend neural architectures.

These rigorous experiments with state-of-the-art methods highlight MbML-NAS's ability to discover superior architectures with minimal data. Complementary to the predictive performance discussions, interpretability analyses elucidated the MbML-NAS decision-making process, demonstrating the generalization potential of simple meta-features. Despite adopting a straightforward pipeline with inspectable models and interpretable meta-features, MbML-NAS showed a slight performance disadvantage compared to baselines that prioritize predictive performance. However, it still achieves strong performance results when comparable to more complex and uninterpretable models, such as the Neural Predictor baseline. Extensive experimentation validates MbML-NAS as a well-balanced solution, achieving a favorable compromise between predictive performance, data efficiency, and interpretability. Furthermore, it offers a significantly simpler approach than other NAS baselines, showing its potential as an effective and practical method for NAS research. The following chapter takes a step forward toward the goal of optimizing the NAS framework by presenting a novel method suitable to the more complex task of Image Segmentation, where the focus is to find good architectures using little label data.

CHAPTER

6

ACTIVE DIFFERENTIABLE NETWORK TOPOLOGY SEARCH

This chapter presents a novel method called Active Differentiable Network Topology Search (Active-DiNTS) that addresses the challenge of finding architectures with high predictive performance using limited data. After exploring the conventional tasks of Classification and Regression with simplified models, this chapter takes a step further in exploring Meta-Learning (MtL) to optimize NAS by targeting the more complex task of Image Segmentation using meta-knowledge of prior models, performances, and task properties. Unlike previous chapters that addressed more theoretical applications, such as recommending algorithms for well-known optimization problems and neural architectures for natural image classification, this chapter focuses on the more applied problem of Segmenting Medical Images. Given that the medical field suffers from data scarcity and high-cost labeling, Active-DiNTS introduces an Active Learning (AL) framework in order to maximize the performance of architectures while minimizing labeled data usage. Differing from other NAS methods, the proposal also uses a highly flexible Topology Search Space explored with Gradient-based optimization. By jointly using an AL approach to iteratively label the most impactful examples with a flexible and diverse topological search space, Active-DiNTS can optimize the search and learning process while encouraging the search for architectures with good generalization to fewer labeled examples.

In summary, this chapter presents the following contributions:

- A novel One-shot NAS method that uses an Active Learning framework to reduce search time by up to 27 times and training data by up to 5 times while finding state-of-the-art architectures for Medical Imaging Segmentation;
- Exploration of a Pool-based Sampling Scenario with multiple setups to assess the impact of active labeling on adjusting the weights and topology of candidate architectures;

- Implementation and evaluation of uncertainty functions, Entropy, Variance, and Standard Deviation, used to measure example importance in the selection process for active labeling
- An in-depth analysis of the proposal and baselines encompassing both predictive performance and model efficiency comparisons, such as the number of parameters, FLOPs, memory and hardware usage, and search time, in addition to qualitative analysis of segmentation masks and searched cell complexity.

The remaining content of this chapter is organized as follows: Section 6.1 presents the related literature. Section 6.2 describes the research contribution in detail. Section 6.3 details the experimental setup for reproducible purposes. Section 6.4 discusses the main results. Lastly, the final considerations for the chapter are presented in Section 6.5.

6.1 Related Work

The field of Medical Imaging Segmentation has seen a significant integration of NAS techniques in response to the pressing demand for automated design. Even outside the NAS domain, several notable approaches have emerged as solutions for automated, efficient segmentation. Each of these solutions offers unique contributions to address the complex challenges associated with anatomical structure delineation, both in 2D and 3D imaging modalities. These methods can be broadly categorized into various architectural innovations such as NAS approaches, hand-crafted U-Net-like models, techniques for incorporating multi-planar 2D views, 3D models, supervised, and semi-supervised segmentation, among others.

As with mainstream Image Segmentation, many of the major contributions in the Medical Imaging field involve the use of U-Net architectures. In Perslev *et al.* (2019) is proposed Multi-Planar U-Net (MPUnet) [1], an architecture capable of accurately segmenting anatomical structures in medical images across a diverse range of image modalities without task-specific modifications. The authors combine a fixed U-Net model topology with multi-planar augmentation, allowing the architecture to consider multiple 2D views during training and inference. By doing this, MPUnet can make use of a single 2D architecture to perform 3D volume segmentation while maintaining computational efficiency. Consequently, the model's robustness and ability to outperform 3D ConvNets on various tasks highlight its potential as an open-source alternative for medical image segmentation, particularly in settings with limited computational resources. In Isensee *et al.* (2019) is proposed nnU-Net, a versatile and efficient framework that offers a rich set of U-Net-based architectures optimized for a wide range of 2D and 3D medical imaging segmentation. By adaptively configuring architectures based on specific characteristics of the input data, such as image resolution and anatomical structures, nnU-Net significantly reduces the need for manual intervention in the network setup process and reliance on domain-specific expertise, accelerating

[1] CerebriuDIKU as referred in He *et al.* (2021)

research and clinical applications within the field. nnU-Net accomplishes this by effectively capturing dataset and pipeline characteristics, generating a pipeline fingerprint that leads to high-quality segmentation. By dividing parameters into blueprint, inferred, and empirical categories, nnU-Net automatically handles hyperparameter settings and network configurations.

There are also models that deal with 3D segmentation directly, sharing a common emphasis on improving segmentation for challenging scenarios. In Çiçek *et al.* (2016), a 3D U-Net is used for addressing sparse annotation challenges in 3D volumetric segmentation. The authors introduce a versatile architecture applicable for both semi-automated and fully-automated scenarios, where users provide sparse annotations on specific slices in the semi-automated setup so the network can generate detailed 3D segmentations. Additionally, in the fully automated setting, the network is trained on sparsely annotated datasets, enabling generalization to new volumetric images. This 3D U-Net employs on-the-fly elastic deformations for data augmentation and can be trained from scratch without relying on pre-trained models, offering a promising solution to alleviate the volumetric data annotating burden. In Xia *et al.* (2020) is introduced Uncertainty-aware Multi-view Co-Training (UMCT) [2], an approach tailored for 3D semi-supervised medical image segmentation. UMCT extends dual-view and deep co-training methods from 2D to multi-view 3D training, enhancing robustness by using asymmetrical 3D kernels initialized from 2D pre-trained models. The authors also introduce an Uncertainty-weighted Label Fusion Module (ULF) for improved multi-view predictions. UMCT addresses both semi-supervised and Unsupervised Domain Adaptation (UDA) in volumetric medical image segmentation, which helps promote consensus among different views on unlabeled data. By employing Bayesian Deep Networks with the ULF for pseudo-label generation on unlabeled data, UMCT is adaptable to UDA scenarios even without source domain data. In Oktay *et al.* (2018), an Attention U-Net is proposed for organ localization and segmentation without relying on external modules. The Attention U-Net shows great potential for improving the localization of tissues and organs, especially for smaller structures such as the pancreas. This model can implicitly learn to suppress irrelevant regions in the input image while highlighting important features for the current task. The introduction of an Attention Gate in medical images is notable as they autonomously emphasize target structures of various sizes and shapes. Furthermore, such models show high adaptability for integration with standard ConvNets and U-Net models without considerable computational overhead, while increasing segmentation sensitivity and precision.

The remaining class of models covered encompasses NAS approaches, whose models share the goal of optimizing architectures to increase efficiency in 3D segmentation tasks. Kim *et al.* (2019) present Scalable NAS (SCNAS), a comprehensive NAS framework tailored for 3D segmentation and designed to automatically optimize architectures capable of handling high-resolution 3D medical images. SCNAS systematically explores architectural configurations in a discrete search space, such as neural connections and layer operations, using encoder and decoder

[2] NVDLMED as referred in He *et al.* (2021)

components. To address the complexities of searching through a vast and discrete architectural space while maintaining scalability, the authors introduce an innovative stochastic sampling algorithm based on continuous relaxation. This allows SCNAS to perform efficient gradient-based optimization while reducing computational demands. In Yu *et al.* (2020) is proposed Coarse-to-Fine Neural Architecture Search (C2FNAS), a two-stage NAS method designed for 3D image segmentation that tackles the inconsistency challenge between search and deployment stages, a common issue in NAS algorithms. C2FNAS incorporates a coarse-to-fine strategy to effectively explore the search space of architectures, dividing it into two stages: a coarse stage, which focuses on macro-level topologies exploration, and a fine stage, focused on optimal operations selection within individual neural cells. To effectively manage memory limitations, C2FNAS employs a single-path, One-shot NAS approach with uniform sampling during the fine stage. Consequently, C2FNAS outperforms prevalent 3D models in terms of performance while simultaneously maintaining a relatively compact model size.

In He *et al.* (2021) is proposed Differentiable Network Topology Search (DiNTS), the main baseline for Active-DiNTS and a method to search for multi-resolution and multi-path network topologies for 3D medical segmentation. Through a feature node conversion technique, DiNTS focuses on both macro and micro search of operations and connection patterns in a cell-based space, generating multiple topologies that span various architecture types, including but not limited to U-Nets. By efficiently exploring flexible network topologies through continuous relaxation and discretization techniques, DiNTS significantly reduces search time while managing GPU memory within a budget constraint. The authors also introduce a topology loss to bridge the gap between continuous and discrete models, which is crucial in memory-constrained scenarios and helps optimize network connection for feasibility and effectiveness.

6.2 Active Differentiable NAS with Pool-based Sampling

This section presents a novel method named Active-Differentiable Network Topology Search (Active-DiNTS) that uses a Pool-based Sampling strategy and Uncertainty measures to find promising ConvNet-like architectures for 3D Image Segmentation. Inspired by DiNTS (HE *et al.*, 2021), Active-DiNTS adapts the former method to include an Active Learning framework on top of the bi-level optimization process that updates both the macro and micro structures of architectures in order to reduce costs such as data usage and search time. As seen in Section 6.1, related works contribute to improving the efficiency and accuracy of 3D segmentation while addressing specific challenges related to data availability, robustness, and sensitivity. However, such methods still struggle to achieve good performance with limited data. Therefore, Active-DiNTS is proposed with the goal of learning from fewer examples using active sampling strategies to minimize the use of labeled data while maximizing predictive performance in the Medical Image Segmentation task. An overview of Active-DiNTS' pipeline is seen in Figure 29.

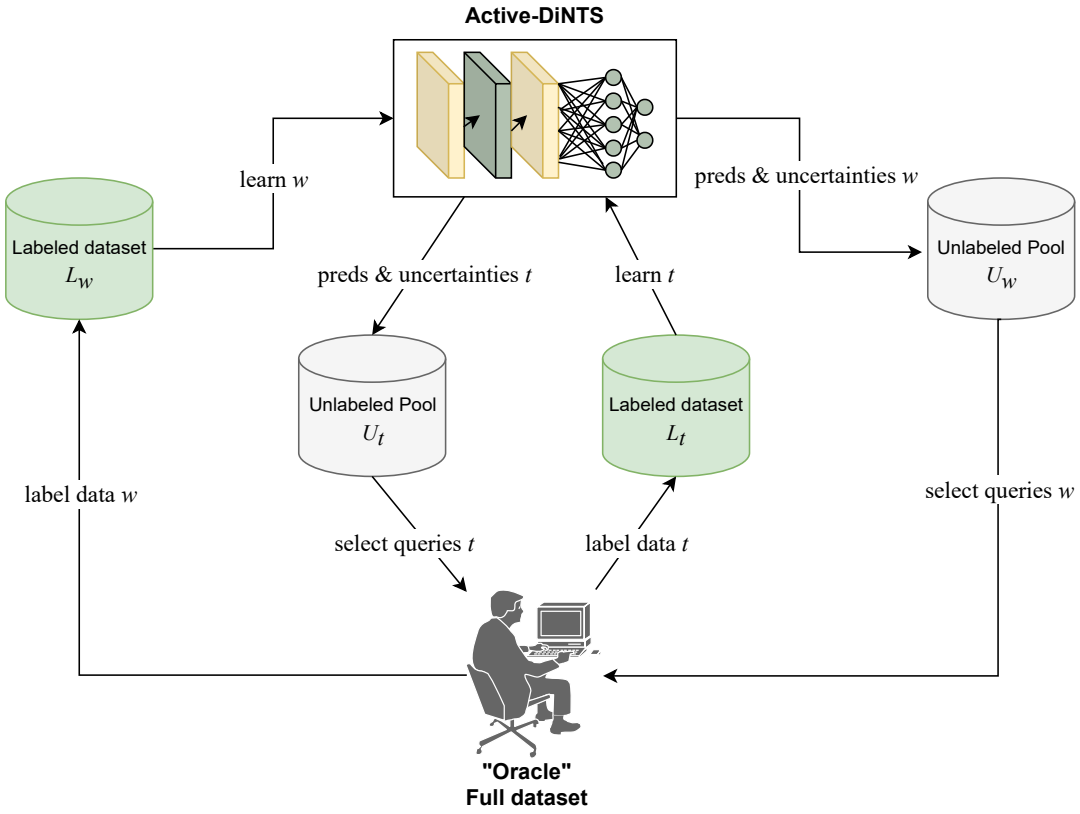


Figure 29 – Active-DiNTS overview.

Figure 29 shows the Pool-based Sampling Scenario adopted by Active-DiNTS. This conventional AL framework starts with an untrained Active-DiNTS only being aware of the number of classes. As input, Active-DiNTS receives an Unlabeled Pool U_W dataset and produces predictions, which are then fed to an uncertainty estimation function that measures their confidence. The most uncertain pool samples are then given to the Oracle, an entity that has knowledge of the original labels. In real-life scenarios, this Oracle is a domain expert, such as a physician who will label medical images with their pathologies or, as in the case of Active-DiNTS application, segment tumors presented in unlabeled images. Here, this Oracle, represented by the complete dataset, labels these samples and creates a new Labeled dataset L_W, simultaneously removing such samples from U_W. This entire process constitutes a query, and with each query, labeled examples composing L_W are used to update Active-DiNTS' weights. This iterative process continues until the Unlabeled Pool U_W is exhausted or the expert user (Oracle) is satisfied with the sample in a practical scenario. Additionally, this framework contains a second interconnected AL loop that updates Active-DiNTS' topology. The inner loop also follows a traditional AL procedure, emptying an Unlabeled Pool U_T while feeding a Labeled dataset L_T using Pool-based Sampling and Uncertainty functions. However, this internal loop occurs every training epoch after adjusting Active-DiNTS' weights with L_W, and only one representative sample at a time is selected from U_T to update L_T. As there will naturally be more removals from U_T than from U_W, U_T is restarted every time it is emptied, in case the number of queries, and therefore of U_W, has

not been emptied too. Therefore, Active-DiNTS first learns the architectural weights with L_W for N training epochs using Q queries in U_W and, subsequently, learns the architectural topology with L_T fed from U_T. Next, Algorithm 3 details the Active-DiNTS process.

Algorithm 3: Active-DiNTS Procedure.

Input: Search Space S, Active-DiNTS $Model$, Oracle dataset O, Labeled dataset L_W, Unlabeled Pool U_W, Labeled dataset L_T, Unlabeled Pool U_T, Query Setup Q_S, Query Function Q_F, Nº of Samples per Query S_Q, Nº of training Epochs E

Output: Final discrete and trained Active-DiNTS $Model$

```
1  Model ← init(S)
2  L_W ← ∅
3  L_T ← ∅
4  while U_W is not empty do
5      predictions ← inference(Model, U_W)
6      if Q_S is on topology only then
7          Q_F ← Random
8      samples ← selection(predictions, Q_F, S_Q)
9      samples ← labeling(samples, O)
10     L_W, U_W ← update_datasets(samples)
11     foreach epoch in E do
12         train(Meta, L_W)
13     end
14     if U_T is empty then
15         U_T ← Refill
16     predictions ← inference(Model, U_T)
17     if Q_S is on weights only then
18         Q_F ← Random
19     samples ← selection(predictions, Q_F, S_Q)
20     samples ← labeling(samples, O)
21     L_T, U_T ← update_datasets(samples)
22     sample ← subsample(L_T, B)
23     train(Meta, sample)
24 end
```

Algorithm 3 has as inputs a cell-based Search Space S; an Active-DiNTS $Model$ instance sampled from S and composed of a flexible topology; an Oracle dataset O containing knowledge for labeling examples; labeled and pool empty sets to be filled in the AL process, L_W, L_T, U_W, and U_T; a query function Q_F that measures examples uncertainties; the number of samples per query S_Q that dictates AL updates pace and duration; and the number of training epochs E. The process starts by instantiating both Active-DiNTS $Model$ and empty datasets L_W and L_T in lines 1 to 3. From lines 4 to 23, the complete AL training process occurs, which continues until the Unlabeled Pool U_W is empty. First, the untrained $Model$ predicts samples from U_W in line 5, which are used to select the most uncertain examples by the selection function in line

8. This query function Q_F returns a number of samples according to S_Q, which are submitted to the Oracle O labeling function in line 9. Once labeled, the L_W dataset is updated with new samples that are also removed from the gradually emptied U_W. Finally, $Model$ is trained with the Labeled dataset L_W for E epochs in lines 11 to 13. It is important to highlight that such training becomes heavier as AL continues, as L_W is filled in each round of active labeling. However, the prediction and estimation of uncertainties in U_W go in the opposite direction as it becomes lighter as examples are labeled, causing the entire process to have a constant cost. After updating the weights, the second AL loop focuses on adjusting the network topology on lines 16 to 23. After being trained on L_W, $Model$ generates predictions for the Unlabeled Pool U_T on line 16, which are used to select and label samples by the Oracle O in lines 19 to 21, updating both L_T and U_T datasets. Finally, the topology is adjusted to take into account the new examples in line 23. However, this training occurs differently than training with L_W. Instead of the entire L_T set, only one sample is used to tune the topology, thus being an incremental bi-level optimization popularly used in NAS (LIU; SIMONYAN; YANG, 2018; LIU *et al.*, 2019; HE *et al.*, 2021).

6.2.1 Active Learning Setups and Query Functions

Active-DiNTS iteratively queries the labels of the most relevant examples from a large unlabeled dataset to find suitable architectures. Such a strategy, combined with bi-level NAS optimization, encourages the search for architectures that generalize well using fewer labeled data. Active-DiNTS tries different feedback loops for the AL setup: (i) Updating the weights of searched architectures with new labeled examples; (ii) Updating the topology of searched architectures with new labeled examples; and (iii) updating both weights and topology of searched architectures with new labeled examples. In combination with such configurations, choosing a function to measure example relevance is of utmost importance. The choice for a query function Q_T depends on the specific AL strategy employed, such as Uncertainty sampling, Query-by-committee, or Expected model change (LEITE; BRAZDIL, 2010). AL aims to select examples that provide the most valuable information to improve model performance while minimizing labeling efforts to achieve a desired level of generalization. In addition to the Random baseline, Active-DiNTS considers Uncertainty sampling to select representative examples, and, for the sake of simplicity and reproducibility, it adopts the measures of Entropy, Variance, and Standard Deviation, which are among the most simple and used approaches in the literature.

Entropy is a measure of disorder within a system, indicating the uncertainty associated with its content (DASGUPTA, 2011). In the context of this work, it quantifies the level of unpredictability of a segmentation mask generated by the learned model after receiving an MRI image as input. Without considering outliers, a higher entropy implies greater uncertainty, indicating that a data point (input image) is more informative and, thus, should be prioritized for labeling in order to improve model performance and reduce uncertainty. The Entropy for an AL

process can be defined according to Equation 6.1,

$$H(D) = -\sum_{y \in Y} P(y) \log_2 P(y) \tag{6.1}$$

where $H(D)$ denotes the entropy of the segmentation mask D with respect to the set of possible labels Y (tumor, or no tumor) for each input pixel, and $P(y)$ as the probability of a particular label y occurring in D.

Variance is a statistical measure that quantifies the spread or dispersion of data points (samples), or in Active-DiNTS case, dispersion within a segmentation mask, in relation to its mean (FRIEDMAN, 1937). In the context of AL, variance can be used to measure the variability of predictions caused by different training samples or pixels in an image, quantifying how sensitive the model is to changes in the training data. High variance suggests that the model's predictions can vary significantly with different labeled samples, making it valuable in AL for selecting diverse and informative data points to improve model robustness and generalization. The Variance can be obtained calculated as in Equation 6.2,

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2 \tag{6.2}$$

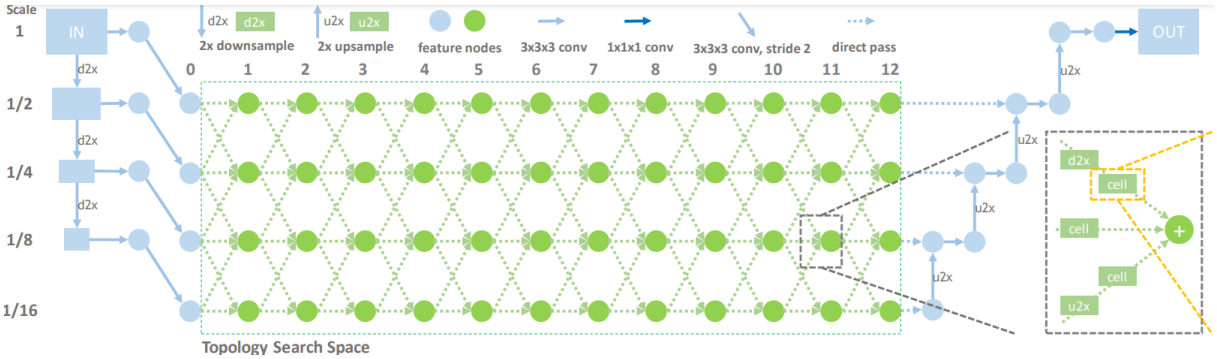
where σ^2 denotes the variance, N is the number of pixels in a segmentation mask, x_i represents each individual pixel, and μ is the mean of the whole mask.

Standard Deviation is also a statistical measure that assesses the amount of variation or dispersion in a dataset or segmentation mask, being the square root of Variance (MCLACHLAN, 2005). Standard Deviation provides a more interpretable measure of how data points deviate from the mean and it is expressed in the same units as the original data. In the context of AL, both Standard Deviation and Variance quantify the uncertainties based on the variability of predictions. However, while Variance quantifies overall prediction uncertainty, Standard Deviation measures individual prediction uncertainty since it can indicate how much individual data points deviate from the mean. Besides, Standard deviation is sensitive to extreme values, which could help to identify outlier examples to actively label. Its calculation is seen in Equation 6.3,

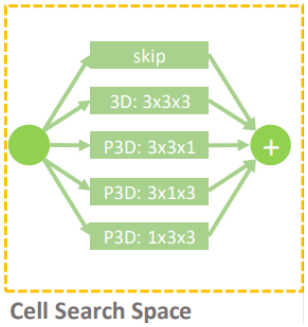
$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2} \tag{6.3}$$

where σ denotes the standard deviation, N is the number of pixels in a segmentation mask, x_i represents each individual pixel, and μ is the mean of the whole mask.

6.2.2 Active-DiNTS Search Space



(a) Topology space.



(b) Cell space.

Figure 30 – Active-DiNTS Search Space. Adapted from He *et al.* (2021).

Figure 30 presents the Search Space of DiNTS (HE *et al.*, 2021), a topology and cell-based space also used by Active-DiNTS. This space supports the search for multiple input image scales and complex multi-path topologies, encompassing L=12 layers and O=5 operation cells on the edges connected to nodes representing the resulting feature maps, dictating the feature flow from input to output and creating a network topology. The blue edges 2x downsample/upsample, 3x3x3 convolution, 1x1x1 convolution, 3x3x3 convolution with stride 2, and direct pass (skip connection) are the predefined operations that form the final architectures. The O=5 cell operations are defined on the green dashed edges, while the green nodes are feature maps. This unique Search Space differentiates itself by featuring fully connected edges that link adjacent resolutions, providing flexibility for exploring various input image scales and complex multi-path topologies. The feature spatial changes are performed by the upsample/downsample operations in the edges searched from the topology level. Together with the original image, four 3×3×3 3D convolutions with stride 2 can be used to create D=4 resolution features. In addition, for each edge containing a cell operation, a 2x downsample/upsample operation is used before the cell. Except for skip connection operators, a cell also includes ReLU activation and Instance Normalization before and after its operations. Thus, a feature node is the summary of all these pre-defined operations and the output features from each input edge.

This cell-based Search Space is defined by a set of basic operations where input and output feature maps preserve the same spatial resolution. Unlike the Search Spaces of popular methods such as DARTS (LIU; SIMONYAN; YANG, 2018) and Auto-Deeplab (LIU *et al.*, 2019), where cells and connections can be searched, but they are repeated at the topology level, the DiNTS space allows searching cells independently. In addition to relying on pseudo-3D convolution (P3D) (ZHU *et al.*, 2019), Active-DiNTS also searches for: (0) skip connections; (1) 3D convolution 3x3x3; (2) P3D 3x3x1 (3x3x1 convolution followed by 1x1x3 convolution); (3) P3D 3x1x3 (3x1x3 convolution followed by 1x3x1 convolution); and (4) P3D 1x3x3 (1x3x3 convolution followed by 3x1x1 convolution).

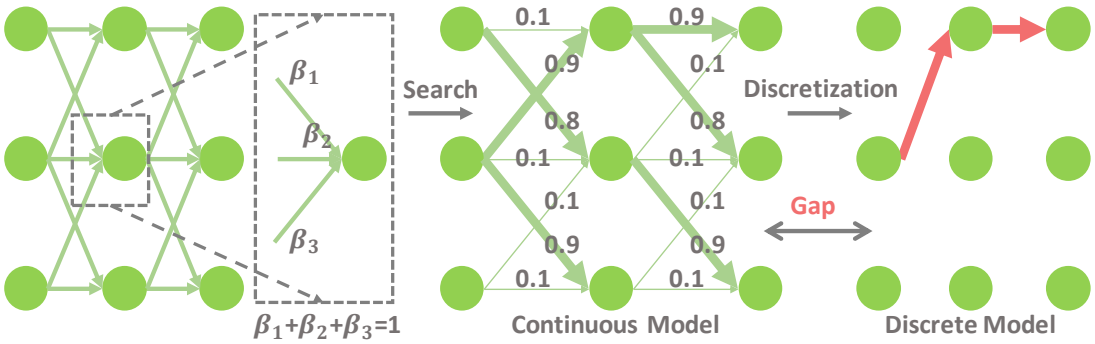


Figure 31 – Discretization gap problem between the feature flow of a searched continuous model and its final discrete version. Adapted from He *et al.* (2021).

In addition to providing an alternative solution to the limited search spaces with simple repetitive cells and inflexible topologies, Active-DiNTS space addresses the discretization gap problem as illustrated in Figure 31. Similar to other NAS One-shot methods, Active-DiNTS relaxes the discrete search space into a continuous representation so that O=5 candidate operations can be associated with trainable parameters (probabilities) optimized by gradient descent. After optimization, the continuous model is discretized, where operations with higher probabilities are selected while low-probability operations are discarded, even those that contribute significantly to the computation graph and subsequent layers. By directly discarding operations with non-zero probabilities, the feature maps information flow can be significantly disturbed, generating a gap between the feature flow of the continuous model and the final discrete model. This problem is known as the "Discretization gap", and is caused mainly by topology restrictions such as the single path restriction that limits possible architectural arrangements.

As seen in Figure 31, most One-shot models search for edges in a competitive and exclusive matter, assuming a single incoming path for each node. After discretization, a single path is maintained while other edges, even with high β probabilities, are discarded, meaning that the feature flow in the searched continuous architecture has a significant gap with the feature flow of the final discrete model. Although previous research has addressed the discretization problem, overcoming topology restrictions requires the search process to be aware of the discretization algorithm as it imposes topology restrictions. This restriction means that certain edges, even

with substantial probabilities in the continuous model, cannot be incorporated into the final discrete model. To avoid generating topologically infeasible edges that would eventually be removed in the discretization process, thus causing the discretization gap problem, a topology loss is introduced by DiNTS and also used in Active-DiNTS. This loss topology encourages connections with high probabilities to be viable and not discarded, reducing the gap between continuous and discrete models. Another interesting point about this loss is that it becomes more important when the parameter RAM factor is smaller. Using small values for this parameter makes the architecture more sparse and with fewer connections, thus being more likely to have topology unfeasibility. In experiments with Active-DiNTS, variations of the memory constraints [0.2, 0.5, 0.8] are tested, in which when increasing the parameter, the researched model is more dense in terms of connection and can achieve better performance at the cost of GPU memory.

6.3 Experimental Setup

This section presents the complete experimental setup for the training and validation of Active-DiNTS in its various configurations. To guarantee experiments' transparency and to facilitate reproducibility, details are given in the following. Section 6.3.1 introduces the employed medical image dataset. Section 6.3.2 lists the baseline methods used in the comparative analyses. Finally, Section 6.3.3 presents the evaluation methodology, which includes data preprocessing procedures, evaluation metrics, implementation details, software tools, and hardware specifications used in the training of Active-DiNTS and baselines.

6.3.1 Medical Image Segmentation Dataset

This section briefly presents the Task01_BrainTumour dataset used for the training and validation of Active-DiNTS. This employed Brain dataset is a subset extracted from the Medical Segmentation Decathlon (MSD) competition (ANTONELLI *et al.*, 2022), which comprises an extensive collection of biomedical data from a vital collaborative initiative of various research institutions. The MSD competition database encompasses an array of challenging medical imaging tasks, each involving intricate segmentation objectives such as organ delineation, tumor localization, and anatomical structure identification within medical images. Moreover, these datasets contain a diverse range of multi-modal data, including CT and MRI scans, spanning multiple clinical domains. The rich and complex nature of such datasets facilitates the development of precise and innovative segmentation algorithms, signifying its profound impact on the field of medical image analysis and healthcare applications in clinical practice (LI *et al.*, 2023).

In particular, the MSD subset identified as Task01_BrainTumour, which focuses on the challenging task of brain tumor segmentation, is the one relevant to Active-DiNTS. This training dataset comprises 484 multiparametric-Magnetic Resonance Images (mp-MRI) derived from patients diagnosed with glioblastoma or lower-grade glioma. It also includes diverse MRI sequences containing the modalities of Fluid-Attenuated Inversion Recovery (FLAIR), T1-weighted image (T1w), post-Gadolinium (Gd) contrast T1-weighted image (T1 wGd), and T2-weighted image (T2w). With such data, the main focus of the original challenge related to this dataset is the precise delineation of complex and heterogeneously located brain tumor sub-regions, where the corresponding Regions of Interest (ROIs) consist of Edema, Enhancing and Non-Enhancing Tumor sub-regions, reflecting its segmentation task complexity. Notably, Task01_BrainTumour is inspired by the Brain Tumor Segmentation (BraTS) challenges (MENZE *et al.*, 2014) and involves cases from the 2016 and 2017 editions. However, to prevent case mapping between the two challenges, filenames suffered alterations and shifting.

6.3.2 Baseline Methods

This section presents an overview of the baseline methods used for experiments and comparisons with the proposed Active-DiNTS. These baselines are categorized into manually defined U-Net architectures and NAS methods that automatically discover the architectures, with such methods covering a variety of approaches like semi-supervised learning, attention models, 2D, pseudo-3D, and pure 3D models. Details about these works have been previously discussed in Section 6.1, but are listed here for reference. Furthermore, setup variations of Active-DiNTS are detailed in this section, including different targets for active sampling, uncertainty function configurations, and hyperparameter adjustment as seen in Table 19.

Two examples of 2D or pseudo-3D methods adopted for comparisons are CerebriuDIKU (PERSLEV *et al.*, 2019), which combines a U-Net architecture with multi-planar 2D augmentations, and nnU-Net (ISENSEE *et al.*, 2019), an adaptable framework to 2D/3D segmentation with U-Net architectures for diverse medical imaging datasets. On the other hand, examples of 3D hand-crafted segmentation models include NVDLMED (XIA *et al.*, 2020), a semi-supervised learning model that uses multi-view data augmentation with dual-view co-training, 3D U-Net (ÇIÇEK *et al.*, 2016), an adaptation of U-Net designed for scenarios with sparse annotations, and Attention U-Net (OKTAY *et al.*, 2018), which uses an attention gate model for organ localization and identification. Among NAS-based models for 3D medical imaging segmentation, there is the main baseline for Active-DiNTS, the original DiNTS (HE *et al.*, 2021), which employs a search for multi-resolution and multi-path network topologies, SCNAS (KIM *et al.*, 2019), another efficient NAS framework featuring stochastic sampling algorithms for fast optimization, and C2FNAS (YU *et al.*, 2020), a two-stage NAS method that addresses the inconsistency between search and deployment stages efficiently.

As seen in Section 6.2, uncertainty functions used by Active-DiNTS include: (i) Entropy, which measures the randomness or disorder of segmentation masks generated from specific input examples; (ii) Variance, which measures the average squared difference of each voxel from the mean in a segmentation mask; and (iii) Standard Deviation, which measures data dispersion by quantifying the square root of Variance to select uncertainty samples. Along with each uncertainty function, associated hyperparameters are tuned during architecture search and training, as seen in Table 19. The Variance function from the MONAI package has a background variable that controls the inclusion of the spatial image background (segmentation mask) to calculate the metric. Besides, it is possible to return the spatial variation/uncertainty map instead of a single scalar value resulting from calculating the mean or sum of variations in the uncertainty map. Additionally, as input data are 4D MRI images (modalities, width, height, slices), it is necessary to define which dimension to apply the uncertainty functions. Applying it to Axis 0 or in_channels would calculate uncertainties regarding all image modalities, namely FLAIR, T1w, T1 wGd, and T2w. On the other hand, opting for Axis 3 would take the entire dimension of voxel slices into consideration, thus calculating uncertainties based on the 3D depth of MRI images.

Active-DiNTS applies different AL setups along with uncertainty functions, as Section 6.2 previously explored. This macro configuration of AL update targets consists of three possibilities: (i) Perform active labeling to optimize the weights of the neural architecture while updating the architecture topology with randomly chosen labels; (ii) Perform active labeling to optimize the architecture topology while updating the weights of the neural architecture with randomly chosen labeled examples; and, finally, (iii) exclude any random processes and update both the weights and the topology of the neural architecture according to the chosen uncertainty function. The remaining hyperparameters shown in Table 19 were tuned via Grid Search along with a 5-Fold Cross-validation adopted for training and testing. Some of these hyperparameter ranges, such as Epochs and Warm-up epochs, were defined after the He *et al.* (2021) setup.

Table 19 – Active-DiNTS hyperparameter space.

Hyperparameter	**Range**
Input channels	(FLAIR, T1w, T1 wGd, T2w)
Query function	(Random, Entropy, Variance, Std)
Axis query	(modalities, slices)
AL setup	(Weights, Topology, Weights and Topology)
AL samples	[2, 100]
Epochs	[1, 1430]
Warm-up epochs	[1, Epochs/2]

Active-DiNTS additional technical details follow the standards outlined in He *et al.* (2021). Unlike DiNTS, which searches and retrains its architecture, Active-DiNTS employs the same searched architecture on the Brain MSD dataset. After the search, the discretized DiNTS model is randomly initialized and retrained with doubled filter numbers and batch sizes. In contrast, Active-DiNTS uses the searched model as it is, following the same 5-fold Cross-validation split as DiNTS. To train the architecture weights W, SGD with a 0.9 momentum, weight decay of $4e-5$, and a Cosine Annealing learning rate scheduler is employed, while the topology parameters are trained with Adam, all initialized with Gaussian distribution. A RAM cost factor of 0.8, representing the most resource-intensive DiNTS version, is also adopted for Active-DiNTS. While DiNTS trains W for the first 1k warm-up and then 10k iterations without updating the architecture, Active-DiNTS only undergoes an initial warm-up of Epochs/2 iterations. The loss function for updating W combines Dice and Cross-entropy losses, while a complex loss is used to update the architecture topology. This loss includes a topology loss, a weighted scheme based on the number of epochs, an entropy loss to encourage topology parameter binarization, a RAM cost loss, and a scaled term. To generate the segmentation masks, a Sigmoid activation is applied to convert the convolution outputs into masks. Lastly, multi-channel pre-processing following the BraTS classes is applied to the Brain MSD dataset, with additional augmentations aligned with DiNTS procedures.

6.3.3 Evaluation Methodology

This section presents the evaluation methodology used to conduct experiments with Active-DiNTS and baselines. In alignment with He *et al.* (2021), all experiments followed a consistent approach for data splitting and validation. Active-DiNTS hyperparameters shown in Table 19 were tuned via Grid Search, and of the 484 public MRI images from the Brain MSD dataset, 388 are used for training and 96 for testing in 5-Fold Cross Validation, following the original dataset splitting patterns. From the training dataset, half of the examples are used to train the architecture weights and the remaining to adjust their topologies. Differing from He *et al.* (2021) that search and re-train on the source dataset, Active-DiNTS' final architecture is deployed after searching on the Brain MSD dataset without re-training. This procedure was implemented due to the achievement of satisfactory performance of Active-DiNTS even in the search phase, surpassing the baselines, including DiNTS, and because of the significant costs associated with re-training each one of the generated architectures.

To evaluate segmentation performance, Dice or the Dice-Sørensen Coefficient (DSC) was adopted, a common metric to evaluate performance in image segmentation tasks. In simple terms, Dice measures the similarity or overlap between predicted and ground truth segmentation masks, ranging from 0 to 1, where 1 indicates perfect overlap. Dice is computed as $DSC = \frac{2|X \cap Y|}{|X|+|Y|}$, where X is the set of pixels in the predicted segmentation mask, Y are the pixels in the ground truth segmentation mask, $|X|$ and $|Y|$ are the cardinality of X and Y, and $|X \cap Y|$ is the cardinality of sets X and Y intersection. Additionally, other indicators were used to measure the computational efficiency of segmentation models: The GPU memory required to search/train the models, architectural learnable parameters, and the Floating Point Operations (FLOPs) that represent the number of arithmetic operations performed during neural computations.

The complete experimental setup covering the entire Active-DiNTS implementation, the datasets used, and all associated documentation have been made openly available via GitHub [3]. The entire codebase was written in Python and only open-source libraries were used. In particular, the popular PyTorch (PASZKE *et al.*, 2019) was used to customize Active-DiNTS and to train its neural architecture. Scipy (VIRTANEN *et al.*, 2020), another widely used library for scientific computing, was employed to implement some of the active sampling strategies. Finally, MONAI (CARDOSO *et al.*, 2022), a PyTorch-based open-source framework for DNN in healthcare imaging, was adopted for manipulation, preprocessing, and evaluation with the medical imaging datasets used in this proposal. Regarding the hardware infrastructure, all training and validation were conducted on a single Linux server equipped with an Intel Xeon Silver 4215 CPU 2.50GHz 8-core processor and 126GB of RAM. Furthermore, only one GPU was used, the Tesla V100 with 32GB VRAM, used to search and train the architectures generated by Active-DiNTS, thus emphasizing the computational efficiency and accessibility of the proposal.

[3] https://github.com/geantrindade/Active-DiNTS

6.4 Results and Discussion

The experiments presented in this section were designed to answer the following research questions: (i) Can simple active sampling strategies used by Active-DiNTS present comparable or better performances than traditional segmentation models? (ii) Can Active-DiNTS with varying AL strategies offer improved computational efficiency when compared to other segmentation models? (iii) Can different AL strategies produce significantly divergent results?
The following sections address these questions through analysis and comparisons between Active-DiNTS and state-of-the-art segmentation models on Task01_BrainTumour from the Medical Segmentation Decathlon (MSD) competition dataset. As for the third-party methods included in the comparative analyses, original results have been cited for reference.

6.4.1 Active Sampling Analysis

This section shows a comparative analysis of AL setups and uncertainty sampling strategies used in Active-DiNTS. Figure 32 provides comparisons between uncertainty functions on the best overall AL setup of Weights and Topology. At the same time, Figure 33 presents comparisons between all AL setups considering an average of the uncertainty sampling strategy models. Additional results for all sampling functions and AL setups are included in Appendix B.

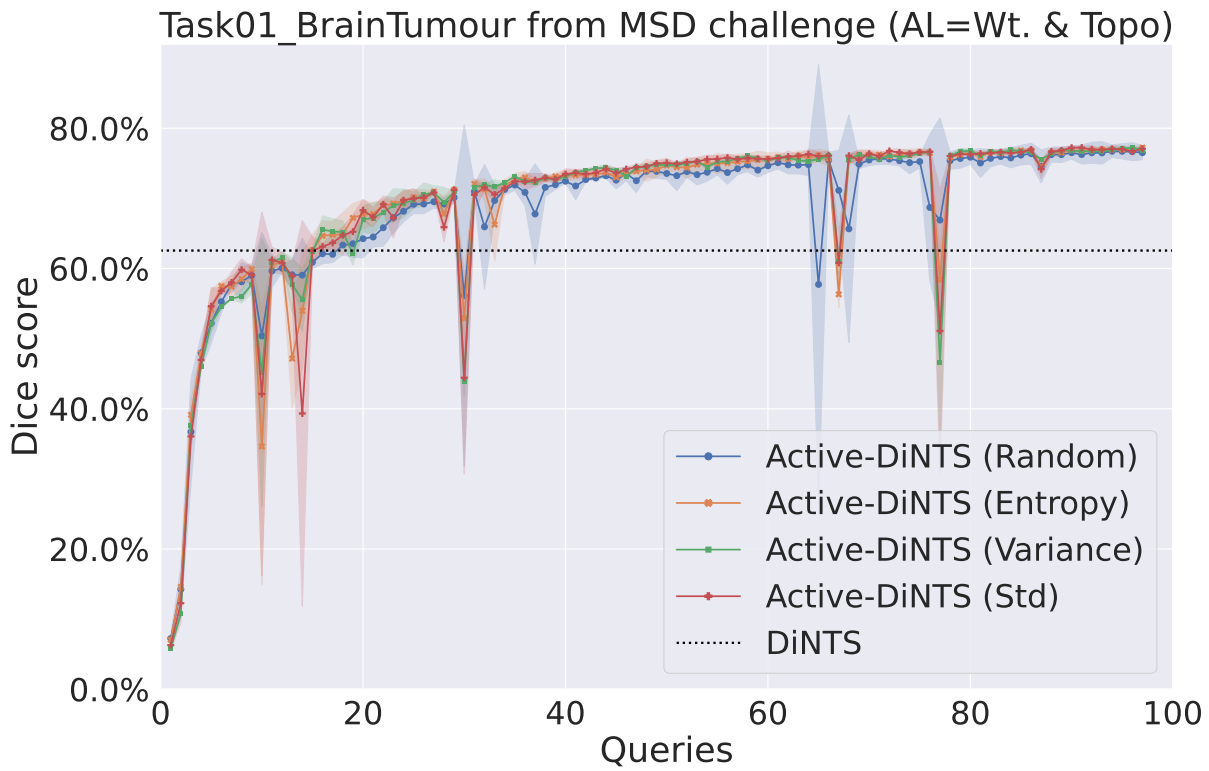


Figure 32 – Active-DiNTS learning curves over the number of Queries. Active Learning was used to select samples to train both the Weights (Wt.) and Topology (Topo) of neural architectures.

Figure 32 shows the consistent progress of Active-DiNTS with different uncertainty functions. At first glance, it is seen that Active-DiNTS outperforms the baseline DiNTS relatively early in the process. Entropy, Variance, and Std versions also outperform the Random baseline with some margin, whose performance is consistently inferior throughout the entire process. In the initial queries, uncertainty functions present comparable performance with a slight superiority of Variance and Entropy when Active-DiNTS surpasses DiNTS, but from query 20 onwards, Std shows a slight superiority. DiNTS, in particular, is outperformed with approximately 18 queries, representing less than 20% of the original dataset that it uses entirely. Another important observation concerns the abrupt performance drops of some queries. Such degradation suggests that these queries can disrupt the training data distribution and make the model substantially adjust its weights, which, due to the short training epochs after each query, is unable to adapt in time, causing the observed performance drops. When observing the Dice curve, there is a substantial increase in performance, especially in the initial queries 1 to 20. This is expected given the limited labeled training set, where each new uncertain sample significantly impacts the data distribution and, consequently, the model's learning. After query 20, performance continues to improve up to the $97th$ query, with consistent and substantial improvement up to about 60% of data usage, after which progress becomes more gradual. It is also worth mentioning that Random presents greater instability and variability of results throughout the process, while the uncertainty techniques display greater variability in initial queries.

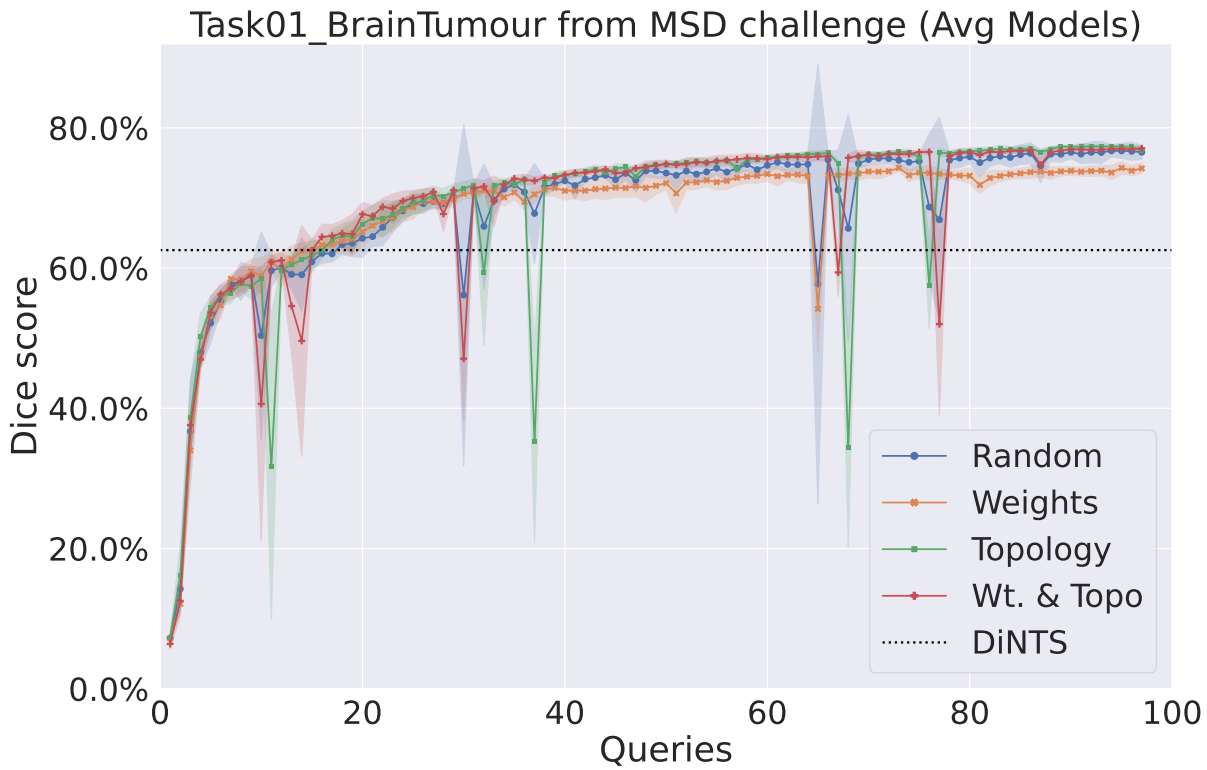


Figure 33 – Active-DiNTS learning curves over the number of Queries. Every curve is an Average (Avg) of each Active Learning setup (Weights, Topology, and Weights & Topology) over all the Query strategies models (Entropy, Variance, and Std).

Figure 33 shows the average outcomes of uncertainty functions for each AL setup, providing a more precise assessment of each configuration's impact. Similar to previous observations, distinct patterns emerge with initial high Dice scores followed by gradual improvements and Active-DiNTS surpassing DiNTS using less than 20% of the original data. On average, the Weights and Topology configuration consistently outperforms others, showcasing the highest Dice scores, especially in the crucial phase of initial queries. Additionally, the Topology-only configuration demonstrates favorable results comparable to the Weights and Topology setup at certain points. In contrast, the Weights-only configuration exhibits comparable performance initially but experiences a decline around the 40th query, maintaining this trend thereafter. Notably, despite Random appearing inferior to Weights after the mid-process, it displays higher variability with a considerable standard deviation, questioning its partial superiority. When assessing the overall results of all AL configurations, the most effective approaches involve actively training the topology, whether by adjusting the topology alone or simultaneously updating both topology and architecture weights. Merely adjusting weights with AL does not yield sustained improvement, but it shows comparable results to the best configurations up to approximately the 35th query. In summary, akin to uncertainty functions, all AL setups demonstrate state-of-the-art results utilizing only about 20% of the original labeled data.

6.4.2 Searched Architectures Analysis

This section presents a qualitative analysis of the architectural topologies found by Active-DiNTS and its uncertainty sampling strategies, along with a topology discovered by the original DiNTS baseline. Based on comparisons between topologies, this section aims to analyze their particularities in order to identify the biases of each generating approach, their similarities, and the complexities of searched cells. Therefore, Figure 34 showcases the architectural topology generated by DiNTS, while Figure 35 presents the topologies discovered by the sampling strategies of Entropy, Variance, Standard Deviation, and Random used in Active-DiNTS.

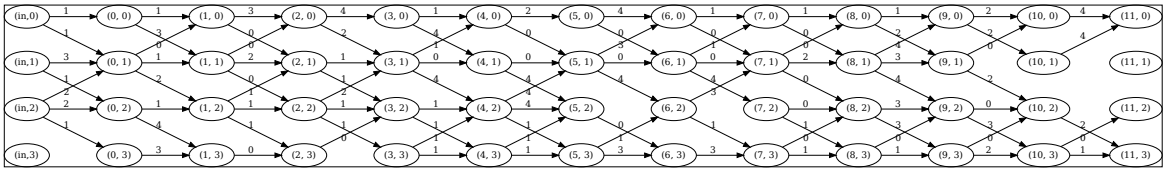
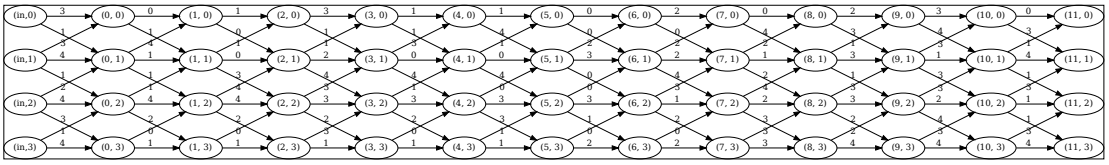
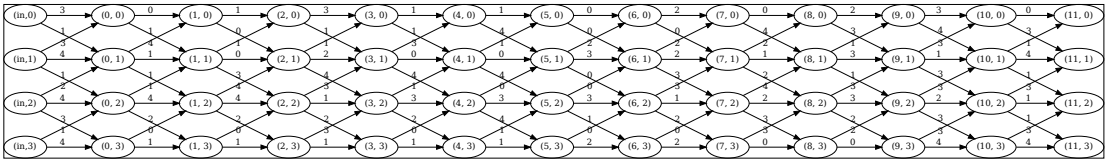


Figure 34 – Original DiNTS (HE *et al.*, 2021) searched architecture on Pancreas dataset. Edges are: 0 = Skip, 1 = 3x3x3, 2 = P3D 3x3x1, 3 = P3D 3x1x3, 4 = P3D 1x3x3.

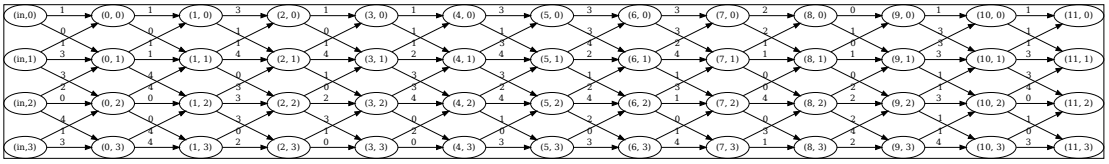
The DiNTS topology shown in Figure 34 contains a total of 97 connections, where each of the 5 possible layer operations is employed. The first operation, the skip connection (0), known for preserving information flow between layers and boosting gradient flow, is used in 24 out of 97 connections, representing 24.7% of the total connections. The 3D convolution operation 3x3x3 (1), which constitutes a 3D convolution with a 3x3x3 kernel, plays a significant role in feature extraction across the entire voxel volume, appears in 33 connections, or approximately 34% of the total number of connections, thus being the most commonly used operation in the topology. Another possible operation is the so-called P3D, which represents the popular pseudo 3D used in other NAS works in the literature (HE *et al.*, 2021; ZHU *et al.*, 2019). The P3D 3x3x1 operation (2) is used in 14 connections and consists of a 3x3x1 convolution followed by a 1x1x3 convolution. In turn, the P3D 3x1x3 operation (3) consists of a 3x1x3 convolution followed by a 1x3x1 convolution, and it is found in 12 connections, representing approximately 12.4% of the total connections and, therefore, the least used operation. Finally, the P3D 1x3x3 operation (4) that consists of a 1x3x3 convolution followed by a 3x1x1 convolution is also used in a total of 14 connections. Therefore, it is concluded that among the valid operations, the most chosen operation in the topology obtained by DiNTS is the 3D 3x3x3 convolution, present in 33/97 connections and thus being the most prominent feature extractor. On the other hand, the P3D 3x1x3 was the least used operation, appearing on only 12/97 connections and making it the least dominant choice for data processing within the architecture.



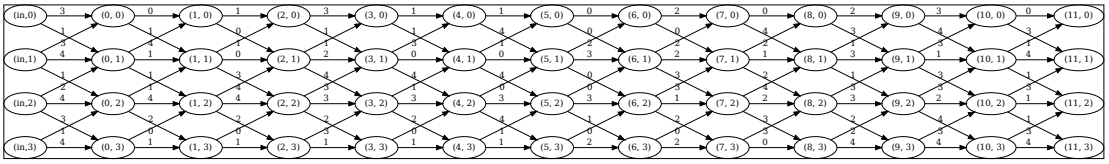
(a) Random at Query 41/97 (82/194 examples).



(b) Entropy at Query 35/97 (70/194 examples).



(c) Variance at Query 36/97 (72/194 examples).



(d) Std at Query 37/97 (74/194 examples).

Figure 35 – Active-DiNTS generated architectures from the least number of queries that surpass DiNTS (Brain). Edges are: 0 = Skip, 1 = 3x3x3, 2 = P3D 3x3x1, 3 = P3D 3x1x3, 4 = P3D 1x3x3.

In contrast to the topologies generated by DiNTS, those discovered by Active-DiNTS uncertainty functions are fully connected, as depicted in Figure 35. Across the uncertainty strategies of Entropy, Variance, Std, and the Random baseline, the 3D convolution 3x3x3 operation (1) consistently emerged as the most common, coinciding interestingly with its prominence in DiNTS topologies. Notably, the skip connection operation (0) was the least frequent for both Random and Std, while for Entropy, it was the P3D 1x3x3 operation (4), and for Variance, it was the P3D 3x3x1 operation (2), diverging from DiNTS that featured P3D 3x1x3 operation (3) as the least frequent. Further comparisons between Random and Std revealed identical operation rankings, with P3D 3x1x3 operation (3) in second place, followed by P3D 3x3x1 operation (1) and P3D 1x3x3 operation (4), despite variations in usage, connection sequences, and information flow. Surprisingly, Variance exhibited a topology distinct from Std, aligning more closely with Entropy. Both shared the P3D 3x1x3 operation (3) as the second most popular, followed by the skip connection operation (0), differing slightly in connection quantity. Remarkably, relative

to the original DiNTS, the uncertainty strategies most akin in topology were Std and Random, sharing common top-used operations. These observations highlight the nuanced differences and similarities in topology generation strategies employed by Active-DiNTS and the original DiNTS, providing insights into their structural preferences and variations.

Table 20 – Cell operation frequency on Active-DiNTS best performance models and original DiNTS. Columns represent the number of times an operation was considered the most or least frequent given all search cells in an architecture.

Method	Most frequent	Least frequent	Total operations
DiNTS	3x3x3 3D convolution (34.0%)	P3D 3x1x3 (12.4%)	97/120
Active-DiNTS (Random)	3x3x3 3D convolution (26.7%)	skip connection (15.0%)	120/120
Active-DiNTS (Entropy)	3x3x3 3D convolution (27.5%)	P3D 1x3x3 (15.0%)	120/120
Active-DiNTS (Variance)	3x3x3 3D convolution (29.2%)	P3D 3x3x1 (13.3%)	120/120
Active-DiNTS (Std)	3x3x3 3D convolution (26.6%)	skip connection (15.8%)	120/120

To complement the qualitative analysis, the statistics from Table 22 elucidate distinct architectural preferences and complexities between methods. All methods showed a reliance on a standard 3x3x3 convolutional operation, suggesting a preference for capturing spatial features in a three-dimensional space. DiNTS, in particular, was the method that proportionally most depended on this 3D convolution operation, emphasizing the extraction of spatial features. In contrast, its less frequent operation, P3D 3x1x3, suggests limited use of a specific convolutional pattern with a horizontal shape followed by a vertical one. Within Active-DiNTS variations, despite a common preference for the 3x3x3 convolutional operation, they do not share the same least frequent operations. Random and Std strategies exhibit a tendency to avoid skip connections, while Entropy uses fewer P3D 1x3x3 operations, suggesting a strategic bias against certain spatial patterns captured by this pseudo-3D convolutional sequence. In turn, the Variance strategy has another P3D operation, the 3x3x1, as the least frequent, reinforcing a shared aversion to these specific convolutional sequences. Another pattern that proved to be constant in all Active-DiNTS variations was the presence of fully connected models, which, as discussed in Table 22, resulted in models with a large number of FLOPs. Thus, this suggests potential challenges in terms of inference speed, particularly in mobile scenarios where latency is important and computationally intense models are less used. However, these same models present reduced memory usage and fewer trainable parameters that could compensate for that. Notably, Active-DiNTS models employ lighter operations than DiNTS, showcasing variations in architectural choices and backtracking costs, with implications for model adaptation and performance, despite some variations such as Active-DiNTS (Std) using fewer skip connections that have minimal cost. These nuanced findings contribute to a comprehensive understanding of shared trends and distinctive features of Active-DiNTS and the baseline DiNTS, offering insights into the strategic preferences of methodologies across cellular operating frequencies.

6.4.3 Segmentation Analysis

This section provides an in-depth qualitative analysis concerning some of the segmentation masks generated by Active-DiNTS. This examination focuses on highlighting qualitative aspects related to precision and recall in brain tumor segmentation, encompassing a fair range of anatomical variations and geometrical patterns, as demonstrated in Figures 36 and 37. The presented MRI scans offer a diverse perspective, incorporating multiple acquired slices and covering the four distinct modalities of FLAIR, T1w, T1 wGd, and T2w, usually employed in the medical field for comprehensive brain analysis. By comparing the predicted segmentation masks from Active-DiNTS with the actual segmentation masks, this examination seeks to provide a thorough understanding of the model's segmentation capabilities, its ability to accurately delineate various brain structures, and its robustness in the face of diverse geometric patterns observed in MRIs. Thus, this analysis is pivotal for assessing Active-DiNTS's performance beyond the quantitative and its potential application in real-world medical image segmentation tasks.

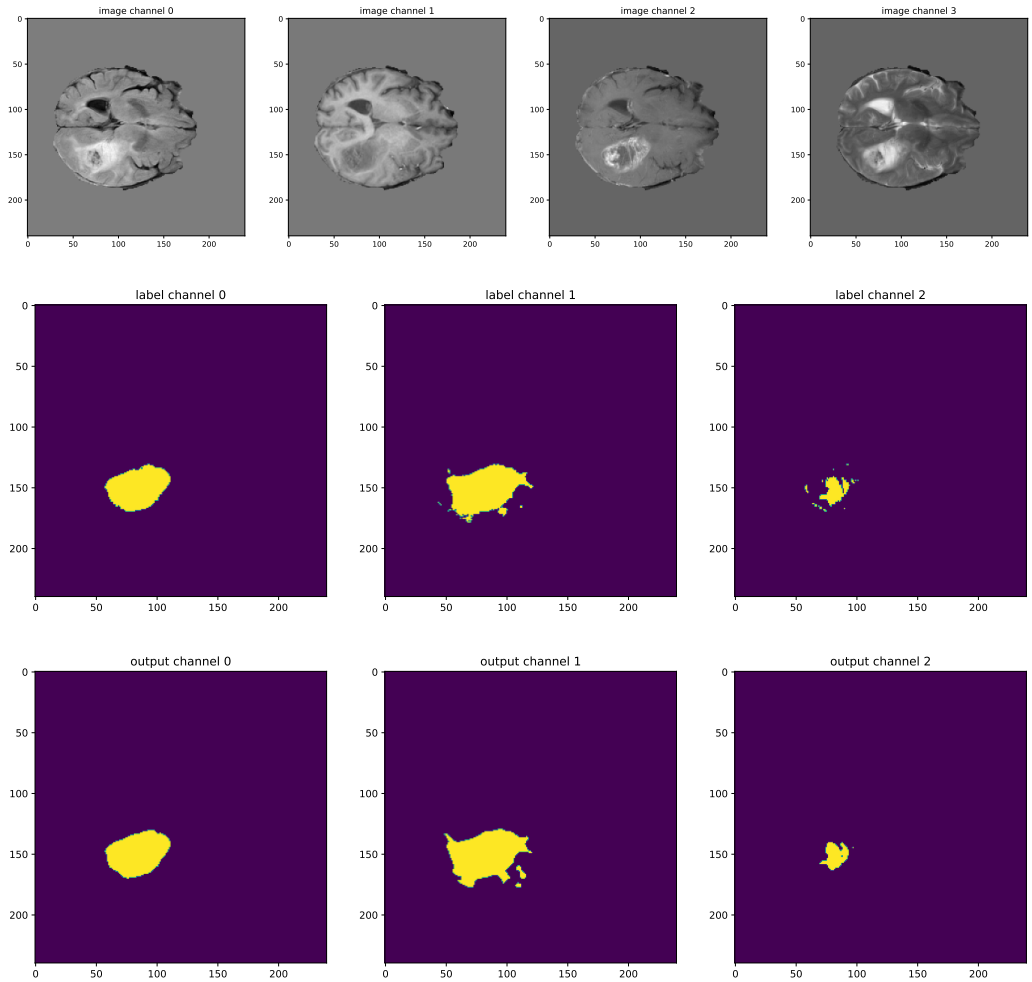


Figure 36 – Brain MRIs and Tumor Segmentation Masks. (Top row) Input image modalities FLAIR, T1w, T1 wGd, and T2w in the 70/150 slice. (Middle row) Ground-truth Segmentation Masks where label channels are Non-enhancing Tumor, Edema, and Enhancing Tumor. (Bottom row) Predicted Segmentation Masks where output channels are for the corresponding classes.

Figure 36 displays a sequence of MRIs and segmentation masks where rounded or elliptical tumor lesions can be clearly detected. These MRI images and segmentation masks are associated with a specific slice of the 3D volume in an axial view, which is the 70th of a total of 150 slices. This specific slice corresponds to the central portion of the brain, where the number of voxels showed the highest concentration of the tumor, displaying a prominent tumor pattern that is visibly denser. This pattern is discernible across all imaging modalities, allowing precise localization of various glioma classes. In these masks, the segmentation process appears to be highly effective. The Edema or entire tumor region in channel 1 exhibits high recall and good precision, successfully identifying subtle patterns. Furthermore, the Non-enhancing Tumor or Tumor core of channel 1 is well captured with good recall and precision. Even the Enhanced Tumor in channel 2, known for its considerably smaller dimension and more complex non-convex shape, exhibits good recall and reasonable precision, although some finer details may be missing. It is worth highlighting that the challenge of the Enhanced Tumor class is attributed to its atypical size and shape compared to other classes of tumors, in addition to being less present in the voxels, requiring greater granularity in segmentation.

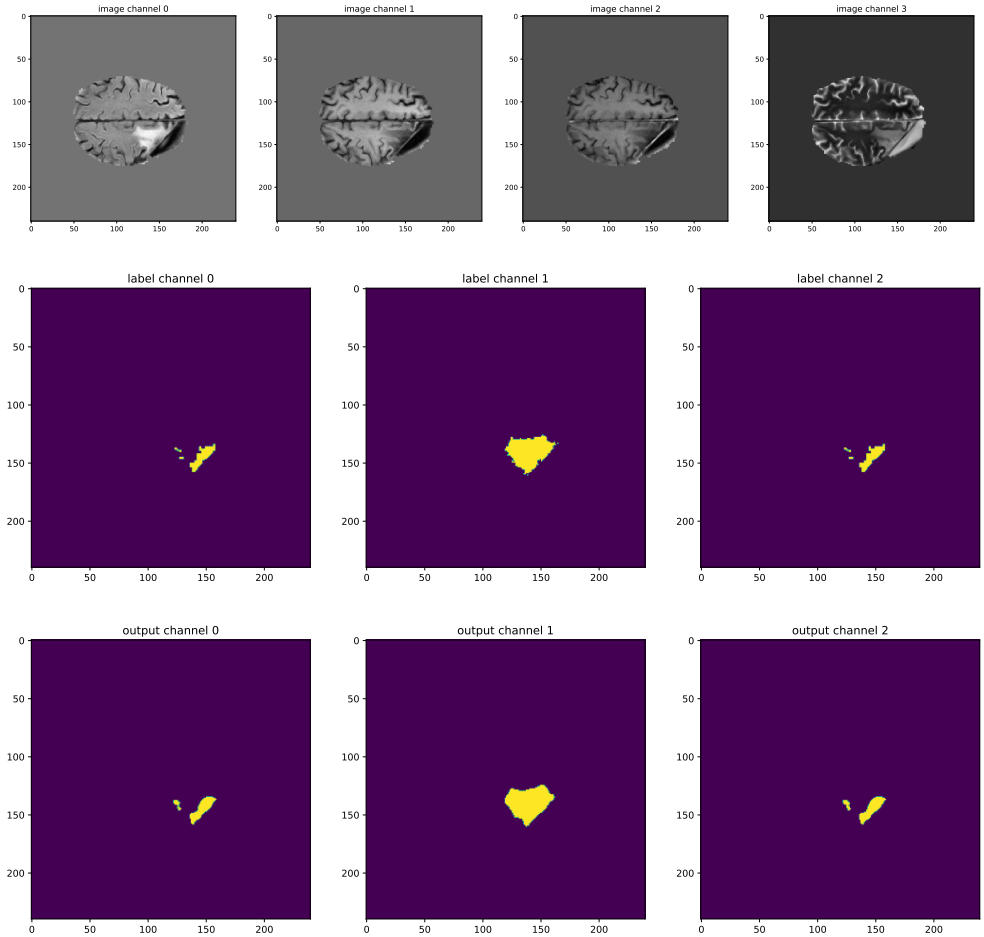


Figure 37 – Brain MRIs and Tumor Segmentation Masks. (Top row) Input image modalities FLAIR, T1w, T1 wGd, and T2w in the 110/150 slice. (Middle row) Ground-truth Segmentation Masks where label channels are Non-enhancing Tumor, Edema Enhancing Tumor. (Bottom row) Predicted Segmentation Masks where output channels are for the corresponding classes.

Unlike the previous case, Figure 37 presents MRI images where it is not possible to detect the tumor lesions clearly, except for channel 0 where it is most evident. Furthermore, as seen in the actual segmentation masks, tumors in this present case show a different pattern, appearing to be sparser, not dense, and following a V-shape. These MRI images and segmentation masks are associated with a slice closest to the end of the 3D volume for the axial view. More precisely, it corresponds to the 110th of a total of 150 slices. Therefore, it is possible to notice a decrease in the number and size of the voxels. Despite the mixed and fuzzy appearance variations, the predicted segmentation masks continue to achieve accurate localization of the tumors, exhibiting high recall and good precision, with only a few errors, such as small fine-grained patterns along the edges of the gliomas. Both the Edema and the Non-enhancing core of the tumor are captured with good recall and precision, where subtle patterns were successfully identified. Lastly, even the most challenging Enhanced core class is well segmented. Active-DiNTS effectively captures the sparse bipartite pattern, similar to the Non-enhanced Tumor pattern, with good recall and precision. In short, considering the two cases in Figures 36 and 37, it was demonstrated that Active-Dints is versatile and can effectively highlight the precise delineation of complex tumor regions across different brain slices, sub-regions, and image modalities, even when dealing with varied patterns and challenging tumor classes.

6.4.4 State-of-the-art Comparative Performance Analysis

This section presents a comprehensive comparative analysis of the best results from all Active-DiNTS variations, along with state-of-the-art methods for the task of Medical Image Segmentation with the Brain MSD dataset. Table 21 showcases a comparison of Dice scores between several Uncertainty strategies used in Active-DiNTS and other baseline methods, while Table 22 compares the relationship of model performances with their complexity measured through FLOPs, trainable parameters, and GPU memory used for training.

Table 21 – Average Dice Score (DSC) results on Task01_BrainTumour from the MSD challenge. DSC1 is for Edema, DSC2 is for Enhancing Tumor, and DSC3 is for Non-enhancing Tumor.

Method	DSC1	DSC2	DSC3	Average
CerebriuDIKU (PERSLEV *et al.*, 2019)	69.52	43.11	66.74	59.79
NVDLMED (XIA *et al.*, 2020)	67.52	45.00	68.01	60.18
SCNAS (KIM *et al.*, 2019)	67.40	45.75	68.26	60.47
nnU-Net (ISENSEE *et al.*, 2019)	68.04	46.81	68.46	61.10
C2FNAS (YU *et al.*, 2020)	67.62	48.60	69.72	61.98
DiNTS (Pancreas) (HE *et al.*, 2021)	69.28	48.65	69.75	62.56
DiNTS (Brain) (HE *et al.*, 2021)	80.20	**61.09**	77.63	72.97
Ours				
Active-DiNTS (Entropy)*	**90.65**	60.04	82.81	**77.83**
Active-DiNTS (Entropy)†	84.58	38.71	67.67	63.65
Active-DiNTS (Entropy)‡	87.45	54.24	78.86	73.52
Active-DiNTS (Variance)*	90.11	59.58	82.50	77.40
Active-DiNTS (Variance)†	81.96	39.64	69.70	63.77
Active-DiNTS (Variance)‡	88.19	53.25	79.25	73.56
Active-DiNTS (Std)*	90.27	60.14	**82.89**	77.77
Active-DiNTS (Std)†	83.16	36.69	69.32	63.06
Active-DiNTS (Std)‡	87.80	54.42	78.85	73.69

* Best final performance
† Overcome DiNTS (Pancreas)
‡ Overcome DiNTS (Brain)

Along with literature methods outlined in Section 6.3.2, the primary DiNTS baseline is presented in two versions in Tables 21 and 22. Both versions are reported in He *et al.* (2021), where DiNTS (Pancreas) is the final model used in most comparisons and obtained through a search on the MSD Pancreas dataset, while DiNTS (Brain) is the alternative version searched in the Brain MSD dataset and reported in only one specific analysis, referenced here to provide broader comparisons on a more equitable basis for analysis. Such comparison analysis also encompasses variations of Active-DiNTS representing final and intermediate models that surpass DiNTS baselines while using fewer resources in terms of training examples, parameters, and

search time. There are three versions for each uncertainty function. Considering the Entropy, for example, there is Active-DiNTS (Entropy)* that is the final model trained with all the data (as the baselines) or, in other words, after all queries. On the other hand, Active-DiNTS (Entropy)† is the intermediate model that overcomes DiNTS (Pancreas) in terms of Average Dice, and Active-DiNTS (Entropy)‡ is the intermediate model that overcomes DiNTS (Pancreas). By intermediate models, it means that not all the data or not all queries were used to search the models. More details regarding those models are discussed after Table 22.

Nearly all Active-DiNTS variations outperform the baselines considering the three tumor classes (DSC1, DSC2, DSC3) and on average in Table 21, except for DiNTS (Brain) in DSC2. Active-DiNTS (Entropy)* achieves the best performance in segmenting the DSC1 class Edema with a Dice of 90.65 and the highest Average Dice of 77.83 across the three classes, proving to be the superior method in predictive performance. The second-best performer, Active-DiNTS (Std)*, stands out on the DSC3 class Non-enhancing Tumor with a Dice of 82.89 and the second-highest Average Dice of 77.77, followed by Active-DiNTS (Variance)* with a 77.40 Average Dice. Among the baselines, DiNTS (Brain) attains the highest results with an Average Dice of 72.97, even securing the top score result for the DSC2 class Enhancing Tumor with a Dice of 61.09. However, when comparing the magnitude of differences, Active-DiNTS clearly outperforms its baseline DiNTS (Brain) with differences exceeding 10% on the DSC1 class, over 5% on DSC3, over 4% on the Average Dice, and on the DSC3 class where DiNTS surpasses the best versions of Active-DiNTS, the difference is only around 1%. For the remaining baselines, they lag significantly behind even compared to DiNTS (Brain). Among Active-DiNTS variations, Entropy and Std consistently achieved the best Dice scores on average, showing their robustness and effectiveness. Specifically, Entropy leads among all variations with the highest average Dice, showing its effectiveness in treating complex tumor structures. Notably, all Active-DiNTS variations consistently demonstrate superior segmentation performance compared to state-of-the-art methods, highlighting its effectiveness in segmenting brain tumors and its ability to take advantage of few multi-modality MRI data, computational resources, and training time.

Table 22 presents a comprehensive evaluation of segmentation models focusing on the computational efficiency attributes discussed previously. At first glance, baseline methods seem to surpass Active-DiNTS in efficiency, generating models with lower FLOPs, trainable parameters, and memory usage. However, these methods consume considerably more resources such as the number of GPUs and the proportional GPU memory usage during search, in addition to the search/re-training time. In contrast, Active-DiNTS generates suitable models already during the search phase, thus not needing to perform the re-training phase. Both DiNTS variations, which have fewer FLOPs and memory usage, have a number of trainable parameters comparable to or higher than some Active-DiNTS variations. Furthermore, DiNTS employs 8 GPUs in a 5.8 GPU days search, while C2FNAS, the most efficient method from a model perspective, also uses 8 GPUs and has the longest search in the comparison, lasting 333 GPU days. In comparison, Active-DiNTS models were obtained in a small fraction of the time, where the fastest, Active-

Table 22 – Models comparison on FLOPs (GigaFLOPs), trainable Parameters (Millions), retraining GPU Memory (MegaBytes), used GPU(s), search Time (GPU days), and Average Dice.

Method	**FLOPs**	**Parameters**	**Memory**	**GPU(s)**	**Time**	**Average**
CerebriuDIKU (PERSLEV *et al.*, 2019)	-	62	-	-	-	59.79
3D UNet (ÇIÇEK *et al.*, 2016)	658	18	9176	**1 TitanX**	3	-
Attention UNet (OKTAY *et al.*, 2018)	1163	104	13465	-	-	-
SCNAS (KIM *et al.*, 2019)	425	-	-	4 V100	1	60.47
C2FNAS (YU *et al.*, 2020)	**151**	**17**	**5730**	8 V100	333	61.98
DiNTS (Pancreas) (HE *et al.*, 2021)	334	152	13018	8 V100	5.8	62.56
DiNTS (Brain) (HE *et al.*, 2021)	334	152	13018	8 V100	5.8	72.97
Ours						
Active-DiNTS (Entropy)*	8500	127	17249	**1 V100**	5.5	**77.83**
Active-DiNTS (Entropy)†	8690	123	14802	**1 V100**	0.33	63.65
Active-DiNTS (Entropy)‡	8770	152	15483	**1 V100**	1.1	73.52
Active-DiNTS (Variance)*	8360	153	17283	**1 V100**	6.4	77.40
Active-DiNTS (Variance)†	8720	125	14835	**1 V100**	**0.21**	63.77
Active-DiNTS (Variance)‡	9650	126	15577	**1 V100**	1.4	73.56
Active-DiNTS (Std)*	8500	127	17307	**1 V100**	4.6	77.77
Active-DiNTS (Std)†	8220	124	14736	**1 V100**	0.42	63.06
Active-DiNTS (Std)‡	8750	154	15545	**1 V100**	1.5	73.69

* Best final performance
† Overcome DiNTS (Pancreas)
‡ Overcome DiNTS (Brain)

DiNTS (Variance)†, the model that surpasses both C2FNAS and DiNTS (Pancreas) in Dice, had a search time up to 27 times faster than DiNTS. Additionally, Active-DiNTS (Entropy)‡, the model that surpasses DiNTS (Brain) in Dice, had a search up to 5 times faster. This search speed-up was mainly attributed to the Active Learning framework, which helped find the smallest set of examples needed to discover suitable architectures. Unlike DiNTS and other models that search using the entire dataset, Active-DiNTS follows an incremental approach that allows training the models with a small but valuable labeled dataset fed with the most informative examples for the current learned model at each query. As a result, unnecessary costs could be reduced, and search time could be shortened. Furthermore, compared to the best Active-DiNTS (Entropy)* variation, C2FNAS lags significantly behind in predictive performance, more precisely, with a difference of 15% in Dice. It is also worth mentioning that the Active-DiNTS search and training is done with a single 32 GB Tesla V100 GPU, which uses less than half the memory in the intermediate versions that surpass DiNTS and just over half the memory in the final versions, ranging from 17GB on the heaviest configurations to around 15GB on configurations that surpass DiNTS (Brain). In turn, DiNTS uses 8 16GB Tesla V100 GPUs and, in comparison, generates models that consume up to 13GB or 81.25% of the available memory, while Active-DiNTS uses around 54% in its heaviest configuration and 48.44% in the version lighter than both versions of DiNTS. Despite generating more computationally intensive models, Active-DiNTS models also have fewer trainable parameters than DiNTS variations in intermediate configurations that outperform them in segmentation performance, with 127 million versus 152 million parameters.

In terms of FLOPs, Active-DiNTS generate denser models, which can be attributed to the RAM factor parameter that controls cell connections and the optimization process that prioritizes performance before computational efficiency. Thus, Active-DiNTS models present 8684 FLOPs on average, with Active-DiNTS (Std)† being the lightest and Active-DiNTS (Variance)‡ the heaviest configuration, generating architectures with 8220 and 9650 FLOPs, respectively. The searched architectures contain 134 million trainable parameters on average, varying from 123 million on Active-DiNTS (Entropy)† to 154 million on Active-DiNTS (Std)‡. Regarding GPU memory usage, the average usage is 15.8GB of VRAM among Active-DiNTS variations, ranging from 14.7GB to 17.3GB from the intermediate model Active-DiNTS (Std)† and the final model Active-DiNTS (Std)*. In summary, Entropy and Variance present higher FLOPs compared to Std, the latter being the most efficient. Memory efficiency varies slightly, with Variance being the most efficient while Std is the most expansive. Notably, despite the computational intensity of Active-DiNTS variations, they provide competitive performance and meet diverse computational resources using only one GPU with reduced search time.

Unlike the baseline methods that search and re-train the discovered architectures with increased complexity, Active-DiNTS already generates sufficiently good models that eliminate the need for retraining and increasing model complexity. In addition, Active-DiNTS offers a collection of possible models from each uncertainty function, whose selection of appropriate variation depends on the available computational resources and the specific requirements of the segmentation task. Furthermore, observing the evolution curve of these efficient indicators reveals a linear correlation between average performance and memory usage, showing greater GPU usage for superior performance. However, this pattern does not apply to FLOPs and trainable parameters. It is also worth highlighting that low FLOPs and fewer parameters are more relevant in mobile configurations. For medical image analysis, a lightweight and low latency model is less important than a better use of the little data commonly available in medical applications and the precision in segmentation, both criteria met by Active-DiNTS.

6.5 Chapter Remarks

This chapter presented a novel NAS method for 3D Medical Image Segmentation called Active Differentiable Network Topology Search (Active-DiNTS). Going beyond conventional Meta-Learning in typical tasks such as Classification, this chapter extended the exploration of meta-knowledge derived from an Active Learning framework applied to Image Segmentation. Distinguished from other NAS methods, Active-DiNTS features a flexible network topology search space explored through fast gradient-based search. This proposal considers a Pool-based Sampling Scenario where training examples are sampled from a large base of unlabeled examples, the pool set. These sampled examples are given as input to Active-DiNTS, which generates predictions and, based on such outputs, selects the most informative examples to be labeled and trained with. To be more precise, Active-DiNTS uses uncertainty functions to evaluate the predictions of unlabeled examples, thus selecting the examples where the model had the most difficulty in predicting. This is done with the idea that these examples are the most informative and, therefore, the most useful for improving and accelerating learning. For such labeling to occur, the selected examples are sent to the Oracle or expert user, an agent that has knowledge of how to label incoming unlabeled examples. Thus, throughout this Active Learning process, Active-DiNTS iteratively receives more labeled examples with the expectation of generating the greatest gain in predictive performance with each new example and, at the same time, aiming to maintain a minimum number of labeled examples for training.

Experiments with Active-DiNTS included comparisons with several segmentation methods from the literature and between variations of the proposed method. These methods included NAS models, such as the original DiNTS, and hand-crafted models that used fixed architecture, such as the popular nn-Unet. Among Active-DiNTS variations are the different AL setups used, as mentioned in Section 6.2: (i) Weights update; (ii) Topology update; and (iii) Both weights and topology update. Among the uncertainty functions adopted to select the most informative examples during the search/training process are Entropy, Variance, and Standard Deviation, in addition to the Random method used as a lower bound baseline.

As mentioned in Section 6.4, this chapter's experiments were designed to answer three research questions. The first question, "Can simple active sampling strategies used by Active-DiNTS present comparable or better performances than traditional segmentation models?", is covered in the class-by-class analyses of Table 21, where predictive performances measured by Dice are compared and discussed. As seen, Active-DiNTS outperforms baselines such as C2FNAS and nn-UNet by a large margin, also extending its superiority to the main baseline DiNTS. More precisely, in the Edema and Non-enhancing Tumor classes, Active-DiNTS surpasses the best version of DiNTS searched in the Brain dataset by 10% and 5%, respectively, and approximately by 5% in the overall average, falling behind only in the Enhancing Tumor class by approximately 1%. Compared to the originally reported DiNTS searched on the Pancreas

dataset, the overall average Dice difference is even greater, being more than 15%. Additionally, a qualitative analysis was carried out to corroborate the Active-DiNTS effectiveness through the inspection of segmentation masks in Figures 36 and 37, highlighting the predictions' reliability and great similarity with ground truth segmentation masks.

The second question, "Can Active-DiNTS with varying AL strategies offer improved computational efficiency when compared to other segmentation models?", is addressed by the analysis of Table 22 that compares computational efficiency indicators such as the number of trainable parameters, FLOPs, and memory usage. At first glance, Active-DiNTS appeared as computationally expensive due to its high FLOPs caused by the denser and more computationally intense models, in addition to an optimization prioritizing performance rather than resource efficiency. Despite such characteristics, models generated by Active-DiNTS presented greater efficiency in other criteria. Active-DiNTS variations had fewer trainable parameters than DiNTS variations, with 127 million compared to 152 million in the best configuration. In terms of hardware and memory usage, Active-DiNTS also presented more efficient solutions. While Active-DiNTS results are obtained using only one Tesla V100 32GB, both DiNTS and C2FNAS perform their searches using eight Tesla V100 16GB GPUs. In terms of GPU memory usage for search, Active-DiNTS uses more memory in absolute values, but in perspective, it uses less or just over half the memory of the single 32GB GPU, ranging from 17GB on the heaviest configurations to about 15GB on configurations that surpass DiNTS (Brain). DiNTS, meanwhile, uses around 13GB out of the 16GB GPU capacity. When the matter is search efficiency, DiNTS takes 139.2 hours or 5.8 GPU days to search, while Active-DiNTS achieves the same result in about 10.4 hours or 0.43 GPU days. Furthermore, to achieve the same results as DiNTS, less than 20% of the original dataset was used by Active-DiNTS. Although some literature methods such as C2FNAS are excellent FLOPs-wise, memory usage, and model size, they lag significantly in predictive performance, falling behind Active-DiNTS (Entropy) with more than 15% difference.

The third research question, "Can different AL strategies produce significantly divergent results?", is discussed in both Figures 32 and 33, where performance curves for each uncertainty function and AL configuration are compared, and in Figure 35 which discusses the different architectural patterns generated by Active-DiNTS variations. Uncertainty functions presented similar results among themselves regarding Dice progress curves over the number of queries in the Active Learning framework, differing more considerably at the beginning of the process to around 30% of queries, and stabilizing during the rest of the learning process. This pattern is observed both when considering a specific AL setup and evaluating each uncertainty function curve, and when considering the uncertainty function averages and evaluating each curve AL setup separately. Additionally, the differences in performance and resource efficiency of Active-DiNTS variations were also addressed in the analyses of Table 22. When observing the evolution of efficiency indicators in Active-DiNTS final and intermediate models, a linear correlation between average Dice performance and memory usage was observed, showing greater use of GPU for superior performance. However, this standard does not apply to FLOPs and trainable

parameters, which are optimized later in the process. The trainable parameters of Active-DiNTS models remained consistent, with small differences between configurations. Memory usage also varies little and consistently, with Active-DiNTS (Std) being the highest in terms of consumption, and Active-DiNTS (Entropy) being the cheapest on average. While Active-DiNTS (Std) has the shortest search time in the final best performance configuration, Active-DiNTS (Variance) and Active-DiNTS (Entropy) are the most efficient for intermediate configuration models that outperform DiNTS (Pancreas) and DiNTS (Brain), respectively. Active-DiNTS variations, despite their computational intensity, provided competitive performance and addressed diverse computational resources. Notably, the Entropy and Variance variations demonstrated slightly higher memory efficiency, which indicates that selecting the appropriate Active-DiNTS configuration depends on the available computational resources and the specific task requirements.

By using an Active Learning process to select the most informative examples to update its internal structure, Active-DiNTS was able to generate architectures using less data, in shorter search times, using comparable or less computational resources, presenting better overall predictive performances than state-of-the-art methods. This showcase

In the following, the final chapter concludes this thesis by discussing the main contributions, limitations, and future research directions.

CHAPTER

7

CONCLUSION

This doctoral thesis investigated Meta-Learning approaches and related topics, such as Transfer Learning and Active Learning, in order to optimize the Neural Architecture Search framework. In addition to proof-of-concept experiments that validated hypotheses involving Meta-Learning and helped define the building blocks for NAS solutions, two architecture search methods were proposed: Model-based Meta-Learning for Neural Architecture Search (MbML-NAS), and Active Differentiable Network Topology Search (Active-DiNTS).

While MbML-NAS consists of a Prediction-based method that selects ConvNet architectures for Image Classification, Active-DiNTS is a One-shot method that generates ConvNet-like architectures such as U-Nets focused on Image Segmentation. Another difference is that the first proposal focuses on natural 2D images, while the second focuses on the more complex 3D images in the medical domain. Furthermore, both methods are built on fundamental ideas from the field of NAS and Meta-Learning, like exploring cell-based search spaces and performance estimation strategies based on meta-knowledge to optimize the search for architectures.

To answer the overall research question that guided the development of this Ph.D., *How to automatically find neural architectures for Image Recognition with comparable predictive performances using fewer data and less complex procedures than popular NAS methods?*, and to validate the formulated hypothesis, *A Meta-Learning approach can leverage prior knowledge to reduce the computational burden and simplify the search for Image Recognition architectures with good predictive performance using fewer data samples than NAS state-of-the-art methods*, several experiments encompassing the design, implementation, and evaluation of the novel methods tailored to address the challenges posed by the research question were performed. Initially, exploratory experiments served as a proof-of-concept that helped advance the application of Meta-Learning to optimize NAS. These validations, although exploratory and of low complexity, had a large and extremely important impact on what later became the main methods of this doctoral research, the MbML-NAS and Active-DiNTS models. The performed experiments

focused on evaluating setups and methods according to various criteria, such as predictive performance indicators and efficiency metrics. Thus, and to also fulfill the research objective of proposing and investigating MtL methods, the proposals were analyzed according to their predictive performances, quantity of used training data, search and training time, overall use of computational resources, pipeline, and model complexity.

As a result, the research hypothesis was not rejected since both MbML-NAS and Active-DiNTS could take advantage of prior knowledge from tasks, performances, and models to generate architectures with good predictive performances while using fewer computational resources, less training data, shorter search times, and lower model complexity. The objectives of the thesis were also achieved, as it investigated the applicability of MtL in NAS, and its impact on performance and data efficiency, in addition to the creation of new meta-datasets suitable for NAS and the presentation of an updated literature review on the topics covered.

The motivations raised were also addressed incrementally and iteratively. For example, the problem of manually defining neural architectures has been addressed by employing suitable NAS approaches. As for the NAS simplification problem, it was mainly addressed by MbML-NAS which employed a simplified and direct approach that achieves comparable or superior results to more complex literature methods. Additionally, the NAS framework optimization was better exemplified in Active-DiNTS, which, through an Active Learning framework, was able to overcome several strong methods focused on predictive performance using only 1/4 of the original labeled data and a small fraction of computational resources. Finally, the third problem involving the study of Meta-Learning approaches applied to NAS was addressed throughout all the experiments in this thesis, where in each chapter, more than one approach was explored, often combinations of different approaches, showing that meta-knowledge is not only applicable but also beneficial to NAS. In essence, this thesis contributed to the fields of NAS optimization and Meta-Learning by introducing new methods that facilitate data-driven architecture design and efficient hyperparameter tuning. Furthermore, it also showed that related approaches, such as Transfer learning and Active Learning, had room for application in NAS and could generate interesting solutions. These findings lay the groundwork for future research and highlight the synergistic potential of Meta-Learning approaches to optimize NAS even further.

7.1 Main Contributions

The main contributions of this doctoral research were the proposals and investigations of two efficient NAS methods based on different Meta-Learning approaches, the MbML-NAS, which learns from task properties and model evaluations, and Active-DiNTS, which leverages knowledge from task properties, previous evaluations, and prior models.

MbML-NAS is a Prediction-based method that uses a small set of six interpretable meta-features representing semantic knowledge about the NAS task. Along with such meta-knowledge in the form of meta-features from ConvNet layers, previous performances from pre-trained architectures are used. Another striking feature of this proposal is the use of few data for training combined with the usage of several traditional and simple inspectable regressors. Such algorithms are categorized into regularized linear models and methods based on decision trees, which turn MbML-NAS into a cheap solution in comparison to its competitors. Therefore, MbML-NAS is trained to generate models capable of predicting the performances of incoming architectures based on architectural characteristics and, based on these predictions, it selects the most promising candidate architectures to accelerate the NAS process. As a result of the experiments carried out with MbML-NAS, the following publication was generated:

- PEREIRA, G. T.; SANTOS, I. B.; GARCIA, L. P.; URRUTY, T.; VISANI, M.; CARVALHO, A. C. de. Neural architecture search with interpretable meta-features and fast predictors. Information Sciences, Elsevier, p. 119642, 2023

Active-DiNTS is a One-shot method that uses a flexible topology search space capable of finding architectures with different topological patterns using an efficient Active Learning framework. This Cell-based search space, in addition to allowing the discovery of popular architectures such as the U-Net and Auto-DeepLab, enables exploring the space of models using fast gradient-based search, generating an efficient method for finding adaptable architectures. Furthermore, Active Learning strategies help to optimize the method even further, which assumes a Pool-based Sampling Scenario to select the most relevant examples for training. Consequently, a drastic reduction in the amount of data is needed for NAS. These training examples are selected using a simple but very popular Uncertainty sampling strategy, where functions such as the calculation of Entropy, Variance, and Standard Deviation are used to select the samples in which the model has the most difficulty in segmenting. The main idea behind this selection process is that the most uncertain examples are possibly the most valuable for learning. Therefore, several experiments carried out with Active-DiNTS and compiled in this thesis are being formatted into a scientific paper to be submitted to an indexed international conference.

As previously mentioned, the experiments with TL and DR also brought important contributions, especially to the proposal of the main methods presented in this thesis. The TL experiments verified that the meta-knowledge related to model recommendation has adequate

generalization capacity across model spaces and multiple datasets. In turn, experiments with DR showed that many meta-databases have redundancy, and thus, by using fewer meta-features, if well selected, it is possible to achieve comparable results when using the original datasets, at the same time that fewer data is used and shorter training times are required. From these two proofs-of-concept experiments, hypotheses were formulated that were later confirmed with MbML-NAS experiments, where few meta-features can generate results comparable to the state-of-the-art while generalizing across datasets, models, and search spaces. In addition, these experiments generated two papers:

- PEREIRA, G. T.; SANTOS, M. d.; ALCOBAÇA, E.; MANTOVANI, R.; CARVALHO, A. Transfer learning for algorithm recommendation. arXiv preprint arXiv:1910.07012, 2019
- PEREIRA, G. T.; SANTOS, M. R. d.; CARVALHO, A. C. P. d. L. F. de. Evaluating meta-feature selection for the algorithm recommendation problem. arXiv preprint arXiv:2106.03954, 2021

Throughout the development of this thesis, some software artifacts were also produced. This includes all the original code written for MbML-NAS, Active-DiNTS, and the TL and DR experiments, everything well documented and open to the public via the GitHub repositories. In addition, the meta-datasets created and used for training, pre-trained models, and a vast variety of experiments and analyses were also made publicly available.

In the initial phase of the Ph.D., other papers were also produced due to the exploration of potential problems related to the primary research theme:

- PEREIRA, G. T.; CARVALHO, A. C. de. Bringing robustness against adversarial attacks. Nature Machine Intelligence, Nature Publishing Group UK London, v. 1, n. 11, p. 499–500, 2019
- TONON, V.; SILVA, T. da; FERREIRA, V.; PEREIRA, G.; REZENDE, S. Evaluating vector representations from user's reviews in a recommendation task. In: SBC. Anais do XVI Encontro Nacional de Inteligência Artificial e Computacional. p. 286–296, 2019.
- ALCOBAÇA, E.; SANTOS, D. P. d.; SANTOS, M. R. d.; PEREIRA, G. T.; MANTOVANI, R. G.; GARCIA, L. P. F.; MASTELINI, S. M.; CARVALHO, A. C. P. d. L. F. de. End-to-end data science (pajé). Resumos, 2019.

Furthermore, the preliminary proposal for this thesis in summary format was internationally recognized by the 2021-2022 Microsoft Research PhD Fellowship award, which helped to validate some of the initial ideas that latter became the main contributions of this Ph.D.

7.2 Prospective Works

While all the experiments and proposals presented in this thesis have successfully achieved their intended purposes, some avenues for potential extensions remain. One possible extension for the Transfer Learning experiments is to increase the model complexity by employing deeper Feed-Forward Networks. Additionally, exploring other types of neural architectures, such as ConvNets and U-Nets, presents a complementary extension to the existing experiments carried out throughout this thesis. Naturally, the application domains would also be extended beyond the Algorithm Recommendation problem to encompass Computer Vision challenges for exploring larger and more complex datasets. As the TL experiment was the simplest possible and of an exploratory nature to validate elementary ideas for the implementation of NAS solutions, increasing its complexity in a proportional matter would be the next logical step. Another interesting avenue would be a detailed examination of individual layers within the TL framework. Although the TL setup already examined the impact of individual layers, it was limited by the simpler models and the low variety of TL configurations. Thus, expanding the experimental setup to include deeper models and more TL configurations, such as freezing more layers or specific groups of layers to observe specific layer behaviors or groups of layers in a more detailed analysis, would be possible. This analysis of individual layers can help identify layers that best encapsulate metaknowledge, presenting the potential for optimizing models by pruning redundant or less valuable layers. Moreover, it can serve as a model inspection tool, or for identifying layers that best transfer meta-knowledge.

Although the investigation of meta-data dimensionality included a wide range of meta-datasets, the experiment focused on the niche domain of recommending algorithms for optimization problems. Thus, a possible future work would be to expand the scope of the experiment to include other domains such as the Image Classification and Image Segmentation problems already addressed in this thesis, also considering both natural images and medical images. Even though the employed meta-datasets presented a good variety of meta-features and classes, and even a good range in the number of examples with datasets of up to 9000 examples, these are small numbers when considering today's state of ML research. Therefore, scaling these experiments to larger databases and analyzing the differences in dimensionality reduction would provide a more comprehensive understanding of the impact of such practice on the model recommendation task. Another natural extension would be to expand the list of FS methods, including the unsupervised and very simple Person or Spearman correlations that do not depend on supervision and are very simple, and expand the supervised FS methods, both those based on statistical filters such as Mutual Information and False Positive Rate test, as well as the model-based that use feature importance such as L1-based and Tree-based FS. Also, expanding FE methods would bring robustness to the study, including popular and more contemporary baselines such as Linear Discriminant Analysis (LDA) (MIKA *et al.*, 1999), t-distributed Stochastic Neighbor Embedding

(t-SNE) (MAATEN; HINTON, 2008), and Uniform Manifold Approximation and Projection (UMAP) (MCINNES; HEALY; MELVILLE, 2018).

The first natural point of improvement for MbML-NAS would be to extend the number of employed meta-features beyond the employed Model-based and Statistical groups, including Information Theory metrics (ALEXANDROS; MELANIE, 2001), Landmarking (BENSUSAN; GIRAUD-CARRIER, 2000), and Complexity Measures (LORENA *et al.*, 2019) to encode and approximate the quality of neural architectures. Additionally, exploring unsupervised ways to extract meta-knowledge from the internal representations of neural architectures using these meta-features as a basis may have great potential to improve the performance of MbML-NAS in more complex tasks. An interesting path involving such meta-features would be to investigate their behavior throughout training, monitoring, and extracting knowledge from their evolution during the end-to-end process. Another promising direction is to extend MbML-NAS to cover other performance measures, such as Precision and Recall, which are especially suitable when dealing with imbalanced tasks. Finally, applying MbML-NAS to other tasks, such as Image Segmentation and exploring different Search Spaces and datasets to validate the effectiveness of the proposal in different domains is also a possibility.

A possible extension for Active-DiNTS is in 2D Image Segmentation with natural images, where popular datasets such as the Common Objects in COntext (Coco) (LIN *et al.*, 2014) and the PASCAL Visual Object Classes (PASCAL VOC) (EVERINGHAM; WINN, 2011) could be used. These widely used benchmarks present a good opportunity to test the generalization power and efficiency of Active-DiNTS against the best segmentation models in the literature. Even though it may be a simple adaption, experiments with natural images could lead to interesting insights into how Active-DiNTS behaves under different data distributions and image patterns different from those of MRIs. Furthermore, adapting the method to treat 2D images instead of 3D may also be interesting to evaluate how robust Active-DiNTS is, whether it may not be able to find more simplistic architectures, or whether the search cost is constant regardless of the complexity of the data. Although Pool-based Sampling is one of the most popular AL scenarios, other popularly used scenarios are also interesting candidates to be adopted in Active-DiNTS. In particular, the Stream-based Sampling Scenario emerges as a viable option due to its proximity to the current Active-DiNTS approach. In this scenario, labels are not queried from a pool of examples but are instead provided one by one for the learner. The learner then queries the label if it deems the example useful, and this usefulness can be measured by prediction uncertainty, for instance. Another possibility is to adapt the Membership Query Synthesis scenario, where the learner generates their own examples from an underlying distribution. This is particularly useful when data is scarce or expensive to label, as is the case with the majority of medical applications for which Active-DiNTS is proposed. In the case of images with multiple complex patterns, the learner could send the Oracle a cropped image of one of these patterns and query if this appendage belongs to the most general class of patterns. Another point of improvement would be in the query strategies for example selection. Within the group of uncertainty sampling functions,

popular options are the Least confidence, Margin confidence, and Ratio confidence methods. In addition to Uncertainty sampling, Query by Committee, where selected examples are determined by the degree of disagreement between a variety of voting models, and Expected Model Change, which as the name suggests, selects examples that will generate the greatest change in model behavior, have great potential for application in future variations of Active-DiNTS.

BIBLIOGRAPHY

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Available: <http://tensorflow.org/>. Citation on page 81.

ALCOBAÇA, E.; MANTOVANI, R. G.; ROSSI, A. L.; CARVALHO, A. C. de. Dimensionality reduction for the algorithm recommendation problem. In: IEEE. **2018 7th Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.], 2018. p. 318–323. Citations on pages 43, 65, 88, 92, and 94.

ALCOBAÇA, E.; SIQUEIRA, F.; RIVOLLI, A.; GARCIA, L. P.; OLIVA, J. T.; CARVALHO, A. C. de. Mfe: Towards reproducible meta-feature extraction. **Journal of Machine Learning Research**, v. 21, n. 111, p. 1–5, 2020. Citation on page 66.

ALEXANDROS, K.; MELANIE, H. Model selection via meta-learning: a comparative study. **International Journal on Artificial Intelligence Tools**, World Scientific, v. 10, n. 04, p. 525–554, 2001. Citation on page 174.

AMARI, S. A theory of adaptive pattern classifiers. **IEEE Transactions on Electronic Computers**, IEEE, n. 3, p. 299–307, 1967. Citation on page 51.

ANTONELLI, M.; REINKE, A.; BAKAS, S.; FARAHANI, K.; KOPP-SCHNEIDER, A.; LANDMAN, B. A.; LITJENS, G.; MENZE, B.; RONNEBERGER, O.; SUMMERS, R. M. *et al.* The medical segmentation decathlon. **Nature communications**, Nature Publishing Group UK London, v. 13, n. 1, p. 4128, 2022. Citations on pages 36, 47, 70, and 148.

AZIZI, S.; MOUSAVI, P.; YAN, P.; TAHMASEBI, A.; KWAK, J. T.; XU, S.; TURKBEY, B.; CHOYKE, P.; PINTO, P.; WOOD, B. *et al.* Transfer learning from rf to b-mode temporal enhanced ultrasound features for prostate cancer detection. **International journal of computer assisted radiology and surgery**, Springer, v. 12, p. 1111–1121, 2017. Citation on page 70.

BAEHRENS, D.; SCHROETER, T.; HARMELING, S.; KAWANABE, M.; HANSEN, K.; MÜLLER, K.-R. How to explain individual classification decisions. In: JMLR ORG. **JMLR**. [S.l.], 2010. p. 1–18. Citation on page 130.

BAKER, B.; GUPTA, O.; NAIK, N.; RASKAR, R. Designing neural network architectures using reinforcement learning. **arXiv preprint arXiv:1611.02167**, 2016. Citations on pages 34 and 63.

BAKER, B.; GUPTA, O.; RASKAR, R.; NAIK, N. Accelerating neural architecture search using performance prediction. **arXiv preprint arXiv:1705.10823**, 2017. Citations on pages 56, 61, and 62.

BAKKER, B.; HESKES, T. Task clustering and gating for bayesian multitask learning. **Journal of Machine Learning Research**, v. 4, n. May, p. 83–99, 2003. Citation on page 67.

BANDARU, S.; NG, A. H.; DEB, K. Data mining methods for knowledge discovery in multi-objective optimization: Part a-survey. **Expert Systems with Applications**, Elsevier, v. 70, p. 139–159, 2017. Citation on page 60.

BELLMAN, R. Dynamic programming. **Science**, American Association for the Advancement of Science, v. 153, n. 3731, p. 34–37, 1966. Citation on page 79.

BENDER, G.; KINDERMANS, P.-J.; ZOPH, B.; VASUDEVAN, V.; LE, Q. Understanding and simplifying one-shot architecture search. In: **International Conference on Machine Learning**. [S.l.: s.n.], 2018. p. 549–558. Citations on pages 34, 62, and 109.

BENGIO, Y.; DUCHARME, R.; VINCENT, P. A neural probabilistic language model. **Advances in neural information processing systems**, v. 13, 2000. Citation on page 51.

BENGIO, Y.; LODI, A.; PROUVOST, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. **European Journal of Operational Research**, Elsevier, v. 290, n. 2, p. 405–421, 2021. Citation on page 94.

BENSUSAN, H.; GIRAUD-CARRIER, C. Discovering task neighbourhoods through landmark learning performances. In: SPRINGER. **European Conference on Principles of Data Mining and Knowledge Discovery**. [S.l.], 2000. p. 325–330. Citations on pages 66 and 174.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. **Journal of machine learning research**, v. 13, n. Feb, p. 281–305, 2012. Citation on page 97.

BERGSTRA, J.; YAMINS, D.; COX, D. D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. Jmlr, 2013. Citation on page 60.

BISCHL, B.; KERSCHKE, P.; KOTTHOFF, L.; LINDAUER, M.; MALITSKY, Y.; FRÉCHETTE, A.; HOOS, H.; HUTTER, F.; LEYTON-BROWN, K.; TIERNEY, K. *et al.* Aslib: A benchmark library for algorithm selection. **Artificial Intelligence**, Elsevier, v. 237, p. 41–58, 2016. Citations on pages 67, 78, 81, 93, and 98.

BISHOP, C. M. **Pattern Recognition and Machine Learning (Information Science and Statistics)**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 0387310738. Citations on pages 42, 96, and 114.

BOMMERT, A.; SUN, X.; BISCHL, B.; RAHNENFÜHRER, J.; LANG, M. Benchmark for filter methods for feature selection in high-dimensional classification data. **Computational Statistics & Data Analysis**, Elsevier, v. 143, p. 106839, 2020. Citations on pages 88 and 89.

BRAZDIL, P.; CARRIER, C. G.; SOARES, C.; VILALTA, R. **Metalearning: Applications to data mining**. [S.l.]: Springer Science & Business Media, 2008. Citations on pages 35, 60, 65, and 129.

BREIMAN, L. Random forests. In: **Machine learning, Springer**. [S.l.: s.n.], 2001. p. . Citation on page 121.

BRODERSEN, K. H.; ONG, C. S.; STEPHAN, K. E.; BUHMANN, J. M. The balanced accuracy and its posterior distribution. In: IEEE. **2010 20th International Conference on Pattern Recognition**. [S.l.], 2010. p. 3121–3124. Citation on page 97.

CAI, H.; ZHU, L.; HAN, S. Proxylessnas: Direct neural architecture search on target task and hardware. **arXiv preprint arXiv:1812.00332**, 2018. Citation on page 60.

CALIŃSKI, T.; HARABASZ, J. A dendrite method for cluster analysis. **Communications in Statistics-theory and Methods**, Taylor & Francis, v. 3, n. 1, p. 1–27, 1974. Citation on page 66.

CARDOSO, M. J.; LI, W.; BROWN, R.; MA, N.; KERFOOT, E.; WANG, Y.; MURREY, B.; MYRONENKO, A.; ZHAO, C.; YANG, D. *et al.* Monai: An open-source framework for deep learning in healthcare. **arXiv preprint arXiv:2211.02701**, 2022. Citation on page 151.

CASTIELLO, C.; CASTELLANO, G.; FANELLI, A. M. Meta-data: Characterization of input features for meta-learning. In: SPRINGER. **International Conference on Modeling Decisions for Artificial Intelligence**. [S.l.], 2005. p. 457–468. Citation on page 66.

CHEN, W.; GONG, X.; WANG, Z. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In: **International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2021. p. 1–12. Citations on pages 34, 63, and 109.

CHOLLET, F. *et al.* **Keras**. 2015. <https://keras.io>. Citation on page 81.

CHRABASZCZ, P.; LOSHCHILOV, I.; HUTTER, F. A downsampled variant of imagenet as an alternative to the cifar datasets. **arXiv preprint arXiv:1707.08819**, 2017. Citations on pages 61 and 115.

ÇIÇEK, Ö.; ABDULKADIR, A.; LIENKAMP, S. S.; BROX, T.; RONNEBERGER, O. 3d u-net: learning dense volumetric segmentation from sparse annotation. In: SPRINGER. **Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19**. [S.l.], 2016. p. 424–432. Citations on pages 47, 139, 149, and 163.

COHN, D.; ATLAS, L.; LADNER, R. Improving generalization with active learning. **Machine learning**, Springer, v. 15, p. 201–221, 1994. Citations on pages 35 and 71.

COHN, D. A.; GHAHRAMANI, Z.; JORDAN, M. I. Active learning with statistical models. **Journal of artificial intelligence research**, v. 4, p. 129–145, 1996. Citations on pages 71 and 72.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of control, signals and systems**, Springer, v. 2, n. 4, p. 303–314, 1989. Citation on page 50.

DASGUPTA, S. Two faces of active learning. **Theoretical computer science**, Elsevier, v. 412, n. 19, p. 1767–1781, 2011. Citations on pages 35, 72, and 143.

DAVIS, M.; LOGEMANN, G.; LOVELAND, D. A machine program for theorem-proving. **Communications of the ACM**, ACM New York, NY, USA, v. 5, n. 7, p. 394–397, 1962. Citation on page 67.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. **The Journal of Machine learning research**, v. 7, n. 1, p. 1–30, 2006. Citation on page 98.

DENG, B.; YAN, J.; LIN, D. Peephole: Predicting network performance before training. **CoRR**, 2017. Available: <http://arxiv.org/abs/1712.03351>. Citations on pages 34, 108, 120, and 127.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. **2009 IEEE conference on computer vision and pattern recognition**. [S.l.], 2009. p. 248–255. Citations on pages 44 and 115.

DOMHAN, T.; SPRINGENBERG, J. T.; HUTTER, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: **Twenty-Fourth International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2015. Citations on pages 60 and 62.

DONAHUE, J.; JIA, Y.; VINYALS, O.; HOFFMAN, J.; ZHANG, N.; TZENG, E.; DARRELL, T. Decaf: A deep convolutional activation feature for generic visual recognition. In: **International conference on machine learning**. [S.l.: s.n.], 2014. p. 647–655. Citation on page 35.

DONG, S.; WANG, P.; ABBAS, K. A survey on deep learning and its applications. **Computer Science Review**, Elsevier, v. 40, p. 100379, 2021. Citations on pages 33, 36, 42, and 45.

DONG, X.; YANG, Y. Searching for a robust neural architecture in four gpu hours. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2019. p. 1761–1770. Citation on page 60.

DONG, X.; YANG., Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. **arXiv preprint arXiv:2001.00326**, 2020. Citation on page 116.

DU, S. S.; KOUSHIK, J.; SINGH, A.; PÓCZOS, B. Hypothesis transfer learning via transformation functions. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2017. p. 574–584. Citation on page 69.

DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern Classification (2Nd Edition)**. [S.l.]: Wiley-Interscience, 2000. ISBN 0471056693. Citations on pages 45, 47, and 114.

DUDZIAK, L.; CHAU, T.; ABDELFATTAH, M.; LEE, R.; KIM, H.; LANE, N. Brp-nas: Prediction-based nas using gcns. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2020. v. 33, p. 10480–10490. Citations on pages 62, 108, 109, 126, and 127.

DUNN, J. C. Well-separated clusters and optimal fuzzy partitions. **Journal of cybernetics**, Taylor & Francis, v. 4, n. 1, p. 95–104, 1974. Citation on page 66.

ELSKEN, T.; METZEN, J.; HUTTER, F. Neural architecture search: A survey. **The Journal of Machine Learning Research**, JMLR. org, v. 20, n. 1, p. 1997–2017, 2019. Citations on pages 35, 52, 58, 59, and 61.

ELSKEN, T.; METZEN, J. H.; HUTTER, F. Efficient multi-objective neural architecture search via lamarckian evolution. **arXiv preprint arXiv:1804.09081**, 2018. Citations on pages 34 and 60.

ENGELS, R.; THEUSINGER, C. Using a data metric for preprocessing advice for data mining applications. In: **ECAI**. [S.l.: s.n.], 1998. p. 430–434. Citation on page 66.

EVERINGHAM, M.; WINN, J. The pascal visual object classes challenge 2012 (voc2012) development kit. **Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep**, v. 8, n. 5, p. 2–5, 2011. Citation on page 174.

EVGENIOU, T.; MICCHELLI, C. A.; PONTIL, M. Learning multiple tasks with kernel methods. **Journal of Machine Learning Research**, v. 6, n. Apr, p. 615–637, 2005. Citation on page 67.

FALKNER, S.; KLEIN, A.; HUTTER, F. Bohb: Robust and efficient hyperparameter optimization at scale. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2018. p. 1437–1446. Citations on pages 108, 109, 120, and 125.

FILCHENKOV, A.; PENDRYAK, A. Datasets meta-feature description for recommending feature selection algorithm. In: IEEE. **Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT), 2015**. [S.l.], 2015. p. 11–18. Citations on pages 88 and 92.

FINN, C.; ABBEEL, P.; LEVINE, S. Model-agnostic meta-learning for fast adaptation of deep networks. In: JMLR. ORG. **Proceedings of the 34th International Conference on Machine Learning-Volume 70**. [S.l.], 2017. p. 1126–1135. Citations on pages 64 and 68.

FINN, C.; LEVINE, S. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. **arXiv preprint arXiv:1710.11622**, 2017. Citations on pages 65 and 68.

FISHER, R. A.; MACKENZIE, W. A. Studies in crop variation. ii. the manurial response of different potato varieties. **The Journal of Agricultural Science**, Cambridge University Press, v. 13, n. 3, p. 311–320, 1923. Citation on page 80.

FLACH, P. **Machine learning: the art and science of algorithms that make sense of data**. [S.l.]: Cambridge University Press, 2012. Citations on pages 88 and 89.

FORSYTH, D. A.; PONCE, J. **Computer vision: a modern approach**. [S.l.]: Prentice Hall Professional Technical Reference, 2002. Citations on pages 44, 46, and 49.

FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. **Journal of the american statistical association**, Taylor & Francis, v. 32, n. 200, p. 675–701, 1937. Citations on pages 97 and 144.

GAMA, J.; CARVALHO, A. C. P. L. F.; FACELI, K.; LORENA, A. **Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina**. [S.l.]: LTC, 2011. Citations on pages 42, 50, 88, and 114.

GIRAUD-CARRIER, C.; PROVOST, F. Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper. In: **Proceedings of the ICML-2005 Workshop on Meta-learning**. [S.l.: s.n.], 2005. p. 12–19. Citation on page 44.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: MIT press, 2016. Citations on pages 33, 36, 48, 50, 51, 53, and 78.

GUYON, I.; ELISSEEFF, A. An introduction to variable and feature selection. **Journal of machine learning research**, v. 3, n. Mar, p. 1157–1182, 2003. Citations on pages 88 and 89.

GUYON, I.; GUNN, S.; NIKRAVESH, M.; ZADEH, L. A. **Feature extraction: foundations and applications**. [S.l.]: Springer, 2008. Citation on page 88.

HAN, J.; PEI, J.; KAMBER, M. **Data mining: concepts and techniques**. [S.l.]: Elsevier, 2011. Citation on page 97.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. H.; FRIEDMAN, J. H. **The elements of statistical learning: data mining, inference, and prediction**. [S.l.]: Springer, 2009. Citations on pages 42, 43, 50, 51, and 114.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0132733501. Citations on pages 33, 48, and 49.

HAYKIN., S. **Neural Networks and Learning Machines, 3/E**. [S.l.]: Pearson Education India, 2010. Citations on pages 19, 48, 49, 50, 51, 53, and 80.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778. Citations on pages 33 and 45.

HE, Y.; YANG, D.; ROTH, H.; ZHAO, C.; XU, D. Dints: Differentiable neural network topology search for 3d medical image segmentation. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2021. p. 5841–5850. Citations on pages 20, 138, 139, 140, 143, 145, 146, 149, 150, 151, 155, 161, and 163.

HUANG, G.; LIU, Z.; MAATEN, L. V. D.; WEINBERGER, K. Q. Densely connected convolutional networks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 4700–4708. Citation on page 33.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated Machine Learning-Methods, Systems, Challenges**. [S.l.]: Springer, 2019. Citations on pages 19, 33, 56, 57, 58, 65, 66, 67, 69, 108, and 134.

ISENSEE, F.; JÄGER, P. F.; KOHL, S. A.; PETERSEN, J.; MAIER-HEIN, K. H. Automated design of deep learning methods for biomedical image segmentation. **arXiv preprint arXiv:1904.08128**, 2019. Citations on pages 36, 46, 138, 149, and 161.

JOLLIFFE, I. T.; CADIMA, J. Principal component analysis: a review and recent developments. **Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences**, The Royal Society Publishing, v. 374, n. 2065, p. 20150202, 2016. Citation on page 92.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015. Citation on page 42.

JR, R. G. M. **Beyond ANOVA: basics of applied statistics**. [S.l.]: CRC press, 1997. Citation on page 91.

KALOUSIS, A.; THEOHARIS, T. Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection. **Intelligent Data Analysis**, IOS Press, v. 3, n. 5, p. 319–337, 1999. Citation on page 66.

KANDA, J. Y.; CARVALHO, A. C. de; HRUSCHKA, E. R.; SOARES, C. Using meta-learning to recommend meta-heuristics for the traveling salesman problem. In: IEEE. **2011 10th International Conference on Machine Learning and Applications and Workshops**. [S.l.], 2011. v. 1, p. 346–351. Citation on page 67.

KANDASAMY, K.; NEISWANGER, W.; SCHNEIDER, J.; POCZOS, B.; XING, E. P. Neural architecture search with bayesian optimisation and optimal transport. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2018. p. 2016–2025. Citations on pages 34 and 60.

KIM, S.; KIM, I.; LIM, S.; BAEK, W.; KIM, C.; CHO, H.; YOON, B.; KIM, T. Scalable neural architecture search for 3d medical image segmentation. In: SPRINGER. **MICCAI**. [S.l.], 2019. p. 220–228. Citations on pages 46, 139, 149, 161, and 163.

KLEIN, A.; FALKNER, S.; BARTELS, S.; HENNIG, P.; HUTTER, F. Fast bayesian optimization of machine learning hyperparameters on large datasets. **arXiv preprint arXiv:1605.07079**, 2016. Citation on page 61.

KLEIN, A.; FALKNER, S.; SPRINGENBERG, J. T.; HUTTER, F. Learning curve prediction with bayesian neural networks. 2016. Citation on page 62.

KOCH, G.; ZEMEL, R.; SALAKHUTDINOV, R. *et al.* Siamese neural networks for one-shot image recognition. In: LILLE. **ICML deep learning workshop**. [S.l.], 2015. v. 2, n. 1. Citation on page 68.

KOHAVI, R. *et al.* A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA. **Ijcai**. [S.l.], 1995. v. 14, n. 2, p. 1137–1145. Citation on page 90.

KRIZHEVSKY, A.; HINTON, G. *et al.* Learning multiple layers of features from tiny images. Citeseer, 2009. Citations on pages 52 and 115.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105. Citations on pages 35, 45, and 61.

KUMAGAI, W. Learning bound for parameter transfer learning. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2016. p. 2721–2729. Citation on page 69.

LAKE, B. M.; ULLMAN, T. D.; TENENBAUM, J. B.; GERSHMAN, S. J. Building machines that learn and think like people. **Behavioral and brain sciences**, Cambridge University Press, v. 40, 2017. Citations on pages 64 and 65.

LAZAR, C.; TAMINAU, J.; MEGANCK, S.; STEENHOFF, D.; COLETTA, A.; MOLTER, C.; SCHAETZEN, V. de; DUQUE, R.; BERSINI, H.; NOWE, A. A survey on filter techniques for feature selection in gene expression microarray analysis. **IEEE/ACM Transactions on Computational Biology and Bioinformatics**, IEEE, v. 9, n. 4, p. 1106–1119, 2012. Citation on page 88.

LECOUTRE, C. **Constraint Networks: Targeting Simplicity for Techniques and Algorithms**. [S.l.]: John Wiley & Sons, 2013. Citation on page 79.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015. Citations on pages 33, 45, 49, and 51.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. **Neural computation**, MIT Press, v. 1, n. 4, p. 541–551, 1989. Citations on pages 33, 52, and 59.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Ieee, v. 86, n. 11, p. 2278–2324, 1998. Citations on pages 19, 50, 52, and 54.

LEE, H.; HYUNG, E.; HWANG, S. J. Rapid neural architecture search by learning to generate graphs from datasets. **arXiv:2107.00860**, 2021. Citations on pages 110 and 127.

LEIBNIZ, G. W. **The early mathematical manuscripts of Leibniz**. [S.l.]: Courier Corporation, 2012. Citation on page 51.

LEITE, R.; BRAZDIL, P. Active testing strategy to predict the best classification algorithm via sampling and metalearning. In: **ECAI**. [S.l.: s.n.], 2010. p. 309–314. Citations on pages 71, 72, and 143.

LEMKE, C.; BUDKA, M.; GABRYS, B. Metalearning: a survey of trends and technologies. **Artificial intelligence review**, Springer, v. 44, n. 1, p. 117–130, 2015. Citation on page 43.

LI, J.; CHEN, J.; TANG, Y.; WANG, C.; LANDMAN, B. A.; ZHOU, S. K. Transforming medical imaging with transformers? a comparative review of key properties, current progresses, and future perspectives. **Medical image analysis**, Elsevier, p. 102762, 2023. Citation on page 148.

LI, L.; TALWALKAR, A. Random search and reproducibility for neural architecture search. In: PMLR. **Uncertainty in Artificial Intelligence**. [S.l.], 2020. p. 367–377. Citations on pages 59 and 109.

LI, Y.; HAO, C.; LI, P.; XIONG, J.; CHEN, D. Generic neural architecture search via regression. **Advances in Neural Information Processing Systems**, v. 34, p. 20476–20490, 2021. Citations on pages 35, 62, 108, 109, 110, 120, 125, and 127.

LI, Y.; VINYALS, O.; DYER, C.; PASCANU, R.; BATTAGLIA, P. W. Learning deep generative models of graphs. **CoRR**, abs/1803.03324, 2018. Available: <http://arxiv.org/abs/1803.03324>. Citation on page 108.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. In: SPRINGER. **Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13**. [S.l.], 2014. p. 740–755. Citation on page 174.

LINNAINMAA, S. **The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors**. Phd Thesis (PhD Thesis) — Master's Thesis (in Finnish), Univ. Helsinki, 1970. Citation on page 51.

LIPTON, Z. C. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. In: ACM NEW YORK, NY, USA. **Commun. ACM**. [S.l.], 2018. p. 31–57. Citations on pages 121 and 130.

LIU, C.; CHEN, L.-C.; SCHROFF, F.; ADAM, H.; HUA, W.; YUILLE, A. L.; FEI-FEI, L. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In: **CVPR**. [S.l.: s.n.], 2019. p. 82–92. Citations on pages 36, 57, 143, and 146.

LIU, C.; ZOPH, B.; NEUMANN, M.; SHLENS, J.; HUA, W.; LI, L.-J.; FEI-FEI, L.; YUILLE, A.; HUANG, J.; MURPHY, K. Progressive neural architecture search. In: **Proceedings of the European Conference on Computer Vision (ECCV)**. [S.l.: s.n.], 2018. p. 19–34. Citations on pages 34, 44, 56, 57, and 62.

LIU, H.; SIMONYAN, K.; VINYALS, O.; FERNANDO, C.; KAVUKCUOGLU, K. Hierarchical representations for efficient architecture search. **arXiv preprint arXiv:1711.00436**, 2017. Citations on pages 56, 59, and 60.

LIU, H.; SIMONYAN, K.; YANG, Y. Darts: Differentiable architecture search. In: **International Conference on Learning Representations**. [S.l.: s.n.], 2018. p. 1–12. Citations on pages 35, 60, 62, 108, 120, 125, 143, and 146.

LIU, P.; WU, B.; MA, H.; SEOK, M. Memnas: Memory-efficient neural architecture search with grow-trim learning. In: **CVPR**. [S.l.: s.n.], 2020. p. 2108–2116. Citation on page 57.

LIU, Y.; TANG, Y.; SUN, Y. Homogeneous architecture augmentation for neural predictor. In: **Proceedings of the IEEE/CVF International Conference on Computer Vision**. [S.l.: s.n.], 2021. p. 12249–12258. Citations on pages 63, 109, and 128.

LOMAX, R. G.; HAHS-VAUGHN, D. L. **Statistical Concepts-A Second Course**. [S.l.]: Routledge, 2013. Citation on page 91.

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 3431–3440. Citation on page 55.

LORENA, A. C.; GARCIA, L. P.; LEHMANN, J.; SOUTO, M. C.; HO, T. K. How complex is your classification problem? a survey on measuring classification complexity. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 52, n. 5, p. 1–34, 2019. Citations on pages 66 and 174.

LOUPPE, G. **Understanding Random Forests: From Theory to Practice**. Phd Thesis (PhD Thesis) — University of Liège, Belgium, 2014. Available: <https://orbi.uliege.be/handle/2268/170309>. Citation on page 121.

LUKASIK, J.; FRIEDE, D.; ZELA, A.; HUTTER, F.; KEUPER, M. Smooth variational graph embeddings for efficient neural architecture search. In: IEEE. **International Joint Conference on Neural Networks**. [S.l.], 2021. p. 1–10. Citations on pages 108, 120, and 127.

LUO, R.; TIAN, F.; QIN, T.; CHEN, E.; LIU, T.-Y. Neural architecture optimization. In: **NeurIPS**. [S.l.: s.n.], 2018. p. 1–12. Citations on pages 34, 57, 58, 108, 120, and 125.

MAATEN, L. Van der; HINTON, G. Visualizing data using t-sne. **Journal of machine learning research**, v. 9, n. 11, 2008. Citation on page 174.

MANTOVANI, R. G.; ROSSI, A. L.; VANSCHOREN, J.; BISCHL, B.; CARVALHO, A. C. To tune or not to tune: recommending when to adjust svm hyper-parameters via meta-learning. In: IEEE. **Neural Networks (IJCNN), 2015 International Joint Conference on**. [S.l.], 2015. p. 1–8. Citation on page 67.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943. Citation on page 49.

MCINNES, L.; HEALY, J.; MELVILLE, J. Umap: Uniform manifold approximation and projection for dimension reduction. **arXiv preprint arXiv:1802.03426**, 2018. Citation on page 174.

MCLACHLAN, G. J. **Discriminant Analysis and Statistical Pattern Recognition**. Newark, NJ: Wiley, 2005. Citations on pages 43 and 144.

MELLOR, J.; TURNER, J.; STORKEY, A.; CROWLEY, E. J. Neural architecture search without training. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2021. p. 7588–7598. Citations on pages 34, 63, 108, 120, and 125.

MENZE, B. H.; JAKAB, A.; BAUER, S.; KALPATHY-CRAMER, J.; FARAHANI, K.; KIRBY, J.; BURREN, Y.; PORZ, N.; SLOTBOOM, J.; WIEST, R. *et al.* The multimodal brain tumor image segmentation benchmark (brats). **IEEE transactions on medical imaging**, IEEE, v. 34, n. 10, p. 1993–2024, 2014. Citations on pages 19, 46, 47, and 148.

MICHIE, D.; SPIEGELHALTER, D. J.; TAYLOR, C. C.; CAMPBELL, J. (Ed.). **Machine Learning, Neural and Statistical Classification**. USA: Ellis Horwood, 1995. ISBN 013106360X. Citations on pages 42 and 66.

MIKA, S.; RATSCH, G.; WESTON, J.; SCHOLKOPF, B.; MULLERS, K.-R. Fisher discriminant analysis with kernels. In: IEEE. **Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468)**. [S.l.], 1999. p. 41–48. Citation on page 173.

MILLER, G. F.; TODD, P. M.; HEGDE, S. U. Designing neural networks using genetic algorithms. In: **ICGA**. [S.l.: s.n.], 1989. v. 89, p. 379–384. Citation on page 59.

MINSKY, M.; PAPERT, S. Perceptrons. MIT Press, 1969. Citation on page 50.

MISIR, M.; SEBAG, M. Alors: An algorithm recommender system. **Artificial Intelligence**, Elsevier, v. 244, p. 291–314, Mar. 2017. Available: <https://hal.inria.fr/hal-01419874>. Citation on page 79.

MITCHELL, T. M. Machine learning. 1997. **Burr Ridge, IL: McGraw Hill**, v. 45, n. 37, p. 870–877, 1997. Citations on pages 41, 42, and 114.

MOLNAR, C. Interpretable machine learning. In: **.** [S.l.: s.n.], 2020. p. 1–15. Citations on pages 35, 88, and 130.

MUÑOZ, M. A.; VILLANOVA, L.; BAATAR, D.; SMITH-MILES, K. Instance spaces for machine learning classification. In: **Machine Learning**. [S.l.: s.n.], 2018. p. 1–20. Citation on page 126.

NEMENYI, P. Distribution-free multiple comparisons. In: INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210. **Biometrics**. [S.l.], 1962. v. 18, n. 2, p. 263. Citation on page 97.

NICHOL, A.; ACHIAM, J.; SCHULMAN, J. On first-order meta-learning algorithms. **arXiv preprint arXiv:1803.02999**, 2018. Citation on page 68.

NIELSEN, M. A. **Neural networks and deep learning**. [S.l.]: Determination press San Francisco, CA, 2015. Citations on pages 48, 50, 52, and 53.

NING, X.; ZHENG, Y.; ZHAO, T.; WANG, Y.; YANG, H. A generic graph-based neural architecture encoding scheme for predictor-based nas. In: SPRINGER. **Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII**. [S.l.], 2020. p. 189–204. Citations on pages 63 and 109.

OKTAY, O.; SCHLEMPER, J.; FOLGOC, L. L.; LEE, M.; HEINRICH, M.; MISAWA, K.; MORI, K.; MCDONAGH, S.; HAMMERLA, N. Y.; KAINZ, B. *et al.* Attention u-net: Learning where to look for the pancreas. **arXiv preprint arXiv:1804.03999**, 2018. Citations on pages 139, 149, and 163.

OLSON, R. S.; MOORE, J. H. Tpot: A tree-based pipeline optimization tool for automating machine learning. In: **Workshop on Automatic Machine Learning**. [S.l.: s.n.], 2016. p. 66–74. Citation on page 96.

OLSON, R. S.; URBANOWICZ, R. J.; ANDREWS, P. C.; LAVENDER, N. A.; KIDD, L. C.; MOORE, J. H. Automating biomedical data science through tree-based pipeline optimization. In: SQUILLERO, G.; BURELLI, P. (Ed.). **Applications of Evolutionary Computation**. Cham: Springer International Publishing, 2016. p. 123–137. ISBN 978-3-319-31204-0. Citation on page 47.

PAN, S. J.; YANG, Q. A survey on transfer learning. **IEEE Transactions on knowledge and data engineering**, IEEE, v. 22, n. 10, p. 1345–1359, 2009. Citations on pages 69 and 70.

PARMEZAN, A. R. S.; LEE, H. D.; WU, F. C. Metalearning for choosing feature selection algorithms in data mining: Proposal of a new framework. **Expert Systems with Applications**, Elsevier, v. 75, p. 1–24, 2017. Citations on pages 88 and 90.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KOPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In: **33rd Conference on Neural Information Processing Systems (NeurIPS)**. [S.l.: s.n.], 2019. p. 8024–8035. Citations on pages 121 and 151.

PAVEL, Y. P. P. A. F.; SOARES, B. C. Decision tree-based data characterization for meta-learning. **IDDM-2002**, p. 111, 2002. Citations on pages 65 and 66.

PEARSON, K. Liii. on lines and planes of closest fit to systems of points in space. **The London, Edinburgh, and Dublin philosophical magazine and journal of science**, Taylor & Francis, v. 2, n. 11, p. 559–572, 1901. Citation on page 92.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. Citations on pages 80, 81, 89, 98, and 121.

PEREIRA, G. T.; SANTOS, I. B.; GARCIA, L. P.; URRUTY, T.; VISANI, M.; CARVALHO, A. C. de. Neural architecture search with interpretable meta-features and fast predictors. **Information Sciences**, Elsevier, p. 119642, 2023. Citations on pages 20, 23, 25, 26, 111, 112, 113, 123, 125, 127, 128, 131, and 133.

PEREIRA, G. T.; SANTOS, M. d.; ALCOBAÇA, E.; MANTOVANI, R.; CARVALHO, A. Transfer learning for algorithm recommendation. **arXiv preprint arXiv:1910.07012**, 2019. Citations on pages 25, 82, 83, 84, and 88.

PEREIRA, G. T.; SANTOS, M. R. d.; CARVALHO, A. C. P. d. L. F. de. Evaluating meta-feature selection for the algorithm recommendation problem. **arXiv preprint arXiv:2106.03954**, 2021. Citations on pages 20, 25, 94, 95, 99, 100, 101, 102, and 103.

PERRONE, V.; JENATTON, R.; SEEGER, M. W.; ARCHAMBEAU, C. Scalable hyperparameter transfer learning. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2018. p. 6845–6855. Citations on pages 35 and 70.

PERSLEV, M.; DAM, E. B.; PAI, A.; IGEL, C. One network to segment them all: A general, lightweight system for accurate 3d medical image segmentation. In: SPRINGER. **MICCAI**. [S.l.], 2019. p. 30–38. Citations on pages 47, 138, 149, 161, and 163.

PFAHRINGER, B.; BENSUSAN, H.; GIRAUD-CARRIER, C. G. Meta-learning by landmarking various learning algorithms. In: **ICML**. [S.l.: s.n.], 2000. p. 743–750. Citation on page 67.

PHAM, H.; GUAN, M. Y.; ZOPH, B.; LE, Q. V.; DEAN, J. Efficient neural architecture search via parameter sharing. **arXiv preprint arXiv:1802.03268**, 2018. Citations on pages 59 and 62.

PRINCE, S. J. **Computer vision: models, learning, and inference**. [S.l.]: Cambridge University Press, 2012. Citations on pages 44, 46, and 47.

PROBST, P.; BISCHL, B.; BOULESTEIX, A.-L. Tunability: Importance of hyperparameters of machine learning algorithms. **arXiv preprint arXiv:1802.09596**, 2018. Citation on page 67.

RAVI, S.; LAROCHELLE, H. Optimization as a model for few-shot learning. 2016. Citations on pages 64 and 68.

RAWAL, A.; MIIKKULAINEN, R. From nodes to networks: Evolving recurrent neural networks. **arXiv preprint arXiv:1803.04439**, 2018. Citation on page 62.

REAL, E.; AGGARWAL, A.; HUANG, Y.; LE, Q. V. Regularized evolution for image classifier architecture search. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2019. v. 33, p. 4780–4789. Citations on pages 34, 56, 57, 60, 108, 109, 120, and 125.

REAL, E.; MOORE, S.; SELLE, A.; SAXENA, S.; SUEMATSU, Y. L.; TAN, J.; LE, Q. V.; KURAKIN, A. Large-scale evolution of image classifiers. In: JMLR. ORG. **Proceedings of the 34th International Conference on Machine Learning-Volume 70**. [S.l.], 2017. p. 2902–2911. Citations on pages 34, 59, and 60.

REN, P.; XIAO, Y.; CHANG, X.; HUANG, P.-Y.; LI, Z.; CHEN, X.; WANG, X. A comprehensive survey of neural architecture search: Challenges and solutions. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 4, p. 1–34, 2021. Citations on pages 36 and 53.

RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. "why should i trust you?" explaining the predictions of any classifier. In: **22nd ACM SIGKDD international conference on knowledge discovery and data mining**. [S.l.: s.n.], 2016. p. 1–14. Citations on pages 121 and 130.

RICE, J. R. The algorithm selection problem. In: **Advances in computers**. [S.l.]: Elsevier, 1976. v. 15, p. 65–118. Citations on pages 43 and 67.

RIJN, J. N. V.; ABDULRAHMAN, S. M.; BRAZDIL, P.; VANSCHOREN, J. Fast algorithm selection using learning curves. In: SPRINGER. **International symposium on intelligent data analysis**. [S.l.], 2015. p. 298–309. Citation on page 62.

RIVOLLI, A.; GARCIA, L. P.; SOARES, C.; VANSCHOREN, J.; CARVALHO, A. C. de. Characterizing classification datasets: a study of meta-features for meta-learning. **arXiv preprint arXiv:1808.10406**, 2018. Citation on page 65.

RIVOLLI., A.; GARCIA, L. P.; SOARES, C.; VANSCHOREN, J.; CARVALHO, A. C. de. Meta-features for meta-learning. **Knowledge-Based Systems**, Elsevier, v. 240, p. 108101, 2022. Citation on page 66.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: SPRINGER. **Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18**. [S.l.], 2015. p. 234–241. Citations on pages 19, 47, and 55.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citations on pages 49 and 50.

ROSENBLATT, F. *et al.* **Principles of neurodynamics: Perceptrons and the theory of brain mechanisms**. [S.l.]: Spartan books Washington, DC, 1962. Citation on page 51.

ROUSSEEUW, P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. **Journal of computational and applied mathematics**, Elsevier, v. 20, p. 53–65, 1987. Citation on page 66.

RU, B.; WAN, X.; DONG, X.; OSBORNE, M. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. **arXiv preprint arXiv:2006.07556**, 2020. Citation on page 35.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. **Learning internal representations by error propagation**. [S.l.], 1985. Citation on page 51.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. *et al.* Imagenet large scale visual recognition challenge. **International journal of computer vision**, Springer, v. 115, n. 3, p. 211–252, 2015. Citations on pages 35 and 44.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. [S.l.]: Prentice Hall, 2009. Citations on pages 42, 43, 97, and 114.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. ii—recent progress. **IBM Journal of research and development**, IBM, v. 11, n. 6, p. 601–617, 1967. Citation on page 41.

SCOTT, T.; RIDGEWAY, K.; MOZER, M. C. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2018. p. 76–85. Citation on page 70.

SETTLES, B. Active learning literature survey. University of Wisconsin-Madison Department of Computer Sciences, 2009. Citations on pages 20, 71, and 72.

SIEMS, J.; ZIMMER, L.; ZELA, A.; LUKASIK, J.; KEUPER, M.; HUTTER, F. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. **arXiv preprint arXiv:2008.09777**, 2020. Citations on pages 34 and 62.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014. Citation on page 35.

SPEARMAN, C. The proof and measurement of association between two things. In: APPLETON-CENTURY-CROFTS. **Appleton-Century-Crofts**. [S.l.], 1961. p. 18. Citation on page 121.

SUN, Y.; WANG, H.; XUE, B.; JIN, Y.; YEN, G. G.; ZHANG, M. Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. In: **Transactions on Evolutionary Computation**. [S.l.: s.n.], 2019. p. . Citations on pages 34, 108, 120, and 127.

SUNG, F.; YANG, Y.; ZHANG, L.; XIANG, T.; TORR, P. H.; HOSPEDALES, T. M. Learning to compare: Relation network for few-shot learning. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2018. p. 1199–1208. Citation on page 68.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9. Citation on page 35.

TANG, Y.; WANG, Y.; XU, Y.; CHEN, H.; SHI, B.; XU, C.; XU, C.; TIAN, Q.; XU, C. A semi-supervised assessor of neural architectures. In: **IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. p. 1–12. Citations on pages 108, 120, and 127.

TEAM, T. pandas development. **pandas-dev/pandas: Pandas**. Zenodo, 2020. Available: <https://doi.org/10.5281/zenodo.3509134>. Citations on pages 81 and 98.

THORNTON, C.; HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: ACM. **Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2013. p. 847–855. Citation on page 44.

THRUN, S.; PRATT, L. Learning to learn. Kluwer Academic Publishers, 1998. Citations on pages 35, 59, 64, and 67.

TORRALBA, A.; FERGUS, R.; FREEMAN, W. T. 80 million tiny images: A large data set for nonparametric object and scene recognition. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 30, n. 11, p. 1958–1970, 2008. Citation on page 115.

VANSCHOREN, J.; RIJN, J. N. V.; BISCHL, B.; TORGO, L. Openml: networked science in machine learning. **ACM SIGKDD Explorations Newsletter**, ACM, v. 15, n. 2, p. 49–60, 2014. Citation on page 67.

VAPNIK, V. **The nature of statistical learning theory**. [S.l.]: Springer science & business media, 1999. Citation on page 95.

VILALTA, R.; DRISSI, Y. A perspective view and survey of meta-learning. **Artif. Intell. Rev.**, Kluwer Academic Publishers, v. 18, n. 2, p. 77–95, Oct. 2002. ISSN 0269-2821. Available: <http://dx.doi.org/10.1023/A:1019956318069>. Citations on pages 43 and 64.

VINYALS, O.; BLUNDELL, C.; LILLICRAP, T.; WIERSTRA, D. *et al.* Matching networks for one shot learning. **Advances in neural information processing systems**, v. 29, 2016. Citation on page 68.

VIRTANEN, P.; GOMMERS, R.; OLIPHANT, T. E.; HABERLAND, M.; REDDY, T.; COURNAPEAU, D.; BUROVSKI, E.; PETERSON, P.; WECKESSER, W.; BRIGHT, J.; van der Walt, S. J.; BRETT, M.; WILSON, J.; MILLMAN, K. J.; MAYOROV, N.; NELSON, A. R. J.; JONES, E.; KERN, R.; LARSON, E.; CAREY, C. J.; POLAT, İ.; FENG, Y.; MOORE, E. W.; VanderPlas, J.; LAXALDE, D.; PERKTOLD, J.; CIMRMAN, R.; HENRIKSEN, I.; QUINTERO, E. A.; HARRIS, C. R.; ARCHIBALD, A. M.; RIBEIRO, A. H.; PEDREGOSA, F.; van Mulbregt, P.; SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. **Nature Methods**, v. 17, p. 261–272, 2020. Citation on page 151.

WANG, J.; WU, J.; BAI, H.; CHENG, J. M-nas: Meta neural architecture search. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2020. v. 34, n. 04, p. 6186–6193. Citations on pages 34, 109, 110, and 126.

WEERTS, H.; MEULLER, M.; VANSCHOREN, J. **Importance of tuning hyperparameters of machine learning algorithms**. [S.l.], 2018. Citation on page 67.

WEI, C.; NIU, C.; TANG, Y.; WANG, Y.; HU, H.; LIANG, J. Npenas: Neural predictor guided evolution for neural architecture search. In: IEEE. **IEEE Transactions on Neural Networks and Learning Systems**. [S.l.], 2022. p. 1–14. Citations on pages 34 and 59.

WEI, T.; WANG, C.; RUI, Y.; CHEN, C. W. Network morphism. In: **International Conference on Machine Learning**. [S.l.: s.n.], 2016. p. 564–572. Citation on page 60.

WEN, W.; LIU, H.; CHEN, Y.; LI, H.; BENDER, G.; KINDERMANS, P.-J. Neural predictor for neural architecture search. In: SPRINGER. **Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX**. [S.l.], 2020. p. 660–676. Citations on pages 20, 61, 109, 118, 119, 120, 121, 122, 125, and 127.

WENG, Y.; ZHOU, T.; LI, Y.; QIU, X. Nas-unet: Neural architecture search for medical image segmentation. **IEEE Access**, IEEE, v. 7, p. 44247–44257, 2019. Citations on pages 46 and 55.

WERBOS, P. J. Applications of advances in nonlinear sensitivity analysis. In: SPRINGER. **System Modeling and Optimization: Proceedings of the 10th IFIP Conference New York City, USA, August 31–September 4, 1981**. [S.l.], 2005. p. 762–770. Citation on page 51.

WETZEL, A. Evaluation of the effectiveness of genetic algorithms in combinatorial optimization. 1983. Citation on page 60.

WHITE, C.; SAFARI, M.; SUKTHANKER, R.; RU, B.; ELSKEN, T.; ZELA, A.; DEY, D.; HUTTER, F. Neural architecture search: Insights from 1000 papers. **arXiv preprint arXiv:2301.08727**, 2023. Citations on pages 35, 36, 46, 59, 62, and 63.

WHITE, C.; ZELA, A.; RU, B.; LIU, Y.; HUTTER, F. How powerful are performance predictors in neural architecture search? **arXiv preprint arXiv:2104.01177**, 2021. Citations on pages 34, 36, 61, 63, and 125.

WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In: SPRINGER. **Reinforcement learning**. [S.l.], 1992. Citations on pages 109, 120, and 125.

WISTUBA, M.; PEDAPATI, T. Learning to rank learning curves. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2020. p. 10303–10312. Citation on page 62.

WISTUBA, M.; RAWAT, A.; PEDAPATI, T. A survey on neural architecture search. **arXiv preprint arXiv:1905.01392**, 2019. Citations on pages 35, 46, and 58.

WITTEN, I. H.; FRANK, E.; HALL, M. A.; PAL, C. J. **Data Mining: Practical machine learning tools and techniques**. [S.l.]: Morgan Kaufmann, 2016. Citations on pages 45, 53, and 114.

WOLD, S.; ESBENSEN, K.; GELADI, P. Principal component analysis. **Chemometrics and intelligent laboratory systems**, Elsevier, v. 2, n. 1-3, p. 37–52, 1987. Citation on page 92.

WOLPERT, D. H. The lack of a priori distinctions between learning algorithms. **Neural computation**, MIT Press, v. 8, n. 7, p. 1341–1390, 1996. Citation on page 43.

WONG, C.; HOULSBY, N.; LU, Y.; GESMUNDO, A. Transfer learning with neural automl. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2018. p. 8356–8365. Citations on pages 67 and 69.

XIA, Y.; LIU, F.; YANG, D.; CAI, J.; YU, L.; ZHU, Z.; XU, D.; YUILLE, A.; ROTH, H. 3d semi-supervised learning with uncertainty-aware multi-view co-training. In: **WACV**. [S.l.: s.n.], 2020. p. 3646–3655. Citations on pages 139, 149, and 161.

XIE, S.; ZHENG, H.; LIU, C.; LIN, L. Snas: stochastic neural architecture search. **arXiv preprint arXiv:1812.09926**, 2018. Citation on page 60.

YAN, S.; ZHENG, Y.; AO, W.; ZENG, X.; ZHANG, M. Does unsupervised architecture representation learning help neural architecture search? In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2020. p. 1–10. Citation on page 125.

YAN, X.; JIANG, W.; SHI, Y.; ZHUO, C. Ms-nas: Multi-scale neural architecture search for medical image segmentation. In: SPRINGER. **International Conference on Medical Image Computing and Computer-Assisted Intervention**. [S.l.], 2020. p. 388–397. Citations on pages 47 and 55.

YANI, M. *et al.* Application of transfer learning using convolutional neural network method for early detection of terry's nail. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2019. v. 1201, n. 1, p. 012052. Citations on pages 19, 54, and 70.

YING, C.; KLEIN, A.; CHRISTIANSEN, E.; REAL, E.; MURPHY, K.; HUTTER, F. Nas-bench-101: Towards reproducible neural architecture search. In: **International Conference on Machine Learning**. [S.l.: s.n.], 2019. p. 7105–7114. Citation on page 116.

YU, Q.; YANG, D.; ROTH, H.; BAI, Y.; ZHANG, Y.; YUILLE, A. L.; XU, D. C2FNAS: Coarse-to-Fine Neural Architecture Search for 3D Medical Image Segmentation. In: **CVPR**. [S.l.: s.n.], 2020. p. 4126–4135. Citations on pages 46, 140, 149, 161, and 163.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: SPRINGER. **European conference on computer vision**. [S.l.], 2014. p. 818–833. Citation on page 33.

ZELA, A.; KLEIN, A.; FALKNER, S.; HUTTER, F. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. **arXiv preprint arXiv:1807.06906**, 2018. Citation on page 61.

ZHANG, M.; JIANG, S.; CUI, Z.; GARNETT, R.; CHEN, Y. D-vae: A variational autoencoder for directed acyclic graphs. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2019. p. 1–12. Citations on pages 108, 120, and 127.

ZHONG, Z.; YAN, J.; WU, W.; SHAO, J.; LIU, C.-L. Practical block-wise neural network architecture generation. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 2423–2432. Citation on page 57.

ZHU, Z.; LIU, C.; YANG, D.; YUILLE, A.; XU, D. V-nas: Neural architecture search for volumetric medical image segmentation. In: IEEE. **3DV**. [S.l.], 2019. p. 240–248. Citations on pages 34, 47, 146, and 155.

ZHUANG, F.; QI, Z.; DUAN, K.; XI, D.; ZHU, Y.; ZHU, H.; XIONG, H.; HE, Q. A comprehensive survey on transfer learning. **Proceedings of the IEEE**, IEEE, v. 109, n. 1, p. 43–76, 2020. Citations on pages 20, 69, and 70.

ZOPH, B.; LE, Q. V. Neural architecture search with reinforcement learning. **arXiv preprint arXiv:1611.01578**, 2016. Citations on pages 20, 34, 56, 59, and 108.

ZOPH, B.; VASUDEVAN, V.; SHLENS, J.; LE, Q. V. Learning transferable architectures for scalable image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2018. p. 8697–8710. Citations on pages 34, 56, 57, and 59.

APPENDIX

A

MBML-NAS: COMPLEMENTARY RESULTS

This Appendix presents complementary results for MbML-NAS. This includes more standalone performances, MSE and MAE of meta-predictors, runtime analysis, and additional data exploration analysis of benchmarks, datasets, meta-features, and data distributions.

A.1 More Standalone Performances

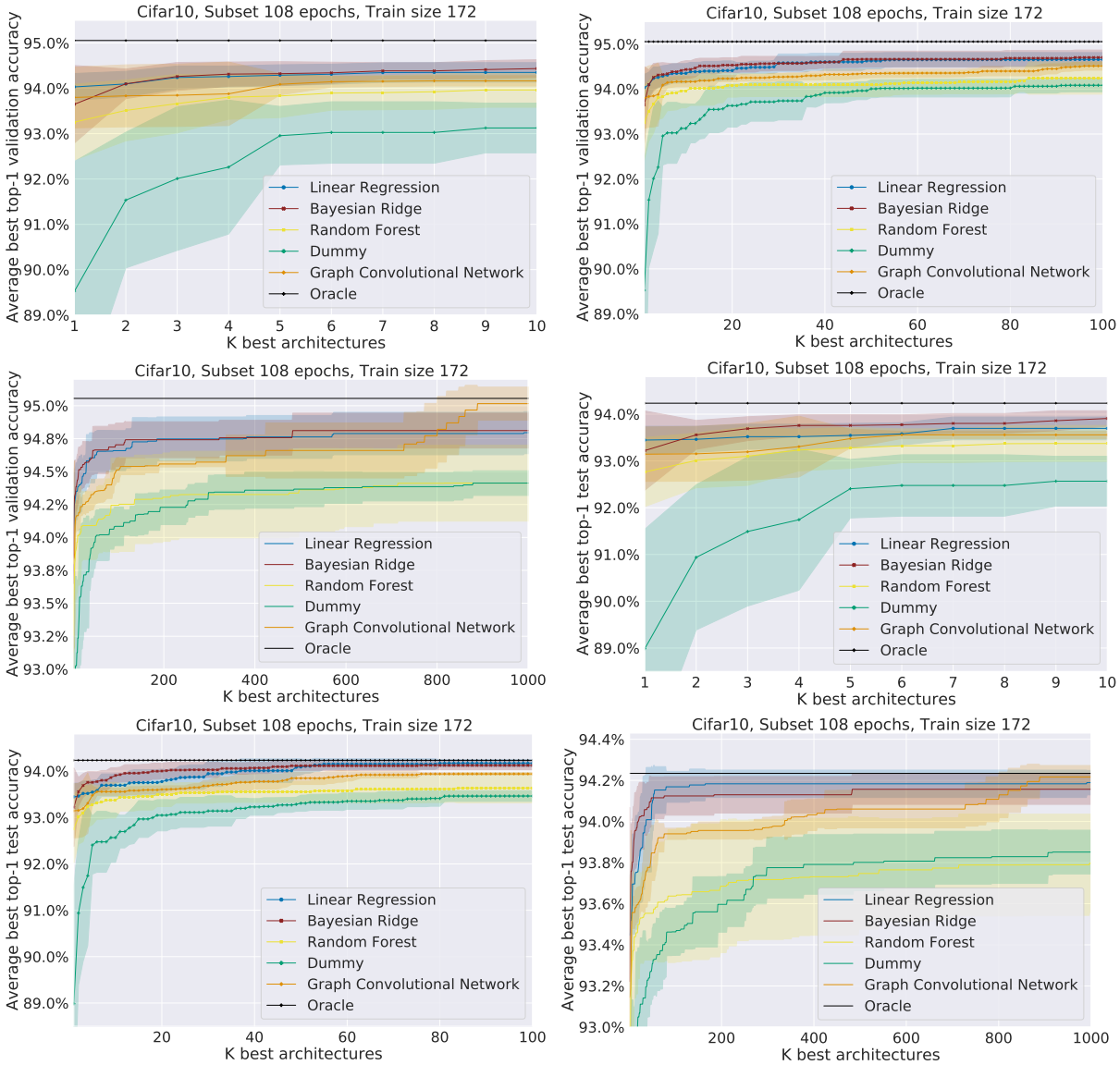


Figure 38 – Averages and Standard Deviations of the best Top-1 Validation and Test Accuracy from K best architectures selected by meta-models on NAS-Bench-101 with CIFAR-10 (108 epochs).

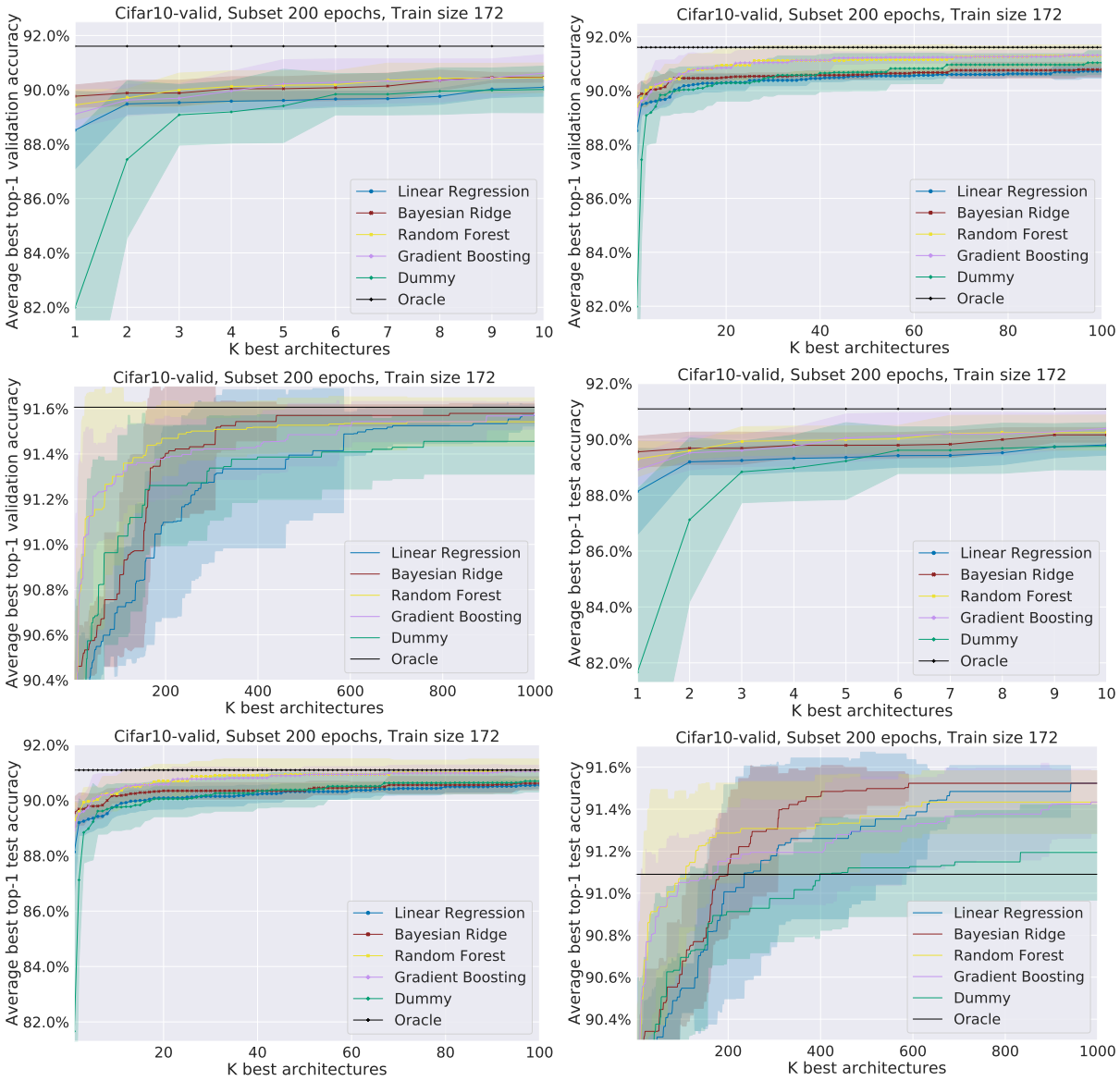


Figure 39 – Averages and Standard Deviations of the best Top-1 Validation and Test Accuracy from K best architectures selected by meta-models on NAS-Bench-201 with CIFAR-10 (200 epochs).

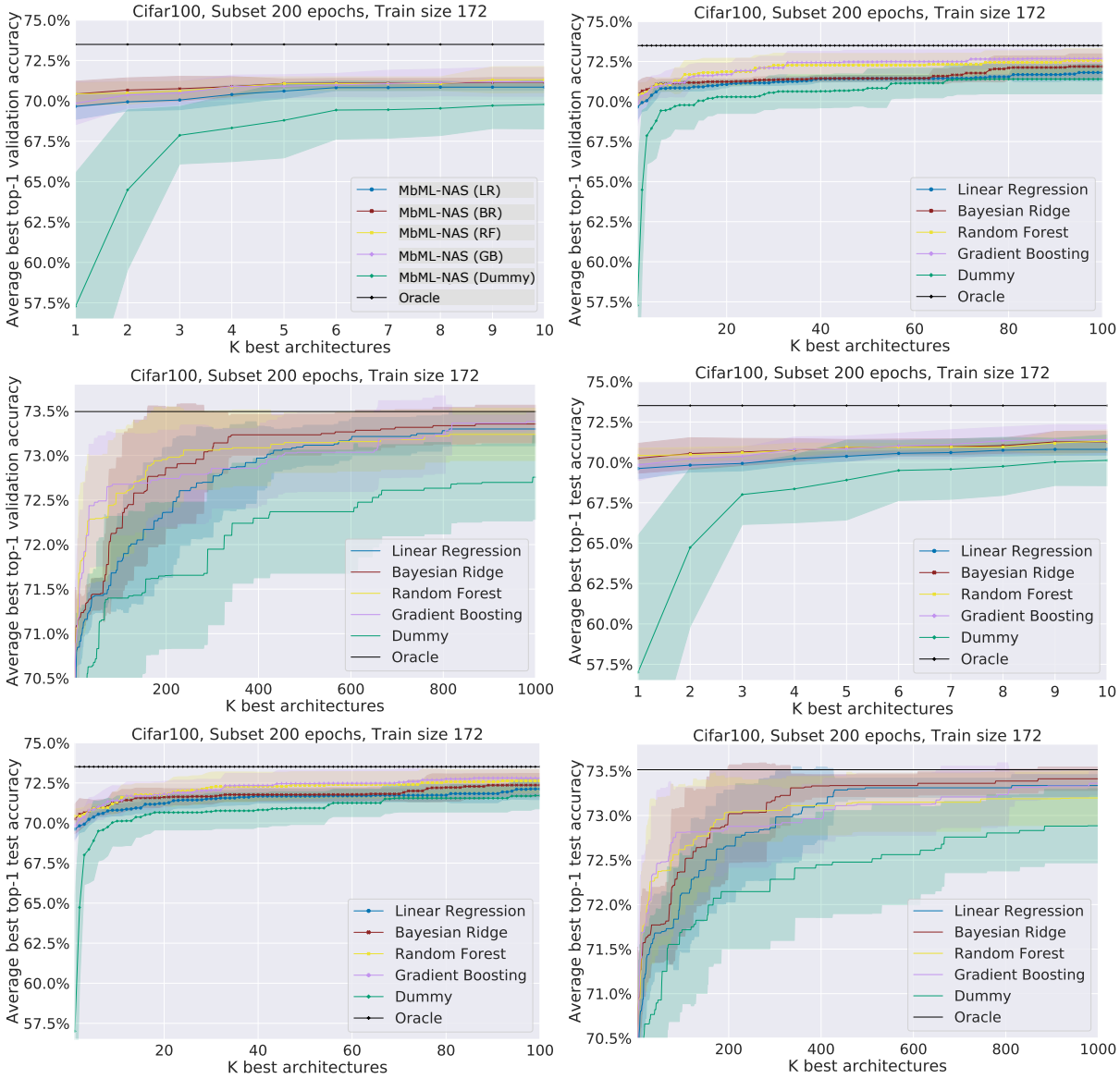


Figure 40 – Averages and Standard Deviations of the best Top-1 Validation and Test Accuracy from K best architectures selected by meta-models on NAS-Bench-201 with CIFAR-100 (200 epochs).

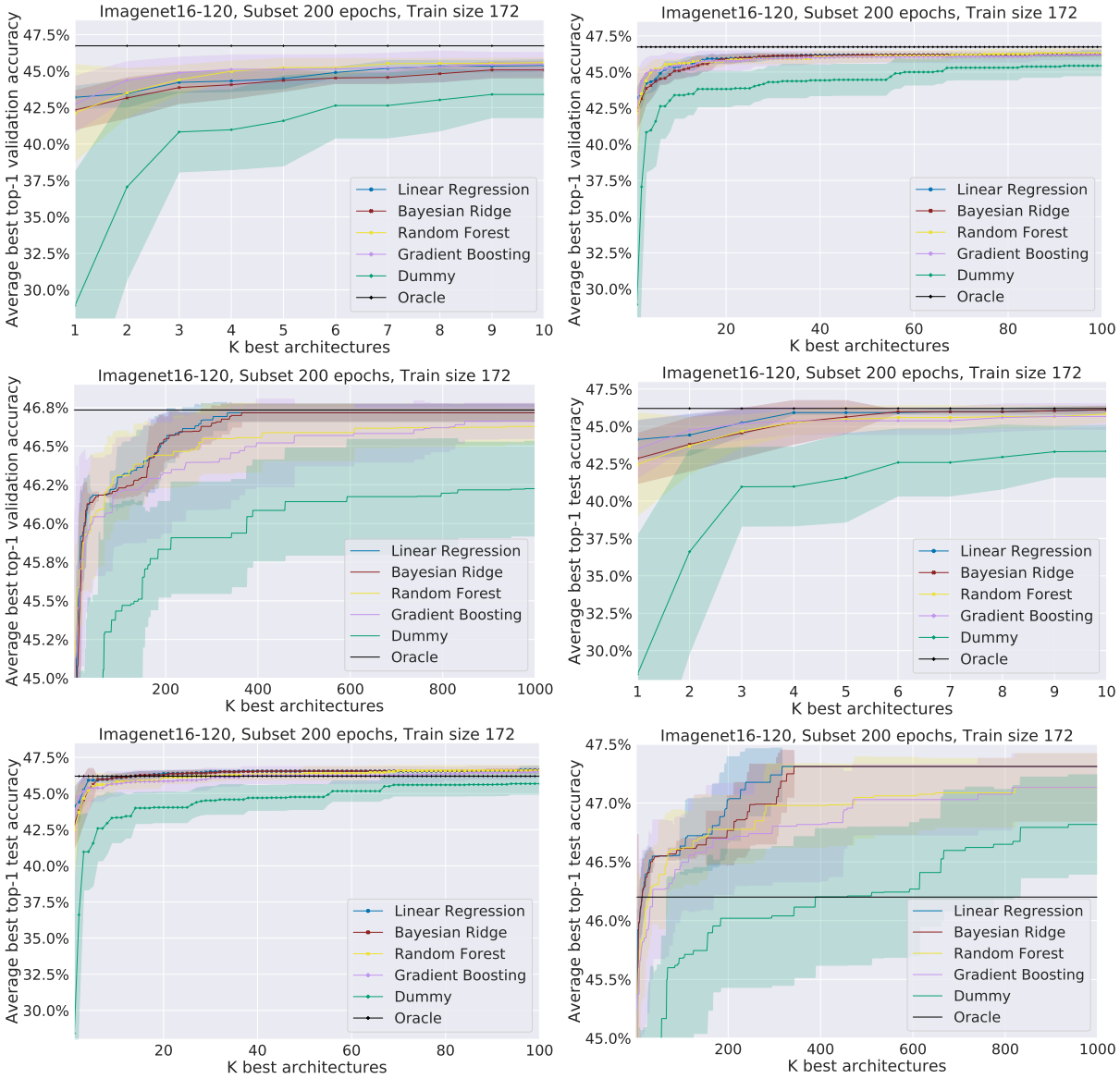


Figure 41 – Averages and Standard Deviations of the best Top-1 Validation and Test Accuracy from K best architectures selected by meta-models on NAS-Bench-201 with ImageNet16-120 (200 epochs).

A.2 MAE/MSE Comparisons and Statistical Tests

Table 23 – MAE and MSE results from pre-trained meta-predictors on CIFAR-10 dataset by 4, 12, 36, and 108 epochs, all from NAS-Bench-101.

Model	Train Size	4 epochs		12 epochs		36 epochs		108 epochs	
		MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
Dummy	43	0.1217±0.0021	0.0209±0.0008	0.1713±0.0018	0.0395±0.0006	0.0548±0.0040	0.0088±0.0002	0.0233±0.0003	0.0035±0.0001
	86	0.1232±0.0031	0.0207±0.0005	0.1707±0.0011	0.0393±0.0006	0.0548±0.0032	0.0087±0.0002	0.0233±0.0007	0.0035±0.0001
	129	0.1224±0.0025	0.0206±0.0003	0.1708±0.0013	0.0392±0.0005	0.0555±0.0024	0.0086±0.0001	0.0237±0.0009	0.0035±0.0001
	172	0.1227±0.0021	0.0206±0.0002	0.1706±0.0013	0.0391±0.0004	0.0576±0.0040	0.0086±0.0002	0.0243±0.0021	0.0035±0.0001
	344	0.1229±0.0016	0.0205±0.0001	0.1702±0.0007	0.0389±0.0002	0.0568±0.0026	0.0085±0.0001	0.0248±0.0020	0.0035±0.0001
	860	0.1228±0.0005	0.0204±0.0001	0.1699±0.0002	0.0388±0.0001	0.0562±0.0007	0.0085±0.0001	0.0245±0.0003	0.0034±0.0001
LR	43	0.1243±0.0054	0.0229±0.0023	0.1723±0.0090	0.0428±0.0060	0.0616±0.0141	0.0096±0.0035	0.0260±0.0133	0.0041±0.0028
	86	0.1203±0.0041	0.0209±0.0009	0.1644±0.0029	0.0377±0.0014	0.0533±0.0023	0.0079±0.0002	0.0218±0.0019	0.0032±0.0001
	129	0.1179±0.0028	0.0200±0.0006	0.1627±0.0013	0.0363±0.0009	0.0540±0.0031	0.0078±0.0002	0.0223±0.0018	0.0032±0.0001
	172	0.1180±0.0025	0.0199±0.0005	0.1623±0.0016	0.0360±0.0009	0.0542±0.0037	0.0078±0.0003	0.0225±0.0016	0.0032±0.0001
	344	0.1182±0.0015	0.0195±0.0002	0.1613±0.0007	0.0353±0.0003	0.0530±0.0022	0.0076±0.0001	0.0215±0.0015	0.0031±0.0001
	860	0.1178±0.0007	**0.0193±0.0001**	0.1606±0.0003	0.0349±0.0001	0.0520±0.0011	0.0075±0.0001	0.0206±0.0008	0.0031±0.0001
RF	43	0.1212±0.0021	0.0211±0.0011	0.1702±0.0035	0.0398±0.0022	0.0533±0.0043	0.0083±0.0003	0.0200±0.0013	0.0032±0.0001
	86	0.1204±0.0043	0.0208±0.0009	0.1657±0.0031	0.0375±0.0016	0.0510±0.0029	0.0080±0.0003	0.0191±0.0005	0.0031±0.0001
	129	0.1190±0.0028	0.0202±0.0004	0.1645±0.0026	0.0371±0.0011	0.0515±0.0025	0.0078±0.0002	0.0190±0.0010	0.0031±0.0001
	172	0.1193±0.0029	0.0204±0.0007	0.1632±0.0024	0.0366±0.0012	0.0525±0.0038	0.0077±0.0002	0.0188±0.0017	0.0032±0.0001
	344	0.1187±0.0016	0.0198±0.0004	0.1591±0.0011	0.0352±0.0005	0.0508±0.0031	0.0075±0.0002	0.0185±0.0015	0.0030±0.0001
	860	**0.1168±0.0010**	**0.0192±0.0002**	**0.1546±0.0010**	**0.0330±0.0004**	**0.0491±0.0019**	**0.0070±0.0002**	**0.0179±0.0018**	**0.0029±0.0001**
BR	43	0.1217±0.0017	0.0208±0.0004	0.1684±0.0030	0.0387±0.0011	0.0552±0.0056	0.0081±0.0003	0.0235±0.0089	0.0037±0.0017
	86	0.1201±0.0045	0.0200±0.0007	0.1651±0.0031	0.0371±0.0015	0.0526±0.0025	0.0079±0.0002	0.0209±0.0014	0.0032±0.0001
	129	0.1189±0.0034	0.0199±0.0004	0.1641±0.0013	0.0364±0.0006	0.0540±0.0030	0.0078±0.0002	0.0215±0.0014	0.0031±0.0001
	172	0.1188±0.0028	0.0198±0.0004	0.1634±0.0013	0.0361±0.0007	0.0543±0.0036	0.0078±0.0002	0.0220±0.0016	0.0031±0.0001
	344	0.1189±0.0017	0.0196±0.0002	0.1620±0.0009	0.0354±0.0003	0.0530±0.0022	0.0076±0.0001	0.0213±0.0014	0.0031±0.0001
	860	0.1181±0.0007	**0.0193±0.0001**	0.1610±0.0005	0.0350±0.0002	0.0520±0.0010	0.0075±0.0001	0.0206±0.0007	0.0031±0.0001
SGD	43	0.1229±0.0069	0.0220±0.0026	0.1744±0.0076	0.0433±0.0056	0.0688±0.0122	0.0112±0.0045	0.0397±0.0143	0.0063±0.0050
	86	0.1215±0.0041	0.0208±0.0010	0.1652±0.0028	0.0373±0.0015	0.0584±0.0087	0.0086±0.0008	0.0278±0.0058	0.0036±0.0004
	129	0.1184±0.0030	0.0202±0.0003	0.1650±0.0022	0.0372±0.0013	0.0567±0.0066	0.0082±0.0005	0.0287±0.0131	0.0040±0.0019
	172	0.1187±0.0030	0.0200±0.0005	0.1648±0.0027	0.0370±0.0017	0.0624±0.0098	0.0086±0.0005	0.0243±0.0036	0.0034±0.0003
	344	0.1194±0.0036	0.0199±0.0005	0.1627±0.0012	0.0359±0.0007	0.0574±0.0077	0.0080±0.0005	0.0228±0.0059	0.0033±0.0002
	860	0.1177±0.0011	0.0194±0.0001	0.1613±0.0006	0.0350±0.0001	0.0518±0.0014	0.0076±0.0002	0.0228±0.0044	0.0032±0.0001
KNN	43	0.1259±0.0087	0.0230±0.0045	0.1706±0.0023	0.0402±0.0021	0.0577±0.0084	0.0090±0.0014	0.0210±0.0015	0.0032±0.0001
	86	0.1244±0.0050	0.0214±0.0024	0.1675±0.0051	0.0392±0.0038	0.0522±0.0022	0.0081±0.0004	0.0212±0.0018	0.0033±0.0003
	129	0.1214±0.0034	0.0207±0.0011	0.1652±0.0018	0.0376±0.0012	0.0524±0.0020	0.0078±0.0002	0.0209±0.0021	0.0033±0.0004
	172	0.1212±0.0025	0.0209±0.0006	0.1646±0.0025	0.0377±0.0014	0.0551±0.0038	0.0082±0.0007	0.0213±0.0013	0.0032±0.0002
	344	0.1206±0.0013	0.0204±0.0006	0.1615±0.0009	0.0361±0.0009	0.0519±0.0023	0.0076±0.0002	0.0210±0.0015	0.0031±0.0001
	860	0.1178±0.0008	0.0198±0.0003	0.1567±0.0012	0.0344±0.0007	0.0505±0.0014	**0.0072±0.0002**	0.0200±0.0012	0.0030±0.0001
DT	43	0.1230±0.0034	0.0221±0.0016	0.1734±0.0070	0.0415±0.0038	0.0574±0.0059	0.0088±0.0006	0.0214±0.0014	0.0034±0.0002
	86	0.1246±0.0035	0.0225±0.0019	0.1686±0.0039	0.0399±0.0023	0.0524±0.0028	0.0084±0.0005	0.0202±0.0010	0.0032±0.0001
	129	0.1220±0.0040	0.0214±0.0012	0.1682±0.0037	0.0398±0.0022	0.0528±0.0038	0.0080±0.0003	0.0197±0.0010	0.0032±0.0001
	172	0.1206±0.0036	0.0210±0.0006	0.1651±0.0037	0.0378±0.0020	0.0555±0.0054	0.0083±0.0011	0.0202±0.0022	0.0032±0.0001
	344	0.1199±0.0019	0.0204±0.0004	0.1616±0.0018	0.0364±0.0006	0.0525±0.0026	0.0077±0.0002	0.0201±0.0024	0.0032±0.0004
	860	0.1184±0.0006	0.0197±0.0004	0.1581±0.0013	0.0346±0.0005	0.0513±0.0015	0.0074±0.0002	0.0185±0.0014	0.0030±0.0001
SVM	43	0.1234±0.0111	0.0228±0.0048	0.1754±0.0110	0.0445±0.0070	0.0738±0.0045	0.0101±0.0011	0.0398±0.0127	0.0046±0.0013
	86	0.1196±0.0051	0.0206±0.0015	0.1658±0.0028	0.0391±0.0019	0.0684±0.0037	0.0090±0.0004	0.0492±0.0143	0.0053±0.0012
	129	0.1201±0.0045	0.0211±0.0021	0.1649±0.0035	0.0383±0.0025	0.0702±0.0039	0.0092±0.0009	0.0576±0.0116	0.0068±0.0020
	172	0.1194±0.0039	0.0207±0.0015	0.1647±0.0029	0.0383±0.0024	0.0718±0.0050	0.0093±0.0007	0.0730±0.0662	0.0213±0.0491
	344	0.1178±0.0015	0.0198±0.0004	0.1608±0.0018	0.0361±0.0010	0.0709±0.0026	0.0089±0.0003	0.0564±0.0110	0.0057±0.0008
	860	**0.1163±0.0009**	**0.0193±0.0002**	0.1582±0.0012	0.0347±0.0006	0.0702±0.0008	0.0087±0.0001	0.0588±0.0061	0.0060±0.0007
AB	43	0.1251±0.0045	0.0237±0.0019	0.1723±0.0070	0.0421±0.0039	0.0563±0.0045	0.0092±0.0011	0.0226±0.0047	0.0038±0.0011
	86	0.1230±0.0053	0.0215±0.0009	0.1658±0.0028	0.0384±0.0023	0.0532±0.0034	0.0086±0.0011	0.0206±0.0030	0.0034±0.0007
	129	0.1214±0.0039	0.0209±0.0006	0.1636±0.0013	0.0364±0.0007	0.0532±0.0026	0.0081±0.0006	0.0215±0.0027	0.0038±0.0011
	172	0.1213±0.0047	0.0208±0.0008	0.1637±0.0023	0.0361±0.0008	0.0578±0.0077	0.0094±0.0023	0.0218±0.0035	0.0040±0.0015
	344	0.1203±0.0014	0.0200±0.0003	0.1607±0.0019	0.0350±0.0004	0.0527±0.0045	0.0077±0.0005	0.0209±0.0017	0.0037±0.0009
	860	0.1186±0.0011	**0.0194±0.0001**	0.1584±0.0007	0.0339±0.0002	0.0517±0.0011	0.0073±0.0001	0.0200±0.0008	0.0030±0.0001
GCN	43	0.1273±0.0011	0.0229±0.0003	0.1785±0.0023	0.0427±0.0007	-	-	-	-
	86	0.1403±0.0138	0.0257±0.0049	0.1779±0.0012	0.0445±0.0027	0.0552±0.0082	0.0099±0.0004	0.0296±0.0139	0.0052±0.0011
	129	0.1827±0.0128	0.0439±0.0062	0.1868±0.0126	0.0534±0.0117	0.0504±0.0018	0.0098±0.0004	0.0237±0.0021	0.0047±0.0001
	172	0.1708±0.0194	0.0381±0.0095	0.1909±0.0119	0.0573±0.0101	0.0503±0.0018	0.0101±0.0006	0.0226±0.0032	0.0046±0.0001
	344	0.1598±0.0148	0.0331±0.0049	0.1890±0.0019	0.0568±0.0016	0.0501±0.0011	0.0102±0.0008	0.0205±0.0026	0.0047±0.0001
	860	0.1667±0.0027	0.0355±0.0012	0.1916±0.0020	0.0592±0.0016	0.0508±0.0014	0.0103±0.0009	0.0202±0.0016	0.0047±0.0001

Table 24 – MAE and MSE results from pre-trained meta-predictors on CIFAR-10 by 108 and 200 epochs, CIFAR-100 for 200 epochs, and ImageNet16-120 for 200 epochs, all from NAS-Bench-201.

Model	**Train Size**	**CIFAR-10 (108 epochs)**		**CIFAR-10 (200 epochs)**		**CIFAR-100 (200 epochs)**		**ImageNet16-120 (200 epochs)**	
		MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
Dummy	43	0.1030±0.0134	3.0436±0.1643	0.0580±0.0138	1.7709±0.0578	0.0709±0.0091	1.6120±0.0728	0.0694±0.0034	0.8944±0.0512
	86	0.1059±0.0130	2.9861±0.1646	0.0602±0.0127	1.7438±0.0779	0.0732±0.0086	1.5711±0.0931	0.0697±0.0033	0.8706±0.0145
	129	0.1131±0.0067	2.8490±0.0367	0.0619±0.0074	1.6792±0.0666	0.0774±0.0069	1.5099±0.0566	0.0697±0.0027	0.8723±0.0252
	172	0.1117±0.0071	2.8523±0.0229	0.0601±0.0023	1.6481±0.0192	0.0733±0.0045	1.5038±0.0434	0.0692±0.0019	0.8630±0.0081
	344	0.1125±0.0051	2.8414±0.0189	0.0621±0.0044	1.6464±0.0143	0.0763±0.0030	1.4785±0.0109	0.0695±0.0013	0.8573±0.0038
	860	0.1128±0.0027	2.8324±0.0121	0.0625±0.0023	1.6459±0.0156	0.0760±0.0016	1.4786±0.0100	0.0696±0.0005	0.8560±0.0031
LR	43	0.0973±0.0093	2.2868±0.1826	0.0649±0.0174	1.6236±0.1576	0.0653±0.0100	1.2113±0.0930	0.0525±0.0035	0.5594±0.0250
	86	0.0924±0.0106	2.0387±0.1671	0.0630±0.0181	1.5042±0.2968	0.0622±0.0117	1.1180±0.1842	0.0498±0.0039	0.5096±0.0419
	129	0.0926±0.0078	2.0117±0.0870	0.0622±0.0107	1.4311±0.0955	0.0614±0.0068	1.0745±0.0615	0.0497±0.0030	0.5018±0.0177
	172	0.0906±0.0061	1.9558±0.0505	0.0592±0.0079	1.3878±0.0490	0.0593±0.0048	1.0414±0.0332	0.0489±0.0025	0.4908±0.0100
	344	0.0889±0.0034	1.9142±0.0229	0.0567±0.0045	1.3443±0.0172	0.0574±0.0027	1.0078±0.0136	0.0479±0.0016	0.4742±0.0047
	860	0.0879±0.0015	1.8829±0.0069	0.0536±0.0023	1.3213±0.0122	0.0554±0.0015	0.9914±0.0077	0.0473±0.0006	**0.4674±0.0025**
RF	43	0.0857±0.0062	2.2854±0.1352	0.0478±0.0076	1.5128±0.0692	0.0537±0.0036	1.1575±0.0575	0.0504±0.0020	0.5590±0.0375
	86	0.0840±0.0078	2.0873±0.1277	0.0457±0.0100	1.4551±0.1238	0.0520±0.0069	1.0730±0.0902	0.0485±0.0030	0.5228±0.0264
	129	0.0847±0.0082	2.0760±0.1918	0.0497±0.0099	1.4221±0.0508	0.0535±0.0064	1.0737±0.0489	0.0484±0.0034	0.5196±0.0327
	172	0.0840±0.0081	1.9986±0.1851	0.0470±0.0065	1.3680±0.0476	0.0505±0.0042	1.0202±0.0422	0.0462±0.0022	0.4901±0.0168
	344	0.0794±0.0050	1.9013±0.0994	0.0476±0.0060	1.3615±0.0560	0.0507±0.0033	1.0044±0.0459	0.0462±0.0017	0.4836±0.0193
	860	**0.0788±0.0013**	**1.8055±0.0213**	**0.0482±0.0017**	**1.2931±0.0169**	**0.0510±0.0012**	**0.9594±0.0098**	**0.0455±0.0005**	**0.4640±0.0041**
BR	43	0.0936±0.0082	2.2341±0.1378	0.0581±0.0124	1.5171±0.0747	0.0599±0.0070	1.1341±0.0590	0.0511±0.0029	0.5353±0.0196
	86	0.0904±0.0091	2.0279±0.1038	0.0578±0.0150	1.4520±0.1872	0.0589±0.0095	1.0824±0.1159	0.0492±0.0035	0.4999±0.0257
	129	0.0912±0.0074	1.9888±0.0807	0.0585±0.0091	1.3993±0.0615	0.0593±0.0060	1.0529±0.0425	0.0494±0.0028	0.4950±0.0121
	172	0.0897±0.0057	1.9563±0.0572	0.0565±0.0069	1.3764±0.0384	0.0578±0.0044	1.0333±0.0284	0.0486±0.0026	0.4875±0.0086
	344	0.0886±0.0033	1.9198±0.0276	0.0554±0.0041	1.3425±0.0166	0.0567±0.0026	1.0066±0.0134	0.0479±0.0016	0.4740±0.0043
	860	0.0877±0.0015	1.8835±0.0068	0.0532±0.0022	1.3235±0.0128	0.0551±0.0014	0.9923±0.0078	0.0473±0.0006	**0.4676±0.0025**
GB	43	0.0940±0.0104	2.6830±0.3376	0.0514±0.0099	1.5830±0.0879	0.0590±0.0064	1.2990±0.2049	0.0532±0.0021	0.5930±0.0435
	86	0.0863±0.0078	2.2565±0.2505	0.0496±0.0125	1.5332±0.1753	0.0557±0.0096	1.1634±0.1318	0.0485±0.0024	0.5474±0.0482
	129	0.0841±0.0088	2.0653±0.1177	0.0490±0.0096	1.4466±0.0564	0.0517±0.0050	1.0908±0.0451	0.0477±0.0023	0.5191±0.0284
	172	0.0826±0.0089	2.0334±0.0743	0.0447±0.0062	1.4187±0.0631	0.0502±0.0039	1.0769±0.0540	0.0469±0.0025	0.5119±0.0216
	344	0.0795±0.0041	1.9510±0.1262	0.0476±0.0067	1.3884±0.0248	0.0513±0.0036	1.0294±0.0261	0.0470±0.0025	0.4967±0.0194
	860	**0.0772±0.0040**	1.8286±0.0363	0.0480±0.0048	1.3223±0.0301	**0.0494±0.0023**	0.9757±0.0282	**0.0446±0.0012**	**0.4657±0.0048**

A.3 Runtime Analysis

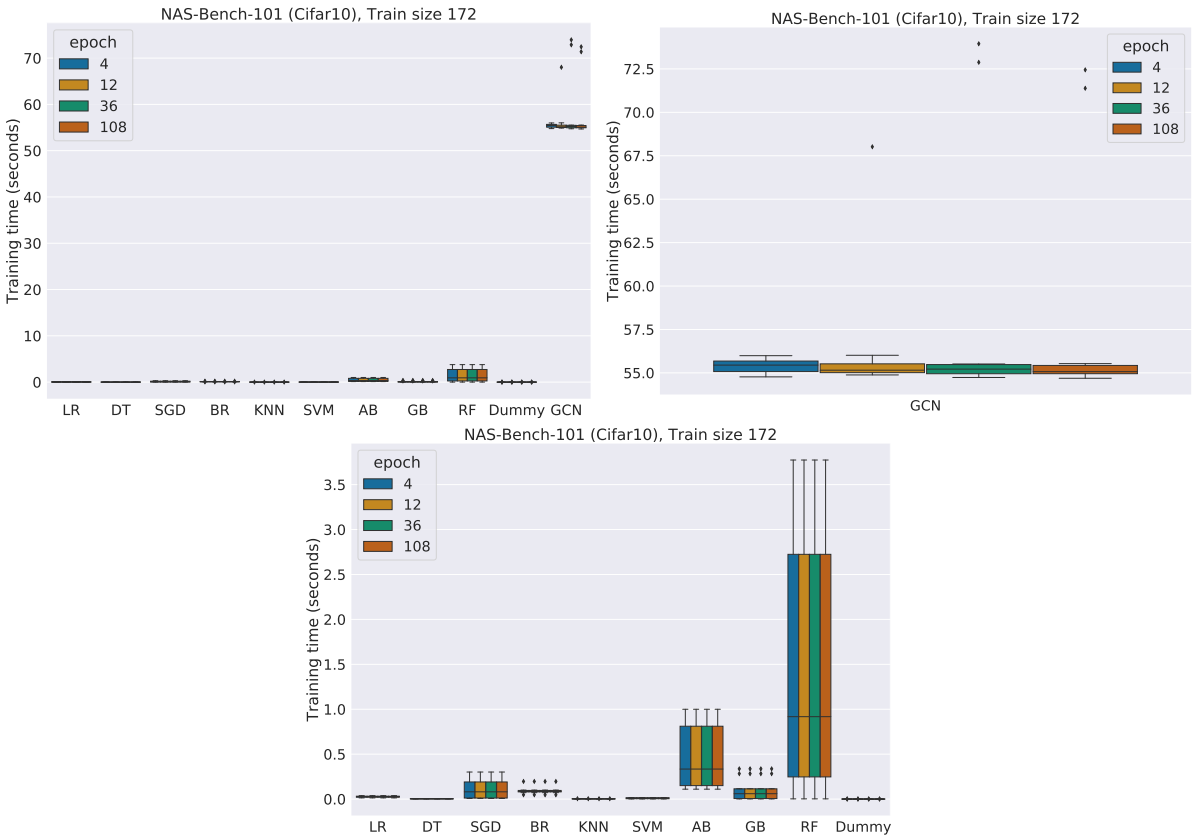


Figure 42 – Runtime comparison of all meta-predictors and Neural Predictor (GCN) on NAS-Bench-101 (CIFAR-10) subsets (epochs) from training with 172 examples.

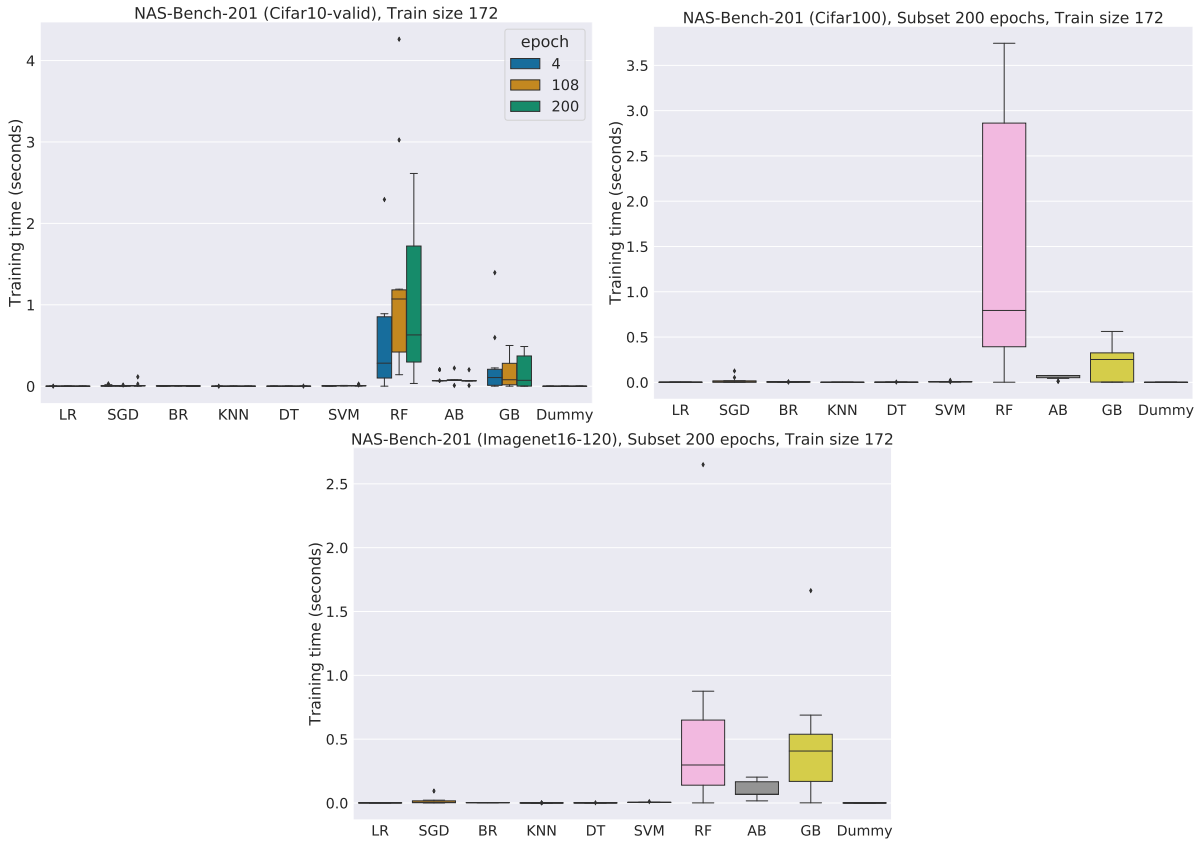


Figure 43 – Runtime comparison of all meta-predictors on NAS-Bench-201 datasets (CIFAR-10, CIFAR-100, and ImageNet16-120) and subsets (epochs) from training with 172 examples.

A.4 MbML-NAS Hyperparameter Tuning Space

Table 25 – Meta-predictors and their hyperparameters tuned via Random Search. "{}" in *Range* represents a discrete set of options, while "[]" represents a continuous inclusive interval of values.

Model	Hyperparameter	Range
Linear Regression	fit_intercept	{false, true}
	normalize	{false, true}
Bayesian Ridge	n_iter	{1000, 3000, 9000}
	tol	{0.001, 0.0001, 0.00001}
	compute_score	{false, true}
	fit_intercept	{false, true}
	normalize	{false, true}
Random Forest	n_estimators	{100, 300, 900}
	criterion	{mse, mae}
	min_samples_split	[2, 50]
	min_samples_leaf	[1, 50]
	max_features	{auto, sqrt, log2}
	oob_score	{false, true}
	warm_start	{false, true}
Dummy	strategy	{mean, median, quantile}
	quantile	{0.0, 0.25, 0.75, 1.0}
Stochastic Gradient Descent	loss	{sqrt, huber, ep_insensitive,sqrt_ep_insensitive}
	penalty	{l2, l1, elasticnet}
	fit_intercept	{false, true}
	max_iter	{1000, 3000, 9000}
	shuffle	{false, true}
	learning_rate	{constant, optimal, invscaling, adaptive}
	early_stopping	{false, true}
	n_iter_no_change	{5, 15, 45}
	warm_start	{false, true}
Decision Tree	criterion	{mse, friedman_mse, mae}
	splitter	{best, random}
	max_depth	[2, 51]
	min_samples_split	[2, 51]
	min_samples_leaf	[1, 51]
	max_features	{auto, sqrt, log2}
Gradient Boosting	loss	{ls, lad, huber, quantile}
	learning_rate	{0.1, 0.01, 0.001}
	n_estimators	{100, 300, 900}
	subsample	{0.1, 0.5, 1.0}
	criterion	{friedman_mse, mse, mae}
	min_samples_split	[2, 51]
	min_samples_leaf	[1, 51]
	max_depth	[3, 51]
	max_features	{auto, sqrt, log2}
	warm_start	{false, true}
	n_iter_no_change	{10, 30, 90, None}
Neural Predictor	classifier	{false, true}
	D	{48, 72, 96, 144, 210, 320}
	epochs	300
	weight_decay	0.001
	dropout	0.1
	train_batch_size	10
	val_acc_thld	{0.23, 0.54, 0.86, 0.91}
AdaBoost	n_estimators	{50, 150, 450}
	learning_rate	{1, 0.1, 0.01}
	loss	{linear, sqrt, exponential}

A.5 Meta-datasets' Correlations

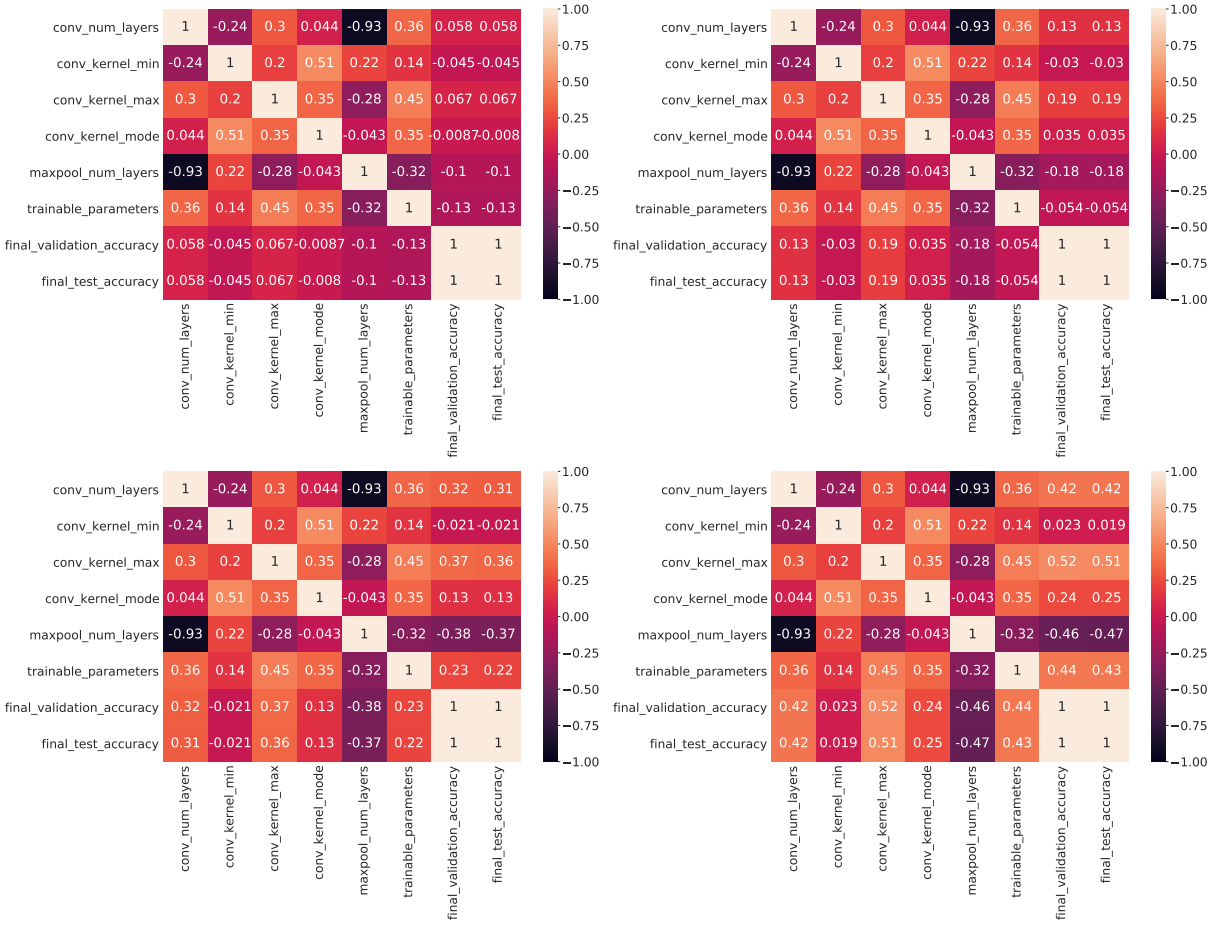


Figure 44 – Spearman Correlations of meta-features extracted from NAS-Bench-101 subsets (4, 12, 36, and 108 epochs) regarding CIFAR-10 and which were used to train and evaluate meta-predictors.

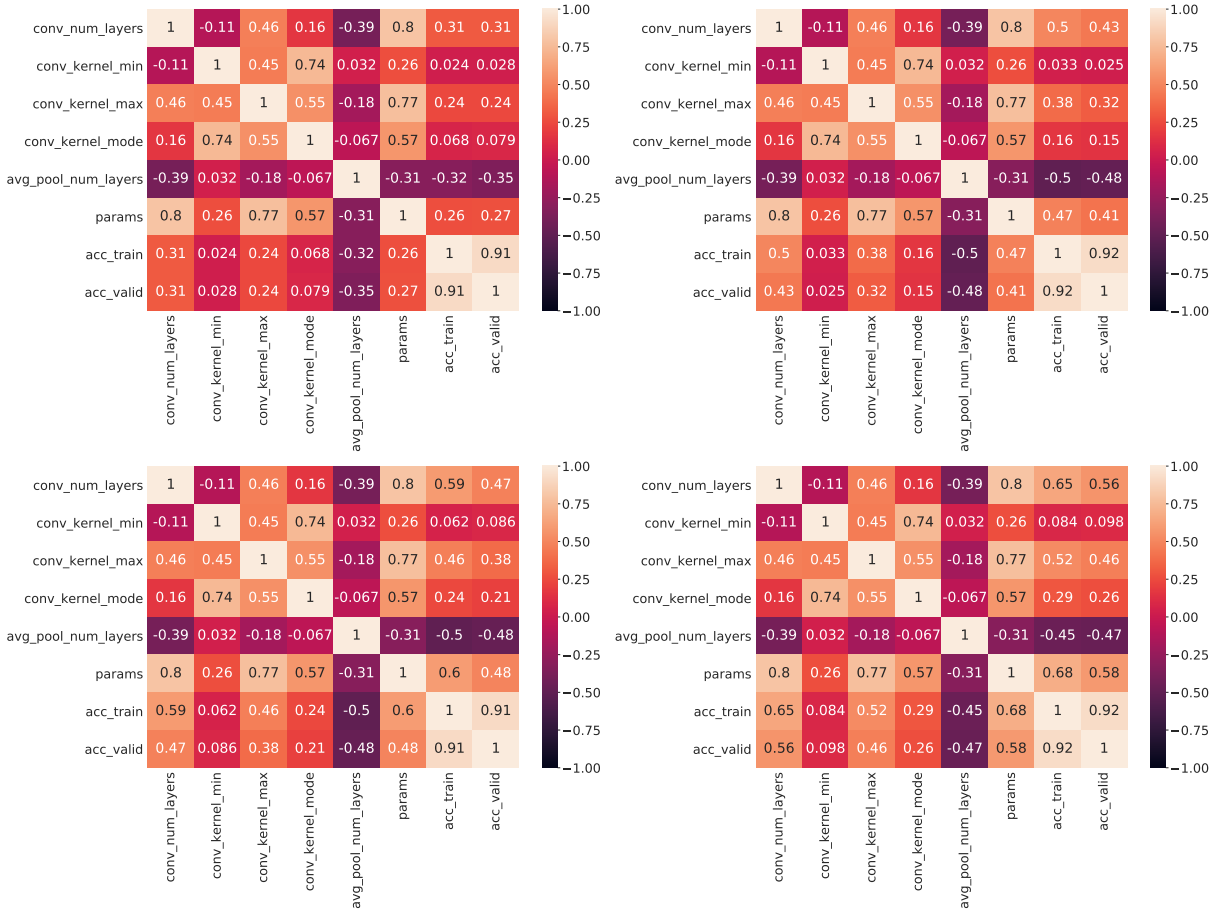


Figure 45 – Spearman Correlations of meta-features extracted from NAS-Bench-201 subsets (4, 12, 36, and 108 epochs) regarding CIFAR-10 and which were used to train and evaluate meta-predictors.

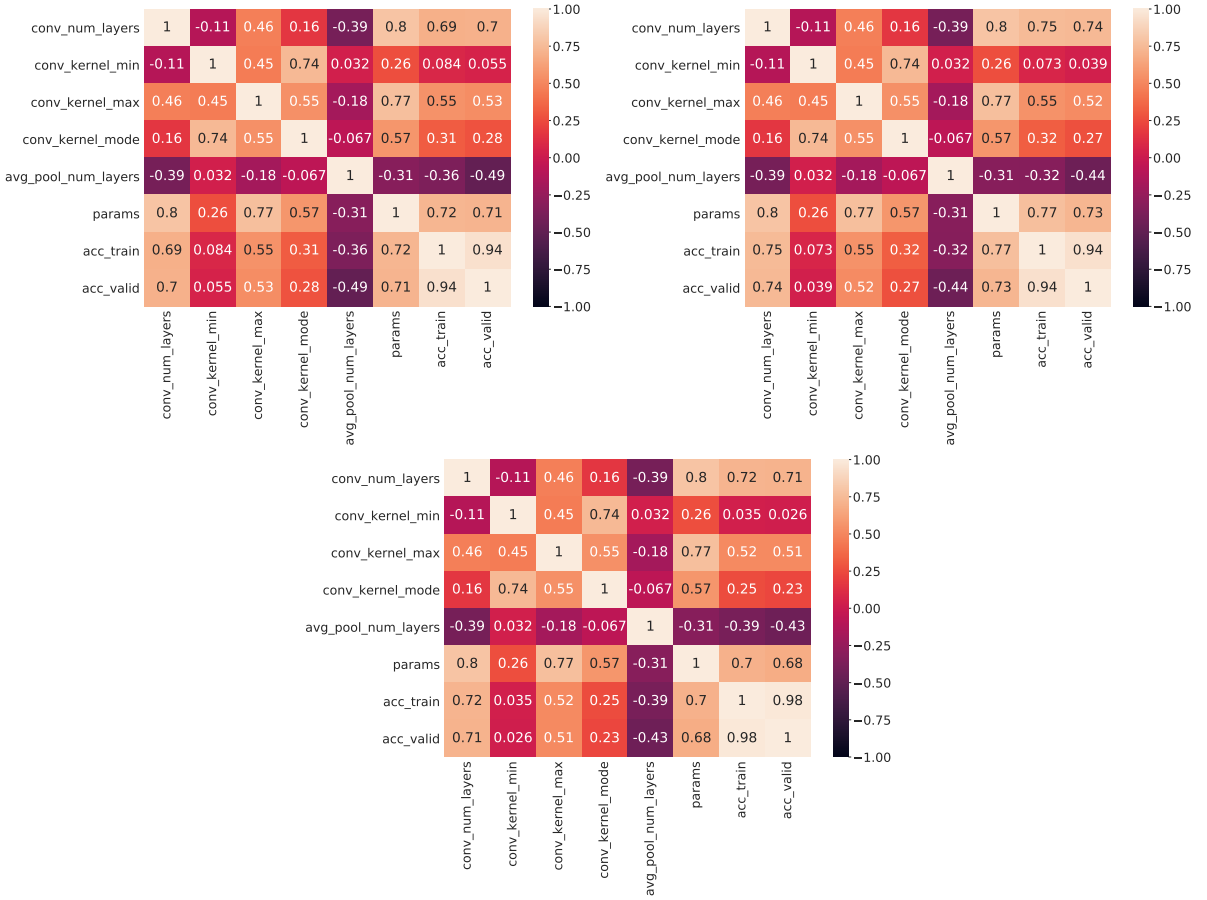


Figure 46 – Spearman Correlations of meta-features extracted from NAS-Bench-201 (with 200 epochs) regarding CIFAR-10 (left), CIFAR-100 (center), and ImageNet16-120 (right), which were used to train and evaluate meta-predictors.

A.6 Benchmarks Validation Accuracy Distributions

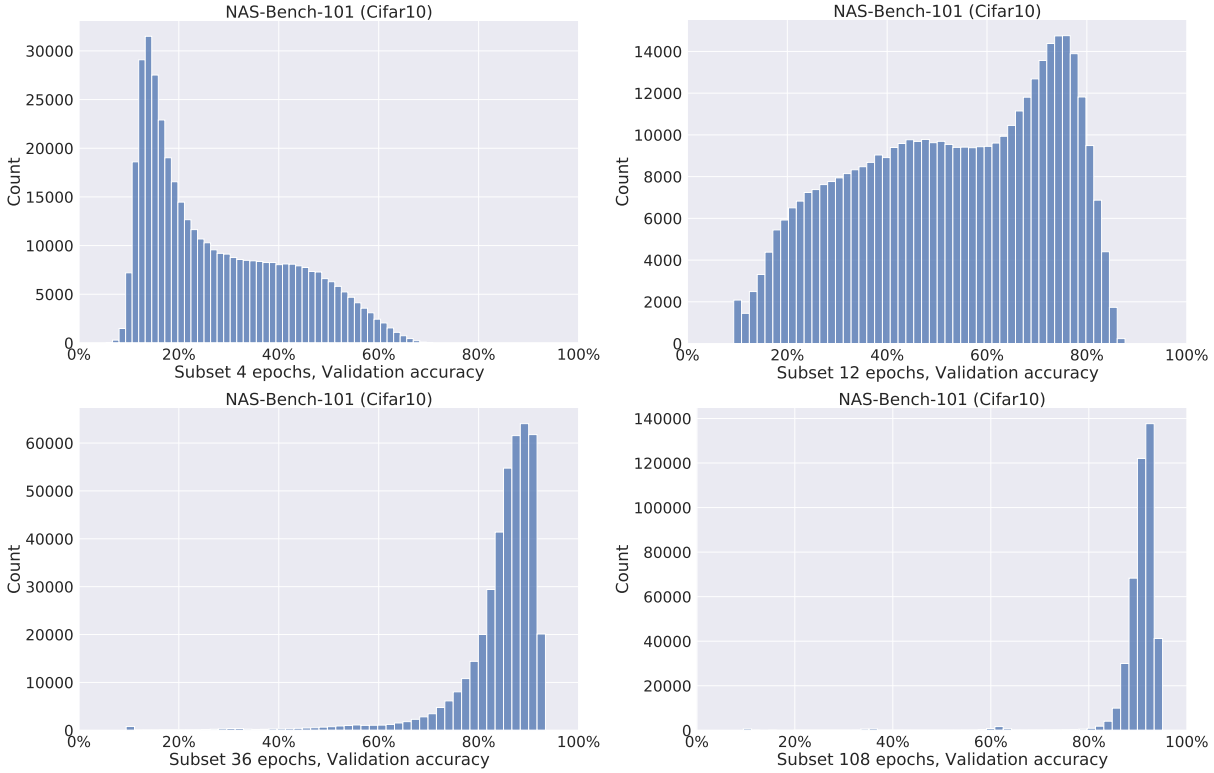


Figure 47 – Validation Accuracy distributions from NAS-Bench-101 (CIFAR-10) at 4, 12, 36 and 108 epochs.

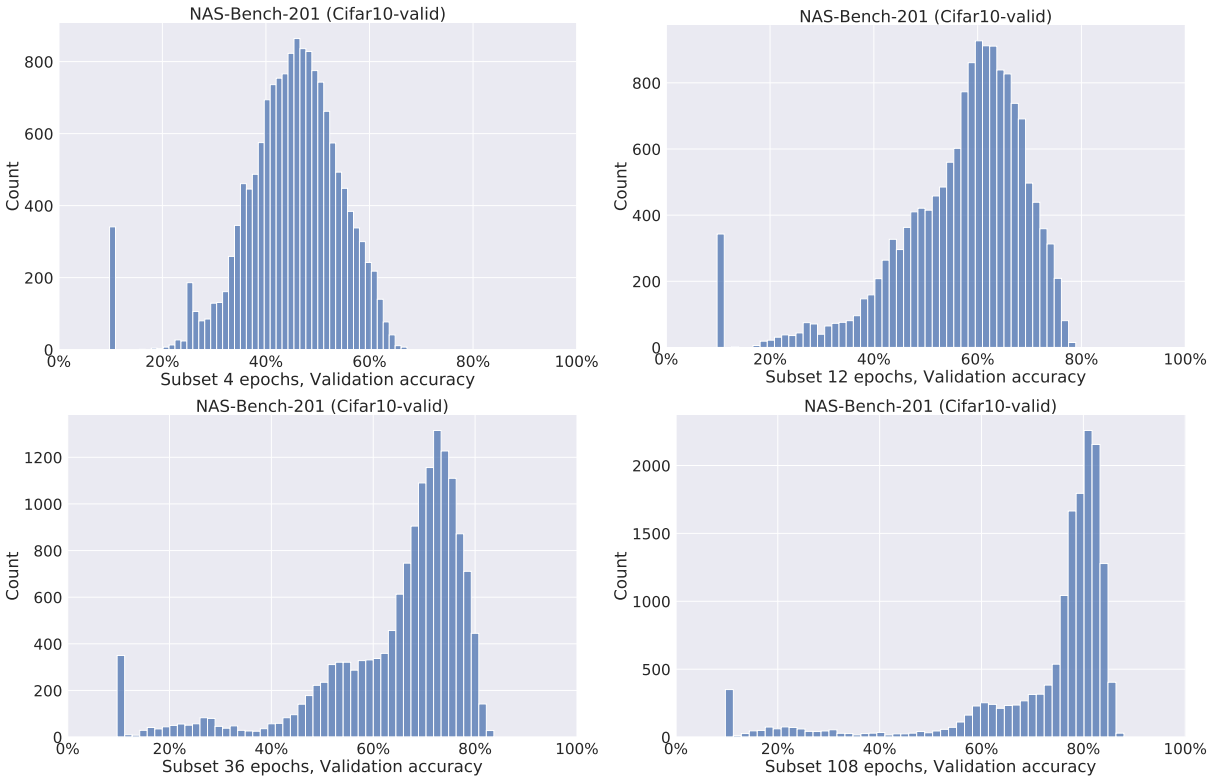


Figure 48 – Validation Accuracy distributions from NAS-Bench-201 (CIFAR-10) at 4, 12, 36 and 108 epochs.

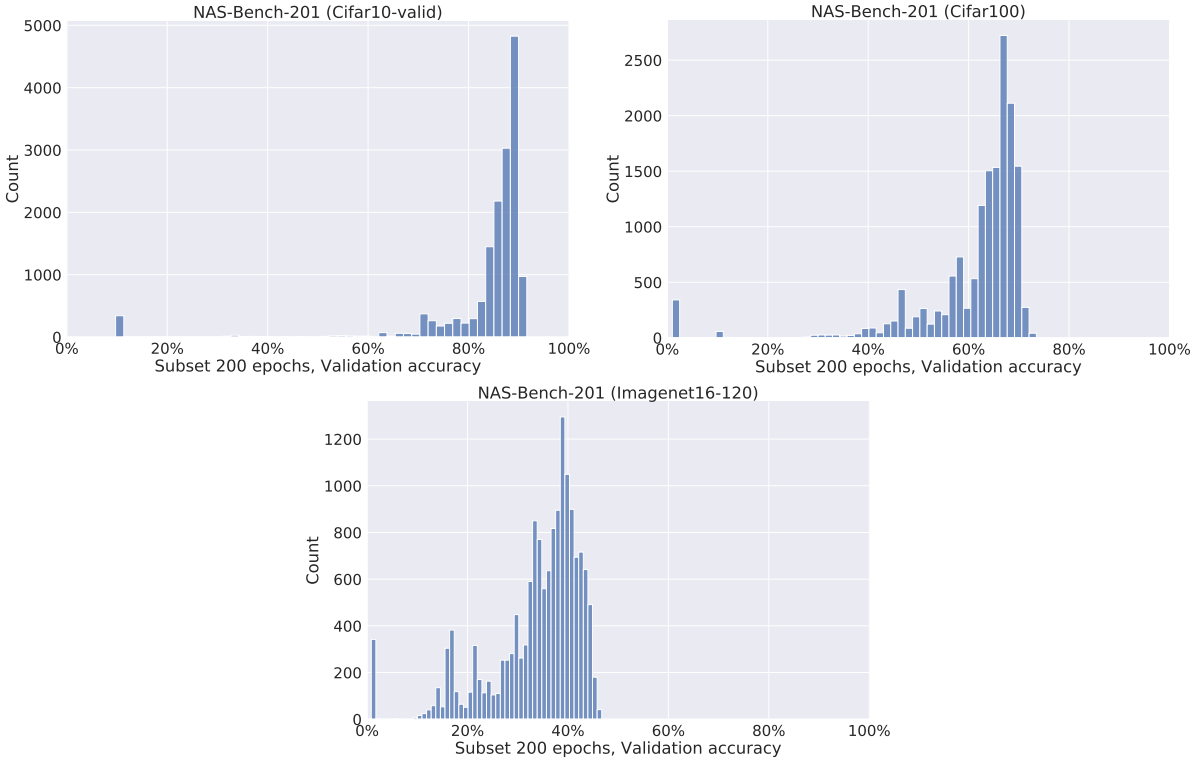


Figure 49 – Validation Accuracy distributions from NAS-Bench-201 regarding CIFAR-10 (left), CIFAR-100 (center), and ImageNet16-120 at 200 epochs.

APPENDIX

B

ACTIVE-DINTS: COMPLEMENTARY RESULTS

This Appendix presents a complementary results analysis of Active-DiNTS. This includes more learning curves of Dice generated by each uncertainty function variation and more searched cells with varying patterns.

B.1 More Learning Curve Performances

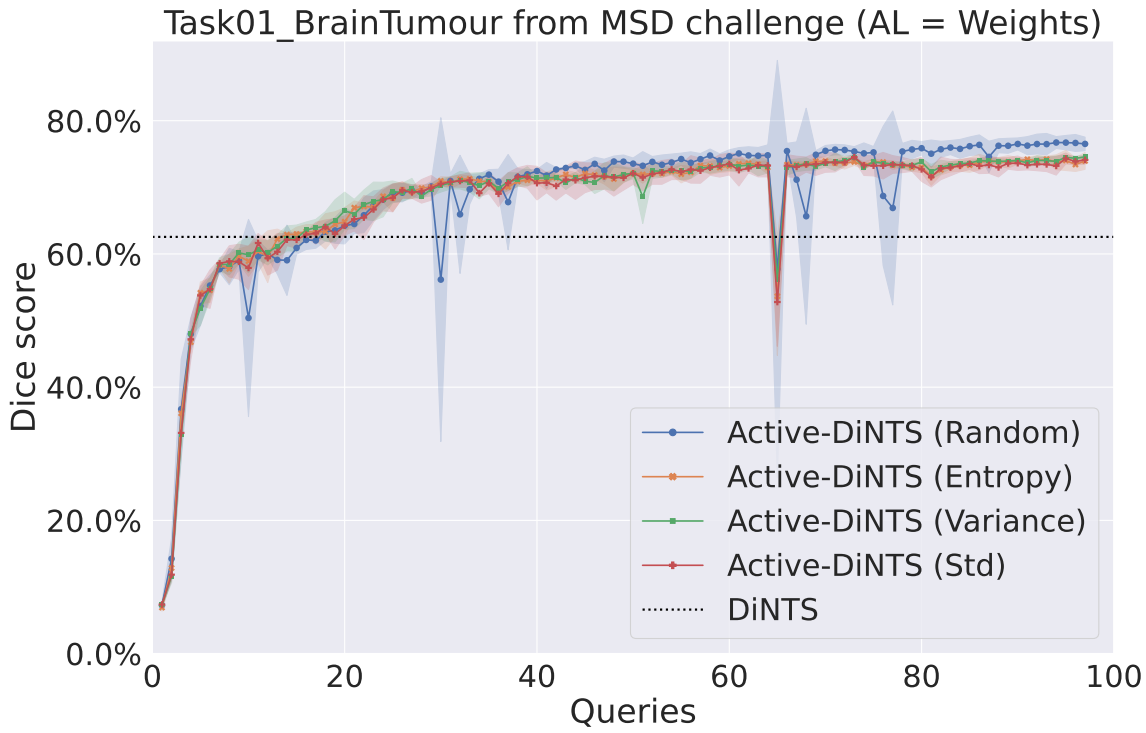


Figure 50 – Learning curves of Active-DiNTS over the number of Queries. Active Learning (AL) was used to select samples to update the Weights of the neural architectures.

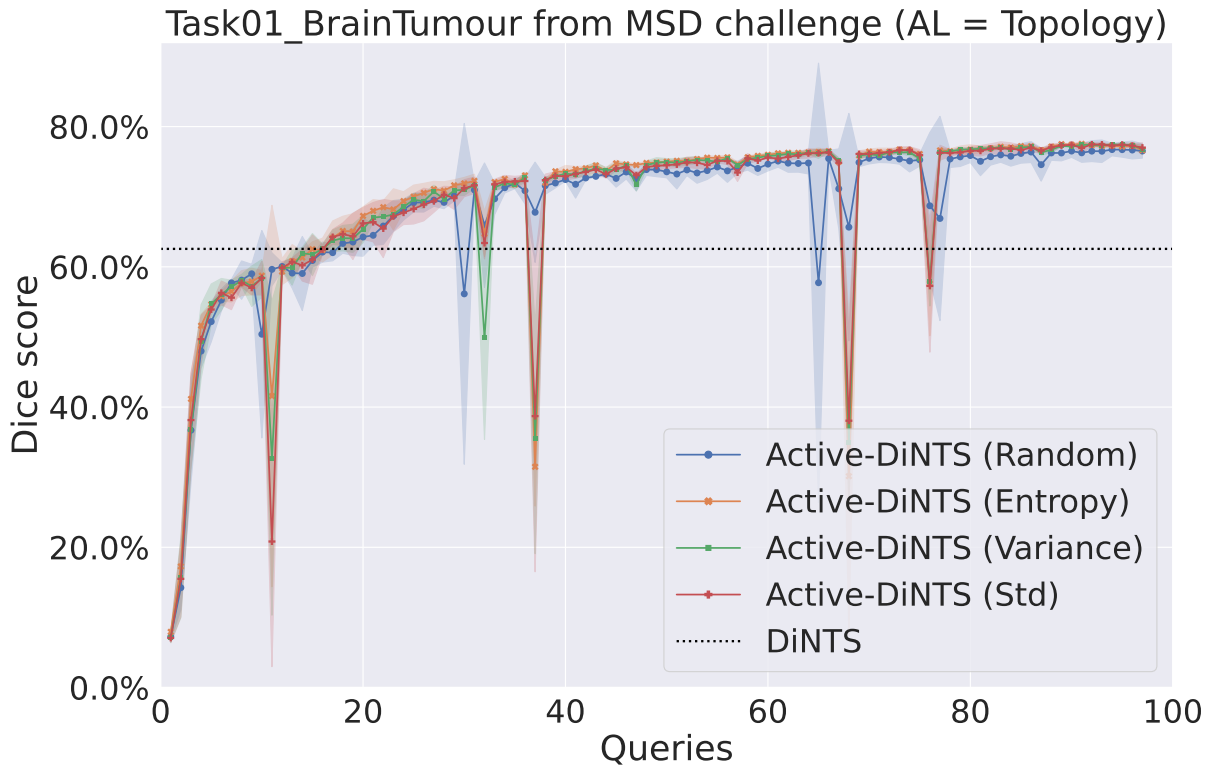


Figure 51 – Learning curves of Active-DiNTS over the number of Queries. Active Learning (AL) was used to select samples to update the Topology of the neural architectures.

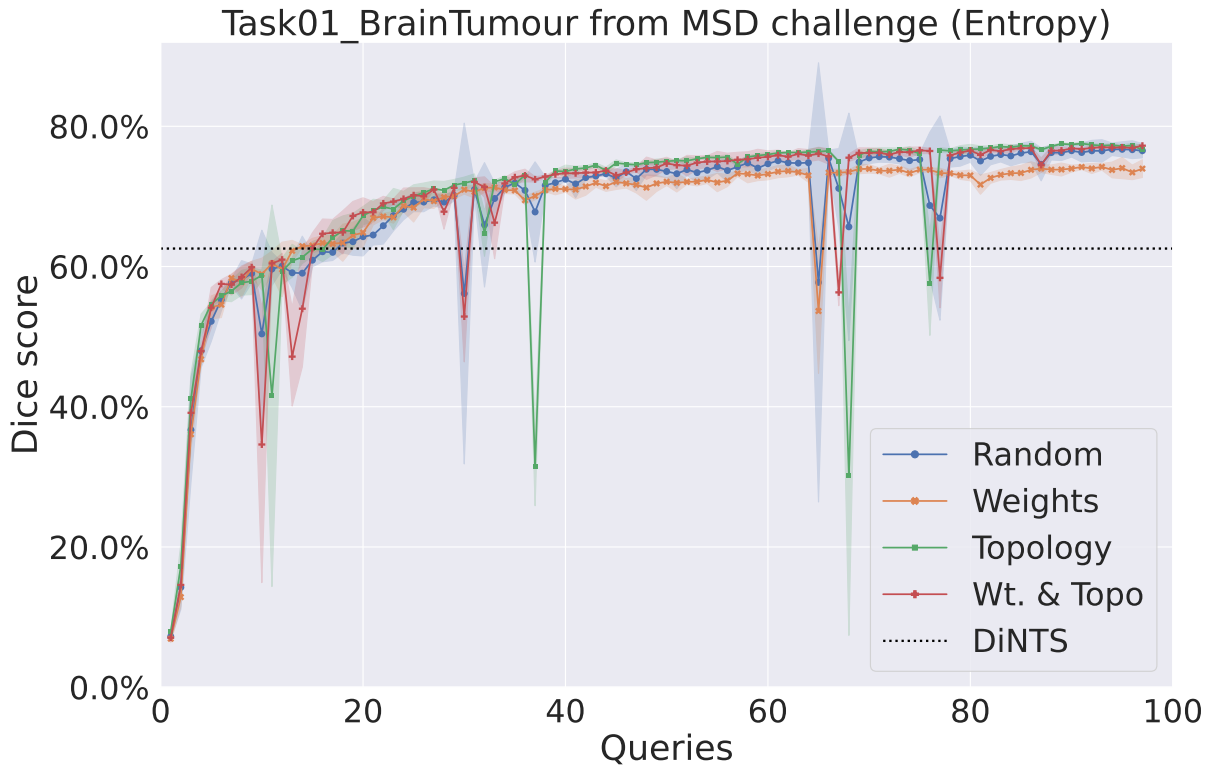


Figure 52 – Learning curves of Active-DiNTS over the number of Queries. Curves are averages of each Entropy run over all the Active Learning (AL) setups (Weights, Topology, and Wt. & Topo).

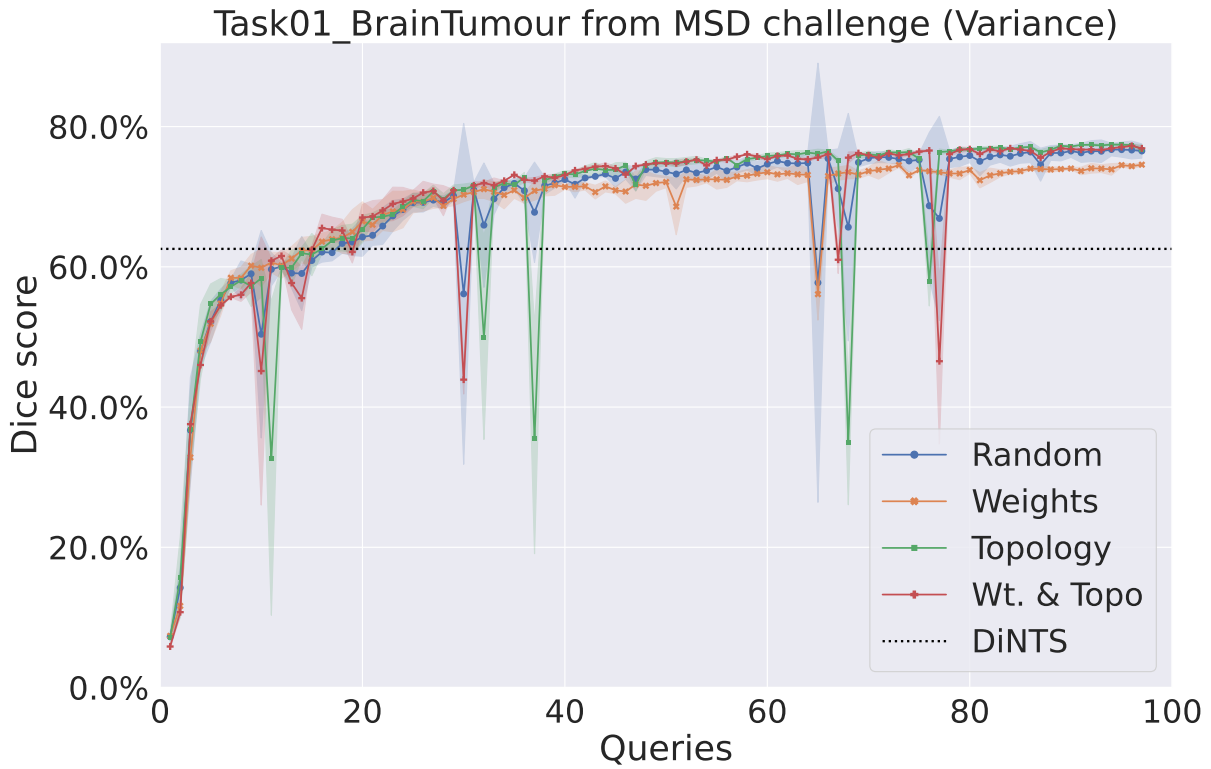


Figure 53 – Learning curves of Active-DiNTS over the number of Queries. Curves are averages of each Variance run over all the Active Learning (AL) setups (Weights, Topology, and Wt. & Topo).

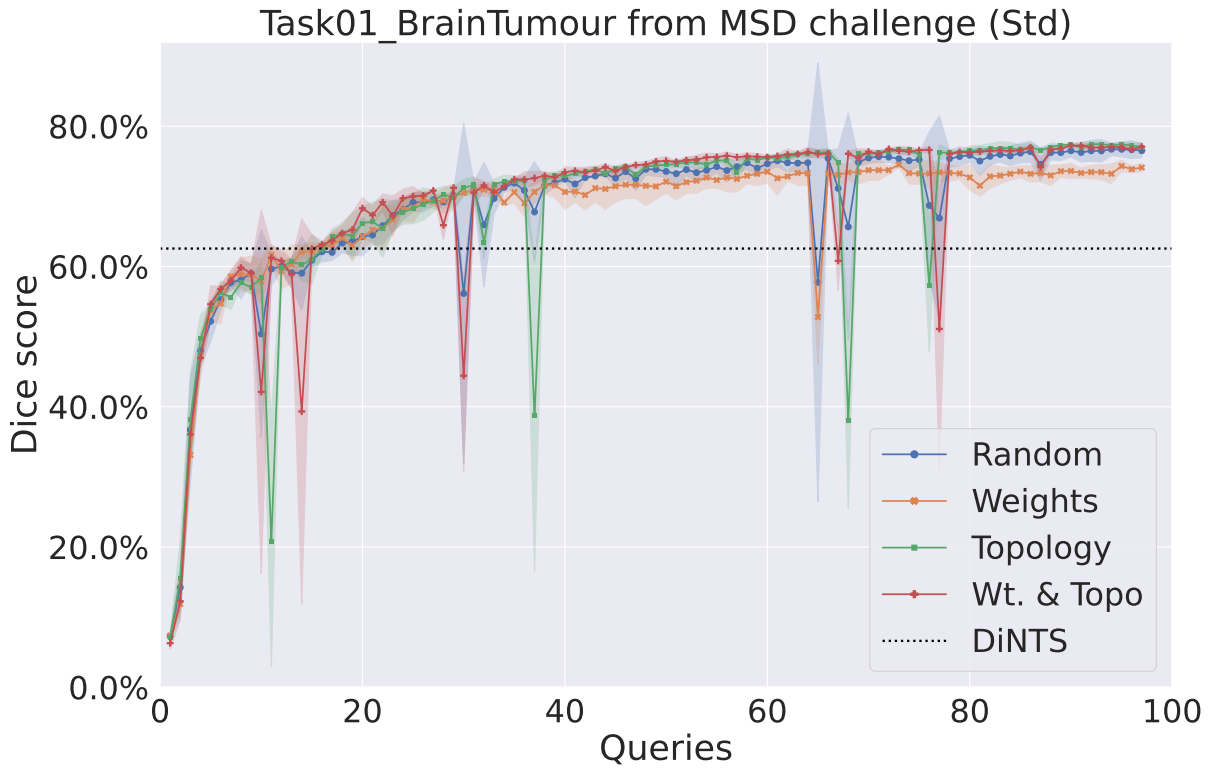


Figure 54 – Learning curves of Active-DiNTS over the number of Queries. Curves are averages of each Std run over all the Active Learning (AL) setups (Weights, Topology, and Wt. & Topo).

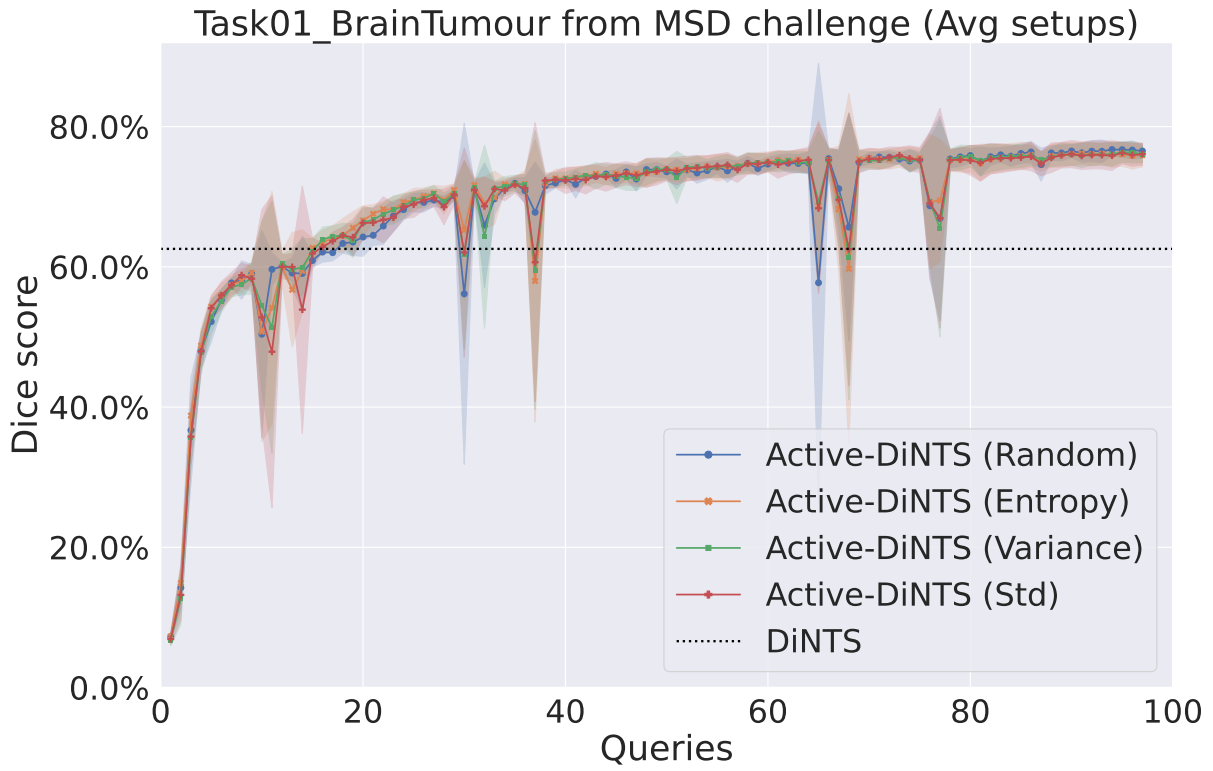
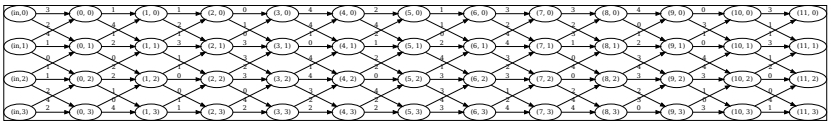
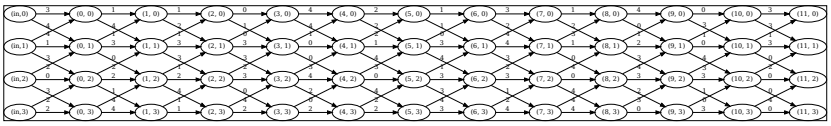


Figure 55 – Active-DiNTS learning curves over the number of Queries. Every curve is an Average (Avg) of each Query strategy model (Entropy, Variance, and Std) over all the Active Learning setups (Weights, Topology, and Weights & Topology).

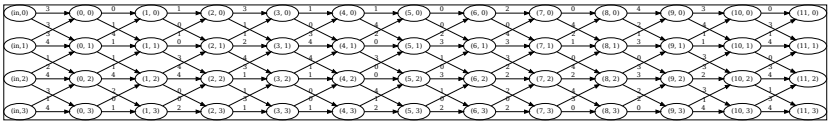
B.2 More Searched Cells



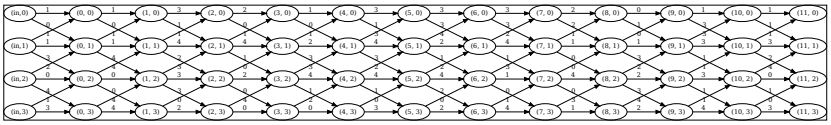
(a) Random at Query 10/97.



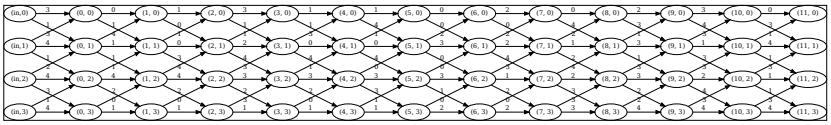
(b) Random at Query 14/97 (overcome DiNTS (Pancreas)).



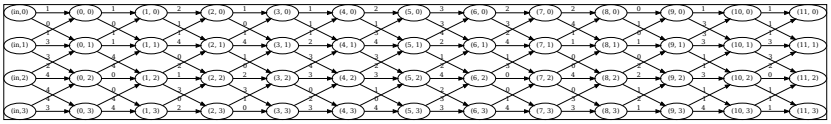
(c) Random at Query 20/97.



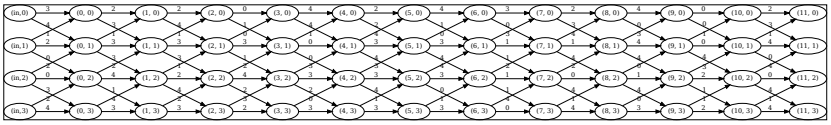
(d) Random at Query 30/97.



(e) Random at Query 41/97 (overcome DiNTS (Brain)).

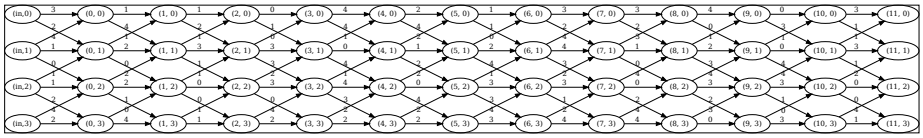


(f) Random at Query 50/97.

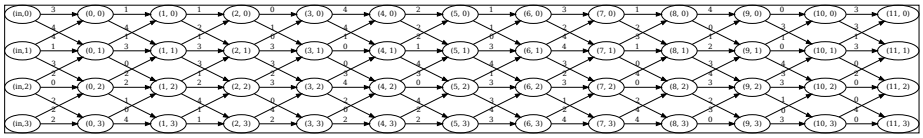


(g) Random at Query 94/97 (best Dice).

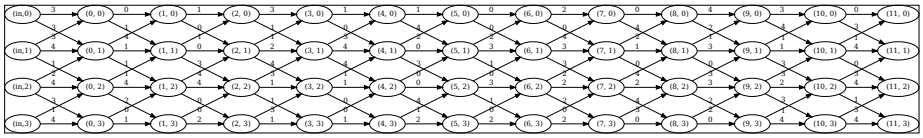
Figure 56 – Generated neural architectures on Task01_BrainTumour using the Random strategy. Edges are: 0 = Skip, 1 = 3x3x3, 2 = P3D 3x3x1, 3 = P3D 3x1x3, 4 = P3D 1x3x3.



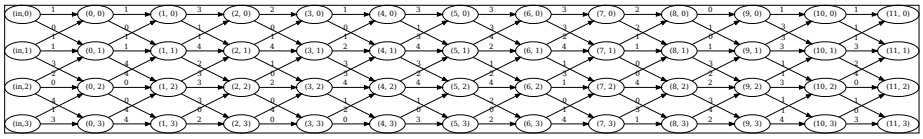
(a) Entropy at Query 10/97.



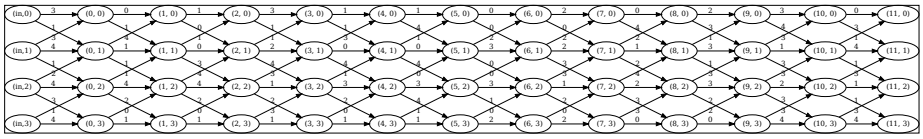
(b) Entropy at Query 13/97 (overcome DiNTS (Pancreas)).



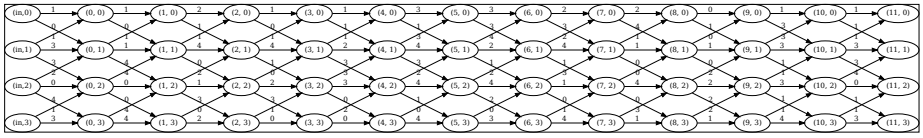
(c) Entropy at Query 20/97.



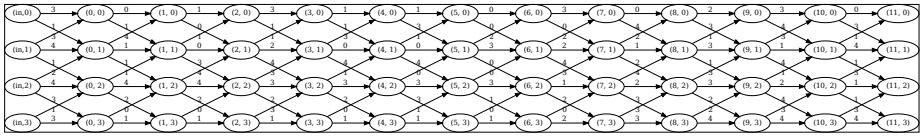
(d) Entropy at Query 30/97.



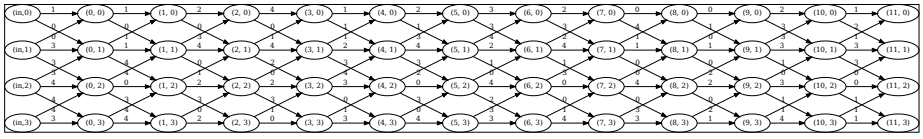
(e) Entropy at Query 35/97 (overcome DiNTS (Brain)).



(f) Entropy at Query 40/97.

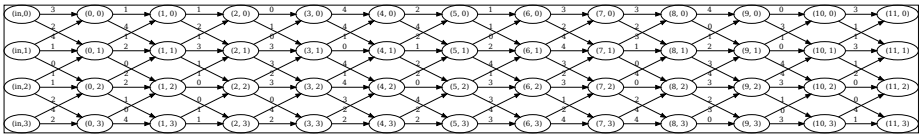


(g) Entropy at Query 50/97.

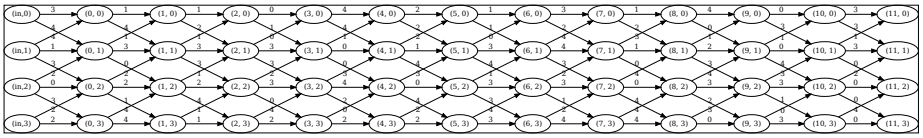


(h) Entropy at Query 91/97 (best Dice).

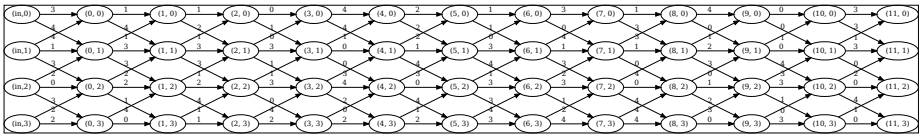
Figure 57 – Generated neural architectures on Task01_BrainTumour using the Entropy strategy. Edges are: 0 = Skip, 1 = 3x3x3, 2 = P3D 3x3x1, 3 = P3D 3x1x3, 4 = P3D 1x3x3.



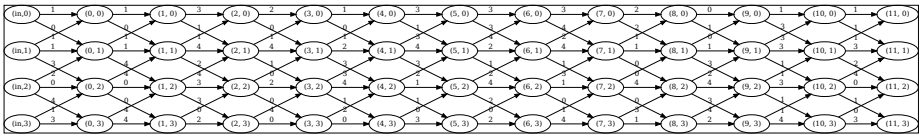
(a) Variance at Query 10/97.



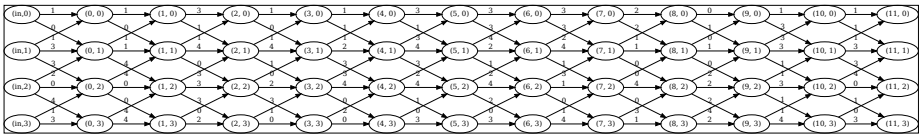
(b) Variance at Query 14/97 (overcome DiNTS (Pancreas)).



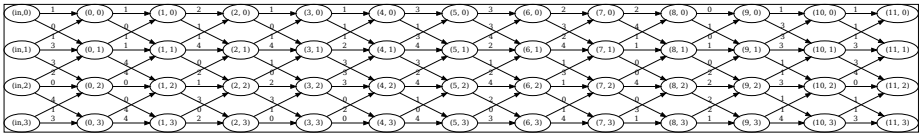
(c) Variance at Query 20/97.



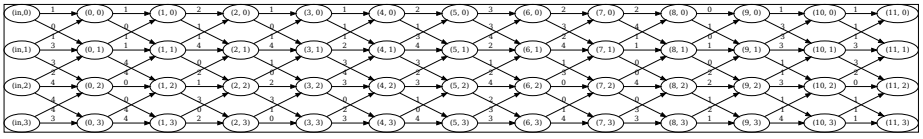
(d) Variance at Query 30/97.



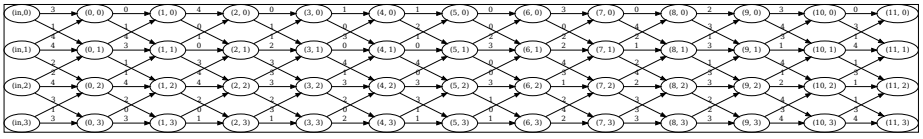
(e) Variance at Query 36/97 (overcome DiNTS (Brain)).



(f) Variance at Query 40/97.

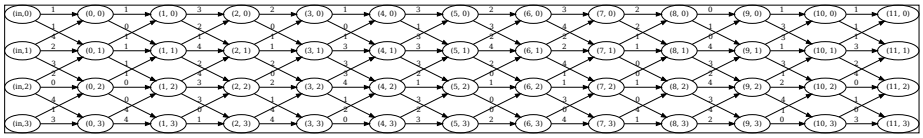


(g) Variance at Query 50/97.

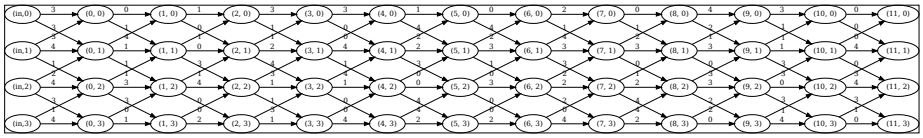


(h) Variance at Query 96/97 (best Dice).

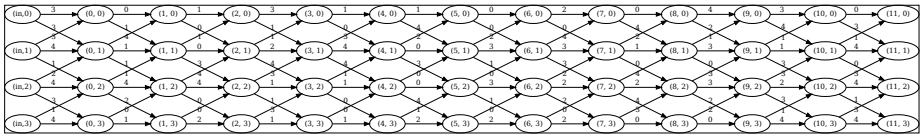
Figure 58 – Generated neural architectures on Task01_BrainTumour using the Variance strategy. Edges are: 0 = Skip, 1 = 3x3x3, 2 = P3D 3x3x1, 3 = P3D 3x1x3, 4 = P3D 1x3x3.



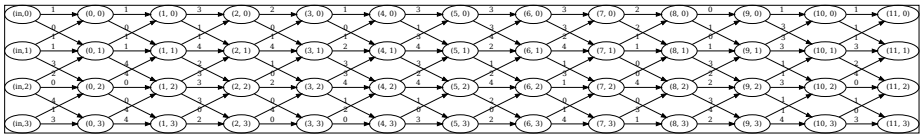
(a) Std at Query 10/97.



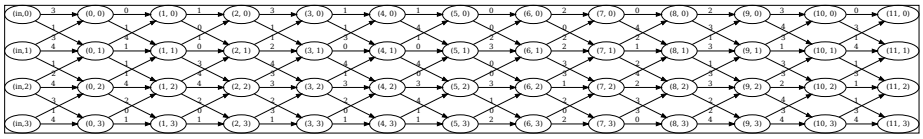
(b) Std at Query 11/97 (overcome DiNTS (Pancreas))



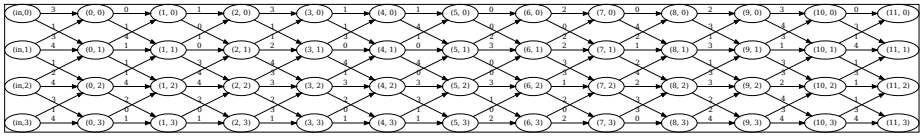
(c) Std at Query 20/97.



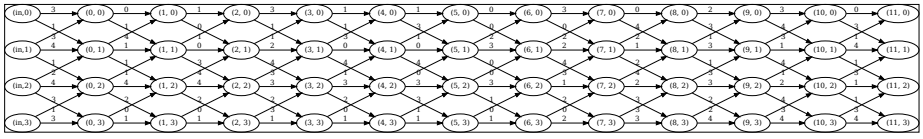
(d) Std at Query 30/97.



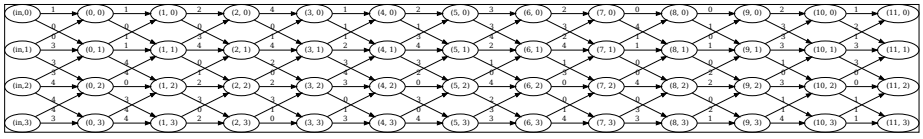
(e) Std at Query 37/97 (overcome DiNTS (Brain)).



(f) Std at Query 40/97.



(g) Std at Query 50/97.



(h) Std at Query 93/97 (best Dice).

Figure 59 – Generated neural architectures on Task01_BrainTumour using the Std strategy. Edges are: 0 = Skip, 1 = 3x3x3, 2 = P3D 3x3x1, 3 = P3D 3x1x3, 4 = P3D 1x3x3.


ICMC
USP
SÃO CARLOS