# Compressão de Modelos em Transferência de Aprendizado de Máquina

Paula Kintschev Santana de Moraes

Dissertação apresentada ao
Instituto de Matemática e Estatística
da Universidade de São Paulo
para obtenção do título de
Mestra em Ciências

Programa: Ciência da Computação
Orientadora: Profª. Drª. Leliane Nunes de Barros

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES.

São Paulo
14 de dezembro de 2021

**Compressão de Modelos em Transferência de Aprendizado de Máquina**

Paula Kintschev Santana de Moraes

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 14 de dezembro de 2021.

Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profª. Drª. Leliane Nunes de Barros (orientadora) – IME-USP
- Prof. Dr. Fernando José Von Zuben – FEEC-UNICAMP
- Profª. Drª. Patrícia Rufino Oliveira – EACH-USP

*À Lígia: missionária, avó e amiga.*

# Agradecimentos

Gostaria de agradecer primeiramente a Deus pela vida, coragem e força para superar as adversidades. Confesso que passei por muitos altos e baixos neste mestrado, mas uma coisa que se manteve constante durante todo o percurso foi a certeza de que todas as coisas cooperam para o bem daqueles que amam a Deus (Romanos 8:28). E realmente, hoje vejo que até o "atraso" no tempo que havia previsto para concluir este trabalho foi algo que rendeu grandes frutos tanto para minha carreira acadêmica quanto profissional. Definitivamente sou grata a todos que contribuíram de alguma forma com o desenvolvimento deste trabalho, mas quero destacar algumas pessoas que foram essenciais neste período.

Meus pais pelo amor, paciência e apoio em todas as minhas empreitadas. Tenho certeza de que se hoje tenho grandes metas e sonhos é porque eles acreditaram e investiram em mim, sempre me estimulando a dar passos maiores ainda que com medo.

Minha avó por me amar mais do que eu a amo (é uma piada nossa, pessoal) e por ser uma inspiração para a minha vida. Vó, obrigada pela disposição em sempre me ouvir falar sobre a pesquisa, infelizmente ou felizmente acho que a senhora sabe mais sobre Deep Learning do que as outras avós por aí.

Meus colegas de pós-graduação que tornaram meus dias mais leves e divertidos no laboratório, em especial: Thiago Bueno, Felipe Salvatore, Lucas Moura, Thiago Lira, Igor Silveira, Julissa Villanueva, Débora Lina, Erika Guetti, Shayenne Moura, Sandro Preto e Renan Jacomassi.

Meus amigos pelo apoio, compreensão e por não desistirem de serem meus amigos mesmo eu dando motivos para tal.

Ao IME pelas disciplinas e infraestrutura e a CAPES pelo auxílio financeiro.

E finalmente, mas não menos importante, à minha orientadora Leliane Nunes pela paciência em discutir a minha pesquisa e pelas minuciosas revisões de todos os textos que produzi. Também gostaria de agradecê-la pela confiança no meu trabalho como monitora do curso de robótica, acredito que essa experiência agregou muito no meu desenvolvimento acadêmico.

# Resumo

Paula Kintschev Santana de Moraes. **Compressão de Modelos em Transferência de Aprendizado de Máquina**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

O principal sucesso de aprendizado de máquina profundo está na extração automática de características dos dados, sem a necessidade de um especialista no domínio. Porém, a qualidade desta extração automática está condicionada a uma grande quantidade de dados. Em vista disso, houve uma popularização do uso de transferência de aprendizado (*transfer learning*) em que redes neurais treinadas em um domínio com muitos dados são transferidas e adaptadas a domínios similares necessitando assim de poucos dados. Essa técnica é amplamente utilizada em tarefas de classificação de imagens, nas quais uma rede neural convolucional (CNN), previamente treinada para uma tarefa origem, tem parte de suas camadas transferidas para uma rede similar e adaptada a uma nova tarefa meta com o uso de poucos dados de treinamento. No entanto, o sucesso das soluções que utilizam transferência de aprendizado depende da complexidade das camadas transferidas: adaptar uma rede com muitos parâmetros para a nova tarefa pode implicar em um alto custo computacional. Para mitigar este problema, investigamos o uso de uma técnica de compressão de modelos conhecida por *model prunning* que define diferentes critérios de eliminação de parâmetros de uma rede neural previamente treinada, gerando uma rede mais compacta sem afetar significativamente sua acurácia. Assim, o objetivo deste trabalho é investigar a viabilidade de podar um modelo treinado para uma tarefa origem antes de transferi-lo para outras tarefas meta. Para isso, utilizamos o arcabouço chamado de PRUNE2TRANSFER, que seleciona a melhor poda de parâmetros antes da transferência de aprendizado. Foram realizados experimentos de transferência de aprendizado com a rede VGG-19 para 22 novas tarefas meta usando poda não-estruturada baseada em magnitude e poda estruturada baseada na norma L1. Também foram realizados experimentos usando um algoritmo de aprendizado por reforço (DQN) para poda estruturada e não-estruturada. Os resultados mostram que, com o uso de um agente de aprendizado por reforço aplicando uma poda não-estruturada, é possível eliminar cerca de 93% dos parâmetros da rede VGG-19 sem afetar sua capacidade de extração de características na tarefa original e nas 22 tarefas investigadas.

**Palavras-chave:** Aprendizado Profundo. Transferência de Aprendizado. Poda de Parâmetros.

# Abstract

Paula Kintschev Santana de Moraes. **Model Compression in Transfer Learning**. Thesis (Masters). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

The great success of deep learning relies on its automatic feature extraction capabilities, that is, without domain expert knowledge, directly from the data. The quality of this feature extraction is conditioned to large amounts of data, which can be unfeasible for learning tasks with small datasets. Transfer learning is a technique that overpasses the supervised learning assumption, extracting knowledge from a model trained in a domain with large amounts of data to a similar domain with fewer data helping to achieve greater performance. This technique is widely used in image classification tasks where some layers of a pre-trained convolutional neural network (CNN) are transferred and adapted to a new task. Although it is popular, learning tasks that use pre-trained models rely on deep models to process each new data, which increases the computational cost of the solution. To mitigate this problem, we investigate model pruning which is a compression technique that reduces complexity by eliminating network parameters without deteriorating the model's performance. Generally pruning is conducted on a model trained for a specific task, however, for pre-trained models used in transfer learning tasks, it would be best to transfer an already reduced model rather than pruning it for each new task. Therefore, the goal of this work is to assess the viability of pruning a model before it is transferred to other tasks and to compare different methods of model pruning for this process. We propose the framework PRUNE2TRANSFER that evaluates the best pruning ratio before transfer. We conducted transfer learning experiments with VGG-19 for 22 target tasks applying unstructured and structured pruning algorithms on the source task. We also investigated pruning techniques that are based on a deep reinforcement learning algorithm (DQN). Our results show that a reinforcement learning agent using unstructured pruning can eliminate close to 93% of the parameters of VGG-19 without damaging its feature extractors.

**Keywords:** Deep Learning. Transfer Learning. Model Pruning.

# Lista de Abreviaturas

| | |
|---|---|
| CNN | *Convolutional Neural Network* |
| DFN | *Deep Feedforward Network* |
| DL | *Deep Learning* |
| DQN | *Deep Q-Network* |
| IA | Inteligência Artificial |
| MDP | *Markov Decision Process* |
| p.p. | ponto percentual |
| PuRL | *Pruning using Reinforcement Learning* |
| RL | *Reinforcement Learning* |
| StructPuRL | *Structured Pruning using Reinforcement Learning* |
| TL | *Transfer Learning* |

# Lista de Símbolos

| | |
|---|---|
| $\odot$ | Produto elemento a elemento (Hadamard) |
| $\mathbb{1}$(condição) | Função indicadora de uma certa condição |

# Lista de Figuras

# Lista de Tabelas

# Lista de Programas

# Sumário

# Apêndices



# Anexos

# Capítulo 1

# Introdução

Na última década, graças aos avanços computacionais, tanto em termos de *software* como *hardware*, modelos de aprendizado profundo (*Deep Learning* – DL) se tornaram estado da arte em tarefas nos domínios de texto, imagem e áudio. O sucesso do aprendizado profundo está na extração automática de características através dos dados (aprendizado de representações), ou seja, na construção de características sem a necessidade de um especialista no domínio (LeCun, Bengio *et al.*, 2015). No aprendizado de representações (*representation learning*) o conhecimento é construído a partir de uma composição de transformações simples não-lineares que permite o aprendizado de funções complexas a partir da interação com uma grande quantidade de dados. Esta modalidade de aprendizado viabiliza o aprendizado de ponta-a-ponta em que há o mapeamento direto de uma entrada sem um tratamento prévio de extração de características, como uma imagem, em uma saída de alto nível, como uma tomada de decisão. Bojarski *et al.* (2016) relata o sucesso desta abordagem ao extrair automaticamente representações internas necessárias para o controle de um veículo a partir de imagens capturadas por uma câmera frontal, permitindo que o sistema aprenda por imitação com as imagens rotuladas pelas ações tomadas por um motorista humano.

Em DL, as representações aprendidas partem de estruturas simples que quando combinadas e transformadas pelas próximas camadas formam estruturas mais complexas, estas por sua vez resumem o dado de entrada num subespaço de menor dimensionalidade e, provavelmente, mais informativo que o dado em seu espaço original. Dado que diferentes domínios podem compartilhar representações elementares é possível transferir conhecimento de um contexto para outro. A transferência de aprendizado (*Transfer Learning* – TL) busca explorar as informações obtidas no aprendizado de uma tarefa (original) para auxiliar no aprendizado de outra tarefa (meta), partindo da premissa que existem representações internas que podem ser relevantes para ambas (Goodfellow *et al.*, 2016). O uso de TL viabiliza a extração automática de características em contextos com dados insuficientes para o ajuste da rede neural a partir de pesos aleatórios, assim como acelera o treinamento de redes profundas.

Esta técnica é amplamente utilizada em tarefas de classificação de imagens nas quais uma rede neural convolucional (*Convolutional Neural Network* – CNN) pré-treinada tem parte de suas camadas transferidas para uma rede similar e adaptada a uma nova tarefa

meta. Yosinski *et al.* (2014) comprova empiricamente a possibilidade de compartilhar representações entre tarefas, apontando uma maior capacidade de generalização dos modelos que utilizaram algum pré-treino ao invés de uma inicialização aleatória. Apesar do sucesso, tarefas que usam modelos pré-treinados adicionam mais uma etapa em sua esteira de processamento, conforme ilustrado na Figura 1.1. Esta etapa eleva o custo computacional da solução, pois cada nova imagem deverá passar por uma rede neural extratora de características, que dependendo do modelo escolhido pode chegar em 500MB de armazenamento e a milhões de parâmetros treináveis.

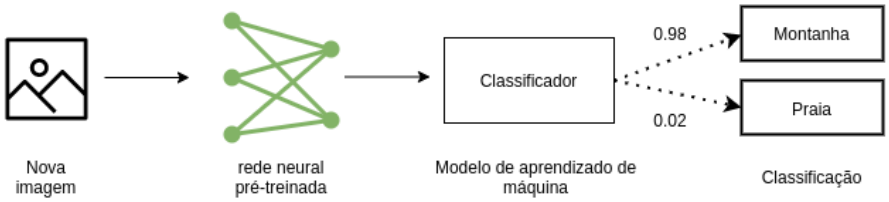



**Figura 1.1:** *Fluxo tradicional de uma solução que utiliza transferência de aprendizado. Nota-se a presença de uma etapa de pré-processamento da imagem antes de entrar no classificador.*

A redução da intensidade computacional de modelos profundos é o objeto de estudo da área de compressão de modelos que investiga formas de tornar computacionalmente eficiente o processamento de redes neurais. Dentro de compressão de modelos, a área de poda de parâmetros visa a máxima remoção de parâmetros da rede minimizando o impacto no desempenho do modelo, reduzindo assim o número de parâmetros treináveis e, em alguns casos, reduzindo o tamanho em memória do modelo.

A forma com que a poda de parâmetros é feita impacta diretamente na topologia do modelo. Uma poda é classificada como não-estruturada quando atua diretamente no nível de conexões individuais entre neurônios; e estruturada quando considera um conjuto de conexões entre os neurônios levando em consideração a estrutura a qual pertencem. O tipo de poda e em que momento ela é realizada na rede neural são tópicos de grande discussão nos últimos anos.

Um dos trabalhos pioneiros nesta área foi publicado por LeCun, Denker *et al.* (1989), no qual o critério de poda baseado na matriz Hessiana dos pesos é incorporado na função de custo do treinamento. Outros métodos de poda (Han, Pool *et al.*, 2015; Li *et al.*, 2017; M. Gupta *et al.*, 2020) foram desenvolvidos nos últimos anos a fim de reduzir a complexidade de modelos profundos e viabilizar inferências em dispositivos com *hardware* limitado.

## 1.1 Motivação

O fluxo tradicional de uma tarefa de classificação que faz uso de transferência de aprendizado (Figura 1.1) está atrelado a um modelo neural profundo que onera computacionalmente a solução. A fim de reduzir a complexidade da solução final, é possível combinar a transferência de aprendizado com algum critério de compressão de modelos, como feito pela empresa de tecnologia NVIDIA®. Em sua ferramenta de transferência de aprendizado (NVIDIA, 2019), a complexidade ao adicionar uma etapa de poda de parâmetros *após a transferência e adaptação da rede à nova tarefa* é reduzida. Esta é uma abordagem

eficaz, conforme os resultados apresentados por Zhu e S. Gupta (2018) que indicam que a poda de parâmetros pode ser utilizada para comprimir uma rede neural tornando-a mais computacionalmente eficiente.

Entretanto, podar um modelo pré-treinado após a transferência de aprendizado para uma tarefa meta traz os seguintes pontos de atenção:

(a) a poda se torna específica para a tarefa em questão, ou seja, a poda é realizada a cada nova transferência e adaptação do modelo pré-treinado;

(b) a poda de parâmetros é realizada em redes neurais o que restringe o tipo de classificador que será utilizado na adaptação à tarefa meta (Figura 1.1); visto que estamos considerando que a poda é feita após a transferência para a tarefa meta.

O primeiro item destaca uma ineficiência da abordagem. Já o segundo item impacta diretamente as tarefas que utilizam transferência de aprendizado para construir um novo conjunto de dados em um espaço vetorial, gerado a partir dos extratores de características, que será utilizado no treinamento de outro classificador sem ser uma rede neural, como por exemplo SVM, florestas aleatórias, kNN, e etc (Vogado *et al.*, 2018; Deepak e Ameer, 2019; Loey *et al.*, 2021).

## 1.2 Objetivo

O objetivo deste trabalho é estudar a transferência de um modelo podado em uma tarefa origem para uma outra tarefa meta, de forma que tarefas que dependam de uma rede pré-treinada possam partir de um modelo com um número reduzido de parâmetros, e consequentemente menos custoso computacionalmente.

Além disso, como um objetivo secundário, expandimos a pesquisa para outras técnicas de poda e comparamos seus desempenhos tanto na tarefa de origem quanto em tarefas de transferência de aprendizado.

## 1.3 Trabalhos Correlatos

Os temas de transferência de aprendizado e poda de redes neurais já foram abordados em outros trabalhos. Seguindo a ordem cronológica, Molchanov *et al.* (2016) propuseram um novo critério para poda de redes neurais baseado na expansão de Taylor, e para averiguar a eficácia da técnica testaram em duas tarefas de classificação (birds-200 e flowers-102) realizando transferência de aprendizado. Os autores realizaram a transferência e adaptação das camadas pré-treinadas antes de realizar a poda do modelo. Portanto, a poda do modelo estava condicionada às características de cada nova tarefa, não podendo ser reaproveitada entre tarefas.

J. Liu *et al.* (2017) propôs uma nova forma de realizar transferência de aprendizado combinando-a com compressão de modelos neurais. A motivação dos autores era reduzir o sobreajuste do modelo (*overfitting*) a fim de melhorar a transferência de representações para novas tarefas (tarefas meta) com poucos dados. A abordagem proposta consiste na criação de três redes neurais: (i) uma rede esparsa obtida através da poda iterativa da

rede neural original (*Sparse-SourceNet*); (ii) uma rede híbrida (*Hybrid-TransferNet*) que combina a saída da rede esparsa adaptada ao inferir uma imagem da nova tarefa juntamente com um extrator de características do conjunto original dos dados, que tem o papel de auxiliar na transferência; e (iii) uma rede esparsa (*Sparse-TargetNet*) obtida através da poda da (*Hybrid-TransferNet*) considerando o conjunto de dados da tarefa meta. Apesar do sucesso na adaptação para as tarefas meta, o processo envolve uma etapa intermediária que deixa a solução mais computacionalmente custosa. E aparenta estar muito atrelado ao pré-treinamento feito com o conjunto de dados do IMAGENET (Deng *et al.*, 2009), que apesar de ser o mais habitual não é uma regra em processos de transferência de aprendizado.

Durante a realização dos experimentos dessa dissertação, foram publicados outros dois trabalhos com metodologias similares ao que propomos nesta dissetarção (Soelen e Sheppard, 2019; Morcos *et al.*, 2019) . Ambos os trabalhos utilizam a mesma técnica de poda de modelos baseada na identificação de uma subrede dentro da rede original que "ganhou na loteria da inicialização dos pesos", ou seja, essa subrede foi inicializada de uma forma tal que seu desempenho é comparável ou até melhor que a da rede completa. Essa subrede, por sua vez, é encontrada a partir da poda da rede original usando uma metodologia proposta por Frankle e Carbin (2019), que consiste numa poda não-estruturada com reinicialização dos pesos originais a cada passo iterativo de poda. Porém, segundo os autores, a presença de um "*winning ticket*" em redes convolucionais com arquitetura similar à ResNet e VGG é dependente da escolha de hiperparâmetros do treinamento, o que torna este critério de poda específico a certas topologias e condições de treino.

## 1.4 Contribuições

As principais contribuições deste trabalho de mestrado são:

- Proposta do arcabouço PRUNE2TRANSFER que realiza a poda de um modelo antes de transferi-lo para uma outra tarefa;
- Adaptação do teste-t pareado bayesiano para técnicas de poda iterativa, permitindo a escolha de um percentual de poda baseado em um intervalo de equivalência prática;
- Modificação do critério de poda estruturada de forma a integrar a análise de sensibilidade e definindo um critério baseado na diferença em pontos percentuais da acurácia do modelo antes da poda e após a poda.
- O método StructPuRL que é uma modificação do algoritmo PuRL (M. Gupta *et al.*, 2020) para poda estruturada de redes neurais convolucionais.
- Análise comparativa empírica de diferentes técnicas de poda para transferência de aprendizado.

## 1.5 Organização

Esta dissertação está organizada nos seguintes capítulos:

- **Capítulo 2 (Fundamentos: Aprendizado de Máquina e Redes Neurais)**: Apresenta uma revisão de aprendizado de máquina supervisionado, aprendizado por

reforço, redes neurais, aprendizado profundo com ênfase em redes convolucionais e transferência de aprendizado;

- **Capítulo 3 (Fundamentos: Compressão de Modelos)**: Apresenta uma revisão dos conceitos e trabalhos seminais da área de compressão de modelos;
- **Capítulo 4 (Compressão de Modelos: Melhorias e Extensões)**: Detalha as modificações realizadas nos métodos de poda estruturada e não-estruturada e a proposta do método StructPuRL;
- **Capítulo 5 (Compressão de Modelos em Transferência de Aprendizado)**: Apresenta e detalha as etapas que compõem o arcabouço PRUNE2TRANSFER;
- **Capítulo 6 (Análise Empírica)**: Apresenta os resultados obtidos com o PRUNE2TRANSFER para cada método de poda de parâmetros;
- **Capítulo 7 (Conclusões e Trabalhos Futuros)**: Apresenta as conclusões e perspectivas de trabalhos futuros.

# Capítulo 2

# Fundamentos: Aprendizado de Máquina e Redes Neurais

Os primórdios da Inteligência Artificial (IA) foram marcados pela resolução de problemas difíceis para humanos, mas relativamente fáceis para computadores (Goodfellow *et al.*, 2016). A característica em comum destes problemas era a descrição por regras formais, abordagem baseada em conhecimento, que quando descritas em código permitiam uma resolução automática.

No entanto, a formalização de regras para a criação de uma base de conhecimento é uma tarefa que requer conhecimento de especialistas no assunto, o que pode ser custoso ou inacessível na resolução de certos problemas. Portanto, grande parte do avanço em IA se dá pelo aprendizado a partir de exemplos/experiências passadas, permitindo que os sistemas reconheçam padrões nos próprios dados. Esta abordagem de construção automática de conhecimento a partir de dados é conhecida por aprendizado de máquina.

## 2.1 Aprendizado de Máquina

Um algoritmo de aprendizado de máquina (*machine learning* - ML) é um algoritmo capaz de aprender com dados (Goodfellow *et al.*, 2016). Uma definição de aprendizado amplamente aceita é a de Mitchell (1997), o qual define que um programa de computador aprende pela *experiência E* em respeito a uma classe de *tarefas T* e *desempenho D*, se o desempenho em tarefas em *T*, mensurado por *D*, melhorar com a experiência *E*.

Uma tarefa $T$ em ML está relacionada a forma com que o algoritmo deverá processar um conjunto de dados de entrada (*dataset*) $\mathbf{X}$, tais dados podem ser imagens, textos ou um conjunto tabular de informações de um objeto ou evento. Cada exemplo do conjunto de dados (*data point*) em $\mathbf{X}$ é um vetor de características (*features*) $x \in \mathbb{R}^n$, em que cada entrada $x_i$ do vetor $n$-dimensional é uma característica que compõe o exemplo. Dois tipos comuns de tarefas são problemas de classificação e regressão. Em uma tarefa de classificação, o objetivo é aprender uma função $f : \mathbb{R}^n \rightarrow \{1, ..., k\}$, sendo $k$ o número de classes (ou categorias), que mapeia uma entrada a uma classe. Já em uma tarefa de regressão, o objetivo é aprender uma função $f : \mathbb{R}^n \rightarrow \mathbb{R}$ que mapeia uma entrada a um valor numérico.

A fim de mensurar a qualidade do aprendizado é necessário estimar o desempenho $D$ do algoritmo através de alguma métrica. Uma métrica comumente utilizada em tarefas de classificação é a acurácia, ou seja, a proporção de classificações corretas produzidas pelo modelo em um conjunto de dados.

Algoritmos de aprendizado de máquina podem ser classificados de acordo com o tipo de experiência $E$ permitida durante o processo de aprendizado (Goodfellow *et al.*, 2016), sendo categorizados em: supervisionado, não-supervisionado, semi-supervisionado e por reforço.

Os algoritmos de aprendizado também podem ser classificados de acordo com sua capacidade de representação, ou seja, as funções que um modelo pode aprender dentro de um espaço de hipóteses. Os chamados *modelos paramétricos* têm sua capacidade de representação limitada pela definição a priori da quantidade de parâmetros necessária para resolver o problema, independente dos dados de treinamento (Russell e Norvig, 2016). Exemplos de algoritmos de aprendizado paramétricos são: regressão linear, regressão logística e *naive bayes*.

Já os *modelos não-paramétricos* não fazem suposições restritivas quanto ao espaço de hipóteses, ou seja, não pressupõem uma quantidade fixa de parâmetros para o modelo. Na prática, a complexidade das funções modeladas por esses algoritmos pode ser definida em função da quantidade de dados disponíveis no treinamento. Exemplos de algoritmos de aprendizado não paramétricos são: árvores de decisão, máquinas de vetores suporte (*support vector machines* - SVM) e redes neurais.

### 2.1.1 Aprendizado Supervisionado

O aprendizado supervisionado é uma técnica que utiliza em seu treinamento um conjunto de pares de dados rotulados na forma $(x_i, y_i)$, sendo $x_i$ a entrada conhecida e $y_i$ um rótulo gerado por uma função desconhecida $y_i = f(x_i)$. O objetivo do aprendizado supervisionado é aprender uma função $h$ que se aproxime da função geradora $f$, a partir de um conjunto finito de dados, amostrados possivelmente com ruído, da função $f$. (Russell e Norvig, 2016). Portanto objetiva-se minimizar o erro do que foi predito pelo modelo ($\hat{y}$) em relação ao rótulo correto ($y$), a fim de que o modelo seja capaz de interpolar/generalizar para outros dados similares.

Neste tipo de aprendizado, a qualidade preditiva do modelo está diretamente relacionada à qualidade do conjunto de dados de treino, o que envolve tamanho da base e representatividade dos dados. Logo, o modelo só é capaz de lidar com situações que se assemelham ao que foi observado durante o treinamento.

### 2.1.2 Aprendizado por Reforço

Ao contrário do aprendizado supervisionado, o aprendizado por reforço (*Reinforcement Learning* – RL) não aprende sobre um conjunto de dados fixo, mas a partir de interações de um agente com um ambiente através da execução de ações e a coleta de recompensas (Sutton e Barto, 2018). Um agente de RL é capaz de observar o estado em que se encontra após executar ações no ambiente. Neste tipo de aprendizado o agente não é informado sobre quais ações deverá tomar: ele deve aprender quais ações maximizam a recompensa

acumulada esperada ao longo de episódios de interações com o ambiente. Isso é feito através de uma estratégia de tentativa e erro. Portanto um problema de aprendizado por reforço é um problema de decisão sequencial (Barto *et al.*, 1989) de um estado inicial até um estado meta, podendo ser formulado como um Processo de Decisão Markoviano (*Markov Decision Process* – MDP) (Puterman, 1995).

**Definição 1** (Processo de Decisão Markoviano)**.** *Um MDP é uma tupla* $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \beta \rangle$ *em que:*

- $\mathcal{S}$ *é o conjunto de estados do ambiente,*
- $\mathcal{A}$ *é o conjunto de ações disponíveis para o agente,*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ *é a função de recompensa,*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ *é a função de transição probabilística que define, para cada par estado-ação* $(s, a)$*, a probabilidade de alcançar o estado* $s'$*, isto é,* $\mathcal{T}(s'|s, a)$*, e*
- $\beta \in [0, 1)$ *é o fator de desconto sobre as recompensas futuras.*

Dado um MDP, podemos definir a função valor $Q(s, a)$ para cada par estado-ação que devolve a recompensa acumulada esperada no estado $s$ após a execução da ação $a$. Com base nesta função, podemos determinar a melhor ação para cada estado, isto é, uma política ótima para o MDP $\pi^* : \mathcal{S} \mapsto \mathcal{A}$ como um mapeamento entre estados e ações tal que:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s, a), \tag{2.1}$$

sendo $\pi^*$ também chamada de política gulosa com relação à função $Q(s, a)$.

Note que, no aprendizado por reforço, consideramos que a função de transição probabilística $\mathcal{T}$ é desconhecida. Assim, um agente RL deve estimar a função valor $Q(s, a)$ através de suas interações com o ambiente. Logo, o problema de RL pode ser formulado como um problema de otimização em busca da política ótima $\pi^*$, ou seja, a política que maximiza o retorno acumulado do agente em cada estado.

O algoritmo *Q-Learning* (Watkins, 1989) estima os Q-valores para uma política ótima interagindo com o ambiente e atualizando uma tabela com todos os Q-valores para todos os pares estado-ação. Este algoritmo utiliza duas políticas: (i) uma política de exploração (por exemplo uma política aleatória ou uma política $\epsilon$-greedy (Sutton e Barto, 2018)), que com uma chance $\epsilon$ amostra aleatoriamente uma ação e com $1 - \epsilon$ amostra uma ação que maximiza o retorno esperado (Linha 5), e (ii) uma política de explotação (gulosa), a qual estamos interessados em aprender.

O Algoritmo 1 descreve o algoritmo básico de RL *Q-Learning* como descrito em (Sutton e Barto, 2018) usando uma representação tabular da função $Q(s, a)$, inicializada arbitrariamente para cada par $(s, a)$. A cada início de episódio, o algoritmo seleciona um estado inicial $s$ (linha 3) e usa uma política de exploração *$\epsilon$-greedy* para definir as ações que serão executadas até o final do episódio (considerando um horizonte finito $H$ de interações ou a existência de um estado terminal). Para cada passo do episódio, ao executar a ação $a$

no estado $s$, observamos a recompensa $r$ e o estado resultante $s'$ e é feita a atualização da estimativa de $Q(s, a)$ através da seguinte operação:

$$Q(S, A) = Q(S, A) + \alpha[R + \beta \max_a Q(S', a) - Q(S, A)], \tag{2.2}$$

sendo $\alpha$ o fator de aprendizado. Note que na Linha 7 ao selecionarmos a ação $a'$ que maximiza o valor esperado partindo do estado sucessor $s'$ estamos selecionando a ação da política gulosa. A política devolvida por esse algoritmo é a política gulosa com relação à estimativa final de $Q(s, a)$.

**Algoritmo 1** *Q-Learning* (Sutton e Barto, 2018)

1: Inicializa arbitrariamente $Q(s, a), \forall s \in \mathcal{S},\ a \in \mathcal{A}(s)$
2: **repita** (para cada episódio):
3: Seleciona um estado inicial $s$
4: **repita** (para cada passo do episódio):
5: Escolha $a$ a partir de $s$ usando uma política de exploração derivada de $Q$
6: Tome a ação $a$, observe $r, s'$
7: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \beta \max_{a'} Q(s', a') - Q(s, a)]$
8: $s \leftarrow s'$
9: **até** fim de episódio
10: **até** execução de todos os episódios
**Devolve:** $\pi^*(s) = \arg\max_a Q(s, a)$

Dada a necessidade de explorar todas as ações em todos os estados por um certo número de iterações, o uso do algoritmo *Q-Learning* pode ser inviável para MDPs com um grande espaço de estados/ações ou ações contínuas. Como veremos no Capítulo 4, uma implementação eficiente deste algoritmo usa redes neurais para aproximar a função $Q(s, a)$, além de incorporar outras técnicas de melhorias como *experience replay*, *mini-batch* e *target-network*. Essa versão do algoritmo é também chamada de DQN (*Deep Q-Network*).

## 2.2 Redes Neurais Artificiais

Inspiradas em neurônios biológicos, as redes neurais artificiais são modelos matemáticos não paramétricos que sintetizam mapeamentos multidimensionais de entrada-saída, capazes de aproximar qualquer função contínua em um espaço compacto (Hornik *et al.*, 1989).

### 2.2.1 Neurônio Artificial

Seguindo a estrutura do neurônio artificial proposto por McCulloch e Pitts (1943) e posteriormente melhorada por Rosenblatt (1958) com a criação do *perceptron*, uma unidade (Figura 2.1) é o elemento mais básico de uma rede neural. Portanto, uma rede neural é um conjunto de unidades conectadas por ligações direcionadas. A forma de conexão das unidades (topologia) e suas propriedades definem a arquitetura da rede (Russell e Norvig, 2016).

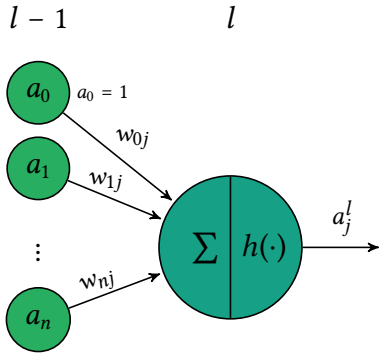



**Figura 2.1:** *Estrutura de um neurônio artificial*

A semelhança entre os neurônios artificiais e os neurônios biológicos está em que ambos respondem ("disparam") a uma entrada dada que uma certa condição foi satisfeita. As diferentes formas de um neurônio ser estimulado, juntamente com as conexões desses neurônios, permitem que padrões complexos sejam aprendidos por essas redes.

### 2.2.2 Topologia do MLP

A estrutura mais típica de rede neural, e base de estruturas mais complexas, é o perceptron multicamada (*multilayer perceptron* - MLP), ou rede neural com alimentação para frente (*feedforward network*), ou rede neural completamente conectada (*fully connected network*). A topologia dessa rede consiste normalmente na organização das unidades por **camadas**, de tal forma que as unidades de uma camada anterior serão a entrada da camada subsequente. Esta ligação direcionada propaga duas informações importantes: o valor da ativação e o **peso** (ou parâmetro) da conexão entre as duas unidades. Nessa arquitetura o objetivo do aprendizado é definir um mapeamento $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, aprendendo os valores dos pesos que resultam em um melhor mapeamento (Goodfellow *et al.*, 2016).

Analisando os elementos da Figura 2.1, é possível identificar três etapas distintas no processamento realizado por uma unidade da camada $l$:

1ª A combinação linear das entradas resultantes da camada $l-1$ somada ao viés (*bias*) da camada $l$. O viés é uma unidade extra presente em todas as camadas da rede que possui sempre ativação igual a 1. Graficamente, não apresenta arestas de entrada apenas arestas de saída. A função do viés é auxiliar na aproximação de funções que não necessariamente passam pela origem.

$$z_j = \sum_{i=0}^{n} a_i^{l-1} w_{ij}^{l} \tag{2.3}$$

Sendo $n$ o número de unidades e $a_i$ os valores das ativações da camada $l-1$. $w_{ij}$ corresponde aos pesos aprendidos para cada conexão entre a camada anterior e a unidade $j$ (Abu-Mostafa *et al.*, 2012; Russell e Norvig, 2016).

2ª O valor $z_j$ é passado para a função de ativação $h(\cdot)$. A função de ativação define um limiar para o qual a unidade será "disparada". Por exemplo, se a função de ativação utilizada fosse a sigmoide (ilustrada na Figura 2.2), a unidade seria disparada para

um $z_j > 0$.

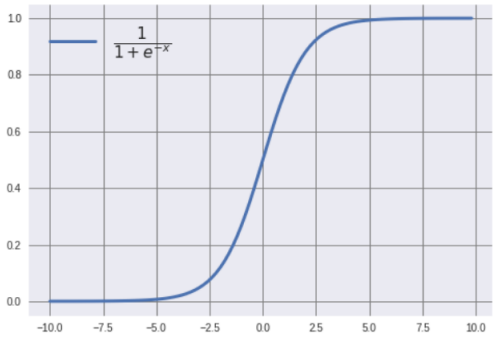



**Figura 2.2:** *Função sigmoide*

3ª A saída da unidade $j$ consiste no resultado da ativação, no caso:

$$a_j^l = h(z_j) \tag{2.4}$$

Na topologia de redes neurais completamente conectadas, as camadas que compõem uma rede são divididas em: **camada de entrada**, **camadas ocultas** e **camada de saída**. Uma rede neural apresenta apenas uma camada de entrada e suas unidades têm apenas arestas saindo para as unidades da próxima camada. Essas unidades de entrada não apresentam uma função de transformação associada, servindo apenas como entrada dos dados para o processamento na rede. O número de unidades dessa camada é equivalente à dimensão de características dos dados de entrada da rede.

As unidades da camada oculta têm como entrada os valores provenientes da camada anterior ponderados pelo peso de cada conexão. As saídas dessas unidades estão conectadas a todas as unidades da próxima camada, com cada aresta ponderada por um peso. O número de camadas e número de unidades de cada camada é arbitrário, sendo determinados de acordo com a complexidade da função a ser aproximada.

A camada de saída é a última etapa de processamento da rede, da qual extraímos o vetor de predição $\hat{\mathbf{y}}$. O número de unidades dessa camada está associado ao que a rede deverá predizer. Em tarefas de classificação multiclasse com $k$ classes, a camada de saída apresenta uma unidade por classe.

A Figura 2.3 apresenta a estrutura de uma rede neural completamente conectada, destacando a organização em camadas. É interessante notar que existem outras topologias de redes neurais a partir de modificações nas conexões entre as camadas e unidades. O compartilhamento de pesos e a retroalimentação de unidades são duas modificações empregadas pelas *redes convolucionais* e *redes recorrentes*, respectivamente, ambas amplamente utilizadas em aprendizado profundo.

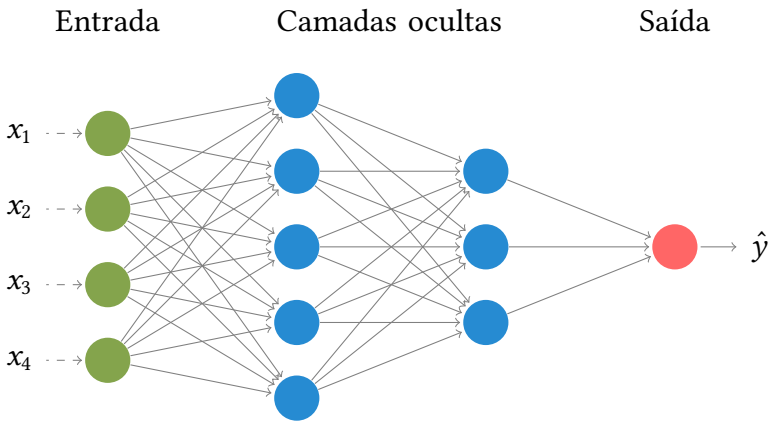



**Figura 2.3:** *Exemplo de rede neural completamente conectada para um problema de classificação binária. Nessa arquitetura, a entrada consiste em 4 unidades que processarão um vetor* $\mathbf{x} \in \mathbb{R}^4$*, seguida por duas camadas ocultas com 5 e 3 unidades, respectivamente, e a saída com apenas uma unidade que computa a predição* $\hat{y}$*.*

## 2.3 Aprendizado Profundo

Aprendizado Profundo (*Deep Learning* – DL) é uma subárea de aprendizado de máquina que utiliza diferentes topologias de redes neurais a fim de aprender representações complexas dos dados de entrada a partir de um encadeamento hierárquico de representações mais simples (Goodfellow *et al.*, 2016). A rede neural base das arquiteturas em DL é o MLP com mais de uma camada oculta chamado de DFN (*Deep Feedforward Network*), em que a quantidade de camadas influencia na complexidade do espaço de funções aproximadas.

O aprendizado de representações mais simples pela própria rede viabilizou a resolução de tarefas que envolvessem dados não-estruturados como imagens, áudio e texto. Isso, porque o uso destes dados requeria o conhecimento de um especialista a fim de extrair características destes dados antes de alimentarem os modelos tradicionais de aprendizado. Diferentemente dos dados estruturados (tabulares), os dados não-estruturados apresentam características que dependem da análise de um contexto.

No caso das imagens, a informação é representada por *pixels* que assumem valores dentro do intervalo de 0 a 255. Os *pixels* são arranjados de forma matricial em uma dimensão, nas imagens em escala de cinza, ou em mais dimensões formando imagens coloridas. Cada dimensão de uma imagem também é chamada de canal (ou componente), para imagens que adotam o espaço de cor RGB (*Red, Blue and Green*) dizemos que estas possuem três canais.

Processar uma imagem em uma DFN requer uma unidade na camada de entrada para cada *pixel* da imagem. Logo, uma imagem colorida com dimensões $224 \times 224 \times 3$ necessitaria de 150 528 unidades de entrada na rede. Além da elevada quantidade de parâmetros cada *pixel* é processado em uma determinada posição do vetor de entrada, acarretando em redundância de parâmetros e problemas com dimensionalidade e posição dos elementos na imagem. Portanto, o ideal para o processamento de imagens seria uma topologia que permitisse uma entrada no espaço tensorial, a fim de preservar a estrutura e posição dos

elementos da imagem, e permitisse o compartilhamento de parâmetros dentro de uma janela contextual, a fim de extrair informações de mesma natureza independente de sua posição na imagem. Estas características estão presentes nas chamadas redes neurais convolucionais.

## 2.4 Redes Convolucionais

As redes convolucionais (*convolutional neural networks* – CNN) são uma família de arquiteturas de redes neurais que apresentam camadas convolucionais em sua topologia (LeCun, Boser *et al.*, 1989). Uma camada convolucional é caracterizada pela organização dos pesos (parâmetros) em **filtros**. Um filtro é um tensor de ordem 3 composto por uma coleção de ***kernels***, que são matrizes que convoluem por cada canal do tensor de entrada, por exemplo um filtro para uma imagem RGB é composto por três *kernels*, um para cada canal R, G e B. Após a etapa de convolução[1] do filtro, cada matriz resultante é somada, formando o chamado **mapa de características** ou mapa de ativação (*feature map*), e este mapa por sua vez fará parte da entrada da próxima camada da rede neural convolucional. A Figura 2.4 ilustra o processo de convolução de um filtro sobre uma entrada (produto interno), neste caso uma imagem colorida, formando uma mapa de características que agregam a informação extraída de cada canal.

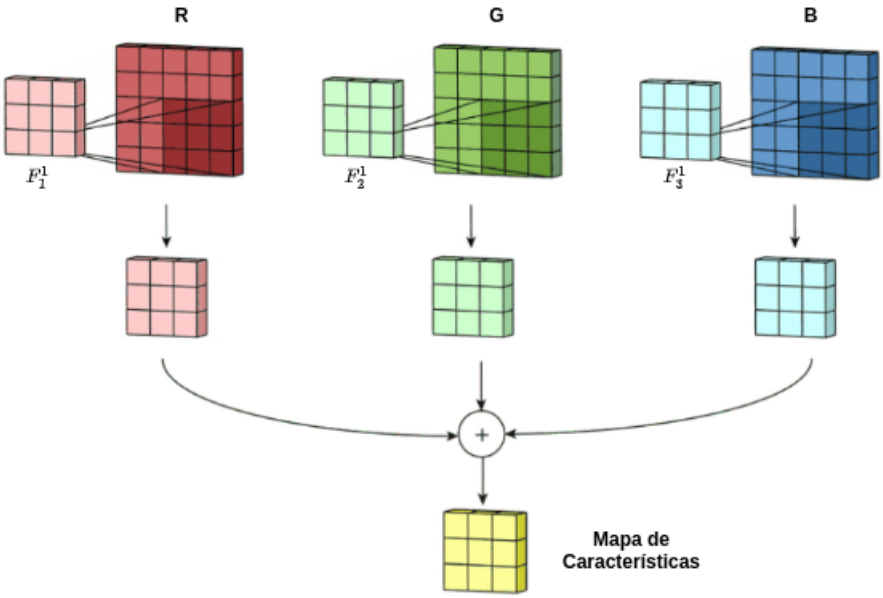



**Figura 2.4:** *Exemplo da convolução de um filtro F sobre uma imagem RGB. Note que o filtro trata-se de um tensor de dimensões* $3 \times 3 \times 3$. *Figura adaptada de* Irhum Shafkat, *2018.*

Além da quantidade de filtros e tamanho do *kernel*, uma camada convolucional também apresenta outros hiperparâmetros como *padding* e *stride*. *Padding* é um preenchimento com zeros ao redor da matriz de entrada de forma a manter suas dimensões espaciais após a convolução, dado que na aritmética convolucional as dimensões espaciais de uma entrada

[1] Neste trabalho chamaremos de convolução a operação de correlação cruzada (*cross-correlation*), amplamente utilizada em processamento de imagens.

$n \times n$ convoluída com um *kernel* $k \times k$ resulta numa matriz $(n - k + 1) \times (n - k + 1)$. Já o *stride* determina o tamanho do passo da convolução, geralmente é utilizado um passo unitário de forma a passar por toda a matriz. Caso o stride $s$ seja maior ou igual a dois, as dimensões espaciais de saída serão determinadas por $\left\lfloor \frac{n-k}{s} \right\rfloor + 1$ (Dumoulin e Visin, 2016).

Outra estrutura presente na arquitetura de uma CNN são as camadas de ***pooling***, que também são utilizadas para reduzir a bidimensionalidade de um volume de entrada a partir de operações de síntese em janelas, sendo as mais comuns maximizar (*max pool*) ou computar a média (*average pool*).

Uma característica da topologia da CNN é o compartilhamento dos pesos no filtro, o que induz a extração de uma característica específica do volume de entrada independente da sua posição na matriz (LeCun, Boser *et al.*, 1989). O que é uma vantagem em relação a DFN em que cada unidade de um neurônio está conectada a um *pixel* de entrada. A DFN além de requerer um elevado número de parâmetros para cada neurônio, inviabiliza a compreensão de um contexto local daquele *pixel* podendo levar a uma redundância de conexões. A invariância espacial da CNN é fundamental no processamento de imagens, pois viabiliza, por exemplo, a detecção de bordas em qualquer região da imagem utilizando o mesmo filtro.

Dada esta capacidade das redes convolucionais de aprenderem filtros durante o treinamento em uma tarefa é possível cogitar o reaproveitamento destes filtros em outras tarefas. Visto que a construção das representações em aprendizado profundo é encadeada da mais simples até uma representação mais complexa, os filtros pertencentes às primeiras camadas convolucionais tendem a extrair características mais gerais das imagens (Yosinski *et al.*, 2014), viabilizando seu reuso. Esta é uma questão investigada na subárea de *transferência de aprendizado*.

## 2.5 Transferência de Aprendizado

A Transferência de Aprendizado (*Transfer Learning* – TL) é uma estrutura de aprendizado que relaxa a hipótese de algoritmos de aprendizado supervisionado de que os dados de treinamento e teste devem ser independentes e identicamente distribuídos (i.i.d.), ou seja, são representados pelas mesmas características e amostrados da mesma distribuição geradora (Pan, 2014). Esta relaxação endereça o problema de quantidade de dados insuficientes para treinar alguns algoritmos de aprendizado de máquina, especialmente em aprendizado profundo.

Em DL, o sucesso de extração de padrões e características latentes está diretamente relacionado ao treinamento com uma quantidade massiva de dados. Neste contexto, a transferência de aprendizado permite que padrões encontrados em um domínio sejam reutilizados em outro domínio, reduzindo a demanda de dados e acelerando a convergência do modelo (Tan *et al.*, 2018).

Na terminologia proposta por Pan e Yang (2010), um domínio $\mathcal{D}$ consiste num espaço de características $\mathcal{X}$ e uma distribuição marginal $P(X)$, em que $X$ é um exemplo da amostra de dados tal que $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Portanto, a diferença entre domínios pode ser atribuída a um espaço de características diferentes (ex. texto para imagem, estudado pela

transferência de aprendizado heterogênea) ou a diferenças nas distribuições marginais das características (ex. adaptação de domínio (Tzeng *et al.*, 2017)).

Ainda na terminologia de Pan e Yang (2010), dado um domínio $\mathcal{D} = \{\mathcal{X}, P(X)\}$ uma tarefa consiste em um espaço de rótulos conhecidos $\mathcal{Y}$ e uma função preditiva $f(\cdot)$, que é aprendida pelos pares de dados $\{x_i, y_i\}$ tal que $x_i \in X$ e $y_i \in \mathcal{Y}$. Em posse destes dois conceitos podemos definir transferência de aprendizado.

**Definição 2** (Transferência de Aprendizado)**.** *Dada uma tarefa origem de aprendizado $\mathcal{T}_s$ com um domínio $\mathcal{D}_s$ e uma tarefa de aprendizado meta $\mathcal{T}_t$ com um domínio $\mathcal{D}_t$, a transferência de aprendizado objetiva melhorar o desempenho da função preditora meta a partir da experiência obtida na tarefa origem, considerando que $\mathcal{D}_s \neq \mathcal{D}_t$ e/ou $\mathcal{T}_s \neq \mathcal{T}_t$.*

No contexto de DL, a função preditora meta é um mapeamento não-linear multidimensional representado por uma rede neural profunda. Tzeng *et al.* (2017) categoriza as diferentes abordagens de TL baseadas em: instância, mapeamento, rede e adversarial. A abordagem baseada em rede será analisada neste trabalho e consiste no reuso parcial de redes neurais pré-treinadas, ou seja, treinadas em um domínio diferente daquele em que será aplicada. A Figura 2.5 ilustra a transferência baseada em rede.

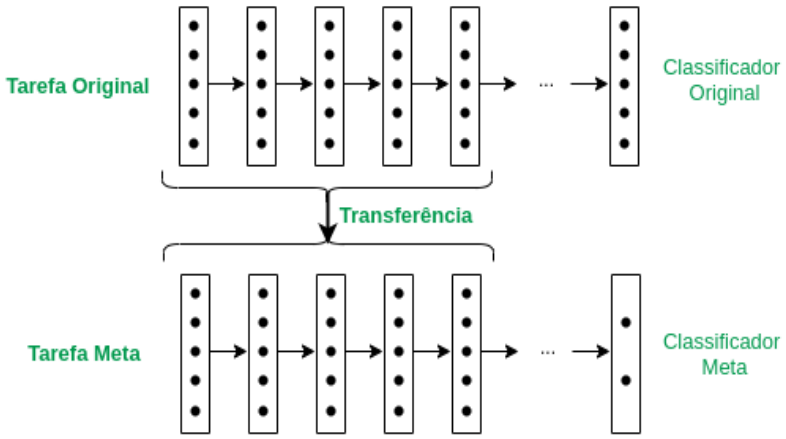



**Figura 2.5:** *Ilustração da transferência de aprendizado baseada em rede em que algumas camadas da rede treinada na tarefa original são transferidas para uma rede similar adaptada à tarefa meta. Adaptada de Tan et al. (2018).*

Na Figura 2.5, as camadas treinadas para uma tarefa são transferidas para uma rede similar a qual será adaptada para uma outra tarefa. Estas camadas transferidas poderão ter seus pesos atualizados durante o treino (*fine-tuning*) ou congeladas, ou seja, não serão ajustadas nas etapas de propagação para trás (*backpropagation*). A escolha de ajustar estas camadas está relacionada com a complexidade do modelo, quantidade de dados disponíveis na tarefa meta e similaridade entre os domínios.

É interessante notar na Figura 2.5 que não há transferência da última camada classificadora, pois esta camada é totalmente dependente da tarefa. No geral, neste tipo de transferência de aprendizado as camadas mais próximas à entrada da rede são mais agnósticas à tarefa, capturando representações mais gerais dos dados de entrada de forma a contribuírem no aprendizado de outras tarefas.

O uso desta técnica de TL se popularizou em DL nos domínios de imagem e texto. No contexto de imagens, geralmente, as camadas convolucionais são transferidas por serem extratores de características hierárquicas, visto que no início extraem estruturas simples e a partir destas estruturas são capazes de compor estruturas mais complexas, tornando-se mais específicas ao domínio a que foram treinadas.

# Capítulo 3

# Fundamentos: Compressão de Modelos

Neste capítulo, apresentaremos os conceitos fundamentais da área de compressão de modelos dando um foco especial para a técnica de *poda de parâmetros* (*model pruning*), para a qual apresentamos uma família de algoritmos que serão usados neste trabalho.

Em geral, redes neurais profundas possuem mais parâmetros que o necessário para a tarefa na qual foram treinadas e apresentam redundância em muitos de seus parâmetros, como demonstrado em Denil *et al.* (2013). Compressão de modelos é uma área que investiga técnicas de redução de redes neurais em termos de: quantidade de parâmetros, tempo de inferência, quantidade de operações de ponto flutuante por segundo (*FLoating-point Operations Per Second* — FLOPS) e tamanho do armazenamento em memória. Este tem sido um tema de grande interesse devido aos avanços nas áreas de carros autônomos, cidades inteligentes, visão computacional e processamento de linguagem natural. Além disso, a principal necessidade de se fazer a compressão de modelos é o armazenamento e execução de modelos de aprendizado profundo de forma eficiente em dispositivos com poder computacional limitado, como por exemplo câmeras de vigilância, *smartphones* e sistemas embarcados no geral (Verhelst e Moons, 2017).

De acordo com Cheng *et al.* (2017), existem quatro classes de técnicas de compressão de modelos: (i) fatoração de baixa ordem (*low-rank factorization*), (ii) compactação de filtros convolucionais (*compact convolutional filters*), (iii) destilação de conhecimento (*knowledge distillation*) e (iv) poda de parâmetros (*model pruning*). A fatoração de baixa ordem estima os parâmetros mais relevantes ao modelo a partir da decomposição de suas matrizes (Sainath *et al.*, 2013). A técnica de compactação de filtros convolucionais investiga a criação de estruturas que utilizam menos parâmetros que os usados em filtros convencionais, produzindo assim novas arquiteturas de redes convolucionais (Cohen e Welling, 2016). Já a destilação de conhecimento busca compartilhar uma função complexa aprendida por um modelo profundo ("rede professor") para um modelo mais compacto ("rede aluno") (Hinton, Vinyals *et al.*, 2015). E por fim, a poda de parâmetros visa eliminar parâmetros da rede de forma a não prejudicar o desempenho do modelo, sendo essa a técnica de compressão de modelos que usaremos nesse trabalho.

Poda de parâmetros (*model pruning*) é uma técnica de compressão de redes neurais amplamente utilizada em aplicações que envolvem aprendizado profundo, e consiste na remoção de parâmetros da rede de tal forma a minimizar o impacto em seu desempenho. Essa remoção é realizada de fato atribuindo-se o valor zero para os parâmetros selecionados para poda. Note que essa técnica é diferente da técnica de regularização *Dropout* (Srivastava *et al.*, 2014), em que a remoção estocástica de unidades (neurônios) da rede é realizada apenas durante o treinamento, de forma que essas unidades estarão presentes ao final do treinamento e em qualquer predição do modelo; na poda de parâmetros, quando removemos uma estrutura (unidade ou filtro), esta passa a ter parâmetros com valor zero para sempre, ou seja, esses parâmetros não são atualizados na etapa de *backpropagation*, nem considerados na inferência do modelo.

De acordo com Blalock *et al.* (2020), métodos de poda de parâmetros podem ser classificados em termos de:

- **Tipo**: define o tipo de poda de parâmetros, podendo ser *poda não-estruturada* (em que removemos apenas um parâmetro) ou *poda estruturada* (em que removemos um neurônio ou um filtro completo). Na poda estruturada, a rede neural resultante será densa, enquanto na poda não-estruturada, a rede neural resultante será esparsa;
- **Contexto**: define o contexto da seleção dos parâmetros para poda, podendo ser local (considera apenas os parâmetros de uma mesma camada) ou global (considera todos os parâmetros da rede);
- **Fluxo de execução (*scheduling*)**: diz respeito à intercalação entre etapas de poda e retreino, podendo ser feito de forma gradual (**poda iterativa**) ou de uma única vez (**poda direta**).

A Figura 3.1 ilustra dois fluxos de execução utilizados pelos métodos estudados neste trabalho que independem do tipo e contexto de poda. O diagrama da esquerda ilustra o método de *poda iterativa* que repete as etapas de poda e retreino por $i$ iterações; e o diagrama da direita ilustra o método de *poda direta*, em que a poda do modelo é realizada de uma só vez antes do retreino. Ambos os fluxos partem do treinamento da rede neural original ("aprender a conectividade") que irá definir a magnitude dos parâmetros da rede para a tarefa na qual a rede foi treinada. Note também que após a poda de parâmetros, ambos finalizam com uma etapa de retreino do modelo, permitindo que os parâmetros restantes se recuperem da poda (LeCun, Denker *et al.*, 1989).

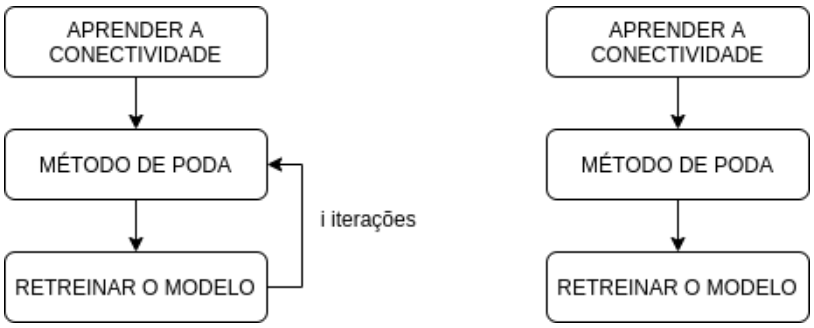



**Figura 3.1:** *Exemplos de fluxo de execução de poda de parâmetros: poda iterativa (esquerda) e poda direta (direita).*

## 3.1 Poda Não-Estruturada

O método de poda não-estruturada consiste na remoção de parâmetros sem considerar a estrutura (unidade ou filtro) na qual esses parâmetros pertencem, ou seja, *prioriza a poda individual de parâmetros tornando esparsa a estrutura da rede.*

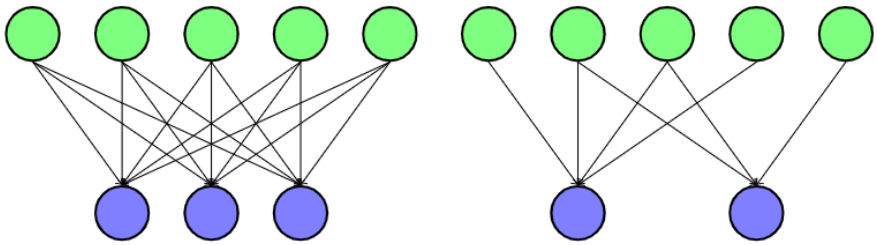


**Figura 3.2:** *Exemplo de poda não-estruturada. Rede neural antes (esquerda) e depois (direita) da poda de parâmetros.*

A Figura 3.2 ilustra o resultado da execução de um método de poda não-estruturada entre duas camadas densamente conectadas de uma rede neural. A figura da esquerda mostra uma camada com 5 unidades com todas as conexões possíveis para a próxima camada com 3 unidades. A figura da direita ilustra as mesmas camadas após uma poda não-estruturada. É interessante notar que, neste exemplo, além da remoção de alguns parâmetros (arestas), uma unidade inteira foi removida, pois todas as conexões que chegavam até ela foram removidas (por exemplo, a poda do neurônio central da camada inferior). Assim, uma poda não-estruturada pode levar à poda completa de uma estrutura, mesmo através da remoção individual dos parâmetros da rede.

### 3.1.1 Algoritmos de poda não-estruturada

Na literatura de poda não-estruturada, é possível encontrar diferentes formas para alcançar uma rede esparsa com baixo impacto no desempenho do modelo. Dentre eles destacamos três trabalhos importantes:

**Poda não-estruturada baseada na matriz hessiana.** Autores pioneiros na área, LeCun, Denker *et al.* (1989) e Hassibi e Stork (1993), propuseram um método de poda baseado na matriz hessiana da função de custo em que os parâmetros de menor valor são eliminados, considerando um contexto local ou global. Este critério utiliza a segunda ordem das derivadas parciais da função que se deseja aproximar para avaliar o impacto de cada parâmetro na convergência do modelo. Dado o custo computacional envolvido no cálculo da matriz Hessiana, esse método não é o mais indicado para o uso em redes profundas que possuem milhões de parâmetros.

**Poda não-estruturada iterativa baseada em magnitude.** Pensando na aplicação em redes neurais profundas, Han e Dally (2017) propuseram um método de poda baseado na magnitude dos parâmetros da rede. Esse método, ao invés de focar nos parâmetros que devem ser podados a cada iteração $i$, foca naqueles que permanecem na rede com valores diferentes de zero. Dado um tensor de parâmetros $\mathcal{W}$, este método recebe o porcentual

$\gamma$ de parâmetros que deverão ser mantidos e realiza a poda de uma porcentagem $1 - \gamma$ dos parâmetros de menor valor. Para isso, após a ordenação dos valores (magnitude) dos parâmetros, computa-se o percentil da iteração $(1 - \gamma)^i$, definindo assim um limiar da magnitude $\lambda$ para poda. A atribuição a seguir computa os valores dos parâmetros da rede na iteração de poda $i$:

$$\mathcal{W}^i \leftarrow \mathcal{W}^0 \odot \mathbb{1}(|\mathcal{W}^{i-1}| > \lambda), \tag{3.1}$$

em que $\mathcal{W}^0$ denota o tensor de valores dos parâmetros após o treinamento inicial da rede, $\mathcal{W}^i$ ($\mathcal{W}^{i-1}$) denota o tensor de valores dos parâmetros da rede após $i$ ($i - 1$) iterações do processo de poda. A expressão $\mathbb{1}(|\mathcal{W}^{i-1}| > \lambda)$ é uma função indicadora que assume valor 1 para os valores em módulo de $\mathcal{W}^{i-1}$ que forem maiores que o limiar $\lambda$ e valor 0 para os demais. O resultado desta função é um tensor binário que atua como uma máscara na multiplicação elemento a elemento (produto Hadamard), indicado pelo operador $\odot$, com $\mathcal{W}^0$, resultando no novo tensor de valores $\mathcal{W}^i$. O valor de porcentagem mantida $\gamma$ é um hiperparâmetro deste método de poda, podendo decair com o número de iterações ou se manter constante durante todo o processo. Apesar desta técnica ser relativamente simples ela é capaz de atingir níveis de esparsidade maior devido à intercalação da poda com o retreino, alcançando uma redução de 13× no número de parâmetros diferentes de zero na VGG-16 (Han, Pool *et al.*, 2015).

**Poda não-estruturada *winning ticket*.** Recentemente, Frankle e Carbin (2019) propuseram um novo critério de poda não-estruturada inspirada na poda iterativa por magnitude. Os autores formularam a hipótese de que existe uma subrede (*winning ticket*) que quando treinada separadamente com uma determinada inicialização, atinge um desempenho próximo à rede original. O algoritmo proposto é similar ao de Han e Dally (2017), exceto que ele não inicia com a rede treinada, mas com uma rede com pesos aleatórios. A cada iteração o modelo é reinicializado com os pesos aleatórios do modelo antes do primeiro treinamento, ou seja, a atualização deste método é definida por:

$$\mathcal{W}^i \leftarrow \mathcal{W}^r \odot \mathbb{1}(|\mathcal{W}^{i-1}| > \lambda), \tag{3.2}$$

sendo $\mathcal{W}^r$ a inicialização aleatória da rede. Apesar de apresentar bons resultados em redes como VGG e ResNet, este método iterativo é sensível à escolha da taxa de aprendizado (hiperparâmetro do treinamento de redes neurais). Os autores concluíram que só é possível encontrar *winning tickets* em redes profundas realizando um "aquecimento" da taxa de aprendizado (*learning rate warmup*), isto é, iniciando com uma taxa de aprendizado muito menor que a taxa definida como inicial até gradualmente atingi-la em algumas épocas de treinamento.

### 3.1.2 Poda Iterativa Não-Estruturada Baseada em Magnitude

O método de *poda iterativa não-estruturada baseada em magnitude* (Han e Dally, 2017) (Algoritmo 2), é um dos métodos de compressão de modelos adotados neste trabalho para transferência de aprendizado e, por isso, será apresentado nesta seção em mais detalhes.

**Algoritmo 2** Poda não-estruturada iterativa baseada em magnitude

**Entrada:** porcentagem mantida $\gamma$, modelo treinado $\mathcal{M}$, número de iterações de poda $n$

1: $\alpha \leftarrow 1$
2: $\mathcal{W}^0 \leftarrow$ Tensores de parâmetros de $\mathcal{M}$
3: **para** $i = 1, \ldots,$ n **faça**
4: $\quad \alpha \leftarrow \alpha * \gamma$
5: $\quad \mathcal{W}^{i-1} \leftarrow$ Tensores de parâmetros de $\mathcal{M}$
6: $\quad \lambda \leftarrow \text{ComputaPercentil}(|\mathcal{W}^{i-1}|, (1 - \alpha))$
7: $\quad \mathcal{P} \leftarrow \mathbb{1}[\, |\mathcal{W}^{i-1}| > \lambda \,]$ ▷ Máscara de poda
8: $\quad \mathcal{W}^i \leftarrow \mathcal{W}^0 \odot \mathcal{P}$ ▷ Produto elemento a elemento
9: $\quad \mathcal{M} \leftarrow \text{treina}(\mathcal{M}, \mathcal{W}^i)$ ▷ Inicializa os parâmetros com $\mathcal{W}^i$
10: **fim para**

**Devolve:** Modelo podado $\mathcal{M}$

O Algoritmo 2 recebe como entrada: um modelo $\mathcal{M}$, inicialmente treinado até convergência; e os hiperparâmetros $\gamma$ e $n$ (número de iterações de poda). O objetivo do algoritmo é induzir esparsidade no modelo a um passo constante $\gamma$ de forma que o número de parâmetros podados (parâmetros com valor zero) crescem monotonicamente a cada iteração. Assim, a cada iteração, calculamos a porcentagem acumulada $\alpha$ como $\alpha \leftarrow \alpha * \gamma$ (Linha 4). Por exemplo, considerando que $\gamma = 0.7$ é constante para todas as iterações, na décima iteração o percentual de parâmetros não nulos será de $0.7^{10} = 0.02824 = 2.82\%$.

No Algoritmo 2, $\alpha$ é inicializado com o valor 1 e $\mathcal{W}^0$ recebe todos os parâmetros do modelo treinado $\mathcal{M}$. A magnitude limiar correspondente à $1 - \gamma$ é calculado pela função *ComputaPercentil*$(|\mathcal{W}^{i-1}|, (1 - \alpha))$ (Linha 6). A máscara de poda $\mathcal{P}$ é um tensor binário de mesma dimensão que $\mathcal{W}$, que tem valor 1 em todas as posições em que o valor absoluto do parâmetro em $W^{i-1}$ é superior ao limiar (Linha 7). Após a construção da máscara, o modelo é inicializado com o produto de Hadamard entre o tensor de parâmetros $\mathcal{W}^0$ e a máscara (Linha 8). Antes de ser retreinado (Linha 9), o modelo é reinicializado com tensor de parâmetros podados $W^i$. Após as $n$ iterações de poda, o algoritmo devolve o modelo podado $\mathcal{M}$.

Uma das limitações deste método é definir a quantidade $n$ de iterações de poda como um hiperparâmetro de entrada do algoritmo. No entanto, o melhor ponto de compressão do modelo, que garanta a mesma acurácia (ou aproximada) do modelo original, depende de uma análise estatística mais detalhada. No Capítulo 4 propomos uma modificação neste algoritmo para incluir uma condição para a repetição da poda.

## 3.2 Poda Estruturada

O método de poda estruturada consiste na remoção de estruturas completas, ou seja, *prioriza a poda de um grupo de parâmetros que pertencem a uma estrutura do tipo um neurônio ou um filtro convolucional, modificando necessariamente a arquitetura do modelo.* Diferentemente da poda não-estruturada, a poda estruturada não induz esparsidade na rede, pois a remoção de estruturas completas implica em um novo modelo menor que se

mantém denso (Li *et al.*, 2017).

A Figura 3.3 ilustra o resultado da execução de um método de poda estruturada entre duas camadas densamente conectadas de uma rede neural. A rede da esquerda mostra uma camada com 5 unidades totalmente conectadas à próxima camada com 3 unidades. Já a imagem da direita ilustra as unidades remanescentes após uma poda estruturada. É interessante notar que as unidades não podadas permanecem com a mesma quantidade de conexões (parâmetros) que tinham antes de realizar a poda.

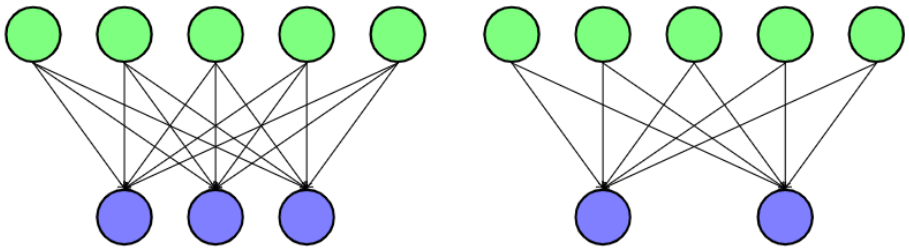


**Figura 3.3:** *Exemplo de poda do tipo estruturada. Rede neural antes (esquerda) e depois (direita) da poda de parâmetros.*

### 3.2.1 Algoritmos de Poda Estruturada

É possível encontrar na literatura diferentes formas para se realizar a poda estruturada com baixo impacto no desempenho do modelo, dentre as quais destacamos três trabalhos relevantes:

**Poda Estruturada baseada em Algoritmos Genéticos.** Anwar *et al.* (2015) foram pioneiros nesta área propondo métodos de poda estruturada com diferentes níveis de granularidade. Sabendo que a complexidade de um problema de poda de uma rede com $p$ parâmetros é da ordem de $\mathcal{O}(2^p)$ (i.e. cada parâmetro do modelo pode estar ativo com valor diferente de zero ou inativo com valor igual a zero), os autores propuseram o uso de Filtro de Partículas (Arulampalam *et al.*, 2002) combinado com um Algoritmo Genético chamado de EPF (*Evolutionary Particle Filter*)(Arulampalam *et al.*, 2002) para escolher máscaras de parâmetros candidatos à poda. Este método é capaz de podar estruturas completas, porém, esta abordagem de compressão de modelos não é capaz de atingir reduções extremas sem afetar o desempenho do modelo.

**Poda Estruturada baseada na norma $L_1$.** Li *et al.* (2017) seguiram uma abordagem diferente para poda estruturada, optando por analisar os valores dos parâmetros dos filtros de cada camada convolucional usando a norma $L_1$[1] para selecionar filtros redundantes e removê-los da rede. Neste método, os mapas de características (Seção 2.4, Capítulo 2) associados ao filtro podado, também são descartados. Ao contrário dos métodos apresentados na poda não-estruturada, os autores adotaram o método de poda direta a fim de economizar tempo com um único retreino. A Equação 3.3 mostra o cálculo realizado para cada um dos $i$ filtros de uma camada $l$ ($F_{l,i}$). Os valores $\phi_l$ são ordenados de forma crescente,

[1] https://pt.wikipedia.org/wiki/Norma_(matemática)

dos quais $(1 - \gamma)\%$ serão removidos. $vec$ é uma função que transforma um tensor em um vetor antes de aplicarmos a norma-$L_1$. $\gamma$, a porcentagem de parâmetros mantidos, é um parâmetro definido pelo usuário, podendo ser o mesmo para todas as camadas ou ajustado por camada de acordo com uma análise de sensibilidade (Li *et al.*, 2017).

$$\phi_{l,i} = \left\| vec(F_{l,i}) \right\|_1 \tag{3.3}$$

**Poda Estruturada *Network Slimming*.** Outro trabalho relevante na área de poda estruturada foi proposto por Z. Liu *et al.* (2017) que introduziu um termo adicional à função de custo do modelo referente a um fator de escala para cada filtro (ou neurônio) de cada camada. Os fatores de escala aprendidos para cada filtro durante o treinamento são usados como método de poda: é feita a poda de todos os filtros ou neurônios que estiverem abaixo de um determinado percentil.

### 3.2.2 Poda Estruturada baseada na norma $L_1$

O principal objetivo do método direto baseado na norma $L_1$ (Li *et al.*, 2017) é realizar uma poda direta de um percentual $1 - \gamma_c$ de filtros menos relevantes de cada camada convolucional $c$ da rede. A relevância de um filtro é considerada como a soma do módulo dos valores dos parâmetros que compõem cada filtro (norma $L_1$). Para isso, é feita uma avaliação prévia do impacto de diferentes intensidades de poda para escolher o percentual de poda $1 - \gamma_c$ por camada. Uma das principais vantagens do uso desta abordagem é evitar o custo computacional do processo iterativo: através de um fluxo de execução de poda direta o modelo é retreinado uma única vez no fim do processo de poda. Desta forma, selecionamos este método para ser modificado (Capítulo 4) e usado na análise proposta sobre compressão de modelos em transferência de aprendizado.

**Algoritmo 3** Poda estruturada baseada na Norma $L_1$

**Requer:** modelo $\mathcal{M}$ e lista de percentuais mantidos $\Gamma$

$C \leftarrow$ Lista todas as camadas convolucionais de $\mathcal{M}$
$j \leftarrow 0$
$novasCamadas \leftarrow$ listaVazia()
**para** $c$ in $C$ **faça**
    $ranks \leftarrow \sum |f_{c,i}|, \forall i \in [1, n]$ ▷ $n$ filtros na camada $c$
    $indice \leftarrow ranks$.ordenaIndices()
    $removeIndice \leftarrow$ Remove $(1 - \Gamma[j])\%$ dos menores valores de $indice$
    $camada \leftarrow$ Deleta em $c$ os índices de $removeIndice$
    $novasCamadas \leftarrow atualizaLista(novasCamadas, camada)$
    $j \leftarrow j + 1$
**fim para**
$\mathcal{M}' \leftarrow$ adaptaModelo($\mathcal{M}, novasCamadas$)
$\mathcal{M}' \leftarrow$ treina($\mathcal{M}'$) ▷ Retreina modelo até convergência

**Devolve:** Modelo podado $\mathcal{M}'$

O Algoritmo 3 descreve o processo para a poda estruturada de um modelo $\mathcal{M}$, inicialmente treinado até convergência. O algoritmo recebe como entrada um modelo $\mathcal{M}$ a ser

podado e a lista Γ, previamente definida, contendo o percentual de filtros a serem preservados em cada camada convolucional da rede. A lista $C$ recebe as camadas convolucionais de $\mathcal{M}$ (Linha 1) e *novasCamadas* é uma lista usada para armazenar as camadas podadas, inicialmente vazia. A poda completa do modelo (Linhas 4-11) é realizada em camadas. Para cada camada $c$ do modelo é feita uma ordenação dos filtros de acordo com a sua relevância (Linhas 5-6). Em seguida, de acordo com um percentual definido para a camada $c$, $\gamma_c \in \Gamma$, os filtros de menores índices são podados (Linha 7). Nas Linhas 8-9 o conjunto de *novasCamadas* é atualizado. Na Linha 12 há uma chamada à função de reconstrução do modelo (*adaptaModel*) para criar um modelo denso com as camadas convolucionais contendo os filtros remanescentes após a poda e as camadas não alteradas do modelo original $\mathcal{M}$.

A lista Γ na entrada do Algoritmo 3 está relacionada ao percentual de poda de cada camada convolucional. Se adotássemos um método de busca exaustiva pelo melhor conjunto de percentuais para um intervalo de 10 possibilidades ([0.1, 0.2, ..., 1.0]) teríamos um problema na ordem de $10^n$ combinações, sendo $n$ o número de camadas. Um aumento linear no número de camadas convolucionais leva a um aumento exponencial na busca pela poda ótima, logo é um problema intratável (Garey e Johnson, 1979). Conforme discutido em Li *et al.* (2017), é possível estimar o percentual ideal de poda para cada camada a partir de uma análise de sensibilidade.

Na análise de sensibilidade, é feita uma poda estrutural em cada camada convolucional (Equação 3.3) independente das demais; em seguida, o modelo é avaliado no conjunto de dados de teste, sem realizar um retreino, a fim de mensurar o impacto na acurácia no modelo podado com relação à acurácia do modelo original. A Figura 3.4 ilustra esta análise feita na VGG-16 com 13 camadas convolucionais, o que numa busca exaustiva implicaria na avaliação de $10^{13}$ combinações distintas de poda.

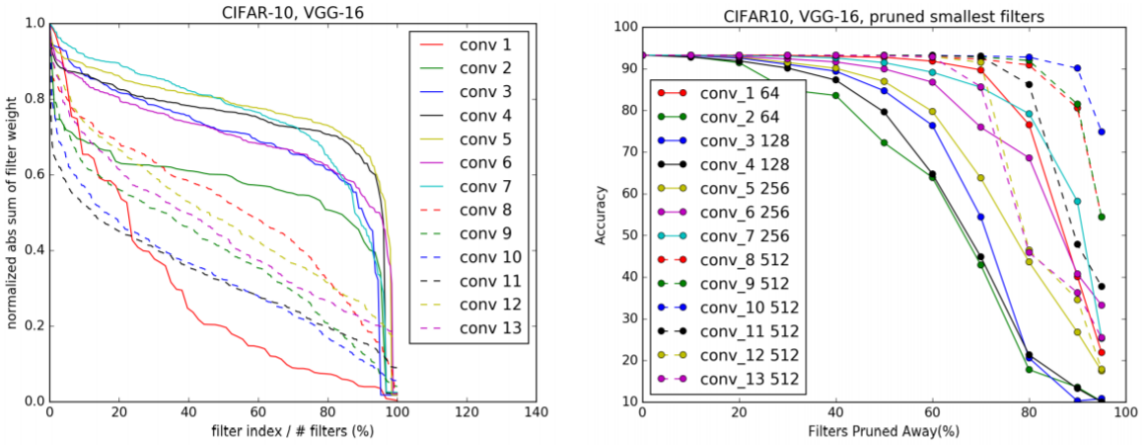



**Figura 3.4:** *Análise de sensibilidade da VGG-16 no CIFAR-10. Ordenação dos filtros de acordo com a sensibilidade $\phi$ para cada camada convolucional (esquerda). Efeitos de diferentes percentuais de poda na acurácia (direita). Fonte: Li et al., 2017.*

A Figura 3.4 da esquerda mostra que camadas com um grande percentual de filtros com valores mais próximos de 1.0 indicam são mais sensíveis à poda (direita), isto é, a remoção de um percentual de seus filtros causa mais impacto no desempenho da rede. Na figura da direita cada cuva representa uma camada, o eixo-y informa a acurácia nos dados

de teste para cada percentual de filtros preservados no eixo-x. Camadas convolucionais mais próximas ao final da rede (curvas tracejadas) possuem o maior número de filtros redundantes, permitindo uma extensa poda sem grandes impactos na acurácia do modelo. Por outro lado, camadas mais próximas à entrada da rede, quando modificadas, impactam mais na acurácia (ex. conv_2). Li *et al.* (2017) analisaram graficamente essas curvas de sensibilidade para definir o percentual de poda por camada que seria utilizado como entrada do Algoritmo 3.

O diferencial da poda estruturada em comparação à não-estruturada é a redução do modelo de uma forma não esparsa, levando a vantagens computacionais que podem ser usufruídas sem a necessidade de *hardware* ou *software* específicos.

Porém, é mais difícil determinar um percentual de poda a priori para a poda estruturada, pois não sabemos qual a relevância de cada camada sem uma análise de sensibilidade. De forma que podar 30% dos filtros menos relevantes em uma camada pode ter um impacto diferente do que podar o mesmo percentual de outra camada. Já na poda não-estruturada num contexto global por analisarmos todas as camadas de uma vez, conseguimos garantir que aquele percentual de pesos podados era menos significativo para o modelo como um todo.

## 3.3 Compressão de Modelos com Aprendizado por Reforço

Até a presente seção, os dois métodos de poda apresentados seguiam critérios ad hoc, dirigidos pela avaliação de desempenho do modelo resultante. Na seção anterior foi apresentada uma análise para definir o percentual ótimo de poda por camada porém, como as camadas de uma rede neural não são independentes, realizar uma poda observando critérios ótimos locais não garante um ótimo global Y. He *et al.* (2018a). Encontrar uma poda ótima global é um problema NP-difícil devido ao espaço de poda ser exponencial em termos dos parâmetros (Dong *et al.*, 2017). Para remediar essa limitação, Y. He *et al.* (2018b) e M. Gupta *et al.* (2020) propõem a formulação do problema de podar uma rede neural como um Processo de Decisão Markoviano (*Markov Decision Process* - MDP) (Puterman, 1995) e o uso de técnicas modernas e eficientes de aprendizado por reforço profundo (*Deep Reinforcement Learning* - DRL) (Silver *et al.*, 2017).

O algoritmo PuRL (*Pruning using Reinforcement Learning*) (M. Gupta *et al.*, 2020) realiza uma poda não-estruturada de modelos profundos usando como base o algoritmo DQN (*Deep Q-Netwok*): uma versão de aprendizado por reforço profundo do algoritmo Q-Learning (1). O critério de poda adotado foi proposto por Han, Pool *et al.* (2015) e consiste numa poda não-estruturada local baseada em magnitude com um valor limiar determinado por um parâmetro de qualidade ($\alpha$) multiplicado pelo desvio-padrão ($\sigma$) dos pesos $W_l$ de uma camada $l$ (similar ao critério de poda não-estruturada descrito na Seção 4.1 porém, ao invés de considerar os parâmetros de toda rede, este critério considera apenas os parâmetros de uma camada por vez). Neste algoritmo, uma ação de poda faz uma reescala do parâmetro $\alpha$ afetando diretamente quais parâmetros serão preservados em cada camada $l$, como mostramos a seguir:

$$W_l = W_l \odot \mathbb{1}(|W_l| < \alpha\sigma(W_l)) \tag{3.4}$$

Note que neste critério, uma vez que se trata de uma medida de dispersão, o objetivo é manter os parâmetros de uma camada $l$ com uma magnitude menor que o desvio-padrão. No PuRL M. Gupta *et al.* (2020), um episódio consiste na poda sequencial de todas as camadas do modelo, terminando na última camada.

### 3.3.1 MDP para poda não-estruturada

Considere um MDP dado pela tupla $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \beta \rangle$ (Definição 1). Para compreendermos como modelar o problema de poda não-estruturada como um MDP, explicamos a seguir cada componente na perspectiva do PuRL.

#### Espaço de Estados

A representação do estado $s \in \mathcal{S}$ é dada pela tupla $s = \langle l, acc, p \rangle$, sendo $l$ o índice da última camada podada, $acc$ é a acurácia obtida com a poda/retreino do modelo e $p$ é o percentual de parâmetros podados desde o início do episódio. O estado inicial de um episódio consiste na tupla $s_0 = \langle 0, 0, 0 \rangle$.

#### Espaço de Ações

No MDP para poda, a ação $a \in \mathcal{A}$ representa o fator multiplicativo $\alpha$ do critério de poda não-estruturada descrito anteriormente, com $\alpha \in \{0.0, 0.2, 0.4, ..., 2.2\}, \forall a \in \mathcal{A}$. Quando aplicamos a ação $a$ num estado $s = \langle l, acc, p \rangle$ gerando o próximo estado $s' = \langle l + 1, acc', p' \rangle$, sendo $p'$ calculado de acordo com $\alpha$ na Equação 3.4.

#### Função de Recompensa

A função de recompensa (ou custo, no caso em que o valor é negativo) quantifica o efeito de uma ação $a \in \mathcal{A}$ que foi aplicada em um estado $s$ levando a um estado $s'$. A Equação 3.5 descreve a função de recompensa, em que $A(s')$ e $P(s')$ representam a acurácia e percentual de poda em $s'$, respectivamente, e $T_A$ e $T_P$ são a acurácia e percentual de poda metas (*targets*) definidos pelo usuário.

$$\mathcal{R}(s, a, s') = -5 \left( max\left(1 - \frac{A(s')}{T_A}, 0\right) + max\left(1 - \frac{P(s')}{T_P}, 0\right)\right) \tag{3.5}$$

Note que na Equação 3.5 a recompensa negativa penaliza estados sucessores $s'$ cuja razão de sua acurácia ($A(s')$) e a acurácia meta ($T_a$) é inferior a 1 (o mesmo acontece para a segunda parcela que também penaliza estados em que a razão do percentual da poda ($P(s')$) e a percentual meta de poda ($T_p$) é inferior a 1).

Um dos grandes diferenciais do PuRL, quando comparado ao trabalho proposto por Y. He *et al.* (2018b), são as recompensas densas. A cada etapa de poda de uma camada o modelo é retreinado com um pequeno conjunto de dados por uma época e então avaliado no conjunto de teste. Desta forma, a relação entre o impacto da escolha de $\alpha$ daquela

camada no desempenho do modelo fica mais evidente ao agente, levando a um menor tempo de convergência.

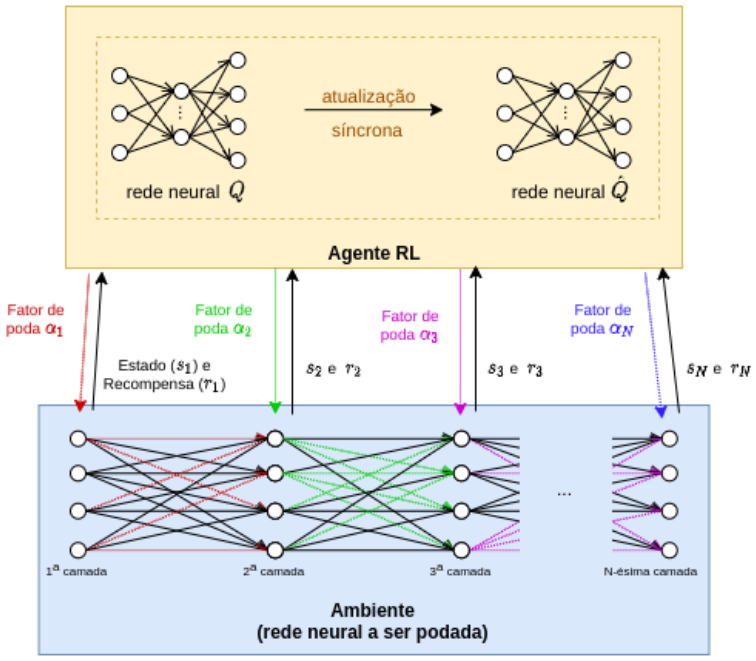



**Figura 3.5:** *A figura detalha a etapa de treinamento da DQN com destaque para as duas redes neurais internas ao agente RL (retângulo amarelo). A rede Q fornece a política exploratória do ambiente e a rede $\hat{Q}$ fornece a política de poda aprendida. Abaixo, o ambiente (retângulo azul) é representado pela rede neural que se deseja podar camada a camada.*

A Figura 3.5 ilustra a interação do agente RL com o ambiente, que é a primeira etapa do método PuRL. Essa primeira etapa consiste no treinamento do algoritmo DQN (*Deep Q-Network*) (Mnih *et al.*, 2015) em que um agente de aprendizado interage com o ambiente, que neste caso é a rede neural a ser podada, enviando um valor limiar ($\alpha$) da camada e recebe um novo estado da rede juntamente com uma recompensa. O ambiente é responsável por: realizar a poda de uma camada específica $l$, retreinar o modelo por uma época, avaliar o percentual de poda e recalcular a acurácia desse modelo no conjunto de dados de teste. Neste contexto um episódio consiste em podar todas as camadas do modelo sequencialmente. Finalizada a etapa de treinamento da DQN, o modelo original é podado seguindo a política aprendida pelo agente, e então o modelo podado é retreinado até convergência (*fine-tune*).

O Algoritmo 4 descreve o processo de poda não-estruturada utilizando aprendizado por reforço (M. Gupta *et al.*, 2020), detalhando como o ambiente responde a uma ação do agente. É interessante notar que, ao contrário das outras técnicas descritas (Seção 3.1.1), o modelo passa após a poda por um retreino mínimo de apenas uma época (Linha 7) e com um pequeno conjunto de dados, para não onerar o tempo de aprendizado do agente.

Portanto, formular o problema de poda de parâmetros como um MDP permite uma melhor exploração do espaço de busca, sem a necessidade de definir um número de iterações para a poda ou realizar uma análise de sensibilidade das camadas. Porém, no PuRL o modelo

**Algoritmo 4** Poda de parâmetros usando aprendizado por reforço - PuRL

**Entrada:** modelo treinado $\mathcal{M}$, número de episódios *max_episodios*

$episodios \leftarrow 0$
**enquanto** $episodios \leq max\_episodios$ **faça**
    $modelo \leftarrow \mathcal{M}$
    **para** cada camada $l$ em $modelo$ **faça**
        $a_l \leftarrow$ amostra uma ação do agente
        Poda camada $l$ tomando a ação $a_l$
        Retreina o $modelo$ em uma amostra dos dados por 1 uma época
        Calcula a recompensa e o próximo estado baseado na esparsidade resultante e acurácia
        Devolve a recompensa e o novo estado para o agente
    **fim para**
    $episodios \leftarrow episodios + 1$
**fim enquanto**
$modelo \leftarrow \mathcal{M}$
Poda $modelo$ usando o agente treinado (média de 5 episódios)
Retreina o $modelo$

**Devolve:** $modelo$ podado com base na política aprendida

resultante da poda ainda é esparso, necessitando de recursos otimizados para alcançar eficiência computacional.

Este algoritmo e a modelagem do ambiente foram implementados neste trabalho e discutidos no Capítulo 6. Além disso, no Capítulo 4 propomos uma modificação baseada em poda estruturada, chamada de StructPuRL.

# Capítulo 4

# Compressão de Modelos: Melhorias e Extensões

Neste capítulo, são feitas propostas de melhorias e extensões nos algoritmos de poda não-estruturada e estruturada descritas no Capítulo 3. Além disso, propomos um novo método de poda estruturada, o StructPuRL, que consiste em uma extensão do PuRL (Seção 3.3).

## 4.1 Modificações na Poda Não-Estruturada

O método de *poda iterativa não-estruturada baseada em magnitude* (Han e Dally, 2017), descrito no Algoritmo 2, não especifica de maneira objetiva um critério de comparação entre as diferentes escolhas de poda ou qual é o melhor ponto de parada do algoritmo. Conforme descrito na Seção 3.1.2 podemos utilizar o critério do "cotovelo" que consiste na inspeção visual de um gráfico *acurácia* × *percentual de pesos preservados* para identificar um ponto máximo no eixo-$x$ (percentual de pesos diferentes de zero) em que não há uma alteração significativa da acurácia (Figura ). Como condição de parada dessa análise comparativa, é adotado um parâmetro $n$ de número de iterações da poda, pré-definido pelo usuário. Nesse caso fica implícito que a poda iterativa deve ser conduzida até antes do desempenho do modelo ser muito afetado pela poda.

A Figura 4.1 ilustra um exemplo do gráfico usado no critério de "cotovelo"e apresenta em destaque o nível de poda do modelo. A reta em vermelho indica que com apenas 1.98% da quantidade de pesos originais do modelo é possível atingir uma acurácia de cerca de 55%, que se equivale à acurácia alcançada pelo modelo com 100% de seus pesos preservados (não zerados). Porém, essa variação na acurácia pode ser resultante da estocasticidade inerente à otimização não-convexa da própria rede, ao invés de um ganho real no desempenho.

A comparação simples de acurácias do critério do "cotovelo" leva a uma pergunta crítica: *As diferenças entre as acurácias do modelo original e do modelo podado são estatisticamente relevantes?* Da forma com que este teste é conduzido, um modelo seria julgado melhor do que o outro apenas por apresentar uma acurácia maior, independente de sua significância estatística. Logo, surge a necessidade de comparar estatisticamente os modelos. Nesta

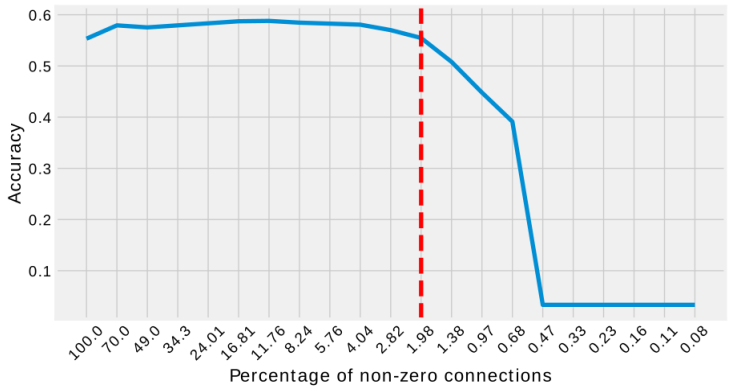



**Figura 4.1:** *Exemplo de gráfico acurácia × percentual de pesos preservados, que retrata a acurácia no conjunto teste para cada nível de redução alcançado aplicando uma poda iterativa.*

seção, especificamos um critério estatístico que, até onde sabemos, não foi aplicado à tarefa de poda de modelos de acordo com a literatura da área. Este critério também permite especificarmos um ponto de parada robusto.

### 4.1.1 Critério do teste-t pareado bayesiano

Como citado em Benavoli *et al.* (2017), uma prática comum na comunidade científica para a comparação empírica de observações é a realização do Teste de Significância de Hipótese Nula (*Null Hypothesis Significance Testing* — NHST). Nesse tipo de teste existe uma hipótese nula ($H_0$), que será testada, e uma hipótese alternativa ($H_a$) que contradiz a hipótese nula. O objetivo do teste é averiguar, dado certo nível de significância, se a condição observada pode ser atribuída ao acaso, ou seja, não é estatisticamente significante.

Mesmo sendo amplamente adotado, o NHST não é o teste mais apropriado para comparar classificadores de aprendizado de máquina (Benavoli *et al.*, 2017). Isso pois a análise do $p$-valor é muito rígida (rejeitar ou não rejeitar $H_0$), inviabilizando quantificar a superioridade de um modelo em relação a outro, ou averiguar o grau de equivalência entre eles. Dado o interesse em estimar probabilidades na comparação de dois modelos, optou-se pela utilização de uma técnica bayesiana.

O teste-t pareado bayesiano (*Bayesian correlated t-test*), proposto por Corani e Benavoli (2015), é utilizado para comparar dois modelos de classificação avaliados através da acurácia por *fold* na validação cruzada. O teste resulta numa distribuição a posteriori que estima a diferença média na acurácia entre os classificadores. Além da comparação de desempenhos de maneira probabilística, o teste permite definir uma região de equivalência prática (*Region of Practical Equivalence* — ROPE) entre os classificadores, geralmente no valor de 1% (Benavoli *et al.*, 2017). Supondo um classificador $A$ e um classificador $B$, podemos estimar as seguintes probabilidades a partir da distribuição a posteriori:

- $P(A >> B)$, que indica que o modelo $A$ é praticamente melhor que o modelo $B$;
- $P(A = B)$, indicando que para aquele intervalo, os modelos são praticamente equivalentes;
- $P(A << B)$, que indica que o modelo $B$ é praticamente melhor que o modelo $A$.

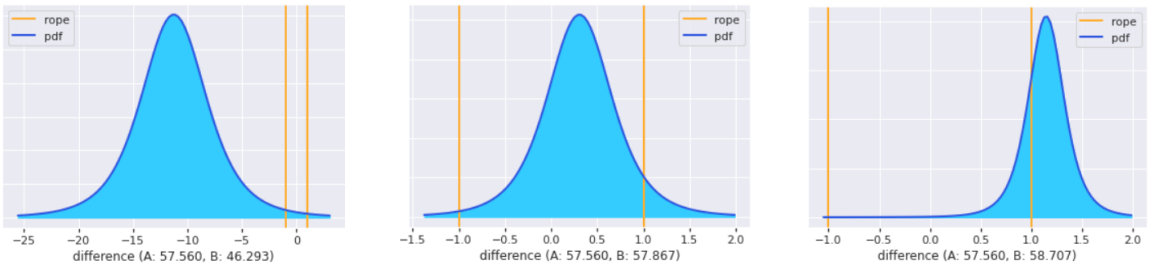



**Figura 4.2:** *Gráficos de três testes distintos usando teste-t pareado bayesiano. Os valores entre parênteses no rodapé de cada gráfico são as médias das acurácias dos classificadores avaliados. As linhas verticais delimitam a região de equivalência prática em que a diferença média não é superior a* ±1%.

Na Figura 4.2 o gráfico da esquerda ilustra o caso em que o modelo $A$ é evidentemente melhor que o modelo $B$ com uma probabilidade de 98%. Já no gráfico da direita, a densidade está mais concentrada à direita do intervalo de equivalência, levando à conclusão que existe uma chance maior do modelo $B$ ser melhor que o modelo $A$ (76%), porém não devemos desprezar a área dentro do intervalo de equivalência, que neste caso corresponde a 23%. O gráfico central ilustra a situação em que podemos afirmar que os modelos são praticamente equivalentes, considerando que em 92% das vezes a diferença na acurácia entre eles será menor que 1%.

### 4.1.2 Poda não-estruturada iterativa usando teste-t pareado bayesiano

O Algoritmo 5 detalha a forma com que o teste-t pareado bayesiano é usado para comparar um modelo sem poda com um modelo podado.

**Algoritmo 5** ContinuarPodaIterativa

**Requer:** conjuntos de $treino$ e $teste$, modelo sem poda $M_0$ e um modelo podado $M_x$

1: $stratifiedFolds \longleftarrow$ conjunto de 5 quebras estratificadas dos dados de $treino$
2: $L_0, L_x \longleftarrow$ listaVazia()
3: **para** $fold$ em $stratifiedFolds$ **faça**
4: $\quad t_0, t_x \longleftarrow$ TreinaModelo($M_0$, $fold$), TreinaModelo($M_x$, $fold$)
5: $\quad acc_0, acc_x \longleftarrow$ Acurácia($t_0$, $teste$), Acurácia($t_x$, $teste$)
6: $\quad L_0, L_x \longleftarrow$ atualizaLista($L_0$, $acc_0$), atualizaLista($L_x$, $acc_x$)
7: **fim para**
8: $pleft,\ prope,\ pright \longleftarrow$ TesteTPareadobayesiano($L_0$, $L_x$, $rope$ = 0.01)
9: $continue \longleftarrow Falso$
10: **se** $prope + pright > 0.5$ **então**
11: $\quad continue \longleftarrow Verdadeiro$
12: **fim se**
13: **devolve** $continue$

O Algoritmo 5 tem acesso aos conjuntos de treino e teste da tarefa original e recebe como entrada o modelo treinado sem poda (Seção 5.2) e o modelo podado em uma das iterações da poda não-estruturada. Na Linha 1, a partir do conjunto de treino são gerados 5 subconjuntos disjuntos amostrados de forma estratificada. Cada um desses subconjuntos

é utilizado no treino de ambos os modelos (Linha 4). Após treinados, os modelos são avaliados no conjunto de teste (Linha 5) e suas acurácias por *fold* são adicionadas às respectivas listas (Linha 6). A Linha 8 consiste em realizar o teste-t pareado bayesiano para as acurácias obtidas com o modelo original ($M_0$) em relação às acurácias do modelo podado ($M_x$), considerando uma região de equivalência de 1%.

Em posse das 3 probabilidades, o critério de escolha adotado é que a soma das probabilidades dos modelos serem equivalentes (*prope*) e a do modelo podado ser praticamente melhor que o original (*pright*) deverá ser maior que 50% (Linha 10). Isso pois, queremos um modelo podado que tenha um desempenho equivalente ou superior ao modelo sem poda. A rotina encerra-se devolvendo VERDADEIRO caso essa condição seja satisfeita ou FALSO caso contrário.

O Algoritmo 6 é uma modificação do Algoritmo 2 de *poda não-estruturada baseada em magnitude* (Seção 3.1.2) que usa o teste-t pareado bayesiano e adiciona um critério de parada. Esta versão do algoritmo foi adotada nos experimentos deste trabalho.

**Algoritmo 6** Poda não-estruturada iterativa com condição de parada

**Entrada:** porcentagem mantida $\gamma$, modelo $M_0$

1: $\alpha \longleftarrow 1$
2: $\mathcal{W}^0 \longleftarrow$ Tensores de parâmetros de $M_0$
3: $continue \longleftarrow Verdadeiro$
4: $M_x \longleftarrow M_0$
5: **enquanto** $continue$ **faça**
6: $\quad \alpha \longleftarrow \alpha * \gamma$
7: $\quad M_{best} \longleftarrow M_x$
8: $\quad \mathcal{W}^{atual} \longleftarrow$ Tensores de parâmetros de $M_x$
9: $\quad \lambda \longleftarrow$ ComputaPercentil($|\mathcal{W}^{atual}|$, $(1 - \alpha)$)
10: $\quad \mathcal{P} \longleftarrow \mathbb{1}[\ |\mathcal{W}^{atual}| > \lambda\ ]$ ▷ Máscara de poda
11: $\quad \mathcal{W}^{novo} \longleftarrow \mathcal{W}^0 \odot \mathcal{P}$ ▷ Produto elemento a elemento
12: $\quad M_x \longleftarrow$ treina($M_0, \mathcal{W}^{novo}$) ▷ Inicializa os parâmetros com $\mathcal{W}^{novo}$
13: $\quad continue \longleftarrow$ ContinuarPodaIterativa($M_0, M_x$)
14: **fim enquanto**

**Devolve:** Modelo podado $M_{best}$

A grande diferença deste novo algoritmo para o Algoritmo 2 (Seção 3.1.2) está na seleção automática do melhor modelo podado ($M_{best}$) de acordo com o teste-t pareado bayesiano (Linha 13). Neste caso o melhor modelo maximiza o percentual de poda e mantém a condição $prope + pright > 0.5$ verdadeira.

## 4.2 Modificações na Poda Estruturada

O método de *poda estruturada baseada na Norma* $L_1$ (Li *et al.*, 2017) descrito no Algoritmo 3 (Seção 3.2.2) depende da definição de percentuais de filtros preservados por camada ($\Gamma$). Os autores afirmam que estes percentuais podem ser obtidos através de uma análise de sensibilidade das camadas convolucionais. Porém, da mesma forma que na poda

iterativa não-estruturada, a escolha dos percentuais é feita pela análise gráfica das curvas de sensibilidade, tornando a tomada de decisão subjetiva.

Em vista disso, modificamos o método de poda estruturada trazendo a análise de sensibilidade para dentro do algoritmo e definindo um critério baseado na diferença em pontos percentuais da acurácia do modelo antes da poda e após a poda de uma de suas camadas. O Algoritmo 7 detalha o processo de construção da lista Γ de percentuais.

**Algoritmo 7** AnaliseDeSensibilidade

**Requer:** modelo $\mathcal{M}$, conjunto de $teste$, diff
$C \leftarrow$ Lista todas as camadas convolucionais de $M$
$acc^{orig} \leftarrow$ CalculaAcuracia($teste, M$)
$\Gamma \leftarrow$ listaVazia()
**para** $c$ in $C$ **faça**
    $best^c \leftarrow 1$
    **para** $\gamma$ in [0.9, ..., 0.1] **faça**
        $M^p \leftarrow$ PodarModelo($M, c, \gamma$)
        $acc^{new} \leftarrow$ CalculaAcuracia($teste, M^p$)
        **se** $acc^{orig} - acc^{new} >$ diff **então**
            **interrompe**
        **fim se**
        $best^c \leftarrow \gamma$
    **fim para**
    $\Gamma \leftarrow$ atualizaLista($\Gamma, best^c$)
**fim para**
**Devolve:** Lista de percentuais preservados por camada convolucional Γ

O Algoritmo 7 tem acesso ao conjunto de teste da tarefa original e recebe como entrada o modelo original e o valor máximo do delta ($diff$) entre as acurácias do modelo sem poda e do podado. O algoritmo realiza automaticamente a escolha dos percentuais de poda analisando o impacto que a retirada de alguns filtros tem em cada camada sem a realização de um retreino após a poda. Optamos por podar sem retreinar o modelo durante a análise de sensibilidade a fim de minimizar o custo computacional. Este algoritmo é utilizado pelo método de *poda estruturada baseada na norma-$L_1$ com análise de sensibilidade* descrito no Algoritmo 8.

O Algoritmo 8 realiza a poda estruturada de $M$ considerando a quantidade de filtros preservados por camada obtidos pela *AnaliseDeSensibilidade* sem depender de uma escolha manual. Uma desvantagem deste método mesmo com esta modificação é que a análise de sensibilidade é feita camada a camada de forma independente, desconsiderando o impacto da poda entre as camadas.

**Algoritmo 8** Poda estruturada baseada na norma-$L_1$ com análise de sensibilidade

**Requer:** modelo $M$, diff

$C \leftarrow$ Lista todas as camadas convolucionais de $M$
$\Gamma \leftarrow$ AnaliseDeSensibilidade($M, diff$)
$j \leftarrow 0$
$novasCamadas \leftarrow$ listaVazia()
**para** $c$ in $C$ **faça**
    $ranks \leftarrow \sum |f_{c,i}|, \forall i \in [1, n]$ ▷ $n$ filtros na camada $c$
    $indice \leftarrow OrdenaIndices(ranks)$
    $removeIndice \leftarrow$ Remove $(1 - \Gamma[j])\%$ dos menores valores de $indice$
    $camada \leftarrow$ Deleta em $c$ os índices de $removeIndice$
    $novasCamadas \leftarrow atualizaLista(novasCamadas, camada)$
    $j \leftarrow j + 1$
**fim para**
$M' \leftarrow$ adaptaModelo($M, novasCamadas$)
$M' \leftarrow$ treina($M'$) ▷ Retreina modelo até convergência

**Devolve:** Modelo podado $M'$

## 4.3 Agente de Aprendizado por Reforço para Poda Estruturada - StructPuRL

O primeiro trabalho, até onde se tem conhecimento, a considerar poda estruturada como um problema de aprendizado por reforço foi Y. He *et al.* (2018a) usando o algoritmo DDPG (*Deep Deterministic Policy Gradient*) com ações contínuas. Os autores justificam o uso do espaço contínuo devido à observação empírica da sensibilidade da acurácia em relação à esparsidade da camada. Porém, como reportado em Stopforth e Moodley (2019) o algoritmo para ações contínuas DDPG tem um desempenho inferior se comparado à sua contraparte discreta, o algoritmo DQN, para um número limitado de episódios. O que indica que a modelagem contínua, apesar de mais fina, requer mais episódios e controle de hiperparâmetros durante o treinamento se comparada a um espaço discreto.

A poda estruturada por aprendizado por reforço (*Structured Pruning using Reinforcement Learning* — StructPuRL) é uma das contribuições deste trabalho, e é construída com base no PuRL propondo modificações no critério de poda e no espaço de ações. O objetivo desta modificação é endereçar um dos grandes problemas da abordagem estruturada que é a escolha do percentual de poda de cada camada convolucional, do ponto de vista de um agente RL. Utilizando o arcabouço da Seção 3.3 com algumas alterações, é possível considerar o efeito de uma poda local nas demais camadas, diferentemente do que é feito na análise de sensibilidade, permitindo ao agente aprender uma política de poda globalmente eficiente.

Toda a estrutura do MDP apresentada na Seção 3.3.1 é reaproveitada, com exceção do espaço de ações. Neste caso, o espaço de ações especifica o percentual de filtros que serão preservados, em que $\alpha \in \{0.1, 0.2, 0.3..., 1.0\}, \forall a \in \mathcal{A}$. No caso mais extremo de poda estruturada, apenas 10% dos filtros são preservados, sendo também possível a preservação de todos os filtros de uma camada com $\alpha = 1$.

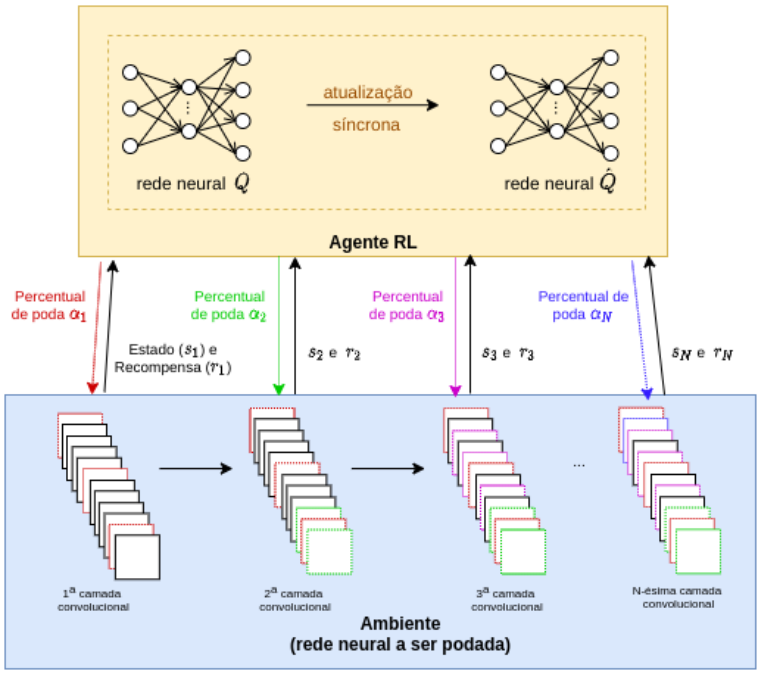



**Figura 4.3:** *A figura detalha a etapa de treinamento da DQN com destaque para as duas redes neurais internas ao agente RL (retângulo amarelo). A rede Q fornece a política exploratória do ambiente e a rede $\hat{Q}$ fornece a política de poda aprendida. Abaixo, o ambiente (retângulo azul) é representado pelas camadas convolucionais da rede neural que se deseja podar camada a camada.*

A Figura 4.3, assim como a Figura 3.5, ilustra a interação do agente RL com o ambiente. A grande diferença está no ambiente que passa a executar as ações do agente RL de forma estruturada, ou seja, elimina um percentual $\alpha$ de filtros por camada ao invés de parâmetros individuais. Neste contexto um episódio consiste em podar todas as camadas convolucionais da CNN sequencialmente.

Para viabilizar a poda estruturada de um agente RL, o critério de poda descrito na Equação 3.5 foi modificado pela Equação 3.3 em que os valores $\phi_{l,i}$ são calculados e ordenados para todos os filtros $i$ da camada $l$ e os filtros $(1 - \alpha)\%$ menos relevantes são removidos.

O pseudo-código 9 descreve o algoritmo DQN aplicado à poda estruturada de uma rede neural convolucional.

**Algoritmo 9** DQN - StructPuRL

**Requer:** modelo: $M$, total de episódios: $E$, tamanho da memória: $N$, quantidade de passos entre atualizações da rede meta: $U$, tamanho do mini-lote: $B$, parâmetro da política exploratória: $\epsilon$

1: $C \leftarrow$ Quantidade de camadas convolucionais em $M$
2: Inicializa a *memória de repetição* $D$ de tamanho $N$
3: Inicializa a rede neural de treinamento $Q$ com parâmetros aleatórios $\theta$
4: Inicializa a rede neural meta $\hat{Q}$ com parâmetros $\theta^{-} = \theta$
5: **para** episode = 1, $E$ **faça**
6: Inicializa com estado inicial $s_1 = <0, 0, 0>$
7: $M^{\text{episode}} \leftarrow M$
8: **para** t = 1, $C$ **faça**
9: Seleciona a ação:

$$a_t = \begin{cases} \text{argmax}_a Q(s_t, a; \theta) & \text{com probabilidade } 1 - \epsilon \\ \text{ação aleatória} & \text{com probabilidade } \epsilon \end{cases}$$

10: $M^{\text{episode}}, r_t, s_{t+1} \leftarrow$ ExecutaPoda($M^{\text{episode}}$, $a_t$, $s_t$)
11: Armazena transição $(s_t, a_t, r_t, s_{t+1})$ em $D$
12: Amostra um mini-lote $B$ de transições $(s_t, a_t, r_t, s_{t+1})$ de $D$
13: Define:

$$y_j = \begin{cases} r_j, & \text{se o episódio terminar no passo } j+1 \\ r_j + \beta \text{max}_{a'} \hat{Q}(s_{j+1}, a'; \theta^{-}), & \text{nos demais passos} \end{cases}$$

14: Calcula o erro:

$$\mathcal{L} = \frac{1}{B} \sum_{j=0}^{N-1} \left( y_j - Q(s_j, a_j; \theta) \right)^2$$

15: Atualiza os parâmetros de $Q$ minimizando $\mathcal{L}$ com gradiente descendente
16: $\hat{Q} = Q$, a cada $U$ passos
17: **fim para**
18: **fim para**

**Devolve:** Rede neural $\hat{Q}$

A Linha 1 define a quantidade de camadas convolucionais que serão podadas, dimensionando assim a duração de um episódio neste domínio. A Linha 2 inicializa a fila de memória $D$ das transições executadas no ambiente com capacidade para armazenar até $N$ transições. As Linhas 3 – 4 inicializam aleatoriamente duas redes neurais idênticas em que a única diferença está na periodicidade de atualização dos pesos. A rede $Q$ é atualizada a cada passo de aprendizado e a rede meta $\hat{Q}$ recebe os pesos de $Q$ a cada $U$ passos de treinamento, garantindo uma maior estabilidade no aprendizado da função Q.

A cada novo episódio o estado $t = 1$ é restaurado para a configuração inicial e o

modelo que será podado sequencialmente durante o episódio é restaurado para a estrutura original do modelo sem poda (Linhas 6 – 7). Para cada passo durante o episódio uma ação é escolhida seguindo alguma política exploratória gulosa no limite do infinito (*Greedy in the Limite of Infinite Exploration* - GLIE). Nesse algoritmo foi utilizada a $\epsilon$–greedy que com probabilidade $\epsilon$ amostra aleatoriamente uma ação (explora o ambiente) e com probabilidade complementar seleciona a ação que maximiza a função $Q(s_t, a; \theta)$, ou seja, explota o conhecimento obtido até o momento (Linha 9).

Na Linha 10 chamamos a função *ExecutaPoda* que executa um passo de poda em $M^{\text{episode}}$, aplicando um fator de poda $\alpha$ definido pela ação $a_t$ em uma camada específica da rede de acordo com o estado atual $s_t$. Esta função retorna um modelo com uma quantidade de filtros menor, a recompensa $r_t$ e o próximo estado $s_{t+1}$. A transição formada por $(s_t, a_t, r_t, s_{t+1})$ é armazenada na fila de memória $D$ (Linha 11). É interessante notar que até este momento estamos apenas acumulando experiência, sem ainda de fato treinarmos as redes $Q$ e $\hat{Q}$. Essa etapa de "coleta de dados" pode ocorrer por vários passos e serve para popular a fila de memória, garantindo que existam ao menos $B$ transições em memória[1] para prosseguirmos para a próxima etapa do algoritmo.

As Linhas 12 – 15 descrevem a etapa de aprendizado supervisionado da DQN. Na Linha 12 um conjunto de $B$ transições é amostrado da memória e para cada transição é definido um rótulo meta ($y_j$) seguindo os critérios descritos na Linha 13. Caso o episódio se encerre num passo seguinte ao da transição $j$ amostrada, o rótulo é simplesmente a recompensa obtida nessa amostra. Já nos demais casos o rótulo é a recompensa obtida na transição amostrada somada ao valor descontado da função estado-ação aplicada em $s_{j+1}$ parametrizada pelos pesos da rede neural meta. Na Linha 14, é calculada a média do erro quadrático médio de cada amostra, ou seja, o desvio de cada rótulo $y_j$ ao valor da função estado-ação aplicada em $s_j$ tomando a ação $a_j$ parametrizada pelos pesos da rede neural de treinamento. A Linha 15 realiza um passo de otimização em direção ao mínimo local e a Linha 16 copia os pesos da rede de treinamento para a rede meta a cada $U$ passos. Após o episódio final o algoritmo retorna a rede meta ($\hat{Q}$), a qual será utilizada para extrair uma política.

[1] Na prática, o algoritmo DQN possui um hiperparâmetro que controla a quantidade de passos iniciais que serão realizados antes do treinamento da rede neural a fim de obter experiência e popular a memória.

# Capítulo 5

# Compressão de Modelos em Transferência de Aprendizado

Neste capítulo, apresentamos o arcabouço PRUNE2TRANFER, que consiste: (i) nas etapas de compressão de um modelo treinado para uma tarefa; (ii) na escolha da melhor poda desse modelo considerando a relação compressão × acurácia; (iii) da transferência e adaptação do modelo podado para uma nova tarefa e (iv) da avaliação e comparação do desempenhos desses modelos. O objetivo principal da proposta do arcabouço PRUNE2TRANFER é permitir uma análise comparativa consistente dos diferentes critérios de poda estudados neste trabalho (Capítulos 3 e 4) .

## 5.1 Arcabouço PRUNE2TRANFER

Na transferência de aprendizado profundo tradicional, um modelo treinado de forma supervisionada em uma tarefa origem é utilizado como inicialização de um modelo com arquitetura similar que é adaptado para outra tarefa (tarefa meta). Após esta adaptação o modelo possui mais parâmetros que o necessário para resolver aquela tarefa. Uma forma de reduzir a complexidade deste modelo é utilizando alguma técnica de poda de parâmetros que zera os pesos que contribuem pouco no aprendizado da tarefa meta. Esta sequência de passos alcança o objetivo de comprimir o modelo com base na tarefa origem, porém os modelos pré-treinados são utilizados pelas mais diversas tarefas. Logo, seria mais eficiente realizar a poda antes da etapa de transferência de aprendizado, reduzindo assim a complexidade e evitando que cada tarefa meta tenha que realizar uma poda específica. E esse é o objetivo do arcabouço PRUNE2TRANFER.

A Figura 5.1 apresenta o diagrama proposto para podar e avaliar um modelo em sua tarefa original, a transferência de extratores de características para outras tarefas meta e a avaliação de seus respectivos desempenhos.

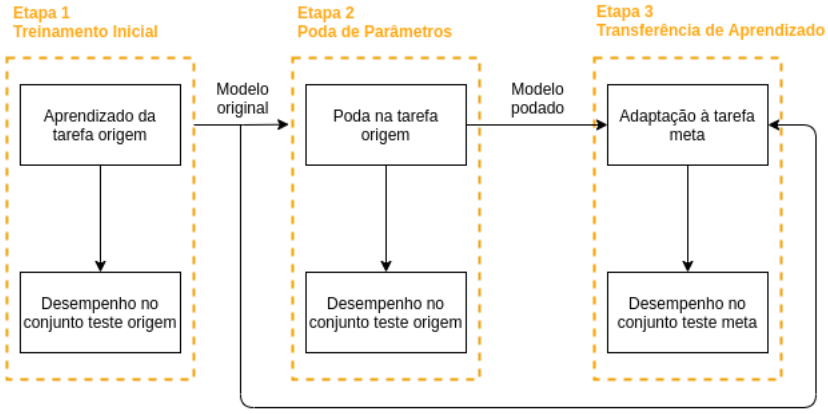



**Figura 5.1:** *Diagrama proposto para o arcabouço* PRUNE2TRANFER.

Nosso arcabouço ordena as etapas descritas acima a fim de obter uma maior compressão de modelos profundos a um baixo custo computacional. Este baixo custo se dá ao fato da poda de parâmetros ser realizada apenas no modelo original, garantindo que as tarefas meta que adaptarem este modelo já partirão de um modelo mais reduzido se comparado à transferência sem poda.

A relevância de cada etapa do diagrama (Figura 5.1) para o PRUNE2TRANFER será detalhada nas seções a seguir.

## 5.2 Etapa 1: Treinamento inicial

A primeira etapa consiste em treinar uma rede neural para a tarefa original e com o modelo treinado medir a acurácia no conjunto de teste, permitindo estimar o desempenho fora da amostra (*out-of-sample*) do modelo. Essa etapa tem dois propósitos: (i) definir a magnitude de cada conexão da rede neural, e (ii) definir um valor de referência de acurácia na tarefa original.

De acordo com HAN e DALLY (2017), é possível avaliar a relevância das conexões de um modelo já treinado a partir do valor absoluto da magnitude de seus parâmetros. Nesse caso, um valor pequeno indica que o parâmetro não tem grande impacto na capacidade preditiva do modelo, sendo um forte candidato à poda na próxima etapa.

O valor da acurácia desse primeiro modelo serve de referência na avaliação do impacto da poda na tarefa original, permitindo uma escolha sistemática de um modelo com melhor relação compressão × acurácia.

## 5.3 Etapa 2: Poda de parâmetros

A segunda etapa consiste em realizar a poda do modelo treinado em 5.2 seguindo um dos critérios apresentados no Capítulo 4, com exceção do método PuRL que foi apresentado no Capítulo 3. A seguir listamos os tipos de poda experimentados e uma breve descrição de cada um.

**Técnicas avaliadas**

- Não-Estruturada — poda iterativa com critério baseado em percentil das magnitudes;
- Estruturada — poda realizada de forma direta com critério baseado na norma $L_1$;
- PuRL — poda não-estruturada guiada por uma política aprendida por reforço;
- StructPuRL — poda estruturada guiada por uma política aprendida por reforço.

Cada modelo após ser podado seja por técnica iterativa ou direta foi avaliado no conjunto reservado de teste, a fim de verificar se houve alguma perturbação na acurácia obtida com o modelo original. Este modelo podado será utilizado nas tarefas que utilizam transferência de aprendizado.

## 5.4 Etapa 3: Transferência de aprendizado

Como discutido no Capítulo 2, a técnica de transferência de aprendizado (*transfer learning*) permite o compartilhamento de estruturas neurais aprendidas em um domínio para serem utilizadas em outro domínio, a fim de acelerar a convergência do otimizador ou viabilizar o aprendizado profundo em tarefas com poucos dados. A ideia por trás do sucesso desta técnica em problemas de classificação de imagens se deve ao aprendizado de filtros convolucionais que detectam elementos básicos de uma imagem (por exemplo contornos e texturas) e que também são fundamentais para outras tarefas de classificação.

A última etapa do PRUNE2TRANFER (Figura 5.1) consiste em adaptar tanto o modelo original quanto o modelo podado para tarefas meta diferentes do conjunto de dados usado nas etapas anteriores. O objetivo aqui é compreender se *as representações aprendidas na tarefa original sofreram algum dano em sua capacidade de generalização por conta da compressão da rede*, ou seja, ficaram super-ajustadas à tarefa original após a poda de uma maneira irreparável.

O tópico de aprendizado de representações (*representation learning*) consiste no aprendizado de estruturas, a partir dos dados, que facilitem a extração de informações relevantes no treinamento de classificadores e outros modelos preditivos (Bengio *et al.*, 2013). Até fim de 2019 não existia um protocolo unificado de avaliação (*benchmark*) para a adaptação de modelos a tarefas de classificação de imagens, até que pesquisadores do Google propuseram o VTAB (*Visual Task Adaptation Benchmark*) (Zhai *et al.*, 2019). O objetivo do VTAB é mensurar a qualidade das representações aprendidas por um modelo através da adaptação deste modelo a novas tarefas usando poucos dados, limitando em 800 imagens para treino e 200 para validação. A métrica de avaliação adotada pelos autores foi a média simples entre as acurácias obtidas nas tarefas meta. A fim de compreender os limites de adaptação, o VTAB também propõe que a avaliação seja feita em 19 tarefas distintas pertencentes às categorias:

- Natural - contém imagens naturais capturadas por câmeras convencionais (e.g. veículos, tipos de flores, peixes, etc);
- Especializado - similar à categoria Natural, porém com imagens capturadas por equipamentos especializados como satélites ou equipamentos para exames de imagem

(e.g. imagens aéreas, tomografias e etc);

- Estruturado - contém imagens que requerem a compreensão da estrutura da cena, como por exemplo a predição de profundidade de uma imagem.

Inspirada no VTAB, a nossa avaliação da capacidade de adaptação dos modelos podados foi feita em relação ao desempenho obtido no mesmo conjunto de dados pelo modelo sem poda. Como o objetivo é mensurar se há uma diferença significativa entre os desempenhos destes modelos, foi realizada a análise da média das diferenças entre as acurácias em todas as tarefas meta, sempre tomando par a par um modelo podado contra o seu respectivo modelo origem.

$$\begin{aligned} \text{SCORE} &= \mathbb{E}[{acc^{target}}_{original} - {acc^{target}}_{pruned}] \\ &= \frac{1}{n}\sum_{i=1}^{n} {acc^{i}}_{original} - {acc^{i}}_{pruned} \end{aligned} \tag{5.1}$$

Na Equação 5.1, $n$ é a quantidade de tarefas meta, ${acc^{target}}_{original}$ é a acurácia do modelo sem poda no subconjunto de teste da tarefa $i$ e ${acc^{target}}_{pruned}$ é a acurácia de um modelo podado no subconjunto de teste da mesma tarefa $i$. O modelo podado ideal deveria ter um SCORE $\leq 0$, pois indicaria que não houve uma degradação na capacidade de representação visto que a acurácia é em média igual ou melhor em comparação à adaptação do modelo original. Esta avaliação é relevante para avaliarmos a efetividade do PRUNE2TRANSFER e compararmos como os diferentes métodos de poda influenciam nos extratores de características.

Neste capítulo apresentamos o arcabouço PRUNE2TRANSFER e detalhamos cada etapa que o compõe. No Capítulo 6 colocaremos este arcabouço à prova variando as técnicas de poda utilizadas na Etapa 2 (Figura 5.1).

# Capítulo 6

# Análise Empírica

Neste capítulo, usaremos o arcabouço PRUNE2TRANSFER introduzido no capítulo anterior, para realizar uma análise comparativa consistente dos diferentes critérios de poda estudados e estendidos neste trabalho. Mais especificamente, analisaremos os seguintes critérios de poda de redes convolucionais em transferência de modelos: poda não-estruturada iterativa com teste-t pareado bayesiano, poda estruturada baseada na norma-$L_1$ com análise de sensibilidade, o método PuRL e nossa modificação, o método StructPuRL.

Todos os experimentos foram realizados utilizando a biblioteca *TensorFlow 2* (Abadi *et al.*, 2015) para linguagem Python. A execução dos códigos foi feita no ambiente de computação na nuvem do *Google Colab Pro*, que conta com uma GPU Tesla V100-16GB, 185GB de espaço temporário em disco e 25GB de memória RAM.

## 6.1 Rede Convolucional

Visto que o foco deste trabalho é a transferência de representações aprendidas de um contexto de imagens para outro, escolhemos para os experimentos uma CNN profunda que tem sido utilizada em diversos *benchmarks* de classificação de imagens: a *VGG-19* proposta por Simonyan e Zisserman (2015).

Vale a pena destacar que, como prova de conceito para as extensões propostas, realizamos a mesma análise empírica deste capítulo com uma rede CNN com arquitetura pouco profunda extraída do trabalho de Katinskaia (2019), que chamamos de *CNN-simples*. Os resultados dessa análise estão disponíveis no Apêndice A.

### A rede VGG-19

A rede convolucional VGG-19[1] é usada por muitos trabalhos na área de classificação de imagens em aprendizado supervisionado (Gatys *et al.*, 2015; Bizopoulos *et al.*, 2020; Jha *et al.*, 2020; Prajapati e Stamp, 2021). Num total de 19 camadas, essa CNN possui 16 camadas convolucionais, 2 camadas densas e 1 camada de saída. Uma das características

[1]Visual Geometry Group (VGG), é um grupo de pesquisa em visão computacional da Universidade de Oxford.

desta CNN é ter sido pré-treinada para processar um número muito grande de imagens e portanto ser muito eficiente na transferência de modelos envolvidos na classificação de imagens. Essa é a razão desta rede estar disponível nas principais plataformas de aprendizado profundo como o TensorFlow, PyTorch e Caffe.

A Figura 6.1 ilustra a arquitetura modificada da VGG-19, mostrando as dimensões de entrada adotada em todos os experimentos, cinco blocos de convolução intercalados por camadas de *max pooling* (laranja escuro), duas camadas densas totalmente conectadas (*fully-connected* — fc) e a saída da rede de 30 unidades de forma a atender nossa tarefa de classificação (vide Seção 6.2).

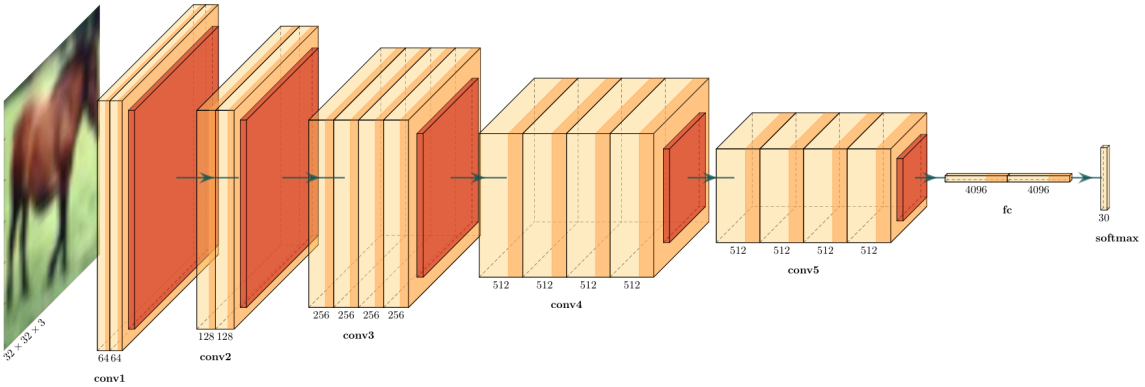



**Figura 6.1:** *VGG-19 adaptada para uma tarefa de classificação de 30 classes com imagens de entrada de dimensões* 32 × 32 × 3.

Na Figura 6.1, os cinco grupos **conv** representam blocos de convolução, cada um com duas ou quatro camadas com a quantidade de filtros variando entre 64 e 512 filtros. Em todas as camadas convolucionais, foi adotado um campo receptivo de 3 × 3. Os elementos em laranja claro representam as camadas de ativação, que na VGG-19 utiliza a função *ReLU*. As camadas de *max pooling* utilizam um tamanho de *pool* de 2 × 2 com *stride* de 2. O número total de parâmetros da VGG-19 é 39 029 854.

## 6.2 Conjuntos de Dados

Os conjuntos de dados (*datasets*) utilizados nesta pesquisa foram divididos em dois grupos, um grupo é utilizado apenas nas tarefas originais e o outro grupo apenas nas tarefas meta, conforme descrito no Capítulo 5. O mesmo cuidado foi tomado nas classes dos dados, para que não houvesse intersecção de imagens visitadas no treinamento da tarefa original durante o aprendizado da tarefa meta. Caso este tipo de "vazamento de dados" (*data leakage*) ocorresse, as tarefas meta seriam beneficiadas por utilizarem um modelo que já aprendeu com base naquelas imagens, levando a um modelo super-ajustado (*overfitting*).

### 6.2.1 Dados da Tarefa Original

Uma tarefa original é aquela em que o modelo é inicialmente treinado. e para a qual a poda do modelo será realizada. Todas as imagens do conjunto de dados da tarefa original são coloridas e possuem as dimensões 32 × 32. Para isso, usaremos uma versão

do consagrado conjunto de imagens naturais do ImageNet (Russakovsky *et al.*, 2015) amostrado para dimensões menores. De todas as 1 000 classes da tarefa original, foram amostradas aleatoriamente 30 classes[2] deste conjunto de imagens para criar o conjunto Downsampled ImageNet-30, que será chamado simplesmente de *ImageNet-30* no decorrer do trabalho. Assim, a tarefa original da VGG-19 é a classificação de uma imagem em uma das 30 categorias do ImageNet-30.

### 6.2.2 Dados da Tarefas Meta

Uma tarefa meta é aquela em que a qualidade das representações aprendidas pelo modelo, tanto o original quanto o podado, é avaliada. Portanto, o objetivo destas tarefas é verificar se a poda de um modelo afeta a generalização das estruturas aprendidas. Seguindo a proposta do VTAB (Zhai *et al.*, 2019), será utilizada apenas uma amostra estratificada dos conjuntos de dados para adaptação do modelo às tarefas meta, limitado em 800 imagens para treino e 200 para validação. Esta escolha serve para dois propósitos: (i) diminuir o tempo de treinamento das redes e (ii) simular um cenário com poucos dados da tarefa meta, que é um caso típico de uso da transferência de aprendizado.

Em todos os conjuntos de dados a seguir as imagens são coloridas e redimensionadas (exceto o CIFAR-100) para as dimensões $32 \times 32$.

- Superclasses CIFAR-100 (Krizhevsky, Hinton *et al.*, 2009) — consiste em imagens naturais categorizadas originalmente em 100 classes e com 20 superclasses[3] (cada superclasse engloba 5 classes do nível mais granular). **Cada uma das superclasses foi considerada como uma tarefa meta** neste trabalho;
- KITTI (Geiger *et al.*, 2013) — consiste em imagens capturadas a partir de um veículo em movimento em zonas urbanas e rurais, a fim de ser utilizado em pesquisas com carros autônomos e robôs móveis. O conjunto de dados original foi adaptado para uma tarefa de classificação em que o objetivo é predizer um dos 4 níveis de profundidade até o veículo a frente (Zhai *et al.*, 2019). Nesta tarefa é necessário que o modelo aprenda elementos da estrutura da imagem (ex. profundidade), portanto estas imagens entram na categoria de estruturais;
- EuroSAT (Helber *et al.*, 2019) — consiste em imagens especializadas capturadas pelo satélite Sentinel-2 categorizadas em 10 classes[4].

Sendo assim, faremos a transferência de aprendizado para 22 tarefas distintas, nas quais avaliaremos o aprendizado de representações dos modelos originais e podados.

[2] As 30 classes de imagens incluem: labirinto, padaria, carneiro, esquilo, maraca, saleiro, limusine, órgão (instrumento musical), cafeteira, cadeado, vassoura, pavê, joaninha, aranha do celeiro, casa de barco, bandeja, tucano, abajur, lancha, crisopídeo, frasco de comprimido, ábaco, pastor húngaro, caixote, telefone público, preguiça-de-três-dedos, apito, dirigível, lagarto crocodilo e buldogue francês.

[3] As 20 superclasses de imagens do CIFAR-100 incluem: mamíferos aquáticos, peixes, flores, recipientes para alimentos, vegetais e frutas, dispositivos elétricos de casa, móveis domésticos, insetos, carnívoros de grande porte, grandes construções humanas, grandes paisagens da natureza, grandes onívoros herbívoros, mamíferos de médio porte, invertebrados (não insetos), pessoas, répteis, mamíferos de pequeno porte, árvores, veículos 1 e veículos 2.

[4] As 10 classes de imagens do EuroSAT incluem: prédios industriais, prédios residenciais, culturas temporárias, culturas permanentes, rios, mares e lagos, vegetações herbáceas, rodovias, pastos e florestas.

## 6.3 PRUNE2TRANSFER — Aprendizado inicial

A primeira etapa do arcabouço PRUNE2TRANSFER consiste em treinar a VGG-19 até a convergência em sua tarefa original, chamada de aprendizado inicial. Este modelo após treinado servirá tanto como base para poda como modelo de referência para as tarefas meta. Em ambos os casos a acurácia obtida serve de referencial na compreensão do impacto de diferentes métodos e níveis de poda aplicados ao modelo.

A rede VGG-19 foi inicializada com os pesos do ImageNet e foi treinada para o ImageNet-30 por 20 épocas em lotes de 64 usando o otimizador *Stochastic Gradient Descent* (SGD) (QIAN, 1999) com taxa de aprendizado fixa em $5 \times 10^{-4}$ e momentum de 0.9. **A acurácia no conjunto de teste do ImageNet-30, que consiste em** 50 **imagens de cada classe, foi de** 55.53%.

Seguindo o diagrama 5.1 do arcabouço PRUNE2TRANSFER, após a etapa de treinamento inicial da rede, temos as etapas de poda e a transferência de aprendizado, respectivamente. As seções a seguir apresentam os resultados destas duas etapas para cada critério de poda avaliado. Note que, nos experimentos, o Aprendizado Inicial é realizado uma única vez para todos os critérios de poda analisados neste trabalho.

## 6.4 Poda Não-Estruturada Iterativa com Teste-t Pareado Bayesiano

### 6.4.1 PRUNE2TRANSFER — Poda do modelo

A segunda etapa do arcabouço PRUNE2TRANSFER consiste da poda do modelo previamente treinado da VGG-19 na primeira etapa (Seção 6.3). Neste experimento realizamos uma poda não-estruturada Iterativa com Teste-t Pareado Bayesiano (5) realizada por 15 iterações com os mesmos hiperparâmetros do treinamento inicial. A Figura 6.2 ilustra o gráfico da acurácia para cada percentual de poda.

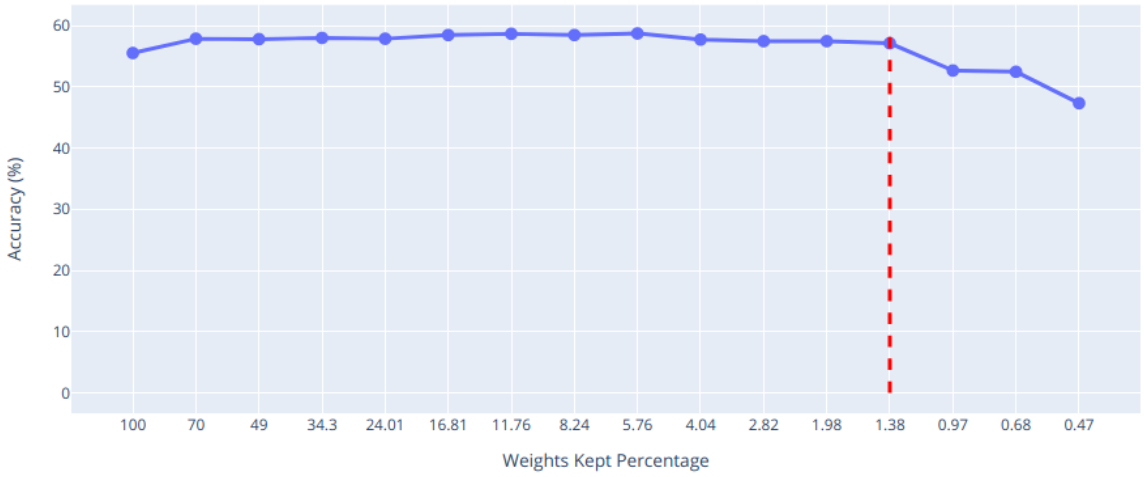



**Figura 6.2:** *Acurácia por iteração de poda não-estruturada da rede VGG-19.*

Na Figura 6.2, a linha tracejada vermelha indica a melhor poda considerando o critério

do "cotovelo" (Seção 4.1), na qual temos um modelo com 1.38% dos parâmetros do modelo original e com uma acurácia de 57.13%. Porém, como discutido na Seção 4.1.1, utilizando um teste probabilístico podemos averiguar a equivalência prática do modelo podado com o modelo sem poda (i.e. o modelo original). E com base neste teste tomar uma decisão mais informada sobre o melhor percentual de poda.

A Tabela 6.1 traz os valores da probabilidade: (i) do modelo original ($P_{\text{original}}$) ser praticamente melhor que o modelo podado, (ii) do modelo podado e sem poda serem praticamente equivalentes ($P_{\text{ROPE}}$) e (iii) do modelo podado ($P_{\text{podado}}$) ser praticamente melhor que o modelo original. Na tabela, os valores de $P_{\text{rope}} + P_{\text{podado}} > 0.5$, que estão destacados em verde, indicam modelos que são praticamente melhores ou equivalentes ao modelo sem poda, logo são modelos candidatos à escolha de modelo ótimo considerando uma região de equivalência de ±1%.

**Tabela 6.1:** *Teste-t pareado bayesiano para a poda não-estruturada iterativa da VGG-19 com região de equivalência prática a* 1% *e* 5*-folds. A linha em negrito salienta a porcentagem de poda ótima de acordo com o teste estatístico e a linha em vermelho o resultado da regra de cotovelo.*

| Pesos preservados (%) | $P_{\text{original}}$ | $P_{\text{ROPE}}$ | $P_{\text{podado}}$ | $P_{\text{ROPE}} + P_{\text{podado}}$ |
|---|---|---|---|---|
| 70.00 | 0.0116 | 0.9230 | 0.0654 | 0.9884 |
| 49.00 | 0.0227 | 0.9741 | 0.0031 | 0.9773 |
| 34.30 | 0.0013 | 0.9794 | 0.0193 | 0.9987 |
| 24.01 | 0.0002 | 0.9963 | 0.0035 | 0.9998 |
| 16.81 | 0.0002 | 0.2335 | 0.7664 | 0.9998 |
| 11.76 | 0.0002 | 0.4150 | 0.5848 | 0.9998 |
| 8.24 | 0.0020 | 0.8763 | 0.1217 | 0.9980 |
| 5.76 | 0.0076 | 0.7516 | 0.2407 | 0.9924 |
| **4.04** | **0.0077** | **0.9138** | **0.0785** | **0.9923** |
| 2.82 | 0.5993 | 0.2735 | 0.1272 | 0.4007 |
| 1.98 | 0.7986 | 0.1065 | 0.0948 | 0.2014 |
| 1.38 | 0.9110 | 0.0743 | 0.0147 | 0.0890 |
| 0.97 | 0.9668 | 0.0126 | 0.0205 | 0.0332 |
| 0.68 | 0.9851 | 0.0065 | 0.0084 | 0.0149 |
| 0.47 | 0.9359 | 0.0091 | 0.0549 | 0.0641 |

O resultado na Tabela 6.1 difere do ponto ótimo do gráfico indicado pela regra do cotovelo, sendo mais conservador na poda, isto é, mantém 4.04% do total de parâmetros, ao invés de 1.38%. Lembrando que o ponto ótimo é determinado pelo maior percentual de poda que mantém a desigualdade $P_{\text{ROPE}} + P_{\text{podado}} > 0.5$ (Seção 4.1.2) verdadeira. Isso pois entendemos que um modelo após ser podado deve pelo menos manter o desempenho da sua contraparte não podada.

A divergência entre o ponto ótimo de acordo com o teste estatístico e o ótimo analisando somente a acurácia é devido ao fato que no teste estamos considerando 5 quebras (*folds*) no conjunto de treino e as respectivas acurácias no conjunto de teste, o que pode prejudicar um modelo caso haja uma queda de desempenho em um dos *folds*.

Visto que queremos podar ao máximo o modelo, minimizando o impacto na acurácia, o critério baseado no teste estatístico indica que **a poda ótima é de** 95.96% **dos parâmetros, o que é equivalente a manter** 4.04% **dos parâmetros diferentes de zero.**

### 6.4.2 PRUNE2TRANSFER — Transferência de Aprendizado

Nesta etapa, todos os modelos podados na etapa anterior bem como modelo original são utilizados para realizar a transferência de aprendizado do modelo considerando cada uma das 22 tarefas meta mencionadas na Seção 6.2.

Dado que a VGG-19 é uma rede profunda com muitos parâmetros e o conjunto de dados das tarefas meta não é superior a 1 000 imagens, é necessário especificar a forma de realizar a transferência de aprendizado. Como as imagens naturais do CIFAR-100 pertencem a um domínio similar ao da tarefa original, na transferência de aprendizado para as 20 tarefas meta desse domínio, parte dos pesos da rede foi "congelada", ou seja, não foram ajustados na etapa de *backpropagation*.

Seguindo o estudo de capacidade de transferência de camadas de Yosinski *et al.* (2014), optamos por congelar os três primeiros blocos convolucionais (conv1, conv2 e conv3) e permitir o ajuste dos demais pesos do modelo (Figura 6.1). Isso com base nos relatos empíricos de que as primeiras camadas convolucionais de uma CNN capturam representações mais gerais das imagens, sendo também presentes em outras tarefas meta. Já para as imagens estruturadas e especializadas (Seção 6.2) das bases EuroSAT e KITTI, foi realizado o *fine-tuning* de todas as camadas, a fim de melhor adaptarmos os filtros para domínios diferentes.

Quanto aos hiperparâmetros de cada treinamento na transferência de aprendizado, as tarefas do CIFAR-100 utilizaram o otimizador SGD com taxa de aprendizado fixa de $5 \times 10^{-3}$ (momentum de 0.9) por 40 épocas em lotes de 16 imagens. Já as tarefas EuroSAT e KITTI, utilizaram o otimizador SGD com taxa de aprendizado fixa em $5 \times 10^{-4}$ (momentum de 0.9 e atualização de Nesterov) por 40 épocas em lotes de 16 imagens. As Figuras 6.3 e 6.4 ilustram os gráficos da acurácia média de 5 repetições do resultado de transferência para cada uma das 22 tarefas meta usando todos os modelos podados e o modelo original (Figura 6.2).

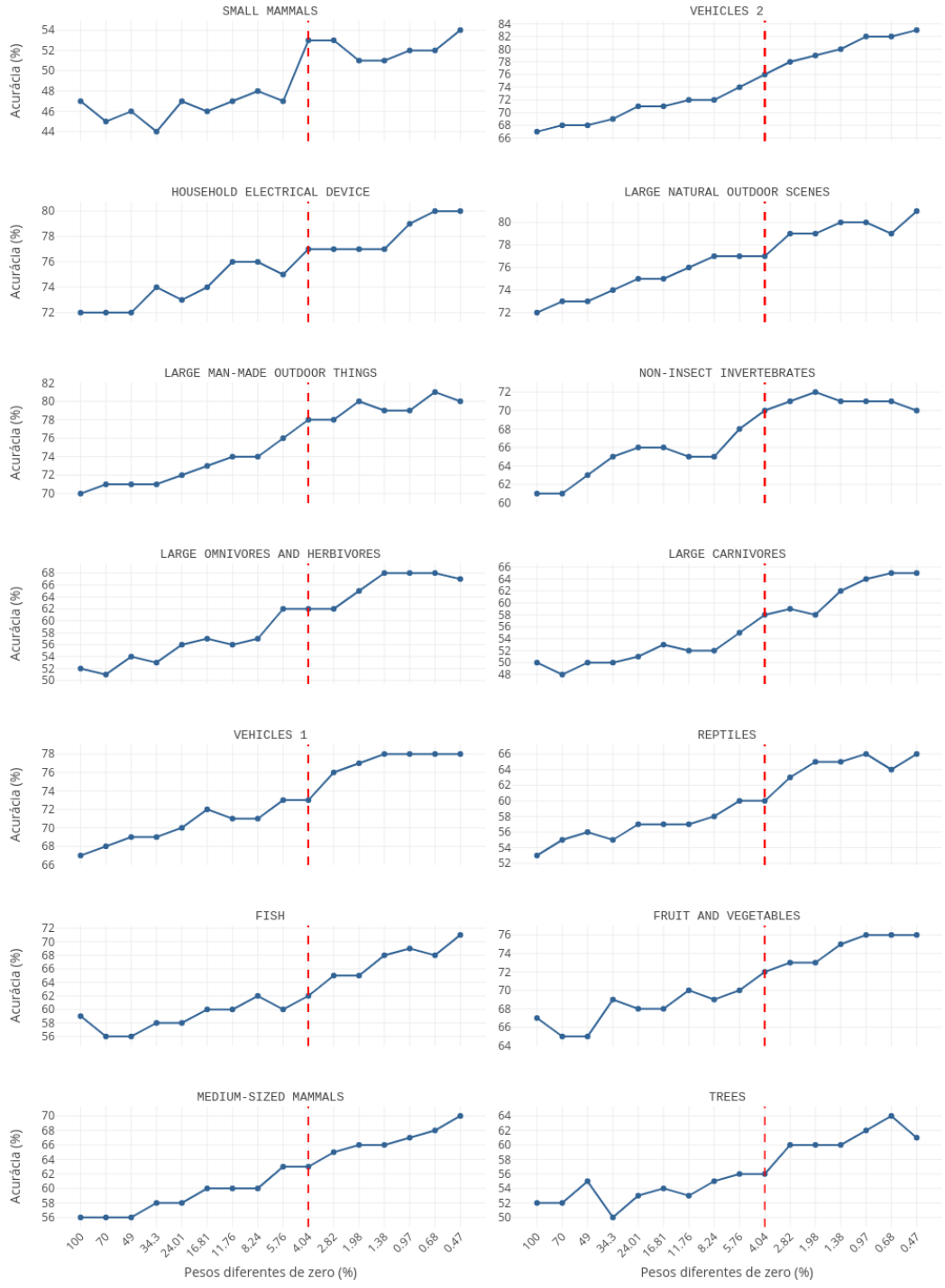



**Figura 6.3:** *Acurácia média de 5 iterações de transferência de aprendizado para* 14 *tarefas meta usando diferentes podas e o modelo original da VGG-19.*

Nas Figuras 6.3 e 6.4, a linha tracejada vermelha indica o modelo com o melhor percentual de poda da etapa anterior, em que foram mantidos 4.04% dos parâmetros originais, selecionado com o critério do teste-t pareado bayesiano. As figuras também mostram o resultado da transferência de outros modelos podados para fins de análise e discussão. É

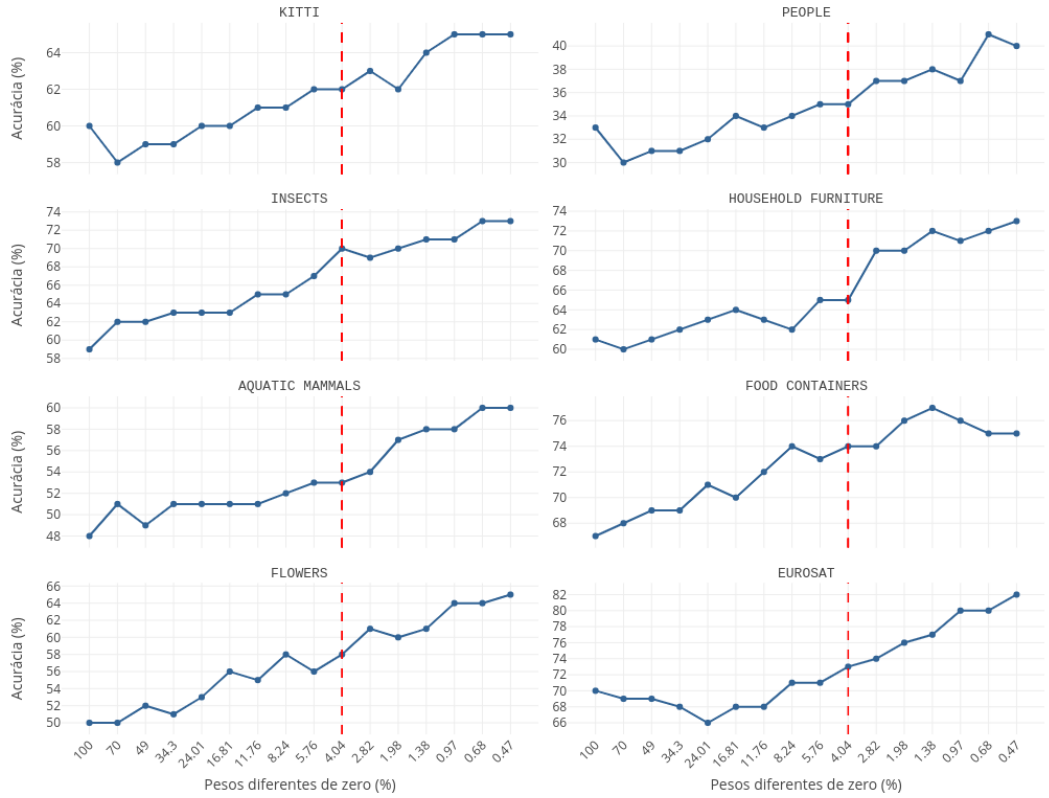



**Figura 6.4:** *Acurácia média de 5 iterações de transferência de aprendizado para* 8 *tarefas meta usando diferentes podas e o modelo original da VGG-19.*

interessante notar que em todas as tarefas meta, apesar de modelos podados subótimos iniciais resultarem em uma acurácia maior (pontos à direita da linha vermelha tracejada), há uma melhora significativa da acurácia usando o modelo podado ótimo ao invés do modelo original. Note ainda que, na medida em que modelos mais esparsos são transferidos, a acurácia também aumenta. Isso indica que modelos mais parametrizados não contribuem com o aprendizado das tarefas meta neste cenário de adaptação com poucos dados.

A Tabela 6.2 mostra as acurácias resultantes após a transferência do modelo sem poda ($\text{Acc}_{\text{original}}$), do modelo ótimo escolhido na etapa de poda ($\text{Acc}_{\text{podado}}$) e a diferença entre elas. Analisando os dados da Tabela 6.2, temos que o modelo podado apresenta um desempenho superior ao modelo original em todas as 22 tarefas, acarretando em um score de −6.21 pontos percentuais. E como este é um valor negativo, **podemos concluir neste experimento que um modelo que mantém** 4.04% **dos parâmetros originais pode ser adaptado à novas tarefas sem perda de generalidade**.

**Tabela 6.2:** *Acurácias usando a poda não-estruturada iterativa com teste-t pareado bayesiano nas 22 tarefas meta: acurácia usando o modelo original ($Acc_{original}$); acurácia usando o modelo podado ($Acc_{podada}$) e a diferença entre elas.*

| Tarefa Meta | $Acc_{original}$(%) | $Acc_{podado}$(%) | $Acc_{original}$-$Acc_{podado}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 47.92 | 52.76 | -4.84 |
| Fish | 58.52 | 62.04 | -3.52 |
| Flowers | 49.56 | 58.40 | -8.84 |
| Food Containers | 67.08 | 73.52 | -6.44 |
| Fruit and Vegetables | 66.60 | 71.80 | -5.20 |
| Household Electrical Device | 71.96 | 76.92 | -4.96 |
| Household Furniture | 60.88 | 65.44 | -4.56 |
| Insects | 59.24 | 69.60 | -10.36 |
| Large Carnivores | 49.92 | 57.72 | -7.80 |
| Large Man-made Outdoor Things | 70.20 | 78.24 | -8.04 |
| Large Natural Outdoor Scenes | 71.68 | 76.92 | -5.24 |
| Large Omnivores and Herbivores | 51.88 | 62.28 | -10.40 |
| Medium-sized Mammals | 56.36 | 62.88 | -6.52 |
| Non-insect Invertebrates | 60.72 | 69.84 | -9.12 |
| People | 32.64 | 34.84 | -2.20 |
| Reptiles | 52.68 | 60.48 | -7.80 |
| Small Mammals | 46.52 | 52.92 | -6.40 |
| Trees | 51.72 | 55.96 | -4.24 |
| Vehicles 1 | 66.92 | 73.40 | -6.48 |
| Vehicles 2 | 66.96 | 75.68 | -8.72 |
| KITTI | 59.74 | 62.29 | -2.55 |
| EuroSAT | 70.48 | 72.79 | -2.31 |
| Média | — | — | -6.21 |

## 6.5 Poda Estruturada baseada na norma-$L_1$ com análise de sensibilidade

### 6.5.1 PRUNE2TRANSFER — Poda do Modelo

A poda estruturada baseada na norma-$L_1$ com análise de sensibilidade da VGG-19 é um método direto baseado na análise de sensibilidade das camadas convolucionais. Desta forma, nos experimentos dessa seção geramos somente um modelo podado para realizar a transferência na próxima seção. Nesta análise, o modelo inicialmente treinado até convergência teve diferentes percentuais de filtros podados por camada e a cada poda a acurácia no conjunto de teste foi calculada (Figura 6.5).

A Figura 6.5 ilustra o gráfico da acurácia do modelo após a poda individual de cada camada considerando diferentes quantidades de filtros preservados. Cada curva está com o nome **conv**$x$_$y$, em que $x$ é a camada convolucional e $y$ é a quantidade de filtros originalmente presentes na CNN (Figura 6.1). Analisando as curvas, temos que as últimas

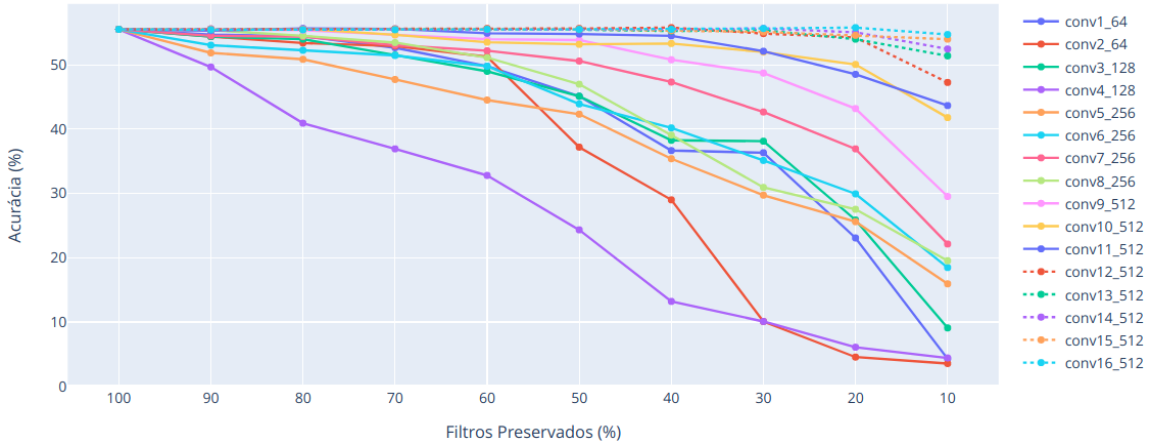



**Figura 6.5:** *Acurácia no conjunto de teste após poda estruturada em cada camada convolucional da VGG-19.*

camadas convolucionais (curvas tracejadas) são mais resilientes à poda, permitindo uma maior compressão. As camadas conv2_64 e conv4_128 são mais sensíveis à remoção de filtros, apresentando um impacto significativo na acurácia quando podadas em 50% e 20%, respectivamente.

Analisando a sensibilidade de cada camada e com uma tolerância de até −2 p.p., o percentual de filtros preservados em cada uma das 16 camadas foi de: 80, 90, 80, 100, 100, 100, 80, 80, 50, 60, 40, 20, 20, 20, 10 e 10. As camadas com 100% dos pesos preservados indicam que não serão podadas, visto que na análise individual de sensibilidade apresentaram um descolamento maior que as outras curvas nas primeiras etapas de poda. Seguindo o método de poda direto, todas as camadas foram podadas e ao final retreinadas uma única vez. Neste treinamento utilizamos o otimizador SGD com taxa de aprendizado fixa em 0.001 (momentum 0.9) por 70 épocas em lotes de 64 imagens com *early stopping*. **Atingindo uma acurácia de** 55.6% **e uma redução de** 45.6% **dos parâmetros originais**. A Tabela 6.3 traz uma comparação do modelo original e do modelo resultante da poda estruturada.

**Tabela 6.3:** *Comparação entre o modelo original da VGG-19 e o modelo podado com redução de* 45.6% *dos parâmetros.*

| Modelo | Total de Parâmetros | Acurácia (%) | Tamanho em Disco (MB) |
|---|---|---|---|
| Original | 39 029 854 | 55.53% | 148.97 |
| Podado | 21 232 080 | 55.60% | 81.08 |

Analisando a Tabela 6.3, temos que o modelo podado mantém a acurácia do modelo original e ocupa menos espaço em disco. Sendo assim, este modelo que manteve 54.4% dos parâmetros originais será adaptado nas tarefas de transferência de aprendizado a seguir e comparado com o modelo sem poda, seguindo o critério de score do arcabouço PRUNE2TRANSFER.

### 6.5.2 PRUNE2TRANSFER — Transferência de Aprendizado

Os hiperparâmetros da transferência para cada tarefa meta foram os mesmos utilizados na Seção 6.4. Dado que com o critério de poda adotado na seção anterior selecionamos um único modelo podado para ser comparado com o original, apresentamos os resultados na Tabela 6.4, para as 22 tarefas meta. Analisando esses resultados, temos que o modelo podado com critério estruturado apresenta um desempenho superior na maioria das tarefas meta com relação ao uso direto do modelo original, com uma diferença média de −1.55 pontos percentuais. Dado que este valor é negativo, **podemos concluir que um modelo com poda estruturada que mantém** 54.4% **dos parâmetros originais pode ser adaptado à novas tarefas sem perda de generalidade**.

**Tabela 6.4:** *Acurácias usando a Poda Estruturada baseada na norma-$L_1$ nas 22 tarefas meta: acurácia usando o modelo original ($Acc_{original}$); acurácia usando o modelo podado ($Acc_{podada}$) e a diferença entre elas.*

| Tarefa Meta | $Acc_{original}$(%) | $Acc_{podado}$(%) | $Acc_{original}$-$Acc_{podado}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 47.92 | 49.76 | -1.84 |
| Fish | 58.52 | 57.32 | 1.20 |
| Flowers | 49.56 | 49.88 | -0.32 |
| Food Containers | 67.08 | 70.88 | -3.80 |
| Fruit and Vegetables | 66.60 | 67.04 | -0.44 |
| Household Electrical Device | 71.96 | 72.00 | -0.04 |
| Household Furniture | 60.88 | 62.40 | -1.52 |
| Insects | 59.24 | 69.68 | -3.44 |
| Large Carnivores | 49.92 | 48.12 | 1.80 |
| Large Man-made Outdoor Things | 70.20 | 73.68 | -3.48 |
| Large Natural Outdoor Scenes | 71.68 | 73.60 | -1.92 |
| Large Omnivores and Herbivores | 51.88 | 55.04 | -3.16 |
| Medium-sized Mammals | 56.36 | 57.20 | -0.84 |
| Non-insect Invertebrates | 60.72 | 65.88 | -5.16 |
| People | 32.64 | 32.92 | -0.28 |
| Reptiles | 52.68 | 56.24 | -3.56 |
| Small Mammals | 46.52 | 45.64 | 0.88 |
| Trees | 51.72 | 53.56 | -1.84 |
| Vehicles 1 | 66.92 | 68.60 | -1.68 |
| Vehicles 2 | 66.96 | 72.78 | -5.32 |
| KITTI | 59.74 | 58.13 | 1.61 |
| EuroSAT | 70.48 | 71.48 | -1.00 |
| Média | — | — | -1.55 |

## 6.6 Poda não-estruturada por Aprendizado por Reforço - o método PuRL

Diferentemente dos métodos clássicos de poda de parâmetros, a poda como uma tarefa de aprendizado por reforço (PuRL) requer a modelagem de um ambiente e um agente de aprendizado. A caracterização do ambiente utilizado nestes experimentos foram definidas na Seção 3.3 e o agente, i.e. o algoritmo que irá interagir com o ambiente, é uma *Deep Q-Network* (DQN) (Mnih *et al.*, 2015) que combina o algoritmo *Q-Learning* (Watkins, 1989) com redes neurais profundas. Existem diversas implementações de DQN disponíveis publicamente, optamos por utilizar a implementação do *OpenAI Baselines* (Dhariwal *et al.*, 2017) por ser uma instituição mundialmente reconhecida pela excelência em pesquisa em Inteligência Artificial.

### 6.6.1 prune2transfer — Poda do Modelo

Os hiperparâmetros da DQN utilizados para realizar a poda da VGG-19 estão descritos na Tabela 6.5. Tendo em vista que um episódio nesta tarefa consiste em percorrer todas as camadas da rede neural a ser podada, os valores dos passos foram ajustados para a arquitetura da rede em questão, no caso uma rede com 19 camadas. A arquitetura da rede neural usada como aproximador da função $Q$ possui duas camadas escondidas com 200 unidades cada.

**Tabela 6.5:** *Hiperâmetros da DQN utilizada na poda não-estruturada da VGG-19.*

| Hiperparâmetro | Valor |
|---|---|
| Total de passos | 2000 |
| Taxa de aprendizado | 0.001 |
| Tamanho do *buffer* | 300 |
| Fração de exploração | 0.7 |
| $\epsilon$ final de exploração | 0.02 |
| Tamanho do lote | 32 |
| Aprendizado inicia a partir do passo | 100 |
| Fator de desconto ($\beta$) | 0.982 |
| Frequência de atualização da rede *target* | 15 |

Os valores definidos como meta para a acurácia e esparsidade foram 55% (isto é, queremos pelo menos manter a acurácia do modelo original) e 40% (mínimo empírico), respectivamente. Portanto, o objetivo desta tarefa de RL é minimizar a função de custo, dado que a recompensa é negativa, mantendo a acurácia maior ou igual ao obtido pelo modelo sem poda (original) e induzindo uma esparsidade de pelo menos 40% no modelo. A Figura 6.6 ilustra as recompensas acumuladas obtidas a cada episódio do treinamento, ou seja, a soma das recompensas obtidas após podar cada camada.

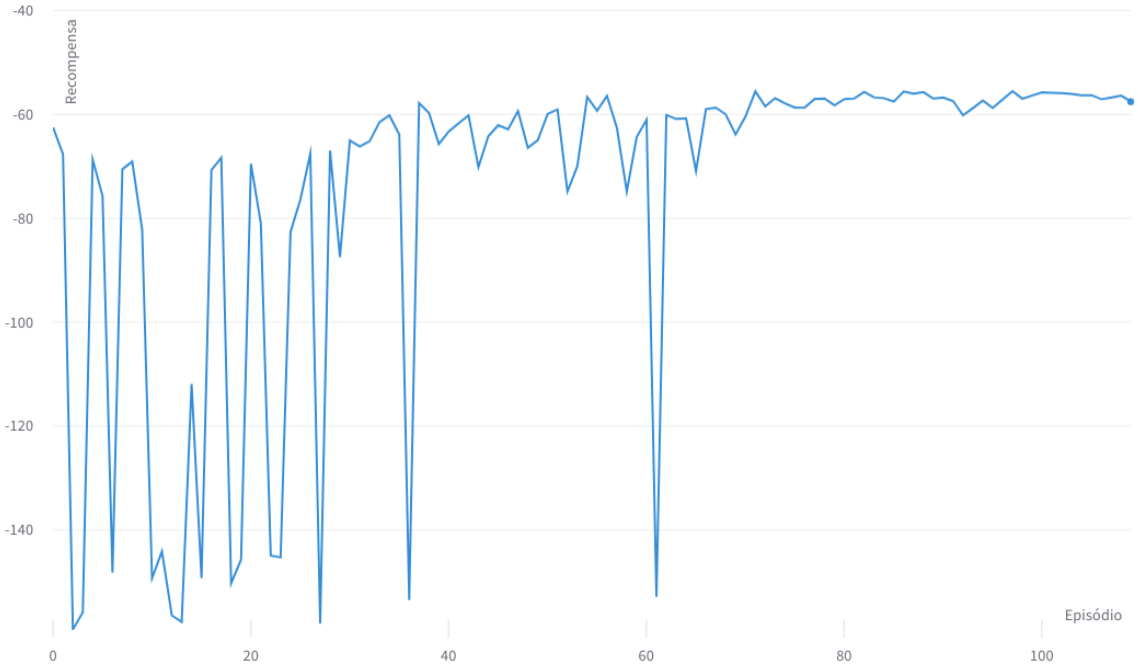



**Figura 6.6:** *Evolução da recompensa acumulada obtida a cada episódio do treinamento da VGG-19 usando Aprendizado por Reforço (método PuRL).*

Analisando a Figura 6.6, é possível notar que a partir de 70 episódios o aprendizado se estabiliza, atingindo um custo acumulado mínimo de aproximadamente −55. A política de poda aprendida neste treinamento foi {0.4, 0.6, 0.6, 0.6, 0.6, 1.0, 1.0, 1.0, 1.6, 1.6, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8 e 1.8}, seguindo a ordem da esquerda para direita das camadas ilustradas na Figura 6.1. Todas as 19 camadas são podadas com um fator de multiplicação que aumenta o valor de corte pelo desvio-padrão, com exceção das camadas iniciais do modelo que serão podadas numa intensidade maior (Seção 3.3.1). **Aplicando esta política de poda e retreinando o modelo, foi possível alcançar uma acurácia de** 57.87% **mantendo** 6.96% **dos parâmetros originais, ou seja,** 93.04% **de esparsidade.**

### 6.6.2 PRUNE2TRANSFER — Transferência de Aprendizado

Visto que o modelo final podado é similar ao modelo da Seção 6.4, pois ambos utilizaram um critério de poda não-estruturada na VGG-19, utilizaremos as mesmas configurações experimentais da sua etapa de a transferência de aprendizado.

**Tabela 6.6:** *Acurácias usando a Poda Não-estruturada com Aprendizado por reforço nas 22 tarefas meta: acurácia usando o modelo original ($Acc_{original}$); acurácia usando o modelo podado ($Acc_{podada}$) e a diferença entre elas.*

| Tarefa Meta | $Acc_{original}$(%) | $Acc_{podado}$(%) | $Acc_{original}$-$Acc_{podado}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 47.92 | 49.64 | -1.72 |
| Fish | 58.52 | 60.32 | -1.80 |
| Flowers | 49.56 | 54.20 | -4.64 |
| Food Containers | 67.08 | 70.84 | -3.76 |
| Fruit and Vegetables | 66.60 | 69.56 | -2.96 |
| Household Electrical Device | 71.96 | 74.60 | -2.64 |
| Household Furniture | 60.88 | 64.72 | -3.84 |
| Insects | 59.24 | 67.72 | -8.48 |
| Large Carnivores | 49.92 | 51.36 | -1.44 |
| Large Man-made Outdoor Things | 70.20 | 75.36 | -5.16 |
| Large Natural Outdoor Scenes | 71.68 | 76.64 | -4.96 |
| Large Omnivores and Herbivores | 51.88 | 58.68 | -6.80 |
| Medium-sized Mammals | 56.36 | 61.32 | -4.96 |
| Non-insect Invertebrates | 60.72 | 69.16 | -8.44 |
| People | 32.64 | 36.92 | -4.28 |
| Reptiles | 52.68 | 59.88 | -7.20 |
| Small Mammals | 46.52 | 49.36 | -2.84 |
| Trees | 51.72 | 55.16 | -3.44 |
| Vehicles 1 | 66.92 | 73.16 | -6.24 |
| Vehicles 2 | 66.96 | 74.48 | -7.52 |
| EuroSAT | 70.48 | 70.54 | -0.06 |
| KITTI | 59.74 | 61.09 | -1.35 |
| Média | — | — | -4.30 |

Analisando os dados da Tabela 6.6, temos que o uso do modelo podado na transferência de aprendizado, apresenta um desempenho superior ao uso do modelo original em todas as 22 tarefas meta, acarretando em uma diferença de −4.30 pontos percentuais em média. E como este é um valor negativo, **podemos concluir neste experimento que um modelo que mantém** 6.96% **dos parâmetros originais pode ser adaptado às novas tarefas meta sem perda de generalidade**.

## 6.7 Poda estruturada por Aprendizado por Reforço - o método StructPuRL

A poda estruturada como uma tarefa de aprendizado por reforço (StructPuRL) foi proposta neste trabalho e faz uso do arcabouço do PuRL com modificações no espaço de ações e critério de poda, conforme descrito na Seção 4.3. Assim como no PuRL, o algoritmo de aprendizado para poda utilizado foi uma *Deep Q-Network* (DQN).

### 6.7.1 PRUNE2TRANSFER — Poda do Modelo

Os hiperparâmetros da DQN utilizados para realizar a poda com RL da VGG-19 estão descritos na Tabela 6.7. Tendo em vista que um episódio nesta tarefa consiste em percorrer todas as camadas da rede neural a ser podada, os valores dos passos foram ajustados para a arquitetura de uma rede com 16 camadas convolucionais. A arquitetura da rede neural usada como aproximador da função $Q$ possui duas camadas escondidas com 200 unidades cada.

**Tabela 6.7:** *Hiperâmetros da DQN utilizada na poda estruturada da VGG-19.*

| Hiperparâmetro | Valor |
|---|---|
| Total de passos | 3000 |
| Taxa de aprendizado | 0.001 |
| Tamanho do *buffer* | 300 |
| Fração de exploração | 0.8 |
| $\epsilon$ final de exploração | 0.02 |
| Tamanho do lote | 32 |
| Aprendizado inicia a partir do passo | 150 |
| Fator de desconto ($\beta$) | 0.982 |
| Frequência de atualização da rede *target* | 15 |

Os valores definidos como meta para a acurácia e esparsidade foram 53% e 70%, respectivamente. A acurácia meta é um pouco inferior a acurácia do modelo sem poda com o objetivo de atingir uma maior esparsidade, ainda que perdendo um pouco de acurácia. A Figura 6.7 ilustra as recompensas obtidas a cada episódio do treinamento, ou seja, a soma das recompensas obtidas após podar cada camada de forma estruturada.

Analisando a Figura 6.7, é possível notar que a partir de 150 episódios o aprendizado começa a estabilizar, atingindo uma recompensa (negativa) acumulada mínima de aproximadamente −68. A política de poda aprendida neste treinamento para cada camada convolucional foi $\{1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.8, 0.8, 0.8, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.2\}$ seguindo a ordem da esquerda para direita das camadas ilustradas na Figura 6.1. Nesta política as seis primeiras camadas convolucionais não são podadas e as camadas seguintes são podadas com proporções crescentes. Na última camada, apenas 20% dos filtros são mantidos. **Aplicando esta política de poda e retreinando o modelo, foi possível alcançar uma acurácia de 56.20% com uma redução de 44% dos parâmetros do modelo original**. A Tabela 6.8 traz uma comparação do modelo original e deste modelo resultante da poda estruturada por RL.

Analisando a Tabela 6.8, temos que o o modelo podado apresenta uma acurácia superior, além de um tamanho em disco menor que o modelo original. Em comparação ao modelo podado de forma estruturada sem uma política (Seção 6.5) o modelo gerado pelo StructPuRL apresenta uma compressão similar, porém sem a necessidade de analisar individualmente os efeitos da poda em cada camada. Portanto, utilizar aprendizado por reforço nesta tarefa traz o benefício da avaliação do impacto da remoção de filtros no modelo como um todo, além de dispensar a análise de um "especialista" para a tomada de decisão.

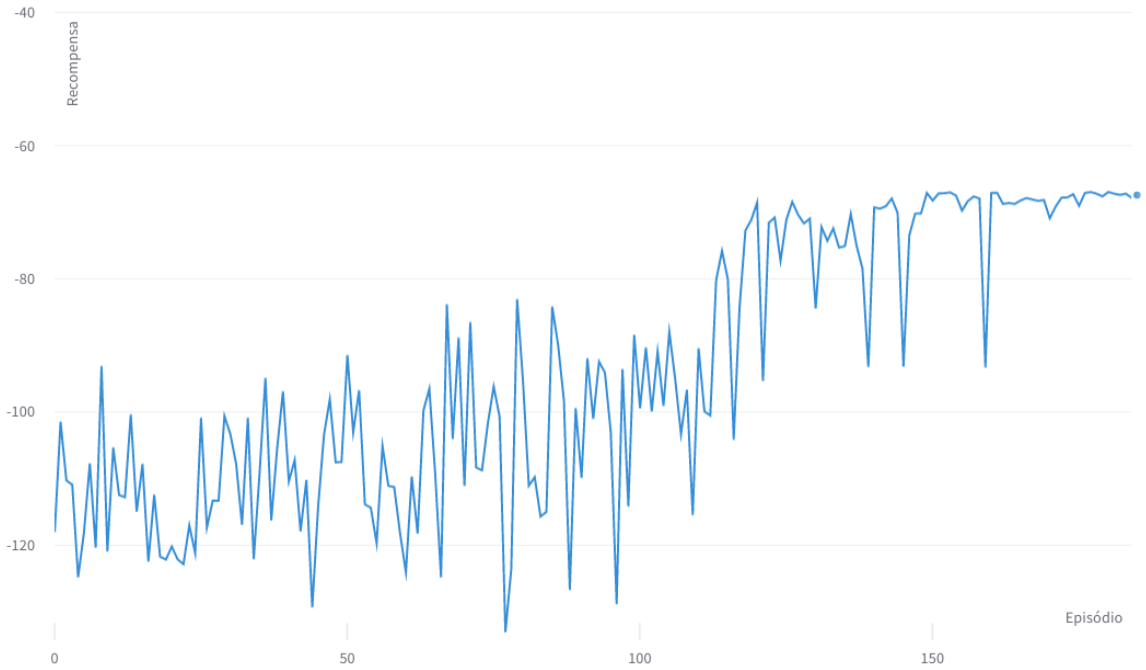



**Figura 6.7:** *Evolução da recompensa acumulada obtida a cada episódio do treinamento da VGG-19 usando Aprendizado por Reforço (método StructPuRL).*

**Tabela 6.8:** *Comparação entre o modelo original da VGG-19 e o modelo podado com redução de* 44% *dos parâmetros.*

| Modelo | Total de Parâmetros | Acurácia (%) | Tamanho em Disco (MB) |
|---|---|---|---|
| Original | 39 029 854 | 55.53% | 148.97 |
| Podado | 21 825 891 | 56.20% | 83.34 |

### 6.7.2 PRUNE2TRANSFER — Transferência de Aprendizado

Visto que o modelo final podado é similar ao modelo da Seção 6.4, pois ambos utilizaram um critério de poda estruturada na VGG-19, utilizaremos as mesmas configurações experimentais da sua etapa de a transferência de aprendizado. A Tabela 6.9 relata as acurácias do modelo sem poda ($\text{Acc}_{\text{original}}$), do modelo podado a partir da política aprendida ($\text{Acc}_{\text{podado}}$) e a diferença entre essas acurácias.

**Tabela 6.9:** *Acurácias usando a Poda Estruturada com Aprendizado por reforço nas 22 tarefas meta: acurácia usando o modelo original ($Acc_{original}$); acurácia usando o modelo podado ($Acc_{podada}$) e a diferença entre elas.*

| Tarefa Meta | $Acc_{original}$(%) | $Acc_{podado}$(%) | $Acc_{original}$-$Acc_{podado}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 47.92 | 50.52 | -2.60 |
| Fish | 58.52 | 57.20 | 1.32 |
| Flowers | 49.56 | 52.92 | -3.36 |
| Food Containers | 67.08 | 72.52 | -5.44 |
| Fruit and Vegetables | 66.60 | 69.88 | -3.28 |
| Household Electrical Device | 71.96 | 73.04 | -1.08 |
| Household Furniture | 60.88 | 60.00 | 0.88 |
| Insects | 59.24 | 64.44 | -5.20 |
| Large Carnivores | 49.92 | 48.16 | 1.76 |
| Large Man-made Outdoor Things | 70.20 | 73.16 | -2.96 |
| Large Natural Outdoor Scenes | 71.68 | 73.32 | -1.64 |
| Large Omnivores and Herbivores | 51.88 | 57.52 | -5.64 |
| Medium-sized Mammals | 56.36 | 59.56 | -3.20 |
| Non-insect Invertebrates | 60.72 | 63.24 | -2.52 |
| People | 32.64 | 30.36 | 2.28 |
| Reptiles | 52.68 | 56.20 | -3.52 |
| Small Mammals | 46.52 | 46.08 | 0.44 |
| Trees | 51.72 | 53.72 | -2.00 |
| Vehicles 1 | 66.92 | 68.96 | -2.04 |
| Vehicles 2 | 66.96 | 72.00 | -5.04 |
| KITTI | 59.74 | 59.11 | 0.63 |
| EuroSAT | 70.48 | 69.44 | 1.04 |
| Média | — | — | -1.87 |

Analisando os dados da Tabela 6.9, temos que o modelo podado apresenta um desempenho superior ao modelo original na maioria das 22 tarefas, acarretando em uma diferença de −1.87 pontos percentuais em média. E como este é um valor negativo, **podemos concluir neste experimento que um modelo que mantém** 56.20% **dos parâmetros originais pode ser adaptado às novas tarefas sem perda de generalidade**.

## 6.8 Discussão dos Resultados

O objetivo de experimentar diferentes critérios de poda no arcabouço PRUNE2TRANSFER é avaliar se existe algum processo de poda que ao mesmo tempo que propicie uma compressão do modelo original, também garanta a preservação dos extratores de características que são fundamentais para uma transferência de aprendizado eficaz. A fim de avaliar os métodos de poda foram estabelecidos quatro métricas, sendo estes:

- M1 — diferença entre a acurácia do modelo sem poda e do modelo podado na tarefa original;

- M2 — percentual de poda atingido;
- M3 — taxa de compressão do modelo podado quanto ao espaço em disco;
- M4 — média das diferenças entre a acurácia do modelo sem poda e do modelo podado nas tarefas meta (Equação 5.1).

A Tabela 6.10 apresenta os valores das quatro métricas adotadas para os critérios de poda aplicados à VGG-19.

**Tabela 6.10:** *Comparação de todos os critérios testados.*

| Tipo de Poda | Critério | M1 | M2 | M3 | M4 |
|---|---|---|---|---|---|
| Não-Estruturada | Teste-t | −2.02 | 95.96% | 1× | −6.21 |
| | PuRL | −2.34 | 93.04% | 1× | −4.30 |
| Estruturada | Norma-$L_1$ | −0.07 | 45.60% | 1.84× | −1.55 |
| | StructPuRL | −0.67 | 44.00% | 1.79× | −1.87 |

Analisando a Tabela 6.10 temos que para a VGG-19, todos os critérios de poda obtiveram resultados melhores nas métricas M1 e M4, uma vez que todos os valores são negativos. Isto pode ser explicado pela redução da complexidade e maior generalização alcançada com o modelo podado quando comparamos com o modelo original (Bartoldson *et al.*, 2018). No caso, a VGG-19 com todos os seus parâmetros pode ser complexa demais para a tarefa original e tarefas meta, levando a um sobreajuste do modelo original, o que é atenuado pela poda.

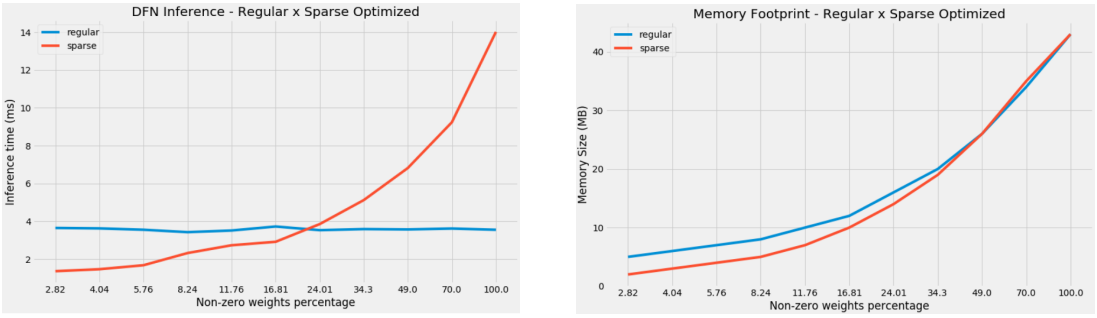
Notamos ainda na Tabela 6.10 que os métodos não-estruturados Teste-t e PuRL são capazes de induzir uma esparsidade maior na rede (95.96% e 93.04%, respectivamente), além de apresentarem modelos mais acurados (−2.02 e −2.34 pontos percentuais, respectivamente) tanto na tarefa original como nas tarefas meta. Ambos os métodos estruturados atingiram percentuais de esparsidade próximos e similares ao obtido pelo modelo sem poda. Apesar de apresentarem resultados semelhantes, o StructPuRL é mais vantajoso que o método estruturado da Norma-$L_1$, pois não requer uma análise de sensibilidade das camadas, realizando a poda de forma totalmente automática (agente RL).

De uma maneira geral, a Tabela 6.10 mostra que os critérios de poda não-estruturada superam os critérios de poda estruturada a menos da taxa de compressão do modelo podado quanto ao espaço em disco (métrica M3). Na próxima seção discutimos como podemos mitigar essa limitação.

### 6.8.1 Poda não-estruturada: eficiência dos modelos esparsos

Um dos benefícios da poda não-estruturada é a redução massiva do número de parâmetros não-zerados de uma rede, zerando conexões pouco relevantes ao desempenho do modelo. Para níveis altos de esparsidade pode-se obter modelos de menor tamanho em disco e menor tempo de inferência porém, para que essas melhorias sejam de fato exploradas, é necessário utilizar estruturas de dados adequadas que otimizem a computação em matrizes esparsas (Han e Dally, 2017). A Figura 6.8 ilustra essa ideia comparando duas implementações diferentes de uma rede neural DFN variando sua esparsidade: (i)

utilizando uma matriz no formato linha esparsa comprimida (*Compressed Sparse Row* — CSR) e (ii) uma matriz no formato tradicional (sem compressão de esparsidade).




**(a)** **(b)**

**Figura 6.8:** *Gráficos comparativos entre matrizes esparsas em formato tradicional e em CSR, para diferentes níveis de esparsidade (eixo-x).*

A Figura 6.8a mostra uma análise empírica dessas ideias com uma rede mais simples do que a VGG19, pois não encontramos uma otimização para inferências esparsas no TensorFlow. Para isso usamos a rede DFN (Seção 2.3). Na figura, comparamos o tempo de inferência em milissegundos das duas implementações: a DFN implementada com uma matriz CSR (curva vermelha) e a DFN implementada com uma matriz regular (curva azul). Note que, neste exemplo, a implementação com CSR resulta em tempos de inferência menores do que a implementação com uma matriz regular, *somente para modelos com menos de* 24.01% *de seus parâmetros diferentes de zero*. Ou seja, só é possível explorar as vantagens de um modelo comprimido com alto nível de esparsidade.

A Figura 6.8b compara o tamanho em disco dos modelos em *Megabytes* (MB) para diferentes níveis de esparsidade. Apesar deste exemplo apresentar curvas similares, o ganho em termos de memória com o uso de uma CSR (curva vermelha) também pode ser observado somente em modelos com alta esparsidade, isto é, *somente para modelos com menos de* 34.30% *de seus parâmetros diferentes de zero.*

Com base nos resultados da nossa análise (Seção 6.8), os modelos obtidos após uma poda não-estruturada, com precisão satisfatória, alcançaram em média 5.5% de seus parâmetros diferentes de zero, o que indica que o uso de uma implementação da rede VGG-19 com matrizes CSR podem reduzir o espaço de memória em disco para o armazenamento, bem como o tempo de inferência.

# Capítulo 7

# Conclusões e Perspectivas de Trabalhos Futuros

Neste projeto de mestrado propusemos e analisamos empiricamente a metodologia PRUNE2TRANSFER que consiste de um arcabouço para poda de parâmetros do modelo na tarefa origem antes de fazer a transferência de aprendizado para um conjunto de tarefas meta. Além de constatar a viabilidade da proposta, implementamos e comparamos diferentes critérios de poda, populares na área de compressão de modelos, incluindo propostas de extensões e modificações.

## 7.1 Conclusões

Da literatura, o método de poda iterativa não-estruturada baseada em magnitude, não especifica de maneira objetiva um critério de comparação entre as diferentes escolhas de poda ou qual é o melhor ponto de parada do algoritmo. Portanto, neste trabalho, propomos uma extensão usando um método estatístico chamado de *poda não-estruturada iterativa usando teste-t pareado Bayesiano*. Já o método de poda estruturada baseada na Norma $L_1$ depende da definição de percentuais de filtros preservados por camada. Os autores afirmam que estes percentuais podem ser obtidos através de uma análise de sensibilidade das camadas convolucionais. Porém, da mesma forma que na poda iterativa não-estruturada, a escolha dos percentuais é feita pela análise visual gráfica das curvas de sensibilidade, tornando a tomada de decisão subjetiva. Portanto, neste trabalho, também modificamos o algoritmo desse método adicionando um critério para a escolha do melhor percentual de poda de cada camada.

Os experimentos mostram que, independente do critério de poda utilizado na VGG-19, foi possível: (i) podar o modelo em sua tarefa original sem impactar negativamente em seu desempenho e (ii) transferir o modelo podado para as tarefas meta alcançando em média acurácias melhores do que as obtidas com a transferência do modelo sem poda. Estes resultados indicam que redes pré-treinadas utilizadas em transferência de aprendizado poderiam ser compactadas e então compartilhadas sem perda de generalidade. Portanto, realizar a poda dos filtros convolucionais não levou a um sobreajuste do modelo à tarefa original, que permaneceu com a capacidade de extrair características relevantes

ao aprendizado de outras tarefas meta.

Quanto à comparação dos métodos de poda temos que a poda não-estruturada atinge uma esparsidade maior que a poda estruturada, porém seu potencial ganho computacional depende de implementações de *software* eficientes para operações esparsas, como exemplificado na Seção 6.8.1. Analisando os resultados dos critérios não-estruturados (Tabela 6.10), o PuRL apresentou resultados próximos ao Teste-t com a vantagem de aprender a melhor poda, ao invés de deixar para o usuário a escolha do número de iterações de poda.

Já nos casos em que não é possível realizar inferências que levem em consideração a alta esparsidade do modelo, é recomendável o uso da poda estruturada visto que essa gera uma arquitetura menor ao remover as estruturas zeradas na rede original. Apesar dos resultados obtidos com os métodos estruturados serem próximos, o método StructPuRL proposto automatiza a escolha do percentual de poda por camada, não dependendo de uma análise prévia de sensibilidade, sendo essa uma proposta original, não encontrada na literatura.

Em suma, nossos estudos mostram que os critérios de poda não-estruturada superam os critérios de poda estruturada a menos da taxa de compressão do modelo podado quanto ao espaço em disco (métrica M3).

## 7.2 Perspectivas de Trabalhos Futuros

A seguir destacamos algumas direções futuras de pesquisa decorrentes deste trabalho de mestrado, a saber:

**Adicionar uma etapa de poda após a transferência de aprendizado**. O objetivo desta segunda poda é reduzir ainda mais o modelo transferido só que desta vez após o aprendizado para a tarefa meta. É esperado que esta poda seja computacionalmente menos custosa do que a poda inicial por ser realizada em um modelo previamente podado.

**Expandir o estudo para outras redes neurais e disponibilizar os modelos podados**. Neste trabalho investigamos apenas a VGG-19, mas seria interessante aplicar o arcabouço PRUNE2TRANSFER para outras redes populares em transferência de aprendizado, como por exemplo: ResNet (K. He *et al.*, 2016), DenseNet (Huang *et al.*, 2017) e MobileNet (Howard *et al.*, 2017). Além de apenas replicar o estudo, acreditamos que seria proveitoso para a comunidade se disponibilizássemos os modelos pré-treinados já podados, a fim de que estes fossem utilizados como modelo base para transferência de aprendizado.

**Expandir o estudo para outros critérios de poda**. A escolha de quais critérios de poda implementar neste trabalho foi feita selecionando um representante significativo de cada critério conhecido na literatura. Nossa pesquisa consistiu em métodos estruturados diretos e automáticos e em métodos não-estruturados iterativos e automáticos, porém seria interessante aprofundar a pesquisa em outros métodos estruturados.

**Melhorar o método StructPuRL**. Uma das contribuições deste trabalho foi o StructPuRL que é uma modificação do método de poda automático não-estruturada PuRL para a realização de uma poda estruturada. Devido ao critério de poda adotado ser destinado apenas às camadas convolucionais, uma possível melhoria do método seria também remover

neurônios, viabilizando a poda de camadas totalmente conectadas.

**Ampliar o escopo de tarefas meta**. Nas tarefas meta estudadas consideramos majoritariamente tarefas de classificação de imagens naturais e apenas duas tarefas de imagens não-naturais. Seria interessante aplicar o arcabouço PRUNE2TRANSFER de transferência dos modelos podados para tarefas relacionadas a imagens médicas e comparar com a transferência tradicional, visto que muitas aplicações atuais de transferência de aprendizado pertencem a este domínio.

# Apêndice A

# Experimentos e Resultados – CNN-simples

Os experimentos e resultados aqui apresentados seguem a mesma estrutura do Capítulo 6, porém com a rede convolucional CNN-simples.

## A.1 CNN-simples

A Figura A.1 ilustra a arquitetura da CNN-simples, mostrando as dimensões de entrada adotada em todos os experimentos ($32 \times 32 \times 3$), dois blocos convolucionais intercalados por camadas de agrupamento *max pooling* (laranja escuro), seguido por uma camada densa com 256 unidades e finalmente uma saída com 10 unidades com uma ativação *softmax*.

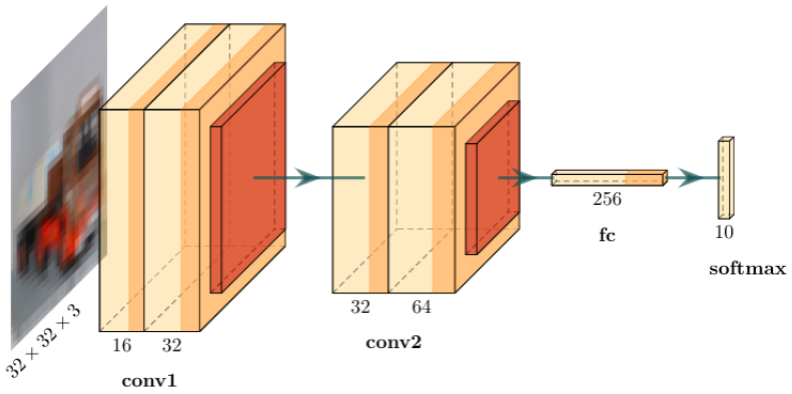



**Figura A.1:** *Arquitetura da CNN-simples.*

Na Figura A.1, conv_1 e conv_2 representam dois blocos de convolução, cada um com duas camadas de 16 e 32 filtros, e 32 e 64 filtros, repectivamente, para todas as camadas convolucionais foi adotado um campo receptivo de $3 \times 3$. Os elementos em laranja claro representam as camadas de ativação, que neste caso é a função *Leaky ReLU*. As camadas de *max pooling* utilizam um tamanho de *pool* de $2 \times 2$.

O número total de parâmetros da CNN-simples é 1 084 234.

## A.2 Tarefa Original

Todas as imagens do conjunto de dados a seguir são coloridas e possuem as dimensões $32 \times 32$.

- CIFAR-10 (Krizhevsky, Hinton *et al.*, 2009) — consiste em imagens naturais categorizadas em 10 classes sendo estas: avião, automóvel, ave, gato, cachorro, cervo, sapo, cavalo, barco e caminhão. Os dados são balanceados com 50 000 imagens para treino e 10 000 imagens para teste.

A tarefa original da CNN-simples é aprender a classificar uma imagem em uma das 10 categorias do CIFAR-10.

## A.3 Tarefas Meta

As tarefas meta utilizadas para a CNN-simples foram as mesmas descritas na Seção 6.2.2.

## A.4 Etapa 1 — Aprendizado inicial

O treinamento da CNN-simples foi feito em 80 épocas em lotes (*batches*) de 32 usando o otimizador Adamax (Kingma e Ba, 2015) com taxa de aprendizado iniciando em $5 \times 10^{-3}$ com decaimento exponencial a cada época. **A acurácia no conjunto de teste do CIFAR-10, que consiste em** 1 000 **imagens de cada classe, foi de** 80.32%.

Seguindo o diagrama 5.1 do método PRUNE2TRANSFER, após a etapa de treinamento inicial da rede, temos as etapas de poda e *transfer learning*, respectivamente. As seções a seguir apresentarão os resultados destas duas etapas para cada critério de poda estudado. Ao final, serão discutidos os resultados de todos os modelos.

## A.5 PRUNE2TRANSFER — Poda Não-Estruturada

### A.5.1 Etapa 2 — Poda

A poda não estruturada do modelo foi realizada por 20 iterações, com os mesmos hiperparâmetros do treinamento inicial. A Figura A.2 ilustra o gráfico da acurácia para cada percentual de poda não estruturada.

Na Figura A.2, a linha tracejada vermelha indica a melhor poda considerando o critério do "cotovelo", na qual temos um modelo com 5.76% dos parâmetros do modelo original e com uma acurácia de 79.65%. Porém, como discutido na Seção 4.1.1, é possível utilizar um teste probabilístico para averiguar a equivalência prática do modelo podado ao modelo sem poda. E com base neste teste tomar uma decisão mais informada sobre o melhor percentual de poda.

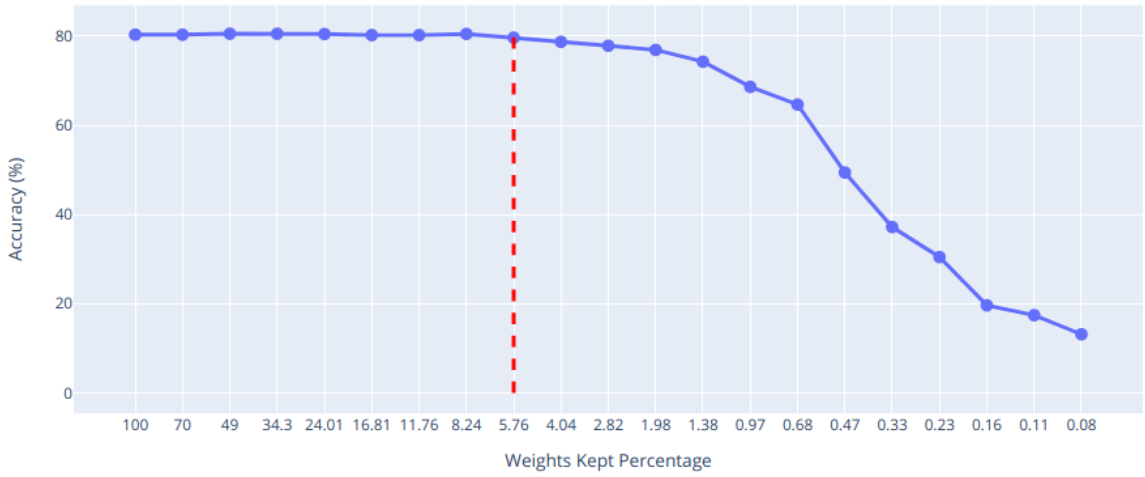



**Figura A.2:** *Acurácia no conjunto de teste de cada modelo após poda iterativa não estruturada no modelo CNN-simples.*

A Tabela A.1 traz os valores da probabilidade: (i) do modelo original ($P_{\text{unpruned}}$) ser praticamente melhor que o modelo podado, (ii) de serem praticamente equivalentes ($P_{\text{rope}}$) e (iii) do modelo podado ($P_{\text{pruned}}$) ser praticamente melhor que o modelo original. Na tabela, os valores de $P_{\text{rope}} + P_{\text{pruned}} > 0.5$, que estão destacados em verde, indicam modelos que são praticamente melhores ou equivalentes ao modelo sem poda, logo são modelos em que não há uma diferença prática nas acurácias considerando uma região de ±1%.

**Tabela A.1:** *Teste-t pareado Bayesiano para a poda não estruturada iterativa da CNN-simples com região de prática equivalência a* 1% *e* 5*-folds.*

| Weights kept (%) | $P_{\text{unpruned}}$ | $P_{\text{rope}}$ | $P_{\text{pruned}}$ | $P_{\text{rope}} + P_{\text{pruned}}$ |
|---|---|---|---|---|
| 70.00 | 0.0009 | 0.9988 | 0.0003 | 0.9991 |
| 49.00 | 0.0042 | 0.9885 | 0.0073 | 0.9958 |
| 34.30 | 0.0005 | 0.999 | 0.0005 | 0.9995 |
| 24.01 | 0.0 | 0.9999 | 0.0001 | 1.0000 |
| 16.81 | 0.0024 | 0.9961 | 0.0015 | 0.9976 |
| 11.76 | 0.0069 | 0.9907 | 0.0024 | 0.9931 |
| 8.24 | 0.044 | 0.9491 | 0.0069 | 0.9560 |
| **5.76** | **0.0206** | **0.9774** | **0.0020** | **0.9794** |
| 4.04 | 0.8398 | 0.1565 | 0.0037 | 0.1602 |
| 2.82 | 0.9871 | 0.0123 | 0.0006 | 0.0129 |
| 1.98 | 0.99 | 0.0082 | 0.0019 | 0.0100 |
| 1.38 | 0.9999 | 0.0001 | 0.0000 | 0.0001 |
| 0.97 | 0.9964 | 0.0015 | 0.0021 | 0.0036 |
| 0.68 | 0.9998 | 0.0001 | 0.0001 | 0.0002 |
| 0.47 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.33 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.23 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.16 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.11 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.08 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |

Visto que queremos podar ao máximo o modelo, minimizando o impacto na acurácia, **a poda ótima é de** 94.24% **dos parâmetros o que é equivalente a manter** 5.76% **dos parâmetros diferentes de zero.**

### A.5.2 Etapa 3 — Transferência de Aprendizado

Na etapa final, todos os modelos podados e o modelo original são utilizados como base para realizar o *transfer learning* do modelo para cada uma das 22 tarefas meta. Dado que a CNN-simples é uma rede neural com poucos parâmetros, em todas as tarefas foi realizado o *fine-tuning* do modelo, ou seja, todos os pesos diferentes de zero foram ajustados durante o treinamento em cada tarefa. A arquitetura da CNN-simples é semelhante ao ilustrado na Figura A.1, com exceção da última camada que foi modificada de acordo com o número de classes das tarefas meta.

Quanto aos hiperparâmetros de cada treinamento, as tarefas do CIFAR-100 utilizaram o otimizador RMSProp (Hinton, Srivastava *et al.*, 2012) com taxa de aprendizado fixa em $5 \times 10^{-5}$ por 40 épocas em lotes de 16 imagens. No caso da tarefa KITTI, o otimizador escolhido foi o SGD (momentum de 0.9 e atualização de Nesterov) com uma taxa de aprendizado fixa em $5 \times 10^{-4}$, treinando por 40 épocas em lotes de 16. Já para a tarefa EuroSAT foi utilizado o otimizador RMSProp com taxa de aprendizado fixa em $5 \times 10^{-5}$ e regularização de parada precoce (*early stopping*), treinando por até 80 épocas em lotes de

16. As Figuras A.3 e A.4 ilustram os gráficos da acurácia média de 5 adaptações a cada uma das tarefas meta usando o modelo original e todos os modelos podados (Figura A.2).

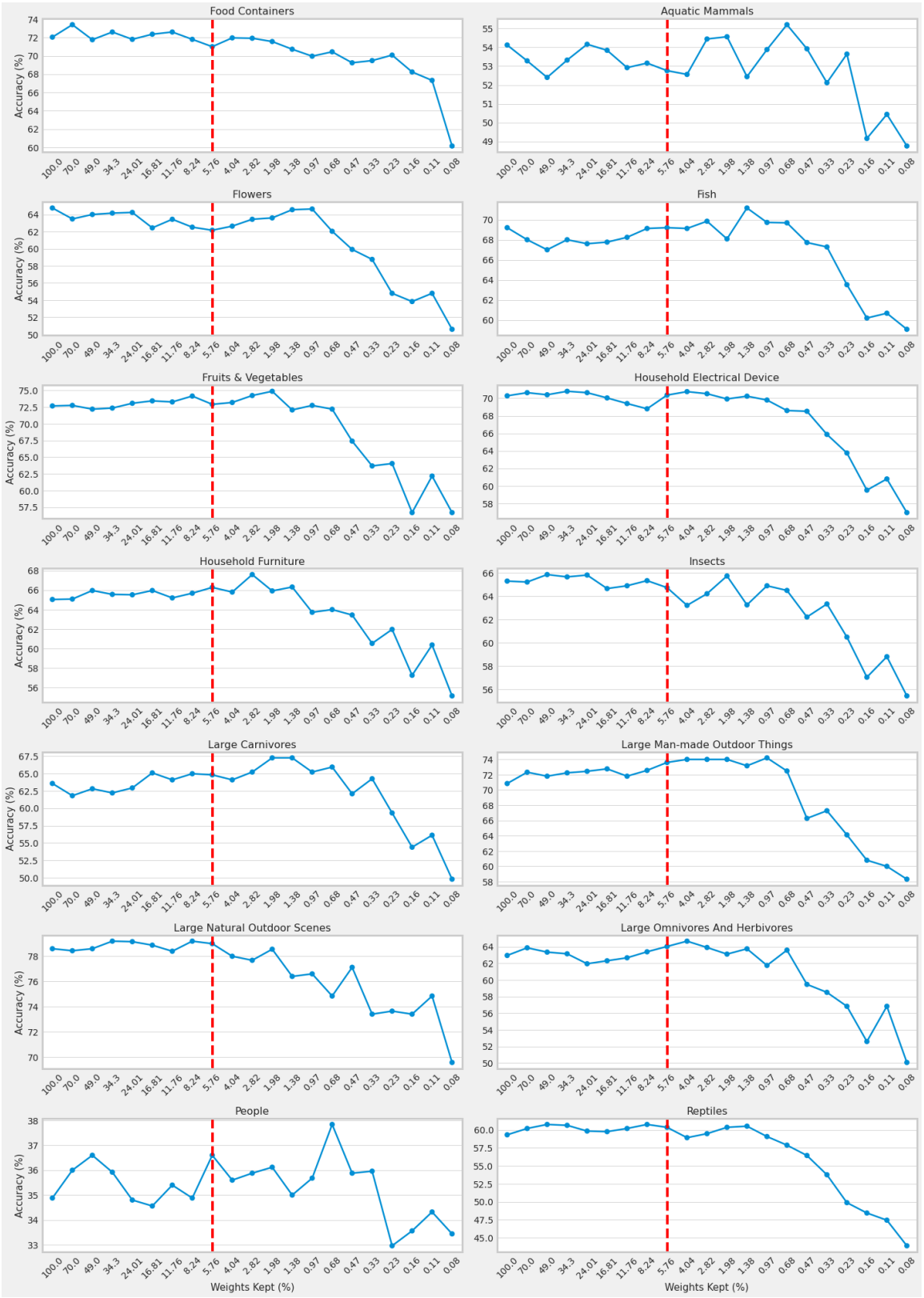



**Figura A.3:** *Acurácia média de 5 iterações de transfer learning com fine-tuning para as tarefas meta usando diferentes podas e modelo original da CNN-simples.*

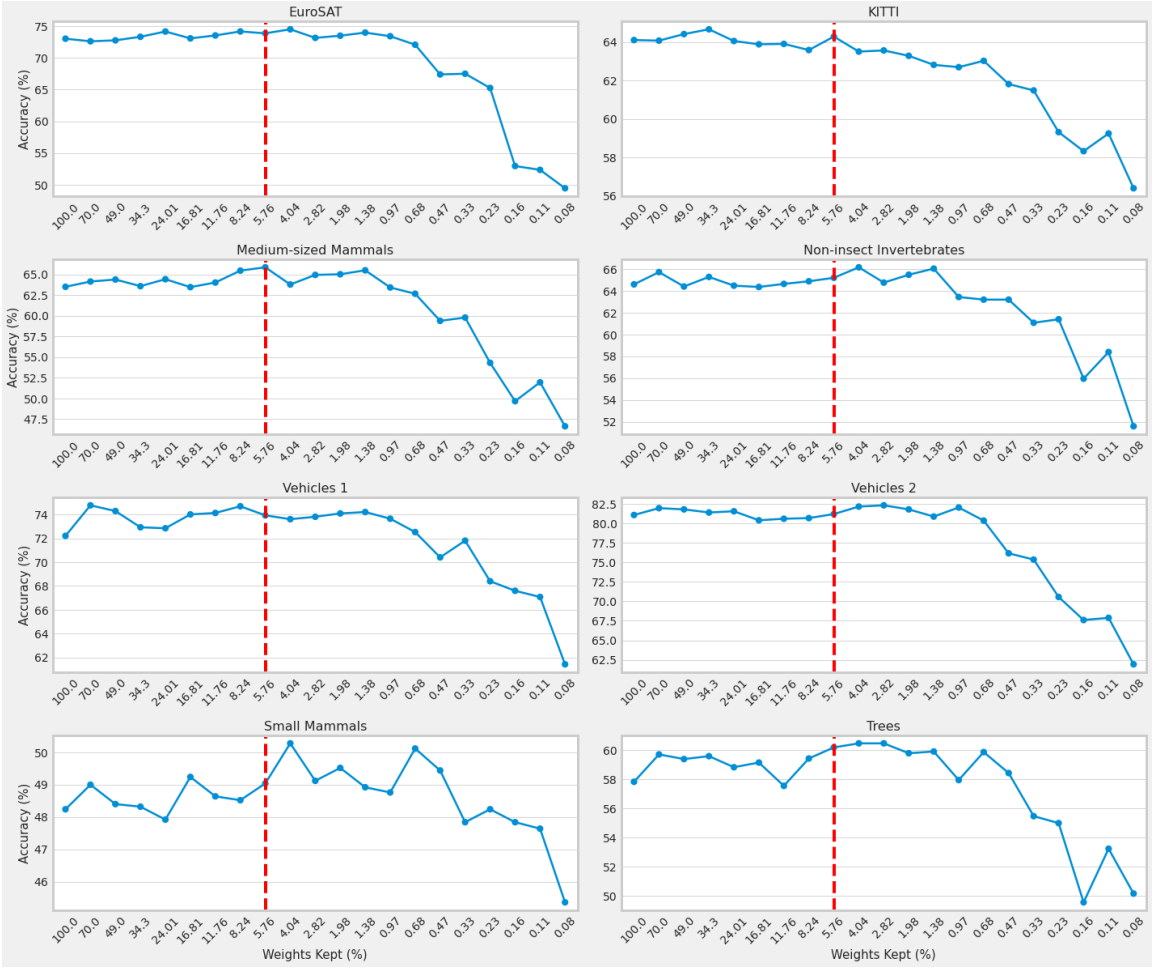



**Figura A.4:** *Continuação da Figura A.3.*

Nas Figuras A.3 e A.4, a linha tracejada vermelha indica o modelo com o melhor percentual de poda, mantendo 5.76% dos parâmetros originais, selecionado com o critério do teste-t pareado Bayesiano. As figuras também mostram o resultado da adaptação de outros modelos podados para fins de análise e discussão. É interessante notar que em todas as tarefas meta, existe um modelo podado que tem um desempenho igual ou melhor do que o desempenho obtido com o modelo original sem poda. O que possivelmente indica que alguns níveis de poda favorecem mais um domínio de tarefas do que outros, ou seja, é possível extrair representações mais significativas de algumas imagens do que de outras. Também é notório que podas muito extremas levam a modelos com baixa capacidade de adaptação à novas tarefas, impactando significativamente a acurácia.

Outra observação do experimento é que uma poda mais extrema não necessariamente gera um decaimento no desempenho, como ilustrado na Figura A.2, sendo possível recuperar ou até mesmo melhorar a acurácia. Este fenômeno é evidente na Figura A.4 nas tarefas *Small Mammals* e *Trees*, nos quais há uma melhora na acurácia após quedas consecutivas. Uma possível explicação talvez seja a teoria dos *winning tickets* (Frankle e Carbin, 2019) que afirma que dentro de uma rede existe uma subrede que dependendo da forma que for inicializada tem um desempenho igual ou superior a rede completa. Deste modo, os modelos mais podados podem iniciar de um ponto mais favorável na otimização destas

tarefas.

Uma vez que o objetivo do método PRUNE2TRANSFER é podar um modelo numa tarefa original sem grandes impactos e transferir/adaptar este modelo para tarefas meta, a Tabela A.2 relata as acurácias do modelo sem poda ($Acc_{original}$), do modelo ótimo escolhido na etapa de poda ($Acc_{podado}$) e a diferença entre essas acurácias.

**Tabela A.2:** *Acurácias nas tarefas meta do modelo sem poda e do modelo com uma poda ótima para a CNN-simples.*

| Target Task | $Acc_{original}$(%) | $Acc_{podado}$(%) | $Acc_{original}$-$Acc_{podado}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 54.12 | 52.76 | 1.36 |
| Fish | 69.20 | 69.20 | 0.00 |
| Flowers | 64.76 | 62.16 | 2.60 |
| Food Containers | 72.04 | 71.00 | 1.04 |
| Fruit and Vegetables | 72.68 | 72.92 | -0.24 |
| Household Electrical Device | 70.28 | 70.36 | -0.08 |
| Household Furniture | 65.04 | 66.28 | -1.24 |
| Insects | 65.28 | 64.72 | 0.56 |
| Large Carnivores | 63.56 | 64.80 | -1.24 |
| Large Man-made Outdoor Things | 70.84 | 73.60 | -2.76 |
| Large Natural Outdoor Scenes | 78.60 | 79.00 | -0.40 |
| Large Omnivores and Herbivores | 62.96 | 64.04 | -1.08 |
| Medium-sized Mammals | 63.52 | 65.88 | -2.36 |
| Non-insect Invertebrates | 64.60 | 65.20 | -0.60 |
| People | 34.88 | 36.60 | -1.72 |
| Reptiles | 59.32 | 60.36 | -1.04 |
| Small Mammals | 48.24 | 49.04 | -0.80 |
| Trees | 57.84 | 60.20 | -2.36 |
| Vehicles 1 | 72.20 | 73.92 | -1.72 |
| Vehicles 2 | 81.08 | 81.20 | -0.12 |
| EuroSAT | 73.02 | 73.86 | -0.84 |
| KITTI | 64.11 | 64.29 | -0.18 |
| Mean | — | — | -0.60 |

A última linha da Tabela A.2 traz a média da diferença entre as acurácias (−0.6 p.p.), que é o score de avaliação do *transfer learning* proposto na Seção 5.4. Como este é um valor negativo, **podemos concluir neste experimento que um modelo que mantém** 5.76% **dos parâmetros originais pode ser adaptado à novas tarefas sem perda de generalidade**. No geral, não acarretando em impactos negativos no desempenho se comparado à transferência do modelo original para as tarefas meta.

# A.6 PRUNE2TRANSFER — Poda Estruturada

## A.6.1 Etapa 2 — Poda

A poda estruturada da CNN-simples não foi realizada de forma iterativa, mas utilizando o método *one shot* baseado na análise de sensibilidade das camadas convolucionais. Portanto, nesta seção não analisaremos mais de um modelo podado como na Seção 6.4.

A Figura A.5 ilustra o gráfico da acurácia do modelo após poda individual de cada camada, considerando diferentes quantidades de filtros preservados. Cada curva está com o nome **block***x*_**conv***y*_*z*, em que *x* identifica o bloco convolucional, *y* identifica qual a camada dentro do bloco e *z* é a quantidade de filtros originalmente presente no modelo (Figura A.1).

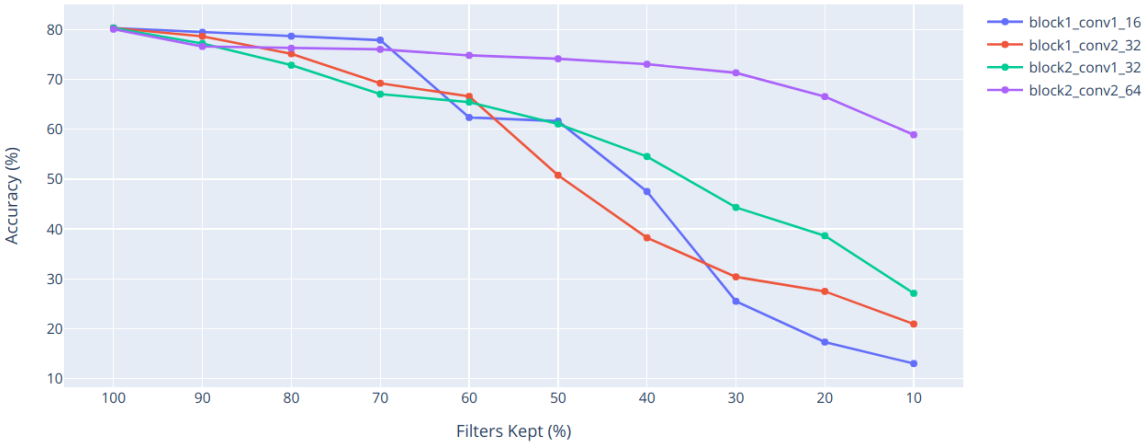



**Figura A.5:** *Acurácia no conjunto de teste após poda estruturada em cada camada convolucional da CNN-simples.*

Analisando a Figura A.5, temos que as camadas iniciais são mais sensíveis à poda e que a última camada convolucional (block2_conv2_64) apresenta uma maior redundância em seus filtros, viabilizando podas mais agressivas sem grandes perturbações na acurácia do modelo. Sendo assim, a quantidade de filtros que serão preservados por camada será de 70% para todas as camadas.

Seguindo o método de poda *one shot*, todas as camadas foram podadas e ao final retreinadas a fim de retomarem o desempenho. Neste treinamento final utilizamos o otimizador Adamax com taxa de aprendizado fixa em $5 \times 10^{-4}$ por 50 épocas em lotes de 32 imagens. **Atingindo uma acurácia de** 78.97% **e uma redução de** 32% **dos parâmetros originais**. Diferentemente dos modelos esparsos da poda não estruturada, a poda estruturada gera modelos menores e densos que têm impacto direto no tamanho em disco do modelo. A Tabela A.3 traz uma comparação do modelo original e deste modelo resultante da poda estruturada.

Analisando a Tabela A.3, temos que o o modelo podado apresenta uma pequena perda em acurácia, mas em compensação possui um tamanho em disco menor que o modelo original. O que favorece o uso deste modelo em dispositivos com baixa capacidade computacional. A redução no número de parâmetros poderia ser maior se removéssemos

**Tabela A.3:** *Comparação entre o modelo original da CNN-simples e o modelo podado com redução de 32% dos parâmetros.*

| Model | Total Parameters | Accuracy (%) | Memory footprint (MB) |
|---|---|---|---|
| Unpruned | 1 084 234 | 80.32% | 4.17 |
| Pruned | 739 364 | 78.97% | 2.86 |

unidades das camadas densas do modelo, porém o critério de poda estruturada adotado (Li *et al.*, 2017) se atém apenas a poda de filtros convolucionais (Capítulo 7).

O modelo podado ótimo, que manteve 68% dos parâmetros originais, será adaptado nas tarefas de *transfer learning* a seguir e, comparado com o modelo original seguindo o critério de score do método PRUNE2TRANSFER.

### A.6.2 Etapa 3 — Transferência de Aprendizado

Os hiperparâmetros de adaptação para cada tarefa foram os mesmos que os utilizados na Etapa 3 da Poda Estruturada. Dado que selecionamos um único modelo podado para ser comparado com o original apresentamos os resultados na Tabela A.4.

**Tabela A.4:** *Acurácias nas tarefas meta do modelo original da CNN-simples e do modelo podado de forma estruturada.*

| Target Task | $Acc_{original}$(%) | $Acc_{podado}$(%) | $Acc_{original}$-$Acc_{podado}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 54.12 | 49.00 | 5.12 |
| Fish | 69.20 | 66.32 | 2.88 |
| Flowers | 64.76 | 57.32 | 7.44 |
| Food Containers | 72.04 | 70.20 | 1.84 |
| Fruit and Vegetables | 72.68 | 66.84 | 5.84 |
| Household Electrical Device | 70.28 | 68.32 | 1.96 |
| Household Furniture | 65.04 | 65.60 | -0.56 |
| Insects | 65.28 | 66.16 | -0.88 |
| Large Carnivores | 63.56 | 56.00 | 7.56 |
| Large Man-made Outdoor Things | 70.84 | 68.76 | 2.08 |
| Large Natural Outdoor Scenes | 78.60 | 74.24 | 4.36 |
| Large Omnivores and Herbivores | 62.96 | 55.60 | 7.36 |
| Medium-sized Mammals | 63.52 | 59.72 | 3.80 |
| Non-insect Invertebrates | 54.60 | 64.96 | -0.36 |
| People | 34.88 | 35.32 | -0.44 |
| Reptiles | 59.32 | 54.96 | 4.36 |
| Small Mammals | 48.24 | 48.64 | -0.40 |
| Trees | 57.84 | 58.56 | -0.72 |
| Vehicles 1 | 72.20 | 69.76 | 2.44 |
| Vehicles 2 | 81.08 | 77.80 | 3.28 |
| EuroSAT | 73.02 | 71.52 | 1.50 |
| KITTI | 64.11 | 63.17 | 0.94 |
| Mean | — | — | 2.70 |

Analisando os dados da Tabela A.4, temos que o modelo podado apresenta um desempenho inferior ao modelo original na maioria das tarefas, acarretando em um score de +2.70 p.p. Dado que este valor é positivo, **podemos concluir neste experimento que um modelo com poda estruturada que mantém** 68% **dos parâmetros originais sofre uma perda de acurácia de** 2.70 **p.p. em média**.

Uma possível explicação para essa piora do modelo podado de forma estruturada é que a redução no número de filtros pode prejudicar a generalização para novas tarefas. No caso da poda não estruturada a quantidade de filtros permanece e mesmo deletando certos parâmetros de um filtro ainda é possível extrair características das imagens relevantes para a sua correta classificação.

## A.7 PRUNE2TRANSFER — Poda não estruturada por aprendizado por reforço

### A.7.1 Etapa 2 — Poda

Os hiperparâmetros utilizados para realizar a poda da CNN-simples estão descritos na Tabela A.5. Tendo em vista que um episódio nesta tarefa consiste em percorrer todas as camadas da rede neural a ser podada, os valores dos passos foram ajustados para a arquitetura da rede em questão, no caso uma rede com 6 camadas. A arquitetura da rede neural usada como aproximador da função $Q$ possui duas camadas escondidas com 200 unidades cada.

**Tabela A.5:** *Hiperâmetros da DQN utilizada na poda não estruturada da CNN-simples.*

| Hyperparameter | Value |
|---|---|
| Total timesteps | 4000 |
| Learning rate | 0.001 |
| Buffer size | 500 |
| Exploration fraction | 0.7 |
| Exploration final episode | 0.02 |
| Batch size | 32 |
| Learning starts | 400 |
| Gamma | 0.982 |
| Target network update frequency | 10 |

Os valores definidos como meta para a acurácia e esparsidade foram 80% e 70%, respectivamente. Portanto, o objetivo desta tarefa de RL é minimizar a função de custo, dado que a recompensa é negativa, mantendo a acurácia no patamar obtido pelo modelo sem poda e induzindo uma esparsidade de pelo menos 70% no modelo. A Figura A.6 ilustra as recompensas obtidas a cada episódio do treinamento, ou seja, a soma das recompensas obtidas após podar cada camada.

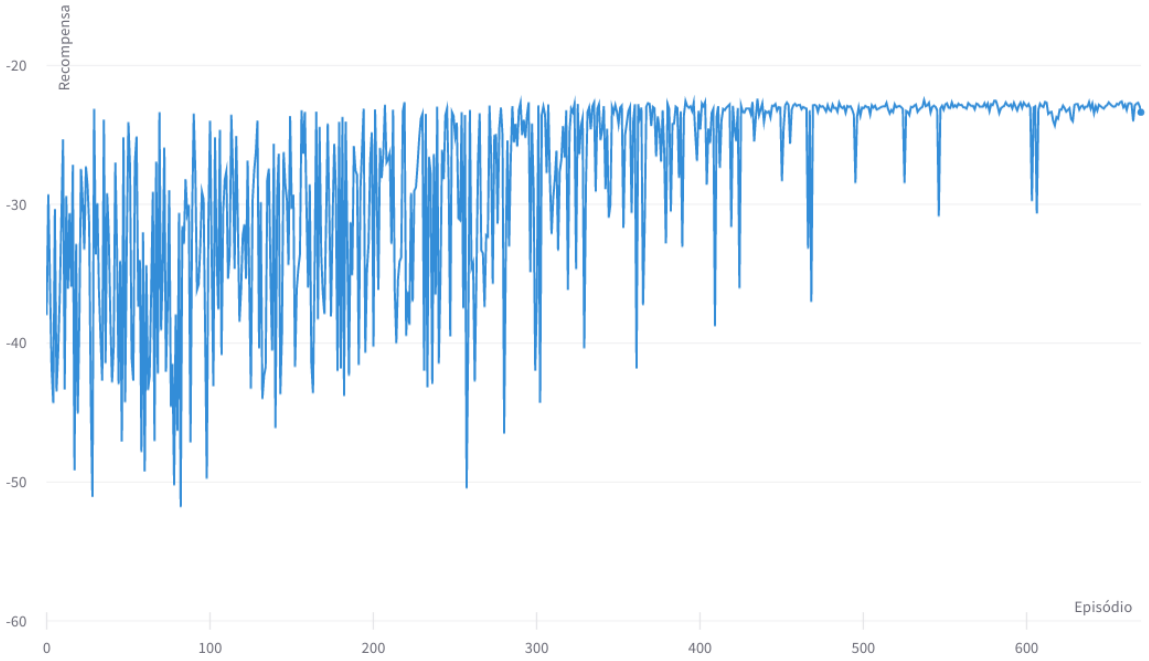



**Figura A.6:** *PuRL CNN-simples: evolução da recompensa obtida a cada episódio do treinamento.*

Analisando a Figura A.6, é possível notar que a partir de 500 episódios o aprendizado se estabiliza, atingindo um custo mínimo de aproximadamente −23. A política de poda aprendida neste treinamento foi {0.2, 0.4, 0.4, 0.4, 1.6 e 0.0} seguindo a ordem da esquerda para direita das camadas da Figura A.1. Aqui o valor da ação 0.0 na última camada indica que esta não será podada. **Aplicando esta política de poda e retreinando o modelo uma última vez, foi possível alcançar uma acurácia de 78.8% mantendo 13.10% dos parâmetros originais**.

### A.7.2 Etapa 3 — Transferência de Aprendizado

Visto que o PuRL também utiliza um método não-estruturado para realizar a poda, utilizaremos as mesmas configurações experimentais da etapa de transferência de aprendizado da Poda Não-Estruturada. A Tabela A.6 relata as acurácias do modelo sem poda ($\text{Acc}_{\text{original}}$), do modelo podado a partir da política aprendida ($\text{Acc}_{13.1\%}$) e a diferença entre essas acurácias.

**Tabela A.6:** *Acurácias nas tarefas meta do modelo sem poda da CNN-simples e do modelo podado de forma não estruturada por uma política.*

| Target Task | $\text{Acc}_{\text{original}}$(%) | $\text{Acc}_{\text{podado}}$(%) | $\text{Acc}_{\text{original}}$-$\text{Acc}_{\text{podado}}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 54.12 | 54.36 | -0.24 |
| Fish | 69.20 | 69.16 | 0.04 |
| Flowers | 64.76 | 66.88 | -2.04 |
| Food Containers | 72.04 | 71.40 | 0.64 |
| Fruit and Vegetables | 72.68 | 74.16 | -1.48 |
| Household Electrical Device | 70.28 | 69.52 | 0.76 |
| Household Furniture | 65.04 | 65.20 | -0.16 |
| Insects | 65.28 | 65.00 | 0.28 |
| Large Carnivores | 63.56 | 63.08 | 0.48 |
| Large Man-made Outdoor Things | 70.84 | 70.36 | 0.48 |
| Large Natural Outdoor Scenes | 78.60 | 78.00 | 0.60 |
| Large Omnivores and Herbivores | 62.96 | 63.44 | -0.48 |
| Medium-sized Mammals | 63.52 | 63.60 | -0.08 |
| Non-insect Invertebrates | 64.60 | 66.20 | -1.60 |
| People | 34.88 | 37.56 | -2.68 |
| Reptiles | 59.32 | 61.60 | -2.28 |
| Small Mammals | 48.24 | 47.00 | 1.24 |
| Trees | 57.84 | 59.00 | -1.16 |
| Vehicles 1 | 72.20 | 73.88 | -1.68 |
| Vehicles 2 | 81.08 | 80.80 | 0.28 |
| EuroSAT | 73.02 | 73.53 | -0.51 |
| KITTI | 64.11 | 63.83 | 0.28 |
| Mean | — | — | -0.42 |

Analisando os dados da Tabela A.6, temos que o modelo podado apresenta um desempenho superior ao modelo original na maioria das 22 tarefas, acarretando em um score de −0.42 p.p. E como este é um valor negativo, **podemos concluir neste experimento que um modelo que mantém** 13.1% **dos parâmetros originais pode ser adaptado à novas tarefas sem perda de generalidade**.

## A.8 PRUNE2TRANSFER — Poda estruturada por aprendizado por reforço

### A.8.1 Etapa 2 — Poda

Os hiperparâmetros utilizados para realizar a poda da CNN-simples estão descritos na Tabela A.7. Tendo em vista que um episódio nesta tarefa consiste em percorrer todas as camadas da rede neural a ser podada, os valores dos passos foram ajustados para a arquitetura de uma rede com 4 camadas convolucionais. A arquitetura da rede neural usada como aproximador da função $Q$ possui duas camadas escondidas com 200 unidades

cada.

**Tabela A.7:** *Hiperâmetros da DQN utilizada na poda estruturada da CNN-simples.*

| Hyperparameter | Value |
|---|---|
| Total timesteps | 2000 |
| Learning rate | 0.001 |
| Buffer size | 300 |
| Exploration fraction | 0.7 |
| Exploration final episode | 0.02 |
| Batch size | 32 |
| Learning starts | 100 |
| Gamma | 0.982 |
| Target network update frequency | 15 |

Os valores definidos como meta para a acurácia e esparsidade foram 80% e 40%, respectivamente. Portanto, o objetivo desta tarefa de RL é minimizar a função de custo mantendo a acurácia no patamar obtido pelo modelo sem poda e induzindo uma esparsidade de pelo menos 40% no modelo. A Figura A.7 ilustra as recompensas obtidas a cada episódio do treinamento, ou seja, a soma das recompensas obtidas após podar cada camada de forma estruturada.

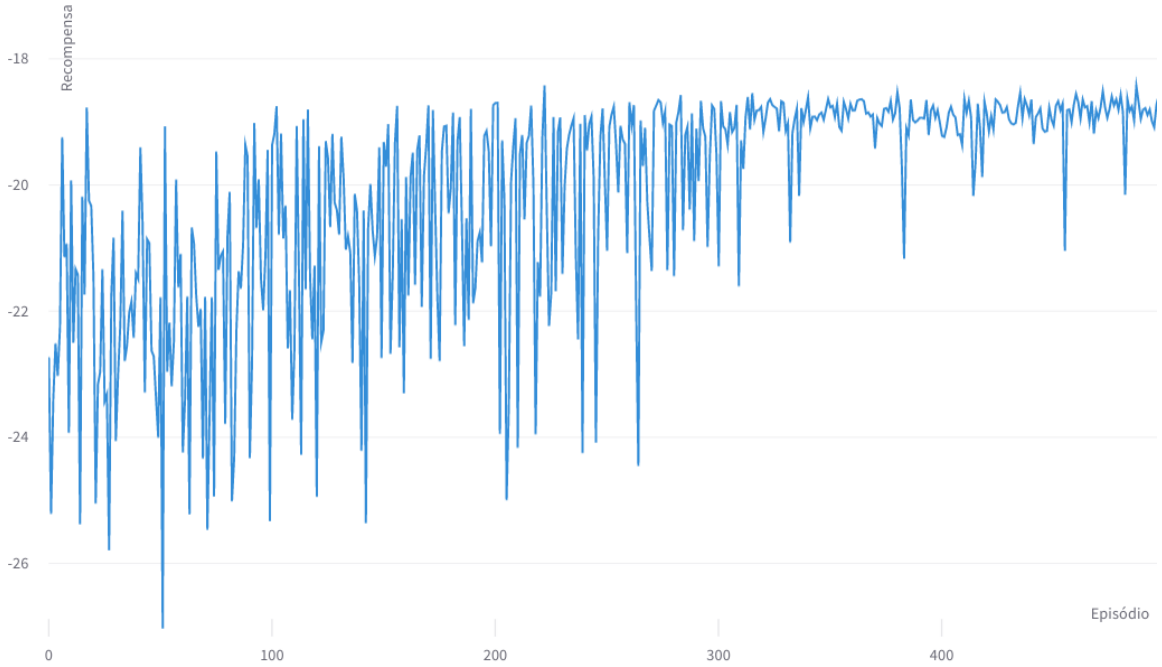



**Figura A.7:** *StructPuRL CNN-simples: evolução da recompensa obtida a cada episódio do treinamento.*

Analisando a Figura A.7, é possível notar que a partir de 350 episódios o aprendizado se estabiliza, atingindo um custo mínimo de aproximadamente −19. A política de poda aprendida neste treinamento para cada camada convolucional foi $\{1.0, 1.0, 1.0, 0.5\}$ seguindo a

ordem da esquerda para direita das camadas ilustradas na Figura A.1. Nesta política as três primeiras camadas convolucionais não serão podadas e a última convolucional sofrerá uma redução de 50% de seus filtros. **Aplicando esta política de poda e retreinando o modelo uma última vez, foi possível alcançar uma acurácia de 78.75% com uma redução de 49% do modelo original**. A Tabela A.8 traz uma comparação do modelo original e deste modelo resultante da poda estruturada.

**Tabela A.8:** *Comparação entre o modelo original da CNN-simples e o modelo podado com redução de 49% dos parâmetros.*

| Model | Total Parameters | Accuracy (%) | Memory footprint (MB) |
|---|---|---|---|
| Unpruned | 1 084 234 | 80.32% | 4.17 |
| Pruned | 550 698 | 78.75% | 2.14 |

Analisando a Tabela A.8, temos que o o modelo podado apresenta uma pequena perda em acurácia, mas em compensação possui um tamanho em disco menor que o modelo original. Em comparação ao modelo podado de forma estruturada sem uma política (Seção A.6), o modelo gerado pelo StructPuRL apresenta uma compressão maior de 1.46× para 1.95× atingindo praticamente a mesma acurácia.

### A.8.2 Etapa 3 — Transferência de Aprendizado

Visto que o modelo final podado é similar ao modelo da Seção A.6, pois ambos utilizaram um critério de poda estruturada na CNN-simples, utilizaremos as mesmas configurações experimentais da sua etapa de *transfer learning*.

**Tabela A.9:** *Acurácias nas tarefas meta do modelo sem poda da CNN-simples e do modelo podado de forma estruturada por uma política.*

| Target Task | $Acc_{original}$(%) | $Acc_{podado}$(%) | $Acc_{original}$-$Acc_{podado}$(p.p.) |
|---|---|---|---|
| Aquatic Mammals | 54.12 | 52.36 | 1.76 |
| Fish | 69.2 | 67.0 | 2.2 |
| Flowers | 64.76 | 61.16 | 3.6 |
| Food Containers | 72.04 | 70.8 | 1.24 |
| Fruit and Vegetables | 72.68 | 72.76 | -0.08 |
| Household Electrical Device | 70.28 | 67.24 | 3.04 |
| Household Furniture | 65.04 | 65.32 | -0.28 |
| Insects | 65.28 | 64.08 | 1.2 |
| Large Carnivores | 63.56 | 65.0 | -1.44 |
| Large Man-made Outdoor Things | 70.84 | 70.0 | 0.84 |
| Large Natural Outdoor Scenes | 78.6 | 75.32 | 3.28 |
| Large Omnivores and Herbivores | 62.96 | 59.48 | 3.48 |
| Medium-sized Mammals | 63.52 | 58.2 | 5.32 |
| Non-insect Invertebrates | 64.6 | 61.92 | 2.68 |
| People | 34.88 | 38.04 | -3.16 |
| Reptiles | 59.32 | 53.32 | 6.0 |
| Small Mammals | 48.24 | 53.08 | -4.84 |
| Trees | 57.84 | 59.52 | -1.68 |
| Vehicles 1 | 72.2 | 71.36 | 0.84 |
| Vehicles 2 | 81.08 | 80.6 | 0.48 |
| EuroSAT | 73.02 | 73.39 | -0.37 |
| KITTI | 64.11 | 62.14 | 1.97 |
| Mean | — | — | 1.18 |

Analisando os dados da Tabela A.9, temos que o modelo podado apresenta um desempenho inferior ao modelo original na maioria das tarefas, acarretando em um score de +1.18 p.p. Dado que este valor é positivo, **podemos concluir neste experimento que um modelo com uma poda estruturada aprendida por um agente que mantém** 51% **dos parâmetros originais sofre uma perda de acurácia de** 1.18 **p.p. em média**. Este resultado é melhor do que o da poda estruturada clássica, diferença média de 2.70 p.p., neste mesmo conjunto de tarefas meta.

## A.9 Discussão dos Resultados

A Tabela A.10 apresenta os valores das quatro métricas avaliadas para os métodos de poda aplicados à CNN-simples.

Analisando a tabela, os métodos de poda não estruturada atingem níveis maiores de poda do que métodos estruturados (M2), com prejuízo de menos de 2% na acurácia da tarefa original (M1). Isto pode ser atribuído ao critério de poda estruturada adotado que é destinado a camadas convolucionais, portanto não realiza a poda das camadas densas. E

**Tabela A.10:** *Comparação de todos os critérios testados.*

| Tipo de Poda | Critério | M1 | M2 | M3 | M4 |
|---|---|---|---|---|---|
| Não-Estruturada | Teste-t | 0.67 | 94.24% | 1× | −0.60 |
| | PuRL | 1.52 | 86.9% | 1× | −0.42 |
| Estruturada | Norma-$L_1$ | 1.35 | 32% | 1.47× | 2.70 |
| | StructPuRL | 1.57 | 49% | 1.95× | 1.18 |

também pode ser atribuído à própria natureza não estruturada dos critérios, que permite uma poda mais granular do que os métodos estruturados.

Além de atingirem uma maior esparsidade, os métodos Não Estruturado e PuRL obtiveram as melhores adaptações às tarefas meta (M4), conseguindo em média alcançar uma acurácia melhor que o modelo sem poda. Apesar dos resultados positivos, estes dois critérios não favorecem a compactação do modelo no armazenamento em disco (M3), pois a remoção dos parâmetros é realizada pela substituição do peso original por zero. Logo, a arquitetura da rede permanece a mesma porém com esparsidade em sua estrutura de pesos.

A vantagem dos métodos Estruturado e StructPuRL está no critério M3, pois os métodos estruturados levam a mudanças na arquitetura da rede que impactam diretamente no tamanho do modelo em memória e consequentemente no seu tempo de inferência. Dentre estes dois métodos, o StructPuRL foi capaz de atingir uma maior esparsidade na rede e alcançou um resultado melhor na etapa de *transfer learning*, com uma redução média na acurácia original de 1.18 p.p. contra 2.70 p.p. do método Estruturado.

# Referências

[Abadi *et al.* 2015] Martin Abadi *et al. TensorFlow: Large-scale machine learning on heterogeneous systems*. 2015 (citado na pg. 45).

[Abu-Mostafa *et al.* 2012] Yaser S Abu-Mostafa, Malik Magdon-Ismail e Hsuan-Tien Lin. *Learning from data*. Vol. 4. AMLBook New York, NY, USA: 2012 (citado na pg. 11).

[Anwar *et al.* 2015] Sajid Anwar, Kyuyeon Hwang e Wonyong Sung. "Structured pruning of deep convolutional neural networks". Em: *CoRR* abs/1512.08571 (2015). arXiv: 1512.08571. URL: http://arxiv.org/abs/1512.08571 (citado na pg. 24).

[Arulampalam *et al.* 2002] M. Sanjeev Arulampalam, Simon Maskell, Neil J. Gordon e Tim Clapp. "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking". Em: *IEEE Trans. Signal Process.* 50.2 (2002), pgs. 174–188. DOI: 10.1109/78.978374. URL: https://doi.org/10.1109/78.978374 (citado na pg. 24).

[Barto *et al.* 1989] Andrew G Barto, Richard S Sutton e CJC Watkins. "Sequential decision problems and neural networks". Em: *Advances in neural information processing systems* 2 (1989), pgs. 686–693 (citado na pg. 9).

[Bartoldson *et al.* 2018] Brian Bartoldson, Adrian Barbu e Gordon Erlebacher. "Enhancing the regularization effect of weight pruning in artificial neural networks". Em: *CoRR* abs/1805.01930 (2018). arXiv: 1805.01930. URL: http://arxiv.org/abs/1805.01930 (citado na pg. 62).

[Benavoli *et al.* 2017] Alessio Benavoli, Giorgio Corani, Janez Demšar e Marco Zaffalon. "Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis". Em: *The Journal of Machine Learning Research* 18.1 (2017), pgs. 2653–2688 (citado na pg. 32).

[Bengio *et al.* 2013] Yoshua Bengio, Aaron Courville e Pascal Vincent. "Representation learning: a review and new perspectives". Em: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pgs. 1798–1828 (citado na pg. 43).

[Bizopoulos *et al.* 2020] Paschalis Bizopoulos, Nicholas Vretos e Petros Daras. "Comprehensive comparison of deep learning models for lung and COVID-19 lesion segmentation in CT scans". Em: *CoRR* abs/2009.06412 (2020). arXiv: 2009.06412. URL: https://arxiv.org/abs/2009.06412 (citado na pg. 45).

[Blalock *et al.* 2020] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle e John Guttag. "What is the state of neural network pruning?" Em: *arXiv preprint arXiv:2003.03033* (2020) (citado na pg. 20).

[Bojarski *et al.* 2016] Mariusz Bojarski *et al.* "End to end learning for self-driving cars". Em: *CoRR* abs/1604.07316 (2016). arXiv: 1604.07316. URL: http://arxiv.org/abs/1604.07316 (citado na pg. 1).

[Cheng *et al.* 2017] Yu Cheng, Duo Wang, Pan Zhou e Tao Zhang. "A survey of model compression and acceleration for deep neural networks". Em: *arXiv preprint arXiv:1710.09282* (2017) (citado na pg. 19).

[Cohen e Welling 2016] Taco Cohen e Max Welling. "Group equivariant convolutional networks". Em: *International conference on machine learning*. 2016, pgs. 2990–2999 (citado na pg. 19).

[Corani e Benavoli 2015] Giorgio Corani e Alessio Benavoli. "A bayesian approach for comparing cross-validated algorithms on multiple data sets". Em: *Machine Learning* 100.2 (2015), pgs. 285–304 (citado na pg. 32).

[Deepak e Ameer 2019] S. Deepak e P. M. Ameer. "Brain tumor classification using deep CNN features via transfer learning". Em: *Comput. Biol. Medicine* 111 (2019). DOI: 10.1016/j.compbiomed.2019.103345. URL: https://doi.org/10.1016/j.compbiomed.2019.103345 (citado na pg. 3).

[Deng *et al.* 2009] Jia Deng *et al.* "Imagenet: a large-scale hierarchical image database". Em: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pgs. 248–255 (citado na pg. 4).

[Denil *et al.* 2013] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato e Nando De Freitas. "Predicting parameters in deep learning". Em: *Advances in neural information processing systems*. 2013, pgs. 2148–2156 (citado na pg. 19).

[Dhariwal *et al.* 2017] Prafulla Dhariwal *et al. OpenAI Baselines*. https://github.com/openai/baselines. 2017 (citado na pg. 56).

[Dong *et al.* 2017] Xin Dong, Shangyu Chen e Sinno Jialin Pan. "Learning to prune deep neural networks via layer-wise optimal brain surgeon". Em: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. por Isabelle Guyon *et al.* 2017, pgs. 4857–4867. URL: https://proceedings.neurips.cc/paper/2017/hash/c5dc3e08849bec07e33ca353de62ea04-Abstract.html (citado na pg. 27).

[Dumoulin e Visin 2016] Vincent Dumoulin e Francesco Visin. "A guide to convolution arithmetic for deep learning". Em: *CoRR* abs/1603.07285 (2016). arXiv: 1603.07285. URL: http://arxiv.org/abs/1603.07285 (citado na pg. 15).

[Frankle e Carbin 2019] Jonathan Frankle e Michael Carbin. "The lottery ticket hypothesis: finding sparse, trainable neural networks". Em: (2019). URL: https://openreview.net/forum?id=rJl-b3RcF7 (citado nas pgs. 4, 22, 75).

[Garey e Johnson 1979] Michael R Garey e David S Johnson. "Computers and intractability". Em: *A Guide to the* (1979) (citado na pg. 26).

[Gatys *et al.* 2015] Leon A. Gatys, Alexander S. Ecker e Matthias Bethge. "A neural algorithm of artistic style". Em: *CoRR* abs/1508.06576 (2015). arXiv: 1508.06576. URL: http://arxiv.org/abs/1508.06576 (citado na pg. 45).

[Geiger *et al.* 2013] Andreas Geiger, Philip Lenz, Christoph Stiller e Raquel Urtasun. "Vision meets robotics: the kitti dataset". Em: *The International Journal of Robotics Research* 32.11 (2013), pgs. 1231–1237 (citado na pg. 47).

[Goodfellow *et al.* 2016] Ian J. Goodfellow, Yoshua Bengio e Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN: 978-0-262-03561-3. URL: http://www.deeplearningbook.org/ (citado nas pgs. 1, 7, 8, 11, 13).

[M. Gupta *et al.* 2020] Manas Gupta, Siddharth Aravindan, Aleksandra Kalisz, Vijay Chandrasekhar e Lin Jie. "Learning to prune deep neural networks via reinforcement learning". Em: *CoRR* abs/2007.04756 (2020). arXiv: 2007.04756. URL: https://arxiv.org/abs/2007.04756 (citado nas pgs. 2, 4, 27–29).

[Han e Dally 2017] Song Han e B Dally. "Efficient methods and hardware for deep learning". Em: *University Lecture* (2017) (citado nas pgs. 21, 22, 31, 42, 62).

[Han, Pool *et al.* 2015] Song Han, Jeff Pool, John Tran e William J. Dally. "Learning both weights and connections for efficient neural network". Em: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. por Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama e Roman Garnett. 2015, pgs. 1135–1143. URL: https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html (citado nas pgs. 2, 22, 27).

[Hassibi e Stork 1993] Babak Hassibi e David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993 (citado na pg. 21).

[K. He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren e Jian Sun. "Deep residual learning for image recognition". Em: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pgs. 770–778. DOI: 10.1109/CVPR.2016.90. URL: https://doi.org/10.1109/CVPR.2016.90 (citado na pg. 66).

[Y. He *et al.* 2018a] Yihui He *et al.* "AMC: automl for model compression and acceleration on mobile devices". Em: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*. Ed. por Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu e Yair Weiss. Vol. 11211. Lecture Notes in Computer Science. Springer, 2018, pgs. 815–832. DOI: 10.1007/978-3-030-01234-2_48. URL: https://doi.org/10.1007/978-3-030-01234-2%5C_48 (citado nas pgs. 27, 36).

[Y. He *et al.* 2018b] Yihui He *et al.* "AMC: automl for model compression and acceleration on mobile devices". Em: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*. Ed. por Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu e Yair Weiss. Vol. 11211. Lecture Notes in Computer Science. Springer, 2018, pgs. 815–832. DOI: 10.1007/978-3-030-01234-2_48. URL: https://doi.org/10.1007/978-3-030-01234-2%5C_48 (citado nas pgs. 27, 28).

[Helber *et al.* 2019] Patrick Helber, Benjamin Bischke, Andreas Dengel e Damian Borth. "Eurosat: a novel dataset and deep learning benchmark for land use and land cover classification". Em: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 12.7 (2019), pgs. 2217–2226 (citado na pg. 47).

[Hinton, Srivastava *et al.* 2012] Geoffrey Hinton, Nitish Srivastava e Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". Em: *Cited on* 14.8 (2012) (citado na pg. 72).

[Hinton, Vinyals *et al.* 2015] Geoffrey Hinton, Oriol Vinyals e Jeff Dean. "Distilling the knowledge in a neural network". Em: *arXiv preprint arXiv:1503.02531* (2015) (citado na pg. 19).

[Hornik *et al.* 1989] Kurt Hornik, Maxwell Stinchcombe e Halbert White. "Multilayer feedforward networks are universal approximators". Em: *Neural networks* 2.5 (1989), pgs. 359–366 (citado na pg. 10).

[Howard *et al.* 2017] Andrew G. Howard *et al.* "Mobilenets: efficient convolutional neural networks for mobile vision applications". Em: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861 (citado na pg. 66).

[Huang *et al.* 2017] Gao Huang, Zhuang Liu, Laurens van der Maaten e Kilian Q. Weinberger. "Densely connected convolutional networks". Em: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pgs. 2261–2269. DOI: 10.1109/CVPR.2017.243. URL: https://doi.org/10.1109/CVPR.2017.243 (citado na pg. 66).

[Irhum Shafkat 2018] Irhum Shafkat. *Intuitively Understanding Convolutions for Deep Learning*. https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1, Acessado em: 2021-05-10. 2018 (citado na pg. 14).

[Jha *et al.* 2020] Debesh Jha, Michael A. Riegler, Dag Johansen, Pal Halvorsen e Havard D. Johansen. "Doubleu-net: A deep convolutional neural network for medical image segmentation". Em: *33rd IEEE International Symposium on Computer-Based Medical Systems, CBMS 2020, Rochester, MN, USA, July 28-30, 2020*. Ed. por Alba Garcia Seco de Herrera *et al.* IEEE, 2020, pgs. 558–564. doi: 10.1109/CBMS49503.2020.00111. url: https://doi.org/10.1109/CBMS49503.2020.00111 (citado na pg. 45).

[Katinskaia 2019] Anisia Katinskaia. *CNN on CIFAR-10*. https://colab.research.google.com/github/Askinkaty/IntroDL/blob/master/CNN_on_CIFAR_10.ipynb. Accessed: 2019-09-01. 2019 (citado na pg. 45).

[Kingma e Ba 2015] Diederik P. Kingma e Jimmy Ba. "Adam: A method for stochastic optimization". Em: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. por Yoshua Bengio e Yann LeCun. 2015. url: http://arxiv.org/abs/1412.6980 (citado na pg. 70).

[Krizhevsky, Hinton *et al.* 2009] Alex Krizhevsky, Geoffrey Hinton *et al.* "Learning multiple layers of features from tiny images". Em: (2009) (citado nas pgs. 47, 70).

[LeCun, Bengio *et al.* 2015] Yann LeCun, Yoshua Bengio e Geoffrey E. Hinton. "Deep learning". Em: *Nat.* 521.7553 (2015), pgs. 436–444. doi: 10.1038/nature14539. url: https://doi.org/10.1038/nature14539 (citado na pg. 1).

[LeCun, Boser *et al.* 1989] Yann LeCun, Bernhard E. Boser *et al.* "Handwritten digit recognition with a back-propagation network". Em: *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*. Ed. por David S. Touretzky. Morgan Kaufmann, 1989, pgs. 396–404. url: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network (citado nas pgs. 14, 15).

[LeCun, Denker *et al.* 1989] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard e Lawrence D Jackel. "Optimal brain damage." Em: *NIPs*. Vol. 2. Citeseer. 1989, pgs. 598–605 (citado nas pgs. 2, 20, 21).

[Li *et al.* 2017] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet e Hans Peter Graf. "Pruning filters for efficient convnets". Em: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. url: https://openreview.net/forum?id=rJqFGTslg (citado nas pgs. 2, 24–27, 34, 78).

[J. Liu *et al.* 2017] Jiaming Liu, Yali Wang e Yu Qiao. "Sparse deep transfer learning for convolutional neural network". Em: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017 (citado na pg. 3).

[Z. Liu *et al.* 2017] Zhuang Liu *et al.* "Learning efficient convolutional networks through network slimming". Em: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pgs. 2736–2744 (citado na pg. 25).

[Loey *et al.* 2021] Mohamed Loey, Gunasekaran Manogaran, Mohamed Hamed N. Taha e Nour Eldeen M. Khalifa. "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic". Em: *Measurement* 167 (2021), pg. 108288. issn: 0263-2241. doi: https://doi.org/10.1016/j.measurement.2020.108288. url: https://www.sciencedirect.com/science/article/pii/S0263224120308289 (citado na pg. 3).

[McCulloch e Pitts 1943] Warren S McCulloch e Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". Em: *The bulletin of mathematical biophysics* 5.4 (1943), pgs. 115–133 (citado na pg. 10).

[Mitchell 1997] Thomas M. Mitchell. *Machine Learning*. 1ª ed. New York, NY, USA: McGraw-Hill, Inc., 1997. isbn: 0070428077, 9780070428072 (citado na pg. 7).

[Mnih *et al.* 2015] Volodymyr Mnih *et al.* "Human-level control through deep reinforcement learning". Em: *Nat.* 518.7540 (2015), pgs. 529–533. doi: 10.1038/nature14236. url: https://doi.org/10.1038/nature14236 (citado nas pgs. 29, 56).

[Molchanov *et al.* 2016] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila e Jan Kautz. "Pruning convolutional neural networks for resource efficient inference". Em: *arXiv preprint arXiv:1611.06440* (2016) (citado na pg. 3).

[Morcos *et al.* 2019] Ari S. Morcos, Haonan Yu, Michela Paganini e Yuandong Tian. "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers". Em: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. por Hanna M. Wallach *et al.* 2019, pgs. 4933–4943. url: https://proceedings.neurips.cc/paper/2019/hash/a4613e8d72a61b3b69b32d040f89ad81-Abstract.html (citado na pg. 4).

[NVIDIA 2019] NVIDIA. *Transfer Learning Toolkit*. https://developer.nvidia.com/transfer-learning-toolkit. Accessed: 2021-06-28. 2019 (citado na pg. 2).

[Pan 2014] Sinno Jialin Pan. "Transfer learning". Em: *Data Classification: Algorithms and Applications*. Ed. por Charu C. Aggarwal. CRC Press, 2014. Cap. 21. isbn: 978-1-4665-8674-1. url: http://www.crcnetbase.com/doi/book/10.1201/b17320 (citado na pg. 15).

[Pan e Yang 2010] Sinno Jialin Pan e Qiang Yang. "A survey on transfer learning". Em: *IEEE Trans. Knowl. Data Eng.* 22.10 (2010), pgs. 1345–1359. doi: 10.1109/TKDE.2009.191. url: https://doi.org/10.1109/TKDE.2009.191 (citado nas pgs. 15, 16).

[Prajapati e Stamp 2021] Pratikkumar Prajapati e Mark Stamp. "An empirical analysis of image-based learning techniques for malware classification". Em: *CoRR* abs/2103.13827 (2021). arXiv: 2103.13827. url: https://arxiv.org/abs/2103.13827 (citado na pg. 45).

[Puterman 1995] Martin L Puterman. "Markov decision processes: discrete stochastic dynamic programming". Em: *Journal of the Operational Research Society* 46.6 (1995), pgs. 792–792 (citado nas pgs. 9, 27).

[Qian 1999] Ning Qian. "On the momentum term in gradient descent learning algorithms". Em: *Neural networks* 12.1 (1999), pgs. 145–151 (citado na pg. 48).

[Rosenblatt 1958] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." Em: *Psychological review* 65.6 (1958), pg. 386 (citado na pg. 10).

[Russakovsky *et al.* 2015] Olga Russakovsky *et al.* "ImageNet Large Scale Visual Recognition Challenge". Em: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pgs. 211–252. doi: 10.1007/s11263-015-0816-y (citado na pg. 47).

[Russell e Norvig 2016] Stuart J Russell e Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016 (citado nas pgs. 8, 10, 11).

[Sainath *et al.* 2013] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy e Bhuvana Ramabhadran. "Low-rank matrix factorization for deep neural network training with high-dimensional output targets". Em: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pgs. 6655–6659 (citado na pg. 19).

[Silver *et al.* 2017] David Silver *et al.* "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". Em: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. url: http://arxiv.org/abs/1712.01815 (citado na pg. 27).

[Simonyan e Zisserman 2015] Karen Simonyan e Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". Em: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. por Yoshua Bengio e Yann LeCun. 2015. url: http://arxiv.org/abs/1409.1556 (citado na pg. 45).

[Soelen e Sheppard 2019] Ryan Van Soelen e John W. Sheppard. "Using winning lottery tickets in transfer learning for convolutional neural networks". Em: *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*. IEEE, 2019, pgs. 1–8. doi: 10.1109/IJCNN.2019.8852405. url: https://doi.org/10.1109/IJCNN.2019.8852405 (citado na pg. 4).

[Srivastava *et al.* 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever e Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". Em: *The journal of machine learning research* 15.1 (2014), pgs. 1929–1958 (citado na pg. 20).

[Stopforth e Moodley 2019] Julius Stopforth e Deshendran Moodley. "Continuous versus discrete action spaces for deep reinforcement learning". Em: *Proc. of the South African Forum for Artificial Intelligence Research FAIR*. Cape Town, South Africa, dez. de 2019. url: http://ceur-ws.org/Vol-2540/FAIR2019_paper_47.pdf (citado na pg. 36).

[Sutton e Barto 2018] Richard S Sutton e Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (citado nas pgs. x, 8–10).

[Tan *et al.* 2018] Chuanqi Tan *et al.* "A survey on deep transfer learning". Em: *Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III*. Ed. por Vera Kurková, Yannis Manolopoulos, Barbara Hammer, Lazaros S. Iliadis e Ilias Maglogiannis. Vol. 11141. Lecture Notes in Computer Science. Springer, 2018, pgs. 270–279. doi: 10.1007/978-3-030-01424-7_27. url: https://doi.org/10.1007/978-3-030-01424-7%5C_27 (citado nas pgs. 15, 16).

[Tzeng *et al.* 2017] Eric Tzeng, Judy Hoffman, Kate Saenko e Trevor Darrell. "Adversarial discriminative domain adaptation". Em: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pgs. 2962–2971. doi: 10.1109/CVPR.2017.316. url: https://doi.org/10.1109/CVPR.2017.316 (citado na pg. 16).

[Verhelst e Moons 2017] Marian Verhelst e Bert Moons. "Embedded deep neural network processing: algorithmic and processor techniques bring deep learning to iot and edge devices". Em: *IEEE Solid-State Circuits Magazine* 9.4 (2017), pgs. 55–65 (citado na pg. 19).

[Vogado *et al.* 2018] Luis H. S. Vogado, Rodrigo M. S. Veras, Flávio H. D. Araújo, Romuere Rôdrigues Veloso e Silva e Kelson Rômulo Teixeira Aires. "Leukemia diagnosis in blood slides using transfer learning in cnns and SVM for classification". Em: *Eng. Appl. Artif. Intell.* 72 (2018), pgs. 415–422. doi: 10.1016/j.engappai.2018.04.024. url: https://doi.org/10.1016/j.engappai.2018.04.024 (citado na pg. 3).

[Watkins 1989] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". Em: (1989) (citado nas pgs. 9, 56).

[Yosinski *et al.* 2014] Jason Yosinski, Jeff Clune, Yoshua Bengio e Hod Lipson. "How transferable are features in deep neural networks?" Em: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. por Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence e Kilian Q. Weinberger. 2014, pgs. 3320–3328. URL: https://proceedings.neurips.cc/paper/2014/hash/375c71349b295fbe2dcdca9206f20a06-Abstract.html (citado nas pgs. 2, 15, 50).

[Zhai *et al.* 2019] Xiaohua Zhai *et al.* "A large-scale study of representation learning with the visual task adaptation benchmark". Em: *arXiv preprint arXiv:1910.04867* (2019) (citado nas pgs. 43, 47).

[Zhu e S. Gupta 2018] Michael Zhu e Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression". Em: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=Sy1iIDkPM (citado na pg. 3).