UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Classificação de Imagens de Resíduos Eletrônicos Usando Redes Neurais Convolucionais

José Alexandre Ferreira da Silva

Dissertação de Mestrado do Programa de Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria (MECAI)

ICMC USP SÃO CARLOS

José Alexandre Ferreira da Silva

Classificação de Imagens de Resíduos Eletrônicos Usando Redes Neurais Convolucionais

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria. *VERSÃO REVISADA*

Área de Concentração: Matemática, Estatística e Computação

Orientador: Prof. Dr. Francisco Louzada Neto

USP – São Carlos
Julho de 2023

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

S586c

Silva, Jose Alexandre Ferreira da
Classificação de Imagens de Resíduos Eletrônicos Usando Redes Neurais Convolucionais / Jose Alexandre Ferreira da Silva; orientador Francisco Louzada Neto. -- São Carlos, 2023.
97 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. Redes Neurais Convolucionais. 2. Classificação de Imagens. 3. Resíduos Eletrônicos. I. Neto, Francisco Louzada, orient. II. Título.

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

José Alexandre Ferreira da Silva

E-Waste Images Classification Using Convolutional Neural Networks

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Professional Master's Program in Mathematics Statistics and Computing Applied to Industry, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Mathematics, Statistics and Computing

Advisor: Prof. Dr. Francisco Louzada Neto

USP – São Carlos
July 2023

Este trabalho é dedicado à memória de John Forbes Nash Jr (1928 - 2015)

AGRADECIMENTOS

Agradeço à minha família pelo apoio e compreensão.

Agradeço ao Professor Francisco Louzada pela oportunidade, pela orientação e por todos os ensinamentos durante este tempo. Também agradeço aos professores de todas as disciplinas do MECAI, que fizeram um excelente trabalho em tempos desafiadores de pandemia.

Agradeço aos colegas pela parceria durante as diversas atividades desenvolvidas ao longo do curso.

“Eu quero colocar uma marca no universo.”
(Steve Jobs)

RESUMO

SILVA, J. A. **Classificação de Imagens de Resíduos Eletrônicos Usando Redes Neurais Convolucionais**. 2023. 97 p. Dissertação (Mestrado – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

O descarte ambientalmente correto de equipamentos elétricos e eletrônicos em fim de vida útil é um assunto de extrema relevância na sociedade atual, em que o constante surgimento de novos dispositivos incentiva seu consumo desenfreado, o que acaba por resultar na geração de enormes volumes de resíduos (*e-waste*). Logo, o desenvolvimento e a aplicação de novas técnicas para a otimização do processo de reciclagem consiste em fator preponderante para a expansão de uma Gestão Verde da Cadeia de Suprimentos. Neste sentido, alguns trabalhos têm sido propostos na área de classificação de imagens de resíduos, mas poucos abordam especificamente aqueles oriundos de equipamentos elétricos e eletrônicos, cujas especificidades devem ser tratadas de maneira apropriada quaisquer que sejam os métodos propostos.

Este projeto teve como objetivo apresentar e implementar uma abordagem para classificação de imagens de resíduos eletrônicos através de técnicas de Aprendizado Profundo, notadamente Redes Neurais Convolucionais (*Convolutional Neural Networks – CNNs*), já que estas têm se destacado na área de Visão Computacional. Dessa forma, foram construídos 18 modelos de CNNs, a partir de modelos originalmente pré-treinados, para que as eficácias de diferentes arquiteturas pudessem ser comparadas no contexto da identificação de *e-waste*. O processo de *transfer learning* foi conduzido a partir de uma base de dados com 1.670 imagens de resíduos eletrônicos, previamente rotuladas em 11 classes desbalanceadas. Para mitigar este cenário, aplicaram-se técnicas tais como *oversampling* e *data augmentation*, aumentando assim o número e a diversidade de imagens no conjunto de treinamento.

Após a avaliação dos resultados de acordo com métricas tais como *top-1 accuracy*, *top-5 accuracy*, *precision*, *recall* e *f1-score*, destaca-se o sucesso da estratégia de correção da resolução das imagens (FixResNet e FixEfficientNet), em comparação com as arquiteturas originais (ResNet e EfficientNet). O modelo *fixresnet152* obteve a melhor acuracidade top-1 média (83,4%), calculada entre todas as 5 *folds* originadas pelo método *Stratified K-Fold*, mas é importante enfatizar que o modelo *inceptionresnet_v1* obteve a segunda melhor acuracidade top-1 média (82,1%), ainda que tenha sido treinado com menos do que a metade do número de parâmetros e com um tempo total cerca de 10 minutos mais rápido em relação ao modelo *fixresnet152*. Os resultados poderão servir de base para futuros trabalhos que almejem a automação do processo de classificação e separação de materiais nos estágios iniciais de uma planta de reciclagem.

Palavras-chave: Redes Neurais Convolucionais, Aprendizado Profundo, Ciência de Dados, Classificação de Imagens, Resíduos Eletrônicos.

ABSTRACT

SILVA, J. A. **E-Waste Images Classification Using Convolutional Neural Networks** . 2023. 97 p. Dissertação (Mestrado – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

The correct disposal of end-of-life electric and electronic equipment is a matter of extreme relevance in today's society, where the constant emergence of new devices encourages its unrestrained consumption, resulting in the generation of huge volumes of waste. Therefore, the development and application of new techniques to optimize the recycling process is a predominant factor for expanding a Green Supply Chain Management. In this sense, some works have been proposed in the field of waste image classification, but few specifically deal with electrical and electronic equipment, whose specificities must be properly addressed whatever the proposed methods.

This project aimed to present and implement an approach for classifying electronic waste images through Deep Learning techniques, notably Convolutional Neural Networks (CNNs), as these have stood out in the field of Computer Vision. To this end, 18 CNN models were built from originally pre-trained models, so that the efficiencies of different architectures could be compared in the context of e-waste identification. The transfer learning process was carried out from a database with 1,670 electronic waste images, previously labeled into 11 imbalanced classes. In order to mitigate this scenario, techniques such as *oversampling* and *data augmentation* were applied, thus increasing the number and diversity of images in the training set.

After evaluating the results according to metrics such as top-1 accuracy, top-5 accuracy, precision, recall, and f1-score, the success of the image resolution correction strategy (FixResNet and FixEfficientNet) stands out, compared to the original architectures (ResNet and EfficientNet). The *fixresnet152* model achieved the highest average top-1 accuracy (83.4%), calculated among all 5 folds originated by the Stratified K-Fold method, but it is important to emphasize that the *inceptionresnet_v1* model obtained the second highest average top-1 accuracy (82.1%), even though it was trained with less than half the number of parameters and with a total time about 10 minutes faster than the *fixresnet152* model. The results may serve as a basis for future work aimed at automating the classification and separation of materials in the initial stages of a recycling plant.

Keywords: Convolutional Neural Networks, Deep Learning, Data Science, Image Classification, E-Waste.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
BatchNorm	Batch Normalization
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network
GA	Genetic Algorithm
GB	Gigabyte
GPU	Graphical Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LRN	Local Response Normalization
MAC	Multiply-and-Accumulate
MLH-CNN	Multilayer Hybrid Convolutional Neural Network
NIR	Near-InfraRed
PCB	Printed Circuit Board
R-CNN	Region Based Convolutional Neural Network
RAM	Random Access Memory
ReLU	Rectified Linear Units
ResNet	Residual Network
RGB	Red, Green, Blue
RMSProp	Root Mean Square Propagation
RSU	ReSidual U-blocks
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VGG	Visual Geometry Group
WEEE	Waste from Electrical and Electronic Equipment

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

Com a crescente adesão das pessoas à economia digital, através de oportunidades geradas por redes cada vez mais rápidas, novas aplicações e serviços providos em velocidades cada vez maiores, nota-se o uso crescente de equipamentos elétricos e eletrônicos, que, consequentemente, acabam por dar origem a grandes volumes de resíduos eletrônicos (*electronic waste* ou *e-waste*), quando são descartados sem o propósito de reúso.

Apenas para se ter uma ideia da dimensão de tais volumes, durante 2016 foram geradas cerca de 44,7 milhões de toneladas métricas de lixo eletrônico, em escala global, o que equivale a aproximadamente 4.500 Torres Eiffel. Entretanto, estima-se que apenas 20% deste montante foi reciclado através dos canais apropriados, embora 66% da população mundial esteja coberta por legislação associada ao tratamento de lixo eletrônico. Cabe ainda destacar que poucos países no mundo apresentam estatísticas regulares e harmonizadas sobre *e-waste*. Para cada produto elétrico ou eletrônico, sua função original, relevância ambiental, peso, tamanho e composição variam consideravelmente (BALDÉ *et al.*, 2017).

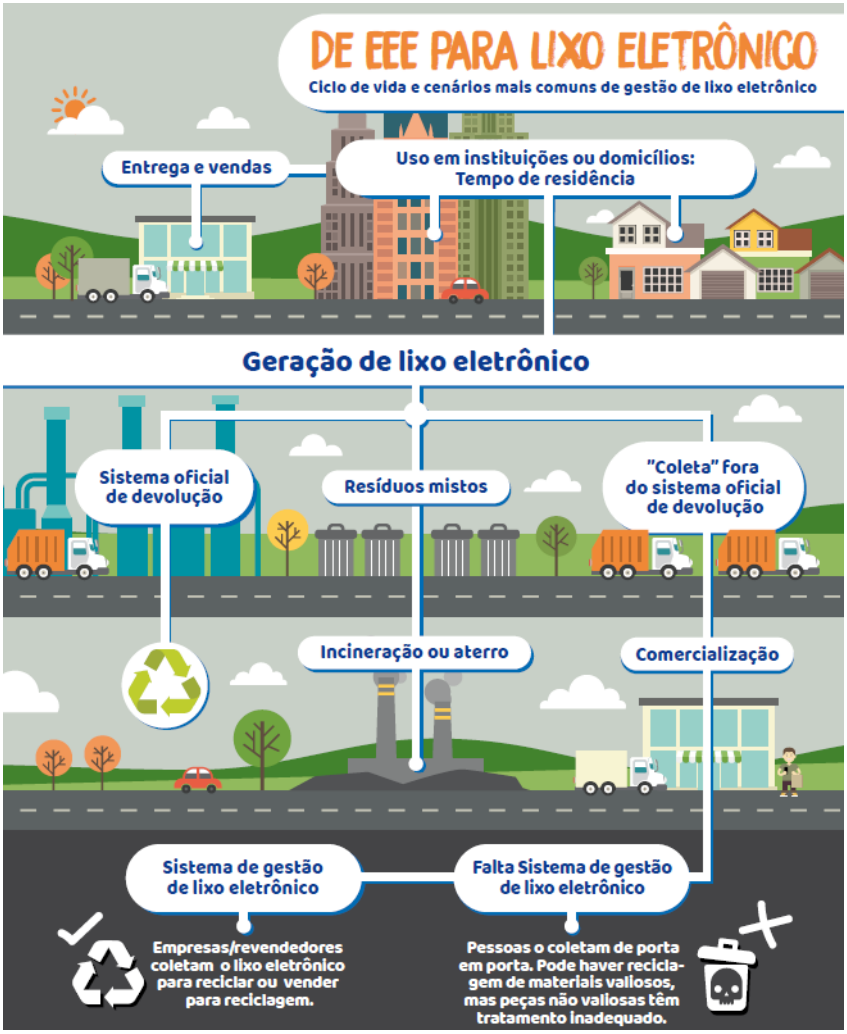
Equipamentos descartados, tais como telefones, computadores, geladeiras, sensores e TVs contêm substâncias que oferecem consideráveis riscos ao meio ambiente e à saúde, especialmente se tratados inadequadamente. Ao mesmo tempo, grandes fluxos de resíduos eletrônicos desafiam os esforços associados a uma economia circular, à medida em que recursos valiosos e escassos são desperdiçados.

A disponibilidade de material para fabricação de novos produtos de alta tecnologia é um grande problema na cadeia de suprimentos. Metais de terras raras, por exemplo, são geralmente minerados em países politicamente instáveis, onde há um terreno fértil para conflitos e violação de direitos humanos, ou em países que geralmente aplicam restrições de exportação. Consequentemente, flutuações nos preços das *commodities* pressionam os fabricantes a diminuírem suas margens de lucro. A conversão de resíduos em matéria-prima oferece uma grande oportunidade para que os fabricantes possam diminuir sua dependência, além de extrair valor adicional dos

produtos que foram vendidos e alcançaram seu final de vida útil (*End Of Life*). Para que isso seja possível, inovações tecnológicas e otimização do processo de reciclagem são fatores cruciais (STRATFOR, 2019).

O principal método de gerenciamento de resíduos é através de aterro (*landfill*), que é um destino ineficiente e caro, e que ainda coloca em risco o meio ambiente. Um outro método comum de gerenciamento de resíduos é a sua queima (*incineration*), que pode gerar poluição do ar e, consequentemente, danos à saúde das pessoas que situam-se próximas às áreas onde tal método é executado. Logo, é necessário reciclar os resíduos para proteger o meio ambiente e a saúde humana, sendo a separação adequada de materiais uma etapa significativa neste processo (BALDÉ; FORTI; KUEHR, 2018). A Figura 1 mostra os cenários mais comuns no gerenciamento de *e-waste*.

Figura 1 – Ciclo de vida da transformação de EEE (Equipamentos Elétricos e Eletrônicos) em *e-waste* (lixo eletrônico), e os cenários de gerenciamento mais comuns.



Fonte: Forti (2019).

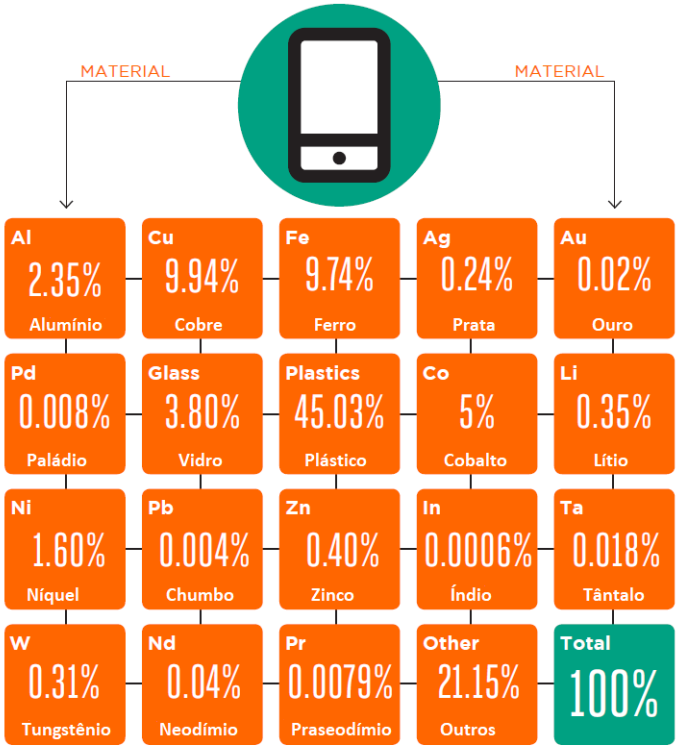
1.1 Motivação

De maneira geral, a cadeia de valor para reciclagem de resíduos eletrônicos consiste em três passos principais:

- Coleta de resíduos eletrônicos;
- Pré-processamento mecânico: desmantelamento e/ou trituração e separação de resíduos;
- Processamento final e refinamento: transformação dos resíduos em matéria-prima.

Para maximizar a eficiência do processo de reciclagem e a consequente recuperação de matéria-prima, é preponderante que ocorra uma ampla otimização nesta cadeia logística. O desmantelamento de produtos eletrônicos em frações menores é essencial para a recuperação de materiais críticos ou preciosos. Dessa forma, ineficiências no processo de separação causam perda de frações de materiais que não podem ser recuperados posteriormente. Fragmentos de plásticos, por exemplo, podem contaminar a pureza das Placas de Circuito Impresso (*Printed Circuit Boards - PCBs*) e vice-versa (MAGALINI; KUEHR; BALDÉ, 2015). Para ilustrar alguns tipos de materiais encontrados em resíduos eletrônicos, a Figura 2 mostra a composição simplificada, por peso, de um telefone celular.

Figura 2 – Composição simplificada de telefone celular.



Fonte: Adaptada de Magalini, Kuehr e Baldé (2015).

Neste contexto, há alguns desafios importantes:

- Complexidade dos produtos: cada nova geração de dispositivos elétricos e eletrônicos entrega mais funcionalidades, o que é normalmente obtido através de um *design* de circuitos mais integrados a um custo mais reduzido. A destruição em massa desses dispositivos eletrônicos totalmente integrados cria uma mistura homogênea de fragmentos de resíduos, de modo que a classificação desses pequenos fragmentos em grupos como metais ferrosos e não ferrosos, plásticos, PCBs, etc. está se tornando cada vez mais difícil (MAGALINI; KUEHR; BALDÉ, 2015);
- Pontos de contato humano: para executar tarefas repetitivas, mas variáveis, tais como desmontagem e separação, os pré-recicladores de resíduos eletrônicos empregam mão-de-obra manual. Disponibilidade e retenção de recursos consistem em um grande desafio devido à natureza monótona das tarefas. Além disso, o manuseio do material pode favorecer a proliferação de doenças oriundas de substâncias nocivas presentes nos resíduos (BALDÉ *et al.*, 2017);
- Valor desconhecido: informação insuficiente acerca da composição dos fragmentos de resíduos eletrônicos, tais como PCBs, que deverão ser convertidos em matéria-prima, torna necessária a realização de amostragem e análise física. Como o valor bruto por tonelada de material pode variar significativamente, pré-recicladores têm pouca segurança em seu planejamento financeiro (BALDÉ *et al.*, 2017).

Logo, a pesquisa justifica-se pelas enormes oportunidades que a classificação otimizada de resíduos eletrônicos pode trazer ao processo de reciclagem, com claros benefícios econômicos advindos do melhor aproveitamento do material que poderá ser convertido em matéria-prima, possibilitando inclusive o compartilhamento de informação em tempo real entre pré-recicladores e recicladores finais para otimizar as etapas de previsão, refinamento e pagamento.

Além disso, há a possibilidade de direcionar os resíduos eletrônicos aos métodos mais adequados de disposição final, gerando uma grande contribuição à preservação do meio ambiente e à manutenção da saúde humana.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral da pesquisa é propor uma abordagem para classificação de imagens de resíduos eletrônicos, com base em conceitos de Visão Computacional (*Computer Vision*), Processamento de Imagens (*Image Processing*), Aprendizado Profundo (*Deep Learning*) e Redes Neurais Convolucionais (*Convolutional Neural Networks*), dentre outras técnicas associadas à Ciência de Dados (*Data Science*). Desta forma, espera-se obter um resultado que seja útil para futuros trabalhos envolvendo a automação do processo de separação de resíduos eletrônicos, de modo que este seja executado com maior precisão, ajudando a prever o valor intrínseco do material e a direcioná-lo para o método mais adequado de reciclagem.

1.2.2 Objetivos Específicos

Dentre os objetivos específicos da pesquisa estão:

1. Analisar uma base de dados de imagens de resíduos eletrônicos oriundos da operação de *Scrap* e *Waste Disposal* de uma grande empresa da área de Tecnologia da Informação;
2. Comparar algoritmos de *Deep Learning* já existentes, com a finalidade de entender suas estruturas, diferenças, vantagens e desvantagens;
3. Selecionar e implementar os algoritmos que serão utilizados para a classificação das imagens;
4. Analisar o desempenho dos algoritmos no processo de classificação das imagens, utilizando métricas relevantes para o problema em questão.

1.3 Trabalhos Relacionados

Ao longo dos anos, alguns trabalhos foram desenvolvidos para a classificação de imagens de resíduos. Em 2016, por exemplo, durante o evento *TechCrunch's Disrupt San Francisco*, foi apresentada uma *Auto-Trash* capaz de diferenciar vários tipos de resíduos, usando um módulo Raspberry Pi, equipado com câmera e *software* de reconhecimento de imagem baseado na plataforma de Aprendizado de Máquina TensorFlow, desenvolvida pelo Google (TECHCRUNCH, 2016).

Um outro projeto foi o desenvolvimento de um aplicativo móvel para segmentar uma pilha de lixo numa imagem. O objetivo do aplicativo era permitir que cidadãos pudessem monitorar e reportar lixo em suas vizinhanças. A base de dados foi obtida através de busca no

Bing Image, e os autores extraíram pacotes de imagens para treinar sua rede neural. Foi utilizado um modelo AlexNet pré-treinado, obtendo cerca de 87,68% de acurácia média (MITTAL *et al.*, 2016).

Já Yang e Thung (2016) propuseram uma abordagem para classificar resíduos em diferentes categorias (por exemplo, metal, papel, vidro, plástico, papelão) usando uma Máquina de Vetor de Suporte (*Support Vector Machine – SVM*), com Transformação de Característica Invariante à Escala (*Scale Invariant Feature Transform - SIFT*), e uma Rede Neural Convolucional (*Convolutional Neural Network – CNN*). Tal metodologia alcançou uma acurácia de 63% e 22% para a SVM e a CNN, respectivamente. Utilizando uma abordagem similar, Awe, Mengistu e Sreedha (2017) propuseram um método para categorizar os diferentes tipos de resíduos em três categorias diferentes (aterro, reciclagem e papel), usando uma CNN baseada em Região (*Region-based Convolutional Neural Network – R-CNN*).

Adedeji e Wang (2019) apresentaram uma combinação entre ResNet-50 e SVM para classificar 4 classes de resíduos (vidro, papel, plástico e metal) e obtiveram uma acurácia de cerca de 87% com otimização através de aumento de dados (*data augmentation*). A principal limitação, contudo, recaiu sobre o pequeno número de imagens disponíveis para os testes, já que o modelo foi testado apenas no *dataset* TrashNet (YANG; THUNG, 2016).

Møgaard (2017) apresentou um abordagem envolvendo fusão de sensores, empregando redes neurais para classificar imagens de *Waste from Electrical and Electronic Equipment - WEEE* (incluindo *laptops*, telefones celulares, liquidificadores, furadeiras e chaleiras elétricas), obtidas através de uma câmera RGB (*Red, Green, Blue*) com capacidade NIR (*Near-InfraRed*), em um cenário de separação de resíduos. Foi demonstrado que Redes Neurais Convolucionais treinadas em imagens naturais podem ser usadas como extratores de características antes que uma camada de saída adapte-se ao domínio de classificação de imagens, contando com uma quantidade relativamente pequena de dados de treinamento. O método obteve uma acurácia de 73% para imagens de objetos sobrepostos pertencentes a 10 classes.

Nowakowski e Pamula (2020) apresentaram uma nova abordagem para identificação e categorização de *e-waste*, usando CNN para classificar o tipo de lixo eletrônico e R-CNN para detectar a categoria e o tamanho dos resíduos nas imagens dos equipamentos. A acurácia obtida foi de 90 - 96,7%, de modo que a R-CNN alcançou menor acurácia média (90%) em relação à melhor CNN, mas permitiu a identificação e a determinação do tamanho do objeto na imagem.

Zhang *et al.* (2021) propuseram um modelo de classificação de diferentes resíduos recicláveis (papelão, papel, metal, plástico, vidro e outros) com base em uma rede residual de 18 camadas (ResNet-18), que foi aprimorada para integrar as características relevantes de todos os mapas de canal, comprimir as características do espaço dimensional e ter um campo receptivo global, porém mantendo o mesmo número de canais. O modelo foi testado no *dataset* TrashNet (YANG; THUNG, 2016), obtendo uma acurácia de 95,87% na classificação dos resíduos.

Mao *et al.* (2021) aplicaram diversas CNNs, tais como DenseNet-121, alcançando uma acurácia máxima de 99,6% no *dataset* TrashNet (YANG; THUNG, 2016). Além de adotar a técnica de *data augmentation*, o estudo restaurou o classificador da DenseNet-121 para obter duas camadas totalmente conectadas, e então usou GA (*Genetic Algorithm*) para otimizar os hiperparâmetros de cada camada totalmente conectada. Cabe destacar também que o mapeamento de ativação de classe ponderada por gradiente (*gradient-weighted class activation mapping*) ajudou a destacar as características grosseiras das imagens dos resíduos, fornecendo uma visão adicional sobre a explicabilidade da DenseNet-121 otimizada.

Ramsurrun *et al.* (2021) testaram 12 variantes de CNNs no *dataset* TrashNet (YANG; THUNG, 2016), utilizando três diferentes classificadores: SVM, *sigmoid* e *softmax*; os resultados mostratram que a implementação da rede VGG-19 com *softmax* alcançou acurácia em torno de 88%.

Shi *et al.* (2021), por sua vez, analisando o mesmo *dataset*, propuseram uma MLH-CNN (*Multilayer Hybrid Convolutional Neural Network*), cuja estrutura é similar à de uma rede VGG, porém mais simples, por conta de mudanças no número de módulos e canais. Comparada às implementações das redes AlexNet, ResNet-50 e VGG-16, a MLH-CNN obteve uma acurácia de 92,6%, bem maior do que aquela alcançada pelas outras três redes mencionadas.

1.4 Organização deste Trabalho

Neste capítulo, foi disposta uma introdução, explicando os motivos que levaram à escolha do tema e as necessidades que a pesquisa busca preencher, incluindo motivações, objetivos e trabalhos relacionados. A seguir, descreve-se o conteúdo dos demais capítulos.

No Capítulo 2, encontra-se a fundamentação teórica com os os principais conceitos acerca de CNNs, incluindo suas principais camadas, assim como as arquiteturas testadas durante este trabalho, tais como VGG, ResNet, Inception, Inception-ResNet, ResNeXt, FixRes, EfficientNet e FixEfficientNet. É descrita ainda a arquitetura U^2-Net, aplicada ao processo de segmentação do *background* das imagens.

No Capítulo 3, descrevem-se os materiais e métodos utilizados neste trabalho. Há detalhes referentes às características da base de dados (classes e quantidades de imagens) e às ferramentas e recursos computacionais empregados durante o processo de treinamento dos modelos de CNNs. Em seguida, detalham-se as abordagens de treinamento, incluindo a maneira como foram divididos os conjuntos de treinamento e de teste, a técnica de balanceamento para ajuste das quantidades de amostras de cada classe, o processo de *data augmentation*, bem como os demais hiperparâmetros configurados para implementar os modelos de CNNs. Além disso, foram descritas as métricas de avaliação adotadas para medir o desempenho dos modelos propostos.

No Capítulo 4, demonstram-se os resultados dos experimentos e as reflexões associadas, através de análises comparativas envolvendo cada um dos modelos propostos, a partir das

métricas descritas no capítulo anterior.

Finalmente, no Capítulo 5, apresentam-se as conclusões, sumarizando o conteúdo abordado, as principais contribuições e as limitações envolvidas nos experimentos. Descrevem-se ainda as sugestões e as perspectivas para trabalhos futuros.

CAPÍTULO

2

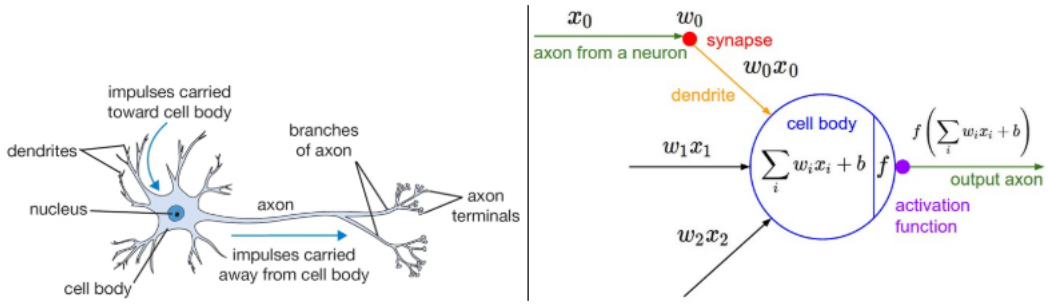
FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentados os principais conceitos relativos a CNNs, desde suas características estruturais até as principais arquiteturas que foram testadas durante este trabalho, tais como VGG, ResNet, Inception, Inception-ResNet, ResNeXt, FixRes, EfficientNet e FixEfficientNet. Será descrita ainda a arquitetura U^2-Net, aplicada ao processo de segmentação do *background* das imagens.

2.1 Redes Neurais Convolucionais

As CNNs constituem uma área de Aprendizado Profundo (*Deep Learning*) que foi inspirada no córtex visual e que contribuiu para o progresso da área de Visão Computacional, fundamental ao processo de classificação de imagens. Redes Neurais consistem em conjuntos de funções matemáticas dependentes. Cada função individual pode ser representada por um neurônio (*perceptron*), cujos componentes básicos são: entrada, peso, função de ativação e saída. A Figura 3 demonstra o neurônio biológico e sua representação matemática.

Figura 3 – Neurônio biológico (esquerda) e seu modelo matemático (direita).



Fonte: Konaté (2019).

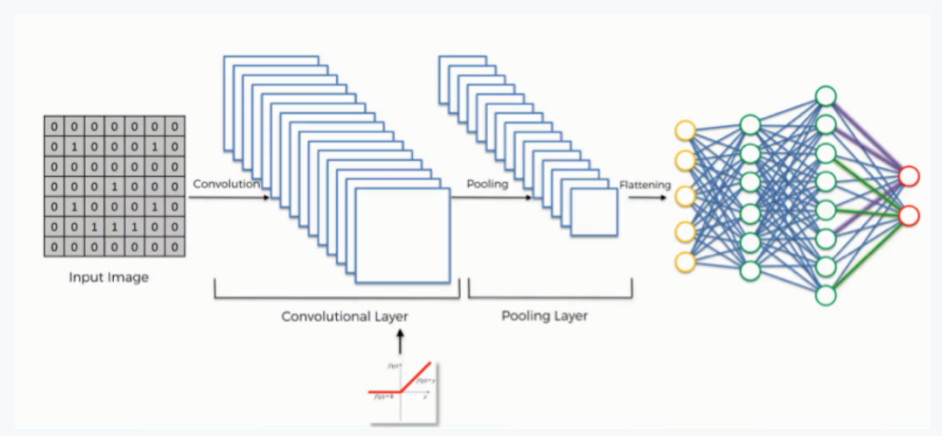
A tarefa de classificação consiste em atribuir à imagem uma determinada classe ou a

probabilidade de pertencer a esta classe. Ao processar uma imagem, um computador a interpreta como uma série de *pixels* dispostos em uma matriz, com valores variando entre 0 e 255, possuindo três canais de cores quando trata-se de uma imagem RGB. Segundo Wu (2017), uma imagem composta por H linhas, W colunas e três canais (R, G e B) também pode ser representada como um tensor de ordem 3, ou ordem 2, caso esteja em escalas de cinza.

É possível definir os tensores como estruturas matemáticas que representam relações lineares entre vetores, escalares e outros tensores, ou como uma série de valores básicos organizados em uma matriz com um número específico de dimensões. De maneira simplificada, pode-se considerar um tensor como uma matriz multidimensional. Os tensores podem ser classificados em diferentes ordens, como, por exemplo, os tensores de ordem zero, que são compostos por apenas um número ou por apenas um escalar.

Na estrutura de uma CNN, a imagem de entrada é processada através de várias camadas, buscando-se obter uma saída que possa ser utilizada para promover sua classificação. Há diversos tipos de camadas em uma CNN, dentre as quais temos: camadas de convolução (*convolutional*), camadas de agrupamento (*pooling*) e camadas totalmente conectadas (*fully connected*). As primeiras etapas de uma CNN, ou seja, suas primeiras camadas convolucionais e de *pooling*, são consideradas como as mais importantes, tendo contribuição significativa em grande parte do processamento computacional dispendido. A Figura 4 mostra a arquitetura simplificada de uma Rede Neural Convolucional.

Figura 4 – Arquitetura simplificada de uma Rede Neural Convolucional.



Fonte: SuperDataScience (2018).

De acordo com Wu (2017), a última camada corresponde à camada de perda. Considerando t como o objetivo relacionado à entrada x^i, então o custo (ou perda) calcula a diferença entre o objetivo t e a predição x^l. Pode-se representar uma função de perda genérica por:

$$z = \frac{1}{2}\left\|t - x^l\right\|^2. \tag{2.1}$$

Geralmente, esta equação é utilizada para problemas de regressão. No contexto de classificação de imagens, a função de perda utilizada com maior frequência é a entropia cruzada (*cross-entropy*) definida como:

$$H(p,q) = -\sum_{x} p(x)\log(q(x)). \tag{2.2}$$

A entropia cruzada ajuda a determinar a discrepância entre duas distribuições de probabilidade: a distribuição "verdadeira" representada por p e a distribuição prevista representada por q.

2.1.1 Camadas Convolucionais - Convolutional Layers

De acordo com Rawat e Wang (2017), as camadas convolucionais funcionam como extratores de características (*feature extractors*) e, portanto, aprendem as representações das características das imagens de entrada. Os neurônios associados às camadas convolucionais são dispostos em mapas de características (*feature maps*). Cada neurônio em um mapa de características tem um campo receptivo (*receptive field*), que está conectado a neurônios das camadas anteriores por meio de um conjunto de pesos treináveis, comumente denominados filtros. As imagens de entrada são "varridas" pelos filtros, através do cálculo do produto escalar entre a matriz de valores que representa a imagem de entrada e a matriz de pesos dos filtros, produzindo um mapa de ativação (*activation map*) bidimensional. Considerando que os filtros são menores do que os valores de entrada, permite-se que o mesmo filtro seja multiplicado diversas vezes pela matriz da imagem de entrada em diferentes pontos. Especificamente, o filtro é aplicado sistematicamente a cada parte sobreposta, da esquerda para a direita, de cima para baixo. Os resultados são transmitidos para a próxima camada mediante a utilização de uma função de ativação não linear. Todos os neurônios em um mapa de características têm pesos que são condicionados a serem iguais; entretanto, em uma mesma camada de convolução, diferentes mapas de características têm pesos distintos para permitir a extração de várias características em cada local. Assim, o k-ésimo mapa de características $\mathrm{Y_k}$ resultante pode ser computado como:

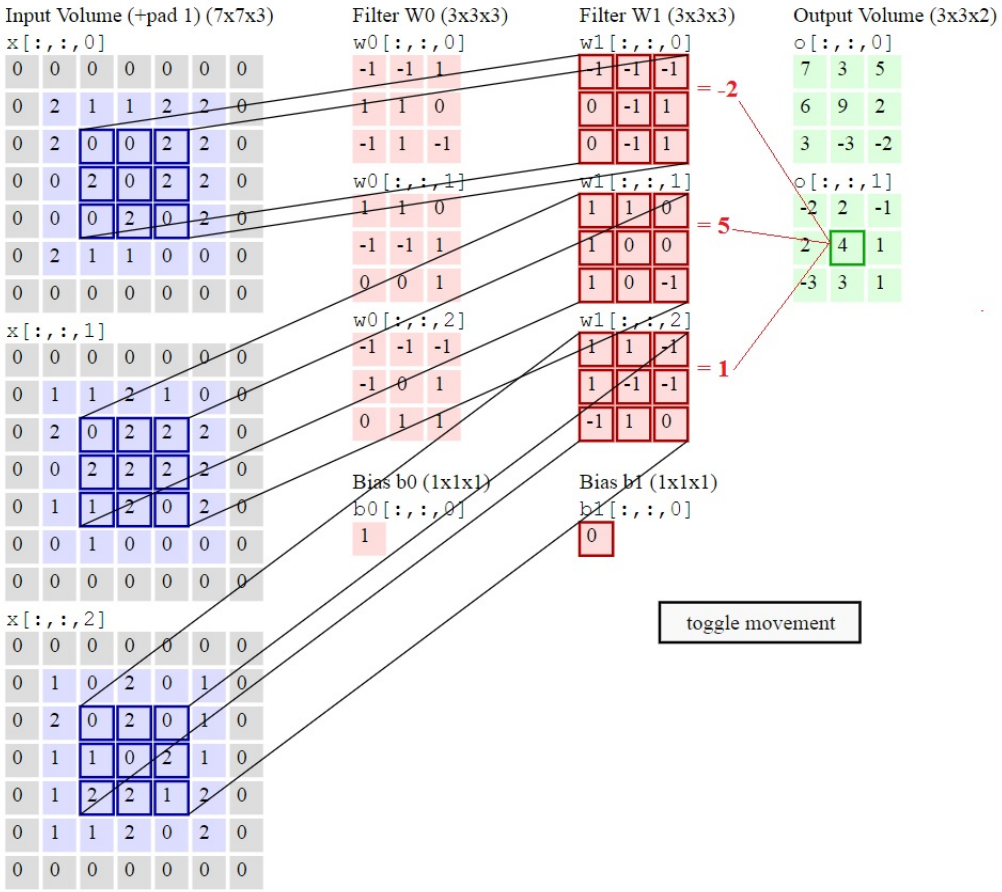
$$Y_{\mathrm{k}} = f(W_{\mathrm{k}} * x), \tag{2.3}$$

em que a imagem de entrada é designada por x; o filtro associado ao k-ésimo mapa de características é designado por $\mathrm{W_k}$; o sinal de multiplicação neste caso está associado ao operador convolucional bidimensional, que serve para determinar o produto interno do filtro em cada local da imagem de entrada; e $f()$ caracteriza a função de ativação não linear. Funções de ativação tradicionais tais como sigmóide e tangente hiperbólica, representadas respectivamente por $s(x) = \frac{1}{1+e^{-x}}$ e $tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, têm cedido espaço para Unidades Lineares Retificadas (*Rectified Linear Units - ReLUs*), que tornaram-se populares para Redes Neurais Profundas por

Krizhevsky, Sutskever e Hinton (2012), devido à sua considerável superioridade em relação aos tempos de treinamento.

Uma camada convolucional é demonstrada na Figura 5: o volume de entrada é de tamanho $W_1 = 5$ (*Width*), $H_1 = 5$ (*Height*), $D_1 = 3$ (*Depth*), e os parâmetros da camada convolucional são K = 2 (*Filters*), F = 3 (*Spatial Extent*), S = 2 (*Stride*) e P = 1 (*Padding*). Ou seja, há 2 filtros de tamanho 3x3, que são aplicados com movimento de 2 saltos (*Stride = 2*), e nota-se ainda um *Padding* P = 1 aplicado ao volume de entrada, tornando sua borda externa igual a zero. O tamanho espacial do volume resultante equivale a W_2 x H_2 x D_2, em que $W_2 = (W_1 - F + 2P)/S + 1$, $H_2 = (H_1 - F + 2P)/S + 1$ e D_2 = K. Neste caso, W_2 x H_2 x D_2 é equivalente a 3x3x2.

Figura 5 – Demonstração de uma Camada Convolucional.



Fonte: Stanford (2021).

2.1.2 Camadas de Agrupamento - Pooling Layers

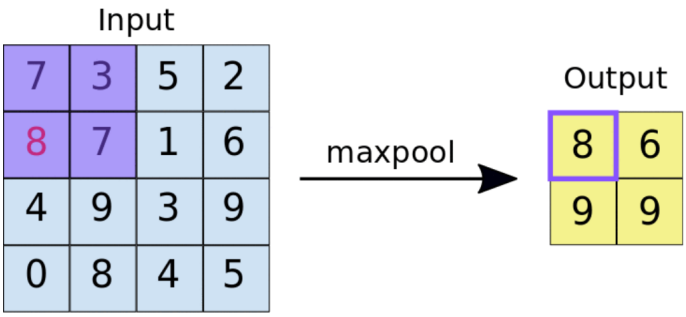
Uma camada de agrupamento simplifica a informação da camada anterior, reduzindo o espaço de resolução dos mapas de características, alcançando assim invariância espacial para inserir distorções e traduções. Em modelos mais antigos, aplicavam-se operações de agrupamento com média (*mean pooling*) para propagar a média de todos os valores de entrada. Em modelos

mais recentes, aplicam-se operações de agrupamento com máximo (*max pooling*) para propagar à camada seguinte o valor máximo em um campo receptivo, ou seja, o maior elemento dentro de cada campo receptivo é selecionado, de forma que:

$$Y_{\text{kij}} = \max_{(p,q)\in\Re_{\text{ij}}} x_{\text{kpq}}, \tag{2.4}$$

em que o resultado da operação de agrupamento, associada ao k-ésimo mapa de características, é denotado por Y_{kij}, enquanto x_{kpq} corresponde ao elemento no local *(p, q)* contido na região de agrupamento $\Re_{\text{ij}}$, que incorpora um campo receptivo ao redor da posição *(i, j)*. Dada uma imagem 4x4, se um filtro 2x2 com movimento de 2 saltos (*Stride = 2*) é aplicado, então uma operação de agrupamento com máximo resulta no valor máximo de cada região 2x2. A Figura 6 exemplifica a aplicação de *max pooling*.

Figura 6 – *Max pooling* com área 2x2.

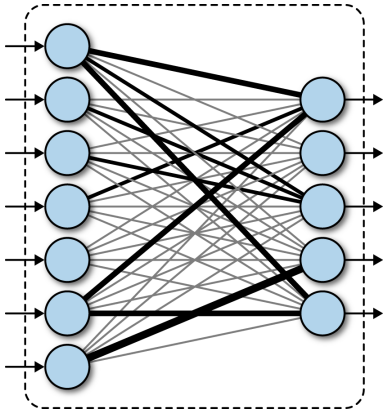


Fonte: nueronio.ai (2018).

2.1.3 Camadas Totalmente Conectadas - Fully Connected Layers

As camadas totalmente conectadas ligam cada neurônio de uma camada a todos os nurônios de outra camada, de modo que várias camadas convolucionais e de agrupamento são empilhadas para extrair representações mais abstratas ao longo da rede, conforme ilustrado na Figura 7. Para problemas de classificação, usa-se comumente um operador *softmax*, que converte um vetor de números em um vetor de probabilidades, em que as probabilidades de cada valor são proporcionais à escala relativa de tais valores no vetor. Embora muitos modelos adotem *softmax* para executar as predições, pode haver uma pequena vantagem ao substituir tal operador por uma Máquina de Vetor de Suporte (*Support Vector Machine*), o que resultaria em maior acuracidade em tarefas de classificação, dependendo do conjunto de dados em questão (TANG, 2013).

Figura 7 – Camadas Totalmente Conectadas em uma Rede Neural Profunda.



Fonte: Ramsundar e Zadeh (2018).

2.2 Principais Arquiteturas de Redes Neurais Convolucionais

Desde a década de 90, as Redes Neurais Convolucionais têm demonstrado excelente desempenho em tarefas associadas à Visão Computacional. Mas, foi a partir do desempenho recorde alcançado pela AlexNet, uma grande arquitetura de Rede Neural Convolucional desenvolvida por Krizhevsky, Sutskever e Hinton (2012) na Universidade de Toronto - vencedora do ILSVRC 2012 (*ImageNet Large Scale Visual Recognition Challenge*), alcançando uma taxa de erro de 15,3% (mais de 10 pontos percentuais mais baixa do que a segunda melhor arquitetura na competição) - que as Redes Neurais Convolucionais popularizaram-se para classificação de imagens.

Posteriormente, diversas melhorias foram propostas, impulsionadas pela disponibilidade de grandes bases de dados com imagens anotadas e pelo desenvolvimento de *Graphical Processing Units (GPUs)* mais poderosas, que ajudaram a acelerar os pesados processos computacionais relacionados às redes, reduzindo o tempo necessário para a execução das etapas de treinamento. Ao mesmo tempo, novas ideias e algoritmos melhoraram as arquiteturas, inclusive no que diz respeito à redução de *overfitting* (SZEGEDY *et al.*, 2014). Nas seções a seguir, serão listadas algumas das principais arquiteturas de Redes Neurais Convolucionais.

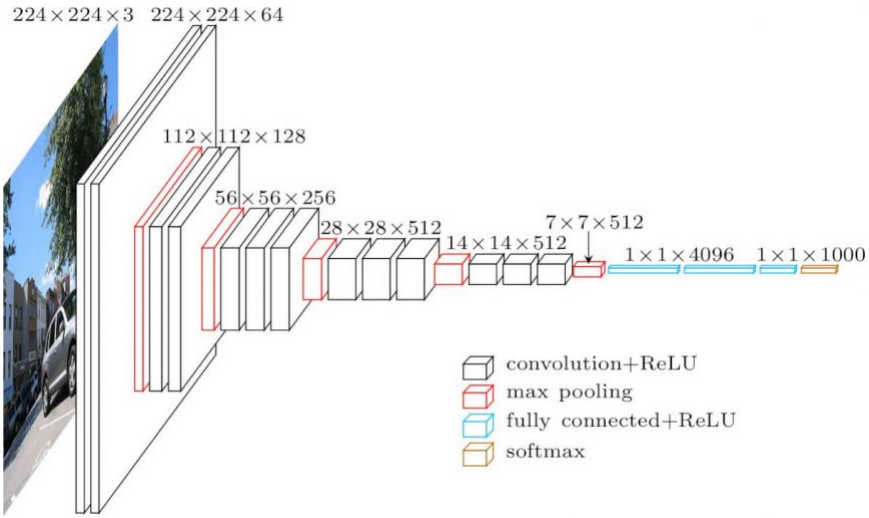
2.2.1 VGG - Visual Geometry Group

Enquanto os derivados anteriores da AlexNet concentraram-se em tamanhos de janela menores e avanços na primeira camada convolucional, a rede VGG aborda outro aspecto muito importante das CNNs: profundidade. Existem inúmeras variantes desta rede, com profundidades até 19, sendo VGG-16 a mais popular, com 16 camadas de profundidade e uma estrutura constituída exclusivamente de convoluções 3x3 (menor tamanho possível para capturar a noção de

equerda/direita, cima/baixo e centro). Há também filtros 1x1 que atuam como transformações lineares dos canais de entrada (seguidos por não linearidade). O passo de convolução (*convolutional stride*) é fixado em 1 *pixel*, enquanto o preenchimento espacial da camada convolucional de entrada garante que a resolução espacial seja mantida. O agrupamento espacial (*spatial pooling*) é realizado por cinco camadas *max pooling* que seguem algumas das camadas convolucionais. O agrupamento com máximo é realizado sobre uma janela de 2x2 *pixels*, com salto (*stride*) igual a 2 (SIMONYAN; ZISSERMAN, 2014).

A VGG-16 conta com 3 camadas totalmente conectadas, sendo que as duas primeiras possuem 4.096 canais e a terceira possui 1.000 canais, um para cada classe. Todas as camadas ocultas (*hidden layers*) utilizam ReLU. A VGG geralmente não utiliza LRN (*Local Response Normalization*), já que isto aumenta o consumo de memória e o tempo de treinamento, sem melhoria na acurácia. A Figura 8 ilustra a arquitetura de uma rede VGG-16.

Figura 8 – Arquitetura da VGG-16.



Fonte: Neurohive (2018).

Embora a rede tenha alcançado um *top-5 error* no valor de 7,3%, ela contém quase 140 milhões de pesos, resultando em um modelo de tamanho enorme, que demanda 16 bilhões de operações MAC (*Multiply-and-Accumulate*) para processar uma imagem 224x224. Alternativamente, pode-se importar uma versão pré-treinada com um número reduzido de pesos, excluindo, por exemplo, as camadas totalmente conectadas (*fully connected layers*). Isso pode ser útil quando dispõe-se de recursos computacionais limitados ou quando pretende-se usar o modelo como um extrator de características para *transfer learning*, em que as características aprendidas pelo modelo são aplicadas em uma tarefa ou conjunto de dados diferente.

2.2.2 ResNet - Residual Network

A arquitetura ResNet (*Residual Network*) foi originalmente introduzida por He *et al.* (2015) para resolver o problema de Desvanecimento do Gradiente (*Vanishing Gradient*). O artigo apresentou uma arquitetura ResNet com profundidade de 18 camadas (ResNet-18), que foi usada como *baseline* para comparação com redes mais profundas e obteve desempenho relativamente inferior em relação às versões de 34, 50, 101 e 152 camadas.

A ResNet foi a primeira rede neural a obter acurácia de classificação superior a humanos em uma tarefa de reconhecimento de imagens, vencendo o ILSVRC 2015, ao atingir um *top-5 error* no valor de 3,57% com um conjunto (*ensemble*) de redes residuais.

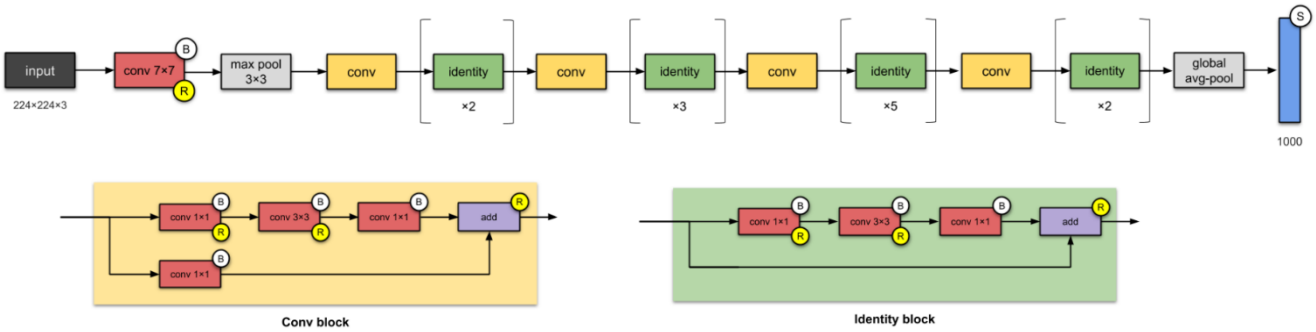
A estrutura de aprendizado residual facilita que as redes sejam significativamente mais profundas, melhorando o desempenho em tarefas visuais e não visuais. Essas redes residuais são muito mais profundas do que suas contrapartes "simples", mas exigem um número semelhante de parâmetros (pesos). Ao invés de esperar que cada pilha de camadas ajuste-se diretamente a um mapeamento subjacente desejado, permite-se claramente que essas camadas ajustem-se a um mapeamento residual. O mapeamento original é reformulado em $F(x)+x$, partindo do pressuposto de que é mais simples otimizar o mapeamento residual do que otimizar o mapeamento original não referenciado. De modo excepcional, se um mapeamento de identidade fosse ótimo, seria mais fácil empurrar o residual para zero do que ajustar um mapeamento de identidade por uma pilha de camadas não lineares (HE *et al.*, 2015).

A arquitetura ResNet-34 é uma variante da arquitetura ResNet original. Ela consiste em 34 camadas, que incluem conexões de atalho (*shortcut connections*) que são fundamentais para a estrutura de aprendizado residual. Estas conexões permitem que a rede aprenda através de caminhos curtos, permitindo camadas mais profundas enquanto ainda é capaz de aprender corretamente. Em comparação com as arquiteturas VGG, as ResNets possuem menos filtros e menor complexidade, e foram inspiradas na ideia de usar filtros de convolução menores com tamanho 3x3. Além disso, os autores da arquitetura ResNet seguiram duas regras simples de projeto: as camadas possuem o mesmo número de filtros para o mesmo tamanho de mapa de características de saída e o número de filtros é dobrado se o tamanho do mapa de características é dividido pela metade, a fim de preservar a complexidade de tempo por camada.

Embora a arquitetura ResNet-50 seja baseada no modelo acima descrito, existe uma diferença fundamental: o bloco de construção (*building block*) foi modificado para um *bottleneck design* devido às preocupações com o tempo decorrido para treinar as camadas. O modelo ResNet-50 utilizou uma pilha de 3 camadas ao invés de duas, obtendo maior acuracidade em relação ao modelo ResNet-34, sendo inclusive a primeira versão a obter acuracidade top-5 acima de 98% no conjunto de dados ImageNet. É possível carregar uma versão pré-treinada com mais de um milhão de imagens oriundas do ImageNet. A rede pré-treinada pode classificar imagens em 1.000 categorias de objetos, tais como teclado, *mouse*, e muitos animais. Como resultado,

a rede tem aprendido ricas representações de categorias para uma ampla gama de imagens. A rede possui um tamanho de imagem de entrada de 224x224 (MATHWORKS, 2017), conforme ilustrado na Figura 9.

Figura 9 – Arquitetura da ResNet-50, baseada no código do GitHub do keras-team.



Fonte: TowardsDataScience (2019).

As características primárias da imagem são obtidas através do suporte das primeiras camadas. À medida que o treinamento avança para as camadas seguintes, mais e mais características complexas e detalhadas são extraídas, com ajuda da função *softmax*, que converte um valor de entrada em um vetor de valores normalizados, em que cada elemento segue uma distribuição de probabilidade cuja soma é igual a 1. Os valores de saída ficam no intervalo [0,1], o que evita uma classificação binária, permitindo a acomodação de muitas classes ou dimensões no modelo da rede neural. É possível remover a camada de classificação do topo para que as características extraídas sejam passadas para um modelo de SVM.

SVM pode ser usada tanto para resolver problemas de classificação, quanto de regressão. Trata-se de uma técnica de Aprendizado de Máquina, considerada como um dos melhores algoritmos de classificação, em que os dados são plotados como pontos em um espaço n-dimensional, sendo que cada característica é uma coordenada específica. O método tenta encontrar o hiperplano que maximize a margem entre as diferentes classes, isto é, a distância entre o hiperplano e os pontos mais próximos de cada classe, denominados vetores de suporte. De maneira mais específica, temos que o objetivo de otimização de uma SVM está representado da seguinte maneira:

$$\begin{aligned} &\min_{\gamma,w,b} \frac{1}{2}\|w\|^2 \\ &t.q. \quad y^{(i)}(w^T x^i + b) \geq 1, \quad i = 1,\ldots,m\,, \end{aligned} \tag{2.5}$$

em que w, b são parâmetros da nossa função de hipótese, $y^{(i)}$ representa o rótulo (*label*) para um exemplo específico, x^i é o i-ésimo exemplo de m, e γ é a margem geométrica mínima de todos os exemplos de treinamento (YANG; THUNG, 2016).

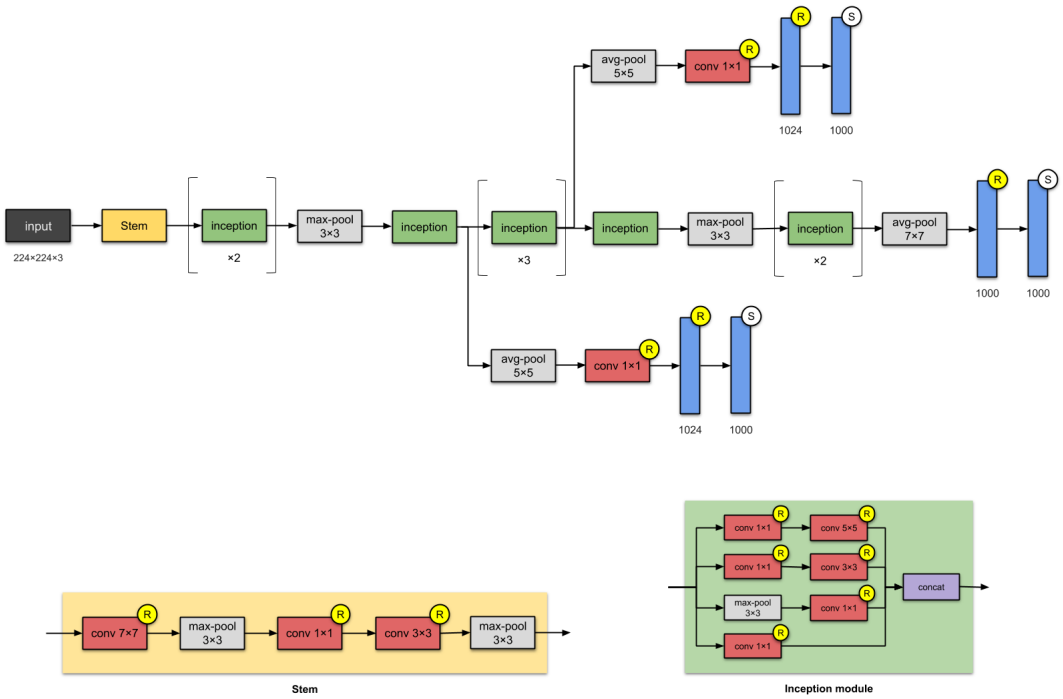
A ResNet-101 e a ResNet-152 são variantes mais profundas da arquitetura ResNet, uma vez que possuem mais camadas de blocos de resíduos. Isso significa que elas têm mais capacidade de representação e, portanto, podem lidar melhor com problemas mais complexos. Além disso, possuem mais parâmetros em comparação com a ResNet-50, o que aumenta a sua capacidade de ajuste aos dados de treinamento. No entanto, isso também significa que elas podem sofrer mais com *overfitting*, que ocorre quando um modelo estatístico ajusta-se muito bem aos dados de treinamento, mas falha em generalizar corretamente para novos dados. Logo, requerem-se mais recursos computacionais e dados de treinamento para alcançar uma performance satisfatória.

2.2.3 Inception e Inception-ResNet

Os módulos Inception são usados em Redes Neurais Convolucionais para permitir maior eficiência computacional, através de uma redução de dimensionalidade, com convoluções 1x1 empilhadas. Os módulos são desenvolvidos para resolver problemas tais como *overfitting*.

De acordo com DeepAI (s/d), como uma rede neural lida com um vasto conjunto de imagens, com uma larga variação no conteúdo das imagens apresentadas, o que é também conhecido como partes salientes, a rede precisa ser arquitetada de maneira adequada. A versão mais simples de um módulo Inception executa convoluções com três diferentes tamanhos de filtro (1x1, 3x3, 5x5). Além disso, o módulo aplica a técnica de *max pooling*, através da qual uma representação de entrada (tal como uma imagem) tem sua dimensionalidade reduzida, permitindo suposições acerca das características contidas nas sub-regiões classificadas. Posteriormente, as saídas resultantes são concatenadas e enviadas para a próxima camada. Ao executar suas convoluções no mesmo nível, a rede fica progressivamente mais larga, ao invés de ficar mais profunda. Para tornar o processo ainda mais eficiente, pode-se adicionar uma convolução extra 1x1, computacionalmente mais barata, antes das camadas 3x3 e 5x5. A Figura 10 mostra a arquitetura de uma rede Inception-v1.

Figura 10 – Arquitetura da Inception v-1.



Fonte: TowardsDataScience (2019).

O *design* do módulo Inception inicial é conhecido como GoogLeNet, ou Inception-v1. Variações adicionais têm sido desenvolvidas, reduzindo problemas tais como o Desvanecimento do Gradiente (*Vanishing Gradient*), que ocorre em redes neurais com técnicas de aprendizado fundamentadas em gradiente e retropropagação (*backpropagation*). Nestes métodos, cada um dos "pesos" da rede neural é atualizado de maneira proporcional à derivada parcial da função de erro em relação ao peso atual em cada iteração do treinamento. Porém, eventualmente, o gradiente será muito pequeno, impedindo efetivamente o peso de mudar de valor. Na pior das hipóteses, isso pode inibir integralmente a rede neural de executar treinamentos adicionais.

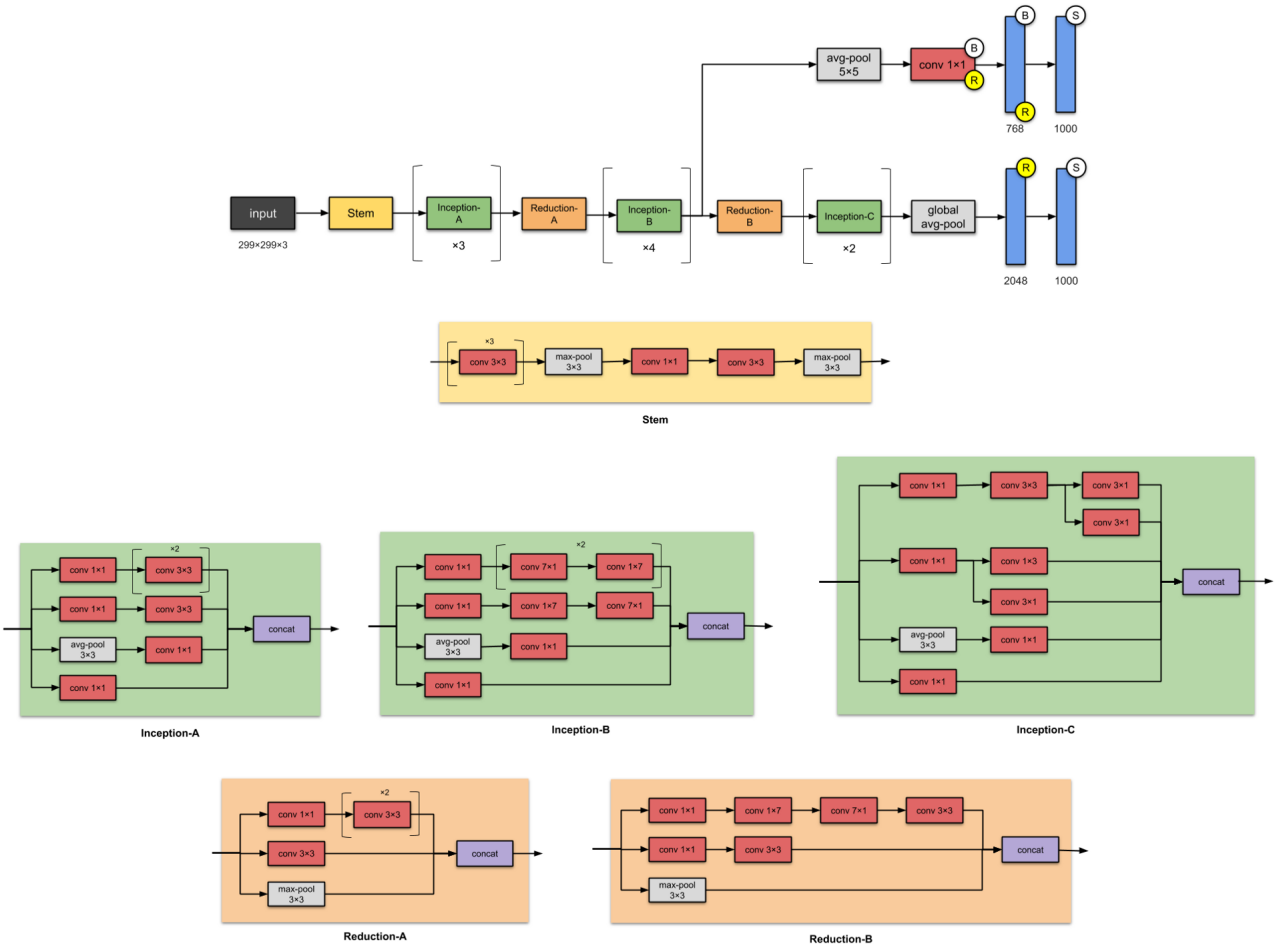
Constantes evoluções levaram à criação de diferentes versões. Inception-v2 e Inception-v3 foram apresentadas por Szegedy *et al.* (2015) com atualizações que aumentaram a acuracidade e reduziram a complexidade computacional.

A Inception-v2 considerou que redes neurais funcionam melhor quando as convoluções não alteram drasticamente as dimensões dos dados de entrada, já que reduzir as dimensões demasiadamente, tornando o módulo mais profundo, pode ocasionar perda de informação, conhecida como "gargalo representacional" (*representational bottleneck*). Assim, os bancos de filtros no módulo foram expandidos (mais amplos ao invés de mais profundos) e, com o uso de métodos de fatoração inteligentes, as convoluções ficaram mais eficientes. A Inception-v2 fatorou convoluções 5x5 em duas convoluções 3x3 para aumentar a velocidade computacional. Embora

isso possa parecer contraintuitivo, uma convolução 5x5 é 2,78 vezes mais cara do que uma convolução 3x3. Logo, o empilhamento de duas convoluções 3x3 eleva o nível de performance. Além disso, houve fatoração de convoluções com filtro de tamanho nxn para uma combinação de convoluções 1xn e nx1; por exemplo: uma convolução 3x3 é equivalente a primeiro executar uma convolução 1x3, e depois executar uma convolução 3x1 na saída, o que torna o método 33% mais barato do que depender de uma única convolução 3x3.

Quando a Inception-v2 surgiu, vários experimentos foram feitos e os autores registraram diversos ajustes bem sucedidos. Inception-v3 é a rede que incorpora esses ajustes, relativos ao otimizador (*RMSProp Optimizer*), à função de perda (*Label Smoothing* para prevenção de *overfitting*), à adição de normalização em lote (*BatchNorm*) nas camadas auxiliares e à fatoração de convoluções 7x7. A Figura 11 ilustra a arquitetura de uma rede Inception-v3.

Figura 11 – Arquitetura da Inception v-3.

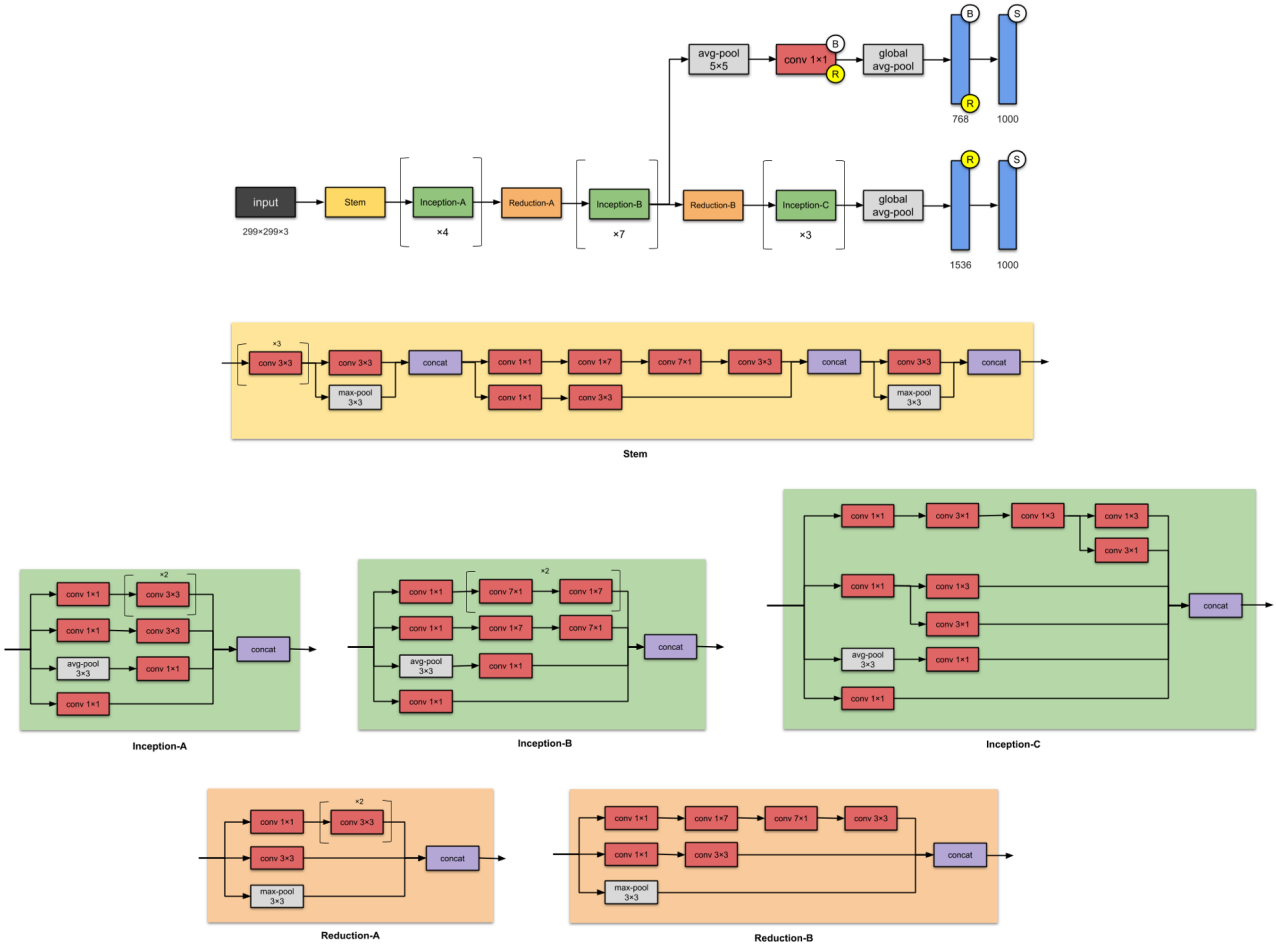


Fonte: TowardsDataScience (2019).

Inception-v4 e Inception-ResNet foram introduzidas por Szegedy *et al.* (2016) com o intuito de tornar os módulos mais uniformes e incrementar a performance. O conjunto de operações iniciais (*stem*) executadas antes da introdução dos blocos do módulo foi modificado na

Inception v-4, que possui três módulos principais: A, B e C. A Inception-v4 introduziu Blocos de Redução (*Reduction Blocks*) especializados, que são usados para mudar a largura e a altura da grade (*grid*). A Figura 12 ilustra a arquitetura de uma rede Inception v-4.

Figura 12 – Arquitetura da Inception v-4.

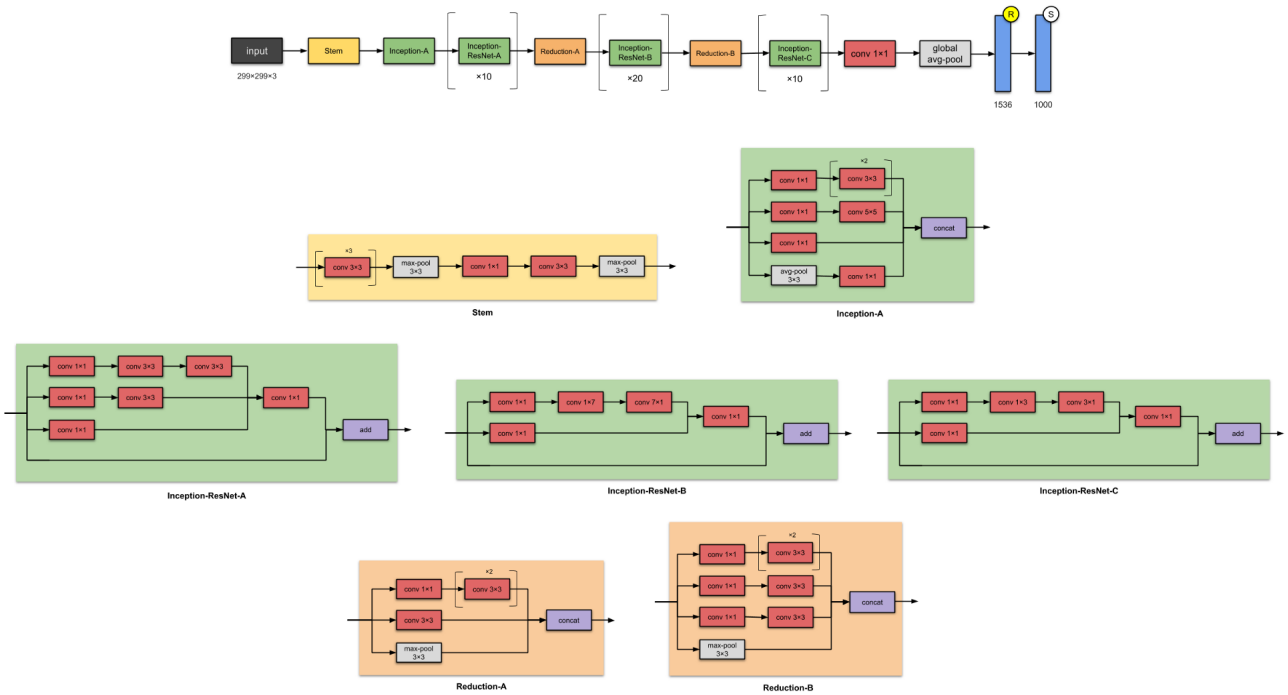


Fonte: TowardsDataScience (2019).

Inspirada pela performance da ResNet, um módulo Inception híbrido foi proposto. Há duas sub-versões da Inception-ResNet: v1 (custo computacional similar à Inception-v3) e v2 (custo computacional similar à Inception-v4). Seus *stems* são diferentes, mas ambas têm a mesma estrutura de módulos A, B e C e Blocos de Redução. A configuração dos hiperparâmetros é diferente. Sob a premissa de introduzir conexões residuais que adicionam a saída da operação convolucional do módulo à entrada, são usadas convoluções 1x1 após as convoluções originais para combinar os tamanhos das profundidades (que são incrementadas após as convoluções). As operações de agrupamento (*pooling*) no interior dos módulos Inception principais foram substituídas em favor das conexões residuais, embora ainda possam ser encontradas nos Blocos de Redução. Foi constatado que modelos de Inception-ResNet alcançaram maiores acúracias, empregando um menor número de *epochs*, as quais representam uma passagem completa por

todas as imagens no conjunto de dados. A Figura 13 ilustra a arquitetura de uma rede Inception-ResNet-v2.

Figura 13 – Arquitetura da Inception-ResNet-v2.



Fonte: TowardsDataScience (2019).

2.2.4 ResNeXt

Em 2016, Xie *et al.* (2016) apresentaram uma arquitetura de rede denominada ResNeXt, aumentando o número de torres paralelas (“cardinalidade”) dentro de um módulo. A cardinalidade consiste, portanto, em uma dimensão adicional sobre a largura e a profundidade da rede e define o tamanho do conjunto de transformações. Assim, com uma cardinalidade de 32, as mesmas transformações são aplicadas 32 vezes, e o resultado é agregado ao final.

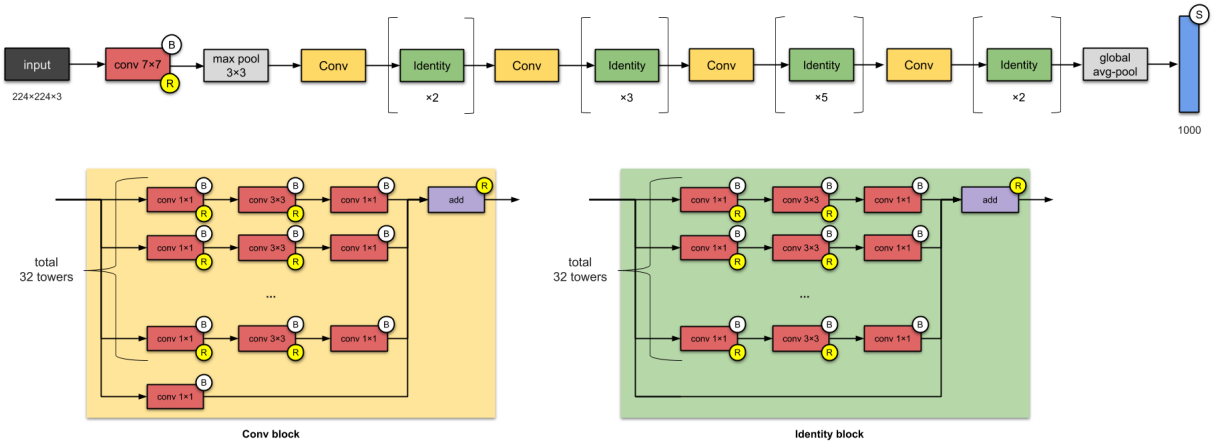
Apesar das ótimas performances obtidas por redes VGG, ResNet e Inception, estas ainda enfrentam algumas limitações. Seus modelos são adequados para vários conjuntos de dados, mas devido à existência de múltiplos hiperparâmetros e cálculos envolvidos, adaptá-los a novos conjuntos de dados não é uma tarefa trivial. Para superar tais problemas, a estratégia de repetição da ResNet foi combinada com a estratégia de divisão-transformação-mesclagem da rede Inception.

A estratégia de repetição da ResNet é utilizada para construir blocos em que as camadas residuais são adicionadas à entrada, enquanto a estratégia de divisão-transformação-mesclagem da Inception é utilizada para criar grupos de camadas com diferentes canais de saída, permitindo

que a rede capture características diferentes e mais complexas ao longo de sua profundidade, aumentando a capacidade de reconhecimento e performance.

A arquitetura básica da ResNeXt, ilustrada na Figura 14, é definida por duas regras: (i) se os blocos produzem os mesmos mapas de espaço dimensional, eles compartilham o mesmo conjunto de hiperparâmetros (largura e tamanho dos filtros), e (ii) se o mapa de espaço dimensional é reduzido por um fator de 2, a largura do bloco é multiplicada por um fator de 2.

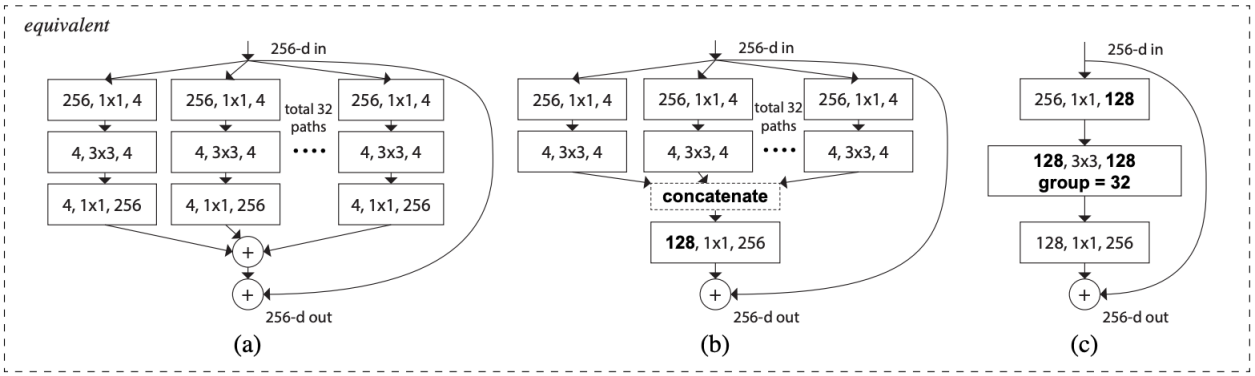
Figura 14 – Arquitetura da ResNeXt.



Fonte: TowardsDataScience (2019).

A estrutura de rede na Figura 15 explica o que uma convolução agrupada significa e como ela supera as outras duas estruturas de rede.

Figura 15 – ResNeXt Building Blocks: **(a)** Transformações residuais agregadas; **(b)** Bloco equivalente àquele apresentado em (a), implementado como concatenação inicial; **c)** Bloco equivalente àqueles apresentados em (a) e (b), implementado como convoluções agrupadas. Notações em **negrito** destacam as mudanças em cada reformulação. Uma camada é indicada como número de canais de entrada, tamanho do filtro e número de canais de saída.



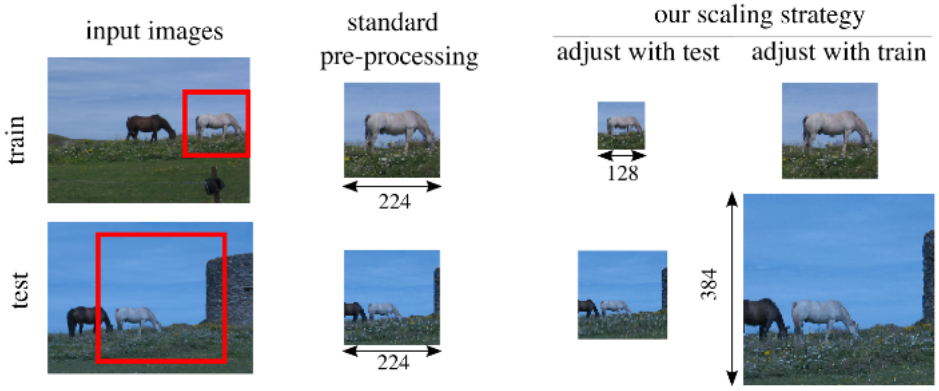
Fonte: Xie *et al.* (2016).

2.2.5 FixRes, EfficientNet e FixEfficientNet

Em 2019, Touvron *et al.* (2019), membros do time de AI do Facebook, apresentaram uma estratégia (FixRes) de dimensionamento de imagem que busca otimizar o desempenho do classificador. É motivada pela observação de que o aumento de dados induz uma discrepância significativa entre o tamanho dos objetos vistos pelo classificador durante o período de treinamento e durante o período de teste: na verdade, uma resolução mais baixa durante o treinamento melhora a classificação durante o teste. FixRes é uma estratégia simples para otimizar o desempenho do classificador, que emprega diferentes resoluções de treinamento e de teste.

De fato, o emprego de transformações de treinamento típicas tais como *RandomResizedCrop* resulta em objetos em imagens de treinamento aparecendo maiores do que são em relação ao conjunto de testes, conforme ilustrado na Figura 16. Diante disso, os seguintes ajustes foram empregados: (a) calibrar os tamanhos dos objetos ajustando o tamanho da colheita (*crop size*) e (b) ajustar as estatísticas antes do agrupamento espacial (*spatial pooling*).

Figura 16 – Seleção das regiões da imagem fornecidas à rede no momento do treinamento e do teste, com aumento de dados típico. A região vermelha de classificação é reamostrada como uma colheita (*crop*) que alimenta a rede neural. Para objetos que têm tamanhos semelhantes na imagem de entrada, como o cavalo branco, os processos padrões de *augmentation* normalmente os tornam maiores durante o período de treinamento quando comparados ao período de teste (segunda coluna). Para contrariar este efeito, *FixRes* reduz a resolução do conjunto de treinamento ou aumenta a resolução do conjunto de testes (terceira e quarta colunas). O cavalo tem então o mesmo tamanho no treinamento e no teste, exigindo menos invariância de escala para a rede neural.



Fonte: Touvron *et al.* (2019).

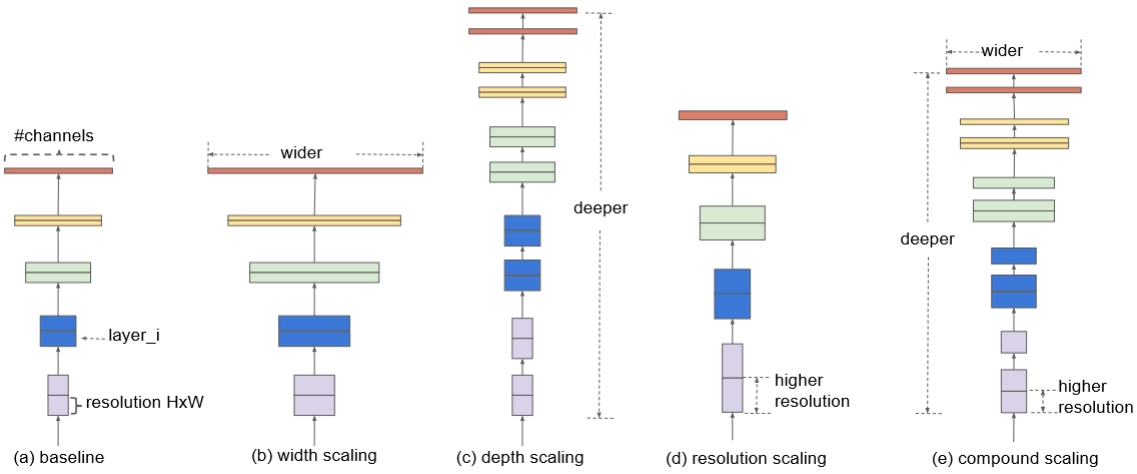
Touvron *et al.* (2019) concentraram-se principalmente no uso da arquitetura ResNet-50 como *baseline* para seus experimentos, inclusive adaptando-a através de treinamento com resolução 224x224 e teste com resolução 320x320 para obter uma *top-1 accuracy* de 79,8% no *dataset* ImageNet.

Já a EfficientNet, introduzida por Tan e Le (2019), membros do time de AI do Google, é

capaz de estabelecer uniformemente todas as dimensões de profundidade, largura e resolução através de um conjunto de coeficientes de dimensionamento fixos. Se for preciso usar 2^N mais recursos computacionais, por exemplo, então somente incrementamos a profundidade da rede por α^N, a largura por β^N, e o tamanho da imagem por γ^N, em que α, β e γ são coeficientes constantes estabelecidos por uma *grid search* no modelo original.

O método de escalonamento composto (*compound scaling method*) é justificado pela intuição de que se a imagem de entrada for maior, são necessárias mais camadas para aumentar o campo receptivo e mais canais para identificar os detalhes da imagem com maior precisão. A Figura 17 ilustra a diferença entre o método de escalonamento proposto pela EfficientNet e os métodos convencionais.

Figura 17 – *Model Scaling*: (a) é um exemplo base; (b) a (d) são escalonamentos convencionais que incrementam somente uma das dimensões da rede: largura, profundidade, ou resolução; (e) é o método de escalonamento composto proposto pela EfficientNet, que dimensiona uniformemente todas as três dimensões com uma proporção fixa.



Fonte: Tan e Le (2019).

A combinação das técnicas FixRes e EfficientNet resultou na apresentação da FixEfficientNet por Touvron *et al.* (2020). Recentemente, este modelo alcançou, com 480 milhões de parâmetros, excelentes resultados no *dataset* da ImageNet, com uma *top-1 accuracy* de 88,5% e uma *top-5 accuracy* de 98,7%.

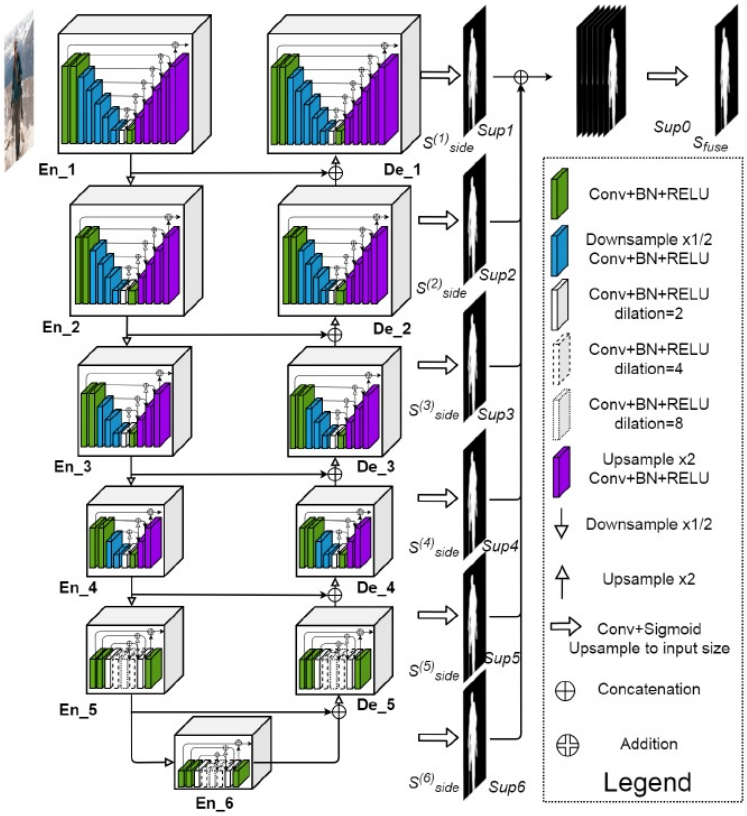
2.2.6 U²-Net

A segmentação de uma imagem, resultando na obtenção mais adequada da região de interesse, pode favorecer a precisão de sua classificação através de Redes Neurais Convolucionais. Neste contexto, podemos destacar a arquitetura de rede profunda U^2-Net, cuja estrutura em U aninhada de dois níveis, com blocos RSU (*ReSidual U-blocks*), permite que a rede capture com maior riqueza de detalhes características locais e globais de camadas rasas e profundas,

independentemente das resoluções. Para tanto, aplica-se o conceito de extração de características em múltiplas escalas (*multi-scale feature extraction*), as quais são extraídas de mapas de características (*feature maps*) gradualmente reduzidos e codificados em mapas de características de alta resolução através de *upsampling* progressivo, concatenação e convolução. Este processo ajuda a melhor identificar a região de interesse e remover o *background*.

A arquitetura ilustrada na Figura 18 demonstra uma estrutura *Encoder-Decoder* em U, em que cada estágio apresenta novos blocos U residuais.

Figura 18 – Arquitetura U^2-Net.



Fonte: Qin *et al.* (2020).

CAPÍTULO

3

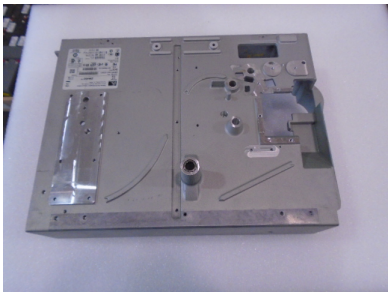
MATERIAIS E MÉTODOS

Neste capítulo, serão discutidos os materiais e métodos utilizados neste trabalho. Os detalhes abrangem as características da base de dados, como as classes e as quantidades de imagens, bem como as ferramentas e recursos computacionais utilizados durante o treinamento dos modelos de Redes Neurais Convolucionais. Também serão descritas as abordagens de treinamento adotadas, incluindo a divisão dos conjuntos de treinamento e de teste, a técnica de balanceamento para ajustar as quantidades de amostras de cada classe, o processo de *data augmentation* e os demais hiperparâmetros configurados para implementar os modelos de redes neurais. Além disso, apresentam-se as métricas de avaliação utilizadas para medir o desempenho dos modelos propostos.

3.1 Base de Dados

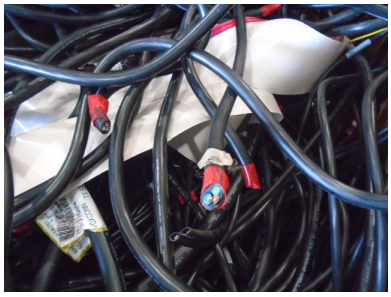
Para executar o trabalho proposto nesta pesquisa, foram obtidas 1.670 imagens de resíduos eletroeletrônicos encontrados na operação de *Scrap* e *Waste Disposal* de uma grande empresa da área de Tecnologia da Informação, capturadas através de uma câmera Sony, modelo DSC-W800, com 20,1 *megapixels*. Tais imagens foram previamente rotuladas em 11 classes: (1) Alumínio ou Ferro, (2) Baterias, (3) Cabos ou Fios, (4) Cobre, (5) Conectores, (6) Fitas Magnéticas, (7) Papel ou Papelão, (8) Placas de Circuito Impresso, (9) Plástico, (10) *Ribbon* ou *Toner* de Impressoras e (11) Tubos ou Telas. A Figura 19 ilustra os tipos de resíduos em questão.

Figura 19 – Figuras de resíduos eletroeletrônicos oriundos da operação de *Scrap* e *Waste Disposal* de uma grande empresa da área de Tecnologia da Informação.

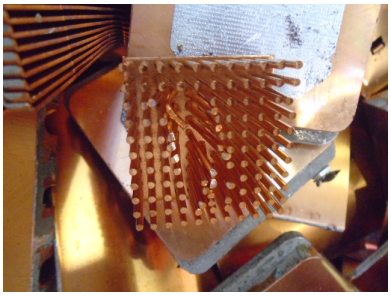


(1) Alumínio ou Ferro

(2) Baterias



(3) Cabos ou Fios



(4) Cobre

(5) Conectores

(6) Fitas Magnéticas

(7) Papel ou Papelão

(8) Placas de Circuito Impresso

(9) Plástico

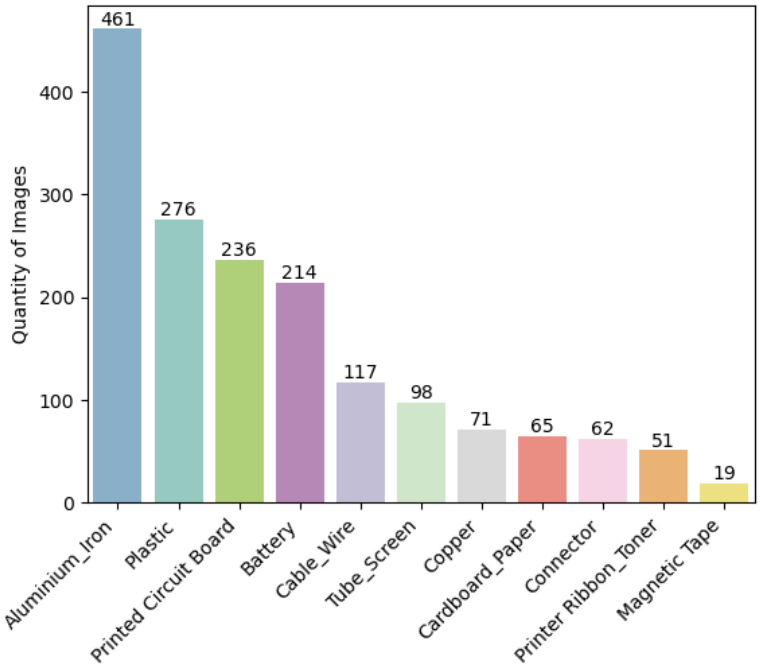
(10) *Ribbon* ou *Toner* de Impressoras

(11) Tubos ou Telas

Fonte: Elaborada pelo autor.

A Figura 20 exibe as quantidades de imagens disponibilizadas para cada classe.

Figura 20 – Classes e quantidades de imagens.



Fonte: Elaborada pelo autor.

Segundo Møgaard (2017), aplicar Redes Neurais Convolucionais para classificação de resíduos eletrônicos traz alguns desafios, tais como definir o tamanho do *dataset* (tipicamente, o treinamento requer centenas de imagens anotadas), o que pode ser bastante custoso de se conseguir. A maior parte dos esforços para desenvolver CNNs tem focado em imagens RGB, pois estão prontamente disponíveis em centenas de bases de dados de imagens anotadas.

Para contornar o problema de obter dados suficientes, pode-se adotar a combinação de duas técnicas: transferência de aprendizado (*transfer learning*) e aumento de dados (*data augmentation*). A transferência de aprendizado é uma técnica que tenta alavancar o conhecimento obtido em um problema e transferi-lo para outro problema relacionado. Na prática, isso pode ser feito através do treinamento de uma CNN em um *dataset* robusto (como ImageNet para objetos naturais) e então refinar a rede treinada no problema-alvo. Isto mostrou-se um jeito viável de aplicar com sucesso Redes Convolucionais a problemas com *training sets* pequenos.

Já o aumento de dados é usado para incrementar o número efetivo de imagens de um *dataset* pequeno através de um número diferente de transformações de imagens. No contexto da classificação de resíduos eletrônicos, os objetos no fluxo de resíduos podem ter qualquer orientação e podem ser obscurecidos de várias maneiras durante o processo de separação. Portanto, os dados de treinamento (*training data*) podem ser aumentados através de inúmeras transformações que podem emular estes tipos de alterações de imagens.

Além disso, para mitigar o problema de *class imbalance*, ou seja, a presença de classes

com número significativamente mais alto de exemplos no *dataset*, podem-se adotar métodos de balanceamento, tais como *oversampling* que, em sua versão básica, consiste em replicar aleatoriamente determinadas *samples* de classes minoritárias. Geralmente, o processo de *oversampling* não ocasiona *overfitting* em CNNs, ao contrário do que pode ocorrer com outros modelos clássicos de *Machine Learning* (BUDA; MAKI; MAZUROWSKI, 2018).

3.2 Técnicas e Recursos

Todos os modelos de CNNs foram treinados através da plataforma Google Colab, utilizando uma GPU de 40 GB, em ambiente de execução com RAM de 87,2 GB. Foi empregada a linguagem de programação Python com a biblioteca fast.ai, construída sobre o *framework* PyTorch.

Desde sua introdução, em 2017, PyTorch tornou-se largamente aplicado em pesquisas acadêmicas e conferências de alto nível. Cabe frisar que PyTorch funciona melhor como uma biblioteca *low-level*, provendo as operações básicas para funcionalidades *high-level*, que podem ser implementadas através da biblioteca fast.ai, pois esta é única no fornecimento de um *software* de camadas profundas, tal como descrito em Howard e Gugger (2020b).

3.3 Metodologia Proposta

O método proposto visa a implementação de diferentes tipos de arquiteturas de CNNs, bem como distintas abordagens de treinamento, com o intuito de comparar seus resultados tanto entre si como entre os trabalhos que envolvem classificação de imagens de resíduos eletrônicos.

Neste sentido, foram construídos 18 modelos de CNNs, conforme descritos na Tabela 1. Antes do processo de *transfer learning*, todos foram pré-treinados na base de dados ImageNet, com exceção daqueles com prefixo “fix”, que foram implementados a partir dos modelos *resnet34*, *resnet50*, *resnet101*, *resnet152* e *efficientnet_b0b* desenvolvidos durante este trabalho e que, por sua vez, já haviam sido originalmente pré-treinados na ImageNet.

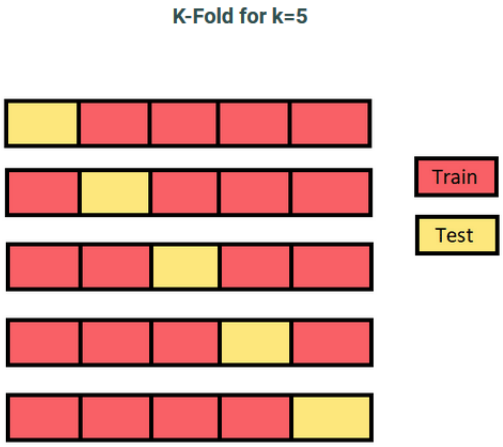
Tabela 1 – Modelos de Redes Neurais Convolucionais com seus respectivos números de parâmetros e tamanhos de imagem de entrada.

#	Arquitetura	Parâmetros (M)	Tamanho de Entrada
1	vgg16	15,2	224
2	resnet34	21,8	224
3	resnet50	25,6	224
4	resnet101	44,6	224
5	resnet152	60,3	224
6	resnext50_32x4d	25,1	224
7	resnext101_32x4d	44,2	224
8	inception_v3	23,9	299
9	inception_v4	42,7	299
10	inceptionresnet_v1	24,4	299
11	inceptionresnet_v2	55,9	299
12	efficientnet_b0b	5,3	224
13	efficientnet_b3b	12,3	300
14	fixresnet34	21,8	320
15	fixresnet50	25,6	320
16	fixresnet101	44,6	320
17	fixresnet152	60,3	320
18	fixefficientnet_b0b	5,3	320

Para segregar o conjunto total de amostras em dois subconjuntos mutuamente exclusivos, ou subconjuntos de treinamento e de teste, foi adotada uma técnica de Validação Cruzada (*Cross Validation*) através da aplicação do método *Stratified K-Fold*, que é uma variante do método *K-Fold*. Costuma-se utilizar a Validação Cruzada para verificar a habilidade de generalização de um modelo a partir de um conjunto de dados, isto é, definir como os resultados serão generalizados para o conjunto de dados independentes (KOHAVI, 1995).

O método *Stratified K-Fold* divide o conjunto total de amostras em k subconjuntos (*folds*) mutuamente exclusivos do mesmo tamanho. Posteriormente, um dos k subconjuntos é selecionado para testes e os demais k-1 são usados como dados de treinamento. Repete-se então este procedimento por k vezes, de modo que a utilização de cada subconjunto como conjunto de testes ocorre uma única vez. A partir disso, os k resultados são agregados para originar uma estimativa de erro específica, conforme ilustrado na Figura 21.

Figura 21 – Método *K-Fold*.



Fonte: Subramanian (2019).

Cabe ressaltar que o método *Stratified K-Fold* mantém a mesma proporção de classes em cada subconjunto, para que o processo de aprendizado não fique enviesado.

Para os experimentos referentes a este projeto, foram utilizados 5 subconjuntos (*folds*), de modo que o conjunto de treinamento corresponde a 80% do *dataset*, enquanto o conjunto de testes corresponde a 20%, em cada iteração. A acuracidade de cada *fold* é obtida por:

$$accuracy\_fold\_i = \frac{number\ of\ correctly\ classified\ samples\ in\ fold\ i}{total\ number\ of\ samples\ in\ fold\ i}, \tag{3.1}$$

em que i é o número da *fold*.

Para calcular o número de amostras classificadas corretamente em cada *fold*, pode-se usar a seguinte equação:

$$number\ of\ correctly\ classified\ samples\ in\ fold\ i = \sum_{j=1}^{n} (y_j == y_{pred_j}), \tag{3.2}$$

em que y_j é o rótulo verdadeiro da j-ésima amostra e y_{pred_j} é o rótulo previsto da j-ésima amostra.

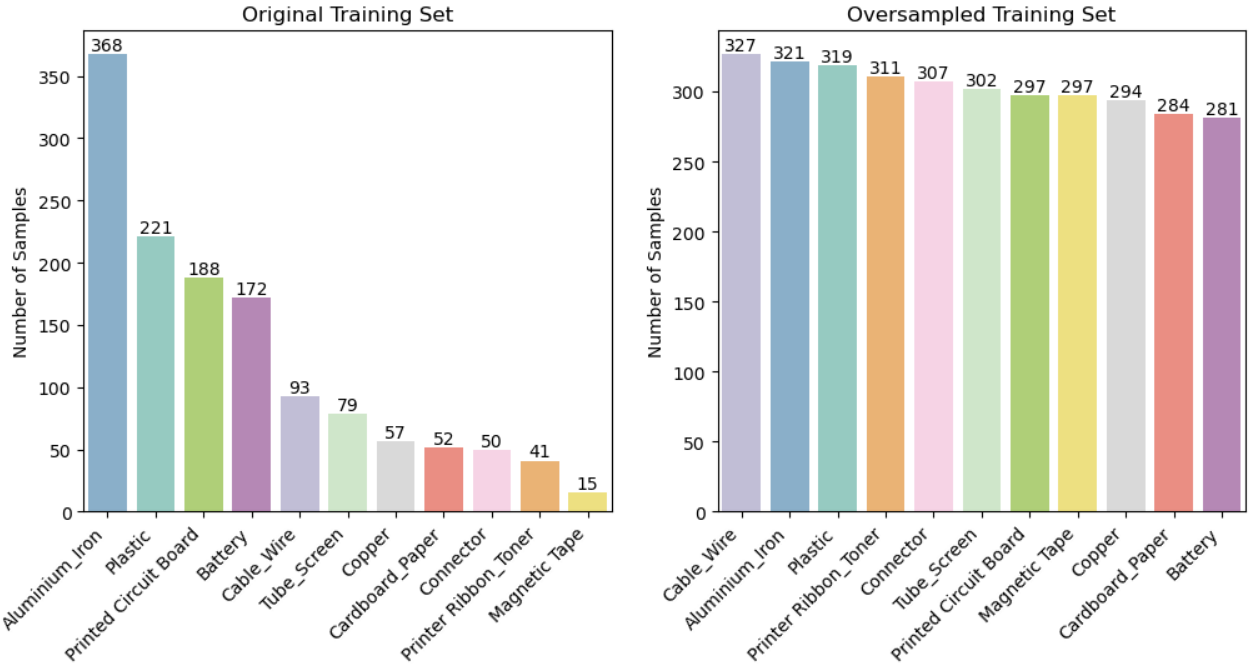
Para calcular a média em todas as *folds* e obter a acuracidade geral da Validação Cruzada:

$$cross\_validation\_accuracy = \frac{\sum_{i=1}^{k} accuracy\_fold\_i}{k}, \tag{3.3}$$

em que $accuracy\_fold\_i$ é a acurácia obtida para a i-ésima *fold* e $\sum_{i=1}^{k}$ é a somatória das acurácias (variando de 1 até k *folds*), que é então dividida pelo número total de *folds*.

Diante do problema de *class imbalance* encontrado no *dataset*, foi aplicada a técnica de *oversampling* para ajuste das amostras. Neste caso, optou-se por duplicar o número de amostras do *dataset* original, redistribuindo-as no conjunto de treinamento segundo pesos previamente calculados para retornar um número mais igualitário de índices correspondentes a cada classe. Os números de amostras de cada classe no conjunto de treinamento, antes e depois do ajuste proposto, podem ser observados na Figura 22.

Figura 22 – Comparação entre as quantidades de amostras no conjunto de treinamento antes e depois do processo de *oversampling*.



Fonte: Elaborada pelo autor.

Além disso, como estratégia para *data augmentation*, as seguintes variações foram implementadas: *flipping*, *rotation*, *zooming*, *lighting* e *warping* em cada *mini-batch* através do método *aug_transforms*, da bilioteca fast.ai.

O conceito de *mini-batch* refere-se à possibilidade de calcular a média da função de perda (*loss function*) para um grupo específico de itens ao mesmo tempo. Desse modo, um *batch size* maior resulta numa estimativa mais precisa e estável dos gradientes oriundos da função de perda, embora aumente o tempo de processamento e reduza o número de *mini-batches* processados por época (*epoch*). Como foi utilizada uma GPU de 40 GB, procedeu-se com *batches* de 64 itens em todos os experimentos sem que fosse ocasionado problema de memória insuficiente.

A função de perda determina o cálculo dos gradientes, que medem, para cada peso, como estes mudariam a perda para promover melhorias no modelo. Neste caso, optou-se por *Cross-Entropy Loss*, que funciona inclusive quando a variável dependente tem mais do que duas categorias e suaviza o processo de treinamento do modelo, uma vez que a linearidade de

seus gradientes previne a ocorrência de saltos súbitos ou aumentos exponenciais (HOWARD; GUGGER, 2020a). Cada probabilidade de classe prevista é comparada com a saída desejada 0 ou 1 da classe real e calcula-se uma pontuação/perda que penaliza a probabilidade com base na distância entre ela e o valor real esperado. A penalidade é de natureza logarítmica, resultando em uma grande pontuação para grandes diferenças próximas a 1 e uma pequena pontuação para pequenas diferenças tendendo a 0.

A fórmula usada para calcular a *Cross-Entropy Loss*, em um problema de classificação multiclasse, é:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(p_{i,c}), \tag{3.4}$$

em que N é o número de amostras, C é o número de classes, $y_{i,c}$ é uma variável indicadora que assume o valor 1 se a amostra i pertencer à classe c e 0 caso contrário, e $p_{i,c}$ é a probabilidade prevista de que a amostra i pertença à classe c, para $c = 1,...,C$.

Para garantir que todas as imagens tivessem as mesmas dimensões, alcançando um processamento mais eficiente na GPU, foi inserido no parâmetro *item_tfms* do objeto *DataBlock* o método *RandomResizedCrop*, que recorta uma parte aleatória da imagem e redimensiona para um determinado tamanho. Dessa forma, em cada época, seleciona-se aleatoriamente uma parte diferente de cada imagem, permitindo ao modelo aprender a reconhecer diferentes características. Optou-se por um tamanho de 512 *pixels* com *min_scale* correspondente a 70%, para enquadrar as variações obtidas com *data augmentation* sem criar zonas vazias (*empty_zones*).

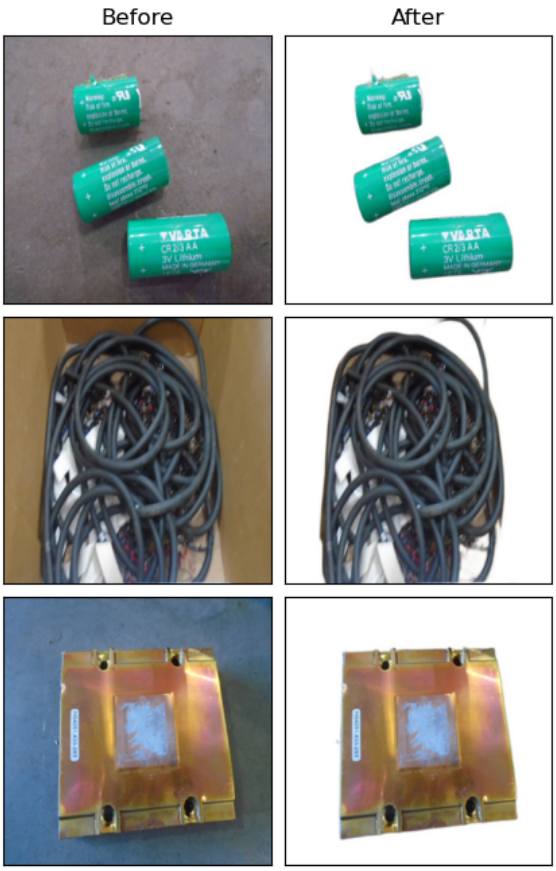
Vale frisar que os tamanhos de imagem de entrada escolhidos para as transformações em cada *batch* foram selecionados de acordo com a documentação relativa ao pacote pytorchcv. Para os modelos *vgg16*, *resnet34*, *resnet50*, *resnet101*, *resnet152*, *resnext50_32x4d*, *resnext101_32x4d* e *efficientnet_b0b*, foi considerado o padrão de 224 *pixels*. Já para os modelos *inception_v3*, *inception_v4*, *inceptionresnet_v1* e *inceptionresnet_v2*, foram considerados 299 *pixels*. Para o modelo *efficientnet_b3b*, 300 *pixels*. Ademais, para modelos implementados através da estratégia de correção da resolução das imagens nos conjuntos de treinamento-teste (*fixresnet34*, *fixresnet50*, *fixresnet101*, *fixresnet152* e *fixefficientnet_b0b*), foi adotado um tamanho de 320 *pixels* ao realizar o processo de *fine-tuning* das últimas camadas.

Antes de treinar os modelos, também foi implementado um processo de normalização, já que a maioria das imagens e bibliotecas de visão computacional usam valores entre 0 e 255 *pixels*, ou entre 0 e 1, impedindo que os dados de entrada estejam normalizados, ou seja, tenham média igual a zero e desvio-padrão igual a 1. Neste sentido, a bilioteca fast.ai permite a adição da transformação *Normalize*, que age no *mini-batch* inteiro. Isso pode ser feito utilizando-se média e desvião-padrão já existentes na base de dados ImageNet, com a inserção do comando *Normalize.from_stats(*imagenet_stats)* no parâmetro *batch_tfms* do objeto *DataBlock*. O processo de normalização é especialmente importante quando modelos pré-treinados são

considerados.

Ainda anteriormente ao treinamento dos modelos, observa-se que a grande maioria das imagens passou por um processo de segmentação de *background* através da arquitetura de rede profunda denominada U^2-Net. A Figura 23 ilustra algumas amostras antes e depois da segmentação.

Figura 23 – Exemplos de amostras antes e depois do processo de segmentação de *background*.

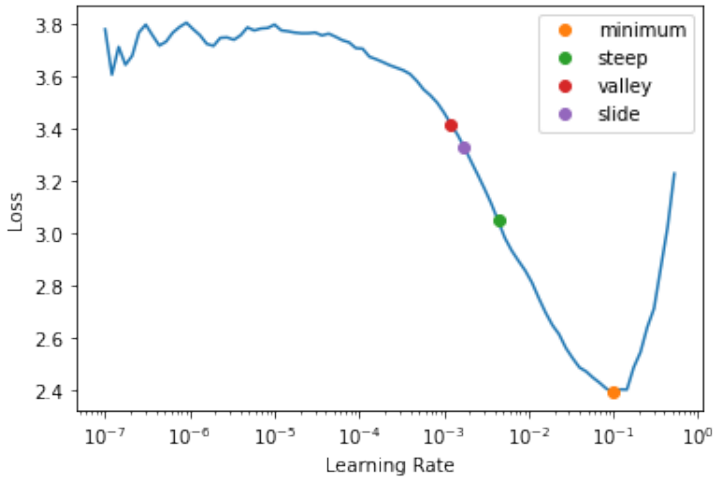


Fonte: Elaborada pelo autor.

A maioria dos modelos foi treinada por 10 *epochs* com as camadas iniciais "congeladas" (*freeze*) (ou seja, foram utilizados somente os parâmetros de treinamento das *randomly added final layers*, isto é, das camadas finais adicionadas aleatoriamente) e, posteriormente, tais modelos foram ajustados por mais 10 *epochs* após o "descongelamento" (*unfreeze*) de todas as camadas. Antes de cada "congelamento" / "descongelamento", foram determinadas as taxas de aprendizado (*learning rates*) apropriadas. É importante frisar que os modelos *fixresnet34*, *fixresnet50*, *fixresnet101*, *fixresnet152* e *fixefficientnet_b0b* foram treinados somente por 10 *epochs* com as camadas iniciais "congeladas" (*freeze*), utilizando os modelos *resnet34*, *resnet50*, *resnet101*, *resnet152* e *efficientnet_b0b* previamente treinados durante este projeto, para implementar o conceito de correção da resolução das imagens, conforme explicado por Mashru (2021) no cenário de *transfer learning using fast.ai*.

Assegurar-se de que temos a *learning rate* correta é um dos fatores mais importantes para o processo de treinamento, já que taxas muito baixas podem tomar muito tempo, ocasionando ainda problemas de *overtfitting*, enquanto taxas muito altas podem ultrapassar a perda mínima (*minimum loss*) no mecanismo de Gradiente Descendente (*Gradient Descent*). Para mitigar este problema, foi aplicado o processo de *learning rate finder* proposto por Smith (2015), segundo o qual deve-se iniciar com uma *learning rate* muito pequena, incrementando a cada *mini-batch* até que a perda passe a piorar ao invés de melhorar. A Figura 24 demonstra a relação entre as taxas de aprendizado e as perdas, obtidas a partir do método *lr_find*, da biblioteca fast.ai, após o "congelamento" (*freeze*) das camadas iniciais.

Figura 24 – Gráfico com a relação entre as perdas e as taxas de aprendizado, conforme obtidas pelo método *lr_find*, da biblioteca fast.ai. O eixo x do gráfico representa a taxa de aprendizado e o eixo y representa a perda. Conforme a taxa de aprendizado aumenta, a perda inicialmente diminui, mas em algum ponto, a perda começa a aumentar novamente. O ponto onde a perda começa a aumentar novamente é denominado *lr_min*.



Fonte: Elaborada pelo autor.

Dentre os pontos destacados no gráfico, optou-se por selecionar a opção *valley*, pois ao lado da opção *slide*, sugere uma taxa muito mais apropriada, sem a necessidade de efetuar uma interpretação gráfica (MUELLER, 2021).

Já para a etapa do treinamento em que as camadas foram "descongeladas", aplicou-se o conceito de *discriminative learning rates*, segundo o qual taxas de aprendizado menores são usadas para as camadas iniciais da rede neural e taxas maiores para as camadas finais (especialmente aquelas adicionadas aleatoriamente). Trata-se de uma ideia proposta por Yosinski *et al.* (2014), tendo em vista o fato de que as camadas iniciais aprendem conceitos mais simples, tais como detectores de borda e gradiente, enquanto as camadas finais aprendem conceitos mais complexos (por exemplo: "olhos" e "boca"), que podem não ser úteis para a tarefa em questão. Logo, faz sentido deixar as camadas finais ajustarem-se mais rapidamente do que as camadas iniciais.

Em ambos os cenários (camadas "congeladas" ou "descongeladas"), foi empregado o método *fit_one_cycle(n_epoch, lr_max)*, que, através da estratégia *1cycle policy*, inicia com uma taxa baixa, gradualmente aumentando-a na primeira parte do treinamento, e depois gradualmente decrescendo-a na parte final. Para o cenário com camadas iniciais "congeladas", aplicou-se uma *maximum learning rate (lr_max)* correspondente à opção *valley*, do método *lr_find*, conforme descrito acima. Para o cenário com as camadas "descongeladas", aplicou-se a estratégia de *discriminative learning rates* através do parâmetro *lr_max* = *slice(start, end)*, o que significa que o primeiro valor passado (*start*) é a taxa de aprendizado na camada mais inicial da rede neural e o segundo valor (*end*) é a taxa de aprendizado na camada final; as camadas intermediárias utilizam taxas que são multiplicativamente equidistantes no intervalo considerado.

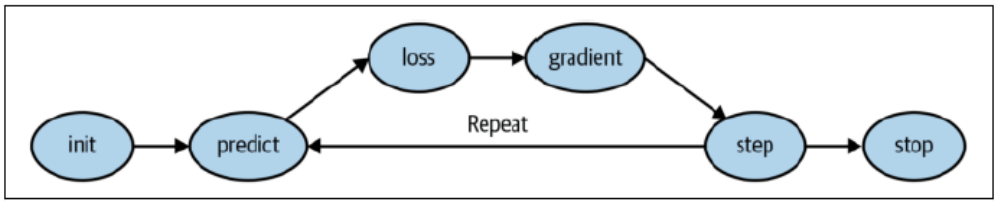
O processo de otimização de redes neurais é, de fato, uma tarefa complexa, de modo que, tradicionalmente, evitaram-se as dificuldades do processo geral de otimização através do desenvolvimento cuidadoso da função objetivo e de suas restrições para garantir que o problema seja convexo (GOODFELLOW; BENGIO; COURVILLE, 2016). A maioria dos algoritmos de *Deep Learning* empregam uma técnica denominada *Stochastic Gradient Descent* (SGD), que inicia a partir de um ponto aleatório de uma função e move sua inclinação (*slope*) em etapas até atingir o ponto mais baixo dessa função. Ou seja, o objetivo é minimizar a função de perda (*loss function*), calculando iterativamente a derivada da função em dado ponto (gradiente; inclinação) e, posteriormente, determinando o passo (*step*) na direção oposta à do gradiente. O tamanho do passo (*step size*) é calculado através da multiplicação entre o gradiente e a taxa de aprendizado (*learning rate*). Este processo é executado com respeito a cada peso (*weight*) do modelo em questão, isto é, ocorre uma atualização dos pesos a cada iteração, dada por:

$$new\_weight = weight - lr * weight.grad, \tag{3.5}$$

em que lr corresponde à *learning rate*, *weight* corresponde ao peso anterior, $weight.grad$ é o gradiente da função de perda com respeito ao peso anterior e *new_weight* corresponde ao peso atualizado.

Dessa forma, os passos ilustrados na Figura 25 são fundamentais para o treinamento de qualquer modelo de *Deep Learning*.

Figura 25 – Processo de Gradiente Descendente.



Fonte: Howard e Gugger (2020a).

Ocorre que a atualização do peso em um dado momento (t) é governada pela taxa de aprendizado e pelo gradiente naquele momento apenas. Não leva-se em consideração as etapas anteriores realizadas ao atravessar o espaço de custo.

Isso pode fazer com que o cálculo do gradiente da função de perda em pontos de sela (*saddle*) torne-se insignificante, de modo que passe a descrescer ou mesmo interromper a atualização dos pesos e, consequentemente, o processo de aprendizado. O método denominado *momentum* foi desenvolvido justamente para mitigar a ocorrência desse tipo de problema, calculando a média móvel dos gradientes anteriores, porém atribuindo um peso maior aos valores mais recentes, processo este conhecido por *Exponential Moving Average*. Diante disso, o processo de atualização dos pesos passa a ser dado por:

$$\begin{aligned} weight.avg &= beta * weight.avg + (1 - beta) * weight.grad, \\ new\_weight &= weight - lr * weight.avg, \end{aligned} \tag{3.6}$$

em que $beta$ é o coeficiente que representa o grau de acréscimo do peso e $weight.avg$ é a média móvel dos gradientes, que precisa ser armazenada para cada parâmetro do modelo, já que são independentes.

Embora o método *momentum* aplicado ao *Gradient Descent* permita uma convergência melhor e mais rápida, ainda assim requer a configuração manual da *learning rate* ou, mesmo quando a *learning rate* é baixa, oscilações podem ocorrer. Para contornar esta situação, variantes de SGD foram introduzidas, como por exemplo RMSProp, que utiliza uma *adaptive learning rate*, ou seja, ao invés de aplicar a mesma taxa para cada parâmetro, aplica-se uma taxa específica controlada por uma *global learning rate*. Deste modo, o treinamento pode ser acelerado com uma taxa mais alta para pesos que precisam mudar bastante, enquanto àqueles já estáveis é atribuída uma taxa menor (HOWARD; GUGGER, 2020a).

Já o algoritmo *Adam (Adaptive Moment Estimation)* combina os conceitos de SGD *with momentum* e RMSProp, usando a média móvel dos gradientes como direção e dividindo pela raiz quadrada da média móvel dos gradientes ao quadrado para atribuir uma *adaptive learning*

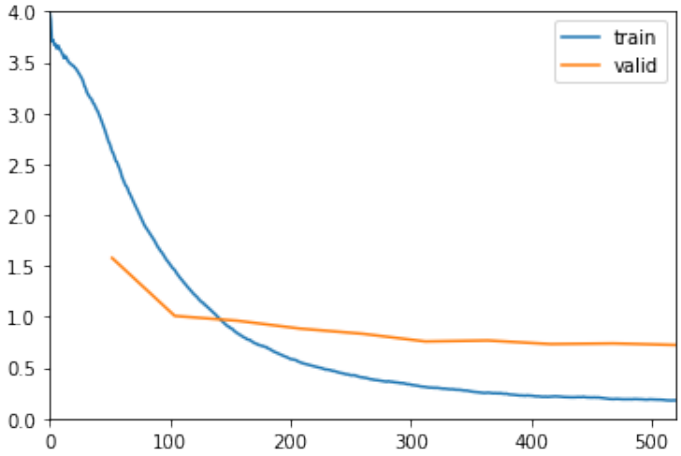
rate a cada parâmetro. Inclui ainda uma correção de viés (*bias*) para as estimativas de média móvel. O Algoritmo 1 ilustra este procedimento.

Algoritmo 1 – *Adam Optimization*

Input: $f(\theta), \theta_0$ ▷ Objective function and initial parameters
Input: α ▷ Learning rate
Input: β_1, β_2 ▷ Exponential decay rates for the first and second moment estimates
Input: ε ▷ Small value for numerical stability
Initialize: $m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$ ▷ Initialize first and second moment estimates and time step
enquanto θ not converged **faça**
    $t \leftarrow t + 1$
    $g \leftarrow \nabla f(\theta)$ ▷ Compute the gradient of the objective function
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g$ ▷ Update the first moment estimate
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g^2$ ▷ Update the second moment estimate
    $m'_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute the bias-corrected first moment estimate
    $v'_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute the bias-corrected second moment estimate
    $\theta \leftarrow \theta - \alpha m'_t / (\sqrt{v'_t} + \varepsilon)$ ▷ Update the parameters
fim enquanto

Para monitorar o progresso do treinamento do modelo, no que diz respeito à perda em cada época, optou-se por implementar uma função de *callback* denominada *ShowGraphCallback*, da biblioteca fast.ai. Na Figura 26, pode-se observar a evolução do erro ou perda nos conjuntos de treinamento e de validação.

Figura 26 – Gráfico com a evolução da perda (erro) nos conjuntos de treinamento e de validação, conforme demonstrado pela função *ShowGraphCallback*.



Fonte: Elaborada pelo autor.

Em geral, deseja-se que a *Training Loss* seja baixa e que a *Validation Loss* seja semelhante à *Training Loss*, pois se esta for muito menor do que a *Validation Loss*, isso pode indicar que o modelo está superajustando (*overfitting*) os dados de treinamento e pode não ter um bom desempenho em dados não vistos. Por outro lado, se a *Training Loss* for muito maior do que a *Validation Loss*, isso pode indicar que o modelo está subajustando (*underfitting*) os dados de treinamento e pode não ser capaz de capturar os padrões subjacentes nos dados.

A Figura 27 ilustra de forma simplificada o algoritmo de implementação dos modelos de CNNs com base nos conceitos acima discutidos.

Figura 27 – Versão simplificada do algoritmo de implementação dos modelos de CNNs, usando a biblioteca fast.ai.

```
from sklearn.model_selection import StratifiedKFold
from fastai.basics import *
from fastai.vision.all import *
from fastai.vision.augment import aug_transforms
from fastai.vision.data import imagenet_stats, ImageBlock
from torchvision.models import *
import os
import pandas as pd

# Define splitting function that reads from a dataframe column
def stratifiedsplitter(df):
        train = df.index[~df['is_valid']].tolist()
        valid = df.index[df['is_valid']].tolist()
        return train, valid

# Prepare transforms: Image is larger initially (512 px) for later augmentations
item_tfms = [RandomResizedCrop(512, min_scale=0.7)]

# Batch transformations include a standard set of augmentations and a normalization process
batch_tfms = [*aug_transforms(size=img_size), Normalize.from_stats(*imagenet_stats)]

# Prepare DataBlock
data = DataBlock(blocks=(ImageBlock, CategoryBlock),
                         splitter=stratifiedsplitter, # Stratified train/validation split
                         get_x=lambda x: x['Path'],   # Get the image path
                         get_y=lambda y: y['Class']   # Get the label
                         item_tfms=item_tfms,         # Crop images with RandomResizedCrop
                         batch_tfms=batch_tfms)       # Implement augumentations and normalize the images

# Get source data from each folder (Class) in the drive
source = []
for folder in sorted(os.listdir(path)):
        for file in sorted(os.listdir(path+'/'+folder)):
                source.append((folder, path+folder+'/'+file))

dataframe = pd.DataFrame(source, columns=['Class', 'Path'])
X, y = dataframe['Path'], dataframe['Class']

# Create stratified split using sklearn
kf = StratifiedKFold(n_splits=n_splits, shuffle=shuffle, random_state=random_state)
splits = list(kf.split(X, y))

# Prepare the selected fold to generate a new df
fold = fold
train_df = dataframe.loc[splits[fold][0]]
val_df = dataframe.loc[splits[fold][1]]
train_df['is_valid'] = False
val_df['is_valid'] = True
df = train_df.append(val_df)

# Get Dataloader specifying dataframe and batch size
dls = data.dataloaders(df, bs=bs)

# Get Optimization function, where 'mom': momentum decay rate; 'sqr_mom': second momentum decay rate; 'eps': small constant used to
↪  prevent division by zero; 'wd': weight decay rate; 'decouple_wd': decouple the weight decay from the gradient updates
opt_func = partial(Adam, mom=mom, sqr_mom=sqr_mom, eps=eps, wd=wd, decouple_wd=True)

# Get Metrics
metrics = [accuracy, top_k_accuracy, error_rate, Precision(average='micro'), Recall(average='micro'), F1Score(average='micro')]

# Setup Callbacks
callbacks = [CSVLogger(fname=log_name, append=True), ShowGraphCallback()]

# Setup Learner
learn = vision_learner(dls, # Pass the dataloader
                       model = globals()[arch], # Pass the selected architecture (e.g., 'resnet34')
                       pretrained=True, # Indicate True for pretrained flag
                       opt_func=opt_func, # Pass the Adam optimization function
                       metrics=metrics,  # Pass the desired metrics
                       cbs=callbacks) # Pass the callbacks

learn.freeze() # Freeze the initial layers
_,_,valley_freeze,_ = learn.lr_find(suggest_funcs=(minimum, steep, valley, slide)) # Obtain the valley point in the lr_find graph
learn.fit_one_cycle(epochs, lr_max=valley_freeze) # Train with maximum learning rate equal to valley

learn.unfreeze() # Unfreeze all layers
learn.lr_find(suggest_funcs=(minimum, steep, valley, slide)) # Find new suggested rates by analyzing lr_find graph
learn.fit_one_cycle(epochs, lr_max=slice(min_rate, max_rate)) # Train with discriminative learning rates
```

Fonte: Elaborada pelo autor.

Em relação à reproducibilidade, embora resultados completamente reproduzíveis não sejam garantidos, é possível executar alguns procedimentos para limitar o número de fontes de comportamento não determinístico para uma plataforma, dispositivo e versão do PyTorch específicos. O estabelecimento de uma constante no comando *torch.manual_seed()*, associado à eliminação de outras fontes de operações não determinísticas, permite que a mesma série de números aleatórios seja gerada toda vez que o aplicativo for executado no mesmo ambiente. Além disso, a biblioteca cuDNN (*CUDA Deep Neural Network*), usada pelas operações de convolução CUDA (*Compute Unified Device Architecture*), pode ser uma fonte de não determinismo em várias execuções de um aplicativo. Desativar o recurso de *benchmarking* com *arch.backends.cudnn.benchmark = False* faz com que cuDNN selecione um algoritmo de forma determinística, possivelmente ao custo de desempenho reduzido. Ainda assim, embora desabilitar o *benchmarking* de convolução CUDA garanta a seleção do mesmo algoritmo toda vez que um aplicativo for executado, esse algoritmo em si pode ser não determinístico, a menos que a opção *torch.backends.cudnn.deterministic = True* seja utilizada.

3.4 Métricas de Avaliação

Para medir o desempenho dos modelos propostos, foram utilizados procedimentos comumente empregados em Ciência de Dados (*Data Science*), tais como as técnicas a seguir.

3.4.1 Confusion Matrix, Accuracy, Precision, Recall, F1-Score

Em *Machine Learning* e *Deep Learning*, uma matriz de confusão (*confusion matrix*), também denominada como matriz de erro ou matriz de contingência, refere-se a uma tabela que possibilita avaliar o desempenho de um algoritmo de classificação. De maneira geral, esta tabela compara os resultados obtidos na base de testes com os resultados previamente aprendidos na base de treinamento, indicando as quantidades de resultados corretos e incorretos gerados pelo algoritmo.

De acordo com Jensen (2015), acuracidade (*accuracy*) é um parâmetro estatístico simples derivado da matriz de confusão, resultante da divisão da soma dos valores da diagonal principal pelo número total de amostras usadas para o cálculo dessa matriz.

A partir de uma matriz de confusão m, pode-se calcular a acuracidade da seguinte maneira:

$$Accuracy = \frac{\sum_{i=1}^{r} m_{ii}}{n}, \tag{3.7}$$

em que r é o número de linhas e colunas da matriz de confusão, m_{ii} é o número de observações da linha i e coluna i, correspondente às previsões corretas para i-ésima classe, e n é o número total de observações.

Todavia, em conjuntos de dados com desequilíbrio de classes, deve-se analisar a acuracidade com precaução, já que neste caso pode-se originar uma falsa impressão de bom desempenho. Tomando-se como exemplo um conjunto de dados em que 80% das amostras estejam associadas a uma determinada classe, ao classificar todas estas amostras corretamente, já se garante um desempenho de 80%, mesmo que todas as amostras da outra classe sejam classificadas incorretamente (NETO, 2018).

Uma tabela de confusão permite efetuar uma avaliação mais completa da situação do classificador, já que ela separa os resultados em quatro categorias:

- Verdadeiro Positivo (*True Positive*): número de exemplos positivos classificados corretamente;
- Falso Positivo (*False Positive*): número de exemplos positivos classificados incorretamente;

- Verdadeiro Negativo (*True Negative*): número de exemplos negativos classificados corretamente;
- Falso Negativo (*False Negative*): número de exemplos negativos classificados incorretamente.

Dessa forma, uma nova acuracidade pode ser calculada a partir da equação:

$$Accuracy = \frac{\sum TruePositive + \sum TrueNegative}{\sum TruePositive + \sum TrueNegative + \sum FalsePositive + \sum FalseNegative}. \quad (3.8)$$

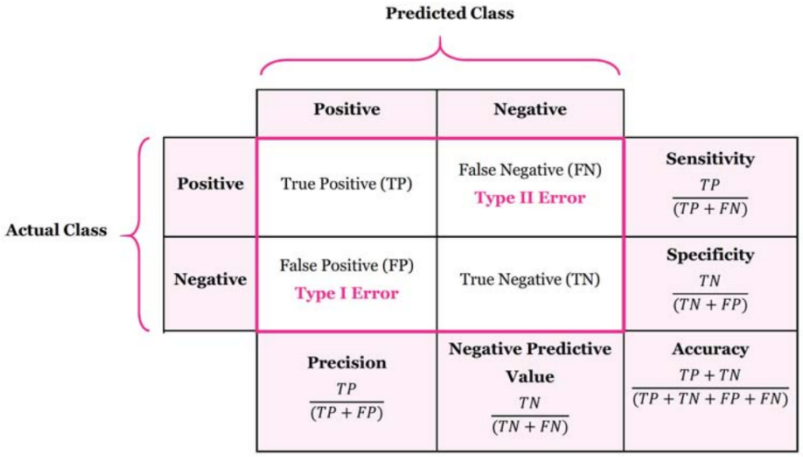
E pode-se calcular também a confiabilidade positiva (*precision*), que representa a correção do sistema de previsão de classificação, e a sensibilidade (*recall*), que representa a eficácia do sistema:

$$Precision = \frac{\sum TruePositive}{\sum TruePositive + \sum FalsePositive}, \quad (3.9)$$

$$Recall = \frac{\sum TruePositive}{\sum TruePositive + \sum FalseNegative}. \quad (3.10)$$

A Figura 28 ilustra uma matriz de confusão e as principais métricas de classificação:

Figura 28 – Matriz de confusão com métricas de classificação avançadas.



Fonte: Sirsat (2019).

Para calcular o desempenho geral de um classificador levando em conta tanto a confiabilidade positiva (*precision*) quanto a sensibilidade (*recall*), emprega-se uma métrica denominada *F1-Score*, que é calculada como a média harmônica de tais elementos:

$$F1 = \frac{2 \times (precision \times recall)}{(precision + recall)}. \tag{3.11}$$

3.4.2 Top-k Accuracy

Na classificação de imagens multiclasse, a acuracidade top-k é uma métrica que mede o desempenho de um classificador considerando as principais k previsões feitas para cada imagem. É calculada determinando primeiro as k previsões com as probabilidades mais altas para cada imagem e, em seguida, comparando essas previsões com os verdadeiros rótulos de classe das imagens. A acuracidade top-k é a proporção de imagens para as quais o verdadeiro rótulo de classe está entre as top-k previsões.

Assim, a acuracidade top-1, ou acuracidade convencional, representa o número de predições corretas (classe com a maior probabilidade) dividido pelo número total de predições, enquanto a acuracidade top-5 significa que qualquer uma das 5 respostas de probabilidade mais alta do modelo deve corresponder à resposta esperada.

3.4.3 Top Losses

Em classificação de imagens, as *top losses* referem-se às imagens de treinamento classificadas incorretamente e cujas perdas (ou erros) atingiram os valores mais altos. Tais imagens podem ser úteis para entender onde o modelo está falhando e fornecer *insights* sobre como melhorá-lo.

É procedimento comum, por exemplo, visualizar as imagens de *top losses* junto às suas previsões e rótulos corretos. Isso permite identificar quais características das imagens o modelo está tendo dificuldade em aprender, além de apontar quais tipos de dados de treinamento adicionais podem ser necessários ou mesmo eventuais problemas relativos ao processo de atribuição de rótulos (*labels*).

3.4.4 Most Confused Pairs of Labels

Os pares de rótulos mais confundidos (*most confused pairs of labels*) referem-se a dois ou mais rótulos que o modelo de aprendizado de máquina frequentemente confunde entre si. Esses pares de rótulos são geralmente identificados a partir de erros de classificação cometidos durante o processo de teste. É possível identificá-los, por exemplo, através de análise da matriz de confusão, que mostra quantas vezes o modelo classificou cada rótulo como sendo outro rótulo. Aqueles que aparecem com mais frequência nas diagonais secundárias da matriz são considerados como sendo os rótulos mais confundidos.

Essa análise pode ajudar a entender por que o modelo está cometendo certos erros, incluindo dificuldades na diferenciação entre dois rótulos que são visualmente semelhantes, para os quais talvez sejam necessários mais dados de treinamento.

3.4.5 Parameters (Weights) x Total Training Time

Em uma CNN, parâmetros (*parameters*) ou pesos (*weights*) referem-se aos valores que são aprendidos durante o processo de treinamento para fazer previsões. Os parâmetros são armazenados nas camadas do modelo e são usados para ajustar a saída de cada camada com base na entrada que ela recebe. O número de parâmetros em uma CNN pode ser um indicador da complexidade do modelo e da capacidade de aprender a função-alvo. Em geral, um modelo com mais parâmetros terá mais capacidade de aprender a partir dos dados, mas também pode ter um risco maior de sobreajuste. No entanto, isso nem sempre é o caso e a capacidade de um modelo também depende de outros fatores, como arquitetura e quantidade de dados de treinamento.

O tempo total de treinamento refere-se à quantidade de tempo que leva para um modelo ser treinado. Esse tempo pode ser afetado por vários fatores, como o número de parâmetros, a quantidade de dados de treinamento e os recursos disponíveis (por exemplo, o número de núcleos de CPU ou GPU). Um tempo total de treinamento mais curto pode indicar que o modelo é mais eficiente e pode ser treinado mais rapidamente, enquanto um tempo total de treinamento mais longo pode indicar que o modelo é mais complexo e requer mais recursos para ser treinado.

Tanto os parâmetros quanto o tempo total de treinamento podem ser úteis como métricas para avaliar o desempenho de um modelo ao desenvolver CNNs. Eles podem fornecer *insights* sobre a complexidade e a eficiência do modelo e ajudar na identificação de *trade-offs* entre esses fatores. Por exemplo, um modelo com um alto número de parâmetros e um tempo total de treinamento mais curto pode ser mais complexo e eficiente do que um modelo com um número menor de parâmetros e um tempo total de treinamento mais longo.

CAPÍTULO

4

RESULTADOS E DISCUSSÃO

Neste capítulo, serão apresentados os resultados dos experimentos realizados de acordo com as metodologias anteriormente descritas. Será feita uma análise comparativa entre os modelos propostos, através dos valores médios e máximos de cada métrica de avaliação *(top-1 accuracy, top-5 accuracy, precision, recall, f1-score)*, tanto em nível global, considerando todas as *folds*, quanto em nível individual, analisando cada *fold* separadamente. Em seguida, para cada modelo, serão analisadas as maiores perdas (*top losses*) dentre todas as *folds*, as matrizes de confusão e os pares de rótulos mais confundidos. Adicionalmente, terá destaque a relação entre as acuracidades médias, os parâmetros e os tempos totais de treinamento. Serão feitas ainda as considerações finais com as reflexões a respeito dos resultados, os limites relativos a comparações com outros trabalhos, bem como as sugestões propostas para futuras melhorias.

4.1 Valores Médios e Máximos *(top-1 accuracy, top-5 accuracy, precision, recall, f1-score)*, considerando todas as *Folds*

Para cada métrica na Figura 29, os números foram obtidos calculando-se os valores médios nas 10 épocas finais (*fine-tuning*) de cada *fold* e, a partir disso, calculando-se a média entre todas as *folds*. Para obter *precision*, *recall* e *f1-score* em cada época, foi aplicado o conceito de *micro-average*, que atribui peso igual a cada amostra do *dataset*, dado que este passou previamente pelo processo de *oversampling*.

No caso da acuracidade top-1, nota-se que os modelos com prefixo "fix" (*fixresnet152*, *fixresnet101*, *fixresnet50*, *fixresnet34*, *fixefficientnet_b0b*) obtiveram melhor desempenho, quando comparados àqueles cujas arquiteturas originais (*resnet152*, *resnet101*, *resnet50*, *resnet34*, *efficientnet_b0b*) foram implementadas sem a estratégia de correção da resolução das imagens. De maneira geral, os modelos *fixresnet152* e *inceptionresnet_v1* alcançaram os melhores resultados,

com acurácias de 83,4% e 82,1%, respectivamente, enquanto os modelos *inception_v3*, *efficientnet_b3b*, *vgg16* e *efficientnet_b0b* obtiveram os piores resultados, ficando abaixo de 78,0%. Considerando que o pior resultado foi de 75,7% (*efficientnet_b0b*), a diferença entre o melhor e o pior desempenho atingiu 7,7 pontos percentuais.

Em relação à acuracidade top-5, podemos observar que os modelos alcançaram um alto desempenho, de modo que *inceptionresnet_v2*, *inceptionresnet_v1* e *resnet34* obtiveram o melhor resultado, com 98,4%, ou seja, uma diferença de apenas 0,8 ponto percentual em relação ao pior desempenho, que corresponde aos 97,6% obtidos pelos modelos *efficientnet_b0b* e *fixefficientnet_b0b*.

Já os números relativos a *precision*, *recall* e *f1-score* demonstram que os modelos *fixresnet152*, *inceptionresnet_v1*, *fixresnet101* e *fixresnet50* conseguiram atingir os melhores desempenhos, todos acima de 81,0%, indicando que foram capazes de prever corretamente um maior número de resultados em relação aos outros modelos e também foram capazes de encontrar a maior proporção de resultados que deveriam ter sido encontrados. Por outro lado, os modelos *vgg16* e *efficientnet_b0b* ficaram com os piores desempenhos (inferiores a 77%), ou seja, foram menos precisos em prever resultados e também encontraram menos resultados que deveriam ter sido encontrados, quando comparados aos outros modelos.

Na Figura 30, os números foram obtidos calculando-se os valores máximos nas 10 épocas finais (*fine-tuning*) de cada *fold* e, a partir disso, calculando-se a média entre todas as *folds*.

Também neste cenário, fica evidente que os modelos com prefixo “fix” obtiveram melhor acuracidade, quando comparados às suas arquiteturas originais. No geral, os modelos *fixresnet152*, *inceptionresnet_v1* e *resnet152* atingiram os melhores resultados, com acuracidade top-1 acima de 83,0%, enquanto os modelos *vgg16* e *efficientnet_b0b* apresentaram os piores desempenhos, ficando abaixo de 80,0%. Neste caso, a diferença entre o melhor desempenho (*fixresnet152*: 84,6%) e o pior desempenho (*efficientnet_b0b*: 78,3%) foi de 6,3 pontos percentuais.

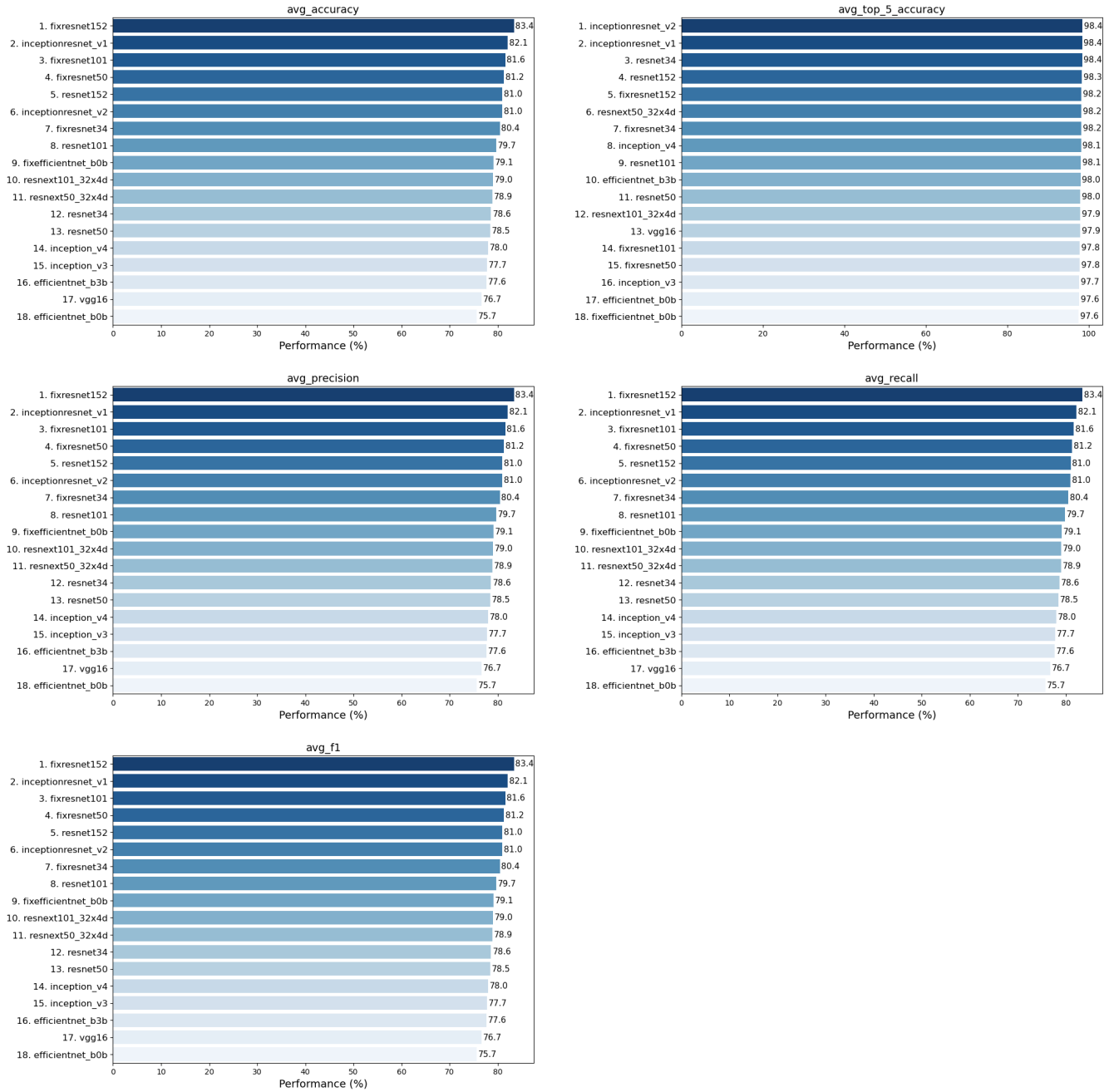
Ademais, os modelos *resnet101* e *resnet152* apresentaram a maior acuracidade top-5, com 98,9%. Os outros modelos apresentaram uma acuracidade top-5 ligeiramente menor, sendo que os modelos *fixefficientnet_b0b*, *vgg16*, *fixresnet101* e *fixresnet50* apresentaram o menor desempenho, com 98,3%. Ou seja, há uma diferença de apenas 0,6 ponto percentual entre os melhores e piores modelos.

Tal como ocorreu com a acuracidade top-1, os modelos *fixresnet152*, *inceptionresnet_v1* e *resnet152* também obtiveram os melhores desempenhos em *precision*, *recall* e *f1-score*, ficando acima de 83,0%, enquanto os modelos *vgg16* e *efficientnet_b0b* ficaram abaixo de 80,0%, tendo assim os piores desempenhos.

Portanto, na comparação entre os números demonstrados na Figura 29 e na Figura 30, constata-se que não houve diferenças significativas nas listas de modelos com melhores e piores desempenhos, no que diz respeito a *accuracy*, *precision*, *recall* e *f1-score*, mas ressalta-se que os

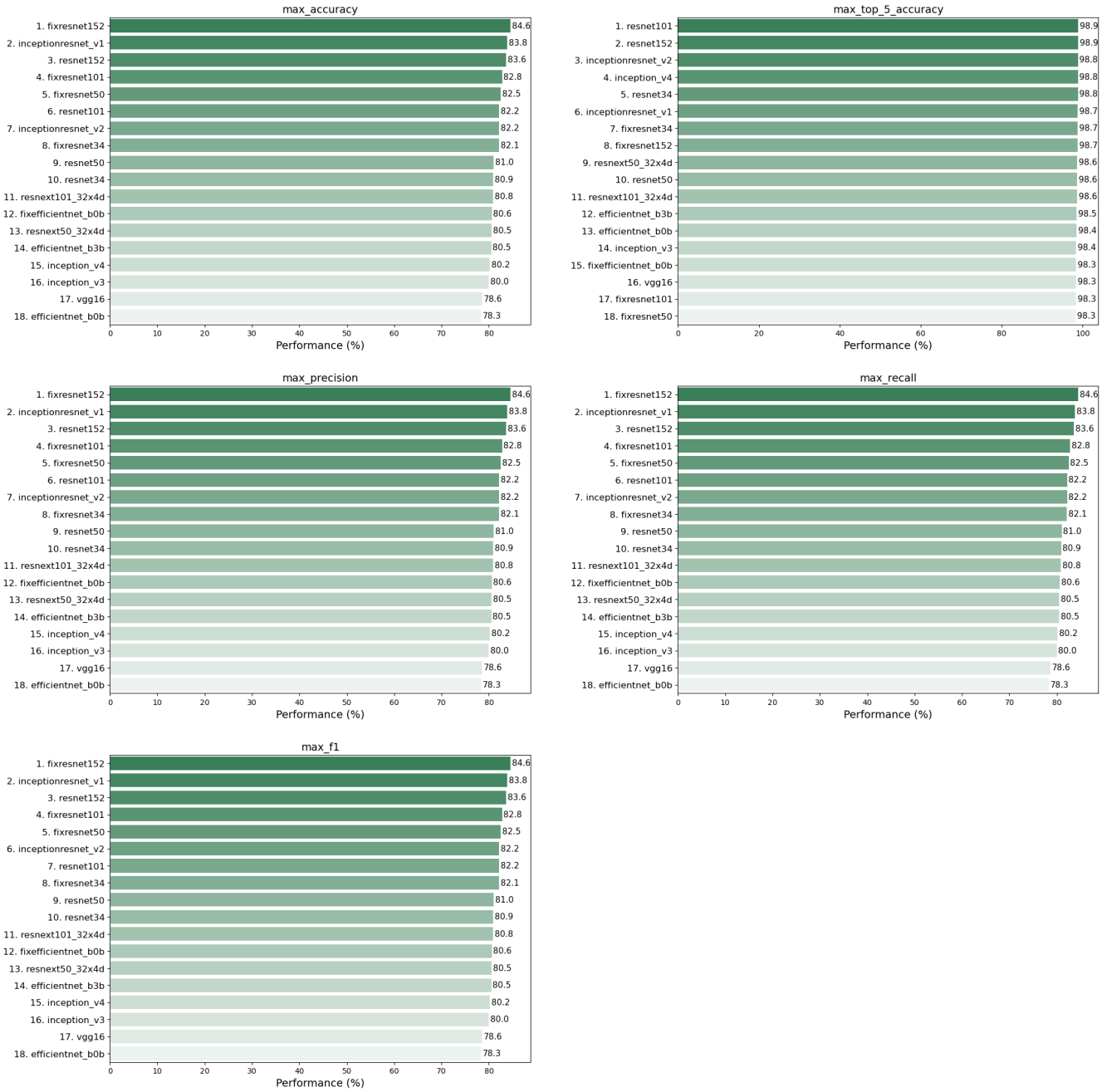
modelos com arquitetura ResNet saíram-se melhor quando observadas as médias (entre todas as *folds*) de seus desempenhos máximos, em detrimento das médias (entre todas as *folds*) de seus desempenhos médios nas 10 *epochs* finais do treinamento. Já em relação à acuracidade top-5, houve maior diversidade entre as posições no *ranking*, mas as distâncias entre as primeiras e últimas colocações podem ser consideradas irrelevantes em ambos os casos.

Figura 29 – Valor médio das métricas durante o processo de *fine-tuning*: obtiveram-se os valores médios nas 10 *epochs* finais de cada *fold* e, em seguida, calculou-se a média entre todas as *folds*.



Fonte: Elaborada pelo autor.

Figura 30 – Valor máximo das métricas durante o processo de *fine-tuning*: obtiveram-se os valores máximos nas 10 *epochs* finais de cada *fold* e, em seguida, calculou-se a média entre todas as *folds*.



Fonte: Elaborada pelo autor.

4.2 Valores Médios e Máximos *(top-1 accuracy, top-5 accuracy, precision, recall, f1-score)* por *Fold* e Análise das *Top Losses*

Na Figura 31 e na Figura 32, os números foram obtidos calculando-se o valor médio e o valor máximo, respectivamente, nas 10 épocas finais (*fine-tuning*) de cada *fold*. Observa-se que há uma clara prevalência dos piores desempenhos concentrada, sobretudo, nas *folds* 2 e 5, já que nestas constataram-se as maiores perdas segundo o critério de *Cross-Entropy Loss*. Ou seja, nota-se uma diferença maior entre as probabilidades previstas e os rótulos verdadeiros na comparação com as demais *folds* utilizadas durante o processo de treinamento dos modelos.

Especificamente no modelo *resnext101_32x4d*, além da *fold* 2, a *fold* 4 também ficou dentre aquelas com pior desempenho, tanto em relação aos valores médios quanto aos valores máximos da maior parte das métricas. No modelo *inception_v3*, a *fold* 1, para os valores médios, e a *fold* 4, para os valores máximos, figuraram dentre os piores desempenhos. Já no modelo *inceptionresnet_v2*, a *fold* 1 apresentou um dos piores desempenhos, ao lado da *fold* 2, em quase todas as métricas. A *fold* 1 também manteve-se dentre os piores desempenhos no modelo *fixresnet34*, particularmente em relação aos valores máximos.

Diante disso, coube destacar quais imagens ocasionaram as quatro maiores perdas dentre todas as *folds*, para cada modelo. Na Figura 33, podemos observar que algumas destas imagens aparecem repetidas vezes.

Nos modelos *resnet34*, *resnet50*, *resnet152*, *fixresnet34* e *fixresnet50*, por exemplo, temos uma imagem prevista como *Tube_Screen*, mas cujo rótulo real é *Plastic*. Aparentemente, trata-se da parte traseira de uma carcaça de plástico desacoplada de monitor e, portanto, confundida com o item em seu estado não desmantelado.

A imagem prevista como *Aluminium_Iron*, mas cujo rótulo real é *Battery*, conforme observada nos modelos *vgg16*, *resnet101*, *inception_v3* e *fixresnet101*, refere-se à parte lateral de uma bateria de chumbo que, neste caso, possui carcaça composta de material metálico, o que corroborou para que fosse confundida efetivamente com sucata de alumínio ou ferro.

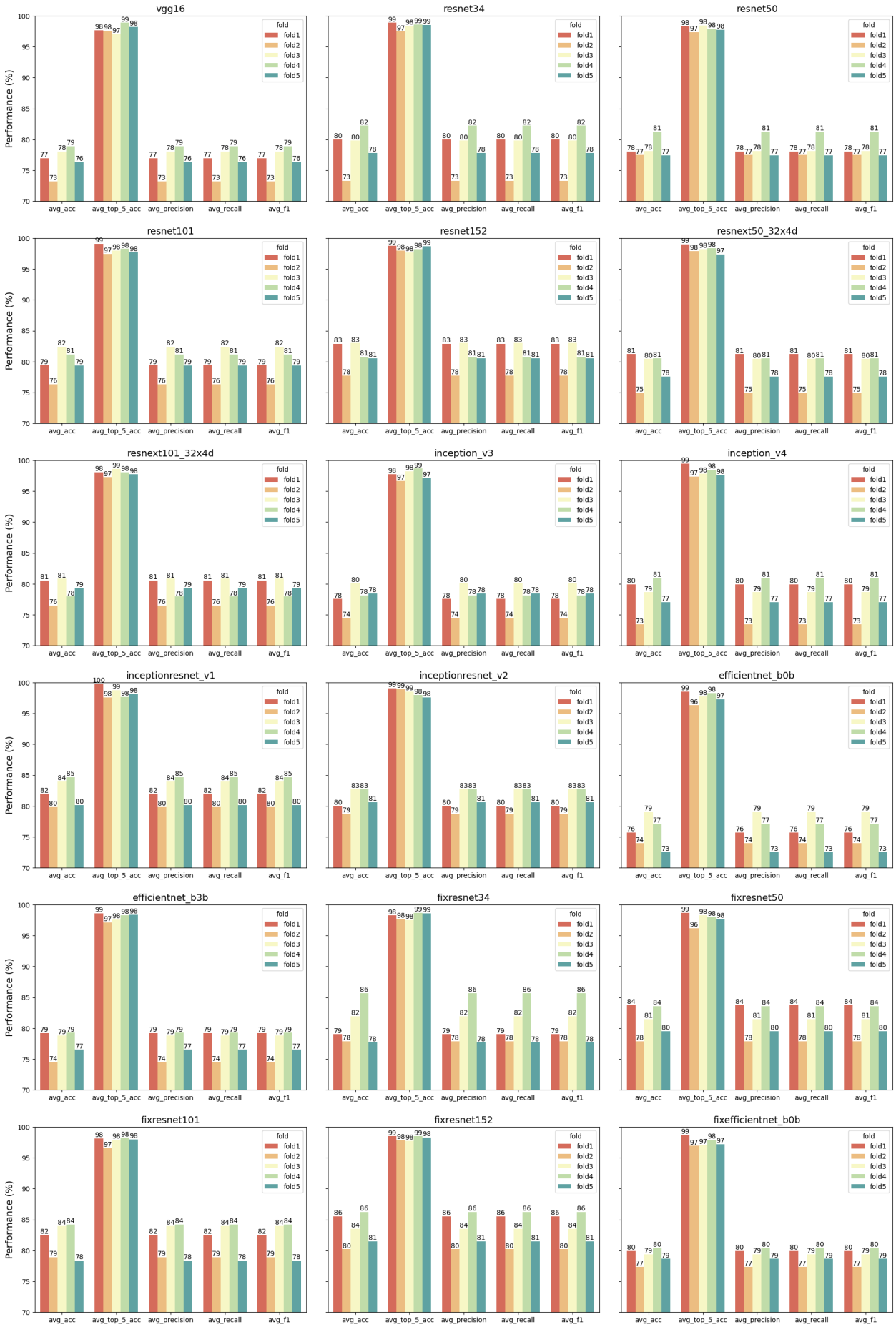
Já as imagens previstas como *Cable_Wire*, mas cujo rótulo real é *Plastic*, conforme observadas nos modelos *resnet50*, *resnet101*, *resnet152*, *fixresnet50*, *fixresnet101* e *fixresnet152*, referem-se a presilhas de plástico com formato longitudinal, fato este que possivelmente contribuiu para que fossem confundidas com cabos ou fios.

Nos modelos *resnet34*, *inception_v4*, *efficientnet_b0b*, *efficientnet_b3b* e *fixresnet34*, podemos observar que as teclas de um teclado, cujo rótulo real é *Plastic*, acabaram sendo confundidas com baterias, possivelmente devido ao formato retangular compartilhado entre as teclas e alguns tipos de baterias existentes no *dataset*.

Há também uma imagem prevista como *Copper*, mas cujo rótulo real é *Cable_Wire*, observada nos modelos *resnet101*, *resnet152*, *resnext50_32x4d* e *fixresnet152*. Neste caso, trata-se de um cabo de cor alaranjada, que provavelmente suscitou que o objeto fosse confundido com material composto por cobre. Situação similar pode ter ocorrido com a imagem prevista como *Copper*, mas cujo rótulo real é *Plastic*, obervada nos modelos *efficientnet_b0*, *efficientnet_b3b* e *fixefficientnet_b0b*, já que o pedaço de plástico em questão também possui cor alaranjada.

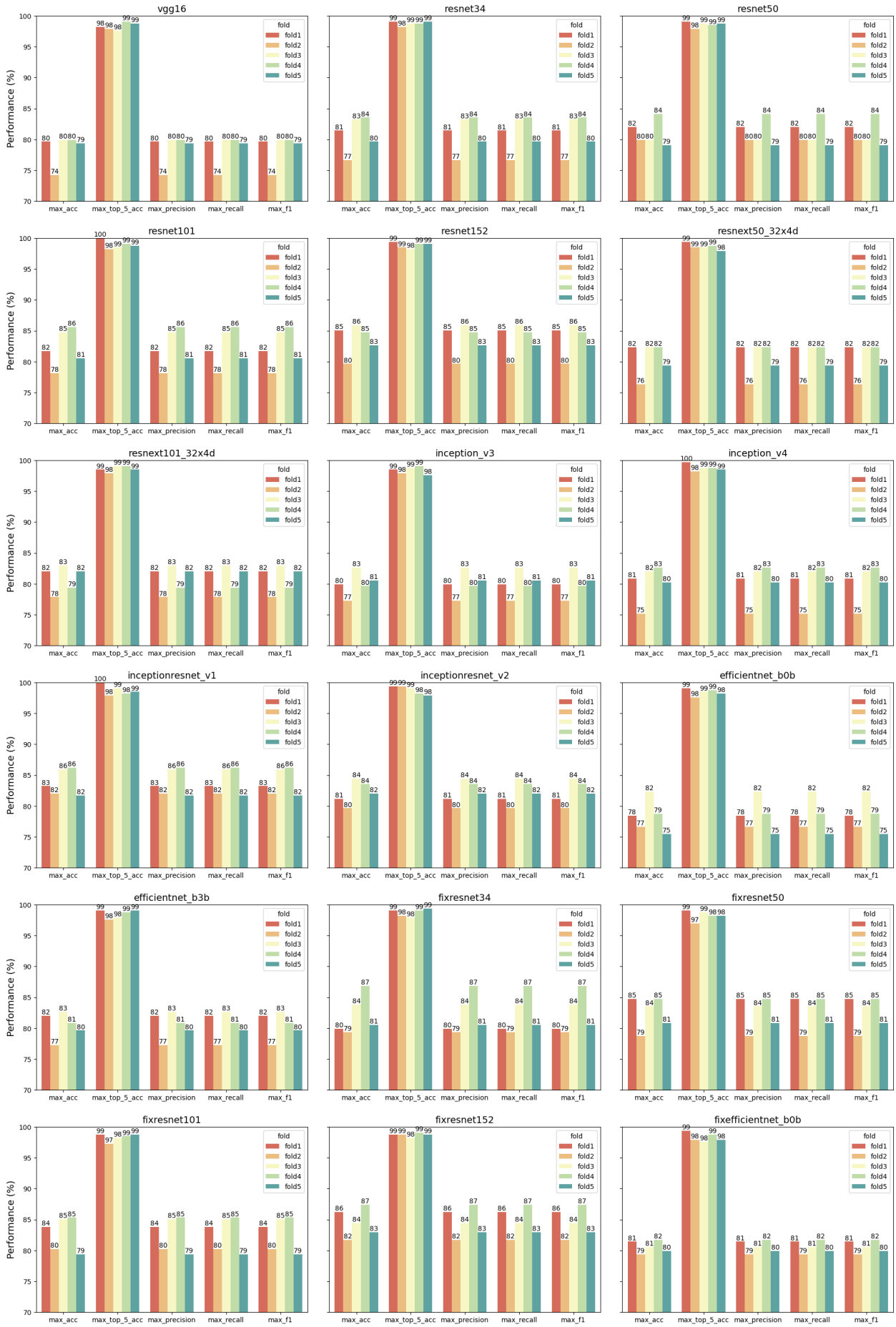
Pode-se destacar ainda a imagem prevista como *Aluminium_Iron*, mas cujo rótulo real é *Tube_Screen*, observada nos modelos *resnet152*, *resnext50_32x4d*, *inceptionresnet_v1*, *inceptionresnet_v2*, *fixresnet152* e *fixefficientnet_b0b*. Aqui, temos aparentemente alguns pedaços avulsos de telas cuja característica brilhante e reflexiva pode ter contribuído para que fossem confundidos com um objeto de alumínio.

Figura 31 – Valor médio de cada métrica nas 10 *epochs* finais de cada *fold*.



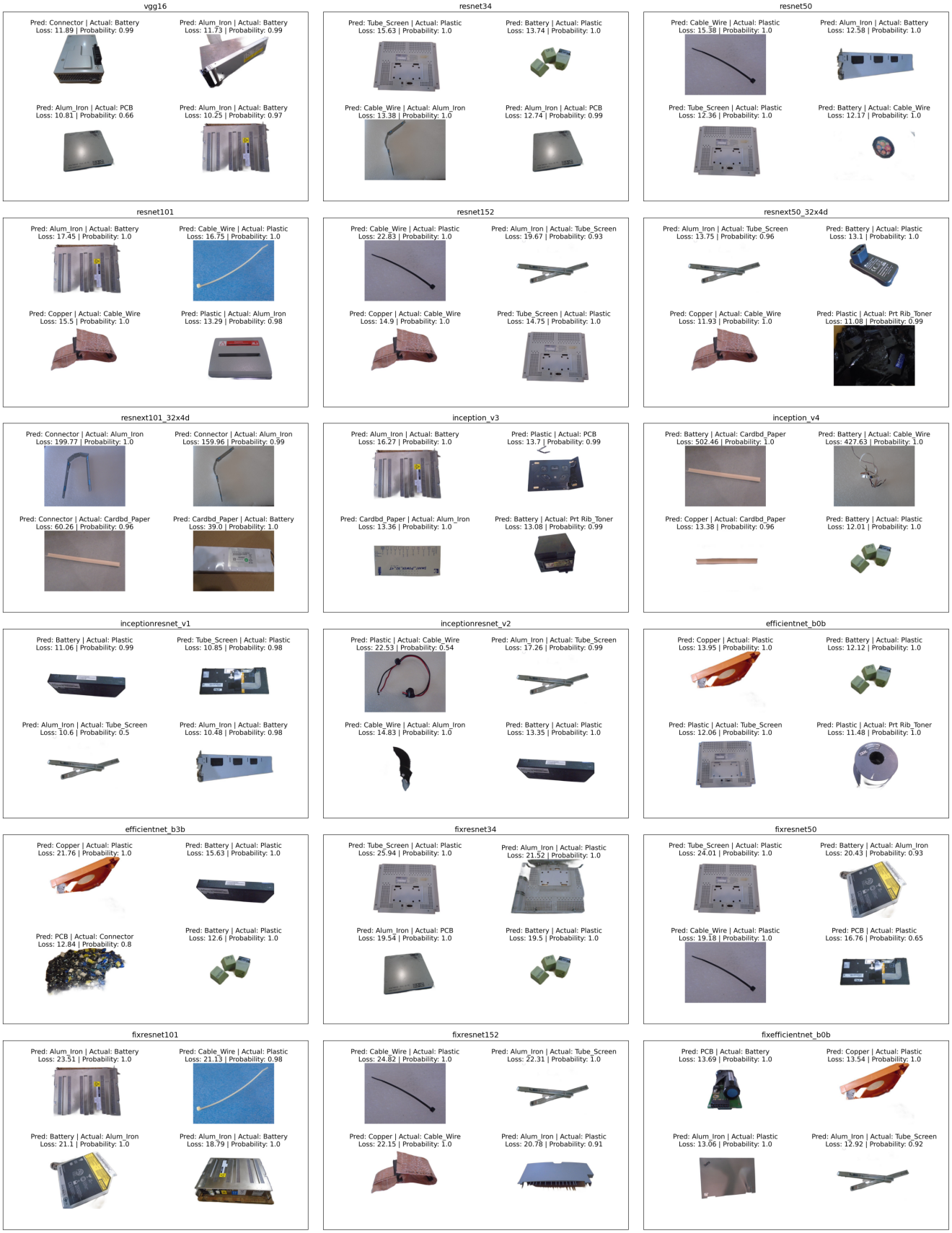
Fonte: Elaborada pelo autor.

Figura 32 – Valor máximo de cada métrica nas 10 *epochs* finais de cada *fold*.



Fonte: Elaborada pelo autor.

Figura 33 – Quatro maiores *top losses* dentre todas as *folds*.



Fonte: Elaborada pelo autor.

4.3 Matrizes de Confusão e Pares de Rótulos mais Confundidos

Na Figura 34 e na Figura 35, temos as matrizes de confusão e os pares de rótulos mais confundidos, respectivamente, considerando as médias entre todas as *folds*. Destacam-se, por exemplo, os pares cujas médias correspondem a pelo menos duas ocorrências na maioria dos modelos treinados, conforme listado na Tabela 2:

Tabela 2 – Pares de rótulos confundidos cuja média de ocorrências é maior ou igual a dois, considerando todas as *folds*.

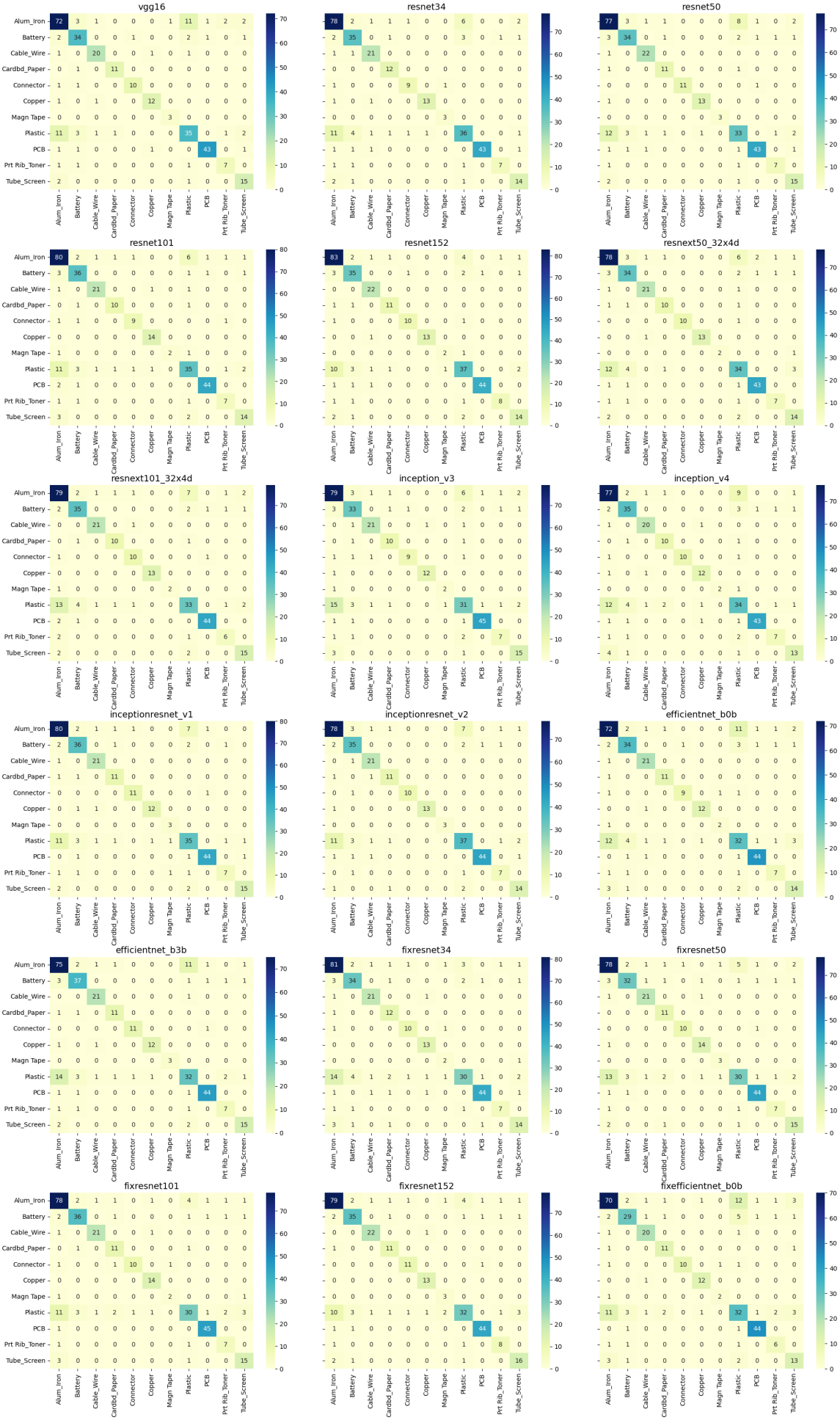
#	Previsto	Real
1	Aluminium_Iron	Plastic
2	Plastic	Aluminium_Iron
3	Plastic	Battery
4	Aluminium_Iron	Battery
5	Battery	Aluminium_Iron
6	Tube_Screen	Aluminium_Iron

A principal razão pela qual os modelos confundiram *Aluminium_Iron* e *Plastic* parece estar associada à similaridade visual de alguns objetos, o que tornou mais complexa a tarefa de distingui-los.

Em relação às discrepâncias notadas entre *Plastic* e *Battery* ou *Aluminium_Iron* e *Battery*, pode-se apontar o fato de que muitas baterias apresentam carcaças compostas por material plástico ou metálico, o que pode contribuir para que sejam confundidas com sucatas avulsas destes mesmos materiais.

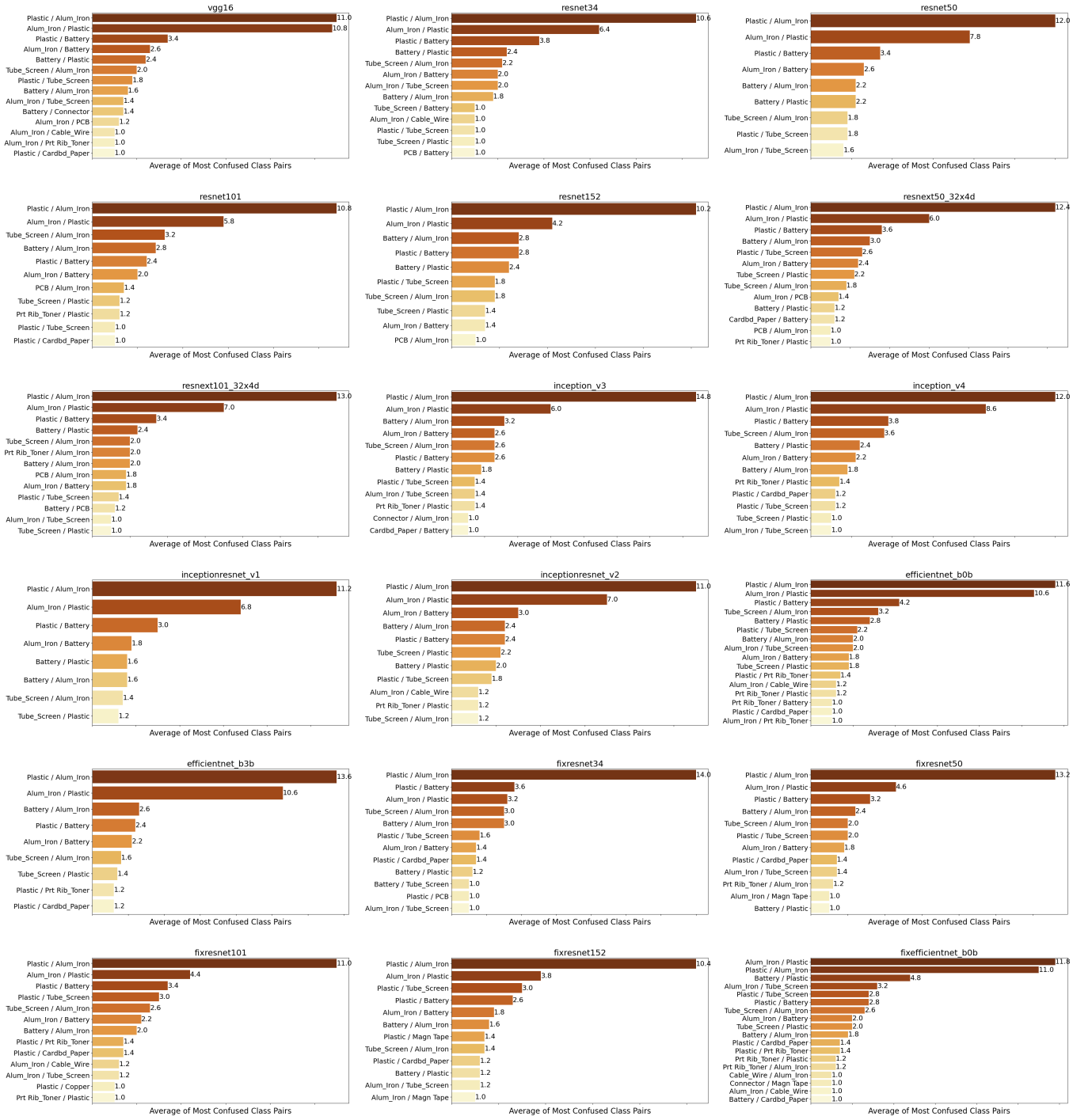
Ademais, nota-se que há imagens de estruturas metálicas cujos formatos e características reflexivas podem ter motivado atribuições errôneas entre *Tube_Screen* e *Aluminium_Iron*.

Figura 34 – Matriz de confusão para cada modelo, calculada como a média entre todas as *folds*.



Fonte: Elaborada pelo autor.

Figura 35 – Médias dos pares de rótulos mais confundidos (*Predicted* / *Actual*), considerando todas as *folds*.



Fonte: Elaborada pelo autor.

4.4 Relação entre Acuracidades Médias, Parâmetros e Tempos Totais de Treinamento

Na Figura 36, encontra-se um gráfico de dispersão exibindo a relação entre a acuracidade média (eixo y) e o tempo total de treinamento (eixo x). Para cada modelo, a acuracidade (top-1) foi obtida através do valor médio nas 10 épocas finais (*fine-tuning*) de cada *fold* e, a partir disso, calculando-se a média entre todas as *folds*. Além disso, o tamanho dos pontos no gráfico é determinado pelo número de parâmetros, em milhões, correspondente aos modelos em questão.

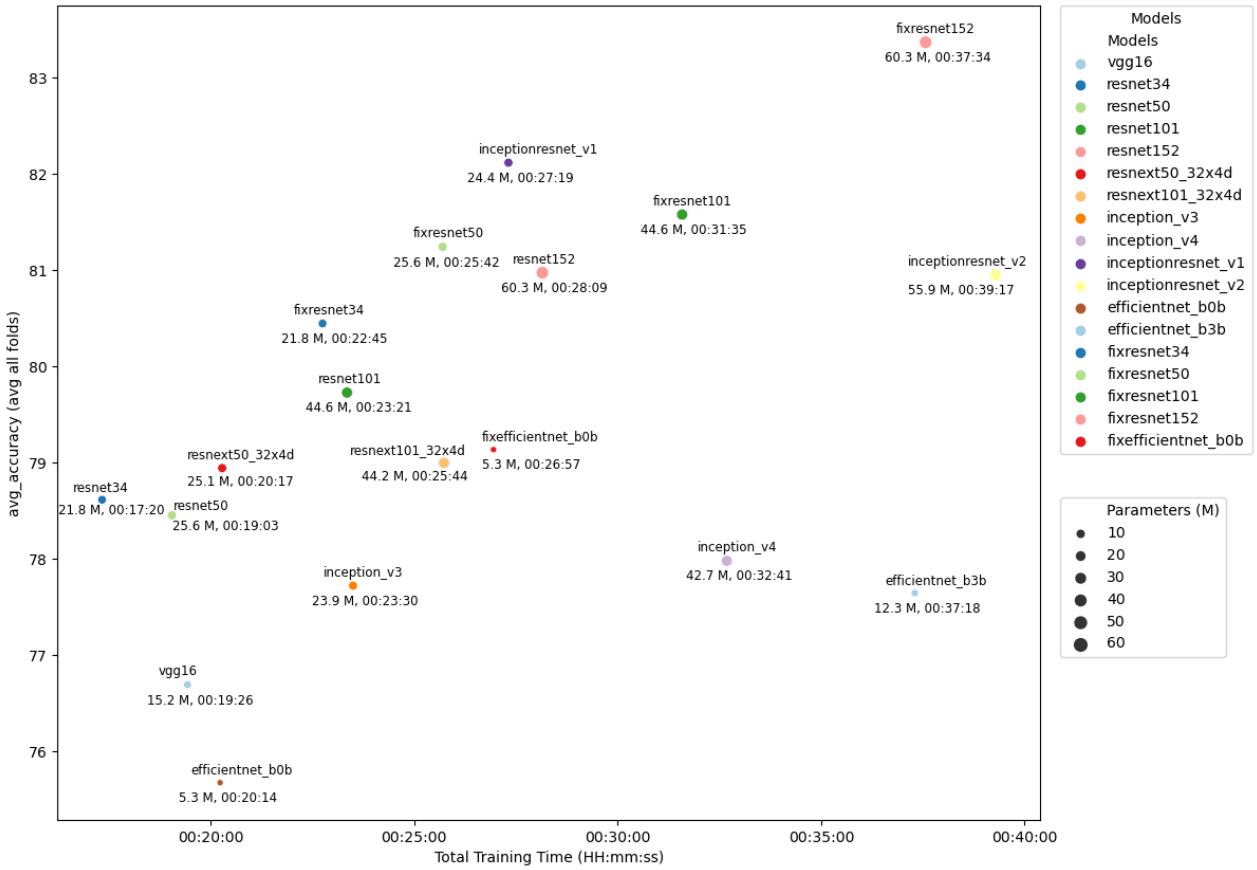
De maneira geral, os modelos que apresentaram maior acuracidade tendem a ter mais parâmetros e requerem maior tamanho de imagem de entrada, levando, portanto, mais tempo para serem treinados. Ou, no caso dos modelos com prefixo "fix", o tempo total de treinamento é maior, em comparação às arquiteturas originais, por conta da estratégia de correção da resolução das imagens, através da qual os modelos originalmente treinados com entrada 224x224x3 passaram por um estágio de *fine-tuning* das últimas camadas utilizando uma resolução maior, correspondente a 320x320x3.

Neste sentido, nota-se que o modelo *fixresnet152* obteve a maior acuracidade média (83,4%), com o maior número de parâmetros (60,3 M) dentre todos os modelos (ao lado do *resnet152*), e com um tempo total de treinamento correspondente a 37 minutos e 34 segundos, ou seja, trata-se do segundo maior tempo dentre todos os modelos, ficando atrás apenas do *inceptionresnet_v2*, que levou 39 minutos e 17 segundos.

No entanto, também é importante observar que o modelo *inceptionresnet_v1* é uma exceção a esta tendência, já que obteve uma acuracidade média relativamente alta (82,1%), mesmo com menos parâmetros (24,4 M) e tamanho de entrada menor (299x299x3) em comparação com alguns outros modelos de alta acuracidade. Com um tempo total de treinamento equivalente a 27 minutos e 19 segundos, pouco mais de 10 minutos mais rápido do que o *fixresnet152*, obteve uma acuracidade média somente 1,3 pontos percentuais inferior, alcançando assim o segundo melhor desempenho.

No Apêndice A, apresenta-se um gráfico de dispersão adicional, com enfoque na relação entre número de parâmetros e tempo total de treinamento.

Figura 36 – Acuracidade média, número de parâmetros e tempo total de treinamento.



Fonte: Elaborada pelo autor.

4.5 Considerações Finais

Os resultados dos experimentos mostraram que a estratégia de correção da resolução das imagens teve um impacto positivo em termos de desempenho, com os modelos "fix" apresentando melhores resultados do que aqueles implementados em suas arquiteturas originais, sem esta estratégia. Dentre os sete modelos que obtiveram acuracidade top-1 média (entre todas as *folds*) acima de 80%, quatro deles apresentam o prefixo "fix": *fixresnet152*, *fixresnet101*, *fixresnet50* e *fixresnet34*. Por outro lado, destaca-se também o desempenho do modelo *inceptionresnet_v1*, que obteve a segunda melhor acuracidade top-1 média (82,1%), mesmo com um número de parâmetros (24,4 M) menor do que a metade do número de parâmetros do modelo com maior acuracidade (*fixresnet152*) e com um tempo total de treinamento pouco mais de 10 minutos mais rápido do que este mesmo modelo.

Além disso, a acuracidade top-5 média (entre todas as *folds*) foi bastante alta, com todos os modelos apresentando resultados acima de 97%. Quanto às médias relativas a *precision*, *recall* e *f1-score*, foi observado que as mesmas equivalem-se às acuracidades médias, possivelmente por conta do balanceamento das classes efetuado através do processo de *oversampling* no *dataset*.

É interessante notar também que os modelos *resnet152*, *resnet101*, *resnet50* e *resnet34* obtiveram melhores posições no *ranking* quando consideradas as médias dos valores máximos de suas acuracidades top-1, em comparação com as médias dos valores médios de suas acuracidades top-1, calculadas entre todas as *folds*.

A comparação com trabalhos anteriores é limitada, tendo em vista que poucos abordam especificamente a classificação de imagens de resíduos eletrônicos. O *dataset* TrashNet (YANG; THUNG, 2016), por exemplo, aborda categorias genéricas, tais como metal, papel, papelão, plástico e vidro, mas com sucatas não oriundas de equipamentos eletroeletrônicos. Já o trabalho desenvolvido por Nowakowski e Pamula (2020) cita geladeiras, máquinas de lavar, monitores e aparelhos de TV. Talvez o trabalho apresentado por Møgaard (2017) seja aquele com determinadas categorias de objetos mais próximas àquelas que foram objeto deste estudo como, por exemplo, ferro, computadores portáteis, telefones celulares e discos rígidos. Neste caso, para um total de 10 classes, o modelo proposto apresentou acurácia de 73%, usando uma arquitetura de rede Inception.

Para promover potenciais melhorias nos modelos propostos neste trabalho, recomenda-se aumentar a variação no conjunto de dados de treinamento, incluindo imagens de itens confundidos com mais frequência, para que os modelos possam aprender a distinguir melhor entre tais itens e seus rótulos verdadeiros. Também podem-se experimentar outras técnicas de *data augmentation* ou novos hiperparâmetros para melhorar a capacidade de generalização e o consequente reconhecimento de imagens inéditas.

CAPÍTULO

5

CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho teve como objetivo apresentar e implementar uma abordagem para classificação de imagens de resíduos eletrônicos (*e-waste*) através de CNNs, contribuindo com a automação dos esforços direcionados ao processo de identificação e separação de resíduos.

Diante disso, foram construídos 18 modelos de CNNs, a partir de modelos originalmente pré-treinados, para avaliar a eficácia de diferentes arquiteturas de redes neurais. O processo de *transfer learning* foi realizado em uma base de dados originalmente composta por 1.670 imagens de resíduos eletrônicos, previamente rotuladas em 11 classes com amostras desbalanceadas. Em decorrência do cenário de *class imbalance*, o número original de amostras foi duplicado e redistribuído no conjunto de treinamento segundo pesos previamente calculados. Adicionalmente, foi adotada uma estratégia de *data augmentation* envolvendo transformações tais como *flipping*, *rotation*, *zooming*, *lighting* e *warping*. A maioria das imagens também passou por um processo de segmentação do *background*, através da rede U^2-Net.

Para medir a perda, ou seja, a diferença entre a probabilidade prevista e os verdadeiros rótulos de classe em uma representação *one-hot encoded*, aplicou-se a função *Cross-Entropy Loss*. Os tamanhos das imagens de entrada foram definidos segundo aqueles utilizados nos modelos pré-treinados. Para a estratégia de correção da resolução, especificamente nos modelos "fix", adotou-se um tamanho de 320 *pixels* durante a etapa de ajuste das últimas camadas.

O método *Stratified K-Fold* foi aplicado para dividir o *dataset* em 5 subconjuntos (*folds*), de modo que os conjuntos de treinamento e de testes correspondessem a 80% e 20% dos dados totais, respectivamente, em cada iteração. O processo de treinamento dos modelos foi conduzido através do "congelamento" das camadas iniciais por 10 *epochs* e, posteriormente, através de *fine-tuning* por mais 10 *epochs* mediante o "descongelamento" de todas as camadas. Ressalta-se, porém, que os modelos "fix" (*fixresnet34*, *fixresnet50*, *fixresnet101*, *fixeresnet152* e *fixefficientnet_b0b*) foram treinados por somente 10 *epochs* com as camadas iniciais "congeladas", a partir de modelos anteriormente desenvolvidos durante este projeto (*resnet34*, *resnet50*, *resnet101*,

resnet152 e *efficientnet_b0b*).

As *learning rates* aplicadas durante a etapa de "congelamento" das camadas iniciais foram encontradas através do processo de *learning rate finder*, segundo o qual inicia-se com uma *learning rate* muito pequena, incrementando a cada *mini-batch* até que a perda passe a piorar ao invés de melhorar. Já para a etapa de "descongelamento" das camadas, aplicou-se o conceito de *discriminative learning rates*, segundo o qual taxas de aprendizado menores são usadas para as camadas iniciais da rede neural e taxas maiores para as camadas finais. Os parâmetros foram ajustados através do algoritmo de otimização *Adam*.

Para avaliar os resultados, as métricas utilizadas foram *top-1 accuracy*, *top-5 accuracy*, *precision*, *recall* e *f1-score*, através dos cálculos de seus valores médios e máximos nas 10 *epochs* finais (*fine-tuning*) do processo de treinamento, tanto em nível global, considerando todas as *folds*, quanto em nível individual, analisando cada *fold* separadamente. Em seguida, para cada modelo, foram analisadas as maiores perdas (*top losses*) dentre todas as *folds*, as matrizes de confusão e os pares de rótulos mais confundidos. Além disso, foram observados aspectos relativos ao número de parâmetros e ao tempo total de treinamento.

Os resultados obtidos mostraram que os modelos com prefixo "fix" obtiveram melhor performance em comparação com suas arquiteturas originais, especialmente no que diz respeito à acuracidade. Em termos gerais, os modelos *fixresnet152* e *inceptionresnet_v1* alcançaram os melhores resultados, com acuracidades top-1 médias (entre todas as *folds*) correspondentes a 83,4% e 82,1%, respectivamente. Já os modelos *inception_v3*, *efficientnet_b3b*, *vgg16* e *efficientnet_b0b* obtiveram os piores resultados, com acuracidade top-1 média abaixo de 78%.

Todos os modelos alcançaram um alto desempenho em relação à acuracidade top-5 média (entre todas *folds*), sendo que *inceptionresnet_v2*, *inceptionresnet_v1* e *resnet34* obtiveram os melhores resultados, com 98,4%. Considerando que o pior desempenho nesta métrica corresponde aos 97,6% obtidos por *efficientnet_b0b* e *fixefficientnet_b0b*, nota-se uma diferença de apenas 0,8 ponto percentual entre o topo e a base dos resultados auferidos.

Também foi observado que as médias relativas a *precision*, *recall* e *f1-score* equivalem-se às acuracidades médias, provavelmente por conta do balanceamento das classes efetuado através do processo de *oversampling* no *dataset*.

O modelo *inceptionresnet_v1* destacou-se pelo fato de ter obtido a segunda melhor acuracidade top-1 média, mesmo com um número de parâmetros (24,4 M) menor do que a metade do número de parâmetros do modelo com maior acuracidade (*fixresnet152*) e com um tempo total de treinamento pouco mais de 10 minutos mais rápido do que este mesmo modelo.

Nota-se também que os modelos *resnet152*, *resnet101*, *resnet50* e *resnet34* obtiveram melhores posições no *ranking* quando consideradas as médias dos valores máximos de suas acuracidades top-1, em comparação com as médias dos valores médios de suas acuracidades top-1, calculadas entre todas as *folds*. Isso sugere que foram capazes de atingir altas pontuações

consistentemente, o que pode indicar estabilidade ao longo do processo de Validação Cruzada.

Em relação às classes confundidas com maior frequência, destacaram-se *Aluminium_Iron* / *Plastic* e vice-versa, *Plastic* / *Battery*, *Aluminium_Iron* / *Battery* e vice-versa, e *Tube_Screen* / *Aluminium_Iron*, sobretudo por conta da similaridade visual de alguns objetos e a consequente dificuldade para que os modelos pudessem distingui-los.

As principais contribuições deste trabalho incluem a demonstração da eficácia de diferentes arquiteturas de redes neurais para classificação de imagens de variados tipos de resíduos eletrônicos, além do sucesso da implementação da estratégia de correção da resolução das imagens nos modelos com prefixo "fix". No entanto, a dissertação também apresenta algumas limitações, como a utilização de uma base de dados com número limitado de imagens e a falta de análise comparativa entre diferentes técnicas de *oversampling* e *data augmentation* ou entre diferentes hiperparâmetros.

Para trabalhos futuros, sugere-se a realização de experimentos com bases de dados mais robustas, aumentando a variação no conjunto de dados de treinamento de modo a incluir imagens de itens frequentemente confundidos, além da utilização de outras técnicas de *oversampling* e *data augmentation*. Posteriormente, podem-se testar novos hiperparâmetros para aprimorar os modelos descritos neste trabalho ou também experimentar outras arquiteturas de Redes Neurais Convolucionais a fim de comparar seus desempenhos no contexto da identificação de *e-waste*.

REFERÊNCIAS

ADEDEJI, O.; WANG, Z. Intelligent waste classification system using deep learning convolutional neural network. **Procedia Manufacturing**, v. 35, p. 607–612, 2019. Citado na página 32.

AWE, O.; MENGISTU, R.; SREEDHA, V. Smart trash net: Waste localization and classification. **CS229 Project Report - Stanford**, p. 1–6, 2017. Citado na página 32.

BALDÉ, C. P.; FORTI, V.; GRAY, V.; KUEHR, R.; STEGMANN, P. The global e-waste monitor. **United Nations University (UNU), International Telecommunication Union (ITU) & International Solid Waste Association (ISWA)**, 2017. Citado nas páginas 27 e 30.

BALDÉ, C. P.; FORTI, V.; KUEHR, R. E-waste statistics: Guidelines on classifications, reporting and indicators. **United Nations University, ViE – SCYCLE**, v. 2, 2018. Citado na página 28.

BUDA, M.; MAKI, A.; MAZUROWSKI, M. A. A systematic study of the class imbalance problem in convolutional neural networks. **Cornell University - arXiv:1710.05381v2**, 2018. Citado na página 56.

DEEPAI. **Inception Module**. s/d. Disponível em: <https://deepai.org/machine-learning-glossary-and-terms/inception-module>. Acesso em: 21/11/2021. Citado na página 44.

FORTI, V. O crescimento do lixo eletrônico e suas implicações globais. **Panorama setorial da Internet**, n. 4, 2019. Citado na página 28.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Optimization for training deep models. In: ____. **Deep Learning**. [S.l.]: MIT Press, 2016. cap. 8. Citado na página 63.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. **Cornell University - arXiv: 1512.03385**, 2015. Citado na página 42.

HOWARD, J.; GUGGER, S. Cross-entropy loss. In: ____. **Deep learning for coders with fastai & PyTorch**. [S.l.]: O'Reilly, 2020. cap. 5. Citado nas páginas 60 e 64.

____. fastai: A layered api for deep learning. **Cornell University - arXiv:2002.04688v2**, 2020. Citado na página 56.

JENSEN, J. R. The error matrix. In: ____. **Introductory Digital Image Processing - A Remote Sensing Perspective**. United States: Pearson Education, 2015. cap. 13, p. 561–562. Citado na página 69.

KOHAVI, R. A study of cross validation and bootstrap for accuracy estimation and model selection. **Joint Conference on Artificial Intelligence**, p. 1–7, 1995. Citado na página 57.

KONATÉ, A. Artificial neural network: a tool for approximating complex functions. **HAL archives-ouvertes.fr**, p. 2, 2019. Citado na página 35.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Neural Information Processing Systems**, v. 25, 2012. Citado nas páginas 38 e 40.

MAGALINI, F.; KUEHR, R.; BALDÉ, C. P. E-waste in latin america. **GSMA Latin America & UNU Institute for the Advanced Study of Sustainability (UNU-IAS)**, 2015. Citado nas páginas 29 e 30.

MAO, W.-L.; CHENA, W.-C.; WANG, C.-T.; LIN, Y.-H. Recycling waste classification using optimized convolutional neural network. **Resources, Conservation and Recycling**, v. 164, n. 105132, 2021. Citado na página 33.

MASHRU, R. S. **Understanding FixRes - Fixing the train-test resolution discrepancy**. 2021. Disponível em: <https://ravimashru.dev/blog/2021-09-22-fix-res/>. Acesso em: 09/12/2022. Citado na página 61.

MATHWORKS. **resnet50**. 2017. Disponível em: <https://www.mathworks.com/help/deeplearning/ref/resnet50.html>. Acesso em: 21/11/2021. Citado na página 43.

MITTAL, G.; YAGNIK, K. B.; GARG, M.; KRISHNAN, N. C. Spotgarbage: Smartphone app to detect garbage using deep learning. **UbiComp '16: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing**, p. 940–945, 2016. Citado na página 32.

MØGAARD, L. **Waste Electrical and Electronics Equipment Classification using Deep Neural Network Sensor Fusion**. [S.l.], 2017. Disponível em: <http://www2.compute.dtu.dk/pubdb/pubs/6976-full.html>. Acesso em: 28 nov. 2021. Citado nas páginas 32, 55 e 87.

MUELLER, Z. **New LR Finder Output?!** 2021. Disponível em: <https://forums.fast.ai/t/new-lr-finder-output/89236/3>. Acesso em: 09/12/2022. Citado na página 62.

NETO, S. **Identificação de Risco de Crédito (Statlog – German Credit Data)**. 2018. <https://ensinandomaquinasblog.wordpress.com/category/sem-categoria/>. Citado na página 69.

NEUROHIVE. **Popular Networks**. 2018. Disponível em: <https://neurohive.io/en/popular-networks/vgg16/>. Acesso em: 04/12/2021. Citado na página 41.

NOWAKOWSKI, P.; PAMULA, T. Application of deep learning object classifier to improve e-waste collection planning. **Waste Management**, v. 109, p. 1–9, 2020. Citado nas páginas 32 e 87.

NUERONIO.AI. **Understanding ConvNets (CNN)**. 2018. Disponível em: <https://medium.com/neuronio/understanding-convnets-cnn-712f2afe4dd3>. Acesso em: 21/11/2021. Citado na página 39.

QIN, X.; ZHANG, Z.; HUANG, C.; DEHGHAN, M.; ZAIANE, O. R.; JAGERSAND, M. U2-net: Going deeper with nested u-structure for salient object detection. **Cornell University - arXiv:2005.09007v1**, 2020. Citado na página 52.

RAMSUNDAR, B.; ZADEH, R. B. Fully connected deep networks. In: ____. **TensorFlow for Deep Learning**. [S.l.]: O'Reilly, 2018. cap. 4. Citado na página 40.

RAMSURRUN, N.; SUDDUL, G.; ARMOOGUM, S.; FOOGOOA, R. Recyclable waste classification using computer vision and deep learning. **2021 Zooming Innovation in Consumer Technologies Conference (ZINC)**, 2021. Citado na página 33.

RAWAT, W.; WANG, Z. Deep convolutional neural networks for image classification: A comprehensive review. **Neural Computation**, v. 29, p. 1–98, 2017. Citado na página 37.

SHI, C.; TAN, C.; WANG, T.; WANG, L. A waste classification method based on a multilayer hybrid convolution neural network. **Applied Sciences**, v. 11(18), n. 8572, 2021. Citado na página 33.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale recognition. **Cornell University - arXiv: 1409.1556v1**, 2014. Citado na página 41.

SIRSAT, M. **What is Confusion Matrix and Advanced Classification Metrics ?** 2019. <https://manisha-sirsat.blogspot.com/2019/04/confusion-matrix.html>. Citado na página 70.

SMITH, L. N. Cyclical learning rates for training neural networks. **Cornell University - arXiv:1506.01186v1**, 2015. Citado na página 62.

STANFORD. **CS231n: Convolutional Neural Networks for Visual Recognition**. 2021. Disponível em: <https://cs231n.github.io/convolutional-networks/>. Acesso em: 21/11/2021. Citado na página 38.

STRATFOR. **The Geopolitics of Rare Earth Elements**. 2019. Disponível em: <https://worldview.stratfor.com/article/geopolitics-rare-earth-elements>. Acesso em: 07/11/2021. Citado na página 28.

SUBRAMANIAN, N. B. **Cross Validation**. 2019. <https://aiaspirant.com/cross-validation/>. Citado na página 58.

SUPERDATASCIENCE. **The Ultimate Guide to Convolutional Neural Networks (CNN)**. 2018. Disponível em: <https://www.superdatascience.com/blogs/the-ultimate-guide-to-convolutional-neural-networks-cnn>. Acesso em: 15/11/2021. Citado na página 36.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V.; ALEMI, A. Inceptions-v4, inception-resnet and the impact of residual connections on learning. **Cornell University - arXiv: 1602.07261**, 2016. Citado na página 46.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. **Cornell University - arXiv: 1409.4842**, 2014. Citado na página 40.

SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J. Rethinking the inception architecture for computer vision. **Cornell University - arXiv: 1512.00567**, 2015. Citado na página 45.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. **Cornell University - arXiv: 1905.11946**, 2019. Citado nas páginas 50 e 51.

TANG, Y. Deep learning using linear support vector machine. **Cornell University - arXiv: 1306.0239**, v. 4, 2013. Citado na página 39.

TECHCRUNCH. **Auto-Trash sorts garbage automatically at the TechCrunch Disrupt Hackathon**. 2016. Disponível em: <https://techcrunch.com/2016/09/13/auto-trash-sorts-garbage-automatically-at-the-techcrunch-disrupt-hackathon>. Acesso em: 15/11/2021. Citado na página 31.

TOUVRON, H.; VEDALDI, A.; DOUZE, M.; JÉGOU, H. Fixing the train-test resolution discrepancy. **Cornell University - arXiv: 1906.06423**, 2019. Citado na página 50.

____. Fixing the train-test resolution discrepancy: Fixefficientnet. **Cornell University - arXiv: 2003.08237**, 2020. Citado na página 51.

TOWARDSDATASCIENCE. **Illustrated: 10 CNN Architectures**. 2019. Disponível em: <https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d>. Acesso em: 21/11/2021. Citado nas páginas 43, 45, 46, 47, 48 e 49.

WU, J. Introduction to convolutional neural networks. **National Key Lab for Novel Software Technology, Nanjing University**, p. 1–31, 2017. Citado na página 36.

XIE, S.; GIRSHICK, R.; DOLLÁR, P.; TU, Z.; HE, K. Aggregated residual transformations for deep neural networks. **Cornell University - arXiv: 1611.05431**, 2016. Citado nas páginas 48 e 49.

YANG, M.; THUNG, G. Classification of trash for recyclability status. **CS229 Project Report - Stanford**, p. 1–6, 2016. Citado nas páginas 32, 33, 43 e 87.

YOSINSKI, J.; CLUNE, J.; BENGIO, Y.; LIPSON, H. How transferable are features in deep neural networks? **Cornell University - arXiv:1411.1792**, 2014. Citado na página 62.

ZHANG, Q.; ZHANG, X.; MU, X.; WANG, Z.; TIAN, R.; WANG, X.; LIU, X. Recyclable waste image recognition based on deep learning. **Resources, Conservation and Recycling**, v. 171, n. 105636, 2021. Citado na página 32.

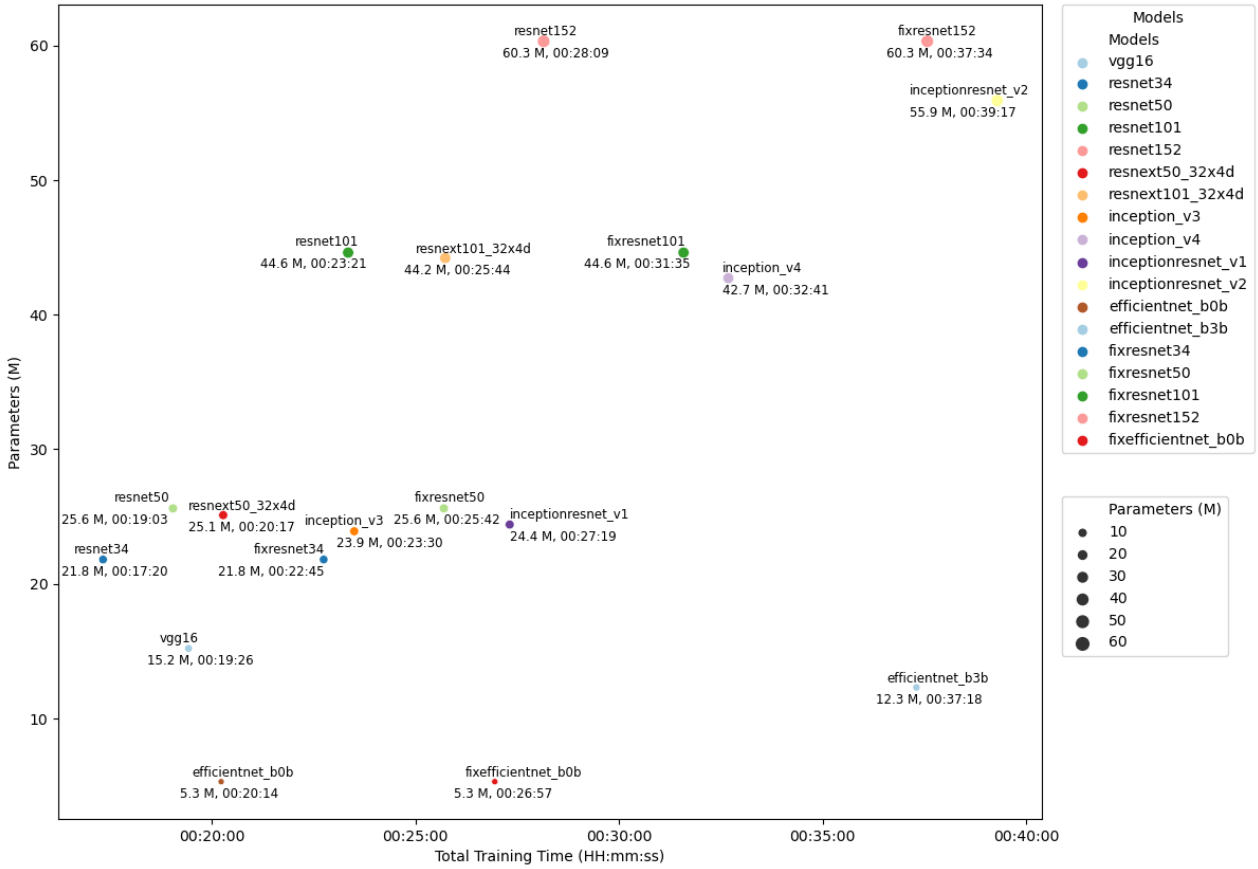
APÊNDICE

A

GRÁFICO DE DISPERSÃO: PARÂMETROS X TEMPO DE TREINAMENTO

O gráfico de dispersão na Figura 37 exibe a relação entre o número de parâmetros (eixo y) e o tempo total de treinamento (eixo x), para cada modelo.

Figura 37 – Número de parâmetros e tempo total de treinamento.



Fonte: Elaborada pelo autor.


ICMC
USP
SÃO CARLOS