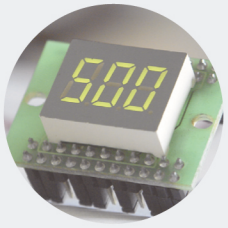
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Classificação de Scrap de Capply com recurso a metodologias Deep Learning

FILIPE MANUEL PEREIRA CASTRO
Julho de 2022

POLITÉCNICO
DO PORTO

POLITÉCNICO DO PORTO
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Classificação de *Scrap* de *Capply* com recurso a metodologias *Deep Learning*

Filipe Manuel Pereira Castro

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
Instituto Superior de Engenharia do Porto

Julho, 2022

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Automação e Sistemas.

Candidato: Filipe Manuel Pereira Castro, Nº 1200114, 1200114@isep.ipp.pt

Orientação Científica: Ramiro de Sousa Barbosa, rsb@isep.ipp.pt

Empresa: Continental Mabor - Indústria de Pneus, S. A.

Orientador: Alberto Pereira, alberto.pereira@conti.de

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Julho, 2022

Agradecimentos

Ao longo de todo este percurso académico até ao momento percorrido, tornou-se uma longa viagem, que demonstrou ser um caminho recheado de inúmeros desafios, incertezas, alegrias e até tristezas. Apesar do caminho percorrido com inúmeros acontecimentos, sempre tive o apoio incondicional de amigos e família.

De forma a que fosse possível a realização deste trabalho, foram necessários apoios e incentivos por parte da família, amigos, colegas e professores, sem os quais, este, não se teria tornado uma realidade.

Trilhar este caminho só foi possível com o apoio de todos, por isso, a todos agradeço eternamente.

Ao Professor Doutor Ramiro de Sousa Barbosa, orientador desta dissertação, desejo manifestar um especial agradecimento pelo apoio, disponibilidade, orientação e amizade.

Ao Engenheiro Alberto Pereira, por todo o apoio, ensinamentos prestados e rigor científico.

Ao Engenheiro Francisco Ferreira, por todo o apoio e incentivo constante.

Ao Departamento de Engenharia III e à Continental Mabor, pela oportunidade em desenvolver esta dissertação e por todo o apoio fornecido.

Aos Professores do Instituto Superior de Engenharia do Porto, por todo o entusiasmo, lições e conhecimentos transmitidos ao longo deste percurso.

A toda a minha família, por todo o apoio, incentivo, carinho e paciência que me deram durante todo este percurso académico.

Resumo

A nível industrial, o processo de controlo de qualidade de produtos é realizado com diferentes técnicas e especificações, adequadas a cada processo. No entanto, por vezes, as técnicas utilizadas têm falhas que se traduzem num prejuízo para as empresas.

Os avanços tecnológicos seguem num ritmo acelerado e o mercado tem vindo a absorver as inovações nos seus mais diversos setores. Numa procura pela transformação digital, as empresas passaram a investir mais em soluções que gerem um diferencial competitivo frente à concorrência. Nesse sentido, o conceito de Inteligência Artificial (IA) ganhou bastante importância.

Nos últimos anos temos assistido a uma adoção acelerada do *Machine Learning* (ML) como parte integrante da Indústria 4.0, na qual a digitalização está refazendo a indústria. Essa última onda de iniciativas é marcada pela introdução de sistemas inteligentes e autónomos, alimentados por grandes quantidades de dados e por *Deep Learning* (DL). Uma poderosa geração de IA que promove a inspeção de qualidade no chão de fábrica.

Este trabalho visa investigar e implementar técnicas supervisionadas de *Deep Learning*, aliadas à visão computacional, para a implementação de um sistema de classificação automático de imperfeições de *Capply*. Para esse efeito, inicialmente, foi realizada uma revisão bibliográfica sobre o estado da arte, passando de seguida à implementação e comparação do desempenho de várias arquiteturas utilizando as métricas adequadas. Para a execução desta tarefa foi necessário recolher e fazer um pré-processamento dos dados (imagens de bobines de *capply*). Foi ainda, desenvolvida uma aplicação Web que permite testar e avaliar os resultados e por último, foi também desenvolvido e implementado um sistema de classificação em contexto real.

Resumidamente, os resultados deste trabalho demonstraram o grande potencial das metodologias de *Deep Learning* aplicadas ao controlo de qualidade na indústria.

Palavras-Chave: Controlo de Qualidade, Inteligência Artificial, *Machine Learning*, *Deep Learning*, Visão Computacional, Indústria 4.0, Classificação, CNN.

Abstract

At an industrial level, the product quality control process is carried out using different techniques and specifications, suitable for each process. However, sometimes the techniques used have flaws that translate into a loss for companies.

Technological advances continue at an accelerated pace and the market has been absorbing innovations in its most diverse sectors. In a search for digital transformation, companies began to invest more in solutions that generate a competitive edge against the competition. In this sense, the concept of Artificial Intelligence (AI) has gained a lot of importance.

In recent years we have seen an accelerated adoption of Machine Learning (ML) as an integral part of Industry 4.0, in which digitalization is remaking the industry. This latest wave of initiatives is marked by the introduction of intelligent and autonomous systems, powered by large amounts of data and by Deep Learning (DL). A powerful generation of AI that drives quality inspection on the shop floor.

This work aims to investigate and implement supervised Machine Learning techniques, combined with computer vision, for the implementation of an automatic classification system for Capply imperfections. For this purpose, initially, a bibliographic review was carried out on the state of the art, followed by the implementation and comparison of the performance of several architectures using the appropriate metrics. To perform this task, it was necessary to collect and preprocess the data (capply reel images). A web application was also developed that allows testing and evaluating the results and finally, a classification system in real context was also developed and implemented.

Briefly, the results of this work demonstrated the great potential of Machine Learning methodologies applied to quality control in the industry.

Keywords: Quality Control, Artificial Intelligence, Machine Learning, Deep Learning, Computer Vision, Industry 4.0, Classification, CNN.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

2D	bidimensional
3D	tridimensional
ACC	*Accuracy*
ADS	*Automation Device Specification*
AE	*AutoEncoder*
AG	*Aktien Gesellschaft*
ANN	*Artificial Neural Networks*
AUC	*Area Under Curve*
BCE	*Binary Cross-Entropy*
CEP	Controlo Estatístico de Processo
CEQ	Controlo Estatístico de Qualidade
CNN	*Convolutional Neural Networks*
CQ	Controlo de Qualidade
CV	*Computer Vision*
DBN	*Deep Belief Networks*
DL	*Deep Learning*
DT	*Decision Trees*
FC	*Fully Connected*
FE	*Feature Extraction*
FN	Falso Negativo
FP	Falso Positivo
FPR	*False Positive Rate*

FT *Fine-Tuning*

GELU *Gaussian Error Linear Unit*

GQT Gestão de Qualidade Total

HSV *Hue, Saturation and Value*

IA Inteligência Artificial

IDE *Integrated Development Environment*

ILSVRC *ImageNet Large Scale Visual Recognition Challenge*

KM *Karcass Machine*

KNN *K-Nearest Neighbor*

LR *Learning Rate*

LSTM *Long Short Term Memory*

ML *Machine Learning*

NB *Naïve Bayesian*

NDT *Non-Destructive Testing*

NN *Neural Networks*

PLC *Programmable Logic Controller*

PU *Pressure Unit*

QC *Quality Control*

RBM *Restricted Boltzmann Machines*

ReLU *Rectified Linear Unit*

RGB *Red, Green and Blue*

RL *Reinforcement Learning*

RNN *Recurrent Neural Networks*

ROC *Receiver Operating Characteristic*

SGD *Stochastic gradient descent*

SiLU *Sigmoid Linear Unit*

Sn	Sensibilidade
Sp	Especificidade
SUV	*Sport Utility Vehicles*
SVM	*Support Vector Machine*
Tanh	*Hyperbolic tangent*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TIC	Tecnologias de Informação e Comunicação
TPR	*True Positive Rate*
TPU	*Tensor Processing Unit*
UDP/IP	*User Datagram Protocol/Internet Protocol*
VGG	*Visual Geometry Group*
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

Capítulo 1

Introdução

Neste capítulo é feito o enquadramento do trabalho e as motivações subjacentes ao seu desenvolvimento. Os principais objetivos são ainda enumerados e é apresentada uma breve descrição da metodologia aplicada. É apresentada uma calendarização do planeamento do trabalho. Este capítulo finaliza com uma breve explicação da estrutura da presente dissertação.

1.1 Contextualização

No final do século XX, as indústrias iniciaram uma procura por soluções para automatizar os seus processos com o auxílio da computação e da automação. Da mesma forma, o processo de Controlo de Qualidade (CQ) foi modernizado e automatizado, embora em muitos casos ainda seja feito por humanos, o que traz vários problemas, nomeadamente erros associados à repetibilidade de tarefas, resultando em custos elevados para a empresa. Recentemente, os impressionantes avanços em Inteligência Artificial (IA), impulsionados pelo aumento exponencial do poder computacional e pela disponibilidade de grandes quantidades de dados, nos levaram a uma nova era industrial, que chamamos de 4ª revolução industrial ou indústria 4.0, que também possibilitou o desenvolvimento de soluções de CQ automáticas e inteligentes.

Deep Learning (DL) é um subcampo de *Machine Learning* (ML) que por sua vez, ML é uma vertente específica de IA. ML tem como propósito treinar máquinas para

aprender com dados, consistindo na execução de algoritmos que criam automaticamente modelos de representação de conhecimento com base num conjunto de dados. Ao deixar o algoritmo "aprender", a ajustar iterativamente o modelo de representação do conhecimento, é possível melhorar o seu desempenho. Após esta fase de treino, o modelo tem potencial para fazer previsões, diagnósticos de qualidade em situações futuras, relacionados a padrões históricos com a mínima intervenção humana, otimizando processos, aumentando a confiabilidade e gerando economia. As soluções de CQ atuais também utilizam técnicas de processamento de imagem aliadas a algoritmos de ML e DL para realizar tarefas de inspeção mais eficientes e rápidas. A aplicação de sistemas automatizados de controlo visual tem sido capaz de melhorar a produtividade da indústria, estabelecendo critérios confiáveis para controlar a qualidade de produtos e serviços com custos reduzidos. Aplicando DL a este tipo de sistemas, é possível tornar os sistemas mais flexíveis e adaptáveis, capazes de aprender de forma evolutiva, tornando os sistemas mais aptos para atingir níveis de eficiência cada vez mais elevados.

Um dos materiais existentes e de grande importância num pneu, e que esta dissertação se focará é o material denominado por *capply*. Este material é constituído por borracha e fio têxtil, sendo que o processo de fabrico deste consiste na união dos dois materiais, dando origem a uma tira e por último esta tira é enrolada numa bobine. Este é um processo delicado e que pode originar várias imperfeições no produto final (bobine de *capply* enrolado), tais como, mau enrolamento/distribuição irregular, excesso de adesividade, tira dobrada/torcida, falha de borracha, entre outros. Para além destas imperfeições que podem ser causadas pela própria máquina, ainda existe uma imperfeição que pode ser causada pelo mau uso da bobine de *capply* que é a bobine amassada/danificada.

As bobines que apresentam imperfeições no material são enviadas para uma máquina denominada por recuperadora de *capply* e serão então analisadas por um operador, que identificará visualmente qual a sua imperfeição. Depois de classificada a imperfeição é então introduzida no sistema, onde dará início ao processo de recuperação da bobine (remover material com defeito) ou em alguns casos segregar por completo o material da bobine. Uma parte importante deste processo é que assim que a bobine que contém material para scrapar se encontre em bom estado, ou seja, o material existente ainda na bobine pode ser usado para produção, a máquina parasse de scrapar a bobine. Atualmente para que isso seja feito o operador tem que estar constantemente na máquina a analisar as bobines para identificar quando é que aparece material "OK", mas isso dificilmente acontece. Este trabalho consiste na aplicação de técnicas de ML de forma a classificar automaticamente diferentes tipos de imperfeições em bobines de *capply* e ao mesmo tempo identificar quando é que aparece material em bom estado.

1.2 Objetivos

Nesta nova revolução industrial, a introdução da IA em ambientes industriais é de enorme importância, permitindo a implementação de análises avançadas de dados para apoiar as tarefas de monitorização, diagnóstico, previsão e otimização. O principal objetivo desta dissertação é o de desenvolver e implementar metodologias de ML eficientes para a classificação de *Scrap* (imperfeições) de *capply*.

De seguida são apresentados alguns dos subobjetivos deste trabalho:

1. Coletar o *dataset* de imagens de bobines de *capply* com características específicas;
2. Comparar o desempenho de diferentes arquiteturas implementadas em classificação de imagens com o *dataset* ImageNet;
3. Comparar o desempenho de diferentes arquiteturas presentes na literatura;
4. Analisar e comparar o impacto do pré-processamento de imagens;
5. Desenvolver uma plataforma de apoio para monitorização, configuração, manutenção e testes;
6. Implementar uma solução protótipo na empresa.

1.3 Metodologia

Na primeira fase do desenvolvimento desta dissertação existiu uma ampla revisão de literatura sobre metodologias de ML e as diferentes arquiteturas utilizadas em classificações de imagens. Paralelamente, foram exploradas metodologias de classificação com viabilidade para o estudo do conjunto de dados.

Depois de um levantamento alusivo às metodologias, métodos e técnicas presentes na literatura foram desenvolvidos métodos de leitura dos dados assim como técnicas de processamento de dados de imagem. Na avaliação de desempenho dos algoritmos implementados as métricas utilizadas foram investigadas, desenvolvidas e implementadas. Adicionalmente, foram implementadas funções para a visualização dos dados, do efeito do processamento aplicado às imagens, e dos resultados obtidos. Foram selecionadas e implementadas arquiteturas/modelos mais adequados para a tarefa, assim como a criação de funções a utilizar no processo de aprendizagem e no processo de teste.

Na fase final deste trabalho foram conduzidos diferentes testes preparatórios no sentido de ajustar e identificar os limites de simulação por forma a que fosse possível fundamentar uma discussão comparativa de resultados. Após esta fase de preparação foram conduzidos todos os testes conjugando com a análise crítica

e comparativa dos resultados obtidos para o conjunto de dados e modelos implementados.

1.4 Plano de Trabalho

Na Tabela 1.1 é apresentada a calendarização do presente trabalho.

Tabela 1.1: Calendarização do trabalho.

Tarefa	Fevereiro	Março	Abril	Maio	Junho	Julho
Exposição do problema						
Estudo bibliográfico						
Caracterização do local de realização do projeto						
Recolha de dados e preparação do *dataset*						
Escolha e treino dos modelos						
Testes e análise dos modelos						
Testes exaustivos do algoritmo e análise dos resultados						
Escrita da Dissertação						

1.5 Organização da Dissertação

Esta dissertação encontra-se organizada em seis capítulos:

Capítulo 1: Introdução

O primeiro capítulo contextualiza o trabalho, estabelece as motivações que levaram ao desenvolvimento do mesmo, assim como, define os objetivos do trabalho. Adicionalmente, está presente uma breve descrição das metodologias utilizadas e a calendarização do trabalho. Este capítulo apresenta, ainda, a organização da dissertação, dando uma breve descrição do que será referido em cada capítulo.

Capítulo 2: Caracterização do local de realização do projeto

O segundo capítulo faz uma breve apresentação da empresa onde se desenvolveu o projeto, Continental Mabor – Indústria de Pneus S.A. Efetua-se uma descrição dos departamentos presentes na empresa, descreve-se os componentes presentes num Pneu e o respetivo processo produtivo da empresa.

Capítulo 3: Revisão Bibliográfica

O terceiro capítulo trata dos precedentes teóricos que suportam o trabalho desenvolvido e por isso é subdividido em quatro subcapítulos. O primeiro subcapítulo é uma breve revisão focada no controlo de qualidade seguido de um subcapítulo sobre visão computacional. Uma revisão sobre manipulação, exploração e processamento de dados é descrita no terceiro subcapítulo. Por último, o quarto subcapítulo consiste numa revisão sobre *Machine Learning* e *Deep Learning* com foco nas metodologias e nas arquiteturas utilizadas em classificação de imagens.

Capítulo 4: Metodologias e Desenvolvimento

O quarto capítulo refere-se aos procedimentos adotados para o desenvolvimento do trabalho, nomeadamente, ferramentas utilizadas e as arquiteturas implementadas.

Capítulo 5: Resultados e Discussão

O quinto capítulo foca-se nos resultados e discussão do trabalho desenvolvido, sendo dividido em quatro subcapítulos, que tratam da discussão dos resultados obtidos ao longo do processo de treino, resultados dos testes após processo de treino, apresentação da plataforma Web e por último o sistema protótipo.

Capítulo 6: Conclusões e Perspetivas Futuras

O sexto e último capítulo foca-se nas principais conclusões e no trabalho futuro.

Capítulo 2

Caracterização do Local de Realização do Projeto

Neste capítulo é feita uma apresentação e caracterização do local de realização do projeto. É apresentado um resumo da história da empresa e do seu processo produtivo.

2.1 História da empresa

Em seguida, será apresentada a história e a caracterização da empresa onde foi realizado o projeto. Começando por abordar a história do grupo Continental, seguida pela história da Mabor e o relacionamento entre as duas empresas. Por fim será descrito o processo de produção da Continental Mabor.

2.1.1 Continental AG

Figura 2.1: Logótipo Continental AG.

A Continental (Figura 2.1) foi fundada em Hanôver em 1871 (Figura 2.2) como uma sociedade por ações "Continental - *Caoutchouc - und guta-percha Compagnie*". A sua fabricação principal na fábrica em Hanôver incluía produtos suaves de borracha, tecidos emborrachados, e pneus maciços para carruagens e bicicletas [1].

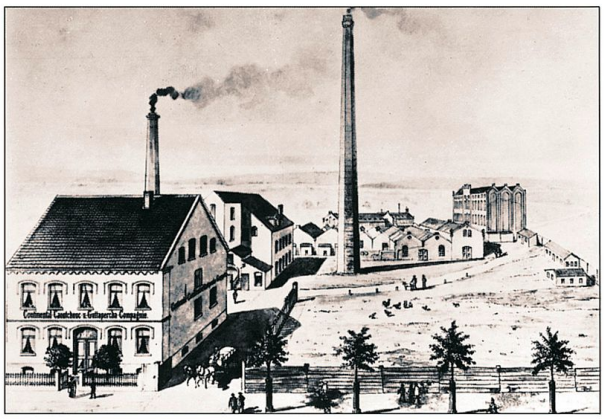


Figura 2.2: Hanôver 1871.

Em 1898, os êxitos iniciais no desenvolvimento e produção foram comemorados com a produção de pneus sem desenho de piso (lisos) [1] para automóveis. No virar do século, o primeiro dirigível alemão LZ 1 (Figura 2.3) utiliza material de borracha Continental para vedação dos recipientes de gás.

Figura 2.3: Dirigível Alemão LZ 1.

Em 1904, a Continental tornou-se na primeira companhia mundial a desenvolver e a produzir pneus para automóveis com desenho de piso e em 1905 inicia a produção de pneus antiderrapantes com rebites (Figura 2.4) percursores dos pneus para neve com pinos, semelhante aos pneus com pregos posteriores, e três anos mais tarde, inventou a jante desmontável para automóveis ligeiros, permitindo assim, a economia de tempo e de esforço na mudança do pneu [1].

Figura 2.4: Pneus com rebites, 1905.

Em 1909, o aviador francês Louis Blériot, efetua a primeira travessia aérea do canal da Mancha. A fuselagem e asas do seu avião são revestidas com material da marca Continental (Figura 2.5) [1].

Figura 2.5: Avião com material da marca Continental, 1909.

No final de 1920, a empresa fundiu-se com as principais empresas de produção de borracha para formar assim, a "Continental Gummi - Weke AG".

Em 1951 iniciou-se a produção de correias transportadores com telas metálicas. Em 1955, foram a primeira empresa a desenvolver molas para camiões e autocarros (Figura 2.6). A produção em série de pneus radiais ligeiros começou em 1960. Cerca de 30 anos depois, trouxeram, para o mercado, os primeiros pneus ecológicos para veículos de passageiros [1].

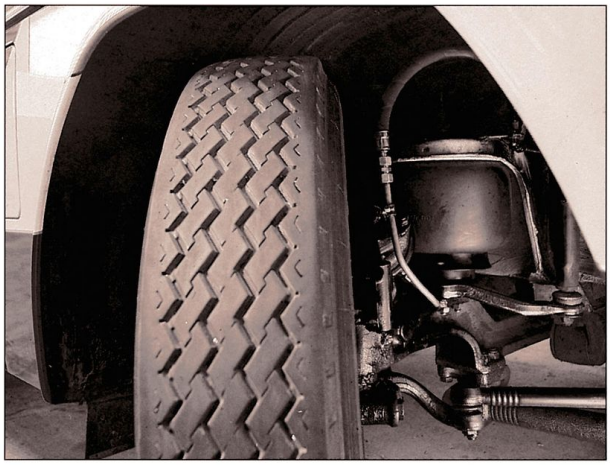


Figura 2.6: Primeira empresa a desenvolver molas, 1955.

Em 1955, a divisão *Automotive Systems* foi fundada com a finalidade de intensificar a atividade comercial e desenvolvimento de sistemas automotivos para a indústria automóvel. Apresentaram, em 1997 a tecnologia-chave para sistemas ecológicos de arranque do motor e geradores convencionais, de híbridos.

Hoje, a Continental está entre os 5 maiores fornecedores mundiais da indústria automóvel. Como fornecedor de sistemas de travagem, sistemas e componentes para acionamentos e chassis, instrumentação, soluções de *infotainment*, eletrónica de veículos, pneus e elastómeros técnicos, a Continental contribui para uma maior segurança na condução e na proteção ambiental global. A Continental é também um parceiro competente na comunicação automobilística em rede.

Em 2007, o grupo Continental adquire a Siemens VDO Automotive AG e traça um novo marco na sua história, tornando-se num dos cinco maiores fornecedores mundiais da indústria automóvel, ao mesmo tempo, fortalecendo a sua posição na Europa, América do Norte e Ásia. Não apenas focada na divisão de pneumáticos, a Continental AG divide-se em dois grandes grupos (Figura 2.7): o grupo Automotivo e o grupo da Borracha. Estes grupos, por sua vez dividem-se em três divisões [2].

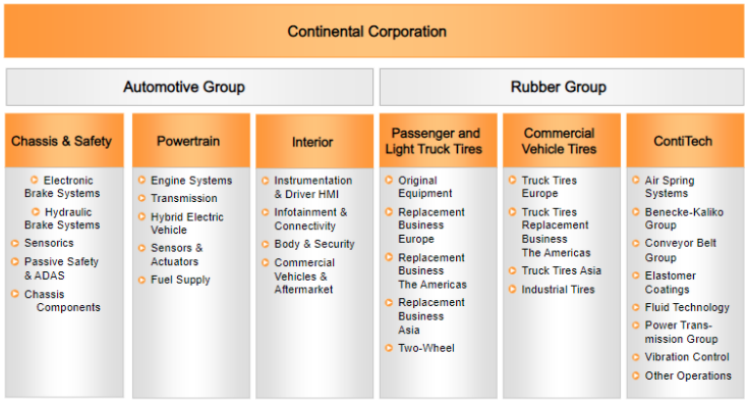


Figura 2.7: Áreas de negócio da Continental AG.

O Grupo Continental é dividido em "Sistemas Automotivos" e "Componentes de Borracha", e consiste em cinco divisões:

- *Chassis and Safety*, concentram-se em tecnologias modernas para a segurança ativa e passiva e de dinâmica do veículo (Figura 2.8).

Figura 2.8: *Chassis and Safety*.

- *Powertrain*, representa soluções de sistemas inovadores e eficientes para o *powertrain* do presente e do futuro, para os veículos de todas as categorias (Figura 2.9).

Figura 2.9: *PowerTrain*.

- Interior, combina todas as atividades relacionadas com a apresentação e gestão de informações do veículo (Figura 2.10).

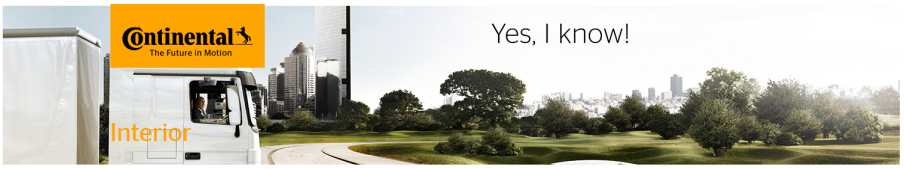


Figura 2.10: Interior.

- Pneus, oferecem os pneus certos para cada aplicação - desde veículos de passageiros, camiões e autocarros, até veículos especiais, motos e bicicletas. A Continental Pneus representa uma excelente transmissão de forças, um rastreamento excecionalmente confiável em todas as condições meteorológicas e um custo elevado de eficácia (Figuras 2.11 e 2.12).

Figura 2.11: Pneus ligeiros.

Figura 2.12: Pneus pesados.

- *ContiTech*, desenvolve e produz peças funcionais, componentes e sistemas para a indústria automóvel e para outras indústrias-chave (Figura 2.13).

Figura 2.13: *ContiTech*.

Hoje a Continental *Aktien Gesellschaft* (AG) encontra-se entre os cinco maiores fornecedores mundiais da indústria automóvel e, desde então, tem vindo a crescer um pouco por todo o mundo. Atualmente está em 527 localizações e em 58 países [3], tal como se observa na Figura 2.14.

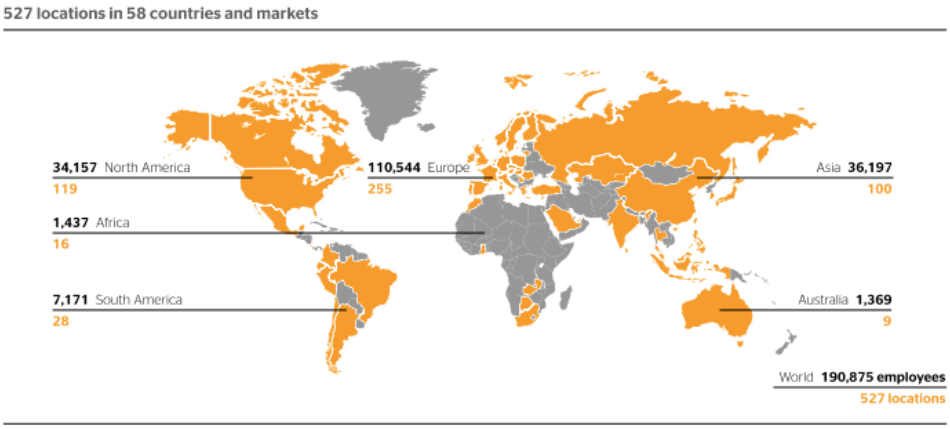


Figura 2.14: Localização das fábricas do grupo Continental AG.

Na divisão de pneus o grupo Continental encontra-se presente em 12 países (Figura 2.15), contando com 13 fábricas.

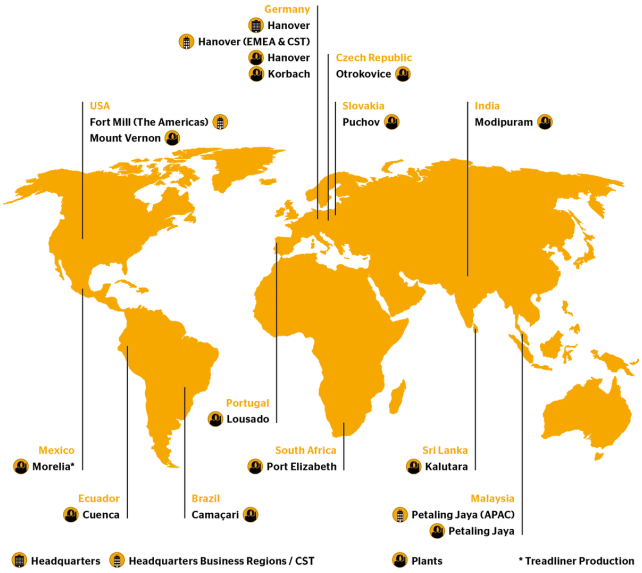


Figura 2.15: Localização das fábricas de pneus da Continental AG.

A Continental AG está representada em Portugal (Figura 2.16) através de seis empresas em quatro concelhos, onde cinco empresas estão situadas no norte do País e uma no centro.

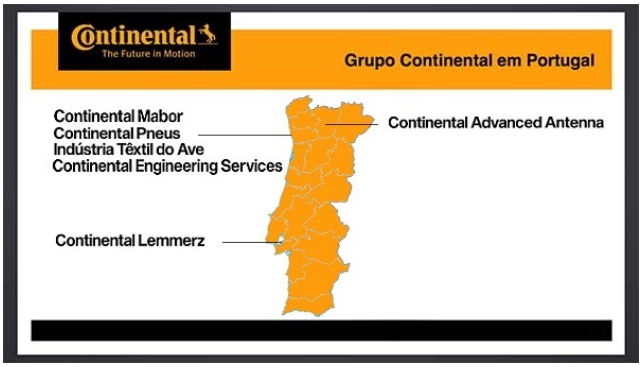


Figura 2.16: Grupo Continental AG em Portugal.

2.1.2 Continental Mabor S.A

A Mabor (Manufatura Nacional da Borracha) (Figura 2.17) nasce em 1940 em Lousado, Vila Nova de Famalicão. A Mabor inicia assim a produção de pneus para pesados, ligeiros, motos, câmaras de ar e pisos para recauchutagem, sob a marca "Mabor General".

Figura 2.17: Mabor S.A. 1946.

A Continental Mabor nasceu em dezembro de 1989, quando em Portugal já se produziam pneus há 41 anos. Quando a AutoEuropa surge em Portugal, e o estado obriga a incorporação de componentes produzidos em Portugal, o grupo Continental AG identifica uma oportunidade de negócio.

Com vista nesta oportunidade de negócio a Continental Mabor nasce da união de duas empresas de renome dentro do ramo da manufatura da borracha. A Mabor (Figura 2.18) [4], com renome a nível nacional, foi a primeira empresa de pneumáticos em Portugal, com a sua laboração a iniciar-se em 1946 com o apoio da Americana General Tire. E a Continental AG com renome a nível mundial nas tecnologias da manufatura da borracha.

Figura 2.18: Logótipo Mabor S.A. [4].

Com a parceria destas empresas, em julho de 1990, inicia-se o programa de restruturação que transformou as antigas instalações da Mabor na mais moderna das 21 unidades existentes no grupo Continental. Partindo de uma produção média diária de 5000 pneus/dia em 1990, a Continental Mabor quadruplica este número em 1996, chegando aos 21000 pneus/dia demarcando-se assim como uma das empresas do grupo com o melhor índice de produtividade.

Em Portugal a Continental Mabor (Figura 2.19) continua a crescer, e conta com uma produção muito variada, quer em tipos de pneus, medidas e marcas. Atualmente a Continental Mabor produz desde pneus destinados a *Sport Utility Vehicles* (SUV), pneus de alto desempenho, pneus ContiSeal aos pneus ContiSilent. Com uma gama de medidas de produção de pneus desde as jantes 14 ”até às jantes 22”. Em 2016 a produção diária de pneus era de 56000 pneus, onde 98% da produção se destina à exportação, neste mesmo ano a Continental Mabor contava com 1901 colaboradores no seu quadro permanente. Atualmente a Continental Mabor conta com mais de 2500 colaboradores na parte dos pneus ligeiros e pesados.

Figura 2.19: Vista aérea da Continental Mabor (2019).

2.2 Fluxograma e Processo Produtivo

O sistema de produção de um pneu na Continental Mabor é constituído por três fases:

- Receção das matérias primas necessárias para o fabrico do produto;
- Processo produtivo para a construção do produto;
- Armazenamento e expedição do produto acabado.

Para perceber o processo produtivo, é necessário conhecer os componentes que dão origem a um pneu. Através da Figura 2.20 podem ser identificados os diferentes componentes que constituem um pneu, formando assim o produto final.

Figura 2.20: Componentes que constituem um pneu [5].

Legenda:

1. Piso;
2. Cinta têxtil (espiral têxtil);
3. Tela metálica (*breaker*);
4. Tela têxtil;
5. Camada;
6. Parede Lateral;
7. Reforço têxtil ou metálico;
8. Cunha (núcleo de talão);
9. Talão.

O processo produtivo da Continental Mabor está dividido em cinco fases essenciais, asseguradas por cinco departamentos (Figura 2.21), que constituem as etapas necessárias para a construção do pneu.

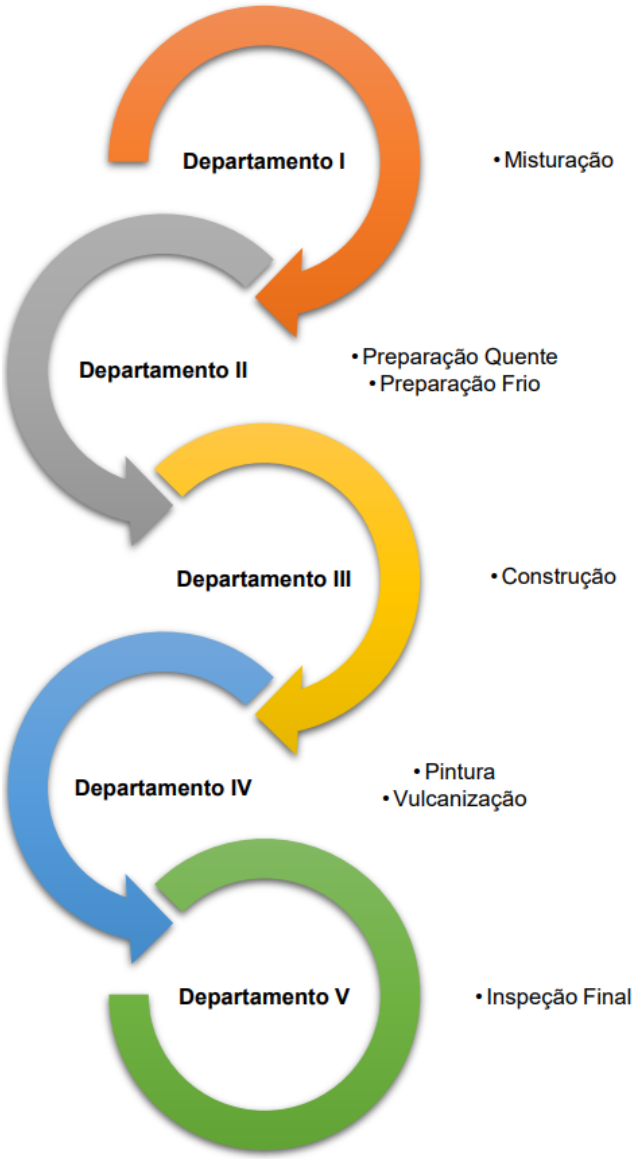


Figura 2.21: Departamentos Continental Mabor.

Na Figura 2.22 são ilustradas de forma sucinta as fases de produção de pneus da Continental Mabor bem como cada um dos seus departamentos, onde é possível perceber a admissão dos diferentes componentes, a que áreas se destinam, assim como o seu percurso por todos os departamentos.

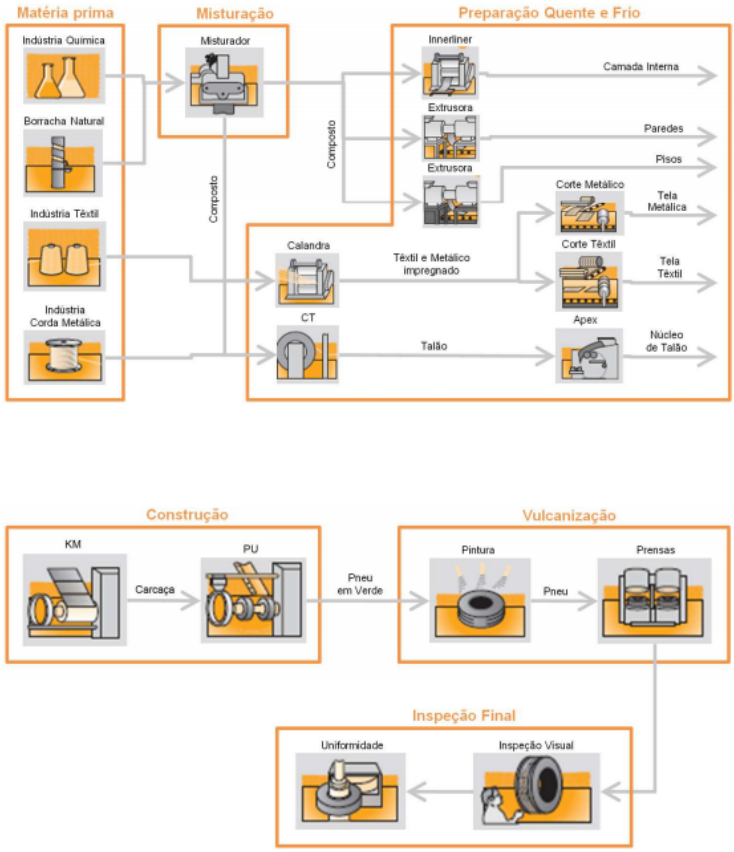


Figura 2.22: Diversas fases da produção de pneus da Continental Mabor com as respetivas matérias-primas e máquinas.

2.2.1 Receção das Matérias-Primas

Para a concretização do processo produtivo são necessárias várias matérias-primas, nomeadamente o negro de fumo, sílicas, óleos minerais, borracha natural e sintética, pigmentos, arame, tecidos têxteis e corda metálica. Para um bom funcionamento do processo, é necessária uma boa receção das diferentes matérias-primas. Para isso, todos os materiais são submetidos a processos de controlo de qualidade, com vista a garantir que se encontram conforme os requisitos e as especificações necessários. Após a aprovação destes, os materiais seguem para cada um dos respetivos processos produtivos, ou para armazenamento.

2.2.2 Departamento I – Misturação

É neste departamento que se dá o início do processo produtivo. O primeiro passo inicia-se com a pesagem e medição das matérias-primas especificadas, de acordo com as propriedades pretendidas para o composto em produção.

O departamento de misturação consiste na misturação de todos os componentes incorporados na borracha (borrachas naturais e sintéticas, pigmentos, sílicas e o negro de fumo). Estas componentes são então colocadas nos misturadores (Figuras 2.23 e 2.24) onde são transformadas e moldadas na borracha, criando assim um novo componente que mais tarde será utilizada por um novo processo. Após a misturação, as borrachas são colocadas em mesas para posteriormente serem transportadas por empilhadores para o próximo departamento, o departamento II.

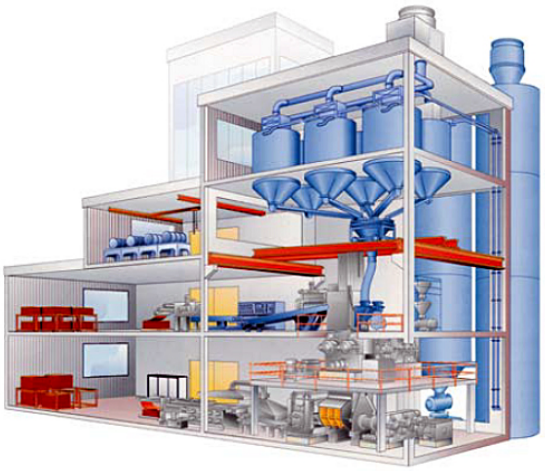


Figura 2.23: Ilustração de uma máquina para misturação de borracha [6].

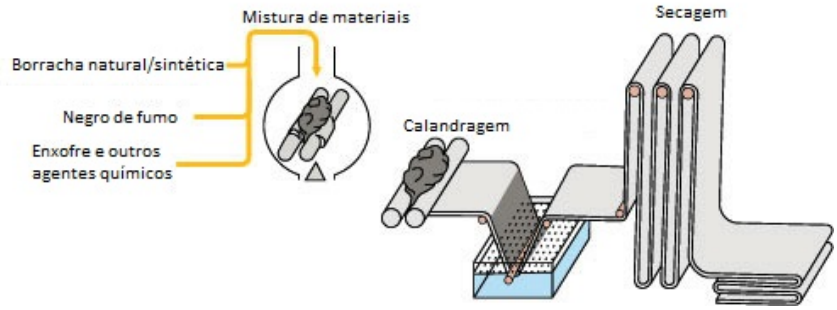


Figura 2.24: Misturador.

2.2.3 Departamento II – Preparação

O departamento II, recebe os componentes produzidos e preparados pelo departamento I, e produz nas suas duas divisões (Preparação Quente e Frio) componentes como por exemplo os pisos e paredes laterais, telas têxteis e metálicas (Figura 2.25), e as cunhas e talões. Ou seja, é nesta divisão que são produzidos todos os componentes, para que na próxima divisão seja possível montar o pneu (pneu em verde).

Analogamente, podemos verificar que, o Departamento II é então o fornecedor do Departamento III, enquanto é o cliente do Departamento I.

Na preparação quente são produzidos os pisos (Figura 2.26), paredes, talões e as cunhas, enquanto que na preparação frio são produzidas as camadas, as telas têxteis e metálicas, as cintas/reforços têxteis e metálicos e a espiral têxtil.

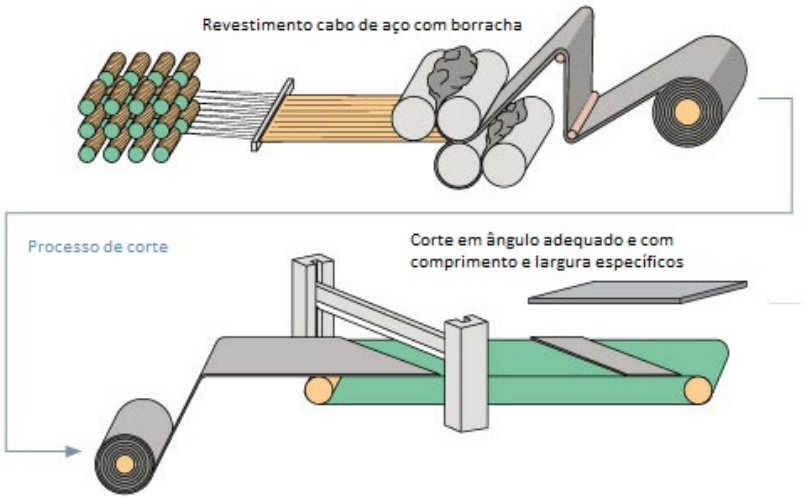


Figura 2.25: Calandra/Corte metálico.

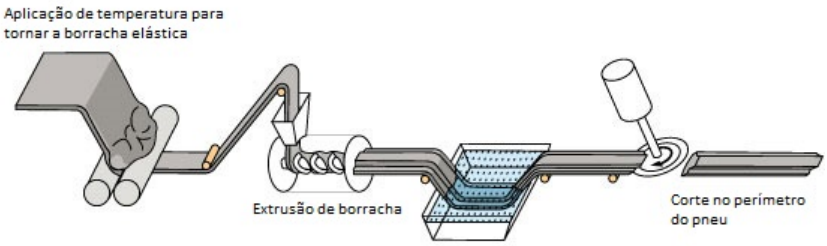


Figura 2.26: Extrusora de pisos.

2.2.4 Departamento III – Construção

O Departamento de Construção é responsável pela montagem dos diferentes componentes provenientes do Departamento de Preparação. Este Departamento é constituído por módulos de construção, onde cada módulo é composto por duas máquinas, as *Karcass Machine* (KM) (Figura 2.27) e as *Pressure Unit* (PU) (Figura 2.28).

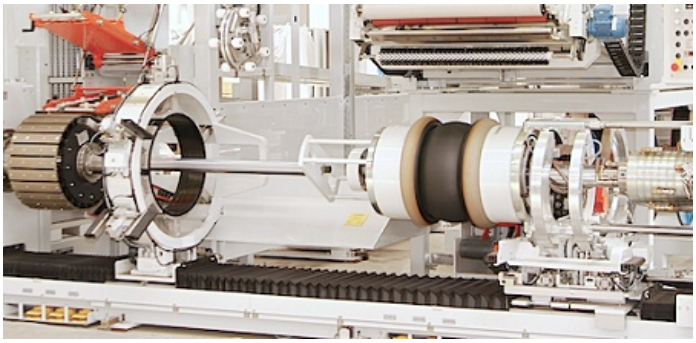


Figura 2.27: Unidade construtora de carcaças (KM) [7].

A KM é a máquina responsável pela construção da carcaça. Numa primeira operação, os talões, previamente confecionados, devem ser corretamente posicionados sobre os aplicadores, de modo a não criar desequilíbrios. O forro interior denominado por camada (ou *innerliner*), é colocado sobre o tambor de construção, o qual rodará uma volta completa. A extremidade da tela calandrada é colocada sobre o tambor de construção o qual deve efetuar uma rotação completa com o auxílio de um rolo pressor. Os talões são então instalados automaticamente, com o auxílio dos aplicadores (anéis). As extremidades da(s) tela(s) de corpo são então viradas para o interior da carcaça, com o auxílio de um tipo de *bladder* (ou diafragma) de borracha que a máquina de construção dispõe. De seguida, as paredes são aplicadas de cada um dos lados do tambor de construção, sobre a carcaça em construção. O tambor efetuará mais uma rotação completa e então as extremidades serão unidas e calcadas com o rolo pressor.

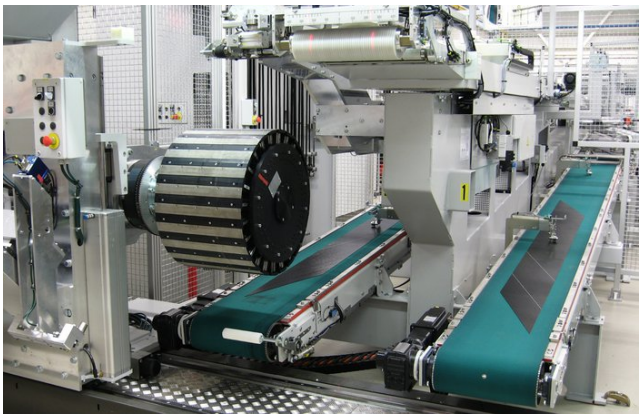


Figura 2.28: Unidade de pressão (PU) [7].

A PU recebe a carcaça da sua respetiva KM (conceito de módulo) e a ela junta as telas metálicas, a espiral têxtil e por fim os pisos. O produto final da PU, neste caso do módulo, é um pneu em verde, pronto a ser transportado por passadeiras aéreas que percorrem as áreas do departamento II, III e IV, para o próximo departamento.

2.2.5 Departamento IV – Pintura e Vulcanização

Tal como o nome do departamento indica, este departamento é composto por uma zona para a pintura interna do pneu, e pela zona final da vulcanização. Finalizado o processo de construção nos módulos do departamento III, os pneus em verde são então transportados por passadeiras até às pinturas do departamento IV. É nestas máquinas onde os pneus em verde são lubrificados internamente para permitir após vulcanização uma desmoldagem mais eficiente, bem como, aumentar o tempo de vida dos diafragmas das máquinas de vulcanização. Após a passagem pelas cabines de pintura, os pneus são carregados em carros de forma automática através de manipuladores denominados por GTAL. Uma vez dispostos em carros, estes são transportados até as máquinas para a vulcanização. Os pneus quando colocados nas prensas para a vulcanização (Figura 2.29) são submetidos a diferentes ciclos de tempo, onde as temperaturas que possibilitam a vulcanização e as pressões internas variam dependendo das diferentes medidas. Os moldes onde os pneus em verde são colocados dispõem das fissuras que proporcionam o aspeto final ao pneu depois deste ter passado pelo processo de vulcanização.

Figura 2.29: Vulcanização (Prensa) [8].

2.2.6 Departamento V – Inspeção Final

Os pneus depois de vulcanizados pelo departamento IV seguem para o departamento V através de passadeiras automáticas. É no departamento V, a inspeção final (Figura 2.30), que é dada a última aprovação de qualidade do pneu.

Figura 2.30: Inspeção final.

Esta aprovação revê todos os requisitos de qualidade, segurança e desempenho dos pneus antes de serem enviados para o cliente. Para isso os pneus são submetidos a várias verificações visuais por operadores qualificados, inspeções automáticas, testes de rolamento para aferir problemas de paralelismo e concentricidade.

Depois de passadas e aprovadas todas as inspeções, os pneus passam por um processo de paletização (Figura 2.31) e são depois transportados para o armazém de produto acabado através de um novo transportador ou através de camião, para que possam ser armazenados e preparados para expedição.

Figura 2.31: Paletização de pneus [9].

2.2.7 Armazém de Produto Acabado

Terminado o processo produtivo, os pneus são armazenados, consoante as suas características no armazém de produto acabado (Figura 2.32), onde são guardados até que sejam expedidos para os clientes finais.

Figura 2.32: Armazém de produto acabado [10].

Capítulo 3

Revisão Bibliográfica

A revisão bibliográfica para o desenvolvimento desta dissertação assenta em quatro subcapítulos, nomeadamente, controlo de qualidade, visão computacional, manipulação, exploração e processamento de dados e, ainda, o subcapítulo mais extenso sobre *Machine Learning* e *Deep Learning*.

3.1 Controlo de Qualidade

O Controlo de Qualidade (CQ) ou *Quality Control* (QC) é um processo através do qual uma empresa procura garantir que a qualidade do produto seja mantida ou melhorada com erros reduzidos ao máximo ou até mesmo levados a zero. O CQ exige que a empresa crie um ambiente no qual tanto a gestão como os funcionários procurem a perfeição. Isso é feito formando os colaborados, criando *benchmarks* para a qualidade do produto e testando os produtos para verificar variações estatisticamente significativas. O CQ é a tarefa que garante que os produtos atinjam um determinado padrão, definido pela empresa ou pelos clientes (Figura 3.1). Para os consumidores, a qualidade de um produto é muitas vezes considerada a "aptidão para uso" do produto [11].

Figura 3.1: Exemplo de Inspeção de Controlo de Qualidade [12].

Várias técnicas de CQ foram desenvolvidas desde a década de 1930, conforme apresentado na Figura 3.2, algumas delas acabaram por se difundir, como o Controlo Estatístico de Qualidade (CEQ), baseado na aplicação de métodos estatísticos, especificamente cartas de controlo e amostragem de aceitação [13].

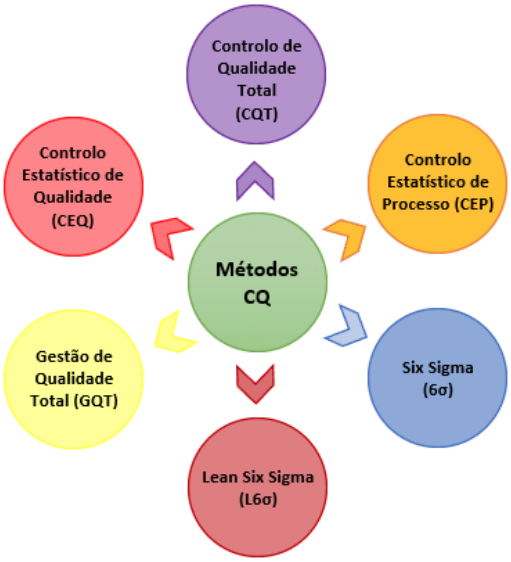


Figura 3.2: Métodos de Controlo de Qualidade.

Outro exemplo é o Controlo Estatístico de Processo (CEP), que consiste em utilizar gráficos de controlo para monitorar um processo industrial individual e retroalimentar o desempenho dos operadores responsáveis por aquele processo, e a Gestão de Qualidade Total (GQT), que utiliza em parte as técnicas de (CEQ) para impulsionar a melhoria contínua [14]. Mais recentemente, o método 6 Sigma, proveniente da Motorola, consiste em CEQ aplicado à estratégia de negócios, é uma metodologia orientada a dados para eliminar defeitos que levam a seis desvios padrão entre a média e o limite de especificação mais próximo em qualquer processo.

Atualmente as indústrias estão utilizando múltiplas Tecnologias de Informação e Comunicação (TIC) para realizar e automatizar diversas tarefas de produção, como programação e planeamento, controlo de processos, rastreamento e CQ [15].

A área de CQ desenvolveu-se rapidamente durante a segunda metade do século XX e hoje é parte integrante da maioria das empresas, sendo atualmente realizada não apenas na fase final com técnicas convencionais, como cartas de controlo e amostragem de aceitação, mas também ao longo do processo de produção [16], permitindo detetar antecipadamente desvios de qualidade, resultando em padrões de qualidade mais elevados e menor custo de produção. Com as atuais grandes quantidades de dados disponíveis no chão de fábrica, há um potencial inexplorado na indústria de fabrico para desenvolver tarefas de CQ mais automatizadas e inteligentes, aplicando técnicas de Inteligência Artificial (IA) e particularmente *Machine Learning* (ML)/*Deep Learning* (DL).

O uso de algoritmos de ML em processos de inspeção permite identificar defeitos ou desvios, bem como realizar diagnósticos de possíveis problemas. Houve algumas implementações práticas na indústria com alguns estudos bem sucedidos publicados sobre a qualidade da fruta detetada com algoritmos de ML [17][18].

Outros exemplos de aplicação real de soluções de ML na indústria alimentar, que é uma das indústrias mais dependentes da gestão de qualidade, onde a falta de qualidade de pelo menos um ingrediente pode ter um impacto diretamente na qualidade do produto final [19], é a Kewpie Corporation (uma grande empresa de alimentos japonesa), que usa as bibliotecas de aprendizagem de máquina TensorFlow do Google em seu sistema de inspeção visual, para detetar automaticamente anomalias em suas batatas cortadas em cubos [20].

A Fujitsu desenvolveu também uma solução para identificar potenciais defeitos no processo de fabrico através da realização de inspeções *Non-Destructive Testing* (NDT) que são combinadas com técnicas de processamento de imagem e *Deep Learning* para realizar um diagnóstico em minutos [21].

A Siemens também usa soluções de IA semelhantes para identificar falhas em lâminas de vidro de até 75 metros, percorrendo cada centímetro para identificar qualquer tipo de falha que possa causar defeitos de produção. O processo de inspeção leva agora 1 hora e meia, quando antes levava 6 horas para ser concluído [22].

Esses exemplos mostram como o processamento de imagens e a IA podem ser aplicados de maneira útil para melhorar o processo de CQ. Com a vinda da indústria 4.0 e a disponibilidade de grandes quantidades de dados (*Big Data*) permitiu o uso de IA para apoiar o controlo de qualidade na indústria.

3.2 Visão Computacional

O conceito principal de qualquer sistema de Inteligência Artificial (IA) é que ele pode perceber o ambiente e realizar ações com base na sua compreensão. A Visão Computacional, ou *Computer Vision* (CV), foca-se na parte da compreensão visual. É a ciência que procura perceber e entender o mundo por meio de imagens e vídeos, construindo um modelo físico do mundo para que um sistema de IA possa tomar as ações apropriadas. Para os humanos, a visão é apenas um aspeto da compreensão. Percebemos o mundo através da nossa visão, mas também através do som, do olfato e dos nossos outros sentidos. O mesmo acontece com os sistemas de IA, a visão é apenas uma maneira de entender o mundo. Dependendo da aplicação que estamos a construir, nós selecionamos o dispositivo (sensor) que melhor captura o mundo [23].

Visão computacional é um campo multidisciplinar que pode ser chamado de subcampo da inteligência artificial (Figura 3.3).

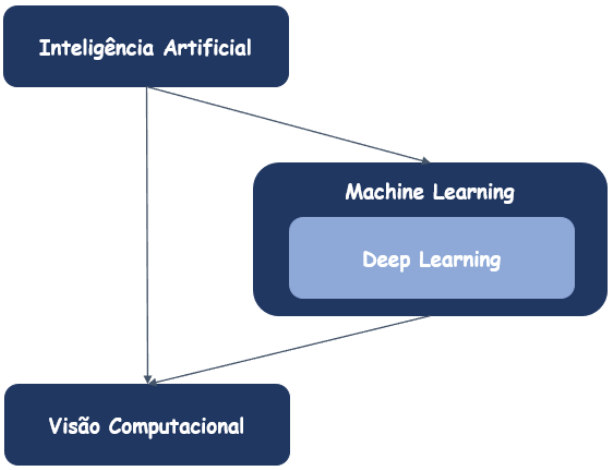


Figura 3.3: Relação entre Inteligência Artificial e Visão Computacional.

A compreensão visual, de uma forma mais simples, é o ato de observar padrões e objetos através da visão. Como um veículo autónomo, por exemplo, a compreensão visual significa entender os objetos ao redor e os seus detalhes específicos, como pedestres, ou se há uma pista específica na qual o veículo precisa estar centralizado, detetar sinais de trânsito e entender o que eles significam. É por isso que a palavra compreensão faz parte da definição. Não estamos apenas procurando capturar o ambiente ao redor. Estamos a tentar construir sistemas que possam realmente entender esse ambiente por meio de uma entrada visual [23].

Nas últimas décadas, as técnicas tradicionais de processamento de imagens eram consideradas sistemas CV, mas isso não é totalmente preciso. Uma máquina

que processa uma imagem é completamente diferente de uma máquina que entende o que está acontecendo dentro da imagem, o que não é uma tarefa comum. O processamento de imagens agora é apenas uma parte de um sistema maior e mais complexo, que procura interpretar o conteúdo da imagem. Os sistemas de visão são praticamente os mesmos para os humanos, animais, insetos e a maioria dos organismos vivos. Eles consistem num sensor ou olho que é usado para capturar a imagem e um cérebro para processar e interpretar a imagem. O sistema então gera uma previsão dos componentes da imagem com base nos dados extraídos da imagem (Figura 3.4).

Figura 3.4: Sistema de visão humano [23].

Para perceber como funciona o sistema de visão humano, suponhamos que queremos interpretar a imagem de cães na Figura 3.4. Nós olhamos e entendemos rapidamente que a imagem consiste numa matilha de cães (três, para ser preciso). É bastante natural para nós classificar e detetar objetos nesta imagem porque fomos treinados ao longo dos anos para identificar cães. Caso nos fosse apresentada uma imagem de um objeto em que estávamos a ver pela primeira vez, definitivamente não saberíamos identificar até que nos dissessem. Ao fim de alguns experimentos como esse, nós teríamos sido treinados para identificar esse novo objeto. Nós podemos treinar o nosso cérebro para identificar quase tudo. O mesmo acontece com os computadores. Nós podemos treinar máquinas para aprender a identificar objetos, mas os humanos são muito mais intuitivos do que as máquinas. São necessárias apenas algumas imagens para nós humanos, aprendermos a identificar a maioria dos objetos, enquanto que as máquinas são necessárias milhares ou, em casos mais complexos, milhões de amostras de imagens para aprender a identificar objetos.

Ao longo dos anos a comunidade científica tem-se inspirado na capacidade visual humana para que as máquinas a possam imitar e para isso são necessários os mesmos dois componentes principais, um dispositivo sensor para imitar a função do olho e um poderoso algoritmo para imitar a função cerebral na interpretação e classificação do conteúdo da imagem.

A interpretação é o cérebro do sistema de visão e o seu papel é captar a imagem de saída do dispositivo sensor e aprender recursos e padrões para identificar objetos. Cientistas se inspiraram como os nossos cérebros funcionam e tentaram fazer a engenharia inversa do sistema nervoso central para obter algumas ideias sobre como construir um cérebro artificial. Dessa forma, nascem as Redes Neuronais Artificiais [23] (Figura 3.5).

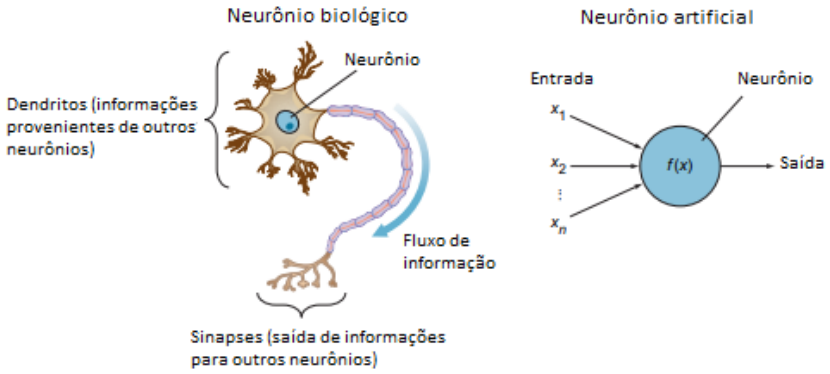


Figura 3.5: Semelhanças entre neurónio biológico e artificial [23].

Na Figura 3.5, podemos ver uma analogia entre neurónio biológico e neurónio artificial. Ambos contêm um elemento de processamento principal, um neurónio, com sinais de entrada ($x1, x2, \ldots, xn$) e uma saída. O comportamento de aprendizagem dos neurónios biológicos inspirou os cientistas a criar uma rede de neurónios que estão conectados uns aos outros. Imitando como a informação é processada no cérebro humano, cada neurónio artificial dispara um sinal para todos os neurónios aos quais está conectado quando uma quantidade suficiente dos seus sinais de entrada é ativa. Assim, os neurónios têm um mecanismo muito simples no nível individual, mas quando temos milhões desses neurónios empilhados em camadas e conectados entre si, cada neurónio é conectado a milhares de outros neurónios, produzindo um comportamento de aprendizagem. A construção de uma rede neuronal de múltiplas camadas é chamada de *Deep Learning* (Figura 3.6).

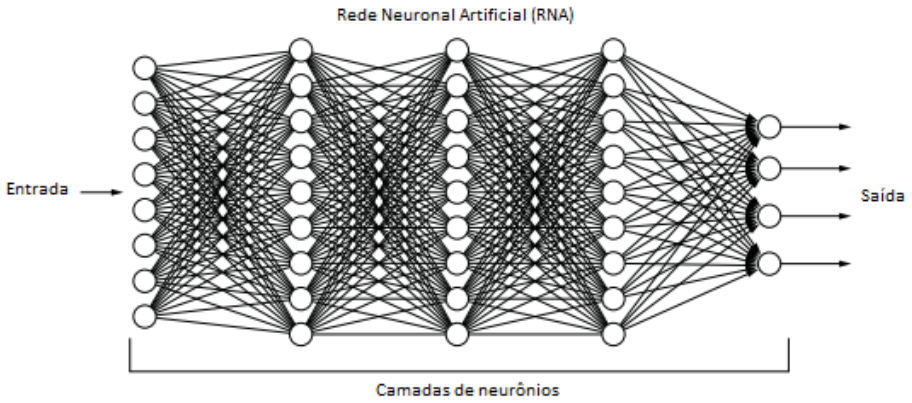


Figura 3.6: Camadas neuronais *Deep Learning* [23].

3.3 Manipulação, Exploração e Processamento de Dados

A implementação e uso de metodologias de *Machine Learning* e de *Deep Learning* requerem um conjunto de competências básicas de manipulação e processamento de dados, álgebra, matemática e estatística. Deste modo, este subcapítulo tem como objetivo a apresentação de tópicos importantes para o desenvolvimento destas metodologias, atuais desenvolvimentos e diferentes arquiteturas de metodologias com base em *machine learning*, reproduzindo desta forma o atual estado da arte.

3.3.1 Manipulação e Exploração de Dados

O armazenamento e manipulação de dados é umas das primeiras etapas na implementação e teste de metodologias de *Machine Learning* (ML) e *Deep Learning* (DL). Existem duas etapas que se destacam na exploração dos dados: i) aquisição; e ii) processamento, depois de adquiridos e armazenados. Quanto à resolução de problemas de aquisição, extração e exploração de dados existe um procedimento geral que envolve cinco passos [24][25]:

1. **Identificar o problema e formular as hipóteses**
   A maioria dos estudos de modelação baseados em dados são realizados num domínio de aplicação. Deste modo, o conhecimento e a experiência de domínio são geralmente necessários para chegar a uma declaração de problema significativa. Nesta etapa, um modelador normalmente especifica um conjunto de variáveis para a dependência desconhecida e, se possível, uma forma geral dessa dependência como hipótese inicial. Pode haver várias hipóteses formuladas para um único problema nesta fase. A primeira etapa requer a experiência combinada de um domínio de aplicação e um modelo de exploração de dados. Na prática, geralmente significa uma interação próxima entre

o especialista em exploração de dados e o especialista em desenvolvimento de aplicações [24].

2. **Recolha de dados**
   Esta etapa foca-se na forma como os dados são gerados e reunidos. Em geral, existem duas possibilidades distintas. A primeira é quando o processo de obtenção de dados está sob o controlo de um especialista. A segunda possibilidade é quando o especialista não pode influenciar os dados (processo de obtenção), isso é conhecido como a abordagem observacional. Uma configuração observacional, ou seja, produção de dados aleatória, é assumida na maioria das aplicações de exploração de dados. Normalmente, a distribuição da amostragem é completamente desconhecida após a recolha de dados, ou é fornecida parcial e implicitamente no procedimento de recolha de dados. É muito importante, no entanto, entender como a recolha de dados afeta a sua distribuição teórica, pois tal conhecimento pode ser muito útil para modelação e, posteriormente, para a interpretação final dos resultados. Além disso, é importante garantir de que os dados usados para estimar um modelo e os dados usados posteriormente para testar e aplicar um modelo vêm da mesma fonte de dados. Se este não for o caso, o modelo estimado não poderá ser utilizado com sucesso na aplicação final dos resultados [24][26].

3. **Pré-processamento de dados**
   No cenário observacional, os dados geralmente são recolhidos de bancos de dados. O pré-processamento de dados geralmente inclui pelo menos duas tarefas comuns [24][27]:

   (a) **Deteção de dados anómalos (e remoção)**
   Dados anómalos ou *outliers* são valores de dados incomuns que não são consistentes com a maioria das observações. Os *outliers* resultam de erros de medição, erros de codificação e recolha e, às vezes, são valores naturais anormais. Tais amostras não representativas podem afetar seriamente o modelo produzido posteriormente. Existem duas estratégias para lidar com *outliers*:

      i. Detetar e, eventualmente, remover valores anómalos como parte da fase de pré-processamento.
      ii. Desenvolvimento de métodos de modelação robustos que sejam insensíveis a *outliers*.

   (b) **Dimensionamento, codificação e seleção de recursos**
   O pré-processamento de dados inclui várias etapas, como dimensionamento de variáveis e diferentes tipos de codificação. Por exemplo, uma característica com valores entre $[0, 1]$ e outra com valores entre $[-100, 1000]$

não terão o mesmo peso, eles também influenciarão os resultados finais da exploração de dados de forma diferente. Desse modo, é recomendável dimensionar e deslocar os dois recursos para o mesmo peso para análise posterior. Além disso, os métodos de codificação específicos da aplicação geralmente atingem a redução de dimensionalidade fornecendo um número menor de recursos informativos para modelação de dados subsequente. Essas duas classes de tarefas de pré-processamento são apenas exemplos ilustrativos de um amplo espectro de atividades de pré-processamento num processo de exploração de dados. As etapas de pré-processamento de dados não devem ser consideradas completamente independentes de outras fases de exploração de dados. Em cada iteração do processo de exploração de dados, todas as atividades, juntas, podem definir conjuntos de dados novos e aprimorados para iterações subsequentes. Geralmente, um bom método de pré-processamento fornece uma representação ideal para uma técnica de exploração de dados, incorporando conhecimento na forma de dimensionamento e codificação específicos da aplicação.

4. **Avaliar o modelo**
   A seleção e implementação da técnica de exploração de dados apropriada é a principal tarefa nesta fase. Geralmente, na prática, a implementação é baseada em vários modelos, e selecionar o melhor é uma tarefa adicional [24].

5. **Interpretar o modelo e tirar conclusões**
   Na maioria dos casos, os modelos de exploração de dados devem ajudar na tomada de decisões. Logo, esses modelos precisam ser interpretáveis para serem úteis. É de notar que os objetivos de precisão do modelo e precisão da sua interpretação são um tanto contraditórios. Normalmente, os modelos simples são mais interpretáveis, mas também são menos precisos. Espera-se que os métodos modernos de exploração de dados produzam resultados altamente precisos usando modelos de alta dimensão. O problema de interpretar esses modelos (também muito importante) é considerado uma tarefa à parte, com técnicas específicas para validar os resultados. Não importa quão poderoso seja o método de exploração de dados usado na etapa 4, o modelo resultante não será válido se os dados não forem recolhidos e pré-processados corretamente ou se a formulação do problema não for bem feita.
   Na prática, os dois principais objetivos da exploração de dados são a previsão e a descrição. A previsão envolve o uso de algumas variáveis ou campos no conjunto de dados para prever valores desconhecidos ou futuros, de outras variáveis de interesse. A descrição, por outro lado, concentra-se em encontrar padrões que descrevam os dados que podem ser interpretados por humanos.

Dessa forma, é possível colocar as atividades de exploração de dados numa das duas categorias [24]:

(a) preditiva, que produz o modelo do sistema descrito pelo conjunto de dados fornecido;

(b) descritiva, que produz informações novas com base no conjunto de dados disponível.

Na extremidade preditiva do espectro, o objetivo da exploração de dados é produzir um modelo, expresso como um código executável, que pode ser usado para realizar classificação, previsão, estimativa ou outras tarefas semelhantes. Na extremidade descritiva do espectro, o objetivo é obter uma compreensão do sistema analisado, descobrindo padrões e relações em grandes conjuntos de dados. A importância da previsão e descrição para aplicações de exploração de dados específicos pode variar consideravelmente. Os objetivos de previsão e descrição são alcançados usando técnicas de exploração de dados, para as seguintes tarefas primárias [24]:

(a) **Classificação**. Identificação de uma função de aprendizagem preditiva que classifica um elemento de dados numa das várias classes predefinidas.

(b) **Regressão**. Identificação de uma função de aprendizagem preditiva que mapeia um elemento de dados para uma variável de predição de valor real.

(c) **Agrupamento**. Uma tarefa descritiva comum na qual se procura identificar um conjunto finito de categorias ou agrupamentos para descrever os dados.

(d) **Sumarização**. Uma tarefa descritiva adicional que envolve métodos para encontrar uma descrição compacta para um conjunto (ou subconjunto) de dados.

(e) **Modelação de Dependência**. Encontrar um modelo local que descreva dependências significativas entre variáveis ou entre os valores de um recurso num conjunto de dados ou numa parte de um conjunto de dados.

(f) **Deteção de Mudanças e Desvios**. Descobrir as mudanças mais significativas no conjunto de dados.

3.3.2 Processamento de Imagem

3.3.2.1 Imagem de entrada

Uma imagem pode ser representada em função de duas variáveis x e y, que definem uma área bidimensional. Uma imagem digital é feita de uma matriz de píxeis e o píxel (do inglês, *pixel - picture element*) é o bloco de construção de uma imagem. Toda a imagem consiste num conjunto de píxeis em que os seus valores representam a intensidade da luz que aparece num determinado local da imagem [23].

Em imagens coloridas, em vez de representar o valor do píxel por apenas um número, o valor é representado por três números que representam a intensidade de cada cor no píxel. Num sistema *Red, Green and Blue* (RGB), por exemplo, o valor do píxel é representado por três números, a intensidade do vermelho, a intensidade do verde e a intensidade do azul. Existem outros sistemas de cores para imagens tais como *Hue, Saturation and Value* (HSV) e LAB ('L' significa a intensidade de brilho, enquanto que 'A' (cor vermelha/magenta) e 'B' (azul/amarelo)). Todos seguem o mesmo conceito ao representar o valor do píxel.

Nas imagens em tons de cinza, cada píxel representa a intensidade de apenas uma cor, enquanto no sistema RGB padrão, as imagens coloridas possuem três canais. Por outras palavras, as imagens coloridas são representadas por três matrizes, uma representa a intensidade do vermelho no píxel, outra o verde e por último o azul, como é apresentado na Figura 3.7.

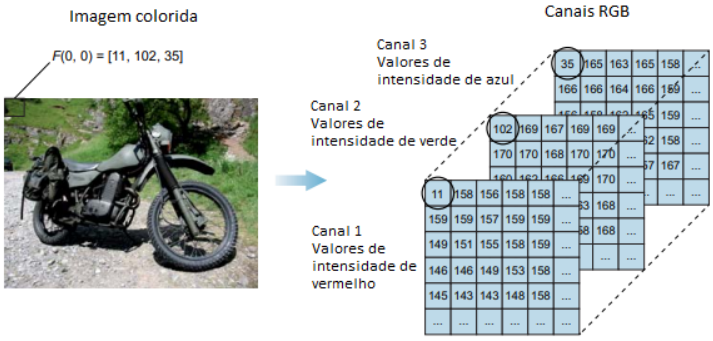


Figura 3.7: Representação de imagens coloridas por canais RGB [23].

3.3.2.2 Como os computadores vêm as imagens

Quando olhamos para uma imagem, vemos objetos, paisagens, cores e assim por diante. O mesmo não acontece com os computadores, tal como é apresentado na Figura 3.8. O cérebro humano pode processar e identificar de imediato que é a imagem de uma mota. Para um computador, a imagem representa uma matriz

bidimensional (2D) dos valores de píxeis, que representam intensidades em todo o espectro de cores.

Figura 3.8: Representação de como um computador interpreta uma imagem [23].

A imagem da Figura 3.8 tem o tamanho 20×20. Este tamanho indica a largura e a altura da imagem, sendo 20 píxeis na horizontal e 20 na vertical. Isso significa que há um total de 400 (20×20) píxeis. Se a imagem fosse 800×600, então a dimensão da matriz era (800, 600), onde cada píxel da matriz representa a intensidade de brilho naquele píxel. O número zero representa a cor preta e 255 representa a cor branca.

3.3.2.3 Pré-processamento de imagem

No desenvolvimento de um projeto de *Machine Learning/Computer Vision*, precisamos sempre de dados. Neste caso, dados de imagem. Infelizmente, existem alguns problemas associados aos dados de imagem, tais como ruído (complexidade, imprecisão e inadequação). Por esse motivo o primeiro passo no processamento de dados de imagem é o pré-processamento, ou seja, melhorar a qualidade da imagem a ser utilizada, realçar as características pertinentes e remover as informações indesejadas como os ruídos mencionados. Antes de construirmos um modelo de visão computacional, é essencial que os dados sejam pré-processados (limpos e processados no formato desejado) para alcançar os resultados desejados. Podem ocorrer diferentes imprecisões na classificação, se o processamento não for devidamente prudente. As imagens digitais são sujeitas a vários tipos de ruído. O ruído é o resultado de erros no processo de aquisição da imagem que resultam em valores de píxel que não refletem as verdadeiras intensidades do panorama real. Existem várias maneiras pelas quais o ruído pode ser introduzido numa imagem, dependendo de como a imagem é criada. Por exemplo [28]:

- Se a imagem for digitalizada a partir de uma fotografia feita num filme, o grão do filme é uma fonte de ruído.
- Se a imagem for adquirida diretamente em formato digital, o dispositivo de captura de dados pode introduzir ruído.

- A transmissão eletrónica de dados de imagem pode introduzir ruído.

Existem diferentes filtros que podem ser aplicados para a remoção de dados indesejados, como por exemplo a filtragem linear que pode ser usado para remover diversos tipos de ruído. Tais filtros, como filtros de média, mediano, adaptativo e *gaussian* são apropriados para remover ruído *gaussiano* e *poisson*. Por exemplo, um filtro de média é útil para remover o ruído de grão de uma fotografia. Como cada píxel é definido para a média dos píxeis na sua vizinhança, as variações locais causadas pela granulação são reduzidas. Com o filtro mediano, o valor de um píxel de saída é determinado pelo mediano dos píxeis vizinhos, em vez da média. O mediano é muito menos sensível que a média a valores extremos (chamados de *outliers*). A filtragem mediana tem uma maior capacidade de remoção desses valores discrepantes sem reduzir a nitidez da imagem. A combinação certa das tarefas de pré-processamento proporciona uma maior precisão nas classificações obtidas com as imagens usadas [28].

3.4 *Machine* e *Deep Learning*

Aprendizagem de máquina ou aprendizagem automática, *Machine Learning* (ML), é um subcampo de inteligência artificial (IA) e da ciência da computação que se concentra no uso de dados e algoritmos para imitar a maneira como os humanos aprendem, melhorando gradualmente a sua precisão. Os desenvolvimentos tecnológicos em torno de armazenamento e poder de processamento permitirão alguns produtos inovadores que conhecemos hoje, como o mecanismo de recomendação da Netflix ou carros autónomos [29][30]. O ML é um componente importante do crescente campo da ciência de dados. Por meio do uso de métodos estatísticos, os algoritmos são treinados para fazer classificações ou previsões, revelando intuições importantes em projetos de exploração de dados. Essas intuições posteriormente impulsionam a tomada de decisões em aplicações e negócios, impactando de forma ideal as principais métricas de crescimento.

Como *Deep Learning* (DL) e ML tendem a ser usados de forma intercalar, importa referir as diferenças entre os dois. *Machine Learning*, *Deep Learning* e *Neural Networks* (NN), são todos subcampos da inteligência artificial. No entanto, o DL é na verdade um subcampo de ML, e as NN são um subcampo de DL. A maneira pela qual o DL e o ML se diferem, está em como cada algoritmo aprende. O DL automatiza grande parte da extração de recursos de um processo, eliminando parte da intervenção manual humana necessária, permitindo o uso de conjuntos de dados (*Datasets*) maiores. Nós podemos pensar em DL como "aprendizagem de máquina escalável". O ML clássico, ou "não profundo", depende mais da intervenção humana para aprender. Especialistas humanos identificam o conjunto de recursos

para entender as diferenças entre as entradas de dados, geralmente exigindo dados mais estruturados para aprender.

DL pode aprender com conjuntos de dados anotados com etiquetas, também conhecido como aprendizagem supervisionada, para ensinar o seu algoritmo, mas não requer necessariamente de um conjunto de dados identificado. Ele pode aceitar dados não estruturados em sua forma bruta (por exemplo, texto, imagens) e pode determinar automaticamente o conjunto de recursos que distinguem diferentes categorias de dados entre si. Ao contrário de ML, ele não requer intervenção humana para processar dados. O DL e as NN são reconhecidos principalmente por acelerar o progresso em áreas como visão computacional, processamento de linguagem natural e reconhecimento de fala.

As redes neuronais artificiais, ou *Artificial Neural Networks* (ANN), são compostas por camadas de nós, contendo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada nó, ou neurónio artificial, conecta-se a outro e tem um peso e um limiar associados. Se a saída de qualquer nó individual estiver acima do valor limite especificado, esse nó é ativo, enviando dados para a próxima camada da rede. Caso contrário, nenhum dado é passado para a próxima camada. O DL refere-se apenas à profundidade das camadas numa rede neuronal. Uma rede neuronal que consiste em mais de três camadas, que inclui as entradas e as saídas, pode ser considerado um algoritmo de DL ou uma rede neuronal profunda. Uma rede neuronal que possuí apenas duas ou três camadas é apenas uma rede neuronal básica [30][31].

Existem diferentes tipos de aprendizagem de máquina, normalmente categorizados de acordo com a forma como os modelos utilizam os seus dados de entrada para o processo de aprendizagem. Em aprendizagem supervisionada, o conjunto de dados que está a ser usado possuí etiquetas que caracterizam os dados, onde são classificadas por humanos para permitir que o algoritmo avalie a precisão de seu desempenho. Na aprendizagem não supervisionada, o conjunto de dados usado não é etiquetado, sendo que o próprio algoritmo identifica padrões nos dados sem a ajuda humana. Aprendizagem semi-supervisionada, o conjunto de dados contém dados estruturados e não estruturados, que orientam o algoritmo para tirar conclusões independentes. A combinação dos dois tipos de dados num *dataset* de treino permite que os algoritmos de aprendizagem de máquina aprendam a etiquetar dados não caracterizados. Em *Reinforcement Learning* (RL), o conjunto de dados usa um sistema de "recompensa/punição", oferecendo *feedback* ao algoritmo para aprender com as suas próprias experiências por tentativa e erro [30].

Por fim, há o conceito de DL, que é uma área mais recente de ML, que aprende automaticamente com o conjunto de dados sem introduzir regras ou conhecimento humano. Isso requer grandes quantidades de dados.

3.4.1 Medidas de Desempenho

Uma das medidas centrais de um processo de exploração de dados e aprendizagem de máquina é a avaliação do desempenho de um modelo. Com o nível de desempenho é possível identificar o quão bem sucedidas são as previsões de um conjunto de dados treinado pelo modelo. Existem quatro tipos principais de tarefas de classificação, sendo eles, classificação binária, classificação de múltiplas classes (multi-classe), classificação de várias etiquetas (*Multi-Label*) e classificação desequilibrada. Para cada um destes tipos de classificação existem medidas de desempenho e no caso desta dissertação iremos trabalhar com a classificação de múltiplas classes. Este desempenho pode ser avaliado por quatro valores diferentes como é apresentado na Figura 3.9 [32].

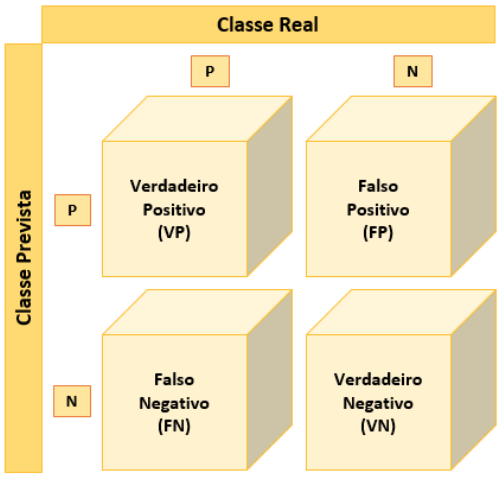


Figura 3.9: Matriz de Confusão.

A correção de uma classificação pode ser avaliada calculando o número de exemplos de classe corretamente identificados como Verdadeiro Positivo (VP), o número de exemplos corretamente identificados que não pertencem à classe Verdadeiro Negativo (VN) e exemplos que foram atribuídos incorretamente à classe Falso Positivo (FP) ou que então, não foram reconhecidos como exemplos da classe Falso Negativo (FN). Estas quatro contagens constituem uma matriz de confusão apresentada na Figura 3.9 [32].

Para obter um exemplo conveniente num problema do mundo real, consideramos um teste de diagnóstico que procura determinar se uma pessoa tem uma determinada doença. Um falso positivo neste caso ocorre quando a pessoa testa positivo, mas na verdade não tem a doença. Um falso negativo, por outro lado, ocorre quando a pessoa tem um teste negativo, indicando que está saudável quando realmente tem a doença.

3.4.1.1 Exatidão (ACC)

A exatidão, ou *Accuracy* (ACC), é a razão simples entre o número de pontos classificados corretamente para o número total de pontos [32]. Este cálculo pode ser realizado com recurso à seguinte equação:

$$ACC = \frac{VP + VN}{VP + FN + FP + VN} \tag{3.1}$$

3.4.1.2 Precisão

Precisão (*Precision*) é a divisão das instâncias classificadas corretamente sobre o total de instâncias classificadas como classes positivas. A precisão ajuda-nos a entender a utilidade dos nossos resultados [32].

$$Precision = \frac{VP}{VP + FP} \tag{3.2}$$

3.4.1.3 Sensibilidade (Sn)

Sensibilidade (Sn) também conhecida por (*Recall*) é a divisão das instâncias classificadas corretamente sobre o total de instâncias classificadas. *Recall* ajuda-nos a entender o quão completos são os nossos resultados [32].

$$Sn = \frac{VP}{VP + FN} \tag{3.3}$$

3.4.1.4 Especificidade (Sp)

Especificidade (Sp) é uma métrica que demonstra com que eficácia um classificador identifica etiquetas negativas [32]. A métrica da especificidade é expressa através da seguinte equação:

$$Sp = \frac{VN}{VP + FN} \tag{3.4}$$

3.4.1.5 Pontuação F1

A pontuação F1 (*F1 Score*) é uma média ponderada de Precisão e Sensibilidade, o que significa que há igual importância dada a FP e FN. Esta é uma métrica muito útil em comparação com a "Precisão". O problema com o uso de precisão é que, se tivermos um conjunto de dados altamente desequilibrado para treino (por exemplo, um conjunto de dados de treino com 95% de classes positivas e 5% de classes negativas), o modelo acabará por aprender como prever a classe positiva corretamente e não aprender a identificar a classe negativa. Mas o modelo ainda terá uma precisão muito alta no conjunto de dados de teste, isto porque, saberá identificar os

positivos muito bem [33].

$$F1Score = \frac{2VP}{2VP + FP + FN} \tag{3.5}$$

3.4.1.6 AUC-ROC

Area Under Curve (AUC) - *Receiver Operating Characteristic* (ROC) é uma métrica de desempenho, baseada em valores de entrada variáveis, para problemas de classificação. Como o nome indica, ROC é uma curva de probabilidade e AUC mede a separabilidade. Por outras palavras, a métrica AUC-ROC nos dirá sobre a capacidade do modelo em distinguir as classes. Quanto maior a AUC, melhor o modelo. Matematicamente, pode ser criado, traçando o *True Positive Rate* (TPR), ou seja, sensibilidade e *False Positive Rate* (FPR), sendo a, especificidade, em vários valores de limite [33]. Equação do AUC é apresentada de seguida:

$$AUC = \frac{1}{2}(\frac{VP}{VP + FN} + \frac{VN}{VN + FP}) \tag{3.6}$$

3.4.2 Funções de Cálculo de Perdas

Os problemas de classificação envolvem a previsão de uma saída de classe discreta. Envolve dividir o conjunto de dados em classes diferentes e exclusivas com base em parâmetros diferentes para que um novo dado (não usado), possa ser identificado numa das classes. Um e-mail pode ser classificado como *spam* ou não *spam*, por exemplo. De seguida são apresentadas algumas das funções de perdas usadas para classificação.

3.4.2.1 Perda de Entropia Cruzada Binária

A entropia cruzada binária, ou *Binary Cross-Entropy* (BCE), é definida como uma medida da diferença entre duas distribuições de probabilidade para um dado aleatório variável ou conjunto de eventos. É amplamente utilizado em classificação e, como a segmentação é uma classificação ao nível de píxel, também pode ser usado [34]. A entropia cruzada binária é expressa da seguinte forma [35]:

$$L_{BCE}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{3.7}$$

Onde, ŷ é o valor previsto pelo modelo.

3.4.2.2 Perda de Entropia Cruzada Categórica

A perda de entropia cruzada categórica, também conhecida como *softmax loss*, é essencialmente a perda de entropia cruzada binária expandida para várias classes, ou seja, é usada em tarefas de classificação de múltiplas classes. Essas são tarefas em que um exemplo só pode pertencer a uma das muitas categorias possíveis, e o modelo deve decidir a qual delas se trata. É projetada para quantificar a diferença entre duas distribuições de probabilidade. Esta propriedade é estendida para uma função de ativação chamada *softmax* [36].

3.4.2.3 Perda de Articulação

Outra função de perda tipicamente usada para classificação é a perda de articulação (*hinge loss*). A perda de articulação é um tipo específico de classificação de "margem máxima", principalmente para máquinas de suporte vetorial. Mesmo que novas observações sejam classificadas corretamente, elas podem incidir em penalização se a margem de limite de decisão não for grande o suficiente. A perda de articulação aumenta linearmente [36].

3.4.3 Algoritmos de *Machine Learning*

3.4.3.1 Árvores de decisão

Uma árvore tem muitas analogias na vida real e acaba influenciando uma ampla área de ML, abrangendo tanto a classificação como a regressão. Na análise de decisão, uma árvore de decisão ou *Decision Trees* (DT), pode ser usada para representar visualmente e explicitamente as decisões e a tomada de decisão. Como o nome sugere, ela usa um modelo de decisões em forma de árvore. Embora seja uma ferramenta tipicamente usada em exploração de dados para derivar uma estratégia para atingir um objetivo específico, também é amplamente usada em ML [37].

3.4.3.2 *Naïve Bayes*

O algoritmo *Naïve Bayesian* (NB) é um classificador probabilístico baseado no "Teorema de Bayes", o qual foi criado por Thomas Bayes (1702-1761) [38]. O algoritmo tornou-se popular na área de ML para classificar texto baseado na frequência das palavras usadas. Por ser muito simples e rápido, possui um desempenho relativamente maior do que outros classificadores. Além disso, o NB só precisa de um pequeno número de dados de teste para concluir classificações com uma boa

precisão. O teorema de Bayes pode ser analisado na seguinte equação [38][39]:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{3.8}$$

Usando o teorema, podemos encontrar a probabilidade de "A" acontecer, dado que "B" ocorreu. Aqui, "B" é a evidência e "A" é a hipótese. A suposição feita aqui, é que os recursos são independentes, ou seja, a presença de um determinado recurso não afeta o outro. Por isso é chamado de "ingénuo".

3.4.3.3 *Naïve Bayes Network*

Uma rede *Naïve Bayesian* é uma rede com uma única raiz, todos os outros nós são filhos da raiz e não há arestas entre os outros nós. A Figura 3.10 mostra uma rede *Naïve Bayesian*. Como é o caso de qualquer rede *Bayesian*, as arestas podem ou não representar influência causal. Muitas vezes, redes *Naïve Bayesian* são usadas para modelar problemas de classificação. Em tais problemas, os valores da raiz são classes possíveis às quais uma entidade pode pertencer, enquanto as folhas são características ou recursos das classes [40].

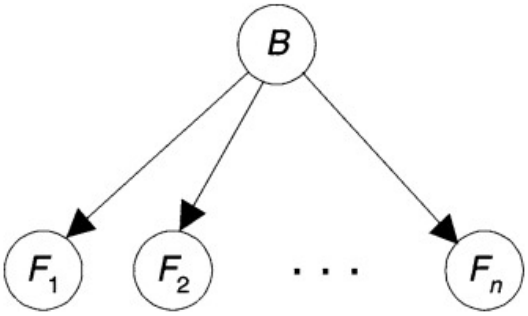


Figura 3.10: Rede *Naïve Bayesian* [40].

3.4.3.4 *K-Nearest Neighbor*

O algoritmo de *K-Nearest Neighbor* (KNN) é um algoritmo de ML supervisionado que pode ser usado para resolver problemas de classificação e regressão. O princípio por trás dos métodos de KNN é encontrar um número predefinido de amostras de treino mais próximas em distância, do novo ponto e prever a etiqueta a partir delas. O número de amostras pode ser uma constante definida, ou variar com base na densidade local de pontos. A distância pode, em geral, ser qualquer medida métrica: a distância euclidiana padrão é a escolha mais comum. Apesar da sua simplicidade, KNN foi bem sucedido num grande número de problemas de classificação e regressão, incluindo dígitos manuscritos e panoramas de imagens

de satélite. Sendo um método não paramétrico, muitas vezes é bem sucedido em situações de classificação onde o limite de decisão é muito irregular [41][42].

3.4.3.5 *Support Vector Machines*

As máquinas de suporte vetorial, ou *Support Vector Machine* (SVM), são um algoritmo de aprendizagem de máquina supervisionado que pode ser usado para resolver problemas de classificação ou regressão. No entanto, é mais usado em problemas de classificação. As SVM foram originalmente introduzidas por Vapnik *et al.* (1999) na última década do século XX [43]. O objetivo do algoritmo de SVM é encontrar um hiperplano num espaço N-dimensional (N - o número de recursos) que classifique distintamente os pontos de dados. Para separar as duas classes de pontos de dados, existem muitos hiperplanos possíveis que podem ser escolhidos. O objetivo é encontrar um plano que tenha a margem máxima, ou seja, a distância máxima entre os pontos de dados de ambas as classes. Maximizar a distância da margem fornece algum reforço para que os pontos de dados futuros possam ser classificados com mais precisão.

Vantagens na utilização de SVM [44]:

- Bom funcionamento com uma clara margem de separação dos dados;
- É eficaz em espaços de alta dimensão;
- É eficaz nos casos em que o número de dimensões é maior que o número de amostras;
- Usa um subconjunto de pontos de treino na função de decisão (chamados vetores de suporte), logo, também é eficiente em termos de memória.

Desvantagens [44]:

- Mau funcionamento quando temos um grande conjunto de dados porque o tempo para o processo de treino necessário é maior;
- Também não funciona muito bem, quando o conjunto de dados tem mais ruído, ou seja, as classes de destino estão sobrepostas;
- O SVM não fornece estimativas de probabilidade diretamente, elas são calculadas usando uma validação cruzada.

3.4.4 Redes Neuronais Artificiais

As redes neuronais artificiais ANN formam a base estrutural de grande parte dos modelos de *Deep Learning*. As redes neuronais artificiais são ferramentas poderosas que podem aprender a resolver problemas de forma semelhante à forma

como o cérebro humano funciona. Uma rede neuronal é um sistema adaptativo que aprende usando nós ou neurónios interconectados numa estrutura em camadas. As redes neuronais podem aprender com os dados, para que possam ser treinadas para reconhecer padrões, classificar dados e prever eventos futuros através da experiência [45].

Uma rede neuronal divide a entrada em camadas de abstração, podendo ser treinada usando muitos exemplos para reconhecer padrões de fala ou imagens, por exemplo, assim como o cérebro humano faz. Seu comportamento é definido pela forma como seus elementos individuais estão conectados e pela força, ou pesos, dessas conexões. Esses pesos são ajustados automaticamente durante o treino de acordo com uma regra de aprendizagem especificada, até que a rede neuronal artificial execute a tarefa desejada corretamente.

As redes neuronais são um tipo de abordagem de *Machine Learning* inspirada em como os neurónios sinalizam uns aos outros no cérebro humano. As redes neuronais são especialmente adequadas para modelar relacionamentos não lineares. Alguns exemplos de como as redes neuronais são usadas em aplicações de ML:

- Segmentar imagens e vídeos semanticamente;
- Deteção de objetos em imagens;
- Deteção de cancro, orientando os patologistas a classificar os tumores como benignos ou malignos, com base na uniformidade do tamanho da célula, espessura do aglomerado, mitose e outros fatores.

As redes neuronais, particularmente as redes neuronais profundas (*Deep Learning*), tornaram-se conhecidas pela sua aptidão em aplicações de identificação complexas, como reconhecimento facial, tradução de texto e reconhecimento de voz. Essas abordagens são uma tecnologia chave que impulsionaram a inovação em sistemas avançados de assistência ao condutor e tarefas, incluindo classificação de faixas e reconhecimento de sinais de trânsito.

A ANN pode consistir em vários milhares de neurónios artificiais, e a saída de um neurónio torna-se uma entrada para outro neurónio. O envio de sinais de neurónio para neurónio em ANN é facilitado pelo uso de uma função de ativação. Existem vários tipos diferentes de funções de ativação que podem ser usadas. Uma das funções de ativação mais comuns é a função *sigmoid* logística que é dada pela seguinte equação:

$$S(x) = \frac{1}{1 + exp^{-(\sum w_i x_i + w_0)}} \tag{3.9}$$

Onde w_i é o peso para cada entrada, x_i e w_0 é o peso de polarização. A polarização é um parâmetro adicional que é usado para ajustar a saída junto com a soma ponderada das entradas para o neurónio. Algumas das entradas têm maior importância para a saída do que outras, sendo chamadas de ativação. Os pesos das

entradas são somados e a função de ativação escolhida envia o sinal de saída para o próximo neurónio [46]. A saída de ativação é dada pela equação:

$$output = f(\sum_{i=0}^{3} w_i x_i) \tag{3.10}$$

Uma representação simples de uma rede com um nó e uma saída de ativação é apresentada na Figura 3.11.

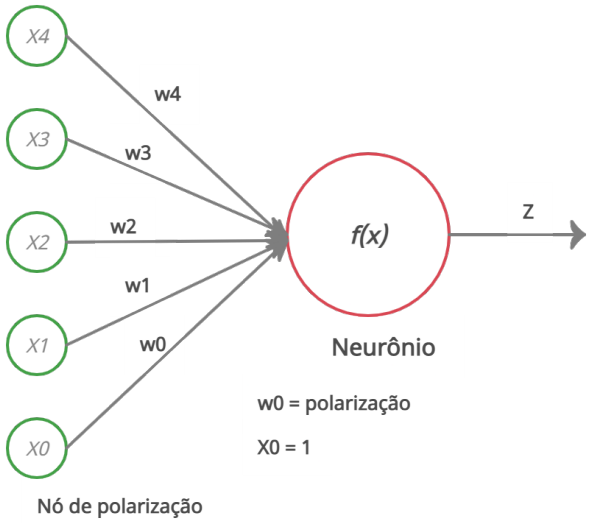


Figura 3.11: Modelo de Neurónio de Rede Neuronal Artificial.

Uma rede neuronal combina várias camadas de processamento, utilizando elementos simples que operam em paralelo. A rede consiste numa camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Em cada camada existem vários nós, ou neurónios, e os nós de cada camada usam as saídas de todos os nós da camada anterior como entradas, de modo que todos os neurónios se interconectem entre si através das diferentes camadas. Cada neurónio normalmente recebe um peso que é ajustado durante o processo de aprendizagem e diminui ou aumenta o peso, alterando a força do sinal desse neurónio.

Tal como outros algoritmos de ML, as redes neuronais podem ser usadas para aprendizagem supervisionado (classificação e/ou regressão) e aprendizagem não supervisionado (reconhecimento de padrões, agrupamento) e os parâmetros do modelo são definidos ponderando a rede neuronal por meio da "aprendizagem" nos dados de treino, normalmente otimizando os pesos para minimizar o erro de previsão.

As ANN são construídas usando múltiplas camadas, divididas em três grupos, camada de entrada, camadas ocultas e camada de saída. O número de neurónios na camada de entrada é equivalente à dimensão (número de variáveis) dos dados

de entrada. A maior parte das camadas numa ANN são as camadas ocultas e são responsáveis por processar a entrada. A saída é então enviada para outra camada oculta para funcionar como entrada, onde o mesmo procedimento é realizado. Isso é chamado de rede neuronal *feed-forward*. Os dados são transferidos em apenas uma direção, ou seja, percorrendo desde a camada de entrada através das camadas ocultas para a camada de saída sem nenhum *loop* ou ciclo. Na Figura 3.12 é apresentada uma topologia geral de uma rede neuronal artificial.

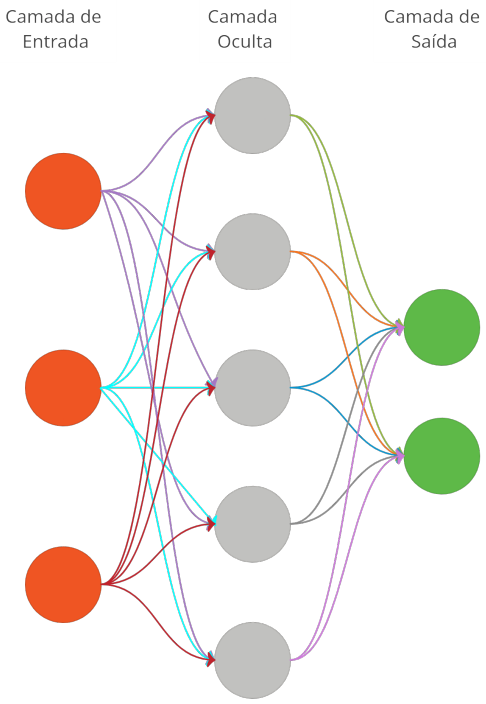


Figura 3.12: Rede Neuronal *Feed-forward* Simples.

A topologia da Figura 3.12 é um exemplo de todas as formas possíveis de estruturar a rede. O programador pode, por exemplo, especificar o número de camadas ocultas, o número de neurónios nas camadas e o número de categorias de saída.

As conexões entre os neurónios podem ser alteradas para enviar sinais de volta pela rede e entre os neurónios da mesma camada. Esses tipos de redes são chamadas de Redes Neuronais Recorrentes, ou *Recurrent Neural Networks* (RNN) [47].

Uma topologia frequentemente usada é a Rede Neuronal Totalmente Ligada (*Fully Connected Neural Network*). Nesse modelo, todos os neurónios nas camadas adjacentes estão conectados. Como todos os neurónios estão conectados entre as camadas, o número de parâmetros aumenta exponencialmente a cada camada adicional. A grande quantidade de parâmetros leva a um grande custo computacional e também pode contribuir para problemas de *overfitting* [48].

A primeira e mais simples rede neuronal foi o *perceptron*, introduzida por Frank

Rosenblatt em 1958 [49]. Consistia num único neurónio e essencialmente um modelo de regressão linear com uma função de ativação *sigmoid*. Desde então, redes neuronais cada vez mais complexas foram exploradas, levando às redes profundas de hoje, que podem conter centenas de camadas.

Deep learning refere-se a redes neuronais com muitas camadas, enquanto redes neuronais com apenas duas ou três camadas de neurónios conectados também são conhecidas como redes neuronais superficiais. O DL tornou-se popularmente conhecido porque elimina a necessidade de extrair recursos de imagens, o que anteriormente desafiava a aplicação de ML no processamento de imagens e sinais. No entanto, embora a extração de recursos possa ser omitida em aplicações de processamento de imagem, alguma forma de extração de recursos ainda é tipicamente aplicada a tarefas de processamento de sinal para melhorar a precisão do modelo. Os tipos de redes neuronais tipicamente usados para aplicações de engenharia incluem:

- Rede neuronal *feedforward*: consiste numa camada de entrada, uma ou mais camadas ocultas e uma camada de saída (uma rede neuronal superficial típica);
- Rede neuronal convolucional: Arquitetura de rede neuronal profunda amplamente aplicada ao processamento de imagens e caracterizada por camadas convolucionais que deslocam janelas na entrada com nós que compartilham pesos, abstraindo a entrada (tipicamente imagem) para mapas de recursos;
- Rede neuronal recorrente: Arquitetura de rede neuronal com *loops* de *feedback* que modelam dependências sequenciais na entrada, como sensores e dados de texto. O tipo de rede neuronal recorrente mais conhecida é uma rede de memória de longo prazo, *Long Short Term Memory* (LSTM).

3.4.5 Funções de Ativação

Uma função de ativação numa rede neuronal define como é que a soma ponderada da entrada é transformada numa saída de um nó ou nós numa camada da rede (mecanismo de ativação é também conhecido por disparo do neurónio) [50]. Por vezes, a função de ativação é também chamada de "função de transferência". Muitas funções de ativação são não lineares e podem ser chamadas de "não linearidade" na camada ou no projeto da rede. A escolha da função de ativação tem um grande impacto na capacidade e desempenho da rede neuronal, e diferentes funções de ativação podem ser usadas em diferentes partes do modelo.

Tecnicamente, a função de ativação é usada dentro ou após o processamento interno de cada nó na rede, embora as redes sejam projetadas para usar a mesma função de ativação para todos os nós numa camada. Todas as camadas ocultas

normalmente usam a mesma função de ativação. A camada de saída normalmente usa uma função de ativação diferente das camadas ocultas e depende do tipo de previsão exigida pelo modelo [51].

As funções de ativação também são tipicamente diferenciáveis, o que significa que a derivada de primeira ordem pode ser calculada para um determinado valor de entrada. Isso é necessário, porque as redes neuronais são normalmente treinadas usando o algoritmo de retro propagação do erro que requer a derivada do erro de previsão para atualizar os pesos do modelo [51].

Existem vários tipos diferentes de funções de ativação usadas em redes neuronais, embora talvez apenas um pequeno número de funções seja usado na prática para camadas ocultas e de saída.

De seguida são apresentadas algumas das funções de ativação de camadas ocultas amplamente utilizadas em arquiteturas:

- *Rectified Linear Unit* (ReLU): A função de ativação linear retificadora, ou função de ativação *Rectified Linear Unit* (ReLU), é talvez a função mais usada para camadas ocultas, isto porque é simples de implementar e eficaz para superar as limitações de outras funções de ativação conhecidas, como *Sigmoid* e *Tanh* [52][53]. Especificamente, é menos suscetível a *vanishing gradient* que impedem o processo de treino dos modelos, embora possa sofrer de outros problemas, como unidades saturadas. A sua grande vantagem diz respeito ao processamento rápido que se deve ao facto de não serem calculadas exponenciais nem divisões [52]. A função ReLU é calculada da seguinte forma [54]:

$$f(x) = max(0, x) \tag{3.11}$$

  Isso significa que, se o valor de entrada x for negativo, um valor zero é devolvido, caso contrário, é devolvido o valor de x (Figura 3.13).

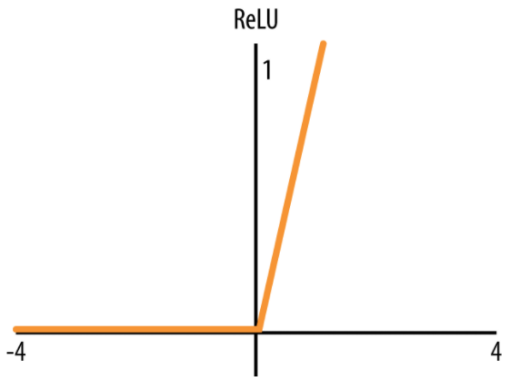


Figura 3.13: Resposta característica da função de ativação: ReLU [55].

- *Sigmoid*: A função de ativação *Sigmoid* também chamada de função logística é uma função não linear utilizada grande parte das vezes em redes neuronais do tipo *feed-forward* [56]. Esta função pode ser utilizada nas camadas ocultas e nas camadas de saída da rede neuronal. A função recebe qualquer valor real como entrada e fornece valores no intervalo entre 0 e 1. Quanto maior o valor de entrada, mais próximo o valor de saída estará de um e quanto menor, mais próximo será o valor de saída de zero (Figura 3.14). A função de ativação *Sigmoid* é calculada da seguinte forma [57][56]:

$$f(x) = \frac{1}{1 + exp^{-x}} \tag{3.12}$$

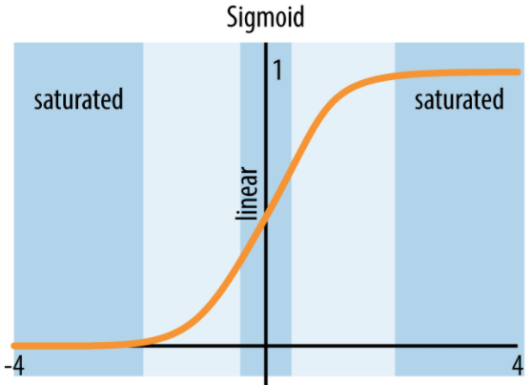


Figura 3.14: Resposta característica da função de ativação: *Sigmoid* [55].

- *Hyperbolic Tangent* (Tanh): A função de ativação da tangente hiperbólica, *Hyperbolic tangent* (Tanh), também conhecida simplesmente por função *Tanh*, é uma função de suavização com o propósito de centrar os valores em zero [58]. É muito semelhante à função de ativação *Sigmoid* e tem a mesma forma de *S*. A função recebe qualquer valor real como entrada e fornece valores no intervalo entre -1 a 1. Quanto maior o valor de entrada, mais próximo o valor de saída estará de 1 e quanto menor, mais próximo será o valor de -1. A função de ativação *Tanh* é calculada da seguinte forma [59][60]:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.13}$$

Uma rede neuronal quase sempre usa a mesma função de ativação em todas as camadas ocultas. Tendencialmente, a função de ativação *Sigmoid* era a função de ativação padrão nos anos 90. Entre os anos 1990 e 2010, a função *Tanh* foi a função de ativação padrão para camadas ocultas [61]. Ambas podem deixar o modelo mais suscetível a problemas durante o processo de treino, através do problema de

vanishing gradient. A função de ativação usada em camadas ocultas normalmente é escolhida com base no tipo de arquitetura da rede neuronal. A Figura 3.15 ilustra as funções de ativação tipicamente usadas para diferentes tipos de arquitetura.

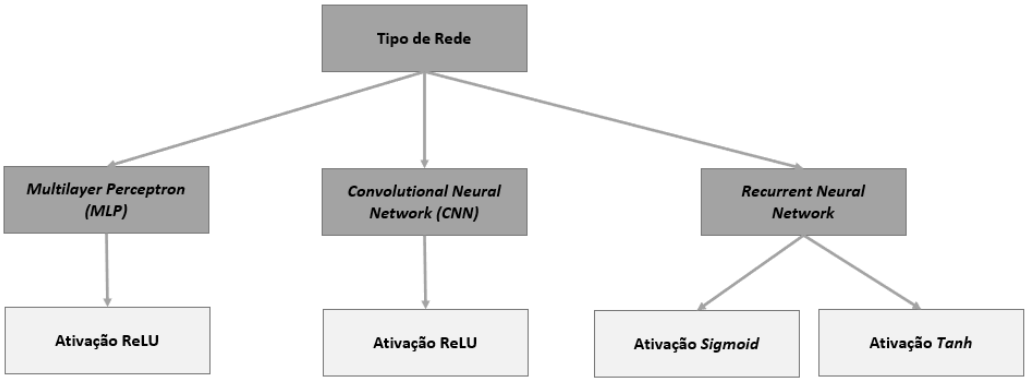


Figura 3.15: Função de ativação de camada oculta.

A camada de saída é a camada num modelo de rede neuronal que emite diretamente uma previsão. Todos os modelos de rede neuronal *feed-forward* têm uma camada de saída. Existem talvez três funções de ativação tipicamente usadas nas camadas de saída [61]:

- Linear: A função de ativação linear também chamada de "identidade" ou "sem ativação". Isso ocorre porque a função de ativação linear não altera a soma ponderada da entrada de forma alguma e, em vez disso, devolve o valor diretamente [51].
- *Sigmoid*: Função *Sigmoid* apresentada anteriormente nas funções de ativação das camadas ocultas.
- *Softmax*: A função *Softmax* é utilizada para calcular a distribuição de probabilidade a partir de um vetor de números reais e produz um vetor de números num intervalo entre 0 e 1. Está relacionada com a função *argmax* que produz um 0 para todas as opções e 1 para a opção escolhida, sendo a função *softmax* uma versão "mais suave" do *argmax*. A Softmax é usada em modelos de múltiplas classes, onde esta devolve as probabilidades de cada uma das classes, tendo como classe alvo a que tiver a maior probabilidade. A função *softmax* é calculada da seguinte forma [61]:

$$f(x) = \frac{exp(x_i)}{\sum_i exp(x_i)} \tag{3.14}$$

A função de ativação usada na camada de saída normalmente é escolhida com base no tipo de problema. A Figura 3.16 ilustra as funções de ativação tipicamente usadas para diferentes tipos de problemas.

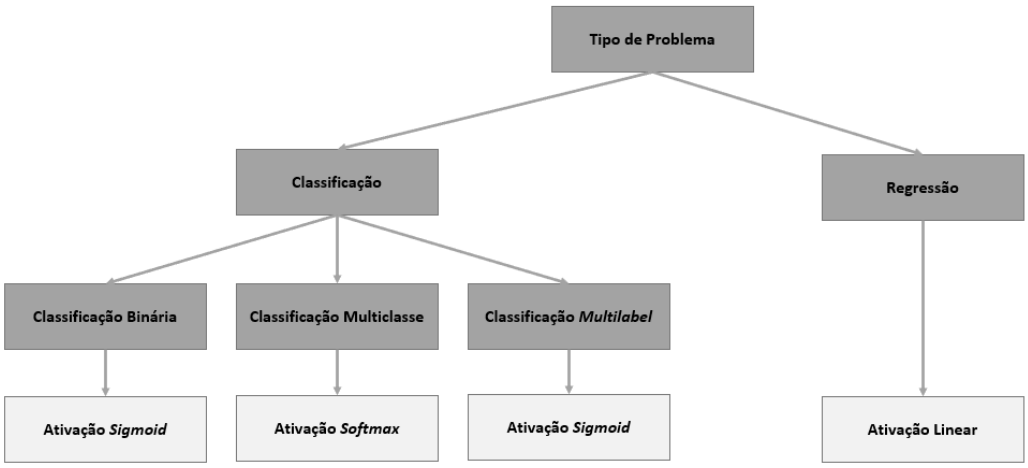


Figura 3.16: Função de ativação da camada de saída.

3.4.6 Algoritmos de *Deep Learning*

Embora os algoritmos de *Deep Learning* apresentem representações de autoaprendizagem, eles dependem de redes neuronais artificiais que imitam a maneira como o cérebro processa informações. Durante o processo de treino, os algoritmos usam elementos desconhecidos na distribuição de entrada para extrair recursos, agrupar objetos e descobrir padrões de dados úteis. Assim como as máquinas de treino para autoaprendizagem, isso ocorre em vários níveis, usando os algoritmos para construir os modelos. Os modelos de *Deep Learning* fazem uso de vários algoritmos. Embora nenhuma rede seja considerada perfeita, alguns algoritmos são mais adequados para executar tarefas específicas.

3.4.6.1 *Autoencoder*

Os auto codificadores, ou *AutoEncoder* (AE) (Figura 3.17), são arquiteturas de DL do tipo *feed-forward* em que a entrada e a saída são idênticas [62]. Geoffrey Hinton projetou codificadores automáticos na década de 1980 para resolver problemas de aprendizagem não supervisionado [62]. São redes neuronais treinadas que replicam os dados da camada de entrada para a camada de saída. AE são usados para fins como investigação farmacêutica, previsão de popularidade e processamento de imagens.

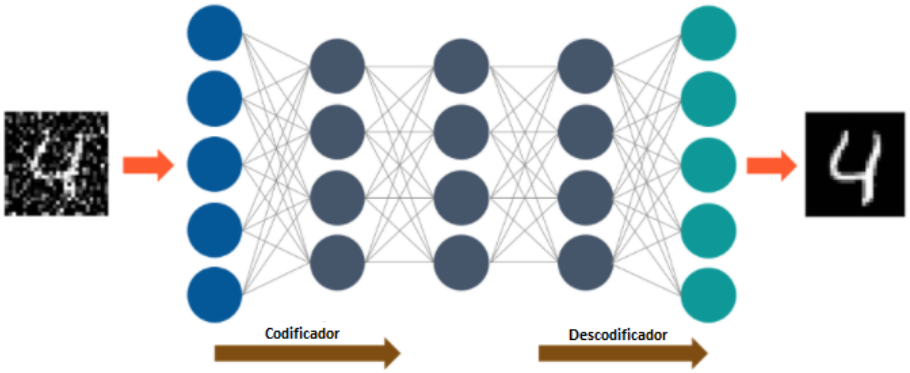


Figura 3.17: *Autoencoder* (AE) [63].

Um auto codificador consiste em três componentes principais: o codificador, descodificador e o cálculo do erro quadrático. AE são estruturados para receber uma entrada e transformá-la numa representação diferente. Eles então tentam reconstruir a entrada original com a maior precisão possível. Quando uma imagem de um dígito não é claramente visível, ela alimenta uma rede neuronal de AE. Os codificadores automáticos primeiro codificam a imagem e, em seguida, reduzem o tamanho da entrada para uma representação menor. Finalmente, o AE descodifica a imagem para gerar a imagem reconstruída.

3.4.6.2 Máquina de *Boltzmann* Restrita

A arquitetura de uma máquina restrita de *Boltzmann*, ou *Restricted Boltzmann Machines* (RBM), foi desenvolvida por Geoffrey Hinton. RBM são redes neuronais estocásticas que podem aprender a partir de uma distribuição de probabilidade sobre um conjunto de entradas. Esse algoritmo de DL é usado para redução de dimensionalidade, classificação, regressão, filtragem colaborativa, aprendizagem de recursos e modelação de tópicos. Os RBM constituem os blocos de construção dos *Deep Belief Networks* (DBN) [64].

RBM consistem em duas camadas (Figura 3.18): unidades visíveis e unidades ocultas. Cada unidade visível está conectada a todas as unidades ocultas. Os RBM têm uma unidade de polarização que está conectada a todas as unidades visíveis e às unidades ocultas e não possuem nós de saída [65].

Os RBM recebem as entradas e traduzem-las num conjunto de números que codifica as entradas na passagem direta. RBM combinam cada entrada com um peso individual e uma polarização geral. O algoritmo passa a saída para a camada oculta e na passagem para trás, os RBM pegam esse conjunto de números e os traduzem para formar as entradas reconstruídas na camada visível. Na camada visível, o RBM compara a reconstrução com a entrada original para analisar a qualidade do resultado.

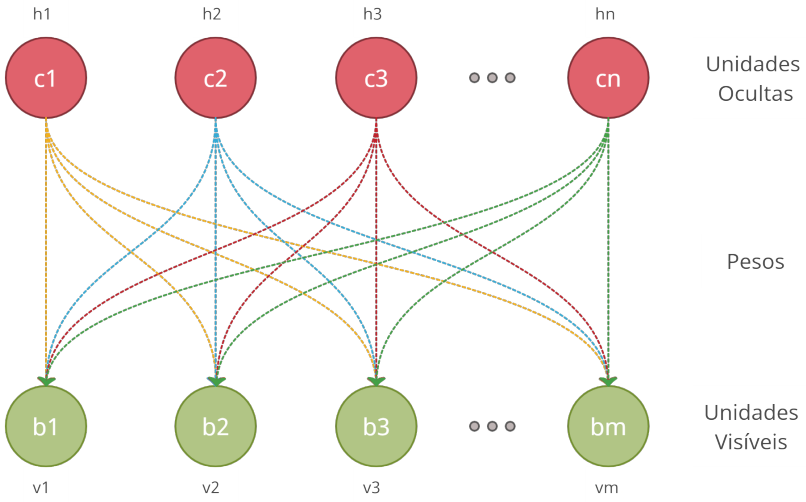


Figura 3.18: Máquina de *Boltzmann* restrita com n unidades ocultas e m unidades visíveis.

3.4.6.3 *Deep Belief Networks*

A arquitetura *Deep Belief Networks* (DBN) são modelos generativos que consistem em múltiplas camadas de variáveis estocásticas e latentes (Figura 3.19) [66]. As variáveis latentes têm valores binários e são frequentemente chamadas de unidades ocultas. DBN são uma pilha de máquinas *Boltzmann* com conexões entre as camadas, e cada camada RBM se comunica com as camadas anteriores e subsequentes. DBN são usados para reconhecimento de imagem, reconhecimento de vídeo e captura de movimento.

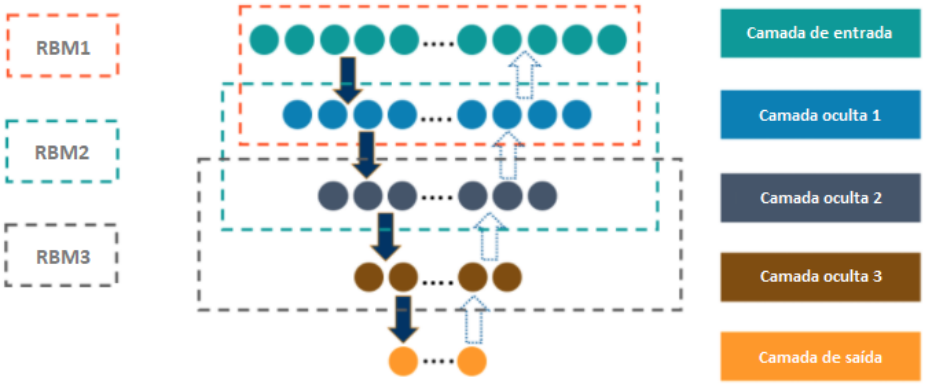


Figura 3.19: *Deep Belief Networks* [67].

As DBN foram desenvolvidas como solução para alguns dos problemas ou dificuldades encontradas em redes neuronais tradicionais, tais como a aprendizagem lenta, insuficiente seleção de parâmetros que origina bloqueios nos mínimos locais e a necessidade de muitos dados para o processo de aprendizagem.

3.4.6.4 Redes Neuronais Recorrentes

As redes neuronais recorrentes ou *Recurrent Neural Networks* (RNN) possuem conexões que formam ciclos direcionados, que permitem que as saídas do LSTM sejam alimentadas como entradas para a fase atual [68]. A saída do LSTM torna-se uma entrada para a fase atual e pode memorizar as entradas anteriores devido à sua memória interna. As RNN são tipicamente usadas para legendar imagens, análise de séries temporais, processamento de linguagem natural, reconhecimento de manuscritos e tradução automática.

As duas abordagens de RNN mais utilizadas dizem respeito às tipologias de *Elman* (Figura 3.20) e à de *Jordan*. A rede de *Elman*, a camada oculta alimenta uma camada de estados nos nós de contexto, onde estas retêm a memória de entradas já passadas. A tipologia de *Jordan*, ao contrário da abordagem anterior, em vez de manter o histórico da camada oculta, armazena a informação da camada de saída na camada de estado [69].

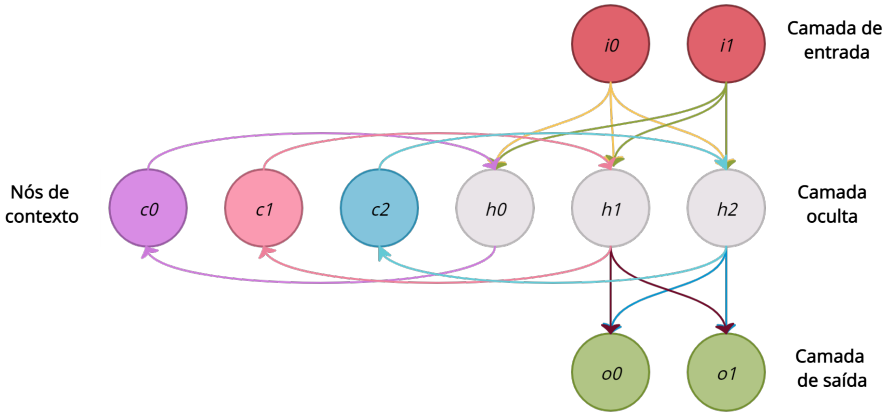


Figura 3.20: Rede Neuronal Recorrente (RNN) - Tipologia de *Elman*.

3.4.6.5 Redes Neuronais Convolucionais

Uma das redes neuronais profundas (*Deep Learning*) mais conhecidas são as redes neuronais convolucionais, *Convolutional Neural Networks* (CNN) (Figura 3.21). Desde a década de 1950, investigadores procuram criar um sistema que possa entender dados visuais. Nos anos seguintes, esse campo passou a ser conhecido como Visão Computacional. Em 2012, a visão computacional deu um salto quântico quando um grupo de investigadores da Universidade de Toronto desenvolveu um modelo de Inteligência Artificial (IA) [70] que superou os melhores algoritmos de reconhecimento de imagem.

O sistema de IA, que ficou conhecido como AlexNet (em homenagem ao seu principal criador, Alex Krizhevsky), venceu o concurso de visão computacional ImageNet de 2012 com uma incrível precisão de 85% [71].

No coração do AlexNet estavam as Redes Neuronais Convolusionais, um tipo especial de rede neuronal que imita aproximadamente a visão humana. Ao longo dos anos, as CNN tornaram-se uma parte muito importante de muitas aplicações de Visão Computacional.

As CNN foram desenvolvidas e usadas pela primeira vez por volta da década de 1980. O máximo que uma CNN podia fazer naquela época era reconhecer dígitos manuscritos. O importante a ser lembrado sobre qualquer modelo de DL, é que ele requer uma grande quantidade de dados para treinar e também requer muitos recursos de computação. Essa foi uma grande desvantagem para as CNN naquele período [72].

Em DL, uma rede neuronal convolucional é uma classe de redes neuronais profundas, tipicamente usadas para analisar imagens. Agora, quando pensamos numa rede neuronal, pensamos em multiplicações de matrizes, mas esse não é o caso do ConvNet (ou CNN). Ele usa uma técnica especial chamada "Convolução". Na matemática, a convolução é uma operação matemática em duas funções que produz uma terceira função que expressa como a forma de uma é modificada pela outra. De uma forma mais simples de entender, o papel do ConvNet é reduzir as imagens num formato mais fácil de processar, sem perder recursos que são críticos para obter uma boa previsão [73].

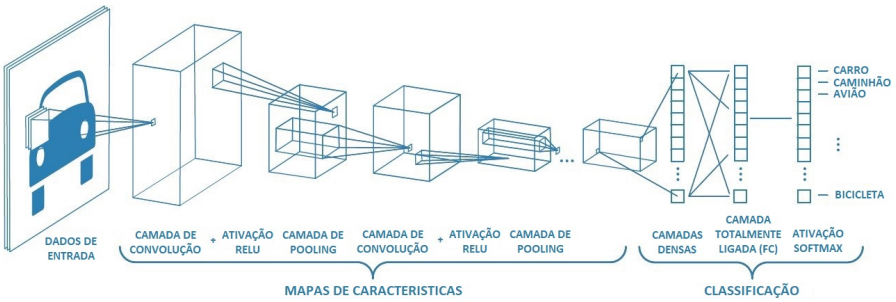


Figura 3.21: Arquitetura tradicional de uma CNN [73].

A rede neuronal convolucional é diferenciada de outras redes neuronais pelo seu desempenho superior com entradas do tipo de imagem. Ela tem três tipos principais de camadas, que são, a camada de convolução, camada de agrupamento (*pooling*) e a camada totalmente ligada, *Fully Connected* (FC).

A camada de convolução é a primeira camada de uma rede convolucional. Enquanto as camadas convolucionais podem ser seguidas por camadas convolucionais adicionais ou camadas de agrupamento, a camada totalmente ligada é a camada final. Ao longo da rede CNN é aumentada a sua complexidade, identificando maiores porções da imagem. As camadas anteriores concentram-se em recursos simples, como cores e superfícies. À medida que os dados da imagem progridem pelas camadas da CNN, ela começa a reconhecer elementos e características da imagem até finalmente identificar o objeto pretendido.

Camada de Convolução

A camada de convolução é o bloco de construção central de uma CNN e é onde ocorre a maior parte da computação. Requer alguns componentes, que são dados de entrada, um filtro e um mapa de recursos. Suponhamos que a entrada é uma imagem colorida, que é composta por uma matriz de píxeis tridimensional (3D). Isso significa que a entrada terá três dimensões, altura, largura e profundidade que correspondem ao RGB numa imagem. Também temos um detetor de fisionomia, também conhecido como *kernel* ou filtro, que irá percorrer os campos recetivos da imagem, verificando se a fisionomia está presente. Esse processo é conhecido como convolução [55].

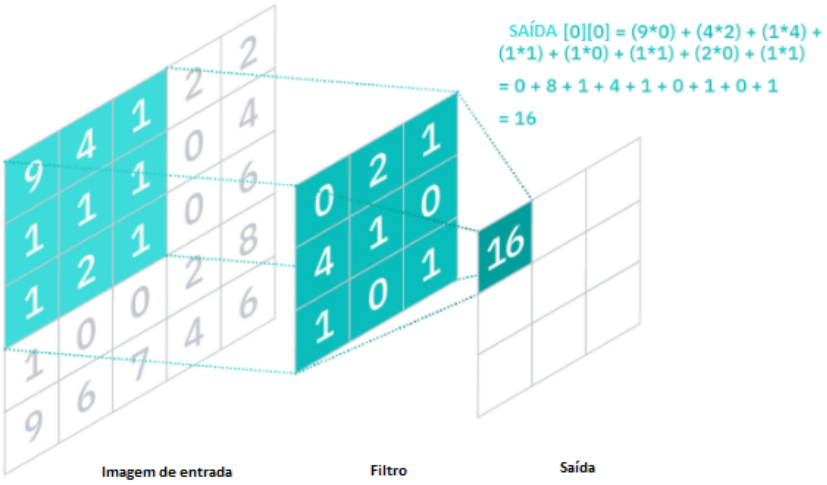


Figura 3.22: Operação de Convolução [74].

Como podemos ver pela Figura 3.22, cada valor de saída no mapa de recursos não precisa de se conectar a cada valor de píxel na imagem de entrada. Ele só precisa conectar-se ao campo recetivo, onde o filtro está sendo aplicado. Como a matriz de saída não precisa mapear diretamente para cada valor de entrada, as camadas de convolução (e de *pooling*) são tipicamente chamadas de camadas "parcialmente ligadas".

Camada de *Pooling*

A camada de agrupamento (*pooling*), também conhecida como *downsampling*, tem como função a redução de dimensão da imagem, reduzindo o número de parâmetros na entrada. Semelhante à camada de convolução, a operação de agrupamento percorre um filtro em toda a entrada, mas a diferença é que esse filtro não possui pesos. Em vez disso, o *kernel* aplica uma função de agregação aos valores dentro do campo recetivo, preenchendo a matriz de saída. Existem dois tipos principais de operações de agrupamento:

- *Max pooling* (Figura 3.23): À medida que o filtro se move pela entrada, ele seleciona o píxel com o valor máximo para enviar para a matriz de saída.
- Agrupamento médio: à medida que o filtro se move pela entrada, ele calcula o valor médio dentro do campo recetivo para enviar para a matriz de saída.

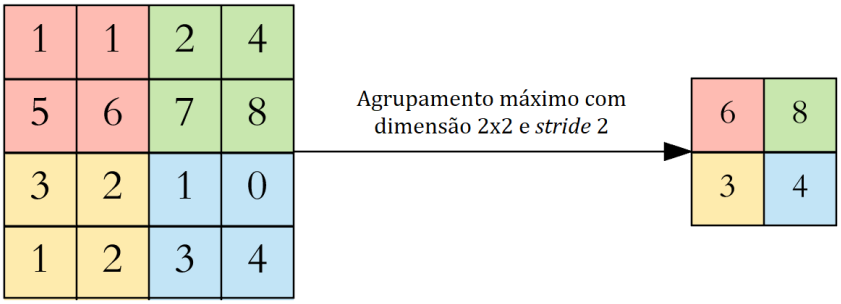


Figura 3.23: Operação de agrupamento máximo [75].

Embora muitas informações sejam perdidas na operação de *pooling*, ela também traz vários benefícios para a CNN, ajudando a reduzir a complexidade, melhorar a eficiência e limitar o risco de *overfitting*.

Camada de Ativação

Uma função de ativação decide se um neurónio deve ser ativado ou não. Isso significa que ele decidirá se a entrada do neurónio para a rede é importante ou não no processo de previsão usando operações matemáticas mais simples, sendo estas, funções não lineares. O papel da função de ativação é derivar a saída de um conjunto de valores de entrada alimentados por um nó (ou camada).

Camadas *Fully Connected* (FC)

Na última camada da CNN é utilizada uma camada totalmente ligada (FC) tal como apresentado na Figura 3.21. Uma camada totalmente ligada forma-se quando a matriz achatada da camada de *pooling* é alimentada como entrada, que classifica e identifica as imagens [55].

Regularização de *Dropout*

O mecanismo de regularização de desistências (*Dropout*) refere-se ao processo de ignorar aleatoriamente certos nós numa camada durante o treino. Esse processo consiste numa ideia simples, mas com grande impacto no desempenho das CNN. A ideia é calcular a média de vários modelos de um conjunto ao invés de utilizar modelos únicos.

Normalização dos *Batch*

A normalização é uma ferramenta de pré-processamento de dados usada para trazer os dados numéricos para uma escala comum sem distorcer a sua forma. Geralmente, quando inserimos os dados numa máquina ou algoritmo de DL, procuramos alterar os valores para uma escala equilibrada. A razão pela qual normalizamos é, em parte, para garantir que o nosso modelo vai generalizar adequadamente. A normalização do lote (*batch*), é um processo para tornar as redes neuronais mais rápidas e estáveis por meio da adição de camadas extras numa rede. A nova camada realiza as operações de uniformização e normalização na entrada de uma camada proveniente [76].

Modelos baseados nas CNN

Alguns modelos, com as respetivas configurações, baseadas na arquitetura CNN são apresentadas de seguida:

- **Inception**: A rede *Inception* (Figura 3.25a) aparece em 2014, quando um grupo de investigadores da Google publicou o artigo *Going Deeper with Convolutions*. A principal característica desta arquitetura foi construir uma rede neuronal mais profunda e melhorar a utilização dos recursos de computação dentro da rede. *Inception* usa uma rede de 22 camadas de profundidade, enquanto reduz o número de parâmetros em 12 vezes, obtendo resultados significativamente mais precisos. A rede usou uma CNN inspirada nas redes clássicas (*AlexNet* e *VGGNet*) [77].

- **Xception**: A rede *Xception* (Figura 3.25b) da Google, significa *Extreme Version of Inception*. Com uma convolução separável em profundidade, é ainda melhor que *Inception*-v3 para conjuntos de dados *ImageNet*. *Xception* é uma rede neuronal convolucional com 71 camadas de profundidade. Pode-se carregar uma versão pré-treinada da rede treinada em mais de um milhão de imagens do conjunto de dados *ImageNet*. A rede pré-treinada pode classificar imagens em 1000 classes de objetos. Como resultado, a rede aprendeu representações ricas em recursos para uma ampla variedade de imagens. *Xception* é uma arquitetura eficiente que se baseia em dois pontos principais: Convolução separável em profundidade e atalhos entre blocos de convolução como em *ResNet* [78].

- **VGG**: A rede *VGGNet* (Figura 3.26) foi desenvolvida em 2014 pelo departamento de Ciências da Engenharia *Visual Geometry Group* (VGG) da Universidade de Oxford (daí o nome VGG). Os componentes de construção são exatamente os mesmos do *LeNet* e *AlexNet*, exceto que o *VGGNet* é uma rede ainda mais profunda com mais camadas convolutivas, de agrupamento e densas. Aparte disso, nenhum novo componente é introduzido. *VGGNet*, também conhecido como VGG16 (Figura 3.26a), consiste em 16 camadas de peso: 13 camadas convolutivas e 3 camadas totalmente ligadas [79].

- **ResNet**: A Rede Neuronal Residual (*ResNet*) foi desenvolvida em 2015 por um grupo de pesquisa da Microsoft. Eles introduziram uma nova arquitetura de módulo residual com conexões de salto. A rede também apresenta uma pesada normalização em lote para as camadas ocultas. Essa técnica permitiu que a equipa treinasse redes neuronais muito profundas com 50 (Figura 3.27a), 101 e 152 camadas de peso enquanto ainda tinha menor complexidade do que redes menores como *VGGNet* (19 camadas)(Figura 3.26b). A *ResNet*

conseguiu atingir uma taxa de erro das 5 principais (TOP-5) de 3,57% na competição *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) 2015, que superou o desempenho de todas as *ConvNets* anteriores [80].

- **EfficientNet**: *EfficientNet* (Figura 3.24) é uma arquitetura baseada em CNN que utiliza um coeficiente composto com escalas fixas. O *EfficientNet* usa um coeficiente composto para redimensionar uniformemente a largura, profundidade e resolução da rede de uma maneira baseada em princípios. O método de redimensionamento composto é justificado pela intuição de que, se a imagem de entrada for maior, a rede precisa de mais camadas para aumentar o campo recetivo e mais canais para identificar padrões mais refinados na imagem maior. A variante *EfficientNet-B0* é baseada nos blocos residuais de bloqueio invertido do *MobileNetV2*, além dos blocos de compressão e excitação [81].

- **ConvNeXT**: O modelo *ConvNeXT* foi proposto em *A ConvNet for the 2020s*, escrito por *Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, Saining Xie*. O *ConvNeXT* é um modelo convolucional puro (*ConvNet*), onde foram reexaminados os espaços de *design* e testadas as limitações do que uma *ConvNet* pura pode alcançar. Através da sua experiência eles encontraram uma maneira de modernizar uma rede puramente baseada em convolução [82].

- **MobileNet**: O modelo *MobileNet* (Figura 3.27b) é uma classe de modelos eficientes para aplicações embebidas e de visão móvel. Os *MobileNets* são baseados numa arquitetura simplificada que usa convoluções separáveis em profundidade para construir redes neuronais profundas leves. Apresentam dois hiperparâmetros globais simples que alternam eficientemente entre latência e precisão. Esses hiperparâmetros permitem que o construtor de modelos escolha o modelo de tamanho certo para a sua aplicação com base nas restrições do problema. Apresenta um forte desempenho em comparação com outros modelos populares na classificação *ImageNet* [83].

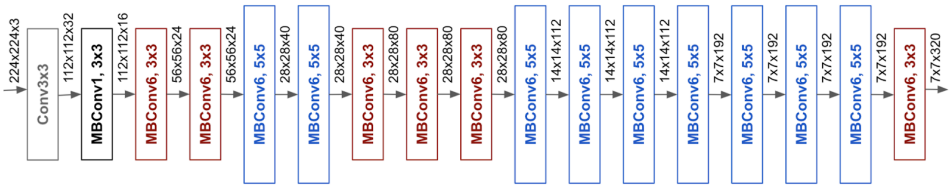
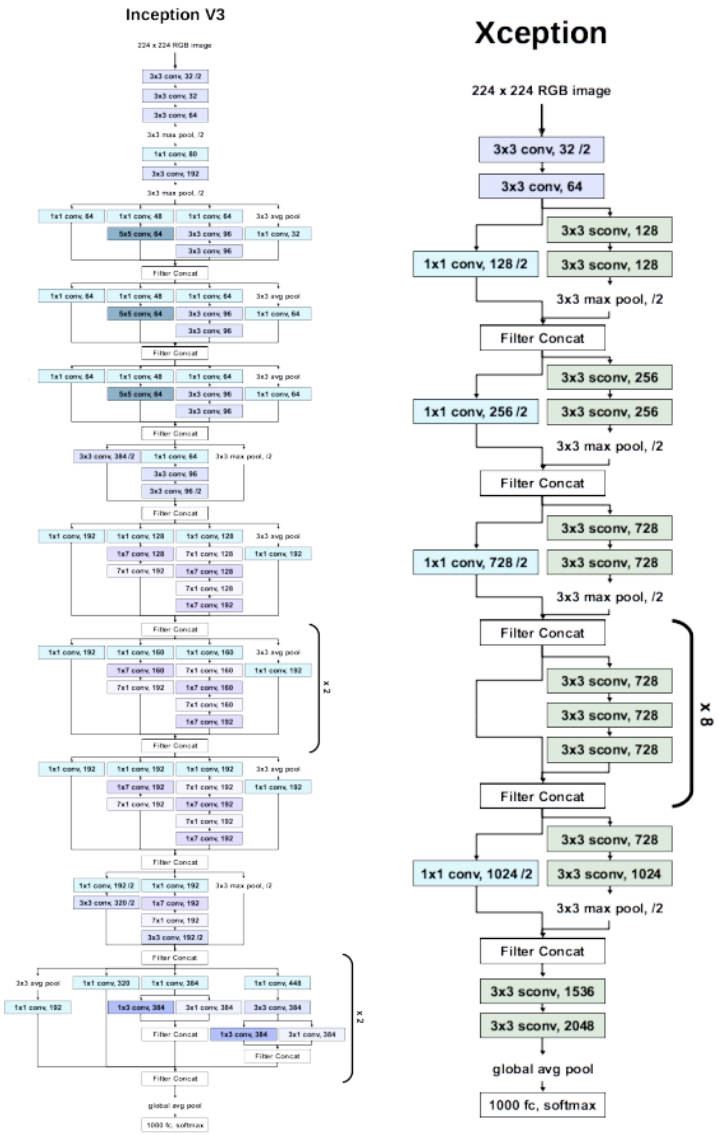
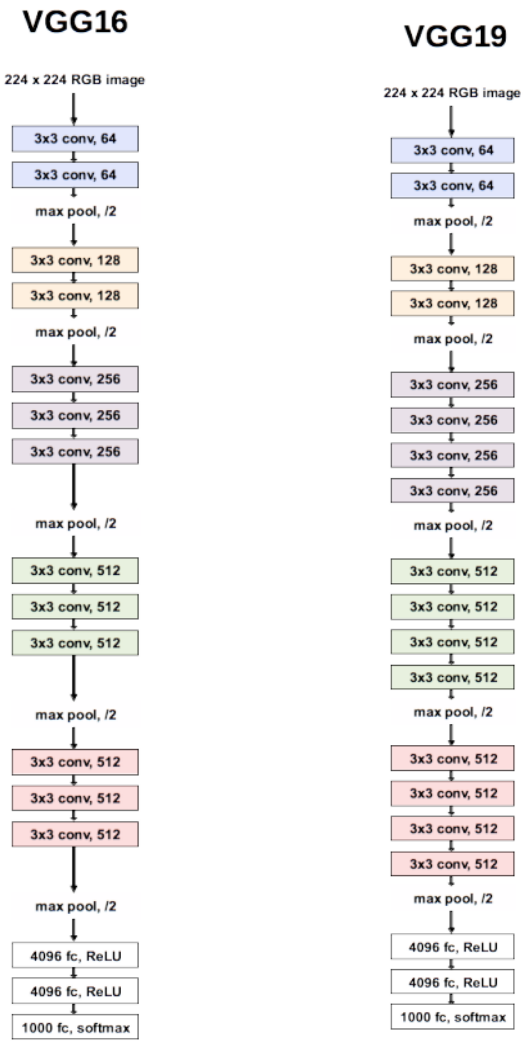


Figura 3.24: *Efficientnet* [81].



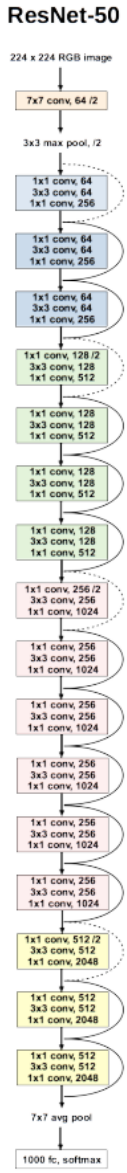
(a) *Inception V3* (b) *Xception*

Figura 3.25: Diagramas *Inception V3* e *Xception* [84].

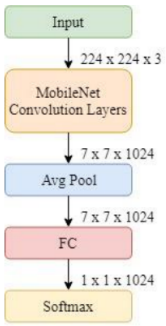


(a) VGG16 (b) VGG19

Figura 3.26: Diagramas VGG [84].



(a) ResNet50



(b) MobileNet

Figura 3.27: Diagramas *ResNet50* [84] e *MobileNet* [85].

3.4.7 *Transfer Learning*

Transfer Learning é uma técnica de ML em que um modelo pré-treinado numa tarefa é reaproveitado numa outra tarefa semelhante. Um modelo pré-treinado é uma rede guardada que foi previamente treinada num grande conjunto de dados, normalmente numa tarefa de classificação de imagem em grande escala. Pode-se usar o modelo pré-treinado sem ter sido treinado ou então usar o *transfer learning* para personalizar esse modelo para uma determinada tarefa. A intuição por detrás do *transfer learning* para classificação de imagens é que, se um modelo for treinado num grande conjunto de dados e com um número de classes significativa, esse modelo servirá efetivamente como um modelo genérico do mundo visual. Pode-se aproveitar esses mapas de recursos (conhecimento) aprendidos sem a necessidade de começar do zero, treinando um modelo grande num pequeno conjunto de dados (em comparação com o conjunto de dados pré-treinado). Quando se fala de treinar um modelo do zero, quer dizer que o modelo começa com zero conhecimento, e a estrutura e os parâmetros do modelo começam como suposições aleatórias. Na prática, isso significa que os pesos do modelo são inicializados aleatoriamente e precisam passar por um processo de treino para serem otimizados.

Existem duas principais técnicas para uso de *transfer learning*:

1. Extração de recursos ou *Feature Extraction* (FE): esta técnica consiste em congelar algumas camadas da rede pré-treinada (tipicamente a camada totalmente conectada) e treinar as restantes camadas para o novo conjunto de dados. A técnica de extração de recursos é útil quando temos um grande conjunto de dados e diferente do original e precisamos reduzir o número de recursos sem perder nenhuma informação importante ou relevante.

2. Ajuste fino ou *Fine-Tuning* (FT): esta técnica consiste em treinar todas as camadas/parâmetros da rede pré-treinada com uma taxa de aprendizagem relativamente baixa para não deteriorar os pesos obtidos do *transfer learning*. Esta técnica pode ser usada quando temos um conjunto de dados grande e semelhante ao conjunto de dados original.

Capítulo 4

Metodologias e Desenvolvimento

Neste capítulo serão apresentadas as metodologias usadas para o desenvolvimento deste trabalho. É feita uma apresentação das ferramentas e do *hardware* usado, etapas do desenvolvimento, classificação de imagens, arquiteturas de classificação e por último a plataforma.

4.1 Ferramentas de Suporte

Para o desenvolvimento desta dissertação foram utilizadas diversas ferramentas de forma a alcançar os objetivos propostos, isto é, linguagem de programação, bibliotecas e *hardware*, que se descreve de seguida. A linguagem de programação utilizada para o desenvolvimento dos algoritmos, preparação e o tratamento dos dados foi o *Python* (versão 3.10.2 64 bits) [86]. *Tensorflow* (versão 2.8.0) [87] e *PyTorch* (versão 1.11.0) [88] foram as duas bibliotecas de ML e DL utilizadas no desenvolvimento e implementação dos algoritmos propostos. *Numpy* (versão 1.22.2) [89] foi a biblioteca utilizada na preparação e manipulação dos dados, enquanto as bibliotecas *Seaborn* (versão 0.11.2) [90] e *Matplotlib* (versão 3.5.1) [91] foram usadas para a visualização de dados assim como dos resultados. *Scikit-learn* (versão 1.0.2) [92] foi utilizada na aplicação de algumas métricas para a classificação dos modelos. Para a platarforma de testes foi criada uma aplicação *Web* com recurso à biblioteca *Streamlit*. O *Integrated Development Environment* (IDE) utilizado para o desenvolvimento de *software* foi o editor de código *Visual Studio Code*.

As características de *hardware* da máquina utilizada no desenvolvimento do trabalho desta dissertação estão presentes na Tabela 4.1.

Tabela 4.1: Características da máquina utilizada para o desenvolvimento do trabalho.

	Máquina
Sistema Operativo	Windows 10 Home (21H2 - 19044.1682)
CPU	i7-4710MQ 2.5 GHz
GPU	NVIDIA GeForce 840M 2 GB
RAM	16 GB

Para a implementação do algoritmo de classificação de *scrap* de *capply* foi usado um *Raspberry Pi* 4 8GB (Figura 4.1) e uma câmara *Raspberry Pi Module* 2 (Figura 4.2).

Nas Tabelas 4.2 e 4.3 são apresentadas as características do *Raspberry Pi* 4 e da câmara *module* 2, respetivamente.

***Raspberry Pi* 4**

Figura 4.1: Raspberry Pi 4 8GB.

Tabela 4.2: Características do Raspberry Pi 4.

Características	
CPU	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memória	8GB LPDDR4
Conectividade	- Wireless SoC SD 2.4 GHz v8) comparável e (ARM 5.0 de GHz quad-core IEEE OpenGL 802.11b/g/n/ac - LAN, Bluetooth 5.0, BLE - Gigabit Ethernet - 2x portas USB 3.0 - 2x portas USB 2.0
GPIO	Standard 40-pin GPIO header
Vídeo e Som	- 2x portas micro HDMI (suporte até 4Kp60) - 2-lane MIPI DSI display port - 2-lane MIPI CSI camera port - Áudio estéreo de 4 pólos e porta de vídeo composto
Multimédia	- H.265 (4Kp60 decode) - H.264 (1080p60 decode, 1080p30 encode) - OpenGL ES, 3.0 graphics
Suporte cartão SD	Micro SD para carregar o sistema operativo e armazenamento de dados
Alimentação	- 5VDC via USB-C (mínimo 3A) - 5VDC via GPIO header (mínimo 3A) - Power over Ethernet (PoE)
Temperatura de operação	0 a 50ºC

Câmara *Module* 2

Figura 4.2: Raspberry Pi câmara *Module* 2.

Tabela 4.3: Características da câmara *Module* 2.

Características	
Sensor	Sony Exmor IMX219 8 Megapixel
Suporte	1080p30, 720p60 e 640x480p90
Compatibilidade	Raspberry Pi Model A/B/B++
Alimentação	2A

4.2 Estrutura de *Hardware*

Na implementação deste trabalho tem-se a seguinte estrutura de *hardware* (Figura 4.3), onde o Raspberry Pi 4 capta as imagens da câmara modelo 2 da Raspberry Pi, sendo de seguida processadas e classificadas, de acordo com o tipo de imperfeição apresentada na bobine. Por último, esta informação é enviada para o *Programmable Logic Controller* (PLC) ou controlador lógico programável da marca Beckhoff, existente na máquina, onde este deverá enviar esta informação para a base de dados.

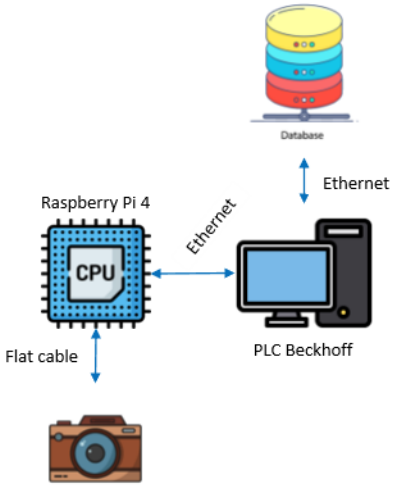


Figura 4.3: Estrutura de *Hardware*.

4.3 Etapas do Desenvolvimento

Para a realização deste trabalho foi necessário recolher imagens de bobines de *capply* com imperfeições. Este processo foi-se repetindo ao longo do desenvolvimento deste trabalho, de forma a que o *dataset* pudesse conter várias características de cada classe de imperfeições de bobines de *capply*. Após a recolha dos dados foi necessária uma preparação dos mesmos. Tendo o *dataset* preparado, foi criado um processo automatizado, com várias etapas, para a tarefa de classificação deste trabalho (Figura 4.4). Assim, numa fase inicial do processo, os dados foram lidos e submetidos a técnicas de pré-processamento, como por exemplo o redimensionamento de imagem. Em seguida, o conjunto de dados foi subdividido em três componentes (treino, validação e teste) e submetidos às técnicas de processamento definidas. Posteriormente, para cada um dos modelos implementados, foi iniciado o processo de treino, parametrização e otimização. De seguida seguem-se os testes e avaliação dos modelos. De acordo com os resultados obtidos dos testes anteriormente feitos é feita uma seleção do modelo que melhores resultados apresenta. Posto isto, é carregado o algoritmo no *hardware* e por último são feitos testes, avaliações e análises em contexto real (na máquina).

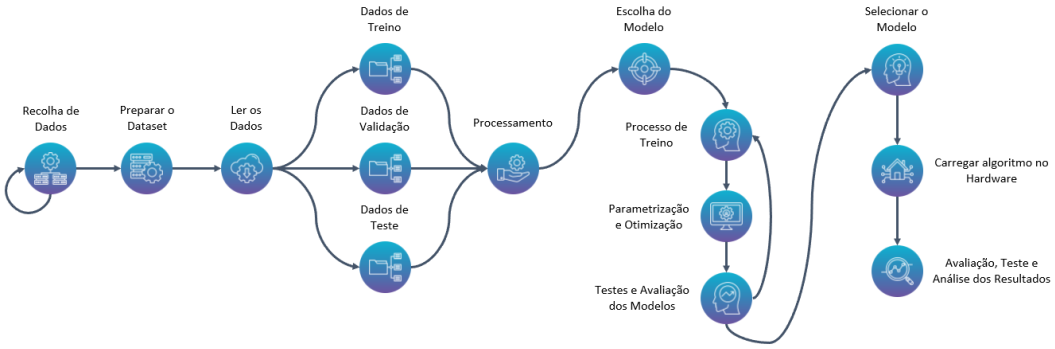


Figura 4.4: Etapas tidas em consideração no desenvolvimento deste trabalho.

4.4 Classificação de Imagens

Dataset

Este trabalho tem como objetivo a implementação de metodologias de *Deep Learning* para classificar automaticamente imagens de bobines de *capply* com imperfeições que são enviadas para uma máquina que irá remover o material com defeito. Existem diversas classes de causas para segregar material de uma bobine, tal como é apresentada na Figura 4.5.

Figura 4.5: Catálogo de imperfeições.

Para este trabalho foram consideradas apenas algumas destas classes de imperfeições, tais como, "(4) Bobine vazia", "(11) Falha de borracha", "*(105) Capply* OK", "(610) Bobine amassada/danificada", "(611) Excesso de adesividade", "(613) Mau enrolamento/distribuição irregular" e "(614) Tira dobrada/torcida".

O conjunto de dados *dataset* utilizado para a implementação de metodologias de classificação de imagens de imperfeições de *capply* foram capturadas de um dispositivo móvel com alta definição.

Este *dataset* é composto por 1935 amostras, até ao momento de escrita desta dissertação, e conta com 7 classes de dados apresentadas anteriormente (4, 11, 105, 610, 611, 613 e 614), e possuem um dimensionamento original de 1600x1600 píxeis.

Na Tabela 4.4 está contida a informação sobre o número de amostras, o número de amostras por classe, as classes presentes no conjunto de dados, assim como, a tarefa de classificação.

Tabela 4.4: Composição do *Dataset*.

Classes	Nº de amostras por classe	Nº de amostras	Tarefa
4: Bobine vazia 11: Falha de borracha 105: Capply OK 610: Bobine amassada/danificada 611: Excesso de adesividade 613: Mau enrolamento/distribuição irregular 614: Tira dobrada/torcida	4: 382 11: 401 105: 405 610: 155 611: 127 613: 400 614: 74	1935	Múltiplas classes

Processamento do *Dataset*

As configurações de processamento efetuadas ao conjunto de dados incluem normalização de imagem (média e desvio), rotações e inversões aleatórias às imagens. Como anteriormente referido, as imagens originalmente têm uma dimensão de 1600x1600 e foram redimensionadas e centradas. Incluído no processamento é ainda convertida a imagem (matriz) num tensor (*array* multidimensional imutável). Na Figura 4.6 encontram-se alguns exemplos das transformações e processamento efetuado ao *dataset*.

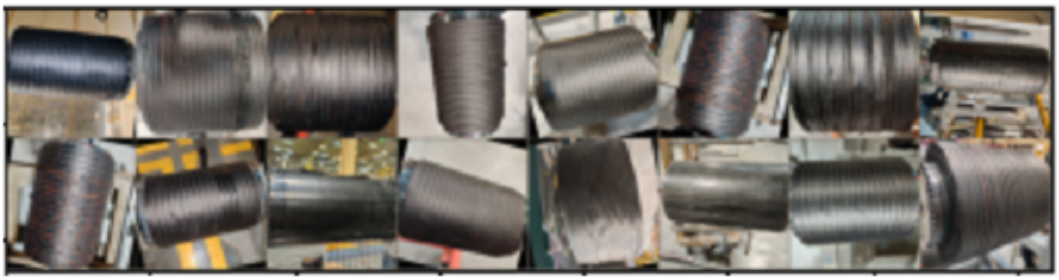


Figura 4.6: Exemplos de imagens do *dataset*, depois de transformadas.

Como este trabalho consistiu na utilização da técnica de *transfer learning* de redes pré-treinadas com o *dataset* ImageNet, as normalizações usadas foram as do ImageNet (média = [0.485, 0.456, 0.406] e desvio = [0.229, 0.224, 0.225]).

Conforme é possível verificar na Tabela 4.4, na coluna do número de amostras por classe, o conjunto de dados encontra-se desequilibrado. Tendo isso em consideração, foram testadas duas técnicas de *Deep Learning* para colmatar este problema. Uma destas técnicas é chamada de sobreamostragem (*oversampling*) aleatória que consiste em duplicar aleatoriamente exemplos da classe minoritária e adicioná-los ao conjunto de dados de treino. A outra técnica usada foi o *stratified* aquando o uso da função de divisão do *dataset* (*train_test_split()*). A amostragem estratificada para dividir um conjunto de dados alivia o problema de amostragem aleatória em

conjuntos de dados com uma distribuição de classes desequilibrada. Aqui, a distribuição de classes em cada um dos conjuntos de treino e validação é preservada.

4.5 Arquiteturas de Classificação de Imagem

De seguida serão apresentadas as arquiteturas implementadas nas tarefas de classificação de imagens. Os modelos implementados incluem a *Xception*, *Inception-v3*, *ResNet18*, *ResNet50*, *ResNet152*, *VGG16*, *VGG19*, *EfficientNet-B0*, *EfficientNet-B7*, *ConvNeXT-Small* e *MobileNet-v2*.

4.5.1 Xception

A arquitetura *Xception* (Figura 4.7) foi implementada com recurso ao *Tensorflow* (tf). A *Xception* contempla uma primeira camada de entrada para redimensionar os dados de entrada (*input_2*), seguida de uma camada *sequential*, uma *rescaling* e uma camada *xception*. Tem-se ainda uma camada de normalização de lote e um agrupamento médio global. Por último tem-se quatro camadas densamente ligadas e um *Dropout*. Nesta arquitetura foram implementadas as técnicas de extração de recursos e ajuste fino, em simultâneo. O algoritmo de otimização usado neste modelo, durante o processo de treino foi o *Adam* com a função de cálculo de perdas de entropia cruzada categórica (*categorical_crossentropy*). A métrica de desempenho utilizada foi a exatidão (ACC).

```
Model: "Xception"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 299, 299, 3)]     0

 sequential (Sequential)     (None, 299, 299, 3)       0

 rescaling (Rescaling)       (None, 299, 299, 3)       0

 xception (Functional)       (None, 10, 10, 2048)      20861480

 batch_normalization_4 (Batc  (None, 10, 10, 2048)     14336
 hNormalization)

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 512)               1049088

 dense_1 (Dense)             (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 8)                 1032

=================================================================
Total params: 22,090,160
Trainable params: 22,025,392
Non-trainable params: 64,768
_________________________________________________________________
```

Figura 4.7: Arquitetura *Xception*.

4.5.2 Inception-v3

Ao contrário do modelo anteriormente apresentado, a *Inception-v3* (Figura 4.8) foi implementada com recurso ao *PyTorch*, tal como as arquiteturas que se seguem. A *Inception-v3* contempla cinco primeiros módulos denominados por *BasicConv2d*. Cada um dos *BasicConv2d* é constituído por uma camada de convolução e uma de normalização de lotes. Entre os três primeiros módulos de *BasicConv2d* e no final dos outros dois, ainda existe um agrupamento máximo. De seguida seguem-se três módulos *InceptionA* que contemplam sete módulos *BasicConv2d*, um módulo *InceptionB* com quatro módulos *BasicConv2d*, quatro módulos *InceptionC* com dez módulos *BasicConv2d*, um módulo *InceptionAux* com dois módulos *BasicConv2d* e uma camada totalmente ligada (FC), um módulo *InceptionD* com seis módulos *BasicConv2d* e ainda dois outros módulos *InceptionE* com nove módulos *BasicConv2d*. As três últimas camadas correspondem a uma camada de agrupamento médio adaptativo (*AdaptiveAvgPool2d*), um *Dropout* e a uma camada FC linear. Nesta arquitetura foi implementada apenas a técnica de ajuste fino. A função de otimização usada, foi a descida de gradiente estocástica, *Stochastic gradient descent* (SGD), juntamente com a implementação do parâmetro *momentum* [93]. A função de cálculo de perdas utilizada foi a entropia cruzada (*CrossEntropyLoss*). As métricas de desempenho utilizadas foram a exatidão (ACC) e a área sob a curva ROC (AUC).

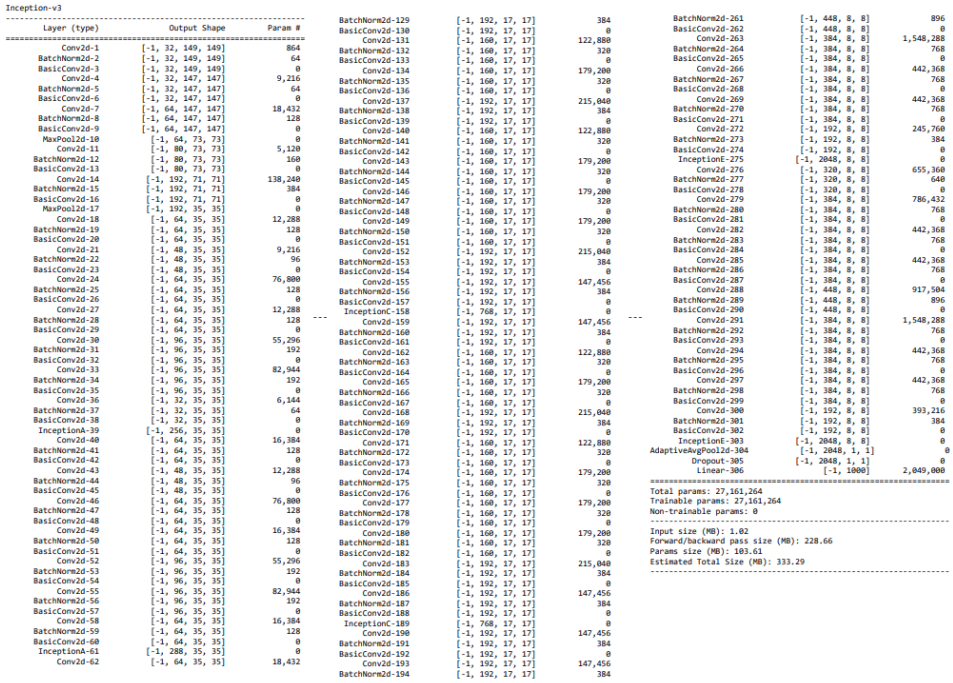


Figura 4.8: Arquitetura *Inception-v3*.

4.5.3 ResNet18

A arquitetura *ResNet18* implementada (Figura 4.9) contempla quatro primeiras camadas, uma de convolução (*Conv2d*), outra para a normalização de lotes (*BatchNorm2d*), uma função de ativação (ReLU) e um agrupamento máximo (*MaxPool2d*). De seguida seguem-se quatro módulos *sequential*, onde cada um destes possuí dois módulos denominados por *BasicBlock*. Cada um dos *BasicBlock* é constituído por duas camadas de convolução, uma de normalização de lotes entre as de convolução e, por fim, uma camada de ligação ao próximo módulo. Após o último módulo *sequential* é utilizada uma camada de agrupamento médio (*avg_pool2d*) e uma última camada FC linear. Nesta arquitetura também foi implementada apenas a técnica de ajuste fino. À semelhança da implementação anterior efetuada, as métricas de desempenho, funções de cálculo das perdas e a função de otimização, foram as mesmas.

ResNet18

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
ReLU-3	[-1, 64, 112, 112]	0
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	36,864
BatchNorm2d-6	[-1, 64, 56, 56]	128
ReLU-7	[-1, 64, 56, 56]	0
Conv2d-8	[-1, 64, 56, 56]	36,864
BatchNorm2d-9	[-1, 64, 56, 56]	128
ReLU-10	[-1, 64, 56, 56]	0
BasicBlock-11	[-1, 64, 56, 56]	0
Conv2d-12	[-1, 64, 56, 56]	36,864
BatchNorm2d-13	[-1, 64, 56, 56]	128
ReLU-14	[-1, 64, 56, 56]	0
Conv2d-15	[-1, 64, 56, 56]	36,864
BatchNorm2d-16	[-1, 64, 56, 56]	128
ReLU-17	[-1, 64, 56, 56]	0
BasicBlock-18	[-1, 64, 56, 56]	0
Conv2d-19	[-1, 128, 28, 28]	73,728
BatchNorm2d-20	[-1, 128, 28, 28]	256
ReLU-21	[-1, 128, 28, 28]	0
Conv2d-22	[-1, 128, 28, 28]	147,456
BatchNorm2d-23	[-1, 128, 28, 28]	256
Conv2d-24	[-1, 128, 28, 28]	8,192
BatchNorm2d-25	[-1, 128, 28, 28]	256
ReLU-26	[-1, 128, 28, 28]	0
BasicBlock-27	[-1, 128, 28, 28]	0
Conv2d-28	[-1, 128, 28, 28]	147,456
BatchNorm2d-29	[-1, 128, 28, 28]	256
ReLU-30	[-1, 128, 28, 28]	0
Conv2d-31	[-1, 128, 28, 28]	147,456
BatchNorm2d-32	[-1, 128, 28, 28]	256
ReLU-33	[-1, 128, 28, 28]	0
BasicBlock-34	[-1, 128, 28, 28]	0
Conv2d-35	[-1, 256, 14, 14]	294,912
BatchNorm2d-36	[-1, 256, 14, 14]	512
ReLU-37	[-1, 256, 14, 14]	0
Conv2d-38	[-1, 256, 14, 14]	589,824
BatchNorm2d-39	[-1, 256, 14, 14]	512
Conv2d-40	[-1, 256, 14, 14]	32,768
BatchNorm2d-41	[-1, 256, 14, 14]	512
ReLU-42	[-1, 256, 14, 14]	0
BasicBlock-43	[-1, 256, 14, 14]	0
Conv2d-44	[-1, 256, 14, 14]	589,824
BatchNorm2d-45	[-1, 256, 14, 14]	512
ReLU-46	[-1, 256, 14, 14]	0
Conv2d-47	[-1, 256, 14, 14]	589,824
BatchNorm2d-48	[-1, 256, 14, 14]	512
ReLU-49	[-1, 256, 14, 14]	0
BasicBlock-50	[-1, 256, 14, 14]	0
Conv2d-51	[-1, 512, 7, 7]	1,179,648
BatchNorm2d-52	[-1, 512, 7, 7]	1,024
ReLU-53	[-1, 512, 7, 7]	0
Conv2d-54	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-55	[-1, 512, 7, 7]	1,024
Conv2d-56	[-1, 512, 7, 7]	131,072
BatchNorm2d-57	[-1, 512, 7, 7]	1,024
ReLU-58	[-1, 512, 7, 7]	0
BasicBlock-59	[-1, 512, 7, 7]	0
Conv2d-60	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-61	[-1, 512, 7, 7]	1,024
ReLU-62	[-1, 512, 7, 7]	0
Conv2d-63	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-64	[-1, 512, 7, 7]	1,024
ReLU-65	[-1, 512, 7, 7]	0
BasicBlock-66	[-1, 512, 7, 7]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 1000]	513,000

Total params: 11,689,512
Trainable params: 11,689,512
Non-trainable params: 0

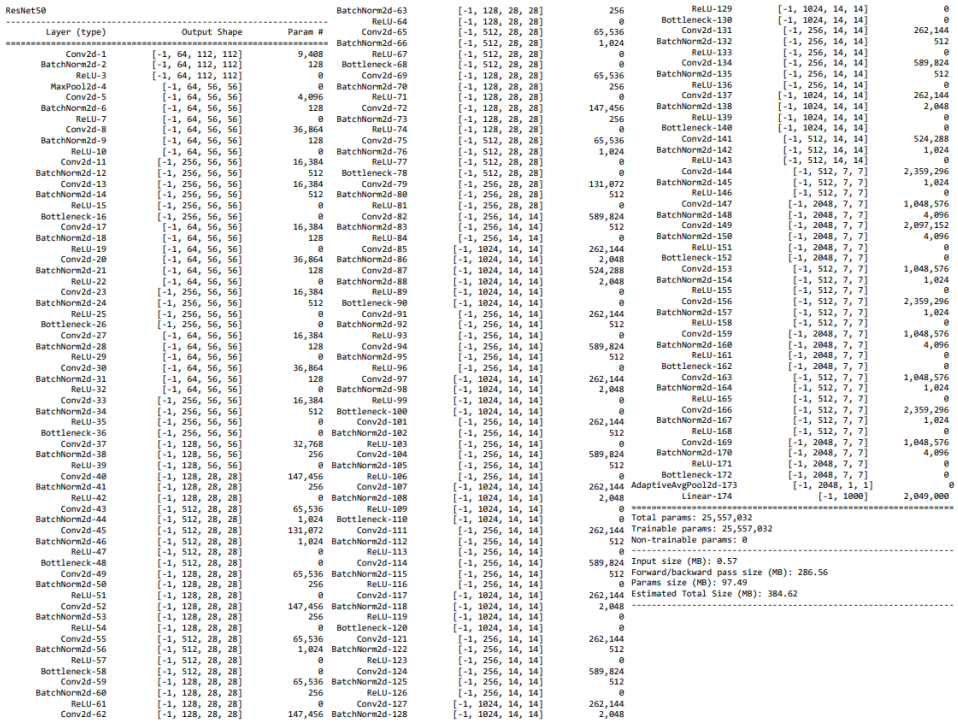
Input size (MB): 0.57
Forward/backward pass size (MB): 62.79
Params size (MB): 44.59
Estimated Total Size (MB): 107.96

Figura 4.9: Arquitetura *ResNet18*.

4.5.4 ResNet50

A *ResNet50* implementada (Figura 4.10), contempla blocos específicos de camadas neste caso denominados por *Bottleneck*. A utilização dos blocos *Bottleneck* permite aumentar a profundida da rede, mantendo contudo, o tamanho dos parâmetros da rede o mais baixo possível [94]. A arquitetura conta com uma primeira camada de convolução, seguida de uma camada de normalização de lotes, uma

função de ativação e um agrupamento máximo. Esta termina com uma camada de agrupamento médio e uma FC linear na saída. A rede contém quatro módulos *sequencial*, tendo o primeiro e o último, dois blocos *Bottleneck*, o segundo módulo *sequencial* quatro blocos *Bottleneck*, e o terceiro módulo *sequencial* seis blocos *Bottleneck*. Por sua vez, cada um dos *Bottleneck* é composto por camadas de convolução, camadas de normalização de lotes intercaladas entre si, e pelas camadas de ativação. O algoritmo de otimização usado neste modelo, durante o processo de treino, foi o SGD. As camadas de ativação na *ResNet50* usam a função de ativação ReLU. À semelhança das implementações anteriores efetuadas, a técnica de *transfer learning*, as métricas de desempenho e as funções de cálculo das perdas, foram as mesmas.



```
ResNet50
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 112, 112]           9,408
       BatchNorm2d-2         [-1, 64, 112, 112]             128
              ReLU-3         [-1, 64, 112, 112]               0
         MaxPool2d-4           [-1, 64, 56, 56]               0
            Conv2d-5           [-1, 64, 56, 56]           4,096
       BatchNorm2d-6           [-1, 64, 56, 56]             128
              ReLU-7           [-1, 64, 56, 56]               0
            Conv2d-8           [-1, 64, 56, 56]          36,864
       BatchNorm2d-9           [-1, 64, 56, 56]             128
             ReLU-10           [-1, 64, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]          16,384
      BatchNorm2d-12          [-1, 256, 56, 56]             512
           Conv2d-13          [-1, 256, 56, 56]          16,384
      BatchNorm2d-14          [-1, 256, 56, 56]             512
             ReLU-15          [-1, 256, 56, 56]               0
       Bottleneck-16          [-1, 256, 56, 56]               0
           Conv2d-17           [-1, 64, 56, 56]          16,384
      BatchNorm2d-18           [-1, 64, 56, 56]             128
             ReLU-19           [-1, 64, 56, 56]               0
           Conv2d-20           [-1, 64, 56, 56]          36,864
      BatchNorm2d-21           [-1, 64, 56, 56]             128
             ReLU-22           [-1, 64, 56, 56]               0
           Conv2d-23          [-1, 256, 56, 56]          16,384
      BatchNorm2d-24          [-1, 256, 56, 56]             512
             ReLU-25          [-1, 256, 56, 56]               0
       Bottleneck-26          [-1, 256, 56, 56]               0
           Conv2d-27           [-1, 64, 56, 56]          16,384
      BatchNorm2d-28           [-1, 64, 56, 56]             128
             ReLU-29           [-1, 64, 56, 56]               0
           Conv2d-30           [-1, 64, 56, 56]          36,864
      BatchNorm2d-31           [-1, 64, 56, 56]             128
             ReLU-32           [-1, 64, 56, 56]               0
           Conv2d-33          [-1, 256, 56, 56]          16,384
      BatchNorm2d-34          [-1, 256, 56, 56]             512
             ReLU-35          [-1, 256, 56, 56]               0
       Bottleneck-36          [-1, 256, 56, 56]               0
           Conv2d-37          [-1, 128, 56, 56]          32,768
      BatchNorm2d-38          [-1, 128, 56, 56]             256
             ReLU-39          [-1, 128, 56, 56]               0
           Conv2d-40          [-1, 128, 28, 28]         147,456
      BatchNorm2d-41          [-1, 128, 28, 28]             256
             ReLU-42          [-1, 128, 28, 28]               0
           Conv2d-43          [-1, 512, 28, 28]          65,536
      BatchNorm2d-44          [-1, 512, 28, 28]           1,024
           Conv2d-45          [-1, 512, 28, 28]         131,072
      BatchNorm2d-46          [-1, 512, 28, 28]           1,024
             ReLU-47          [-1, 512, 28, 28]               0
       Bottleneck-48          [-1, 512, 28, 28]               0
           Conv2d-49          [-1, 128, 28, 28]          65,536
      BatchNorm2d-50          [-1, 128, 28, 28]             256
             ReLU-51          [-1, 128, 28, 28]               0
           Conv2d-52          [-1, 128, 28, 28]         147,456
      BatchNorm2d-53          [-1, 128, 28, 28]             256
             ReLU-54          [-1, 128, 28, 28]               0
           Conv2d-55          [-1, 512, 28, 28]          65,536
      BatchNorm2d-56          [-1, 512, 28, 28]           1,024
             ReLU-57          [-1, 512, 28, 28]               0
       Bottleneck-58          [-1, 512, 28, 28]               0
           Conv2d-59          [-1, 128, 28, 28]          65,536
      BatchNorm2d-60          [-1, 128, 28, 28]             256
             ReLU-61          [-1, 128, 28, 28]               0
           Conv2d-62          [-1, 128, 28, 28]         147,456
      BatchNorm2d-63          [-1, 128, 28, 28]             256
             ReLU-64          [-1, 128, 28, 28]               0
           Conv2d-65          [-1, 512, 28, 28]          65,536
      BatchNorm2d-66          [-1, 512, 28, 28]           1,024
             ReLU-67          [-1, 512, 28, 28]               0
       Bottleneck-68          [-1, 512, 28, 28]               0
           Conv2d-69          [-1, 128, 28, 28]          65,536
      BatchNorm2d-70          [-1, 128, 28, 28]             256
             ReLU-71          [-1, 128, 28, 28]               0
           Conv2d-72          [-1, 128, 28, 28]         147,456
      BatchNorm2d-73          [-1, 128, 28, 28]             256
             ReLU-74          [-1, 128, 28, 28]               0
           Conv2d-75          [-1, 512, 28, 28]          65,536
      BatchNorm2d-76          [-1, 512, 28, 28]           1,024
             ReLU-77          [-1, 512, 28, 28]               0
       Bottleneck-78          [-1, 512, 28, 28]               0
           Conv2d-79          [-1, 256, 28, 28]         131,072
      BatchNorm2d-80          [-1, 256, 28, 28]             512
             ReLU-81          [-1, 256, 28, 28]               0
           Conv2d-82          [-1, 256, 14, 14]         589,824
      BatchNorm2d-83          [-1, 256, 14, 14]             512
             ReLU-84          [-1, 256, 14, 14]               0
           Conv2d-85         [-1, 1024, 14, 14]         262,144
      BatchNorm2d-86         [-1, 1024, 14, 14]           2,048
           Conv2d-87         [-1, 1024, 14, 14]         524,288
      BatchNorm2d-88         [-1, 1024, 14, 14]           2,048
             ReLU-89         [-1, 1024, 14, 14]               0
       Bottleneck-90         [-1, 1024, 14, 14]               0
           Conv2d-91          [-1, 256, 14, 14]         262,144
      BatchNorm2d-92          [-1, 256, 14, 14]             512
             ReLU-93          [-1, 256, 14, 14]               0
           Conv2d-94          [-1, 256, 14, 14]         589,824
      BatchNorm2d-95          [-1, 256, 14, 14]             512
             ReLU-96          [-1, 256, 14, 14]               0
           Conv2d-97         [-1, 1024, 14, 14]         262,144
      BatchNorm2d-98         [-1, 1024, 14, 14]           2,048
             ReLU-99         [-1, 1024, 14, 14]               0
      Bottleneck-100         [-1, 1024, 14, 14]               0
          Conv2d-101          [-1, 256, 14, 14]         262,144
     BatchNorm2d-102          [-1, 256, 14, 14]             512
            ReLU-103          [-1, 256, 14, 14]               0
          Conv2d-104          [-1, 256, 14, 14]         589,824
     BatchNorm2d-105          [-1, 256, 14, 14]             512
            ReLU-106          [-1, 256, 14, 14]               0
          Conv2d-107         [-1, 1024, 14, 14]         262,144
     BatchNorm2d-108         [-1, 1024, 14, 14]           2,048
            ReLU-109         [-1, 1024, 14, 14]               0
      Bottleneck-110         [-1, 1024, 14, 14]               0
          Conv2d-111          [-1, 256, 14, 14]         262,144
     BatchNorm2d-112          [-1, 256, 14, 14]             512
            ReLU-113          [-1, 256, 14, 14]               0
          Conv2d-114          [-1, 256, 14, 14]         589,824
     BatchNorm2d-115          [-1, 256, 14, 14]             512
            ReLU-116          [-1, 256, 14, 14]               0
          Conv2d-117         [-1, 1024, 14, 14]         262,144
     BatchNorm2d-118         [-1, 1024, 14, 14]           2,048
            ReLU-119         [-1, 1024, 14, 14]               0
      Bottleneck-120         [-1, 1024, 14, 14]               0
          Conv2d-121          [-1, 256, 14, 14]         262,144
     BatchNorm2d-122          [-1, 256, 14, 14]             512
            ReLU-123          [-1, 256, 14, 14]               0
          Conv2d-124          [-1, 256, 14, 14]         589,824
     BatchNorm2d-125          [-1, 256, 14, 14]             512
            ReLU-126          [-1, 256, 14, 14]               0
          Conv2d-127         [-1, 1024, 14, 14]         262,144
     BatchNorm2d-128         [-1, 1024, 14, 14]           2,048
            ReLU-129         [-1, 1024, 14, 14]               0
      Bottleneck-130         [-1, 1024, 14, 14]               0
          Conv2d-131          [-1, 256, 14, 14]         262,144
     BatchNorm2d-132          [-1, 256, 14, 14]             512
            ReLU-133          [-1, 256, 14, 14]               0
          Conv2d-134          [-1, 256, 14, 14]         589,824
     BatchNorm2d-135          [-1, 256, 14, 14]             512
            ReLU-136          [-1, 256, 14, 14]               0
          Conv2d-137         [-1, 1024, 14, 14]         262,144
     BatchNorm2d-138         [-1, 1024, 14, 14]           2,048
            ReLU-139         [-1, 1024, 14, 14]               0
      Bottleneck-140         [-1, 1024, 14, 14]               0
          Conv2d-141          [-1, 512, 14, 14]         524,288
     BatchNorm2d-142          [-1, 512, 14, 14]           1,024
            ReLU-143          [-1, 512, 14, 14]               0
          Conv2d-144            [-1, 512, 7, 7]       2,359,296
     BatchNorm2d-145            [-1, 512, 7, 7]           1,024
            ReLU-146            [-1, 512, 7, 7]               0
          Conv2d-147           [-1, 2048, 7, 7]       1,048,576
     BatchNorm2d-148           [-1, 2048, 7, 7]           4,096
          Conv2d-149            [-1, 512, 7, 7]       2,097,152
     BatchNorm2d-150            [-1, 512, 7, 7]           4,096
            ReLU-151            [-1, 512, 7, 7]               0
      Bottleneck-152           [-1, 2048, 7, 7]               0
          Conv2d-153            [-1, 512, 7, 7]       1,048,576
     BatchNorm2d-154            [-1, 512, 7, 7]           1,024
            ReLU-155            [-1, 512, 7, 7]               0
          Conv2d-156            [-1, 512, 7, 7]       2,359,296
     BatchNorm2d-157            [-1, 512, 7, 7]           1,024
            ReLU-158            [-1, 512, 7, 7]               0
          Conv2d-159           [-1, 2048, 7, 7]       1,048,576
     BatchNorm2d-160           [-1, 2048, 7, 7]           4,096
            ReLU-161           [-1, 2048, 7, 7]               0
      Bottleneck-162           [-1, 2048, 7, 7]               0
          Conv2d-163            [-1, 512, 7, 7]       1,048,576
     BatchNorm2d-164            [-1, 512, 7, 7]           1,024
            ReLU-165            [-1, 512, 7, 7]               0
          Conv2d-166            [-1, 512, 7, 7]       2,359,296
     BatchNorm2d-167            [-1, 512, 7, 7]           1,024
            ReLU-168            [-1, 512, 7, 7]               0
          Conv2d-169           [-1, 2048, 7, 7]       1,048,576
     BatchNorm2d-170           [-1, 2048, 7, 7]           4,096
            ReLU-171           [-1, 2048, 7, 7]               0
      Bottleneck-172           [-1, 2048, 7, 7]               0
AdaptiveAvgPool2d-173           [-1, 2048, 1, 1]               0
          Linear-174                 [-1, 1000]       2,049,000
================================================================
Total params: 25,557,032
Trainable params: 25,557,032
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 286.56
Params size (MB): 97.49
Estimated Total Size (MB): 384.62
----------------------------------------------------------------
```

Figura 4.10: Arquitetura *ResNet50*.

4.5.5 ResNet152

A *ResNet152* implementada (Figura 4.11), contempla blocos específicos de camadas denominados por *Bottleneck*, tal como a *ResNet50*. A arquitetura conta com uma primeira camada de convolução, seguida de uma camada de normalização de lotes, uma função de ativação e um agrupamento máximo. Esta termina com uma camada de agrupamento médio e uma FC linear na saída. A rede contém quatro módulos *sequencial*, tendo o primeiro e o último, três blocos *Bottleneck*, o segundo módulo *sequencial* oito blocos *Bottleneck*, e o terceiro módulo *sequencial* 36 blocos *Bottleneck*. Por sua vez, cada um dos *Bottleneck* é composto por camadas de convolução, camadas de normalização de lotes intercaladas entre si, e pelas camadas de

ativação. O algoritmo de otimização usado neste modelo, durante o processo de treino, foi o SGD. As camadas de ativação na *ResNet152* usam a função de ativação ReLU. À semelhança das implementações anteriores efetuadas, a técnica de *transfer learning*, as métricas de desempenho e as funções de cálculo das perdas, foram as mesmas.

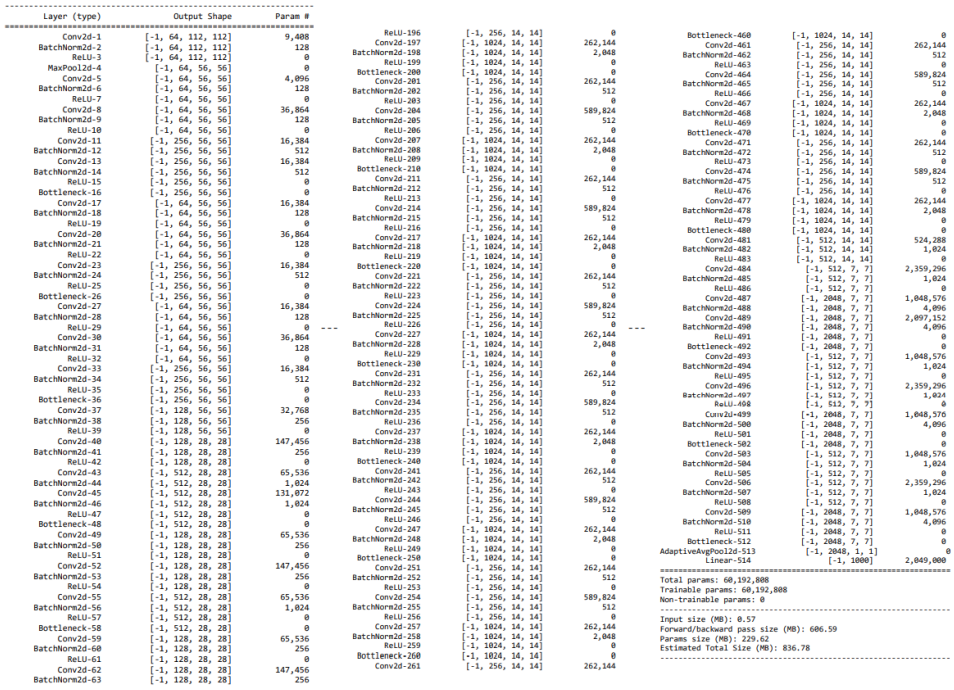


Figura 4.11: Arquitetura *ResNet152*.

4.5.6 VGG16

A arquitetura da *VGG16* implementada pode ser analisada na Figura 4.12. Esta contempla um primeiro módulo *sequential*, constituído por um conjunto de várias camadas de convolução, ativação (ReLU) e de agrupamento máximo (para diminuir o espaço dimensional), e um segundo módulo constituído pelas camadas FC, ativação, e camadas de operação de *Dropout*. Entre os dois módulos está presente uma camada de ligação do tipo *pooling* médio adaptativo. O algoritmo de otimização usado neste modelo, durante o processo de treino, foi o SGD. À semelhança das implementações anteriores efetuadas, a técnica de *transfer learning*, as métricas de desempenho, funções de cálculo das perdas e a função de otimização, foram as mesmas.

```
VGG16
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 299, 299]           1,792
              ReLU-2         [-1, 64, 299, 299]               0
            Conv2d-3         [-1, 64, 299, 299]          36,928
              ReLU-4         [-1, 64, 299, 299]               0
         MaxPool2d-5         [-1, 64, 149, 149]               0
            Conv2d-6        [-1, 128, 149, 149]          73,856
              ReLU-7        [-1, 128, 149, 149]               0
            Conv2d-8        [-1, 128, 149, 149]         147,584
              ReLU-9        [-1, 128, 149, 149]               0
        MaxPool2d-10          [-1, 128, 74, 74]               0
           Conv2d-11          [-1, 256, 74, 74]         295,168
             ReLU-12          [-1, 256, 74, 74]               0
           Conv2d-13          [-1, 256, 74, 74]         590,080
             ReLU-14          [-1, 256, 74, 74]               0
           Conv2d-15          [-1, 256, 74, 74]         590,080
             ReLU-16          [-1, 256, 74, 74]               0
        MaxPool2d-17          [-1, 256, 37, 37]               0
           Conv2d-18          [-1, 512, 37, 37]       1,180,160
             ReLU-19          [-1, 512, 37, 37]               0
           Conv2d-20          [-1, 512, 37, 37]       2,359,808
             ReLU-21          [-1, 512, 37, 37]               0
           Conv2d-22          [-1, 512, 37, 37]       2,359,808
             ReLU-23          [-1, 512, 37, 37]               0
        MaxPool2d-24          [-1, 512, 18, 18]               0
           Conv2d-25          [-1, 512, 18, 18]       2,359,808
             ReLU-26          [-1, 512, 18, 18]               0
           Conv2d-27          [-1, 512, 18, 18]       2,359,808
             ReLU-28          [-1, 512, 18, 18]               0
           Conv2d-29          [-1, 512, 18, 18]       2,359,808
             ReLU-30          [-1, 512, 18, 18]               0
        MaxPool2d-31            [-1, 512, 9, 9]               0
AdaptiveAvgPool2d-32            [-1, 512, 7, 7]               0
           Linear-33                 [-1, 4096]     102,764,544
             ReLU-34                 [-1, 4096]               0
          Dropout-35                 [-1, 4096]               0
           Linear-36                 [-1, 4096]      16,781,312
             ReLU-37                 [-1, 4096]               0
          Dropout-38                 [-1, 4096]               0
           Linear-39                 [-1, 1000]       4,097,000
================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.02
Forward/backward pass size (MB): 386.02
Params size (MB): 527.79
Estimated Total Size (MB): 914.83
----------------------------------------------------------------
```

Figura 4.12: Arquitetura *VGG16*.

4.5.7 VGG19

A arquitetura da *VGG19* implementada pode ser analisada na Figura 4.13. Esta contempla um primeiro módulo *sequential*, constituído por um conjunto de várias camadas de convolução, ativação (ReLU) e de agrupamento máximo, e um segundo módulo constituído pelas camadas FC, ativação, e camadas de operação de *Dropout*. Tal como na *VGG16*, entre os dois módulos está presente uma camada de ligação do tipo *pooling* médio adaptativo. O algoritmo de otimização usado neste modelo, durante o processo de treino, também foi o SGD. À semelhança das implementações anteriores efetuadas, a técnica de *transfer learning*, as métricas de desempenho, funções de cálculo das perdas e a função de otimização, foram as mesmas.

```
VGG19
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 299, 299]           1,792
              ReLU-2         [-1, 64, 299, 299]               0
            Conv2d-3         [-1, 64, 299, 299]          36,928
              ReLU-4         [-1, 64, 299, 299]               0
         MaxPool2d-5         [-1, 64, 149, 149]               0
            Conv2d-6        [-1, 128, 149, 149]          73,856
              ReLU-7        [-1, 128, 149, 149]               0
            Conv2d-8        [-1, 128, 149, 149]         147,584
              ReLU-9        [-1, 128, 149, 149]               0
        MaxPool2d-10          [-1, 128, 74, 74]               0
           Conv2d-11          [-1, 256, 74, 74]         295,168
             ReLU-12          [-1, 256, 74, 74]               0
           Conv2d-13          [-1, 256, 74, 74]         590,080
             ReLU-14          [-1, 256, 74, 74]               0
           Conv2d-15          [-1, 256, 74, 74]         590,080
             ReLU-16          [-1, 256, 74, 74]               0
           Conv2d-17          [-1, 256, 74, 74]         590,080
             ReLU-18          [-1, 256, 74, 74]               0
        MaxPool2d-19          [-1, 256, 37, 37]               0
           Conv2d-20          [-1, 512, 37, 37]       1,180,160
             ReLU-21          [-1, 512, 37, 37]               0
           Conv2d-22          [-1, 512, 37, 37]       2,359,808
             ReLU-23          [-1, 512, 37, 37]               0
           Conv2d-24          [-1, 512, 37, 37]       2,359,808
             ReLU-25          [-1, 512, 37, 37]               0
           Conv2d-26          [-1, 512, 37, 37]       2,359,808
             ReLU-27          [-1, 512, 37, 37]               0
        MaxPool2d-28          [-1, 512, 18, 18]               0
           Conv2d-29          [-1, 512, 18, 18]       2,359,808
             ReLU-30          [-1, 512, 18, 18]               0
           Conv2d-31          [-1, 512, 18, 18]       2,359,808
             ReLU-32          [-1, 512, 18, 18]               0
           Conv2d-33          [-1, 512, 18, 18]       2,359,808
             ReLU-34          [-1, 512, 18, 18]               0
           Conv2d-35          [-1, 512, 18, 18]       2,359,808
             ReLU-36          [-1, 512, 18, 18]               0
        MaxPool2d-37            [-1, 512, 9, 9]               0
AdaptiveAvgPool2d-38            [-1, 512, 7, 7]               0
           Linear-39                 [-1, 4096]     102,764,544
             ReLU-40                 [-1, 4096]               0
          Dropout-41                 [-1, 4096]               0
           Linear-42                 [-1, 4096]      16,781,312
             ReLU-43                 [-1, 4096]               0
          Dropout-44                 [-1, 4096]               0
           Linear-45                 [-1, 1000]       4,097,000
================================================================
Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.02
Forward/backward pass size (MB): 420.63
Params size (MB): 548.05
Estimated Total Size (MB): 969.70
----------------------------------------------------------------
```

Figura 4.13: Arquitetura *VGG19*.

4.5.8 EfficientNet-B0

A arquitetura *EfficientNet-B0* implementada pode ser analisada (em parte) na Figura 4.14. A *EfficientNet-B0* contempla um módulo *sequential* que conta com um módulo de *ConvNormActivation* que possuí uma camada de convolução, uma normalização de lote e uma função de ativação *Sigmoid Linear Unit* (SiLU). De seguida tem-se sete outros módulos *sequential* em que o primeiro contempla um módulo *MBConv* com um módulo *sequential* que possuí dois *ConvNormActivation* e um *SqueezeExcitation*. O bloco *SqueezeExcitation* conta com uma camada de agrupamento

médio adaptativo, duas camadas de convolução e uma função *SiLU* e uma *Sigmoid*. Ainda dentro do primeiro *sequential* tem-se uma *StochasticDepth* que procura diminuir a profundidade de uma rede durante o treino, mantendo-a inalterada durante o teste. Nos seguintes seis módulos *sequential* tem-se, no (2-3) dois módulos *MBConv*, no (4-5) três módulos *MBConv*, no (6) quatro módulos *MBConv* e no último módulo *sequential* tem-se um módulo *MBConv*. De seguida tem-se ainda um *ConvNormActivation*, um agrupamento médio adaptativo e um último *sequential* com um *Dropout* e um FC linear. Na implementação da *EfficientNet* foi utilizada no cálculo das perdas a entropia cruzada, otimização com recurso à função SGD e *Adam* e a exatidão (ACC) e (AUC) como medidas de desempenho. Nesta arquitetura também foram implementadas as duas técnicas de *transfer learning*.

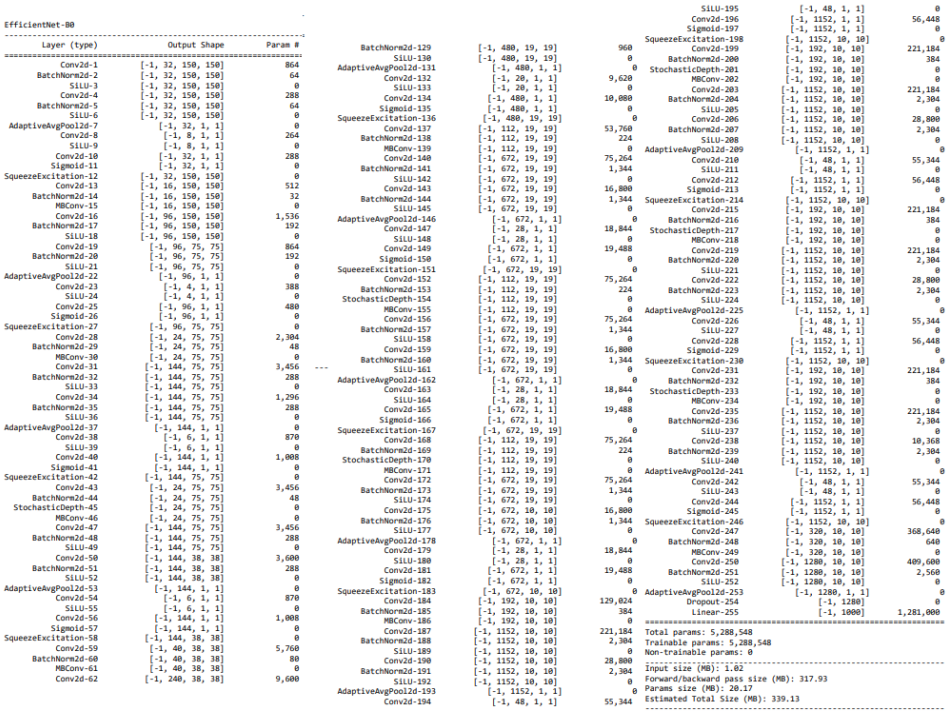


Figura 4.14: Arquitetura *EfficientNet-B0*.

4.5.9 EfficientNet-B7

A arquitetura *EfficientNet-B7* implementada pode ser analisada (em parte) na Figura 4.15. A *EfficientNet-B7* contempla um módulo *sequential* que conta com um módulo de *ConvNormActivation* onde este possuí, uma camada de convolução, uma camada de normalização de lote e uma função de ativação *SiLU*. De seguida tem-se sete outros módulos *sequential* em que o primeiro contempla quatro módulos *MBConv* com um módulo *sequential* que possuí dois *ConvNormActivation* e um *SqueezeExcitation*. O bloco *SqueezeExcitation* conta com uma camada de agrupamento médio adaptativo, duas camadas de convolução e uma função *SiLU* e uma *Sigmoid*.

Ainda dentro do primeiro *sequential* tem-se uma *StochasticDepth*. Nos seguintes seis módulos *sequential* tem-se, no (2-3) sete módulos *MBConv*, no (4-5) dez módulos *MBConv*, no (6) treze módulos *MBConv* e no último módulo *sequential* tem-se quatro módulos *MBConv*. De seguida tem-se ainda um *ConvNormActivation*, um agrupamento médio adaptativo e um último *sequential* com uma camada *Dropout* e um FC linear. A função utilizada no cálculo das perdas foi a entropia cruzada, a otimização com recurso à função SGD e a exatidão (ACC) e (AUC) como medidas de desempenho. Nesta implementação apenas foi utilizada a técnica de ajuste fino.

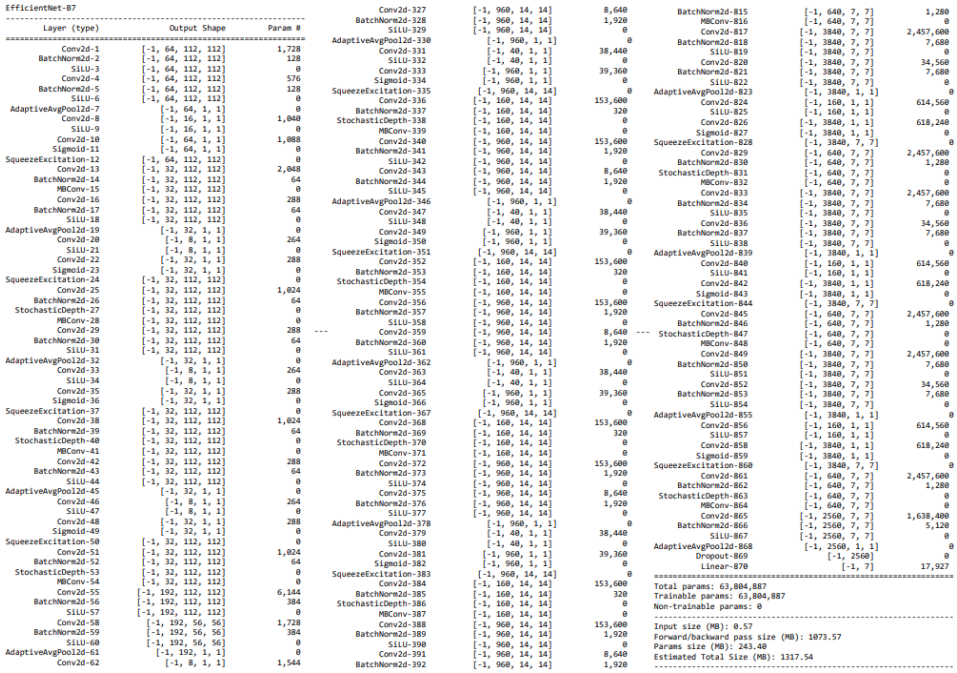


Figura 4.15: Arquitetura *EfficientNet-B7*.

4.5.10 ConvNeXT-Small

A arquitetura *ConvNeXT-Small* implementada pode ser analisada (em parte) na Figura 4.16. A *ConvNeXT-Small* contempla um módulo *sequential* que conta com um módulo de *ConvNormActivation* onde este possuí, uma camada de convolução, uma camada *LayerNorm2d*. De seguida tem-se sete outros módulos *sequential* em que o primeiro contempla três módulos *CNBlock* com um módulo *sequential* que conta com uma camada de convolução, uma camada *Permute* que altera as dimensões da entrada de acordo com um determinado padrão, uma *LayerNorm*, uma FC linear, uma função de ativação *Gaussian Error Linear Unit* (GELU), uma outra FC linear e *Permute*. Por fim, ainda dentro da *sequential* tem-se uma *StochasticDepth*. Nos módulos *sequential* (2, 4 e 6) tem-se apenas uma camada *LayerNorm2d* e uma camada de convolução. Nos módulos *sequential* (3-7) tem-se três módulos *CNBlock*,

no módulo *sequential* (5) tem-se 27 módulos *CNBlock*. Por último tem-se um agrupamento médio adaptativo e um último *sequential* com uma camada *LayerNorm2d*, uma camada *Flatten* e uma FC linear. A função utilizada no cálculo das perdas foi a entropia cruzada, a otimização com recurso à função SGD e a exatidão (ACC) e (AUC) como medidas de desempenho. À semelhança da implementação anterior apenas foi utilizada a técnica de ajuste fino.

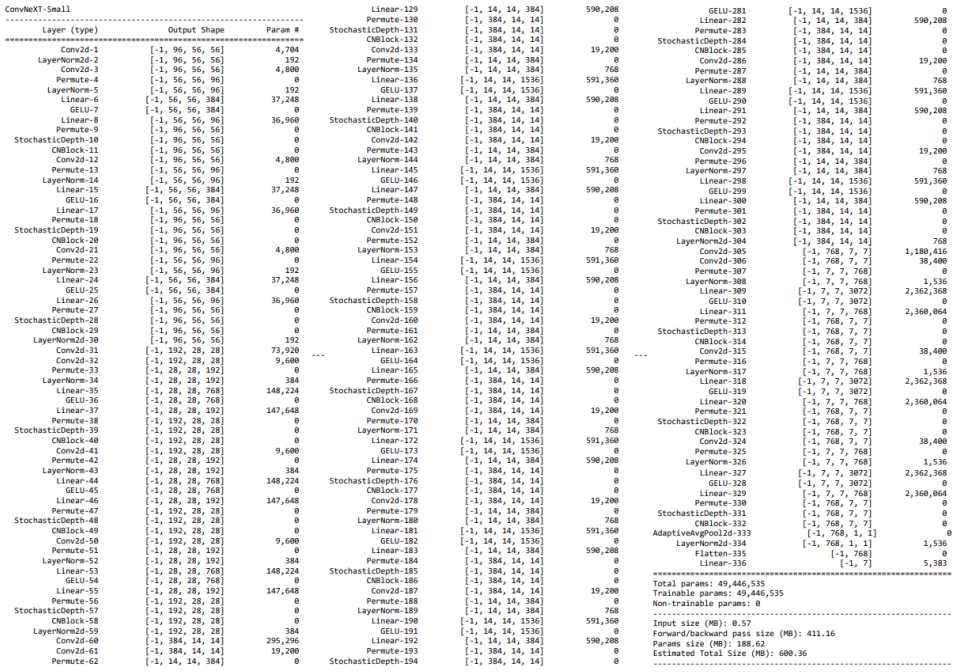


Figura 4.16: Arquitetura *ConvNeXT-Small*.

4.5.11 MobileNet-v2

A arquitetura *MobileNet-v2* implementada pode ser analisada na Figura 4.17. A *MobileNet-v2* contempla um módulo *Sequential* que conta com um módulo de *ConvNormActivation* onde este possuí, uma camada de convolução, uma camada de normalização de lote e ainda uma camada de ativação ReLU. De seguida tem-se dezassete módulos *InvertedResidual* em que o primeiro contempla um módulo *Sequential* seguido de uma camada de convolução e uma camada de normalização de lote. Os restantes dezasseis módulos *InvertedResidual* ainda contam com um módulo *ConvNormActivation* onde este mais uma vez tem uma camada de convolução, uma de normalização de lote e a ativação ReLU. Por último tem-se um módulo *ConvNormActivation* e um último *Sequential* com uma camada *Dropout* e uma FC linear. A função utilizada no cálculo das perdas foi a entropia cruzada, a otimização com recurso à função SGD e a exatidão (ACC) e (AUC) como medidas de desempenho. À semelhança da implementação anterior apenas foi utilizada a técnica de ajuste fino.

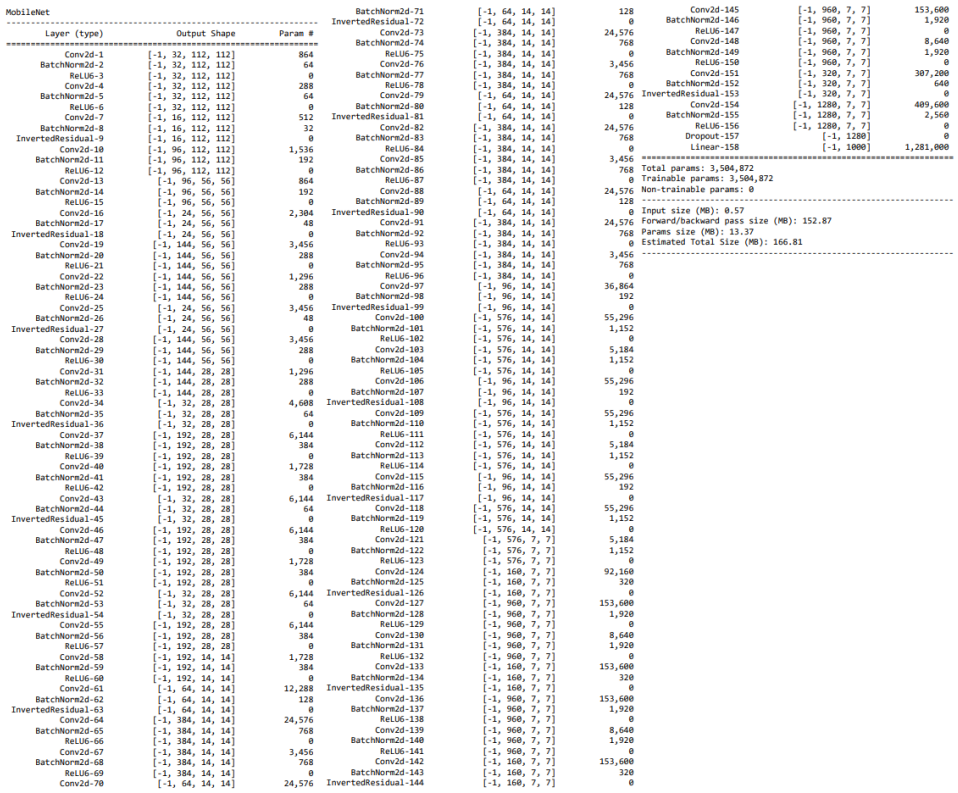


Figura 4.17: Arquitetura *MobileNet-v2*.

4.6 Plataforma de apoio: Demonstração de resultados

Como ferramenta de demonstração de resultados foi desenvolvida uma aplicação *Web*, para testar no conjunto de dados de treino, alguns dos modelos implementados. Para isso, foi utilizada a biblioteca *Streamlit* que oferece um fácil e rápido desenvolvimento de aplicações *Web* em *Python*. Com esta aplicação *Web* é possível testar o modelo, avaliando os seus resultados e ao mesmo tempo os níveis de desempenho, através de imagens guardadas. O processo de treino é muitas vezes demorado, o que não é pretendido para uma aplicação *Web* e como tal foram guardados os modelos durante o processo de aprendizagem para poderem ser utilizados apenas para teste.

Capítulo 5

Resultados e Discussão

Depois de aplicadas as metodologias de pré-processamento de imagem e implementadas as diferentes arquiteturas, os conjuntos de dados foram submetidos ao processo de treino. Neste capítulo serão apresentados e discutidos os resultados obtidos para a classificação de imperfeições de *capply*.

5.1 Resultados obtidos ao longo do processo de treino

Para a classificação de imperfeições de *capply* ao longo do processo de treino, foram criadas e implementadas diferentes configurações. Estas configurações variam entre si no modelo utilizado, nas transformações efetuadas aos dados, nas técnicas de treino dos modelos pré-treinados e na subdivisão do conjunto de dados. Na Tabela 5.1 estão sumarizadas as informações de cada uma das configurações desenvolvidas e testadas. A avaliação do desempenho de cada uma das configurações desenvolvidas teve como base a métrica de exatidão (ACC) e em alguns casos a métrica da área sob a curva ROC (AUC), apesar de também terem sido usadas as métricas de precisão (*precision*), sensibilidade (*recall*) e pontuação F1 (*F1 Score*). Estas foram selecionadas por serem as mais indicadas para este tipo de tarefa. Pelas mesmas razões foram selecionadas as funções utilizadas no cálculo de perdas. A variação de amostras (imagens), que são apresentadas na tabela, deve-se ao facto de que o *dataset* foi sendo composto ao longo de toda a execução desta dissertação.

Tabela 5.1: Configurações aplicadas durante o processo de teste.

Config.	Modelo	Batch	Epochs	Resize	Center Crop	Transf.	Técnicas de treino	LR	Momentum	Otimização	Amostras	Dataset split
1	*Xception*	32	100/ 20	-	299x299		*FE/ FT*	0.001/ 0.00001	-	*Adam*		
2	*ResNet18*					1		0.0001			1044	
3	*ResNet18*		100	256x256			*FT*	0.001				
4	*ResNet18*					2						
5	*ResNet18*		100/ 20	236x236		4	*FE/ FT*	0.001/ 0.00001			1672	
6	*ResNet18*	4			224x224							
7	*ResNet50*			256x256		2					1044	
8	*ResNet50*					5						
9	*ResNet50*			256x480		3					1501	
10	*ResNet50*			236x236		4					1672	
11	*ResNet152*			256x256		3						
12	*ResNet152*	32		300x300	256x256	2					1044	60, 30, 10
13	*ResNet152*			288x288	256x256				0.9	SGD		
14	*ResNet152*			256x256							1501	
15	*ResNet152*			236x236							1672	
16	*VGG16*		100	256x256			*FT*	0.001			1501	
17	*VGG16*	4		236x236	224x224	4					1677	
18	*VGG19*			256x256							1501	
19	*VGG19*			236x236								
20	*Inception_v3*			311x311	299x299						1677	
21	*EfficientNetB0*										1501	
22	*EfficientNetB0*										1677	
23	*EfficientNetB0*					6						
24	*EfficientNetB0*	16				7						
25	*EfficientNetB0*					10					1853	80, 10, 10
26	*EfficientNetB0*		50/ 50	236x236			*FE/ FT*	0.001/ 0.00001	-	*Adam*		60, 30, 10
27	*EfficientNetB0*					4						70, 20, 10
28	*EfficientNetB0*										1935	80, 10, 10
29	*EfficientNetB0*	4			224x224	10						
30	*EfficientNetB7*										1501	
31	*EfficientNetB7*										1677	60, 30, 10
32	*ConvNext-Small*		100	256x256		4	*FT*	0.001	0.9	SGD	1582	
33	*ConvNext-Small*			256x256							1677	
34	*ConvNext-Small*	16				8					1853	
35	*ConvNext-Small*					6					1853	70, 20, 10
36	*ConvNext-Small*			236x236		4						
37	*MobileNet-v2*	4	40			9					1935	80, 10, 10
38	*MobileNet-v2*					10						

Legenda de Transformações:

1. Converter imagem em Tensor, normalização de imagem (média = [0.485, 0.456, 0.406] e desvio = [0.229, 0.224, 0.225]), dimensionamento da imagem, rotações aleatórias e variações horizontais (Transformação típica);
2. Igual à transformação 1 só que também com variações verticais, ajuste automático de contraste e ajuste de nitidez aleatório;
3. Igual à transformação 2, mas com a diferença de que o dimensionamento da imagem é aleatório;
4. Tudo igual à transformação 2 só que com a técnica de *oversampling*;
5. Tudo igual à transformação 2 só que com aumento de dados também aplicado ao conjunto de validação;
6. Tudo igual à transformação 2 mais a técnica de *stratify*;
7. Tudo igual à transformação 5 mais a técnica de *oversampling*;
8. Tudo igual à transformação 5 mais a técnica de *stratify*;
9. Tudo igual à transformação 2 mais as técnicas de *oversampling* e *stratify*;
10. Tudo igual à transformação 5 mais as técnicas de *oversampling* e *stratify*;

Importa referir que em todas as configurações a partir da 2 (Tabela 5.1), foi também usado o *Callback Scheduler* na taxa de aprendizagem, *Learning Rate* (LR), em que este *Callback*, reduz o LR num fator de 0.1 a cada 30 iterações.

Foi ainda utilizado o *Callback ModelCheckPoint* para guardar o modelo e os pesos, quando são obtidos os melhores resultados de desempenho de perdas de validação (*Loss Validation*).

Na Tabela 5.2 estão presentes os resultados obtidos nas diferentes configurações desenvolvidas, para cada uma das classes presentes no conjunto de dados. Estes resultados estão apresentados por ordem crescente de exatidão, *accuracy* (ACC), num gradiente de cores, que varia entre vermelho (valores mais baixos de ACC) e verde (valores mais altos de ACC) (Figura 5.1).

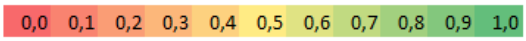


Figura 5.1: Gradiente de cores para apresentação dos valores das métricas.

Tabela 5.2: Resultados obtidos no processo de aprendizagem com as arquiteturas desenvolvidas.

Configuração	Modelo	AUC	ACC	Configuração	Modelo	AUC	ACC
1	*Xception*	-	**0.66**	8	*ResNet50*	-	**0.91**
9	*ResNet50*	-	**0.77**	12	*ResNet152*	-	**0.91**
23	*EfficientNetB0*	**0.958**	**0.77**	14	*ResNet152*	-	**0.91**
26	*EfficientNetB0*	**0.971**	**0.81**	35	*ConvNext-Small*	**0.992**	**0.91**
6	*ResNet18*	**0.973**	**0.82**	17	*VGG16*	**0.993**	**0.91**
32	*ConvNext-Small*	**0.979**	**0.84**	19	*VGG19*	**0.994**	**0.91**
2	*ResNet18*	-	**0.88**	30	*EfficientNetB7*	**0.995**	**0.91**
34	*ConvNext-Small*	**0.988**	**0.88**	28	*EfficientNetB0*	**0.989**	**0.92**
3	*ResNet18*	-	**0.89**	37	*MobileNet-v2*	**0.990**	**0.92**
7	*ResNet50*	-	**0.89**	5	*ResNet18*	**0.992**	**0.92**
13	*ResNet152*	-	**0.89**	10	*ResNet50*	**0.993**	**0.92**
36	*MobileNet-v2*	**0.983**	**0.89**	15	*ResNet152*	**0.994**	**0.92**
16	*VGG16*	-	**0.90**	33	*ConvNext-Small*	**0.994**	**0.92**
18	*VGG19*	-	**0.90**	20	*Inception_v3*	**0.995**	**0.92**
11	*ResNet152*	-	**0.90**	27	*EfficientNetB0*	**0.993**	**0.93**
31	*ConvNext-Small*	**0.985**	**0.90**	22	*EfficientNetB0*	**0.995**	**0.93**
24	*EfficientNetB0*	**0.991**	**0.90**	21	*EfficientNetB0*	**0.997**	**0.93**
29	*EfficientNetB7*	**0.997**	**0.90**	29	*EfficientNetB0*	**0.995**	**0.94**
4	*ResNet18*	-	**0.91**	25	*EfficientNetB0*	**0.989**	**0.95**

Como se observa pela análise dos conjuntos de resultados obtidos, de um modo geral, as configurações sujeitas às técnicas de *feature extraction* e *fine-tuning* (Tabela 5.1), configurações 1, 6 e 26, apresentam valores mais baixos de ACC (Tabela 5.2). Estas três configurações variam entre si no *Batch*, resolução da imagem, transformação aplicada, otimização e quantidade de amostras.

Analisando os modelos *ResNet18*, as configurações 2, 3, 4 e 5 (Tabela 5.1), a configuração que apresenta das quatro, o menor valor de ACC (Tabela 5.2) é a configuração 2 que tem como diferença da configuração 3, apenas a taxa de aprendizagem. Nas configurações 3 e 4 a diferença está na transformação aplicada o que demonstra a importância do aumento de dados num *dataset* pequeno. Por último no modelo *ResNet18*, podemos comparar a configuração 4 e 5, sendo que a 5 é a que apresenta melhor resultado de ACC e isso deve-se principalmente à técnica de *oversampling* aplicado a um *dataset* desequilibrado.

Passando agora a analisar o modelo *ResNet50*, a configuração que apresenta o valor mais baixo de ACC (Tabela 5.2) é a configuração 9 (Tabela 5.1) e isso deve-se

ao facto do dimensionamento de imagem não ser o mais apropriado. Comparando as restantes configurações (7, 8 e 10) (Tabela 5.1) do modelo, as diferenças de ACC são reduzidas, mas mais uma vez a que apresenta melhor resultado, é a configuração 10 com a transformação 4 que possuí a técnica de *oversampling*.

No modelo *ResNet152*, configurações 11, 12, 13, 14 e 15 (Tabela 5.1), a configuração que apresenta o menor resultado de ACC é a 13 (Tabela 5.2). Nesta configuração temos mais uma vez um mau dimensionamento da imagem. Neste modelo é possível verificar a grande importância que tem o dimensionamento das imagens na parte de pré-processamento. Em comparação as configurações 11 e 12, apresentam resultados diferentes e isso deve-se ao tipo de dimensionamento feito nas duas configurações, sendo que na configuração 11 o dimensionamento é aleatório e na 12 é uniforme. A configuração 15 é a que apresenta melhor resultado de ACC, tendo um dimensionamento de imagem que procurou não remover informação importante, sendo este dimensionamento de 236x236 e um corte central de 224x224.

Analisando os modelos *VGG16* e *VGG19* em conjunto, percebe-se mais uma vez a grande importância do pré-processamento dos dados comparando as configurações 16 (VGG16) e 18 (VGG19) com as configurações 17 (VGG16) e 19 (VGG19) (Tabela 5.1).

No modelo *EfficientNetB0*, as configurações 21 e 22 (Tabela 5.1) que podemos desde já comparar e observar que pouca diferença têm nos valores de AUC e ACC, tendo apenas o número de amostras como diferença de configuração. Quanto às configurações 23 e 24 a sua grande diferença é a técnica aplicada aos dados desequilibrados, sendo que a configuração 23 com a técnica de *stratify* foi a que obteve o valor de ACC mais baixo, isto porque, esta técnica por si só não melhora o desempenho. A configuração 25 foi a configuração que melhores resultados obteve tendo por base o uso da função *Adam* na otimização e o uso das duas técnicas (*oversampling* e *stratify*) em simultâneo. A configuração 29 em comparação com a configuração 25, com as mesmas transformações e configurações, apenas com a diferença da função de otimização *SGD*, obteve valores um pouco mais baixos de ACC. Por último, no modelo *EfficientNetB0*, temos as configurações 27 e 28 em que a configuração 27 foi a que obteve melhores resultados de desempenho e isso deve-se à importância na divisão dos conjuntos de dados de treino e validação.

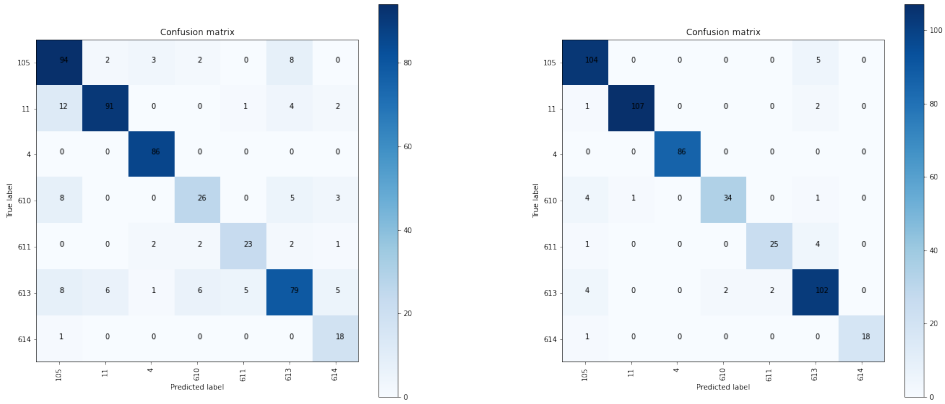
Quanto ao modelo *EfficientNetB7*, as configurações 31 e 31 (Tabela 5.1) pouca variação apresentam nos seus resultados, sendo essa pequena variação refletida pelo aumento de amostras no *dataset*.

Analisando o modelo *ConvNext-Small*, as configurações 32 e 33 (Tabela 5.1) têm como grande diferença o dimensionamento da imagem e isso reflete-se nos resultados de desempenho em que a configuração 33 é a que apresenta os menores valores de AUC e ACC (Tabela 5.2). Nas configurações 34 e 35 existe uma diferença significativa nos resultados quando se aplica um aumento de dados também ao *dataset* de

validação. A configuração 36 comparada com a configuração 34, que varia apenas na técnica de *oversampling*, consegue obter melhores resultados de desempenho.

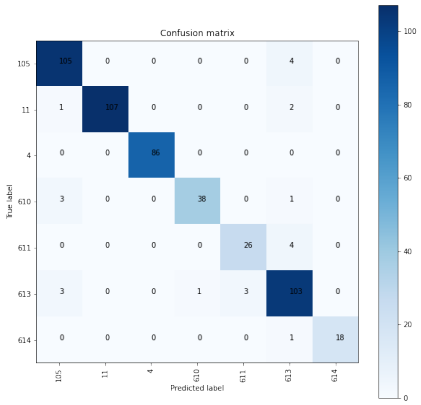
Chegando agora ao modelo *MobileNet-v2* pode-se observar (configurações 37 e 38) (Tabela 5.1), mais uma vez a importância do aumento de dados também aplicado ao conjunto de validação.

De seguida (Figuras 5.2 - 5.4) é possível verificar os resultados de classificação de cada uma das classes, estando presente a relação entre os valores reais (*True label*) e os valores previstos (*Predicted label*), por arquitetura implementada (apenas são apresentados os resultados mais baixos em comparação com os melhores das diferentes configurações por modelo). Deste modo, é possível saber que troca de classificação de classe ocorreu. Por exemplo, para a arquitetura *ResNet18* - Configuração 5 (Figura 5.2b) constata-se que, das 109 amostras de teste da classe 109 (*Capply* OK), 104 apresentaram uma classificação correta enquanto 5 amostras foram classificadas como sendo da classe 613 (Mau enrolamento/distribuição irregular).

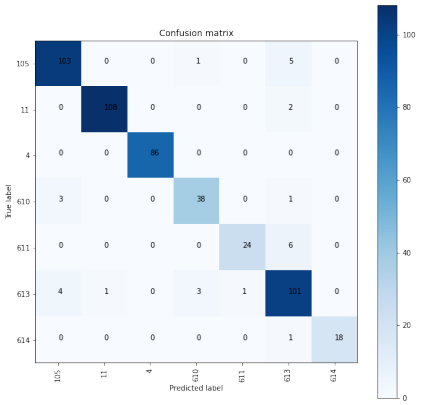


(a) *ResNet18* - Configuração 6 (b) *ResNet18* - Configuração 5

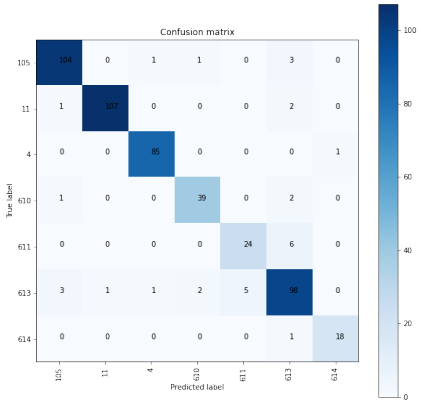
Figura 5.2: Matriz de confusão dos resultados reais *vs* resultados previstos (*ResNet18*).



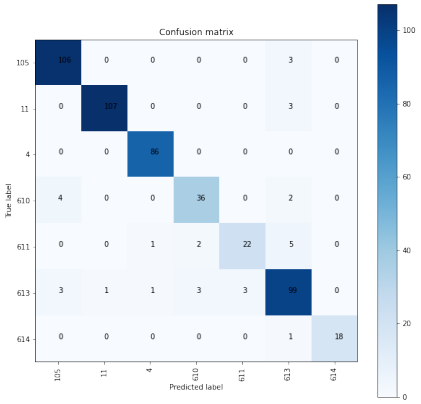
(a) *ResNet50* - Configuração 10



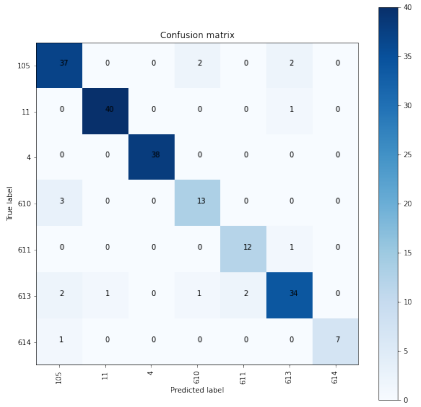
(b) *ResNet152* - Configuração 15



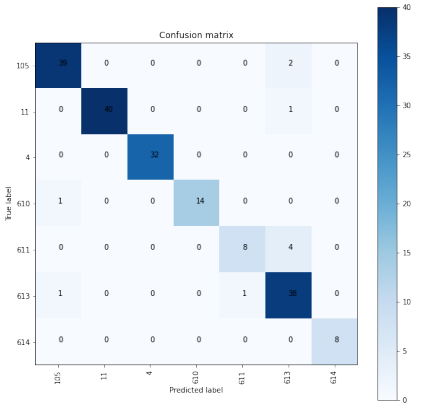
(c) *VGG16* - Configuração 17



(d) *VGG19* - Configuração 19

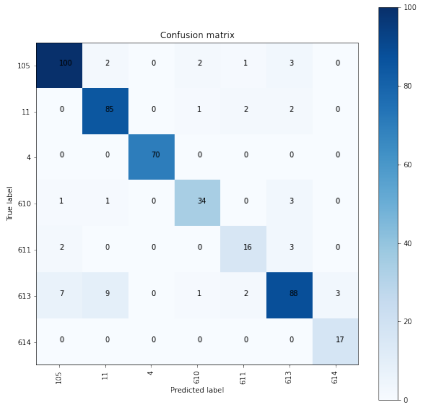


(e) *EfficientNetB0* - Configuração 29

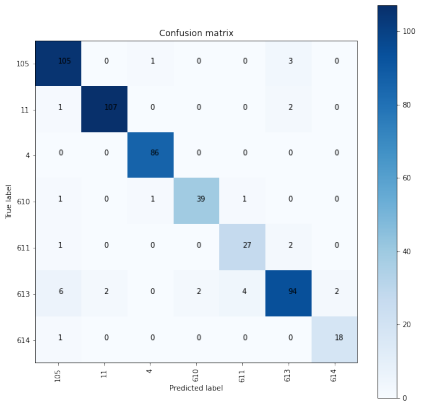


(f) *EfficientNetB0* - Configuração 25

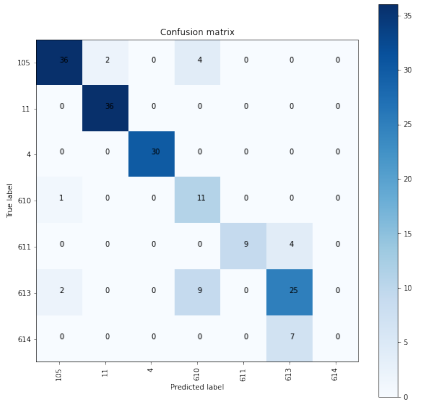
Figura 5.3: Matriz de confusão dos resultados reais *vs* resultados previstos (*ResNet50-152*, *VGG16-19* e *EfficientNetB0*).



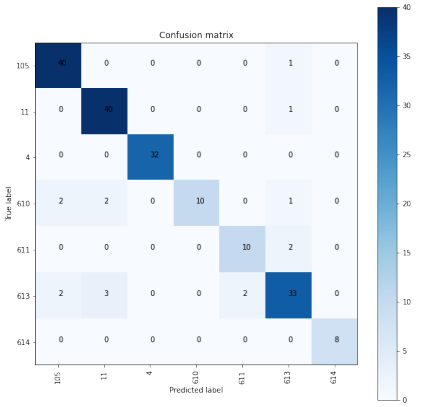
(a) *EfficientNetB7* - Configuração 30



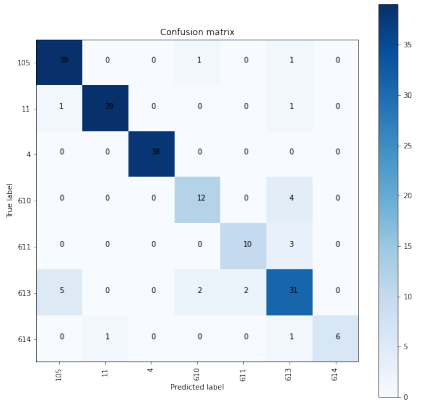
(b) *EfficientNetB7* - Configuração 31



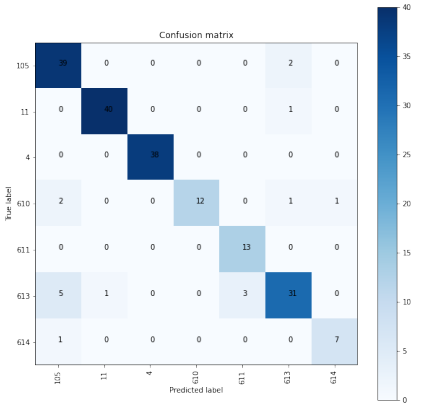
(c) *ConvNext-Small* - Configuração 33



(d) *ConvNext-Small* - Configuração 34



(e) *MobileNet-v2* - Configuração 37



(f) *MobileNet-v2* - Configuração 38

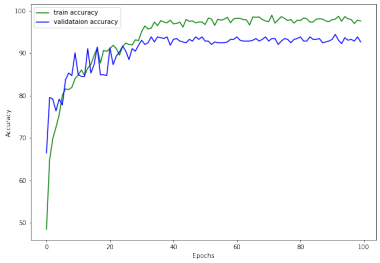
Figura 5.4: Matriz de confusão dos resultados reais *vs* resultados previstos (*EfficientNetB7*, *ConvNext-Small* e *MobileNet-v2*).

Pela análise dos gráficos dos valores de exatidão (ACC) em função do número de iterações (*Epochs*) (Figura 5.5-5.8) e dos gráficos de perdas em função do número de iterações (*Epochs*) obtidos ao longo do processo de treino, verifica-se que no decorrer das iterações no processo de treino (*Train*) há um aumento do valor de ACC, sendo este aumento inversamente proporcional ao valor das perdas, tal como previsto, apesar de que, em alguns casos os valores de validação oscilam ou apresentam um limite quando comparado com os valores de treino.

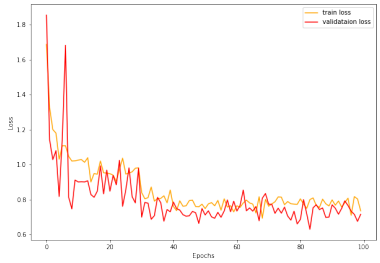
Nas Figuras 5.5c e 5.5d é possível perceber o efeito do *Callback Scheduler* na iteração 30. Tanto os valores de exatidão como de perdas ficam mais estáveis a partir desta iteração.

Pelos gráficos das Figuras 5.6e a 5.6h, podemos concluir que entre as duas arquiteturas (VGG16 e VGG19), não existiram grandes diferenças no processo de aprendizagem.

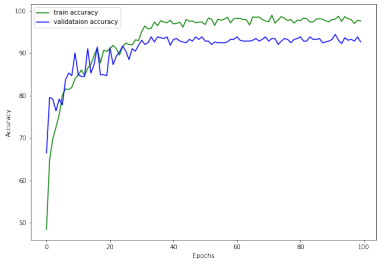
Analisando os gráficos das Figuras 5.7a a 5.7d, percebe-se que a configuração 29 teve um comportamento ao longo de todo o processo de aprendizagem mais estável (função *SGD*) do que a configuração 25 (função *Adam*) e que mesmo assim obteve resultados mais baixos, isto porque a função *Adam - Adaptive Moment Estimation* converge mais rápido e a função *SGD - Stochastic Gradient Descent* generaliza melhor.



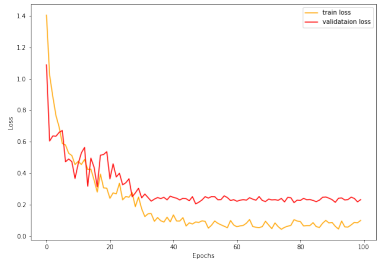
(a) Configuração 6 - Exatidão



(b) Configuração 6 - Perdas

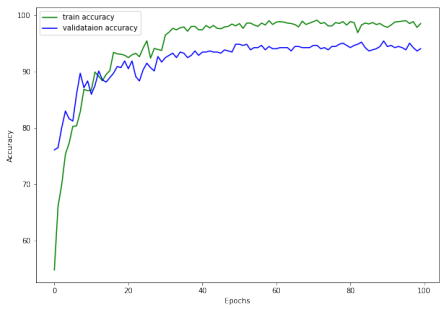


(c) Configuração 5 - Exatidão

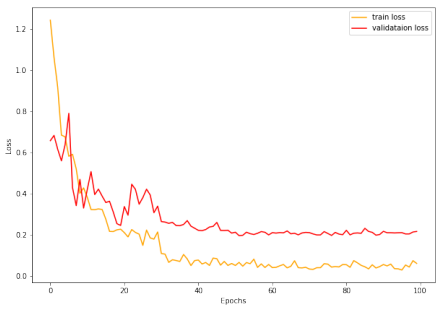


(d) Configuração 5 - Perdas

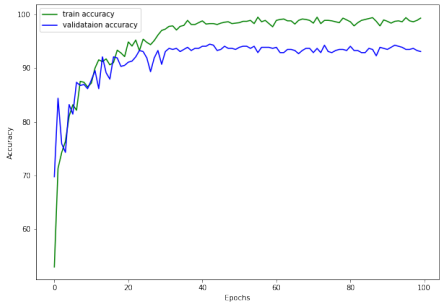
Figura 5.5: Gráficos das métricas obtidas no processo de aprendizagem (*ResNet18*).



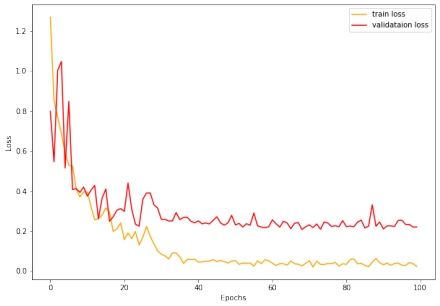
(a) Configuração 10 - Exatidão



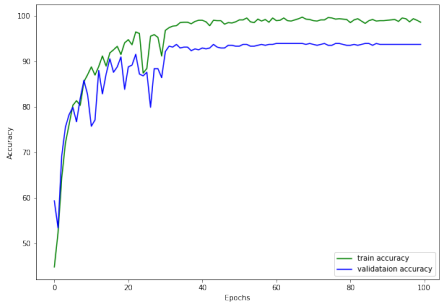
(b) Configuração 10 - Perdas



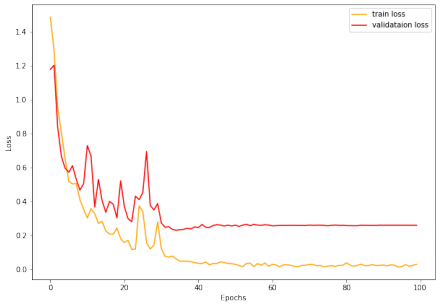
(c) Configuração 15 - Exatidão



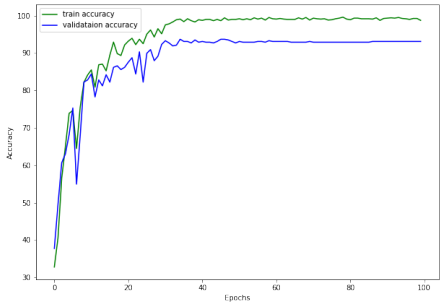
(d) Configuração 15 - Perdas



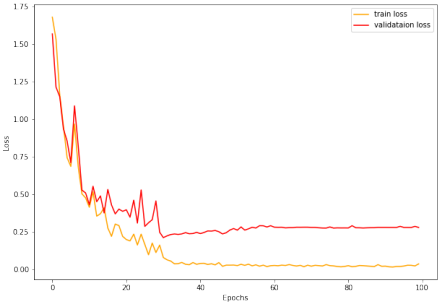
(e) Configuração 17 - Exatidão



(f) Configuração 17 - Perdas

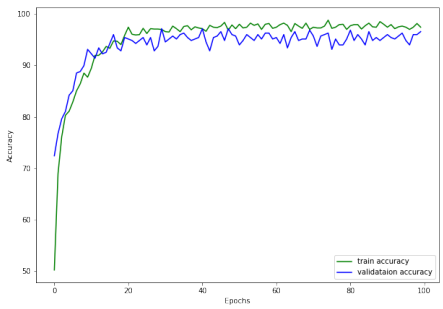


(g) Configuração 19 - Exatidão

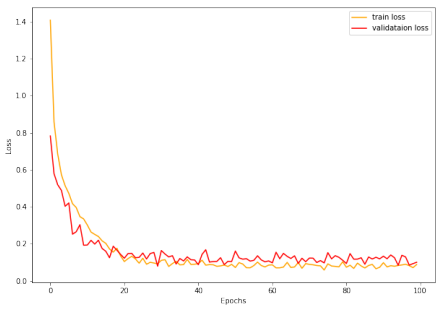


(h) Configuração 19 - Perdas

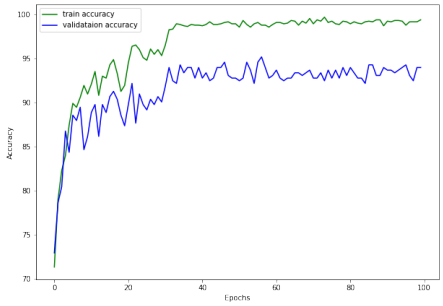
Figura 5.6: Gráficos das métricas obtidas no processo de aprendizagem (**a-b** *ResNet50*, **c-d** *ResNet152*, **e-f** *VGG16* e **g-h** *VGG19*).



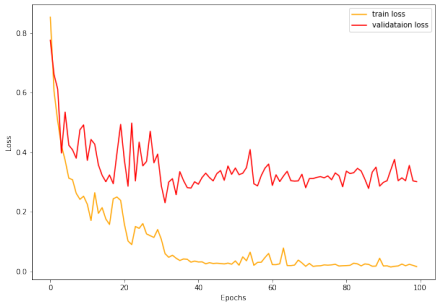
(a) Configuração 29 - Exatidão



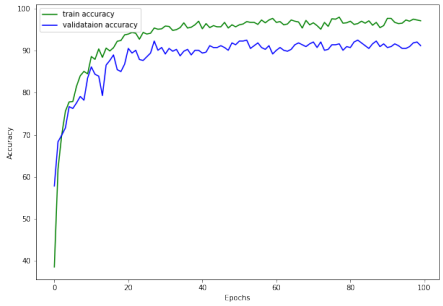
(b) Configuração 29 - Perdas



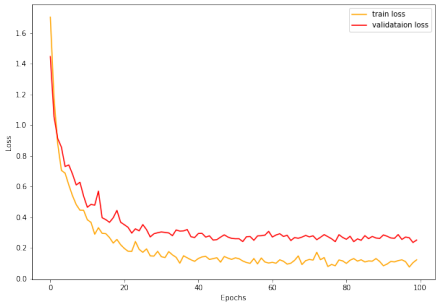
(c) Configuração 25 - Exatidão



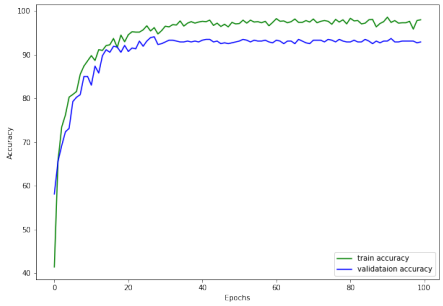
(d) Configuração 25 - Perdas



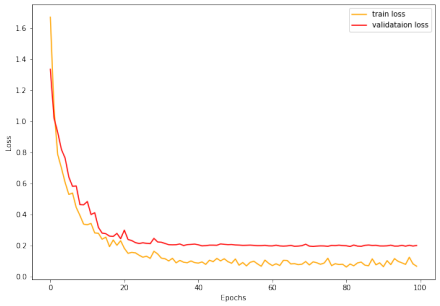
(e) Configuração 30 - Exatidão



(f) Configuração 30 - Perdas

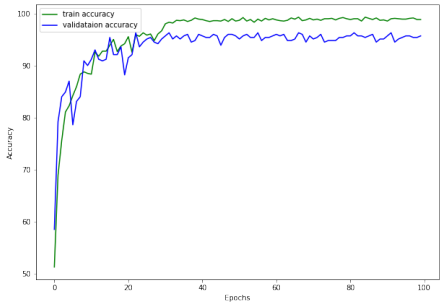


(g) Configuração 31 - Exatidão

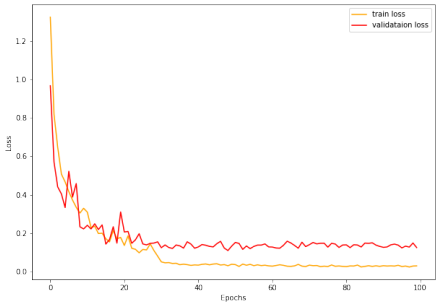


(h) Configuração 31 - Perdas

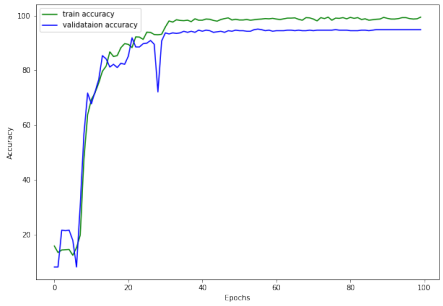
Figura 5.7: Gráficos das métricas obtidas no processo de aprendizagem (**a-d** *EfficientNetB0* e **e-h** *EfficientNetB7*).



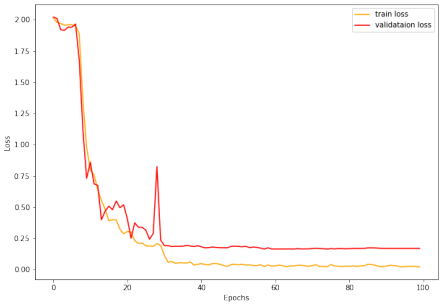
(a) Configuração 33 - Exatidão



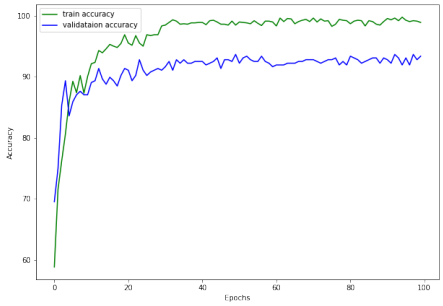
(b) Configuração 33 - Perdas



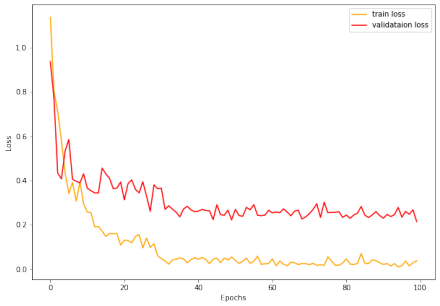
(c) Configuração 34 - Exatidão



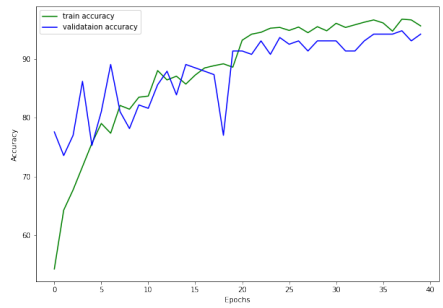
(d) Configuração 34 - Perdas



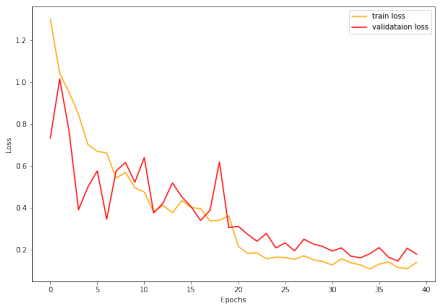
(e) Configuração 37 - Exatidão



(f) Configuração 37 - Perdas



(g) Configuração 38 - Exatidão



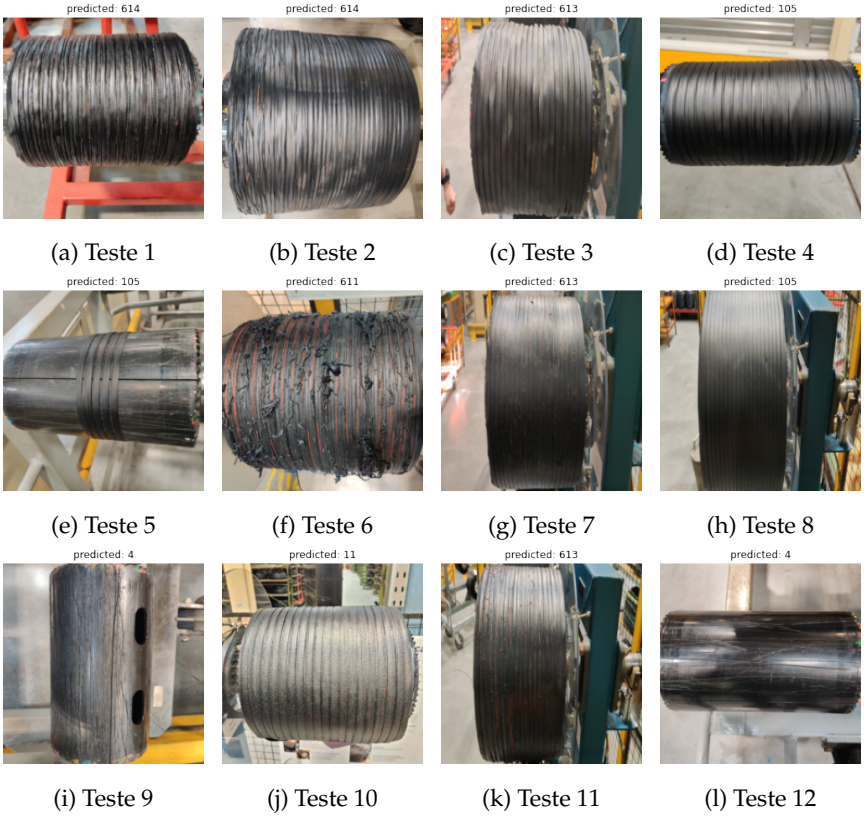
(h) Configuração 38 - Perdas

Figura 5.8: Gráficos das métricas obtidas no processo de aprendizagem (**a-d** *ConvNext-Small* e **e-h** *MobileNet-v2*).

Nas Figuras 5.6f e 5.6h é possível verificar um pequeno *overfitting* (sobreajuste), mas com a utilização do *Callback ModelCheckPoint*, anteriormente apresentado, garantiu-se que apenas os melhores resultados de desempenho foram guardados.

5.2 Resultados dos testes após processo de treino

Depois de concluído o processo de treino, o modelo foi testado com as imagens destinadas ao processo de teste, ou seja, imagens desconhecidas. O conjunto de imagens de teste foi submetido a uma divisão aleatória, tal como os restantes conjuntos. Estes resultados estão presentes na Figura 5.9, a título ilustrativo.



(a) Teste 1 (b) Teste 2 (c) Teste 3 (d) Teste 4

(e) Teste 5 (f) Teste 6 (g) Teste 7 (h) Teste 8

(i) Teste 9 (j) Teste 10 (k) Teste 11 (l) Teste 12

Figura 5.9: Imagens resultantes dos testes de classificação obtidos com as arquiteturas *EfficientNetB0* (**a-f**) e *ConvNext-Small* (**g-l**).

Nas Tabelas 5.3 e 5.4 estão sumarizados os resultados obtidos após o processo de aprendizagem com as arquiteturas, onde é possível verificar a precisão (*precision*), sensibilidade (*recall*) e pontuação F1 (*F1-score*) dos melhores resultados de cada arquitetura.

Tabela 5.3: Resultados obtidos no processo de aprendizagem (métricas Precisão, *Recall* e *F1-score*) (parte 1).

ResNet18					**ResNet50**				
Classe	**Precisão**	***Recall***	***F1-score***	**ACC**	**Classe**	**Precisão**	***Recall***	***F1-score***	**ACC**
105	0.95	0.90	0.93		**105**	0.96	0.94	0.95	
11	0.97	0.99	0.98		**11**	0.97	1.00	0.99	
4	1.00	1.00	1.00		**4**	1.00	1.00	1.00	
610	0.85	0.94	0.89	0.92	**610**	0.90	0.97	0.94	0.92
611	0.83	0.93	0.88		**611**	0.87	0.90	0.88	
613	0.93	0.89	0.91		**613**	0.94	0.90	0.92	
614	0.95	1.00	0.97		**614**	0.95	1.00	0.97	
ResNet152					**VGG16**				
Classe	**Precisão**	***Recall***	***F1-score***	**ACC**	**Classe**	**Precisão**	***Recall***	***F1-score***	**ACC**
105	0.94	0.94	0.94		**105**	0.95	0.95	0.95	
11	0.98	0.99	0.99		**11**	0.97	0.99	0.98	
4	1.00	1.00	1.00		**4**	0.99	0.98	0.98	
610	0.90	0.90	0.90	0.92	**610**	0.93	0.93	0.93	0.91
611	0.80	0.96	0.87		**611**	0.80	0.83	0.81	
613	0.92	0.87	0.89		**613**	0.89	0.88	0.88	
614	0.95	1.00	0.97		**614**	0.95	0.95	0.95	
VGG19					**EfficientNetB0**				
Classe	**Precisão**	***Recall***	***F1-score***	**ACC**	**Classe**	**Precisão**	***Recall***	***F1-score***	**ACC**
105	0.97	0.94	0.95		**105**	0.95	0.95	0.95	
11	0.97	0.99	0.98		**11**	0.98	1.00	0.99	
4	1.00	0.98	0.99		**4**	1.00	1.00	1.00	
610	0.86	0.88	0.87	0.91	**610**	0.93	1.00	0.97	0.95
611	0.73	0.88	0.80		**611**	0.67	0.89	0.76	
613	0.90	0.88	0.89		**613**	0.95	0.84	0.89	
614	0.95	1.00	0.97		**614**	1.00	1.00	1.00	

Tabela 5.4: Resultados obtidos no processo de aprendizagem (métricas Precisão, *Recall* e *F1-score*) (parte 2).

EfficientNetB7					**ConvNext-Small**				
Classe	**Precisão**	***Recall***	***F1-score***	**ACC**	**Classe**	**Precisão**	***Recall***	***F1-score***	**ACC**
105	0.96	0.91	0.94		**105**	0.98	0.91	0.94	
11	0.97	0.98	0.98		**11**	0.98	0.89	0.93	
4	1.00	0.98	0.99		**4**	1.00	1.00	1.00	
610	0.93	0.95	0.94	0.91	**610**	0.67	1.00	0.80	0.92
611	0.90	0.84	0.87		**611**	0.83	0.83	0.83	
613	0.85	0.93	0.89		**613**	0.82	0.87	0.85	
614	0.95	0.90	0.92		**614**	1.00	1.00	1.00	
MobileNet-v2					**Inception-v3**				
Classe	**Precisão**	***Recall***	***F1-score***	**ACC**	**Classe**	**Precisão**	***Recall***	***F1-score***	**ACC**
105	0.95	0.83	0.89		**105**	0.94	0.96	0.95	
11	0.98	0.98	0.98		**11**	0.99	1.00	1.00	
4	1.00	1.00	1.00		**4**	1.00	1.00	1.00	
610	0.75	1.00	0.86	0.92	**610**	0.98	0.87	0.92	0.92
611	1.00	0.81	0.90		**611**	0.73	1.00	0.85	
613	0.78	0.89	0.83		**613**	0.95	0.89	0.92	
614	0.88	0.88	0.88		**614**	0.95	1.00	0.97	

Pela análise das Tabelas 5.3 e 5.4, é possível verificar que, de um modo geral, todas as classes conseguem alcançar uma boa precisão, apesar de que, em alguns casos evidencia-se o desequilíbrio do número de amostras.

5.3 Aplicação Web: Demonstração de resultados

Foi desenvolvida uma aplicação Web com o objetivo de tornar o processo de teste e demonstração de resultados interativo. Na página inicial da aplicação (Figura 5.10) é possível escolher (menu lateral esquerdo) entre os dois modos de teste possíveis (classificação de imagens através de vídeo ou através de imagens guardadas). O modo de vídeo no Raspberry Pi é muito lento e como tal, não foi possível testar em contexto real. Em ambos os modos é possível escolher uma de quatro arquiteturas selecionadas para os testes.

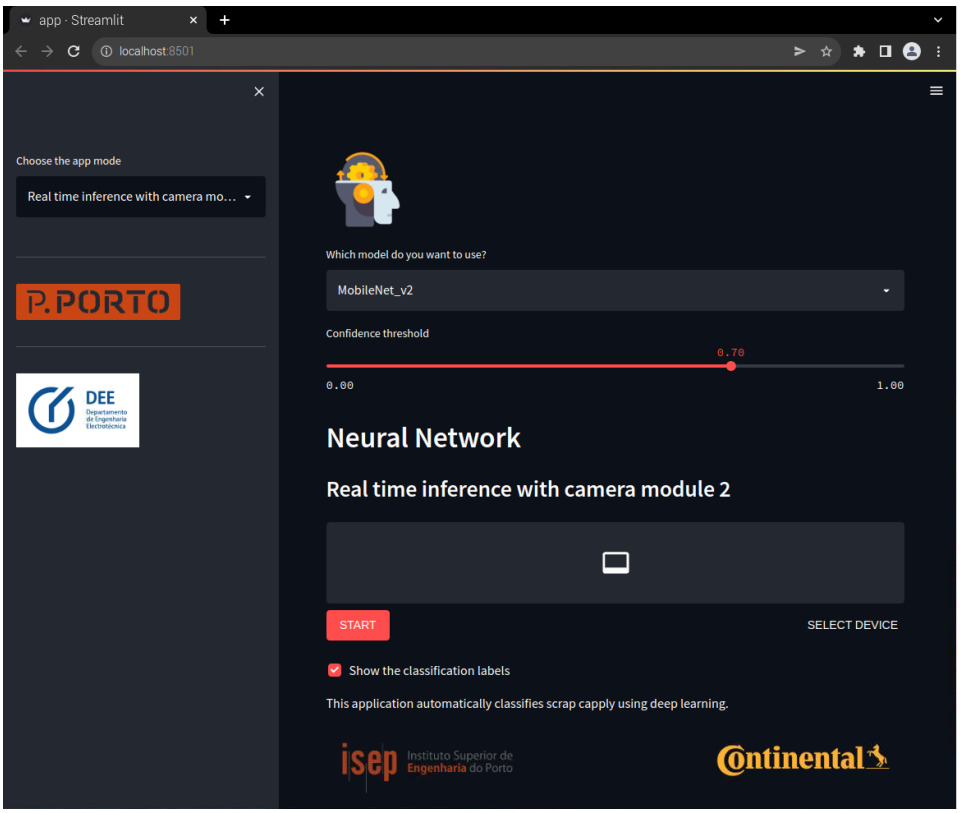


Figura 5.10: Página inicial da aplicação Web.

Na Figura 5.11 temos a página de testes com imagens guardadas localmente. O utilizador, como anteriormente referido, pode escolher a arquitetura que pretende usar (*EfficientNetB0*, *ConvNext-Small*, *ResNet50* e *MobileNet-v2*) e também pode ajustar o limite de confiança do sistema através do elemento *slider*.

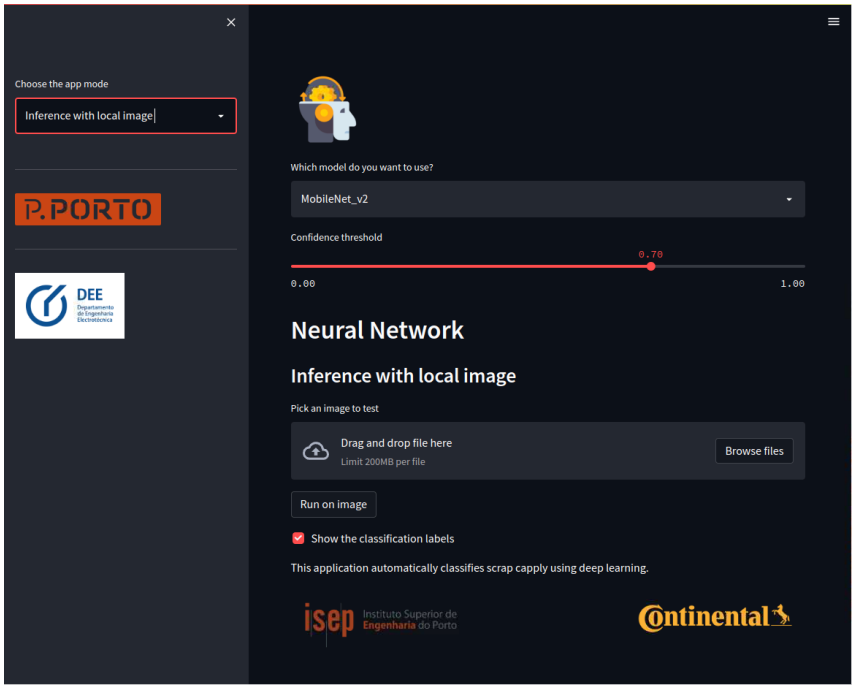


Figura 5.11: Página de testes com imagens guardadas.

Depois de termos escolhido o modelo e ajustado o limite de confiança, caso assim o desejarmos, podemos então carregar uma imagem através do botão "*Browse files*". Na Figura 5.12 é apresentado processo de escolha da imagem, a título ilustrativo.

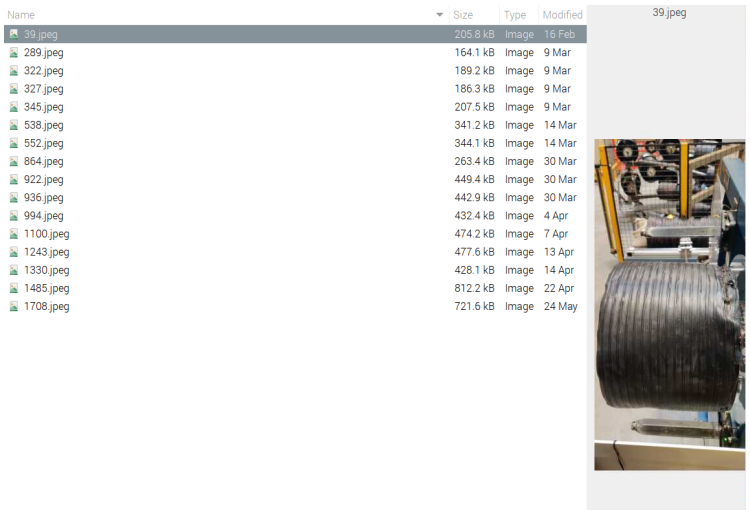


Figura 5.12: Processo de escolha da imagem.

Ao acionar a execução do teste (Figura 5.13 - botão *Run on image*) é apresentada uma mensagem ao utilizador indicando que o resultado está ser calculado (*Calculating results...*) e alguns instantes depois é apresentado o resultado da classificação com a devida etiqueta (classe) e a precisão. Caso não seja apresentado nenhum resultado quer dizer que a precisão da classificação foi inferior ao limite de confiança anteriormente configurado. Neste teste foi selecionada uma imagem de uma bobine amassada/danificada e o sistema classificou corretamente, como é possível verificar pela imagem da Figura 5.13.

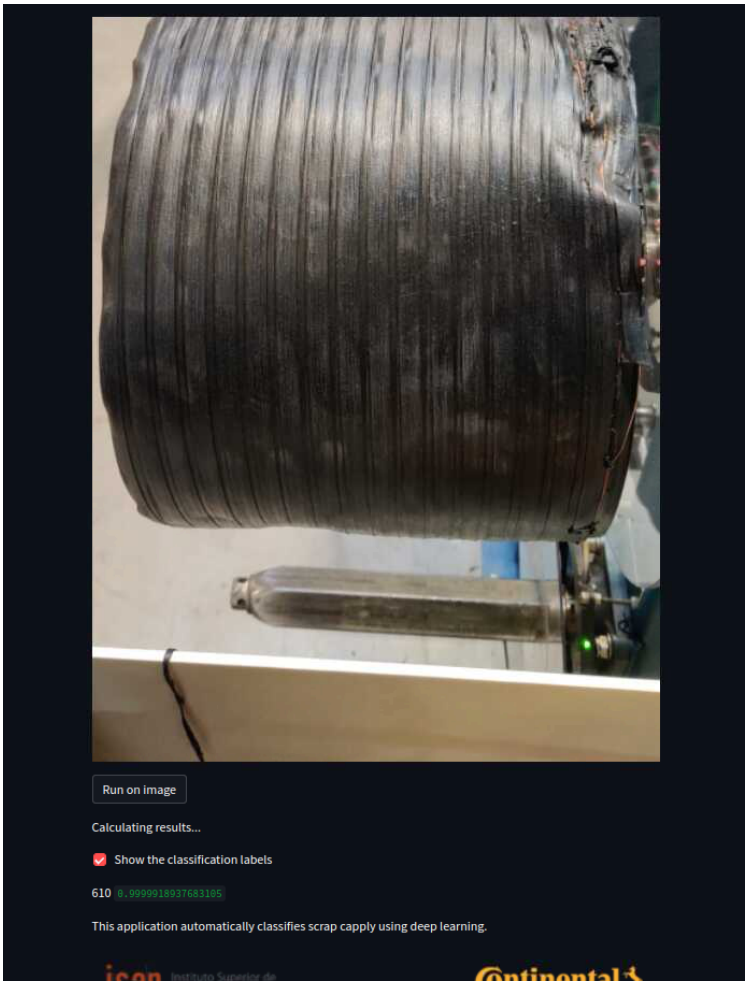


Figura 5.13: Testes na plataforma (Etiqueta 610 - "Bobine amassada/danificada").

Na Figura 5.14 é possível observar um outro exemplo, desta vez com uma imagem de uma bobine com uma imperfeição do tipo “Mau enrolamento/distribuição irregular” (etiqueta 613).

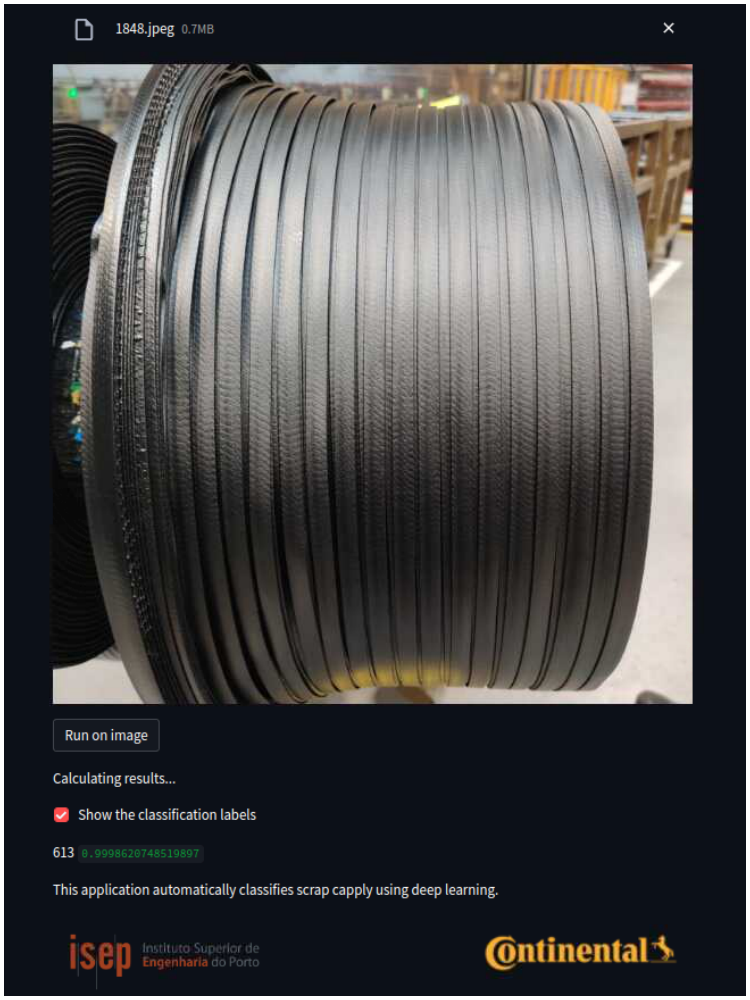


Figura 5.14: Testes na plataforma (Etiqueta 613 - "Mau enrolamento/distribuição irregular").

5.4 Testes em contexto real

Para os testes em contexto real, foi desenvolvido um algoritmo com recurso à biblioteca *OpenCV* [95] para aquisição de imagens através da câmara *module* 2 da Raspberry Pi. A resolução desta câmara foi configurada para 1280x1080 píxeis de forma a que o Raspberry Pi 4 consiga processar os *frames* o mais rápido possível e ao mesmo tempo garantir uma qualidade de imagem boa. Neste algoritmo é possível testar quatro dos modelos implementados (*EfficientNetB0*, *ConvNext-Small*, *ResNet50* e *MobileNet-v2*).

Nas Figuras 5.15 e 5.16 é apresentado um primeiro teste realizado na máquina, em que é possível verificar a imagem da bobine que está ser analisada pelo algoritmo e o seu resultado após a classificação.

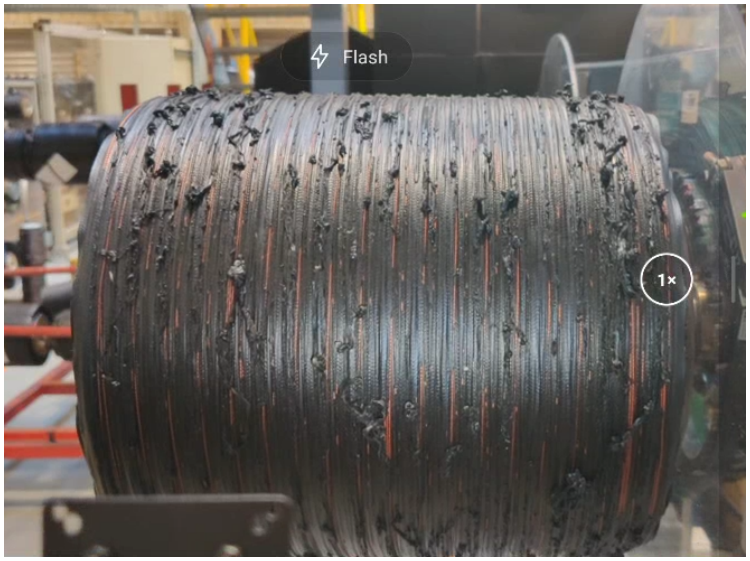


Figura 5.15: Teste na máquina (Etiqueta 611 - "Excesso de adesividade").

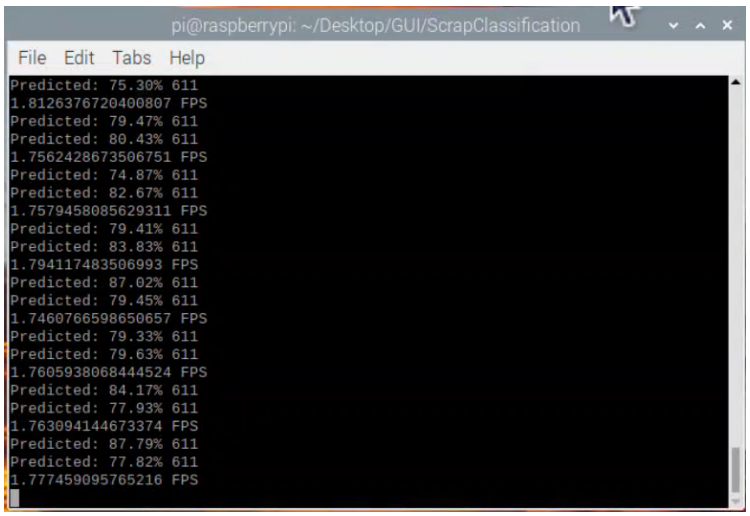


Figura 5.16: Resultado do teste na máquina (Etiqueta 611 - "Excesso de adesividade").

Na Figura 5.17 é apresentado o resultado da classificação enviado do Raspberry Pi para o PLC da máquina (*network* 3).

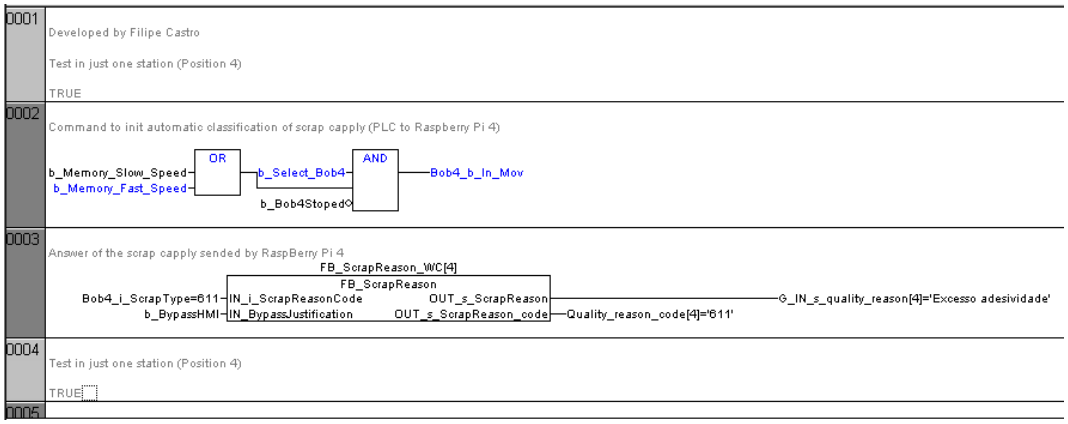


Figura 5.17: Resultado enviado para o PLC (Etiqueta 611 - "Excesso de adesividade").

As Figuras 5.18, 5.19 e 5.20 apresentam um outro teste realizado na máquina, agora com uma bobine de *capply* OK (etiqueta - 105).

Figura 5.18: Teste na máquina (Etiqueta 105 - "*Capply* OK").

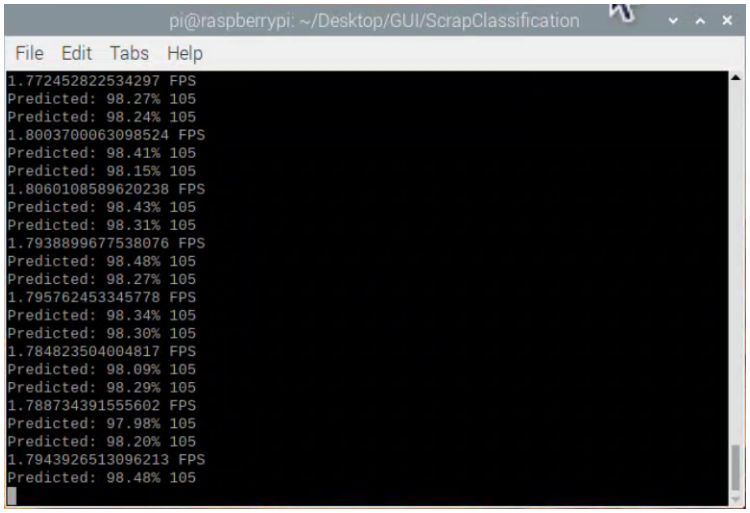


Figura 5.19: Resultado do teste na máquina (Etiqueta 105 - "*Capply* OK").

Figura 5.20: Resultado enviado para o PLC (Etiqueta 105 - "*Capply* OK").

Por padrão o *Pytorch* utiliza todos os núcleos disponíveis para processamento. De forma a melhorar o desempenho do sistema foi utilizada a configuração dos núcleos de processamento com recurso à função *torch.set_num_threads()*.

Para a comunicação entre o Raspberry Pi e o PLC, foi utilizada a biblioteca *PyAds* [96] que dá uso ao protocolo *Automation Device Specification* (ADS). O protocolo ADS é uma camada de transporte dentro do sistema TwinCAT e foi desenvolvido para troca de dados entre os diferentes módulos de *software*. Este protocolo permite a comunicação com outras ferramentas de qualquer ponto dentro do TwinCAT, ou até mesmo com outros dispositivos na rede, tendo por base uma comunicação *Transmission Control Protocol/Internet Protocol* (TCP/IP) ou *User Datagram Protocol/Internet Protocol* (UDP/IP) [97].

Capítulo 6

Conclusões e Perspetivas Futuras

Nesta dissertação foram investigadas e aplicadas metodologias de *Deep Learning* (DL), um ramo da Inteligência Artificial (IA), para a análise de imagens de bobines de *Capply*, nomeadamente, a classificação de imperfeições. Este trabalho permitiu conhecer de forma mais aprofundada as metodologias de DL aplicadas a tarefas de classificação de imagens, estendendo-se desde a parte teórica até ao desenvolvimento prático de um sistema. Como primeira abordagem, existiu a recolha de imagens de diferentes tipos de imperfeições de bobines, as imagens foram pré-processadas para que fossem corrigidas ou realçadas adequadamente as suas características, corrigindo píxeis anómalos ou reduzindo ruído. Seguiu-se a investigação, implementação, teste e avaliação de desempenho dos modelos e parâmetros utilizados para a realização da tarefa.

Tendo em consideração os resultados obtidos, pode-se concluir que as metodologias desenvolvidas, demonstraram um grande potencial do sistema. Em comparação, entre o sistema e o que é feito pelo operador, enquanto que o sistema classifica a bobine em tempo real desde que entra na máquina até que saí, o operador grande parte das vezes, apenas a classifica no momento em que introduz a bobine na máquina. Com este sistema é possível classificar corretamente a imperfeição da bobine como também, identificar quando é que a bobine passa a apresentar material em bom estado para produção, isto porque em alguns casos as bobines só apresentam imperfeições numa parte inicial.

As maiores dificuldades sentidas no desenvolvimento deste trabalho estão associadas à recolha das imagens de imperfeições de *capply* propostas para este trabalho, isto porque algumas destas imperfeições não surgiam com a mesma frequência do que outras, o que deu origem a um *dataset* desequilibrado e pequeno. Existiu também uma dificuldade sentida no que diz respeito ao tempo de treino dos modelos utilizados, que requerem uma grande capacidade de processamento, o que limita a execução da tarefa em questões temporais de execução. Por último, uma outra dificuldade sentida deve-se ao *hardware* utilizado para os testes práticos em contexto real, sendo que a câmara e o Raspberry Pi não apresentaram o melhor desempenho quando a bobine estava em movimento.

Como trabalhos futuros sugerem-se melhorias no conjunto de dados (*dataset*), ou seja, equilibrar as classes com o menor número de amostras e ao mesmo tempo enriquecer mais as outras classes para que os modelos consigam extrair mais informação/características. Explorar outro tipo de *hardware*, como por exemplo, uma câmara de alta velocidade. Quanto ao desempenho do Raspberry Pi, pode-se optar por utilizar o acelerador Coral USB da Google que possuí um *Tensor Processing Unit* (TPU) e que traz poderosos recursos de inferência ML ou então, esquecendo o Raspberry Pi e opta-se por outro microcomputador como por exemplo o Jetson Nano da NVIDIA.

Os objetivos definidos para este trabalho foram cumpridos e a abordagem proposta mostra a aplicação de técnicas de IA, suportadas por sistemas de visão, com a aplicabilidade em diversas tarefas de controlo de qualidade, aumentando a confiabilidade e otimização dos processos, permitindo realizar este tipo de tarefas sem a intervenção humana.

Referências

[1] Continental, "History - continental ag." Available at https://www.continental.com/en/company/history/. [Citado nas páginas 8 e 9]

[2] Continental, "Our structure | continental - continental ag." Available at https://www.continental.com/en/company/corporate-structure/. [Citado na página 10]

[3] Continental, "Company | continental - continental ag." Available at https://www.continental.com/en/company/. [Citado na página 13]

[4] Mabor, "Elevada qualidade com baixo preço | mabor." Available at https://www.mabor.pt/ligeiros. [Citado nas páginas ix e 15]

[5] Continental, "An inside look at the components that make up modern car tires | continental tires." Available at https://www.continental-tires.com/car/tire-knowledge/tire-basics/tire-components. [Citado nas páginas ix e 17]

[6] M. Caetano, "Linhas de mistura automáticas e semi-automáticas | ciência e tecnologia da borracha.." Available at https://www.ctborracha.com/processos/mistura/linhas-de-mistura-automaticas-e-semi-automaticas/. [Citado nas páginas ix e 20]

[7] VMI-Group, "Technology meets success - vmi group." Available at https://www.vmi-group.com/. [Citado nas páginas ix e 22]

[8] E. Abbas, "Continental pneus investe € 45 milhões em planta high-tech." Available at https://borrachatv.blogspot.com/2016/09/continental-pneus-investe-45-milhoes-em.html. [Citado nas páginas ix e 23]

[9] O. N. da Trofa, "Bruxelas aprova ampliação da continental mabor em famalicão - jornal o notícias da trofa." Available at https://www.onoticiasdatrofa.pt/bruxelas-aprova-ampliacao-da-continental-mabor-em-famalicao/. [Citado nas páginas ix e 24]

[10] C. Arquitectos, "Armazém continental mabor." Available at https://centralarquitectos.com/newsletters/newsletter_2013-10-15-15-30-58.html. [Citado nas páginas ix e 25]

[11] A. HAYES, "Quality control definition & example." Available at https://www.investopedia.com/terms/q/quality-control.asp. [Citado na página 27]

[12] Pactecindia, "male-worker-quality-control-inspection-of-metal-part-in-a-factory.jpg (1400×738)." Available at https://pactecindia.com/wp-content/uploads/2021/07/male-worker-quality-control-inspection-of-metal-part-in-a-factory.jpg. [Citado nas páginas x e 28]

[13] A. Walter and E. William, *Statistical method from the viewpoint of quality control*. 1986. [Citado na página 28]

[14] S. W. Ali, "Statistical process control for total quality," 1992. [Citado na página 28]

[15] P. Tambare, C. Meshram, C. C. Lee, R. J. Ramteke, and A. L. Imoize, "Performance measurement system and quality management in data-driven industry 4.0: A review," *Sensors 2022*, vol. 22, p. 224, 12 2021. [Citado na página 29]

[16] N. Rusk, "Deep learning," *Nature Methods 2016 13:1*, vol. 13, pp. 35–35, 12 2015. [Citado na página 29]

[17] R. Pandey, S. Naik, and R. Marfatia, "Image processing and machine learning for automated fruit grading system: A technical review," *International Journal of Computer Applications*, vol. 81, pp. 29–39, 11 2013. [Citado na página 29]

[18] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool, "Deepfruits: A fruit detection system using deep neural networks," *Sensors 2016, Vol. 16, Page 1222*, vol. 16, p. 1222, 8 2016. [Citado na página 29]

[19] C. L. Paiva, "Quality management: Important aspects for the food industry," *Food Industry*, 1 2013. [Citado na página 29]

[20] J. Fingas, "Google ai could keep baby food safe | engadget." Available at https://www.engadget.com/2017-07-25-google-ai-helps-make-safer-baby-food.html?guccounter=1. [Citado na página 29]

[21] Fujitsu, "Fujitsu develops state-of-the-art ai solution to revolutionize non-destructive testing manufacturing inspection - fujitsu cemea&i." Available at https://www.fujitsu.com/fts/about/resources/news/press-releases/2017/emeai-20171002-fujitsu-develops-state-of-the-art-ai.html. [Citado na página 29]

[22] Fujitsu, "Artificial intelligence solution from fujitsu helps siemens gamesa significantly accelerate quality assurance procedures - fujitsu emeia." Available at `https://www.fujitsu.com/emeia/about/resources/news/press-releases/2017/emeai-20171107-artificial-intelligence-solution-from.html`. [Citado na página 29]

[23] M. Elgendy, *Deep Learning for Vision Systems*. 2020. [Citado nas páginas x, 30, 31, 32, 33, 37 e 38]

[24] M. Kantardzic, *Data mining: concepts, models, methods, and algorithms*. second ed., 2011. [Citado nas páginas 33, 34, 35 e 36]

[25] M. Bramer, *Principles of Data Mining*. Springer London, 2007. [Citado na página 33]

[26] J. Jackson, "Data mining; a conceptual overview," *Communications of the Association for Information Systems*, vol. 8, 2002. [Citado na página 34]

[27] S. Gopal, K. Patro, and K. K. Sahu, "Normalization: A preprocessing stage," tech. rep., 2015. [Citado na página 34]

[28] MathWorks, "Noise removal - matlab & simulink." Available at `https://www.mathworks.com/help/images/noise-removal.html#buh9ylp-70`, 2022. [Citado nas páginas 38 e 39]

[29] J. Ambrosio, "How waymo is using ml to build a scalable, autonomous 'driver'." Available at `https://exchange.scale.com/public/blogs/how-ml-waymo-building-scalable-autonomous-driver-dmitri-dolgov`, 2022. [Citado na página 39]

[30] IBM, "Machine learning." Available at `https://www.ibm.com/cloud/learn/machine-learning`, 2020. [Citado nas páginas 39 e 40]

[31] IBM, "Neural networks." Available at `https://www.ibm.com/cloud/learn/neural-networks`, 2020. [Citado na página 40]

[32] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, pp. 427–437, 7 2009. [Citado nas páginas 41 e 42]

[33] Y. Jiao and P. Du, "Performance measures in evaluating machine learning based bioinformatics predictors for classifications," *Quantitative Biology*, vol. 4, pp. 320–330, 12 2016. [Citado na página 43]

[34] Y. D. Ma, Q. Liu, and Z. B. Qian, "Automated image segmentation using improved pcnn model based on cross-entropy," pp. 743–746, 2004. [Citado na página 43]

[35] S. Jadon, *A survey of loss functions for semantic segmentation*. 2020. [Citado na página 43]

[36] P. Christoffersen and K. Jacobs, "The importance of the loss function in option valuation," *Journal of Financial Economics*, vol. 72, pp. 291–318, 5 2004. [Citado na página 44]

[37] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986. [Citado na página 44]

[38] L. Devroye, L. Gyorfi, and G. Lugosi, "A probabilistic theory of pattern recognition," p. 636, 1996. [Citado nas páginas 44 e 45]

[39] IBM, "Naive bayes." Available at `https://www.ibm.com/docs/en/ias?topic=procedures-naive-bayes`, 2021. [Citado na página 45]

[40] ScienceDirect, "Naive bayesian network - an overview | sciencedirect topics." Available at `https://www.sciencedirect.com/topics/computer-science/naive-bayesian-network`. [Citado nas páginas x e 45]

[41] P. Cunningham and S. J. Delany, "K-nearest neighbour classifiers," *ACM Computing Surveys*, 3 2007. [Citado na página 46]

[42] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov, "Neighbourhood components analysis," 2004. [Citado na página 46]

[43] H. Drucker, D. Wu, and V. N. Vapnik, "Support vector machines for spam categorization," *IEEE Transactions on Neural Networks*, vol. 10, pp. 1048–1054, 1999. [Citado na página 46]

[44] Sunil, "Svm | support vector machine algorithm in machine learning." Available at `https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/`. [Citado na página 46]

[45] M. Gordan, C. Kotropoulos, and I. Pitas, "Application of support vector machines classifiers to visual speech recognition," tech. rep., 2002. [Citado na página 47]

[46] K. Priddy and P. Keller, *Artificial Neural Networks: An Introduction*. 2005. [Citado na página 48]

[47] K. M. Almhdi, P. Valigi, V. Gulbinas, R. Westphal, and R. Reuter, "Classification with artificial neural networks and support vector machines: application to oil fluorescence spectra," tech. rep., 2007. [Citado na página 49]

[48] G. James, D. Witten, T. Hastie, and R. Tibshirani, "An introduction to statistical learning with applications in r," tech. rep., 2015. [Citado na página 49]

[49] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958. [Citado na página 50]

[50] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 11 2018. [Citado na página 50]

[51] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *International Journal of Engineering Applied Sciences and Technology*, vol. 4, pp. 310–316, 2017. [Citado nas páginas 51 e 53]

[52] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton, "On rectified linear units for speech processing," tech. rep., 2013. [Citado na página 51]

[53] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," tech. rep., Department of Computer Science, University of Toronto, 2013. [Citado na página 51]

[54] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," tech. rep., 2010. [Citado na página 51]

[55] B. Ramsundar and R. B. Zadeh, "Tensorflow for deep learning," tech. rep., 3 2018. [Citado nas páginas x, 51, 52, 59 e 61]

[56] J. Turian, J. Bergstra, and Y. Bengio, "Quadratic features and deep architectures for chunking," tech. rep., 6 2009. [Citado na página 52]

[57] W. A. Little, "The existence of persistent states in the brain," *Mathematical Biosciences*, vol. 19, pp. 101–120, 2 1974. [Citado na página 52]

[58] R. M. Neal, "Connectionist learning of belief networks," tech. rep., 1992. [Citado na página 52]

[59] Y. Lecun, Y. Bengio, and G. Hinton, *Deep learning*, vol. 521. Nature Publishing Group, 5 2015. [Citado na página 52]

[60] W. Little and G. L. Shaw, "Analytic study of the memory storage capacity of a neural network," *Mathematical Biosciences*, vol. 39, pp. 281–290, 6 1978. [Citado na página 52]

[61] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Citado nas páginas 52 e 53]

[62] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. [Citado na página 54]

[63] A. Biswal, "Top 10 deep learning algorithms you should know in 2022." Available at `https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm`, 2 2022. [Citado nas páginas x e 55]

[64] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," 2007. [Citado na página 55]

[65] A. Fischer and C. Igel, "An introduction to restricted boltzmann machines," pp. 14–36, 2012. [Citado na página 55]

[66] G. E. Hinton and S. Osindero, "A fast learning algorithm for deep belief nets yee-whye teh," tech. rep., 2006. [Citado na página 56]

[67] A. Meshref, K. El-Dash, M. Basiouny, and O. El-Hadidi, "Implementation of a life cycle cost deep learning prediction model based on building structure alternatives for industrial buildings," *Buildings*, vol. 12, p. 502, 4 2022. [Citado nas páginas x e 56]

[68] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997. [Citado na página 57]

[69] W. Wu, S. Y. An, P. Guan, D. S. Huang, and B. S. Zhou, "Time series analysis of human brucellosis in mainland china by using elman and jordan recurrent neural networks," *BMC Infectious Diseases*, vol. 19, 5 2019. [Citado na página 57]

[70] D. Gershgorn, "The data that transformed ai research-and possibly the world." Available at `https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/`, 2017. [Citado na página 58]

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012. [Citado na página 58]

[72] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, 1989. [Citado na página 58]

[73] R. P. Naidu, P. S. Sagar, K. Praveen, K. Kiran, and K. Khalandar, "Stress recognition using facial landmarks and cnn (alexnet)," vol. 2089, IOP Publishing Ltd, 11 2021. [Citado nas páginas x e 58]

[74] A. Kumar, "Convolutional filter." Available at `https://vitalflux.com/wp-content/uploads/2021/11/Input-image-along-with-convolutional-layer.png`, 2021. [Citado nas páginas x e 59]

[75] M. Skarzynski, "Statistical modeling and deep learning image analysis for lung cancer risk prediction." Available at `https://marskar.github.io/frm/#1`, 2020. [Citado nas páginas x e 60]

[76] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," tech. rep., 2015. [Citado na página 61]

[77] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, "Rethinking the inception architecture for computer vision," tech. rep., University College London, 12 2015. [Citado na página 62]

[78] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 10 2016. [Citado na página 62]

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 9 2014. [Citado na página 62]

[80] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 12 2015. [Citado na página 63]

[81] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 5 2019. [Citado nas páginas x e 63]

[82] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," 1 2022. [Citado na página 63]

[83] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 4 2017. [Citado na página 63]

[84] M. M. Leonardo, T. J. Carvalho, E. Rezende, R. Zucchi, and F. A. Faria, "Deep feature-based classifiers for fruit fly identification," pp. 41–47, Institute of Electrical and Electronics Engineers Inc., 1 2019. [Citado nas páginas x, 64, 65 e 66]

[85] S. Phiphiphatphaisit and O. Surinta, "Food image classification with improved mobilenet architecture and data augmentation," pp. 51–56, Association for Computing Machinery, 3 2020. [Citado nas páginas x e 66]

[86] "Python release python 3.10.2 | python.org." Available at `https://www.python.org/downloads/release/python-3102/`. [Citado na página 69]

[87] "Tensorflow." Available at `https://www.tensorflow.org/`. [Citado na página 69]

[88] "Pytorch." Available at `https://pytorch.org/`. [Citado na página 69]

[89] "Numpy." Available at `https://numpy.org/`. [Citado na página 69]

[90] "seaborn: statistical data visualization — seaborn 0.11.2 documentation." Available at `https://seaborn.pydata.org/`. [Citado na página 69]

[91] "Matplotlib — visualization with python." Available at `https://matplotlib.org/`. [Citado na página 69]

[92] "scikit-learn: machine learning in python — scikit-learn 1.1.1 documentation." Available at `https://scikit-learn.org/stable/`. [Citado na página 69]

[93] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *International conference on machine learning*, vol. 28, pp. 1139–1147, 2013. [Citado na página 76]

[94] D. Verma, M. Kumar, and S. Eregala, *Deep Demosaicing Using ResNet-Bottleneck Architecture*, vol. 1148. 2020. [Citado na página 77]

[95] O. team, "Opencv." Available at `https://opencv.org/`, 2022. [Citado na página 105]

[96] PyPi, "Pyads." Available at `https://pypi.org/project/pyads/`, 11 2021. [Citado na página 108]

[97] Beckhoff, "Ads-communication." Available at `https://infosys.beckhoff.com/english.php?content=../content/1033/cx8190_hw/5091854987.html&id=`, 2022. [Citado na página 108]