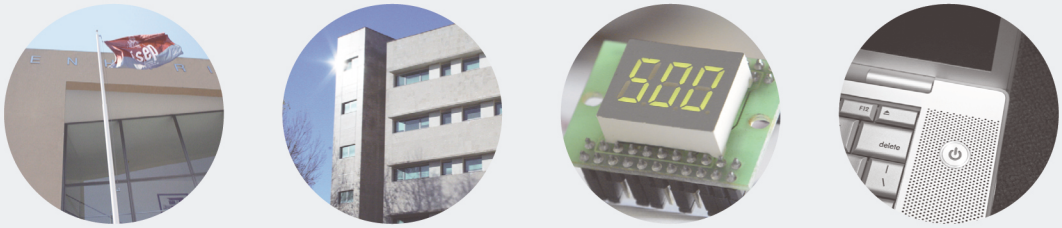
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Sistema pick and place baseado numa rede neuronal profunda treinada em dados sintéticos

NUNO FILIPE LOPES MARQUES
julho de 2023

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Sistema *pick and place* baseado numa rede neuronal profunda treinada em dados sintéticos

Nuno Marques

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

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Julho, 2023

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 Sistemas Autónomos.

Candidato: Nuno Marques, Nº 1161744, 1161744@isep.ipp.pt

Orientação Científica: Maria Isabel Martins, MIS@isep.ipp.pt

Empresa: INEGI - Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial

Orientador: Luís Carlos Moreira, lcmoreira@inegi.up.pt

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

Julho, 2023

Agradecimentos

Em primeiro lugar quero deixar um agradecimento enorme à minha família. Nomeadamente à minha mãe, Fátima Marques, ao meu pai, Nuno Marques, à minha irmã e futura doutora, Joana Marques e à minha namorada, Engenheira Inês Araújo. Afirmo, com toda a certeza, que sem o vosso apoio não conseguiria concluir esta etapa da minha vida.

Um agradecimento especial à minha entidade patronal, o Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial, mais concretamente à equipa Data Driven Automation. Ao Engenheiro Marco Rodrigues, ao Engenhei ro Diogo Sousa, ao Engenheiro Henrique Valente e ao Engenheiro Miguel Ferreira por todas as conversas e ideias dadas que ajudaram na conclusão deste projeto.

Aos meus orientadores Engenheiro Luís Moreira e Engenheira Isabel Martins pela disponibilidade e por todo o conhecimento cedido durante esta jornada.

Resumo

A visão computacional e a aprendizagem profunda têm desempenhado um papel cada vez mais importante na automatização de processos industriais, permitindo o desenvolvimento de sistemas inteligentes capazes de realizar tarefas complexas, como a deteção e reconhecimento de objetos. No contexto dos projetos do INEGI (Instituto de Engenharia Mecânica e Gestão Industrial), surge a necessidade de implementar soluções eficientes que combinem estas tecnologias para otimizar a realização de tarefas específicas.

Um sistema *pick and place*, que consiste na identificação e movimentação automatizada de objetos, é amplamente utilizado em várias indústrias, como a automobilística, eletrónica e logística. No entanto, a implementação bem-sucedida deste sistema requer a capacidade de detetar e reconhecer objetos de forma precisa e eficiente. Neste contexto, este estudo propõe um novo *workflow* baseado em visão computacional e aprendizagem profunda para projetos do INEGI, que visa melhorar o desempenho e a velocidade de implementação de sistemas que recorram a Inteligência Artificial (IA). O objetivo é superar as limitações dos métodos baseados em redes neuronais profundas, que exigem grandes quantidades de dados de treino e são computacionalmente intensivos. Para isso, propõe-se o uso de dados de treino sintéticos, permitindo o treino de uma versão preliminar do sistema ainda antes dos dados reais estarem disponíveis. O estudo envolveu a modelação 3D de ferramentas de oficina selecionadas e a geração de conjuntos de dados recorrendo a técnicas de *image augmentation*. Foram utilizados modelos pré-treinados para treinar diferentes modelos, com recurso a *transfer learning*, com base nesses conjuntos de dados.

Para a implementação física do sistema *pick and place*, utilizou-se um braço mecânico KUKA LBR IIWA 14 R820. O desempenho dos modelos de deteção e reconhecimento das ferramentas, bem como o funcionamento do sistema foram avaliados. Comparando um modelo treinado com dados sintéticos e um treinado com dados reais, verificou-se que o modelo treinado com dados reais teve um desempenho superior em todas as métricas. Por outro lado, um conjunto de dados híbrido entre dados reais e dados sintéticos consegue apresentar, em certos casos, resultados superiores ao conjunto de dados puramente real apresentando, no melhor dos casos, um desempenho superior em 2,89% e, no pior dos casos, um desempenho inferior em 0,62%. Além disso, o treino prévio com dados sintéticos permitiu agilizar o processo de anotação das imagens reais e diminuir o número de iterações necessárias para a

convergência do modelo. Também foi avaliado se, relativamente à geração de dados sintéticos, seria melhor criar réplicas 3D dos objetos a serem identificados ou então utilizar os modelos disponibilizados pela vasta comunidade que existe na Internet. Concluiu-se que se alcança resultados superiores utilizando a combinação da especificidade criada na réplica 3D com a generalidade adquirida nos diferentes modelos 3D semelhantes disponibilizados por outros. Também se recorreu ao *Compute Unified Device Architecture* (CUDA) para acelerar o processo de treino, aumentando a velocidade de treino de uma rede neuronal em 524,25%. Os resultados obtidos com este trabalho permitem validar a abordagem proposta.

Palavras-Chave: Visão computacional, Aprendizagem Profunda, *Transfer Learning*, *pick and place*, dados de treino sintéticos, deteção de objetos, reconhecimento de objetos.

Abstract

Computer vision and deep learning have been playing an increasingly important role in automating industrial processes, enabling the development of intelligent systems capable of performing complex tasks such as object detection and recognition. Within the context of the projects at INEGI (Institute of Mechanical Engineering and Industrial Management), there is a need to implement efficient solutions that combine these technologies to optimize the performance of specific tasks.

A pick-and-place system, which involves the automated identification and movement of objects, is widely used in various industries such as automotive, electronics, and logistics. However, the successful implementation of this system requires the ability to detect and recognize objects accurately and efficiently. In this context, this study proposes a new workflow based on computer vision and deep learning for INEGI projects, aiming to improve the performance and implementation speed of systems that utilize Artificial Intelligence (AI). The goal is to overcome the limitations of methods based on deep neural networks, which require large amounts of training data and are computationally intensive. To achieve this, the use of synthetic training data is proposed, allowing for the training of a preliminary version of the system even before real data is available. The study involved the 3D modeling of selected tools and the generation of datasets using image augmentation techniques. Pretrained models were utilized to train different models using transfer learning based on these datasets.

For the physical implementation of the pick-and-place system, a KUKA LBR IIWA 14 R820 robotic arm was used. The performance of the tool detection and recognition models, as well as the system's functionality, were evaluated. By comparing a model trained with synthetic data and one trained with real data, it was found that the model trained with real data outperformed in all metrics. On the other hand, a hybrid dataset consisting of real and synthetic data was able to achieve, in certain cases, better results than a purely real dataset, with the best-case scenario showing a performance improvement of 2.89% and the worst-case scenario showing a performance decrease of 0.62%. Additionally, pretraining with synthetic data facilitated the annotation process of real images and reduced the number of iterations required for model convergence. It was also evaluated whether it is better to create 3D replicas of the objects to be identified or to use models available from the vast

online community. It was concluded that superior results can be achieved by combining the specificity created in the 3D replicas with the generality acquired from different similar 3D models available from others. CUDA was also utilized to accelerate the training process, resulting in a 524.25% increase in neural network training speed. The results obtained from this work validate the proposed approach.

Keywords: Computer vision, Deep learning, Transfer Learning, Pick and Place, Synthetic Training Data, Object Detection, Object Recognition.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Acrónimos

2D Duas Dimensões

3D Três Dimensões

AP *Average Precision*

API *Application Programming Interface*

BiFPN Weighted Bi-directional Feature Pyramid Network

CAD *Computer-Aided Design*

COCO *Common Objects in Context*

CPU *Central Processing Unit*

CUDA *Compute Unified Device Architecture*

FN *False Negative*

FP *False Positive*

FPN *Feature Pyramid Network*

GPU *Graphics Processing Unit*

IA Inteligência Artificial

INEGI Instituto de Ciência e inovação em Engenharia Mecânica e Engenharia Industrial

IoU *Intersection over Union*

ISEP Instituto Superior de Engenharia do Porto

mAP *Mean Average Precision*

OPC UA *Open Platform Communications Unified Architecture*

RELU *Rectified Linear Unit*

RGB *Red*, *Green*, *Blue*

RNC Redes Neuronais Convolucionais

SSD *Single-Shot Detector*

TP *True Positive*

Capítulo 1

Introdução

No ambiente global e dinâmico em que as empresas estão hoje inseridas, o processo de desenvolvimento de produto é uma componente importante para conquistar uma vantagem competitiva. As empresas estão a explorar novas tecnologias e processos para garantir qualidade de produto e a redução de custos do projeto. O avanço da tecnologia computacional permitiu a implementação de novos processos de desenvolvimento e validação através da introdução de ferramentas virtuais [1].

1.1 Contextualização

As redes neuronais profundas e a aprendizagem de máquina tiveram um impacto significativo em diversos campos, incluindo visão computacional e robótica [2]. O desempenho destas técnicas depende muito do acesso a grandes quantidades de dados de treino fidedignos. No entanto, adquirir dados reais para aprendizagem de máquina pode ser caro e demorado, ou até mesmo impossível, devido a restrições éticas ou logísticas. Por outro lado, recolher e anotar dados requer uma quantidade significativa de esforço e recursos. Uma alternativa para superar estes desafios é o recurso a conjuntos de dados gerados sinteticamente. Por conjunto de dados sintéticos entende-se dados gerados por algoritmos de computador, em vez de adquiridos no mundo real. Os dados sintéticos têm várias vantagens, incluindo a capacidade de simular diversos cenários que são difíceis, perigosos ou caros de replicar no mundo real. Além disso, os dados sintéticos podem reduzir custos e encurtar os ciclos de desenvolvimento ao reduzir a necessidade de dados do mundo real.

Uma área que pode beneficiar significativamente de conjuntos de dados gerados sinteticamente é a visão artificial, uma subárea da visão computacional que se concentra no desenvolvimento de algoritmos para processar e interpretar imagens ou vídeos [3]. As RNC são um tipo popular de rede neuronal profunda usada em tarefas de visão artificial. Estas redes podem atingir alto desempenho quando treinadas em grandes quantidades de dados de alta qualidade. Contudo, mesmo para dados gerados artificialmente, o processo de aquisição de um conjunto de dados extenso, pode ser demorado. Como tal, utilizam-se técnicas de *data augmentation*, mais concretamente *image augmentation* para gerar novos dados a partir de imagens previamente adquiridas com ligeiras variações, tais como orientação do objeto, iluminação da cena, planos de fundo, etc.

No caso específico dos sistemas *pick and place*, a visão artificial desempenha um papel crucial na deteção e reconhecimento de objetos que precisam de ser recolhidos e movidos. Os conjuntos de dados gerados sinteticamente podem fornecer uma maneira eficiente de simular várias formas, tamanhos e orientações de objetos que um sistema *pick and place* pode encontrar, o que pode ser difícil de capturar em dados do mundo real. Na geração de dados sintéticos também é possível simular uma variedade de condições de iluminação, que podem ser difíceis de replicar no mundo real.

1.2 Definição do Problema

O INEGI é uma referência para investigação, inovação e transferência de conhecimento, estando envolvido em múltiplos projetos europeus, onde os temas vão desde gestão de energia até ao uso de Inteligência Artificial (IA) com recurso a visão computacional para diagnósticos médicos. Contudo, um dos focos do INEGI é ajudar a industria portuguesa a alcançar a quarta revolução industrial.

A quarta revolução industrial (industria 4.0) visa transformar a indústria de manufatura por meio da integração de tecnologias avançadas e digitalização no processo produtivo. Esta era de industrialização é caracterizada pelo uso de sistemas ciberfísicos, Internet das Coisas, computação em nuvem e IA , entre outras tecnologias de ponta. O principal objetivo da Indústria 4.0 é criar um ecossistema industrial altamente flexível, automatizado e inteligente que permita processos de fabricação mais rápidos e eficientes. Por meio da implementação da Indústria 4.0, os fabricantes podem reduzir custos, alcançar produtos de maior qualidade e aumentar a produtividade, levando a uma vantagem competitiva no mercado global [1].

Como tal, a sensorização é um conceito-chave da Indústria 4.0, uma vez que envolve equipar as máquinas com sensores que podem recolher e transmitir dados em tempo real para informar os processos de tomada de decisão. Ao projetar máquinas

com a Indústria 4.0 em mente, os fabricantes podem beneficiar de análise de dados aprimorada, manutenção preditiva e controlo de qualidade, entre outras vantagens.

Atualmente, no INEGI, o processo de implementação de um sistema de identificação de objetos com recurso a uma rede neuronal é demorado. Na Figura 1.1 está esquematizado este processo.

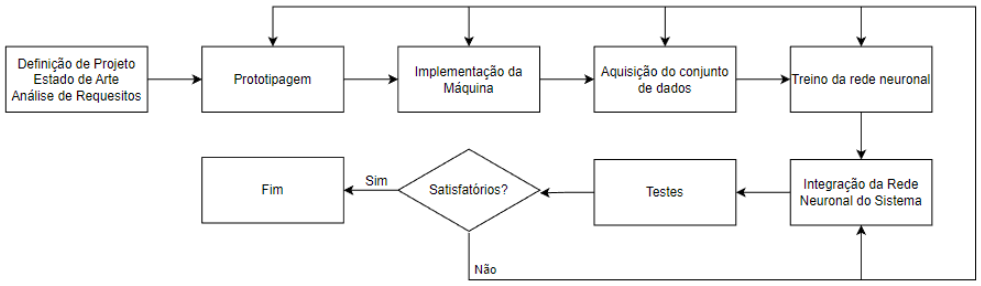


Figura 1.1: *Workflow* do INEGI.

O processo inicia-se com a definição do projeto, identificação do estado da arte e análise de requisitos. Posteriormente, passa-se à fase de prototipagem da máquina e à sua implementação. Após a conclusão do sistema, é possível adquirir o conjunto de dados de treino, uma etapa crucial que atrasa o processo, uma vez que é necessário adquirir um elevado número de dados nas mesmas condições em que a máquina irá atuar. Com o conjunto de dados obtido, inicia-se um processo demorado de treino da rede neuronal, de modo a poderem ser realizados testes ao funcionamento global da máquina. Se os resultados dos testes forem positivos, a implementação da máquina chegará à sua conclusão. No entanto, caso os resultados obtidos não sejam os esperados, é necessário identificar e resolver o problema, o que poderá implicar um regresso à fase de prototipagem da máquina, implementação da máquina e aquisição de um novo conjunto de dados e ao treino da rede neuronal, para se procederem a adaptações ao sistema, com impacto em todo o processo.

Caso a implementação seja urgente, o INEGI, atualmente, não dispõe da capacidade para dar uma resposta rápida a este pedido, devido às limitações impostas pela aquisição do conjunto de dados e pelo subsequente treino da rede neuronal. Adicionalmente, para alguns clientes, a inoperabilidade do sistema representa um custo significativo. A presente dissertação tem como objetivo propor uma solução para este problema. Na Figura 1.2 é apresentada uma proposta de um novo processo de integração de um novo *workflow*.

Com o novo *workflow* proposto, durante a prototipagem da máquina será criado, em paralelo, um conjunto de dados sintético com recurso a modelação 3D (3 dimensões) dos objetos a identificar. Com este conjunto de dados será treinada uma rede neuronal provisória que, posteriormente, será integrada no sistema, quando este estiver fisicamente montado. De seguida, o sistema será testado na sua totalidade, o que gerará *feedback* tanto para a rede neuronal provisória como para a parte física

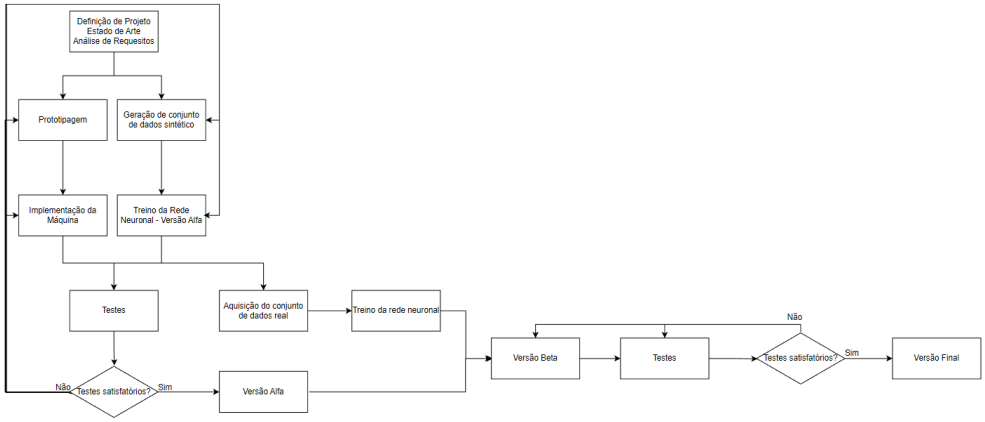


Figura 1.2: Novo *workflow* proposto.

do sistema. Se o resultado dos testes for satisfatório, a máquina fica operacional, porém com o funcionamento limitado que requer supervisionamento, dado que não é expectável que a rede treinada com imagens sintéticas se comporte exatamente da mesma maneira que uma rede treinada com imagens reais.

Paralelamente, quando o sistema físico estiver montado, poderá começar a aquisição do conjunto de dados reais. Desta forma, garante-se a qualidade do conjunto de dados, uma vez que estes são adquiridos nas mesmas condições em que a máquina irá operar, resultando numa rede neuronal mais precisa.

Este *workflow* apresenta algumas vantagens, relativamente ao seu antecessor. A primeira e mais evidente é que, embora o sistema tenha de trabalhar de uma forma limitada, a máquina estará operacional mais cedo. A segunda é que não será preciso chegar quase ao fim do projeto para obter *feedback* do funcionamento da máquina. Por último, durante uma versão Alfa da máquina, a aquisição e anotação do conjunto de dados pode ser automatizada. Uma vez que a máquina irá estar a identificar tanto a posição do objeto na imagem como a classe a que este pertence e, uma vez que esta é a informação contida nas anotações de cada imagem, é possível automatizar este processo com a rede neuronal treinada com imagens sintéticas. É evidente que, no fim da aquisição, este conjunto de dados estará sujeito a uma revisão e a um possível ajuste manual.

1.2.1 Objetivos

O objetivo da presente dissertação é validar a eficácia do novo *workflow* proposto. Para isso, será desenvolvido um sistema *pick and place* com recurso a visão computacional e a IA. Para a identificação dos objetos, será treinado um modelo com imagens sintéticas geradas por computador a partir de modelos 3D dos objetos. Pretende-se avaliar a capacidade de localização e identificação de objetos do sistema *pick and place* e a sua capacidade em se deslocar corretamente até eles.

Pretende-se também minimizar o tempo decorrido desde a criação do conjunto de dados até o modelo estar completamente treinado, uma vez que o objetivo é combater a demora na resposta às necessidades do cliente. Como tal, um dos aspetos importantes a ter em consideração é a rapidez de treino das redes neuronais.

Por último, e do ponto de vista do autor, os objetivos são o estudo e aprendizagem de novas tecnologias e da sua integração em aplicações reais. Entre elas, destacam-se:

- Design em 3D: criação de modelos tridimensionais de objetos ou sistemas através de *softwares* de desenho. Esta habilidade é valiosa em diversas áreas, como a engenharia, arquitetura e produção de conteúdo para jogos e animações;
- IA: desenvolvimento de algoritmos e sistemas que possam realizar tarefas que normalmente exigem inteligência humana, como reconhecimento de imagem, processamento de linguagem natural e tomada de decisão automatizada;
- Comunicação industrial: configuração e integração de sistemas de automação e controle em ambientes industriais, permitindo a comunicação e troca de informações entre eles;
- Programação em tempo real: desenvolvimento de sistemas que possam processar e responder a eventos em tempo real, sem atraso percetível;
- Automatização de fluxos de trabalho: criação de processos automatizados que possam lidar com tarefas rotineiras e repetitivas, permitindo que as pessoas se concentrem em tarefas mais complexas e criativas.

1.2.2 Resultados esperados

Na presente dissertação é expectável alcançar quatro objetivos:

- Criação de um sistema *pick and place* funcional;
- Criação de um modelo funcional para localização e reconhecimento de objetos, baseado em redes neuronais e treinado com imagens sintéticas;
- Diminuição do intervalo de tempo entre iniciação do projeto e o sistema final;
- Automatização do processo de anotação das imagens de treino reais.

Apesar de não ser realista esperar que o funcionamento de um modelo treinado com imagens sintéticas apresente o mesmo desempenho de um modelo treinado com imagens reais, espera-se alcançar um sistema com um desempenho suficientemente bom para poder substituir um modelo treinado com imagens reais, quando este não se encontra disponível.

1.3 Organização da Dissertação

Esta dissertação está estruturada em seis capítulos.

No capítulo 1 é apresentado o contexto da presente dissertação, bem como o problema que se pretende abordar, os objetivos e os resultados esperados.

O capítulo 2 deste documento é dedicado à descrição de conceitos fundamentais na área da IA e visão computacional. Inicia-se com uma breve introdução à IA, seguida de uma abordagem à visão computacional e aprendizagem de máquina. Depois, é apresentada uma descrição detalhada das redes neuronais artificiais, incluindo os seus neurónios, camadas e processo de aprendizagem. São também exploradas as redes neuronais convolucionais, incluindo as suas camadas convolucionais, *pooling* e totalmente conectadas, bem como arquiteturas específicas para deteção de objetos, como o EfficientDet e o MobileNet. Outros tópicos abordados incluem *transfer learning*, *Compute Unified Device Architecture* (CUDA), *Mean Average Precision* (mAP) para detetores de objetos e geração de dados sintéticos. Este capítulo é uma base sólida para os conceitos fundamentais da IA e visão computacional.

No capítulo 3 é descrito todo o sistema desenvolvido. É abordada a modelação das ferramentas e a descrição do conjunto de dados utilizados para o treino do modelo. São apresentados cinco conjuntos de dados, bem como um conjunto de dados de validação. Além disso, é descrita a metodologia de treino do modelo, incluindo o *software* e *hardware* utilizados, o modelo pré-treinado e as adaptações nele feitas. Finalmente, o capítulo discute o sistema *pick and place*, incluindo o sistema de visão, comunicação e controlo. Este capítulo fornece, portanto, uma visão geral dos principais aspetos técnicos abordados neste estudo.

No capítulo 4 são apresentados os resultados obtidos com os diferentes modelos treinados, nomeadamente o desempenho de cada modelo no conjunto de dados de validação durante o processo de treino, bem como o desempenho de cada modelo já treinado num cenário real, juntamente com o desempenho do sistema *pick and place*.

No capítulo 5 são discutidos os resultados obtidos, são feitas comparações entre os vários modelos treinados e é avaliado o desempenho por classe dos modelos. É proposto e testado um método de automatização da aquisição do conjunto de dados e é feita uma análise das decisões tomadas ao longo deste projeto que permitiram reduzir o intervalo de tempo entre a criação do conjunto de dados sintético e o sistema final.

Finalmente, no capítulo 6, são analisadas as contribuições deste trabalho e perspetivado o possível trabalho futuro.

Capítulo 2

Enquadramento Teórico

A deteção e reconhecimento de objetos em imagens ou vídeos tem sido dominada, durante a última década, por algoritmos que utilizam RNC de última geração, devido à sua eficácia no reconhecimento e classificação de imagens. O próximo capítulo introduz os conceitos teóricos fundamentais para a compreensão do que é uma RNC e dos diversos conceitos a ela associados.

2.1 Inteligência Artificial

Diversas áreas estão a investigar a IA [4], desde vertentes tecnológicas e de alto valor científico, até áreas de entretenimento como música ou filmes. Em termos gerais, a IA refere-se ao estudo de métodos e dispositivos capazes de perceber e agir num ambiente, com o intuito de alcançar um objetivo específico, sem a intervenção humana.

A IA é um campo bastante amplo com diversos subcampos, tal como aprendizagem de máquina, visão computacional e aprendizagem profunda, tal como ilustrado na Figura 2.1

Nas próximas subsecções serão abordados cada um destes temas.

2.2 Visão Computacional

A visão computacional, apesar de ser um subcampo de IA, está muito ligada à aprendizagem de máquina, uma vez que recorre a um conjunto de técnicas, que

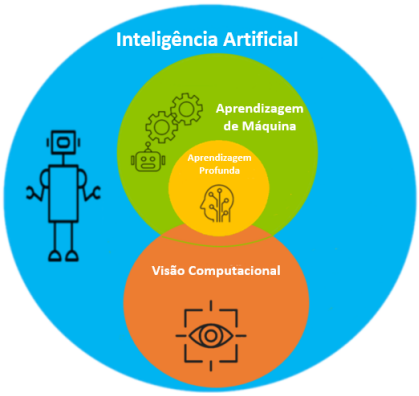


Figura 2.1: Subcampos da intelegência artificial, adaptado de [5].

permite extrair informações de dados multimédia, como imagens e vídeos, oferecendo a possibilidade de se obter uma ampla gama de informações. Dependendo do nível de detalhe da informação são definidas diferentes categorias de tarefas, tais como a classificação de imagem, a deteção de objetos, a segmentação, a segmentação semântica e a estimação e seguimento de um objeto [6]. Este subcapítulo descreve brevemente cada uma destas tarefas.

A classificação de imagem é a tarefa mais elementar, a qual consiste na deteção de um conjunto pré-determinado de objetos numa imagem [7]. A finalidade da

Figura 2.2: Exemplo de classificação de imagem [8].

classificação de imagens é responder à pergunta: "Que tipo de imagem é esta?". Na Figura 2.2 é possível observar um exemplo da classificação de imagem, neste caso, o exemplo apresentado define a imagem do tipo "Cão".

A deteção de objetos específicos de uma categoria, ou abreviando, deteção de objetos é a tarefa que consiste na identificação de um conjunto de objetos predefinidos contidos numa imagem e na estimação da sua posição 2D (2 dimensões). O modelo gera as coordenadas de uma caixa delimitadora retangular, que encapsula a posição do objeto na imagem [9]. Esta técnica é utilizada quando o tipo de objeto é conhecido e se pretende obter a sua posição na imagem. Na Figura 2.3 é apresentada uma imagem com uma caixa delimitadora à volta do objeto de interesse.

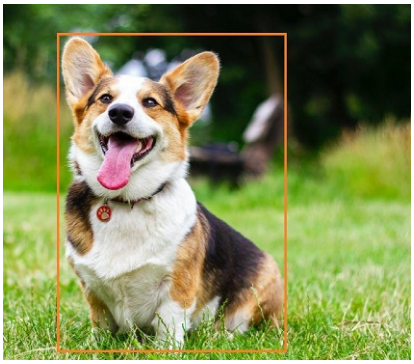


Figura 2.3: Exemplo de deteção de objetos [8].

Outra técnica importante na visão computacional é a segmentação de imagens, que consiste em agrupar os pixels da imagem em regiões que partilham características semelhantes. Esta técnica é especialmente útil para a deteção e reconhecimento de objetos na imagem, pois permite isolar as regiões que contêm um objeto de interesse e eliminar a interferência de outras regiões. Nos últimos 40 anos, diversos métodos de segmentação foram propostos, desde métodos tradicionais de visão computacional até métodos de aprendizagem profunda de última geração [10]. Na Figura 2.4 é apresentada um exemplo onde se aplicou a segmentação de imagem.

Figura 2.4: Exemplo de segmentação de imagem [8].

A segmentação semântica é um dos principais problemas no campo da visão computacional. De uma forma generalista, a segmentação semântica é uma das tarefas de alto nível que abre o caminho para a compreensão completa da cena. Nos dias de hoje, múltiplas aplicações recorrem à informação contida em imagens para tentar alcançar este objetivo [11].

A segmentação semântica engloba um conjunto de técnicas de processamento de imagem que permitem inferências detalhadas sobre cada pixel. Isto é realizado através da atribuição de um rótulo a cada pixel, indicando a classe de objeto à qual pertence. Desta forma, cada pixel é rotulado de acordo com a região da imagem a qual faz parte [13]. A Figura 2.5 mostra um exemplo de segmentação de imagem

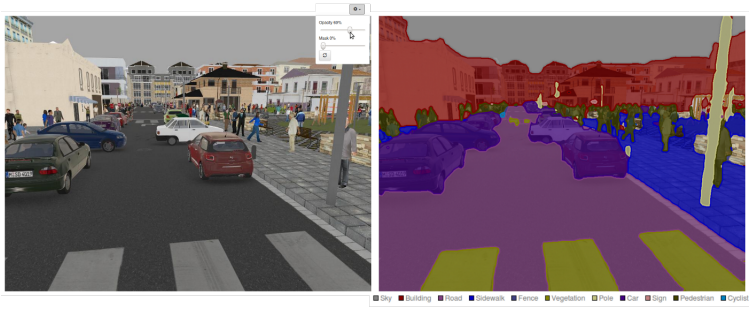


Figura 2.5: Exemplo de segmentação semântica [12].

onde diferentes objetos foram identificados.

A visão computacional inclui ainda as tarefas de estimação de pose, de movimento e seguimento de objetos, que envolvem a deteção, associação e rastreamento de pontos-chave em sequências de imagens. No entanto, o seguimento de pontos-chave em vídeos requer uma grande quantidade de recursos computacionais, o que pode limitar a precisão da estimativa de pose [6].

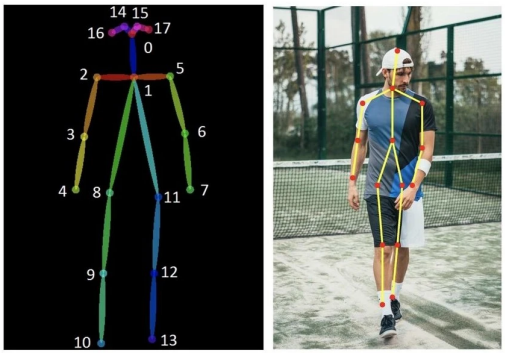


Figura 2.6: Exemplo de estimação de pose [14].

Atualmente, as RNC são a base dos modelos de processamento de imagem mais poderosos. Assim, os métodos de última geração são tipicamente baseados em arquiteturas de RNC, adaptadas especialmente para a estimação da pose humana, como ilustrado na Figura 2.6.

Todas as tarefas de visão computacional mencionadas anteriormente são atualmente realizadas com técnicas baseadas em RNC, que representam o estado da arte nesta área. Os próximos subcapítulos fornecerão os fundamentos teóricos necessários para entender o funcionamento das RNC.

2.2.1 Aprendizagem de máquina

A aprendizagem de máquina é um ramo da inteligência artificial que se concentra no desenvolvimento de algoritmos que podem aprender e fazer previsões ou decisões baseadas em dados. O objetivo da aprendizagem de máquina é construir modelos

que possam identificar automaticamente padrões e relações nos dados e usar essas informações para fazer previsões ou tomar decisões [15].

Existem diversos tipos de algoritmos de aprendizagem, como apresentado na Figura 2.7:

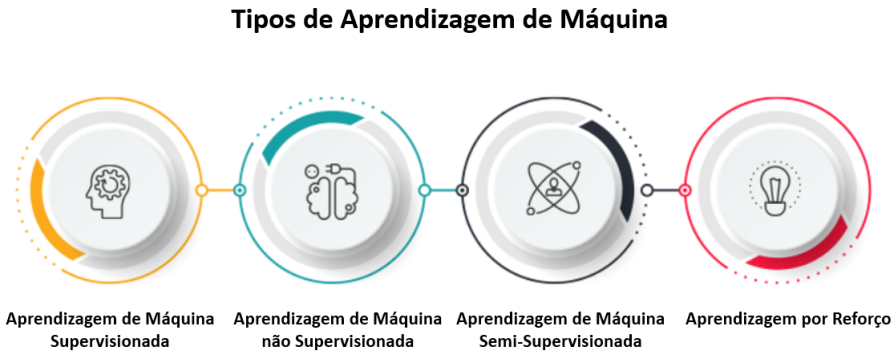


Figura 2.7: Diferentes algoritmos de aprendizagem de máquina, adaptado de [16].

Os algoritmos de aprendizagem supervisionada [17] são treinados usando dados rotulados, o que significa que o algoritmo recebe os dados de entrada e a saída correta para esses dados. De seguida, o algoritmo aprende a fazer previsões com base nos dados de treino. Por outro lado, algoritmos de aprendizagem não supervisionados [18] são treinados utilizando dados não rotulados e são usados para identificar padrões e relações nos dados sem qualquer conhecimento prévio. A combinação da aprendizagem supervisionada com a aprendizagem não supervisionada cria a aprendizagem semi-supervisionada [19]. Este tipo de algoritmos aprende com dados rotulados e não rotulados, para fazer previsões ou tomar decisões. Nesta abordagem, o algoritmo é fornecido com alguns dados rotulados, onde a saída correta para cada entrada já é conhecida e uma quantidade muito maior de dados não rotulados, onde a saída correta não está disponível. A ideia por trás da aprendizagem semi-supervisionada é que os dados rotulados podem fornecer pistas importantes sobre a estrutura subjacente dos dados, que podem ser usados para fazer previsões ou tomar decisões sobre os dados não rotulados. Aproveitando os dados rotulados e não rotulados, a aprendizagem semi-supervisionada pode obter melhor desempenho do que os algoritmos tradicionais de aprendizagem supervisionada que dependem apenas de dados rotulados. Por fim, os algoritmos de aprendizagem por reforço [20] aprendem por tentativa e erro e são frequentemente usados para ensinar as máquinas a realizar diferentes tipos de tarefas.

Uma das principais vantagens da aprendizagem de máquina é que esta pode ser usada para fazer previsões ou tomar decisões com base em grandes quantidades de dados. Por exemplo, a aprendizagem de máquina pode ser usada na área da saúde para analisar grandes conjuntos de dados de informações do paciente e prever quais os tratamentos que serão, provavelmente, mais eficazes para determinado paciente.

No entanto, a aprendizagem de máquina possui limitações. Um dos maiores desafios é garantir que os dados usados para treinar o algoritmo sejam uma representação do mundo real. Se os dados de treino forem tendenciosos ou incompletos, o modelo resultante pode não ser preciso ou produzir previsões tendenciosas. Além disso, os modelos de aprendizagem de máquina podem ser difíceis de interpretar, o que pode dificultar a compreensão de como um modelo está a fazer as suas previsões ou a tomar decisões.

2.3 Redes Neuronais Artificiais

As redes neuronais artificiais [21] são um tipo de algoritmo de aprendizagem de máquina inspirado pelo cérebro humano. Consistem em camadas de nós, designados por neurónios, que processam dados de entrada para produzir resultados de saída. A estrutura e as conexões dos neurónios dentro da rede podem ser ajustadas por treino, o que envolve alimentar a rede com dados rotulados e ajustar os pesos das conexões entre os neurónios para otimizar a precisão da saída. As redes neuronais artificiais têm apresentado um sucesso notável em diversas aplicações, incluindo reconhecimento de imagem, processamento de linguagem natural e reconhecimento de fala, tornando-as uma ferramenta poderosa para análise de dados e tomada de decisões. Nas próximas secções serão abordados os componentes básicos de uma rede neuronal artificial.

2.3.1 Neurónio

Para a identificação de um determinado objeto, por exemplo um alicate, o ser humano analisa as suas características como o comprimento, a largura, a forma e a cor para determinar se o objeto em causa é, ou não, um alicate. Para um sistema de aprendizagem de máquina, será necessário atribuir um valor quantitativo para decidir se estas características, extraídas pelo sistema, correspondem a um alicate. Contudo, diferentes caraterísticas possuem importâncias diferentes. Assim, um alicate pode variar muito em comprimento, largura e cor, porém a sua forma permanece bastante constante. Isto traduz-se numa atribuição de pesos às diferentes variáveis (neste caso o peso atribuído à variável "forma"será superior às outras). Então, para o sistema decidir se o objeto é um alicate ou não, atribui diferentes pesos às variáveis, multiplica-os pelos valores das variáveis de entrada, soma o resultado de cada multiplicação e, por fim, passa o resultado por uma função, designada função de ativação, que converte o resultado numérico num resultado binário [22].

Traduzindo a Figura 2.8 por uma equação matemática obtém-se:

$$y = f(\sum_{i=0}^{n} x_i \times w_i) \tag{2.1}$$

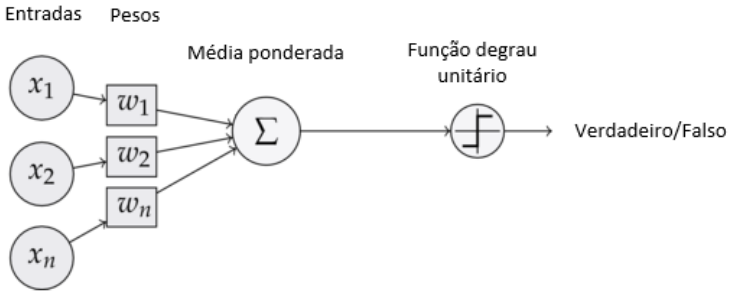


Figura 2.8: Possível modelo de classificação, adaptado de [23].

onde $f(.)$, a função de ativação, é uma função Heaviside, ou seja:

$$f(y) = \begin{cases} 0, \text{ se } y < 0 \\ 1, \text{ se } y \geq 0 \end{cases} \tag{2.2}$$

Para facilitar ou dificultar a ativação da saída é necessário adicionar, a este sistema, um termo de *bias* antes da função de ativação.

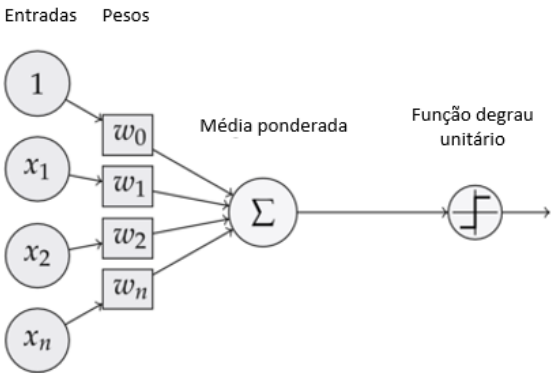


Figura 2.9: Perceptrão, adaptado de [23].

Apesar do conceito do perceptrão ter sido introduzido com uma função Heaviside como função de ativação, muitas vezes pretendem-se valores não binários, por exemplo, a probabilidade da deteção efetuada estar correta. Para tal, recorre-se a funções não lineares como funções de ativação.

Na Figura 2.10 estão representadas as funções de ativação mais usadas. À esquerda encontra-se uma sigmoide, no centro uma tangente hiperbólica e à direita uma unidade linear retificada, ou em inglês *Rectified Linear Unit* (RELU).

Então, o perceptrão pode ser definido como uma unidade que consiste na soma dos produtos dos valores de entrada pelos seus respetivos pesos, agregado a um determinado *bias* e a uma função de ativação que tem como saída apenas um único valor. Esta é a unidade mais elementar de uma rede neuronal e é conhecida também por neurónio [22].

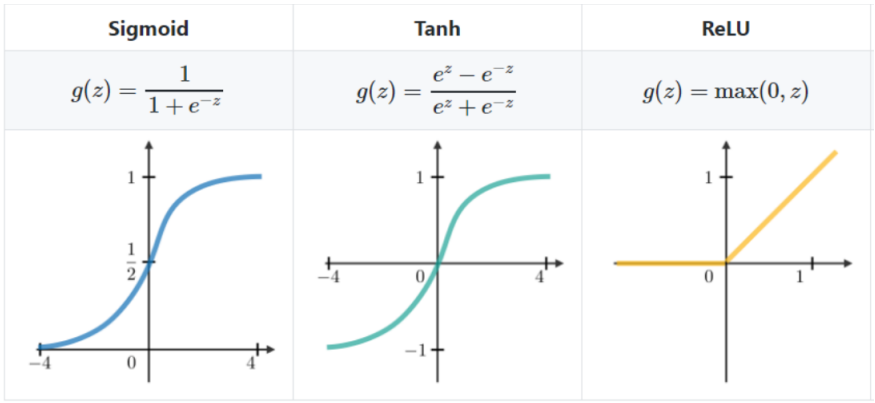


Figura 2.10: Exemplos de funções de ativação [24].

2.3.2 Camada

Com a unidade mais elementar de uma rede neuronal explorada, torna-se mais evidente o conceito de camada. Uma camada é um conjunto de neurónios agrupados no mesmo nível, onde todos possuem a mesma entrada. Se um conjunto de camadas for agrupado sequencialmente, o resultado é uma rede neuronal. As redes neuronais são constituídas por três tipos de camadas [25]:

- Camada de entrada: Recebe os valores de entrada e transfere-os para as camadas seguintes;
- Camada de saída: Fornece a previsão final da rede;
- Camadas ocultas: Encontram-se entre a camada de entrada e a camada de saída e, dependendo das entradas que cheguem a estas camadas, podem, ou não, ser atuadas, o que se traduz no envio, ou não, de informação para as camadas seguintes.

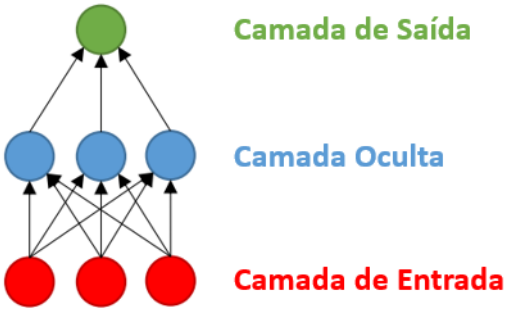


Figura 2.11: Estrutura de uma rede neuronal, adaptado de [25].

Na Figura 2.11 é apresentado um exemplo de uma rede neuronal. É de salientar que todos os neurónios possuem os seus próprios pesos e *bias*, caso contrário todos

os neurónios presentes numa camada seriam atuados com a mesma entrada, o que não é pretendido.

Para redes neuronais dedicadas à classificação de imagem, os dados de entrada vão ser imagens e a saída será, por exemplo, a probabilidade de uma imagem pertencer a determinada classe [26]. Como tal, o número de pixels que uma imagem possui define o número de neurónios que a entrada possui e o valor de ativação de cada neurónio é definido pelo valor numérico do pixel que lhe é atribuído. Isto significa que todas as imagens usadas para o treino da rede têm de possuir o mesmo tamanho. Já o número de neurónios de saída é definido pelo número de classes definidas. Assim, o valor de ativação de cada neurónio da camada de saída indica a probabilidade da imagem pertencer à classe correspondente.

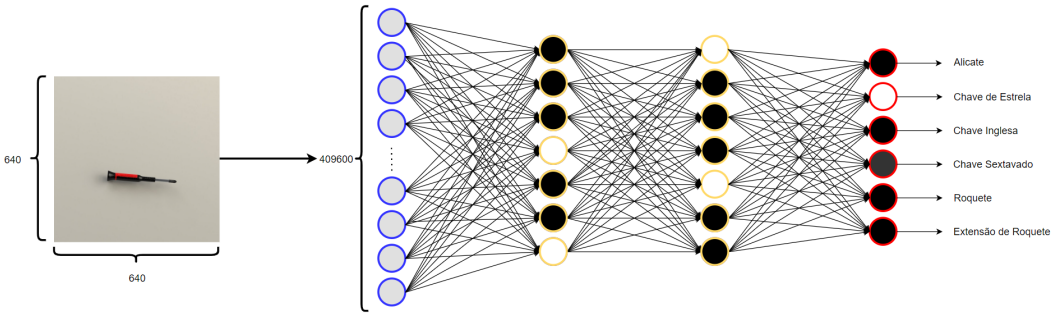


Figura 2.12: Exemplo de uma rede neuronal para classificação de ferramentas, adaptado de [26].

Na Figura 2.12 é ilustrada uma rede neuronal para classificação de ferramentas, sendo as possíveis saídas alicate, chave de estrela, chave inglesa, chave sextavado, roquete e extensão de roquete.

2.3.3 Deep Learning

A profundidade da rede é determinada pelo número de camadas ocultas. Se houver mais de duas camadas ocultas, a rede é considerada uma rede neuronal profunda, sendo que quanto maior o número de camadas, mais profunda a rede será.

Em redes neuronais profundas, cada camada oculta é como um "filtro"que processa a informação que vem da camada anterior, transformando e extraindo novas características dos dados. Cada camada oculta é capaz de reconhecer padrões mais complexos e abstratos que a camada anterior, criando uma hierarquia de representações que se vão tornando mais sofisticadas à medida que as camadas vão ficando mais profundas. Esta capacidade de extrair e aprender características mais complexas é o que torna as redes neuronais profundas tão poderosas na resolução de problemas com dados não lineares, ou seja, dados que não podem ser facilmente descritos por

fórmulas matemáticas simples. Ao processar e transformar os dados através de múltiplas camadas ocultas, a rede neuronal é capaz de identificar relações e padrões mais sofisticados e assim fazer previsões ou tomar decisões mais precisas [27].

A aprendizagem profunda apresenta duas vantagens relativamente a outras técnicas de aprendizagem de máquina. Uma das vantagens da aprendizagem profunda é que o seu desempenho pode continuar a melhorar com a adição de mais dados de treino, enquanto os modelos de aprendizagem de máquina acabam por atingir um ponto de saturação onde o desempenho não é afetado pela adição de novos dados. A outra vantagem é a capacidade que a aprendizagem profunda possui de selecionar automaticamente as características importantes dos dados, enquanto que na aprendizagem de máquina estas têm que ser selecionadas manualmente, como ilustrado na Figura 2.13.

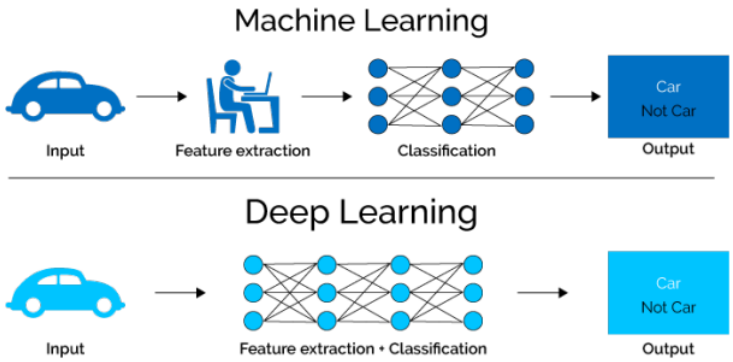


Figura 2.13: Diferença entre aprendizagem de máquina e Deep Learning no processo de extração de recursos [28].

.

2.4 Processo de aprendizagem de uma Rede Neuronal

Quando uma rede neuronal não treinada é alimentada com dados de entrada como, por exemplo, uma imagem de uma ferramenta, a sua saída não será correta. Visto que os pesos e os *bias* utilizados no processo de tomada de decisão possuirão valores aleatórios, a ativação dos neurónios nas camadas ocultas será aleatória e, portanto, os neurónios na camada de saída também serão aleatórios. Assim, para uma rede neuronal aprender, é necessário um algoritmo que a alimente com uma grande quantidade de dados de treino, que no caso de redes de classificação de imagens consiste em imagens e as suas respetivas anotações de classe [29, 30]. Estes dados de treino são usados para ajustar os pesos e os *bias* da rede, tornando-a capaz de produzir saídas mais precisas e, portanto, realizar a tarefa para a qual foi projetada [26]. No entanto, antes de entrar em detalhe sobre este algoritmo é necessário, primeiramente, compreender alguns conceitos.

2.4.1 Função de custo

Utilizando o exemplo apresentado em [26] e recorrendo à rede representada na Figura 2.12, a camada de saída pode ser representada pelo vetor $\vec{v}$, onde cada posição do vetor é a ativação de cada neurónio de saída, mais concretamente, a probabilidade de determinada imagem corresponder a cada uma das classes existentes. Para este exemplo, em que a entrada é uma imagem de uma chave de estrela, o vetor $\vec{v}_d$ é o vetor que contém o valor da saída desejada, ou seja o *Ground Truth*. No entanto, para uma rede treinada incorretamente, o vetor de saída poderá ser, por exemplo, o vetor $\vec{v}$:

$$\vec{v} = \begin{bmatrix} 0.92 \\ 0.27 \\ 0.03 \\ 0.75 \\ 0.23 \\ 0.15 \end{bmatrix}, \vec{v}_d = \begin{bmatrix} 0.00 \\ 1.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \end{bmatrix} \tag{2.3}$$

Em 2.3, o primeiro elemento do vetor $\vec{v}$ é a probabilidade da imagem ser um alicate, o segundo elemento é a probabilidade de ser uma chave de estrela, o terceiro é a probabilidade de ser uma chave inglesa, o quarto é a probabilidade de ser uma chave sextavado, o quinto é a probabilidade de ser um roquete e o sexto é a probabilidade da imagem ser uma extensão de roquete. Como previamente mencionado, a rede foi alimentada com uma imagem de uma chave de estrela e, devido ao treino incompleto da rede, esta identificou a imagem como sendo de um alicate, com 92% de certeza.

Para treinar uma rede corretamente é necessário medir o desempenho do conjunto de dados de treino, ou seja, é necessário obter uma métrica que defina o bom funcionamento da rede com base nos resultados obtidos ($\vec{v}$) e nos resultados desejados ($\vec{v}_d$). O valor que mede a diferença entre a saída da rede e o resultado desejado é conhecido como função de custo ou perda. Existem várias funções de custo que podem ser utilizadas, dependendo do tipo de problema que se pretende resolver [29, 30]. A função mais comum utilizada nas redes neuronais é o erro quadrático médio que, para uma única imagem do conjunto de dados de treino, é dado por:

$$\text{custo}_0 = \sum_i (\vec{v}_i - \vec{v}_{di})^2 \tag{2.4}$$

O valor do custo na equação 2.4 é mais pequeno quando a rede neuronal identifica corretamente a imagem e é maior quando não o faz [26].

Como acima mencionado, existem múltiplas funções que podem ser escolhidas para além do erro quadrático médio, contudo todas as funções têm de satisfazer dois requisitos mínimos:

- A função não pode depender de nenhuma ativação de um neurónio que não seja de saída;
- Tem de ser possível ser escrita como uma média de todos os elementos do conjunto de dados de treino.

O segundo requisito pode ser traduzido por uma equação:

$$\text{custo} = \frac{1}{n_{\text{imagens}}} \sum_{k=0}^{n_{images}-1} (custo_k) \tag{2.5}$$

Os parâmetros que influenciam os resultados da classificação são os pesos e os *bias* da rede. São estes os parâmetros que definem o bom ou mau funcionamento da rede, e que necessitam de ser ajustados para minimizar o valor do custo [26, 31, 25]. Para tal, ir-se-á recorrer a uma função que tem como entrada todos os pesos e *bias* da rede neuronal e tem como saída o valor do custo. Esta função, denominada de função de custo, tem como base todas as imagens do conjunto de dados de treino e pode ser representada por:

$$C(w_1, w_2, ..., w_n) \tag{2.6}$$

onde n é o número de pesos e *bias*. No próximo subcapítulo irá ser discutido como descobrir a combinação de pesos e *bias* para minimizar o custo.

2.4.2 Descida de Gradiente

Para qualquer rede neuronal, a função de custo irá ter um elevado número de entradas. Usando, como exemplo, a rede neuronal representada na Figura 2.12, note-se que na camada de entrada estão presentes 409600 neurónios (derivado da dimensão das imagens de entrada). Para além disto, cada camada oculta possui 7 neurónios e a camada de saída possui 6 neurónios, o que dá um total de 2867291 pesos e 20 *bias*. Por outras palavras, a função de custo irá ter 2867311 entradas. A Figura 2.12 representa uma rede neuronal conceptual pequena e simples, contudo redes neuronais profundas costumam ter milhões de pesos e *bias*. Neste subcapítulo irá ser explorado o método para encontrar o valor mínimo da função de custo, designado por descida de gradiente [26].

Para um melhor entendimento do que é a descida de gradiente, irá ser usada uma função de custo com apenas uma entrada, tal como apresentado na Figura 2.14.

O processo inicia-se com um valor de entrada aleatório e, para minimizar a saída da função de custo, é necessário estudar a direção para onde se deve deslocar. A função de custo visa o valor mínimo à saída com base na variação da entrada. Tendo em conta a Figura 2.14, se o declive (visto como a linha vermelha na imagem) da função for negativo num determinado ponto, então a entrada tem de se deslocar

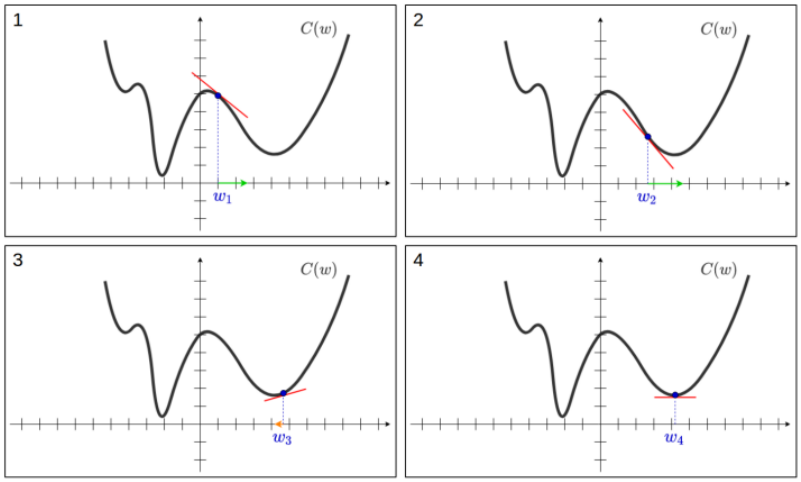


Figura 2.14: Os painéis 1 a 4 mostram uma sequência do processo iterativo da procura do valor mínimo da função. Primeiro, mede-se o declive num ponto inicial aleatório, $w1$. Se o declive for negativo, a entrada é deslocada para a direita (seta verde) e, se o declive for positivo, a entrada é deslocada para a esquerda (seta laranja). Este processo é repetido até o valor mínimo ser encontrado, onde o declive é 0, como se mostra no painel 4 [26].

para a direita(seta verde no primeiro painel). Caso o declive seja positivo, é necessário deslocar a entrada para a esquerda (representado pela seta laranja no terceiro painel). Também é possível ajustar o tamanho do deslocamento. Se o tamanho do deslocamento for proporcional ao declive perde-se o risco de *overshoot* [32]. Este processo é repetido até ser encontrado um ponto onde o declive irá ser nulo, o que significa que o valor mínimo foi encontrado. É de notar que este processo pode não encontrar o mínimo absoluto mas sim um mínimo local. Contudo, para as redes neuronais, isto não se revela um problema, visto que, normalmente, o mínimo local consegue apresentar resultados mais do que aceitáveis [26].

Aumentando a complexidade usando uma função de custo com duas entradas, é possível utilizar um raciocínio semelhante ao apresentado para apenas uma entrada. Pode-se visualizar o espaço de entrada como um plano cartesiano xy, enquanto que a função de custo pode ser representada como uma superfície acima do espaço de entrada. [26]. Neste exemplo, os pontos de início serão aleatórios, mas em vez de calcular o declive, determina-se em que direção, no espaço de entrada, é que o ponto tem de se deslocar de forma a diminuir o valor da função o mais rapidamente possível. É aqui que entra o gradiente da função. Na matemática, o gradiente de uma função de várias variáveis é um vetor que indica em que direção é a subida mais acentuada da função num ponto específico. Mais concretamente, é um vetor de derivadas parciais da função com respeito a cada variável de entrada. Ou seja, indica a direção que a função se deve deslocar para obter o valor máximo possível na saída. Por outras palavras, indica em que direção fica o aumento com maior declive, logo o

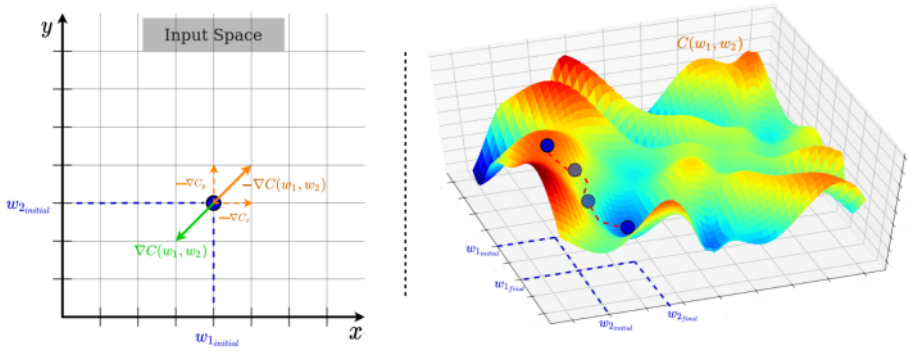


Figura 2.15: Exemplo de como encontrar o valor mínimo de uma função de duas entradas. À esquerda: O espaço de entrada. Direita: Função de custo com o caminho desde o ponto inicial até ao mínimo local [26].

negativo do gradiente indica a direção do declive mínimo. Resumindo, o algoritmo que encontra o valor mínimo da função de custo pode ser descrito por:

1. calcular o gradiente $\nabla C(w_1, w_2)$;
2. Fazer um pequeno ajuste em direção a $-\nabla C(w_1, w_2)$;
3. Continuar a executar os passos anteriores para os pesos e *bias* atualizados até que o valor mínimo seja alcançado.

A descida de gradiente é o processo de ajustar, iterativamente, os parâmetros do modelo para minimizar o erro e, consequentemente, a função de custo, usando a direção e a magnitude da descida mais acentuada dada pelo cálculo do gradiente [33]. É importante destacar que o vetor do gradiente tem uma componente para cada entrada e é a combinação dos vetores nas direções x e y, como ilustrado no gráfico da esquerda da Figura 2.15.

A lista de ajustes para cada entrada é o inverso do vetor de gradiente. Se os valores de todos os pesos e *bias* da rede estiverem num vetor $\vec{W}$, então o vetor $\Delta C(\vec{W})$, com o mesmo número de elementos, indicará como cada peso ou *bias* correspondente em $\vec{W}$ deve ser ajustado, e pode ser visto como [26]:

$$\vec{W} = \begin{bmatrix} w_1 \\ b_1 \\ \vdots \\ w_n \\ b_n \end{bmatrix}, \quad \nabla C(\vec{W}) = \begin{bmatrix} \frac{\delta C}{\delta w_1} \\ \frac{\delta C}{\delta b_1} \\ \vdots \\ \frac{\delta C}{\delta w_n} \\ \frac{\delta C}{\delta b_n} \end{bmatrix} \tag{2.7}$$

onde o sinal de cada valor em $\nabla C(\vec{W})$ indica se a entrada correspondente em $\vec{W}$ tem de subir ou descer e a magnitude indica quais das variações nas mudanças irão

ter um impacto maior no custo. Uma iteração na direção do inverso do gradiente da função traduz-se numa atualização dos pesos e *bias* e pode ser traduzida pela seguinte equação:

$$\vec{W}_{novo} = \vec{W} - \eta \nabla C(\vec{W}), \tag{2.8}$$

onde η é uma constante conhecida como taxa de aprendizagem ou *learning rate*, que determina a magnitude das iterações na descida do gradiente. Por outras palavras, define o comprimento dos vetores nas Figuras 2.14 e 2.15.

O objetivo do algoritmo de descida de gradiente é ajustar os pesos e os *bias* da rede para minimizar a função de custo. Isto significa que o vetor $\vec{v}$ na equação 2.3 vai-se aproximar do vetor $\vec{v}_d$ e, como resultado, a rede irá classificar os dados de treino eficientemente. Como o custo envolve a média de todos os dados no conjunto de treino, como visto na equação 2.5, ao minimizar esta média implica uma melhoria no desempenho da rede em todos os dados [25, 26, 33].

Dada a vasta extensão dos dados usados para treinar, podendo chegar aos milhões de imagens, o algoritmo de descida de gradiente torna-se computacionalmente muito pesado, uma vez que precisa de adicionar a influência de cada imagem individual em cada iteração da descida de gradiente. Para evitar este problema, recorre-se a uma variação do algoritmo de descida de gradiente chamado descida de gradiente de mini lote ou *Mini Batch Gradient Descent* [25].

No *Mini Batch Gradient Descent*, os dados de treino estão divididos em mini lotes, cada um contendo uma pequena porção dos dados de treino. Neste caso, cada iteração da descida de gradiente passa a usar as imagens contidas na pequena porção guardada em cada mini lote em vez de usar o conjunto de dados inteiro. Uma vez que este método utiliza uma quantidade de imagens muito reduzida por iteração, quando comparada com o algoritmo de descida de gradiente tradicional, é dado que cada iteração não vai ser tão eficaz no que toca ao decréscimo do custo da função. Contudo, um mínimo local vai ser encontrado significativamente mais rápido do que pela descida de gradiente básica.

2.4.3 Retropropagação

A retropropagação é o algoritmo responsável por otimizar os pesos e os *bias* de uma rede neuronal através do cálculo do gradiente com respeito a função de custo da rede. Para a explicação do funcionamento deste algoritmo irá ser usada a Figura 2.12. Como mencionado previamente, no caso de uma rede não ter sido devidamente treinada, quando alimentada uma imagem de uma chave de estrela, as ativações na camada de saída podem ser semelhantes às expressas no vetor $\vec{v}$ na equação 2.3. Estas ativações têm de ser acertadas, pois uma vez que não podem ser mudadas diretamente, é necessário alterar os pesos e o *bias* associados a cada neurónio. É

igualmente fundamental saber quais os ajustes a fazer na camada de saída, representado pelas setas na Figura 2.16. Como a classificação desejada é "Chave de Estrela", o pretendido é aumentar o valor no neurónio correspondente, enquanto o valor dos restantes neurónios diminui [26].

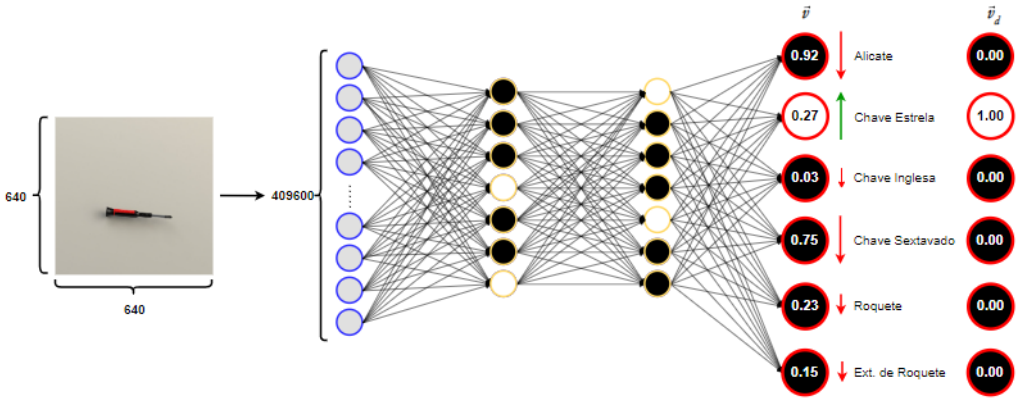


Figura 2.16: Rede neuronal alimentada com uma imagem de uma chave de estrela. À direita, estão presentes os valores obtidos, a forma como é desejado que estes se ajustem e o valor ideal (Adaptado de [26]).

Analisando o neurónio da camada de saída que devia ser ativado, mais concretamente o que diz respeito à chave de estrela, a ativação é dada por:

$$a^{(L)}_{estrela} = g(\sum w^{(L)}_i a^{(L-1)}_i + b^{(L)}_{estrela}) \tag{2.9}$$

onde:

- $a^{(L-1)}_i$ é a ativação da camada anterior;
- $w^{(L)}_i$ são os pesos;
- $b^{(L)}_{estrela}$ é o *bias*;
- L é uma indicação da camada à qual o neurónio pertence;
- g é a função de ativação.

O objetivo é perceber como é que os pesos, o *bias* e a ativação da camada anterior mudam de forma a aumentar a ativação do neurónio correto [26, 31]. Para facilitar a explicação recorrer-se-á à função do erro quadrático médio para a medida de custo.

Começando com os pesos da equação 2.4, o custo para uma única imagem $C_0(\vec{W})$, focando apenas no neurónio responsável pela deteção da "Chave de Estrela", é dado por:

$$C_0(\vec{W}) = (a^{(L)}_{estrela} - y)^2 + ... \tag{2.10}$$

onde y é o valor correto da saída deste neurónio. Depois, se a soma ponderada e o *bias* forem chamados de $z^{(L)}_{estrela}$:

$$z^{(L)}_{estrela} = \sum_i w^{(L)}_i a^{(L-1)}_i + b^{(L)}_{estrela} \tag{2.11}$$

então, a ativação é dada por:

$$a^{(L)}_{estrela} = g(z^{(L)}_{estrela}) \tag{2.12}$$

O objetivo é perceber o impacto de uma pequena alteração num único peso ligado a um neurónio, isto é $\partial w^{(L)}_i$, no custo C_o:

$$\frac{\partial C_0}{\partial w^{(L)}_i} \tag{2.13}$$

Uma vez que uma mudança em $\partial w^{(L)}_i$ implica uma mudança em $z^{(L)}_{estrela}$ (equação 2.11), o que por si só implica uma variação em $a^{(L)}_{estrela}$ (equação 2.12), o que no final irá causar uma mudança em C_0 (equação 2.4), a seguinte regra da corrente (*chain rule*) pode ser escrita:

$$\frac{\partial C_0}{\partial w^{(L)}_i} = \frac{\partial z^{(L)}_{estrela}}{\partial w^{(L)}_i} \cdot \frac{\partial a^{(L)}_{estrela}}{\partial z^{(L)}_{estrela}} \cdot \frac{\partial C_0}{\partial a^{(L)}_{estrela}} \tag{2.14}$$

Calculando as derivadas do segundo termo da equação anterior, obtém-se:

$$\frac{\partial C_0}{\partial a^{(L)}_{estrela}} = 2(a^{(L)}_{estrela} - y) \tag{2.15}$$

$$\frac{\partial a^{(L)}_{estrela}}{\partial z^{(L)}_{estrela}} = g'(z^{(L)}_{estrela}) \tag{2.16}$$

$$\frac{\partial z^{(L)}_{estrela}}{\partial w^{(L)}_i} = a^{(L-1)}_i \tag{2.17}$$

e substituindo na equação 2.14 obtém-se:

$$\frac{\partial C_0}{\partial w^{(L)}_i} = a^{(L-1)}_i g'(z^{(L)}_{estrela}) 2(a^{(L)}_{estrela} - y) \tag{2.18}$$

Analisando as equações 2.15 e 2.18 é evidente que a mudança no custo é proporcional à diferença entre o valor de ativação atual e o valor desejado. Se a ativação de um neurónio estiver mais afastada do que o pretendido, mesmo pequenos ajustes nos pesos conectados a esse neurónio terão um impacto significativo na função de custo. Analisando a equação 2.17, conclui-se que a quantidade de variação criada

por uma pequena mudança num peso, que influencia a ativação de um neurónio, depende do valor de ativação do neurónio da camada anterior a que está ligado [26].

O método para determinar como o *bias* de um neurónio deve ser alterado é idêntico ao método para os pesos. A única diferença é que na equação 2.14 o termo $\partial w_i^{(L)}$ é substituído por $\partial b_{estrela}^{(L)}$, ou seja:

$$\frac{\partial C_0}{\partial b_{estrela}^{(L)}} = \frac{\partial z_{estrela}^{(L)}}{\partial b_{estrela}^{(L)}} \cdot \frac{\partial a_{estrela}^{(L)}}{\partial z_{estrela}^{(L)}} \cdot \frac{\partial C_0}{\partial a_{estrela}^{(L)}} \tag{2.19}$$

e como a derivada parcial de $z_{estrela}^{(L)}$ com respeito a $b_{estrela}^{(L)}$ é 1, a equação da regra de cadeia torna-se:

$$\frac{\partial C_0}{\partial b_{estrela}^{(L)}} = g'(z_{estrela}^{(L)}) 2(a_{estrela}^{(L)} - y) \tag{2.20}$$

Outra forma da saída do neurónio relacionado com a "Chave de Estrela"aumentar é alterando a ativação dos neurónios nas camadas anteriores [34]. Embora não seja possível alterá-las diretamente, é importante registar como devem ser alteradas, tal como mencionado anteriormente para os neurónios de saída. Assim, a forma de mudar a ativação num dado neurónio através do neurónio anterior, isto é, $a_i^{(L-1)}$, pode ser representada por:

$$\frac{\partial C_0}{\partial a_i^{(L-1)}} = \frac{\partial z_{estrela}^{(L)}}{\partial a_i^{(L-1)}} \cdot \frac{\partial a_{estrela}^{(L)}}{\partial z_{estrela}^{(L)}} \cdot \frac{\partial C_0}{\partial a_{estrela}^{(L)}} \tag{2.21}$$

e uma vez que :

$$\frac{\partial z_{estrela}^{(L)}}{\partial a_i^{(L-1)}} = w_i^{(L)} \tag{2.22}$$

a equação pode ser escrita como:

$$\frac{\partial C_0}{\partial a_i^{(L-1)}} = w_i^{(L)} g'(z_{estrela}^{(L)}) 2(a_{estrela}^{(L)} - y) \tag{2.23}$$

A partir das equações 2.21 e 2.23 pode concluir-se que a forma como o custo é afetado com pequenas mudanças na ativação dos neurónios $a_i^{(L-1)}$ é proporcional ao peso que os liga ao neurónio de saída "Chave de Estrela". Para minimizar a função de custo o mais rapidamente possível, é necessário que as ativações nos neurónios da camada anterior mudem proporcionalmente ao valor do peso ligado ao neurónio de saída. Além disso, se um neurónio na camada anterior estiver ligado por um peso negativo, a sua ativação deve diminuir, enquanto que a ativação dos neurónios ligados por pesos positivos deve aumentar. É importante notar que essas mudanças nas ativações não podem ser realizadas diretamente, mas sim através do ajuste dos pesos durante o processo de treino.

Note-se que este processo de alterações de pesos na camada anterior não é influenciado apenas pelo neurónio associado à "Chave de Estrela", pois a ativação dos restantes neurónios de saída afetam o custo, como observado na equação 2.4. Os neurónios que as ativações devem diminuir também têm um impacto, uma vez que são influenciados pelos valores de ativação da camada anterior. Logo, todos os impactos de todos os neurónios são somados, deixando a regra de cadeia para um neurónio na camada anterior como:

$$\frac{\partial C_0}{\partial a_i^{(L-1)}} = \sum_{j=0}^{n_L-1} (w_{ji}^{(L)} g'(z_j^{(L)}) 2(a_j^{(L)} - y_j)) \tag{2.24}$$

com n_L sendo o número de neurónios na camada L (neste caso trata-se da camada de saída) e w_{ji} são os pesos que relacionam o neurónio i da camada anterior ao neurónio da camada de saída j.

Quando a sensibilidade da função de custo em relação à ativação na camada anterior é descoberta, implica que existe uma medida para determinar como cada neurónio na camada anterior deve ser ajustado. Isto permite que o processo de cálculo e atualização dos pesos e dos *bias* seja repetido para as camadas anteriores, propagando o custo para trás. Este processo é conhecido como retropropagação [26, 34].

Então, a forma como um peso relacionado com um neurónio j na camada L ligado a um neurónio i na camada $L-1$ e o *bias* do neurónio j, de uma determinada rede com uma determina função de custo $C(\vec{W})$ e uma determinada função de ativação $g(z)$ deve mudar é dado por:

$$\frac{\partial C_0}{\partial w_{ji}^{(l)}} = a_i^{(l-1)} g'(z_j^{(l)}) . \frac{\partial C_0}{\partial a_j^{(l)}} \tag{2.25}$$

e,

$$\frac{\partial C_0}{\partial b_j^{(l)}} = g'(z_j^{(l)}) . \frac{\partial C_0}{\partial a_j^{(l)}} \tag{2.26}$$

onde, para os neurónios na camada de saída, $\frac{\partial C_0}{\partial a_j^{(l)}}$ é a derivada parcial da função de custo escolhida com respeito a $a_j^{(l)}$. O erro quadrático médio é dado por:

$$\frac{\partial C_0}{\partial a_j^{(l)}} = 2(a_j^l - y_j) \tag{2.27}$$

e:

$$\frac{\partial C_0}{\partial a_j^{(l)}} = \sum_{k=0}^{n_{l+1}-1} (w_{kj}^{(l+1)} g'(z_k^{(l+1)}) . \frac{\partial C_0}{\partial a_k^{(l+1)}}) \tag{2.28}$$

para os neurónios nas camadas ocultas.

A explicação dada até agora só teve em consideração uma única imagem, contudo, devido à forma como o custo real foi apresentado na equação 2.5, para o cálculo real dos parâmetros do vetor de gradientes, é necessário uma média sobre n imagens de treino no conjunto de dados [32]. Como tal, os parâmetros são dados por:

$$\frac{\partial C}{\partial w_{ji}^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w_{ji}^{(l)}} \tag{2.29}$$

$$\frac{\partial C}{\partial b_{j}^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial b_{j}^{(l)}} \tag{2.30}$$

Note-se que estes são os parâmetros de $\Delta C(\vec{W})$ como definido na equação 2.7, com uma notação diferente, concluindo-se assim a explicação do funcionamento do algoritmo de retropropagação e todos os passos e conceitos necessários para se compreender como é feito o treino de uma rede neuronal artificial [26].

2.4.4 Redes Neuronais Artificiais - Treino

O processo de treino de uma rede neuronal torna-se bastante simples, uma vez que os conceitos foram previamente explicados. Na sua essência, o treino consiste em quatro momentos [29]:

1. Passe para a frente (*forward pass*);
2. Computação do custo;
3. Passe para trás (*backward pass*);
4. Atualização dos pesos e dos *bias*.

A diferença entre os resultados da classificação e os resultados verdadeiros é a base do cálculo do custo. Para isto, os dados de treino são transferidos pela rede, para realizar a previsão da classificação dos resultados. Este processo denomina-se de passe para a frente. Com o custo calculado é possível, durante o passe para trás, calcular o vetor de gradientes ∇C usando o algoritmo de retropropagação.

Por fim, é possível atualizar os parâmetros da rede neuronal, ou seja, os pesos e os *bias*, utilizando um múltiplo do vetor de gradientes, conforme observado na equação 2.8 da subsecção de descida de gradiente. Repetindo estes quatro passos iterativamente, espera-se que a rede neuronal aprimore as suas classificações no conjunto de dados de treino [31, 34].

Se o processo de treino for muito extensivo, a rede pode começar a identificar padrões muito específicos nos dados de treino que não são necessariamente inerentes às classes do conjunto de dados, mas sim pequenas variações residuais ou ruídos que o modelo não consegue distinguir das caraterísticas realmente importantes [26].

E apesar da sua precisão aumentar nos dados de treino, a rede deixa de conseguir generalizar para novos dados que sejam diferentes dos dados de treino, o que resulta em baixa precisão para novos dados de entrada. Isto verifica-se ser um problema, uma vez que o objetivo do treino é fazer com que a rede seja capaz de generalizar o que aprendeu para novos dados. Quando o modelo ajusta os seus parâmetros muito especificamente para os dados de treino, desconsiderando caraterísticas de uma classe realmente relevantes, alcançando uma alta precisão nos dados de treino e uma baixa precisão em novos dados, é considerado um sobreajuste, ou *overfitting* [26]. Este *overfitting* pode acontecer por várias razões, tal como um conjunto de dados pequeno e insuficiente, um modelo excessivamente complexo e por excesso de treino.

Para além dos pesos e dos *bias* que são aprendidos autonomamente pela rede, o utilizador precisa de definir uma série de hiperparâmetros, como o número de iterações, o tamanho do lote e a taxa de aprendizagem, para que os resultados do treino sejam positivos [30]. Os hiperparâmetros relacionados com o treino são os que definem o comportamento do momento de treino, desde o número de iterações que este irá possuir, o comportamento do otimizador de custo durante o treino, etc. O tamanho do mini lote é o número de amostras alimentadas ao algoritmo de descida de gradiente em cada iteração. Este tem um impacto significativo durante o processo de treino. Lotes de dimensão maiores são computacionalmente mais exigentes, uma vez que é necessária mais memória para armazenar os efeitos de cada imagem no lote em cada iteração, tornando o processo mais lento. Por outro lado, um lote maior leva a atualizações mais precisas. Com lotes mais pequenos é possível chegar ao mínimo da função de custo de forma mais rápida, mas cada iteração da descida de gradiente não é tão precisa.

A duração do treino de uma rede neuronal é definida pelo número de iterações. Contudo, se o número de iterações for aleatório, para um modelo de deteção multi-classe, isto implica que cada classe não seja uniformemente treinada. Para enfrentar este problema, calcula-se quantas iterações são necessárias para que a rede seja alimentada com o conjunto de dados completo, exatamente uma vez. Este conceito é conhecido por época e, para calcular o número de iterações necessárias para completar uma época, deve-se dividir o tamanho do conjunto de dados pelo tamanho do lote [25]. Repare-se que a duração do treino deve ser regida pela quantidade de épocas. Por exemplo, se o conjunto de dados possuir 1000 amostras e o tamanho do lote for de 10, o número de iterações necessário para completar uma amostra é de 100. Continuando este raciocínio, se o treino tiver a duração pretendida de 11 épocas, o número de iterações de treino é 11x100=1100. Então, e reforçando o que foi previamente afirmado, o número de épocas tem de ser suficientemente baixo para evitar o sobreajuste, mas consideravelmente alto para garantir uma boa precisão.

Para a escolha de uma boa taxa de aprendizagem é importante não escolher um valor demasiado pequeno, uma vez que se traduz num processo de treino muito

lento. Por outro lado, uma valor demasiado alto pode resultar numa ultrapassagem do custo mínimo, ou até numa divergência na descida do gradiente, tornando assim impossível chegar a um custo mínimo [26].

2.5 Redes Neuronais Convolucionais

As redes neuronais normais, tais como as apresentadas previamente, conseguem alcançar bons resultados no que diz respeito às tarefas de classificação de imagem, contudo, quando os dados de entrada para classificação são imagens, estas tornam-se computacionalmente exigentes para ser usadas. Numa imagem de 640x640, num neurónio individual na primeira camada oculta existiriam 409600 conexões para os pesos com a camada de entrada. Se for considerado que a maioria das imagens possui um código de cores *Red*, *Green*, *Blue* (RGB) uma vez que as cores são codificadas pela combinação das cores vermelho, verde e azul, isto significa que as imagens são compostas por três canais distintos, o que implica que o número de pesos de um único neurónio seja três vezes maior. Em contrapartida, as RNC foram projetadas com a assunção de que os dados de entrada são imagens, sendo possível codificar algumas propriedades, o que as torna mais eficientes, reduzindo significativamente a quantidades de parâmetros da rede. Os volumes de neurónios com que estas redes trabalham estão organizados em largura, profundidade e altura. Cada camada numa RNC recebe como entrada volumes 3D de neurónios e transforma-os numa saída constituída por um novo volume 3D, como observado na Figura 2.17. A camada de entrada possui a mesma altura e a mesma largura que a imagem de entrada e a sua profundidade é definida pelo número de canais (três para imagens no espaço de cor RGB , um para imagens monocromáticas), com o valor de ativação de cada neurónio correspondente ao respetivo valor de cada pixel. A camada de saída contém um valor de saída por classe como visto previamente.

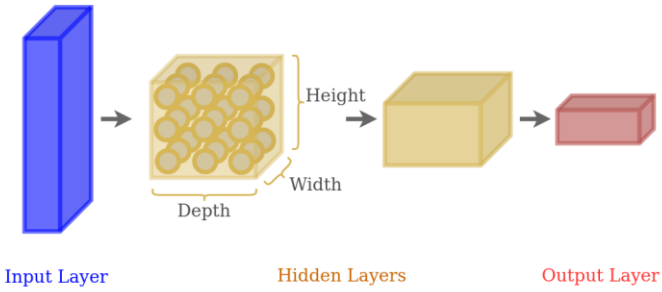


Figura 2.17: Representação de uma RNC [26].

As RNC são constituídas por duas partes essenciais: uma responsável pela extração de características e outra responsável pela classificação. Para cada tarefa

são utilizadas camadas especiais, como ilustrado na Figura 2.18. Como o nome indica estas redes recorrem à convolução e, na próxima subsecção, será abordado este conceito.

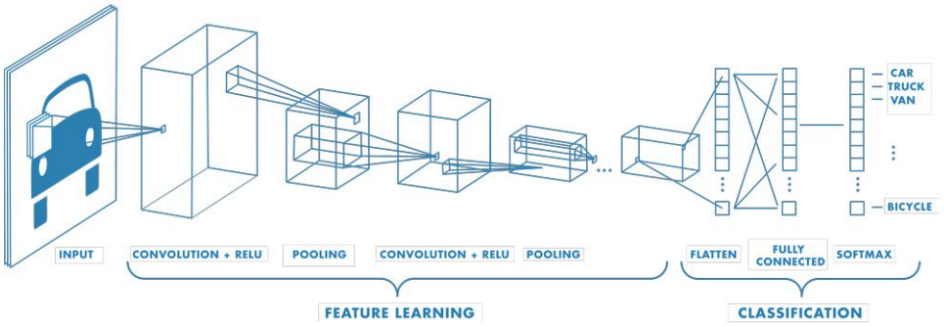


Figura 2.18: Exemplo de uma rede com múltiplas camadas de convolução. [35].

2.5.1 Convoluções

No que diz respeito ao processamento de imagens, uma convolução é uma operação que transforma uma imagem de entrada numa imagem diferente através de uma convolução matemática de imagens e um *kernel* que, neste contexto, é uma matriz ou filtro que é aplicado a uma imagem para realizar operações como *blurring* e deteção de bermas. Para tal, desloca-se a matriz do *kernel* sobre a imagem de entrada, que pode ser visto como uma matriz de valores de pixel, multiplicando os valores sobrepostos das duas matrizes e somando-o para gerar o valor de saída correspondente. Deslocando o *kernel* pela imagem, é gerada uma matriz de saída [36]. Esta operação pode ser melhor compreendida através da Figura 2.19.

Analisando a Figura 2.19 é notório que, depois da entrada ser convoluída com o *kernel*, o tamanho da imagem de saída diminui. Uma possibilidade para manter as dimensões da entrada é a aplicação de preenchimento zero através da adição de linhas e colunas de zeros (*zero padding*) nas bordas da imagem original, seguida da aplicação do *kernel* nesta entrada expandida. Por outro lado, se pretendido, também é possível reduzir o tamanho da imagem, através do movimento do *kernel*. O número de pixels que o *kernel* se desloca é chamado de passo e se for definido como um, o *kernel* desloca-se um pixel de cada vez, enquanto se for definido como dois, o *kernel* desloca-se dois pixels de cada vez [38], como ilustrado na Figura 2.20. É possível calcular o tamanho da imagem de saída através de:

$$A_s = [(A_e + 2 * P - K)/S + 1] \tag{2.31}$$

$$L_s = [(L_e + 2 * P - K)/S + 1] \tag{2.32}$$

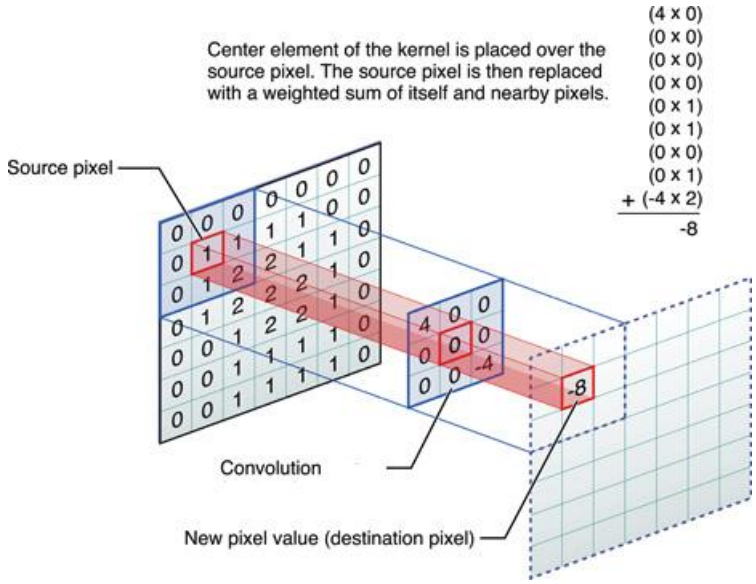


Figura 2.19: Exemplo do cálculo da convolução para um píxel de uma imagem [37].

onde,

- A_s: Altura da imagem de saída;
- A_e: Altura da imagem de entrada;
- L_s: Largura da imagem de saída;
- L_e: Largura da imagem de entrada;
- P: Preenchimento;
- K: Tamanho do kernel;
- S: Passo.

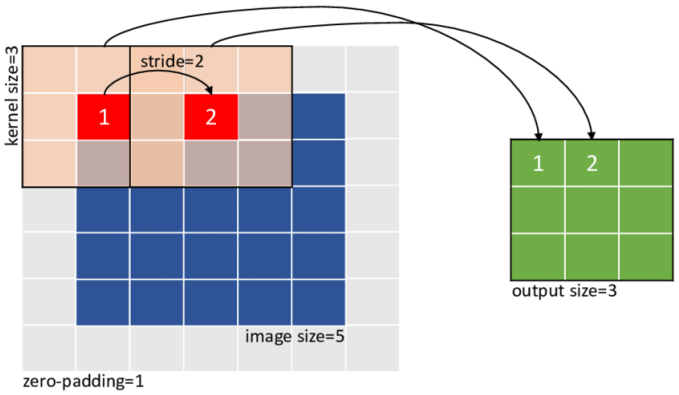


Figura 2.20: Exemplo de preenchimento de zeros e passo [39].

Diferentes *kernels* permitem realizar diferentes operações sobre a imagem, como por exemplo desfoque, nitidez, deteção de bordas e cantos, como ilustrado na Figura 2.21. Isto é o que torna esta operação tão importante para as redes neuronais convolucionais, uma vez que aplicar diferentes *kernels* numa imagem pode destacar diferentes características do objeto contido na imagem. Isto torna-se útil para distinguir objetos de diferentes classes [26].

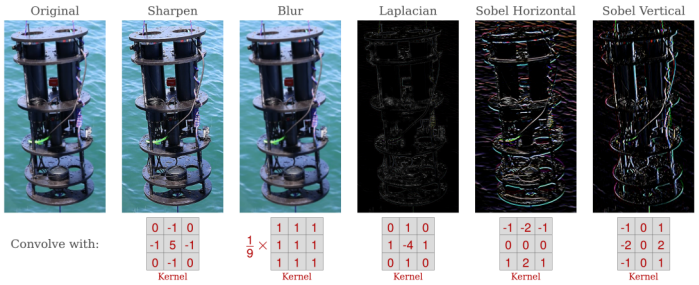


Figura 2.21: Resultado da convolução da mesma imagem com diferentes kernels [26].

2.5.2 Camadas numa RNC

Camadas Convolucionais

As camadas convolucionais são blocos importantes na construção de RNC para classificar imagens. Elas detetam características nas imagens de entrada, com recurso a conectividade local e à partilha de parâmetros. Cada neurónio na camada convolucional está conectado apenas a neurónios dentro de um pequeno volume com dimensões FxFxD1, onde F é designado de campo recetivo e D1 é a profundidade da camada anterior a que está conectado. O hiperparâmetro K é utilizado para definir a profundidade de uma camada convolucional, que corresponde ao número de filtros presentes na camada. Em outras palavras, todos os neurónios que ocupam a mesma largura e altura em profundidade - também chamados de coluna de profundidade ou fibra - estão conectados aos mesmos neurónios da camada anterior [40]. Este conceito é ilustrado na Figura 2.22.

Nesta figura, ao utilizar a conectividade local, um neurónio na camada convolucional introduz 27 pesos e um parâmetro de *bias*, sendo que a fórmula utilizada para calcular o número de pesos é $F \times F \times D1$. Por exemplo, se for considerada uma imagem de entrada com dimensões 640x640x3, um neurónio na camada convolucional ligado a ela com um filtro de tamanho três gera $27(3 \times 3 \times 3 = 27)$ pesos e um *bias*, em vez dos 1228800 pesos que seriam gerados se estivesse ligado de forma totalmente conectada. No entanto, a conectividade local por si só não reduz o número de parâmetros da rede o suficiente para que esta seja viável computacionalmente em arquiteturas reais. Embora cada neurónio possa introduzir menos conexões do que

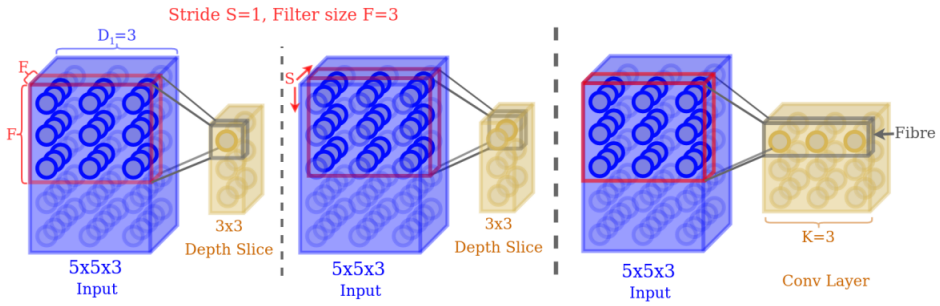


Figura 2.22: Neurónios com a mesma fatia de profundidade são conectados a regiões FxFxD_1 com um passo S ao longo da largura e altura da entrada [26].

se estivesse totalmente conectado à entrada, se a camada em que se encontra for composta por centenas de milhares de neurónios, ainda assim, serão introduzidos demasiados parâmetros [26].

Para reduzir a quantidade de parâmetros na rede, as camadas convolucionais utilizam a partilha de parâmetros, o que faz com que todos os neurónios na mesma profundidade tenham os mesmos pesos e *bias*. Visto isto, pode-se afirmar que as camadas convolucionais processam as imagens de entrada de maneira semelhante às convoluções de imagem. Na prática, uma camada convolucional realiza convoluções de imagem com os seus pesos, utilizando conjuntos de pesos chamados de filtros ou *kernels*. A saída da camada é chamada de mapa de recursos, e o número de filtros é determinado pelo hiperparâmetro K, como ilustrado na Figura 2.23 [26].

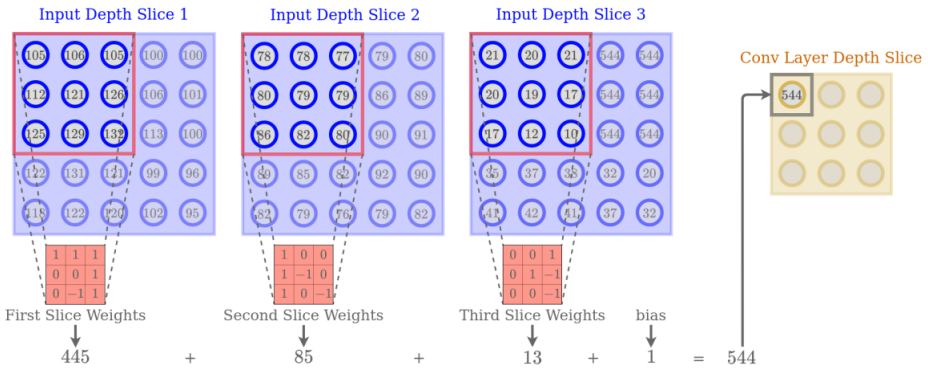


Figura 2.23: Visualização 2D do painel da esquerda da Figura 2.22 [26].

Durante o processo de treino da rede neuronal, cada camada de profundidade aprende um filtro tridimensional que é aplicado na imagem. Cada filtro pode ser treinado para detetar diferentes características, tais como linhas horizontais, verticais, bolhas de cores, entre outras, como ilustrado na Figura 2.24. A saída de

cada camada de profundidade é o resultado da convolução da imagem com o filtro aprendido [30].

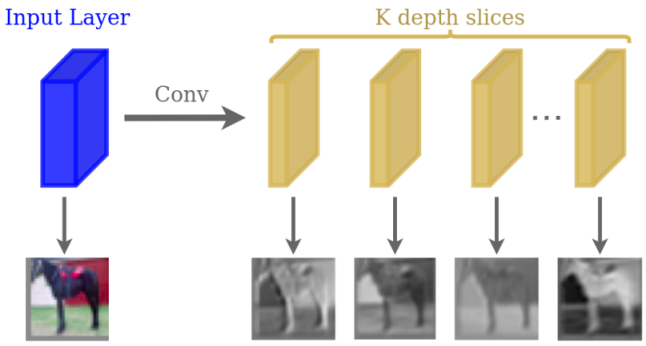


Figura 2.24: Cada fatia de profundidade numa camada de Convolução está a aprender um filtro tridimensional para convoluir com a entrada, e o seu mapa de ativação é o resultado da convolução [26].

A partilha de parâmetros tem como base a hipótese de que, se uma característica é importante numa determinada região da imagem, então essa mesma característica também é importante em diferentes regiões. Esta justificativa contribui para a eficácia da partilha de parâmetros.

Já em relação à função de ativação mais utilizada em RNC, a RELU é a mais comum. Esta escolha faz sentido, uma vez que, no caso das RNC, é esperado que as saídas sejam sempre não-negativas, dada a sua relação com as imagens. Na Figura 2.25 é possível confirmar que a função de ativação RELU é responsável por transformar todas as saídas negativas em 0, enquanto mantém os valores positivos inalterados. Isso explica a sua ampla utilização como função de ativação em RNC.

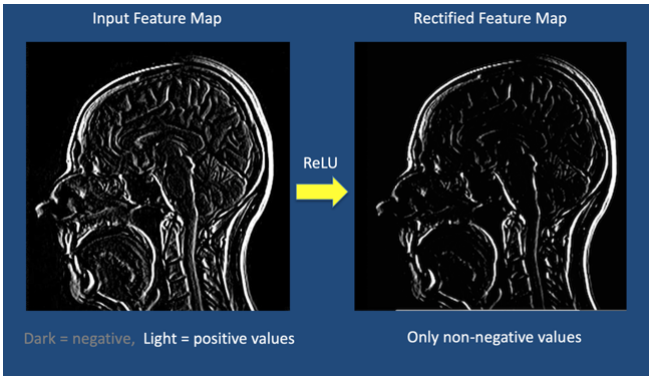


Figura 2.25: Efeito de uma unidade linear retificada num mapa de ativação, adaptado de [41].

Em resumo, as camadas convolucionais possuem quatro hiperparâmetros, sendo eles: o número de filtros K, o tamanho dos filtros F, o passo S e a quantidade de preenchimento de zeros P. Elas recebem como entrada um volume com dimensões $W_1 \times H_1 \times D_1$ e produzem uma saída com dimensões $W_2 \times H_2 \times D_2$, onde $W_2 =$

$(W_1-F+2P)/S+1$, $H_2=(H_1-F+2P)/S+1$ e $D_2=K$. A partilha de parâmetros permite a introdução de $(F \cdot F \cdot D_1) \cdot K$ pesos e K *bias*. Cada fatia de profundidade da saída é gerada pela convolução da entrada com o filtro aprendido para aquele filtro específico, usando um passo S e um deslocamento pelo *bias*. Como referido, a função de ativação mais utilizada é a RELU, que converte todas as saídas negativas para 0 e mantém os valores positivos inalterados [26].

Camadas de Pooling

As camadas de *pooling* são usadas para diminuir o tamanho espacial do volume de entrada, com o objetivo de reduzir tanto o número de parâmetros treináveis quanto o custo computacional da rede, o que também ajuda a controlar o sobreajuste, pois diminui a complexidade do modelo. Para isso, são aplicados filtros FxF de forma independente em cada fatia de profundidade do volume de entrada, com um valor de passo S definido. Os filtros de *pooling* são aplicados aos valores de entrada sobrepostos para gerar um valor por área sobreposta, reduzindo as dimensões dos mapas de recursos, mas mantendo as informações mais relevantes. Os dois tipos mais comuns de *pooling* são o *pooling* médio, que retorna o valor médio da área sobreposta e o *pooling* máximo, que retorna o valor máximo na referida área. O *pooling* máximo é o mais utilizado na prática [40]. Na Figura 2.26, é possível ver este processo utilizando max *pooling* com filtros 2x2 com um passo de dois sobre uma fatia de profundidade 4x4 e o resultado do *pooling* numa camada de entrada.

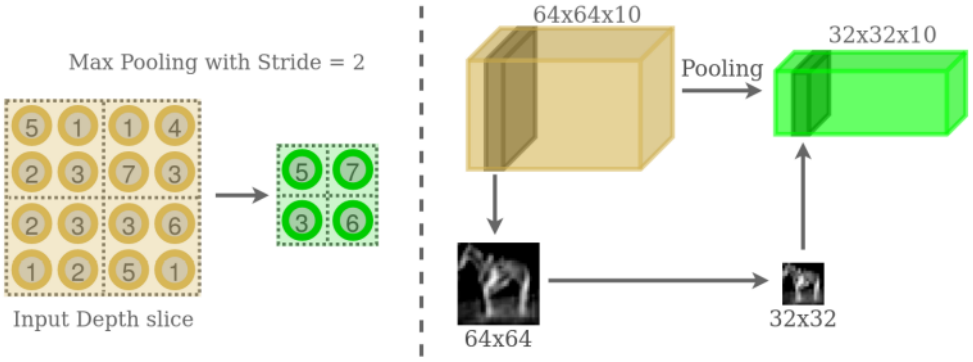


Figura 2.26: Representação de uma camada *pooling* [26].

As camadas de *pooling* não apresentam parâmetros que possam ser aprendidos, já que aplicam uma função fixa às entradas. O objetivo é diminuir a quantidade de parâmetros da rede, descartando algumas ativações [25]. Por outras palavras, uma camada de *pooling* com filtros 2x2 aplicados com um passo de dois, por exemplo, descartará 75% da sua camada de entrada. As aplicações mais comuns usam passos de dois e filtros de tamanho dois ou três mas valores maiores são evitados, pois podem ser muito destrutivos.

Camadas Totalmente Conectadas

Nas camadas referidas até ao momento, não ocorreu nenhuma classificação, apenas extração de características e redução de tamanho. São as camadas totalmente conectadas que são responsáveis pela parte de classificação da rede. Como o nome sugere, os neurónios destas camadas estão conectados a todos os neurónios da camada anterior.

No final das RNC, após a extração de características e a redução de tamanho, são utilizadas as camadas totalmente conectadas. O vetor de neurónios resultante da última camada de extração de características é utilizado como entrada para a primeira camada totalmente conectada da rede neuronal, como mostra a Figura 2.27. Esta camada é responsável por conectar todos os neurónios da camada anterior com todos os neurónios da camada seguinte, aumentando a complexidade e o poder de generalização da rede.

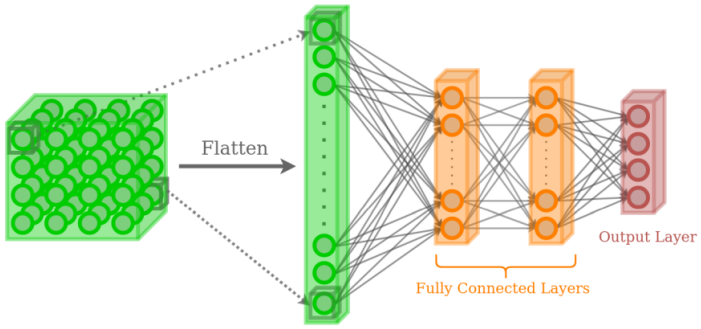


Figura 2.27: Representação de uma camada totalmente conectada [26].

É de salientar que a saída da camada anterior deve conter os mapas de ativação de recursos de alto nível da imagem de entrada e, com isso, as camadas totalmente conectadas podem aprender quais desses recursos de alto nível se correlacionam mais fortemente com uma determinada classe [26].

No geral, as RNC não usam muitas camadas totalmente conectadas, mesmo quando são profundas. Na realidade, a maioria das arquiteturas de RNC utiliza, no máximo, três camadas totalmente conectadas [26]. Isto ocorre porque o número de parâmetros em cada camada totalmente conectada aumenta muito rapidamente com o número de neurónios, o que pode levar a problemas de sobreajuste e a um aumento significativo no tempo de treino da rede. Em vez disso, as camadas convolucionais e de *pooling* são usadas para extrair características das imagens de entrada, e essas características são passadas para as camadas totalmente conectadas apenas no final da rede para classificação ou regressão. A camada de saída, que é totalmente conectada, terá um número de neurónios igual ao número de classes no conjunto de dados, onde cada neurónio fornecerá uma probabilidade da classe correspondente.

Para obter as probabilidades, é comum em RNC e de forma geral em classificação multiclasse, usar-se a função *softmax* como função de ativação da camada de saída:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{2.33}$$

onde $\vec{z}$ é o vetor de valores para cada neurónio relacionado com a classe para inserir a função de ativação e K é o número de classes. Esta função converte o vetor de saída da última camada totalmente conectada numa distribuição probabilística, onde a soma de todos os valores é igual a um. Isto permite que cada neurónio da camada de saída represente a probabilidade de uma classe específica. Esta transformação é obtida pela exponenciação dos valores da saída, seguida da divisão pela soma de todos os valores exponenciados. Com isto, é possível determinar a classe mais provável para uma determinada entrada [25].

2.6 Arquiteturas de RNC - Deteção de Objetos

Esta última década foi marcada pelo desenvolvimento de diversas arquiteturas de RNC que permitiram grandes avanços na classificação de imagens e deteção de objetos em imagens. Neste subcapítulo serão abordadas as arquiteturas EfficientDet [42] e MobileNet [43] utilizadas no âmbito deste trabalho.

2.6.1 EfficientDet

Antes de abordar a arquitetura EfficientDet [42], é necessário entender a arquitetura EfficientNet [44]. Esta rede visa aumentar o desempenho da rede no domínio ImageNet[1]. O conceito central, apresentado em [44], está no dimensionamento composto da rede em todas as dimensões (resolução de entrada, profundidade e largura), em vez de ajustar o tamanho de uma ou duas dimensões.

Como ilustrado na Figura 2.28, as abordagens convencionais geralmente realizam o aumento em apenas uma dimensão da rede, como largura, profundidade ou resolução. Isto pode aumentar o desempenho do modelo ajustando manualmente o coeficiente de escala, mas o processo de encontrar o valor certo é um processo lento e, geralmente, resulta numa precisão e eficiência abaixo do ideal. Quando o dimensionamento composto é realizado na EfficientNet, o ajuste de cada dimensão é realizado de maneira uniforme usando coeficientes de dimensionamento fixos. Por exemplo, quando dado um aumento de 2^N nos recursos computacionais, é possível aumentar a profundidade da rede por α^N, a largura por β^N e o tamanho da imagem por γ^N, onde α, β, γ são coeficientes constantes determinados por uma pesquisa de grade no modelo original que visa a exploração de diferentes combinações de hiperparâmetros num modelo de aprendizagem de máquina. Neste caso em específico, a pesquisa de

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

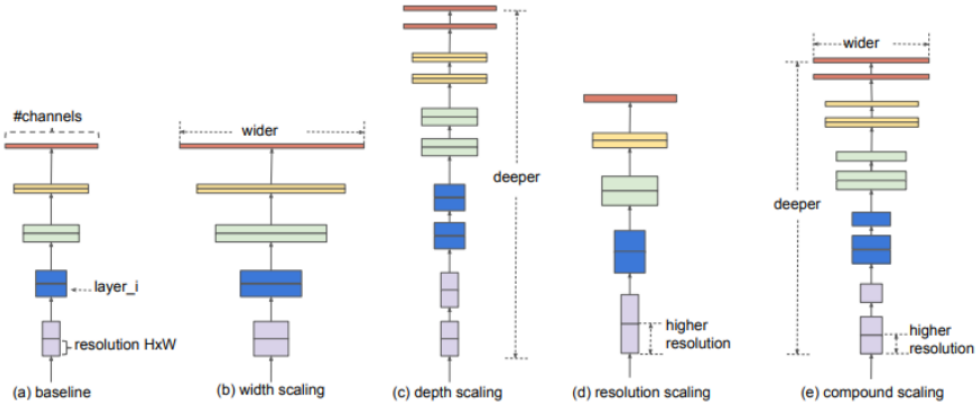


Figura 2.28: Dimensionamento do modelo [44]. (a) Exemplo de rede que vai servir de base. (b)-(d) Dimensionamentos convencionais que aumentam apenas uma dimensão da largura, profundidade ou resolução da rede. (e) Método de dimensionamento composto proposto que dimensiona uniformemente todas as três dimensões com uma proporção fixa.

grade foi realizada no modelo original para determinar os valores ótimos destes coeficientes. Então o processo de encontrar a base EfficientNet-B0 é realizado usando uma pesquisa de arquitetura neuronal multiobjetivo. Esta pesquisa visa a otimização de uma arquitetura de uma rede neuronal tendo em conta diferentes objetivos, como por exemplo, a precisão, a eficiência, etc. Para tal, são exploradas diferentes configurações e combinações de parâmetros. Quando estes parâmetros estiverem otimizados, encontrou-se a base da rede neuronal denominada de EfficientNet-B0. De seguida, a partir da rede base encontrada, o método de dimensionamento composto é aplicado para aumentar a dimensão da rede [44].

Na arquitetura EfficientDet [42], baseada na arquitetura EfficientNet [44], duas grandes contribuições são feitas:

- BiFPN permite fusão rápida de recursos, bidirecional multi-escala ;
- um novo método de dimensionamento composto dimensiona em conjunto a resolução, a profundidade, a largura , o *backbone*, a rede de recursos e a rede de previsão de caixa/classe.

A Figura 2.29 mostra a arquitetura geral da EfficientDet, que segue amplamente o paradigma dos detetores de um estágio [45]. Emprega-se EfficientNets pré-treinadas com ImageNet como rede de base. O BiFPN proposto serve como a rede de recursos, que recebe recursos de nível 3-7 $\{P_3, P_4, P_5, P_6, P_7\}$ da rede de base e aplica repetidamente a fusão de recursos bidirecional de cima para baixo e de baixo para cima. Estes recursos fundidos alimentam uma rede de classe e caixa para produzir previsões de classe de objeto e de caixas delimitadoras, respetivamente.

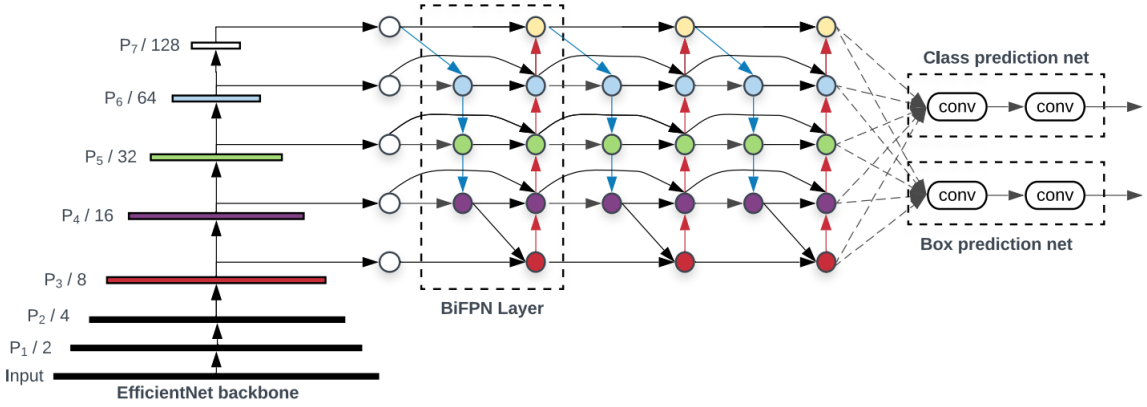


Figura 2.29: Arquitetura EfficientDet. Emprega a arquitetura EfficientNet como a base da rede, BiFPN como a rede de recursos e rede de previsão de classe/caixa compartilhada [44].

2.6.2 MobileNet

Atualmente, a IA está cada vez mais a ser utilizada no dia a dia. Contudo, dispositivos computacionalmente mais fracos são normalmente excluídos de aplicações que recorram a IA. A MobileNet vem resolver este problema, tendo sido projetada especificamente para este tipo de sistemas. Criada por Andrew G. Howard [43], divide a tarefa de convolução em duas partes. Pode-se observar que, inicialmente, ocorre uma convolução em profundidade que extrai informações relevantes de cada canal de entrada. Posteriormente, ocorre uma convolução pontual que busca as interações entre diferentes canais. Este conceito denomina-se de convoluções separáveis e é ilustrado na Figura 2.30.

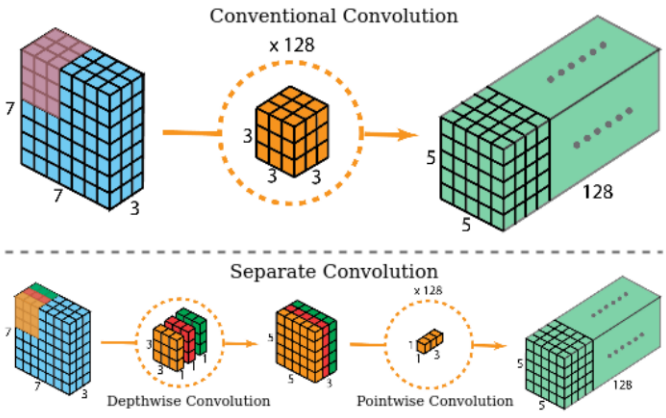


Figura 2.30: Convolução convencional e convolução separada [46].

Esta técnica resulta numa redução considerável do número de operações. Mais concretamente, para uma convolução convencional, o número de multiplicações M é

dado pelo tamanho do filtro F, pelo número de canais de entrada N, pela dimensão de saída D e pelo número de filtros K, de acordo com:

$$M = F^2 \times N \times D^2 \times K \tag{2.34}$$

Já para uma convolução separável, o número de multiplicações M é dado por:

$$M = (F^2 \times D^2 \times N) + (N \times D^2 \times K) \tag{2.35}$$

Tendo como referência a Figura 2.30, onde F=3, N=3, D=5 e K=128, substituindo na equação 2.34, o número de multiplicações é de 86400, enquanto que para a convolução separada, substituindo na equação 2.35 o número de multiplicações é de 10275, o que se traduz numa diminuição de operações em 88,1%. Isto resulta numa redução significativa no custo computacional mantendo a precisão.

2.7 Transfer Learning

Transfer Learning é uma técnica que utiliza o conhecimento adquirido numa tarefa e aplicado a outras tarefas relacionadas. No contexto, esta técnica de aprendizagem profunda permite treinar rapidamente uma RNC sem iniciar os pesos a zero [47]. Em vez disso, são importados os pesos de outra RNC que já foi treinada com dados semelhantes. Os pesos mais populares são aqueles treinados no conjunto de dados ImageNet[2]. Estes pesos podem ser usados para classificar outro conjunto de dados diferente, em vez de começar do zero. Existem quatro estratégias principais de *Transfer Learning*. A primeira estratégia envolve remover as camadas de classificação original, manter os pesos da RNC e adicionar uma nova camada de classificação para a nova tarefa. A segunda estratégia envolve ajustar os pesos da RNC para a nova tarefa e adicionar uma nova camada de classificação. A terceira estratégia envolve ajustar apenas as camadas superiores da RNC e adicionar uma nova camada de classificação. A quarta estratégia consiste em começar a treinar uma nova RNC do zero, mas com uma arquitetura comprovada [47].

2.8 Compute Unified Device Architecture (CUDA)

A *Central Processing Unit* (CPU) é o componente físico mais utilizado em operações sequenciais. Contudo, quando se trata de programação paralela, a *Graphics Processing Unit* (GPU) é mais proeminente devido à sua capacidade de processamento paralelo, utilizando a ferramenta CUDA[3]. Embora no passado as GPU fossem utilizadas apenas como aceleradores gráficos, a programação paralela tornou-as num

[2]https://www.image-net.org/
[3]https://developer.nvidia.com/cuda-zone

processador de dados paralelo poderoso, programável e de propósito geral, para operações que requerem grande capacidade de processamento.

A arquitetura dos núcleos de GPU e CPU é bastante diferente. Uma das diferenças entre o núcleo de CPU e o de GPU é que o primeiro foi projetado para executar programas sequenciais com lógica de controlo complexa, enquanto o segundo foi projetado com lógica de controlo mais simples, com foco em tarefas de dados paralelos [48].

O CUDA é uma ferramenta que pode ser utilizada em aplicações matemáticas graças aos múltiplos *kernels* da placa gráfica. O processamento paralelo é uma ferramenta crucial para lidar com grandes conjuntos de dados em aplicações digitais, pois permite realizar várias tarefas simultaneamente reduzindo efetivamente o tempo de trabalho.

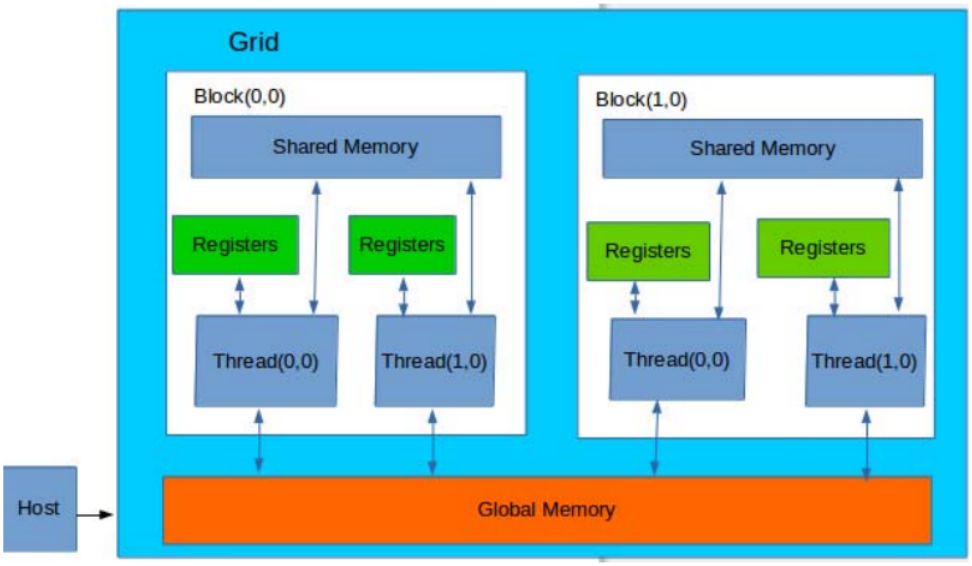


Figura 2.31: Topologia de memória CUDA [48].

A topologia de memória CUDA é ilustrada na Figura 2.31. O CUDA, com centenas de núcleos, tem sido amplamente utilizado para processar grandes quantidades de dados de forma rápida e eficiente. Tal resultou no sucesso da utilização do CUDA em diversas indústrias e áreas de aplicação.

Devido à possibilidade de programação paralela oferecida pela estrutura das RNC, as aplicações que recorrem a algoritmos com RNC podem ser feitas por cálculo paralelo em plataforma de nuvem, com a restrição da programação CUDA só ser possível em computadores com GPU suportada pela NVIDIA. A vantagem desta característica é mais evidenciada em aplicações de processamento de imagem e vídeo, uma vez que devido ao tamanho dos dados, a parte de treino e teste de uma RNC, processada por um CPU sequencial, pode demorar vários dias.

2.9 Avaliação de desempenho de detetores de objetos

Introduzido no campo da deteção de objetos na edição de 2010 do Pascal Visual Object Challenge (Pascal VOC) [49], a mAP tornou-se uma métrica padrão para avaliar o desempenho de detetores de objetos. Contudo, serão identificados isoladamente cada um dos conceitos-chave desta métricas nas próximas secções.

2.9.1 Intersecção sobre União

A Interseção sobre união, ou em inglês *Intersection over Union* (IoU), é usada para medir a sobreposição entre duas caixas delimitadoras. É definida como a razão entre a área em que ambos os polígonos estão sobrepostos e a área obtida pela junção dos dois polígonos. Na Figura 2.32 está ilustrada esta métrica [25].

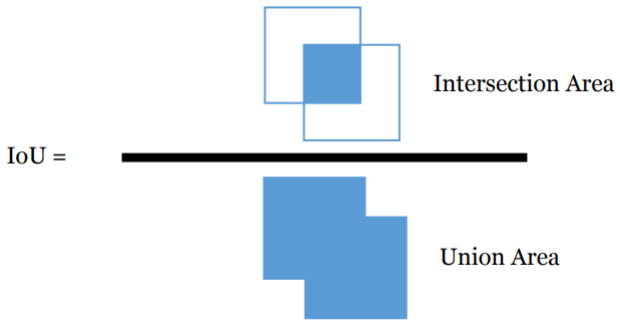


Figura 2.32: Visualização de Interseção sobre União [25].

2.9.2 Precisão e Sensibilidade

Com o objetivo de se obter uma explicação mais objetiva acerca destas métricas será utilizado o exemplo que se segue. Imagine-se que se pretende avaliar o desempenho de um detetor de objetos que foi treinado para identificar maçãs. Suponha-se também que uma imagem com cinco maçãs é inserida no detetor, cuja saída está representada na Figura 2.33.

Para compreender melhor estas métricas importa entender os termos de previsões e *Ground Truth*. Quando uma imagem de entrada alimenta um detetor de objetos, este realiza um conjunto de previsões, nomeadamente a classe a que determinado objeto pertence, o valor da pontuação e a posição relativa, expressa como caixa delimitadora. O *Ground Truth* é um conjunto de anotações de referência que possuí a classe do objeto e a sua posição relativa corretas. Normalmente, estas anotações são feitas manualmente por humanos.

Para que se possa compreender as métricas de Precisão e Sensibilidade, é importante primeiro entender a terminologia utilizada. Verdadeiros positivos, ou em inglês *True Positive* (TP), são deteções onde a classe prevista é a mesma classe de

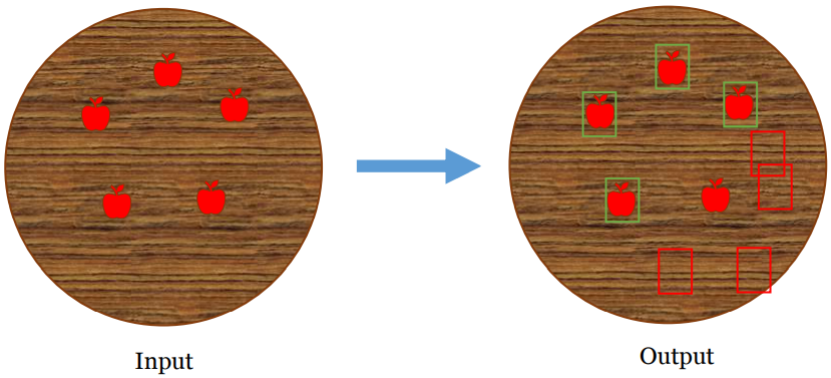


Figura 2.33: Visualização do problema de pontuação de deteção de maçãs. Os quadrados verdes representam deteções corretas e e os quadrados vermelhos representam deteções incorretas [25].

Ground Truth e onde a caixa delimitadora se sobrepõe à caixa de *Ground Truth* com um IoU igual ou superior ao imposto. No caso onde múltiplas previsões cumpram este requisito, apenas a caixa com maior IoU é contada como TP, sendo as restantes consideradas falsos positivos, ou em inglês *False Positive* (FP). Na Figura 2.33 as caixas verdes representam os TP.

Já os FP são previsões que possuem uma classe distinta da classe definida no *Ground Truth*. Por último, se a previsão da classe estiver correta, mas o IoU entre a caixa delimitadora prevista e a caixa delimitadora de *Ground Truth* for inferior ao IoU definido, a previsão é considerada como um FP. Na Figura 2.33 as caixas vermelhas representam os FP.

Com base nas definições mencionadas, é possível estabelecer duas métricas: a Precisão, que consiste na proporção entre a quantidade de TP e o total de previsões realizadas, e a Sensibilidade (em inglês, *recall* ou *sensitivity*), que corresponde à proporção entre a quantidade de TP e o número total de caixas delimitadoras de *Ground Truth* [25]. Por fim, existem os falsos negativos, ou em inglês *False Negative* (FN), que indica quando uma deteção devia ter sido feita mas o modelo não foi capaz de detetar. Estas métricas calculam-se de acordo com:

$$Pre = \frac{TP}{TP + FP} \tag{2.36}$$

$$Rec = \frac{TP}{TP + FN} \tag{2.37}$$

Então, pode-se interpretar a precisão como a probabilidade de uma previsão específica estar correta, enquanto a sensibilidade pode ser vista como a probabilidade de deteção de um objeto.

2.9.3 A curva de Precisão-Sensibilidade

Os conceitos de Precisão e Sensibilidade mencionados anteriormente são aplicáveis para previsões que não têm pontuação atribuída. Para calcular a mAP, é necessário avaliar as previsões do modelo de deteção de objetos em relação a uma medida de confiança, variando de 0 a 1, onde 1 indica alta confiança na previsão. Com base nessas previsões avaliadas, é possível criar a curva de Precisão-Sensibilidade.

A curva Precisão-Sensibilidade é um gráfico que mostra a relação entre a precisão e a sensibilidade para diferentes níveis de confiança no modelo. No eixo horizontal, são apresentados valores de sensibilidade, enquanto que no eixo vertical, são apresentados valores de precisão [25]. Por outras palavras, dadas todas as previsões realizadas para o conjunto de dados de teste, para cada nível de confiança, os valores de precisão e sensibilidade são calculados usando as equações 2.36 e 2.37, mas considerando apenas as previsões com pontuação de confiança igual ou superior ao nível em consideração.

2.9.4 Mean Average Precision, mAP

A definição de mAP vem da definição de *Average Precision* (AP) e é igual à média de onze valores de precisão obtidos pela interpolação da curva Precisão-Sensibilidade em onze valores de sensibilidade igualmente espaçados de 0 a 1 e é dado por [25]:

$$AP = \frac{1}{11} \sum_{r \in (0, 0.1 \ldots 1)} p_{interp}(r) \tag{2.38}$$

Para cada valor de sensibilidade, o valor de precisão interpolado é obtido tomando o valor de precisão máxima entre todas as pontuações de precisão com um valor de sensibilidade associado maior ou igual ao valor de sensibilidade em consideração e é dado por [25]:

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{2.39}$$

onde $p(\tilde{r})$ é o valor de precisão associado com o valor de sensibilidade $\tilde{r}$.

Na Figura 2.34 estão representadas as curvas Precisão-Sensibilidade de duas classes de um dado modelo. Os pontos azuis e verdes representam os valores de sensibilidade e os respetivos valores de precisão interpolados, de cada classe.

Quando é medido o desempenho de uma única classe denomina-se de AP. Contudo, uma das vantagens dos modelos de deteção de objetos é a capacidade de conseguir detetar diferentes classes. Para esta situação, recorre-se à mAP como medida de desempenho de um detetor de objetos multiclasse. O cálculo do mAP é dado pela média das AP de cada classe. Tal como o cálculo da precisão e sensibilidade, é preciso definir o limite IoU. A notação mais comum é dada por mAP@$_{@IoU}$. Uma

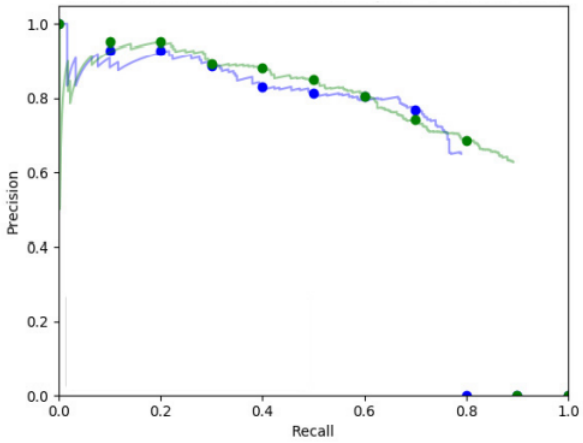


Figura 2.34: Exemplos de curvas Precisão-sensibilidade [25].

das métricas mais universalmente aceites é o $\text{mAP}_{@0.5}$, que utiliza um valor de limiar de sobreposição igual a 0,5, ou seja, se a área de sobreposição entre as caixas delimitadoras for maior ou igual a 0,5, considera-se que a deteção é correta. Além disso, existem outras variantes do mAP, como o mAP@0.75, que utiliza um valor de limiar de sobreposição igual a 0,75. Esta métrica é mais exigente do que o mAP@0.5, pois requer uma sobreposição maior entre as caixas delimitadoras para considerar a deteção correta. Finalmente, o $\text{mAP}_{@0.5-0.95}$, que considera um intervalo de valores de limiar de sobreposição, variando de 0.5 a 0.95. Esta métrica é a mais rigorosa, pois avalia o desempenho do algoritmo em diferentes níveis de sobreposição, desde uma sobreposição moderada até uma sobreposição quase completa.

2.10 Geração de dados sintéticos

Apesar do ótimo funcionamento das RNC no que toca a visão computacional, mais concretamente, a identificação de objetos, esta tem a desvantagem de requerer um conjunto de dados de treino volumoso. Isto coloca não só problemas de memória, visto que o número de imagens se situa nos milhões, como também torna o tempo de treino bastante demorado [33]. Contudo, com recurso a tecnologias como *Transfer Learning* o número de imagens de um conjunto de dados pode ser reduzido [25].

A aquisição deste conjunto de dados e a respetiva anotação é, normalmente, feito manualmente e é um processo muito demorado devido ao enorme volume do conjunto de dados necessário [25]. Para resolver estes problemas, várias técnicas de geração automática de dados sintéticos têm sido propostas. Porém, a que mais se destaca é a técnica baseada em modelos *Computer-Aided Design* (CAD). A revisão de literatura relevante deste método será apresentada na subsecção seguinte.

2.10.1 Modelo baseado em CAD

Antes de se analisar a literatura relevante, é necessário abordar alguns conceitos chave de um modelo baseado em desenho assistido por computador ou CAD. Um modelo CAD consiste numa representação virtual de um objeto físico, criada através de um programa de computador CAD. Todas as propriedades visuais e geométricas do objeto são codificadas neste modelo digital.

Um *software* gráfico de computador é uma aplicação informática (como o Blender [50] ou o Solidworks [51]) projetada para criar, editar e manipular imagens e gráficos digitais. Este tipo de *software* é utilizado em várias áreas, incluindo design gráfico, produção de conteúdo multimédia, arquitetura, engenharia e animação.

Na literatura observa-se que alguns trabalhos tentam maximizar o realismo da cena [52, 53], enquanto outros tentam alcançar um pseudo-realismo [54, 55]. A modelação de cenas realistas apesar de, como o nome indica, serem mais próximas da realidade, têm a desvantagens de exigir uma maior quantidade de tempo para conseguir modelar a cena de forma correta. Por outro lado, em [56, 57], os autores aplicaram a aleatorização de domínio à cena e atingiram resultados semelhantes aos resultados de autores que maximizaram o realismo da cena [56]. A aleatorização de domínio é uma técnica para treinar modelos em imagens simuladas que são transferidas para imagens reais por renderização aleatória no simulador. Para tal, atribui-se valores aleatórios a determinados parâmetros tais como cor, tamanho e textura, entre outros. A ideia base é que se for introduzido um número considerável de variações num modelo do mesmo objeto criado em simulação, quando este objeto for visto pelo modelo numa imagem real vai tratá-la como uma variação do dito objeto e, como tal, vai conseguir fazer a deteção.

Em algumas atividades de processamento de imagem, nas quais é difícil gerar anotações precisas, é comum recorrer-se a imagens geradas por modelos CAD para fins de treino [58, 59]. Conjuntos de dados com imagens reais e geradas a partir de modelos CAD atingem melhores resultados do que conjuntos de dados puramente reais [54, 60, 61]. Aumentar a quantidade de dados de treino ao misturar imagens sintéticas com imagens reais é uma técnica conhecida na área. Um estudo descrito em [61], onde foram adicionadas 100 imagens sintéticas por classe a um conjunto de dados de imagens reais que não foram sujeitas a uma anotação cuidada, concluiu que existe uma melhoria significativa na segmentação de objetos quando avaliadas com imagens reais. Noutro estudo, em [54] foi gerado um conjunto de dados de imagens simuladas através da utilização do software Unity 3D, a fim de auxiliar as tarefas como estimação de profundidade, segmentação semântica, deteção de objetos e fluxo ótico e embora as texturas utilizadas não sejam foto realistas, o conjunto de dados gerado representa cenas urbanas com condições de tráfego realistas. De facto, verificou-se que o aumento do conjunto de dados de imagens reais com imagens geradas sinteticamente melhorou a deteção de objetos em imagens reais.

A técnica de geração de dados apresentada em [60] difere do apresentado acima pela sua simplicidade e generalidade. Em vez de se configurar cenas realistas, apenas foi feita a renderização do objeto num plano de fundo branco e usaram-se algumas técnicas de pós-processamento para juntar o objeto com o plano de fundo pretendido, de forma a que o objeto renderizado pertença a parte do plano de fundo. Concluíram que, no que toca à tarefa de deteção de objetos, a adição de imagens sintéticas consegue melhorar o desempenho de um modelo treinado com imagens reais.

O uso exclusivo de imagens geradas a partir de modelos CAD tem sido amplamente aplicado com sucesso em diversos campos. Um exemplo é descrito em [62], que se concentrou na tarefa de deteção de objetos com recurso a uma técnica de geração semelhante àquela descrita em [60], a fim de avaliar os efeitos de diferentes fundos e texturas na tarefa de treinar detetores de objetos em imagens sintéticas e, posteriormente, testá-los em imagens reais. O algoritmo de geração de dados, semelhante ao apresentado em [56], utiliza aleatorização de domínios para gerar imagens, permitindo que sejam modificados aleatoriamente o fundo e as texturas. Uma técnica semelhante foi usada em [63], onde um conjunto de dados de imagens sintéticas foi criado com o objetivo específico de resolver o problema de adaptação de domínio para deteção de objetos. A técnica de geração de dados descrita em [62] é praticamente idêntica à técnica descrita em [63], com a única diferença de que na última, a textura do objeto renderizado é gerada de forma aleatória.

Em [64] foi apresentada uma prova de conceito em que um modelo de segmentação de objetos foi treinado exclusivamente com o uso de imagens renderizadas a partir de modelos 3D. No decorrer do estudo, os autores compararam o desempenho de dois *softwares* de renderização, o Unity 3D e o Blender. Os testes realizados demonstraram que o desempenho de segmentação do modelo gerado com imagens renderizadas pelo Blender é superior ao obtido através do Unity 3D.

Algumas abordagens têm utilizado a combinação de rede generativas adversárias com imagens produzidas a partir de modelos CAD para gerar conjuntos de dados de treino. Num estudo realizado por Peng et al.[65], uma rede generativa adversária foi empregue para colorir imagens sintéticas geradas a partir de modelos CAD. Os resultados empíricos indicam que esta técnica de geração de imagens realistas superou outras técnicas de última geração na tarefa de deteção de objetos. Embora as imagens geradas por esta técnica sejam realistas, é importante destacar que a abordagem de adaptação de domínio foi a responsável pelo sucesso [65].

2.10.2 Data Augmentation

Data augmentation é uma técnica usada na aprendizagem de máquina para aumentar artificialmente o tamanho do conjunto de dados através da criação de versões

modificadas dos dados originais. Esta técnica pode ser aplicada em diversos domínios, incluindo visão computacional, processamento de linguagem natural e reconhecimento de fala. O foco da presente subsecção é a técnica de *image augmentation*, que é um tipo de *data augmentation* utilizado em visão computacional [66].

A *image augmentation* consiste na aplicação de um conjunto de transformações a uma imagem para criar versões novas e ligeiramente diferentes da imagem original. O objetivo desta técnica é criar um conjunto de dados mais diverso que pode melhorar o desempenho de um modelo de aprendizagem de máquina tornando-o mais robusto e menos sensível a pequenas variações nos dados de entrada [66].

Os métodos existentes de *image augmentation* podem ser divididos numa de duas categorias gerais: métodos tradicionais e métodos baseados em redes neuronais profundas. Nesta subsecção serão abordados apenas os métodos tradicionais [67].

Atualmente, a prática mais frequentemente utilizada para aumentar dados é a combinação de transformações geométricas, de transformações de cor e adição de ruído. As transformações geométricas são um conjunto de técnicas que envolvem a alteração das propriedades espaciais de uma imagem sem alterar o seu conteúdo. Estas transformações podem ajudar a melhorar a generalização de um modelo de aprendizagem de máquina, introduzindo variações nos dados de entrada, que este possam ser encontradas em cenários do mundo real.

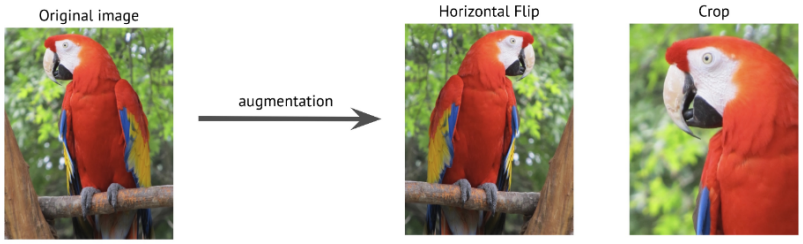


Figura 2.35: A mesma imagem depois de diferentes tipos de transformações, adaptado de [68].

Na Figura 2.35 são representadas algumas transformações geométricas, que incluem [67]:

- Corte: Redimensionar ou cortar uma imagem pode ajudar o modelo a focar em áreas específicas, remover informações irrelevantes ou simular diferentes pontos de vista de um objeto, enriquecendo assim a diversidade dos dados de treino;
- Espelhar : Envolve espelhar a imagem horizontalmente ou verticalmente, que pode ajudar o modelo a reconhecer os objetos vistos de diferentes ângulos;
- Rotação: Rodar a imagem pode ajudar o modelo a aprender a reconhecer objetos vistos de diferentes ângulos;

- Escalar: Envolve aumentar ou diminuir o tamanho do objeto de interesse, o que pode ajudar o modelo a aprender a reconhecer os objetos a diferentes distâncias ou escalas.

As transformações de cor são um conjunto de técnicas utilizadas em *image augmentation* que envolvem a modificação das propriedades de cor de uma imagem. Estas transformações podem ajudar a melhorar a robustez de um modelo de aprendizagem de máquina, tornando-o mais tolerante a variações de iluminação e cor.

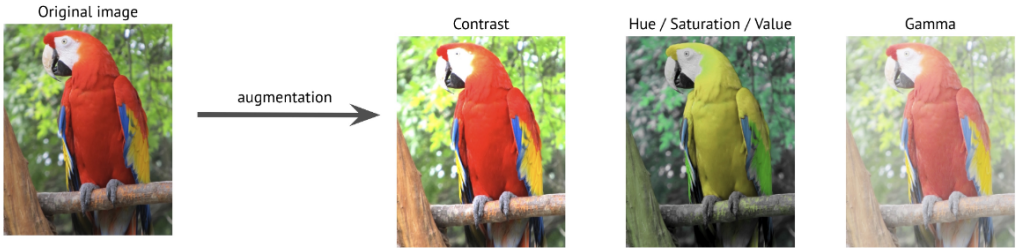


Figura 2.36: A mesma imagem depois de diferentes tipos de mudanças de cor, adaptado de [68].

Na Figura 2.36 observa-se algumas das técnicas de *image augmentation* para mudanças de cor, que incluem:

- Ajuste de brilho: Aumentar ou diminuir o brilho de uma imagem pode ajudar o modelo a aprender a reconhecer objetos em diferentes condições de iluminação;
- Ajuste de contraste: Aumentar ou diminuir o contraste de uma imagem pode ajudar o modelo a aprender a reconhecer objetos com diferentes níveis de contraste.
- Ajuste de saturação: Aumentar ou diminuir a saturação de uma imagem pode ajudar o modelo a aprender a reconhecer objetos com diferentes intensidades de cor.
- Ajuste de matiz: Mudar a matiz de uma imagem pode ajudar o modelo a aprender a reconhecer objetos com diferentes tonalidades de cor.

Ao considerar estas técnicas, é importante mencionar a relação com espaços de cor e histogramas. As propriedades dos pixeis nos espaços de cor são alteradas com o ajuste do brilho, contraste, saturação e matiz. Estas alterações nos espaços de cor podem ter um impacto direto nos histogramas da imagem, que representam a distribuição de intensidades de cor.

Ao realizar *image augmentation* com base nestas técnicas, é possível explorar diferentes regiões do espaço de cor e distribuição de intensidades, permitindo que o modelo seja treinado para reconhecer objetos numa ampla gama de condições de

cor e iluminação. Estas manipulações ajudam a aumentar a diversidade dos dados de treino, tornando o modelo mais robusto e capaz de lidar com variações reais em imagens durante o uso em aplicações práticas.

Por último, a adição de ruído é um conjunto de técnicas utilizadas no aumento de imagens que envolvem a adição de variações aleatórias aos valores de pixel de uma imagem. Estas transformações podem ajudar a melhorar a generalização de um modelo de aprendizagem de máquina, tornando-o mais tolerante a variações nos dados de entrada. No que toca a dados gerados sinteticamente por CAD, esta técnica é especialmente importante. Uma vez que os dados gerados sinteticamente não possuem qualquer tipo de ruído e como todas as câmaras possuem algum tipo de ruído, com recurso a esta técnica é possível melhorar significativamente o desempenho de um modelo treinado com imagens sintéticas no momento de identificar objetos reais.

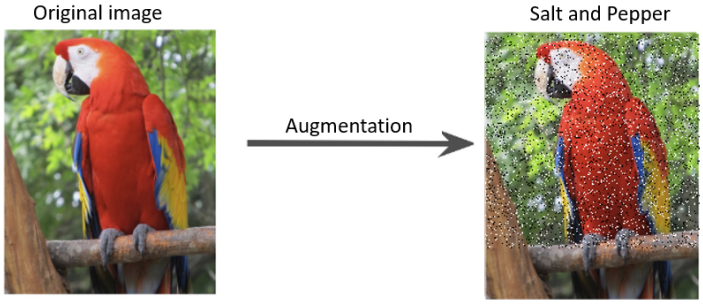


Figura 2.37: A mesma imagem depois de aplicar ruído, adaptado de [68].

Na Figura 2.37 está ilustrada uma imagem à qual foi adicionado ruído. Alguns exemplos de transformações baseadas em ruído incluem:

- Ruído gaussiano: Adicionar ruído aleatório com uma distribuição gaussiana a uma imagem pode ajudar o modelo a aprender a reconhecer objetos em condições ruidosas;
- Ruído sal e pimenta: Adicionar pixels pretos e brancos aleatórios a uma imagem pode ajudar o modelo a aprender a reconhecer objetos em condições desfavoráveis;
- Ruído de Poisson: Adicionar ruído aleatório com uma distribuição de Poisson a uma imagem pode ajudar o modelo a aprender a reconhecer objetos em condições de pouca luz.

Em suma, estes são os métodos mais comuns [67, 69] dentro da *image augmentation* que são normalmente usados para aumentar [70] e equilibrar [71] os dados de treino e melhorar a eficiência dos modelos [72]. Com recurso a estas técnicas, os

modelos de aprendizagem de máquina podem ser treinados para serem mais robustos e precisos. Contudo, a escolha das técnicas e a combinação destas depende da aplicação específica e do resultado desejado.

Capítulo 3

Desenvolvimento do projeto

O objetivo da presente dissertação é criar um sistema de *pick and place* com recurso a uma RNC treinada com um conjunto de dados gerados no domínio da simulação para identificação de objetos no domínio da realidade. A tarefa de *pick and place* passa por identificar determinado objeto, estimar a sua posição, para depois, com ajuda de um sistema robotizado, neste caso um braço mecânico, pegar no objeto e coloca-lo num outro local pré-determinado.

Inicialmente, foi necessário selecionar os objetos alvo de identificação para se proceder à criação dos respetivos modelos, de seguida foram criadas diferentes imagens. Para a criação de diferentes conjuntos de dados, anotou-se cada imagem criada e aplicaram-se técnicas de *image augmentation*. Com os diferentes conjuntos de dados criados, foi escolhida a arquitetura e o *software* para criação e treino das redes neuronais e foram abordadas as descrições do *hardware*. Por fim, a arquitetura geral do sistema *pick and place* é apresentada, a qual é dividida em três sistemas: o sistema de visão, o sistema de comunicação e o sistema de controlo.

3.1 Modelação das ferramentas

Neste caso, os objetos escolhidos para identificação e localização foram ferramentas de trabalho, mais concretamente, uma chave de estrela, uma chave inglesa, uma chave de sextavado, um roquete e um alicate. Estes objetos estão ilustrados na Figura 3.1.

(a) Chave de Estrela

(b) Chave Inglesa

(c) Chave sextavado

(d) Roquete

(e) Alicate

Figura 3.1: Ferramentas reais usadas para modelação.

Com os objetos a identificar definidos, a próxima etapa consiste na determinação da técnica mais adequada para a modelação das ferramentas. Existem diversos métodos disponíveis, sendo os mais preponderantes o *Cut and Paste* usado em [25] e o CAD utilizado em [52, 53, 54, 73]. Estabeleceu-se que as ferramentas seriam modeladas em 3D com recurso ao método CAD, uma vez que na literatura, quando comparado a outros métodos, é este que apresenta resultados superiores [25]. Para além disto, o INEGI apelou ao uso deste método dado o interesse por parte da organização para a sua exploração. O *software* CAD selecionado para a modelação 3D foi o SolidWorks [51] devido às suas capacidades de renderização, à natureza dos objetos, ao facto do Instituto Superior de Engenharia do Porto (ISEP) possuir uma licença de estudante e por interesse do autor. O SolidWorks é um *software* CAD que utiliza o princípio do projeto paramétrico e gera três tipos de arquivos

interconectados: a peça, a montagem e o desenho. Assim, qualquer modificação num desses três arquivos será refletida nos restantes. Na Figura 3.2 está representado o ambiente de trabalho do Solidworks.

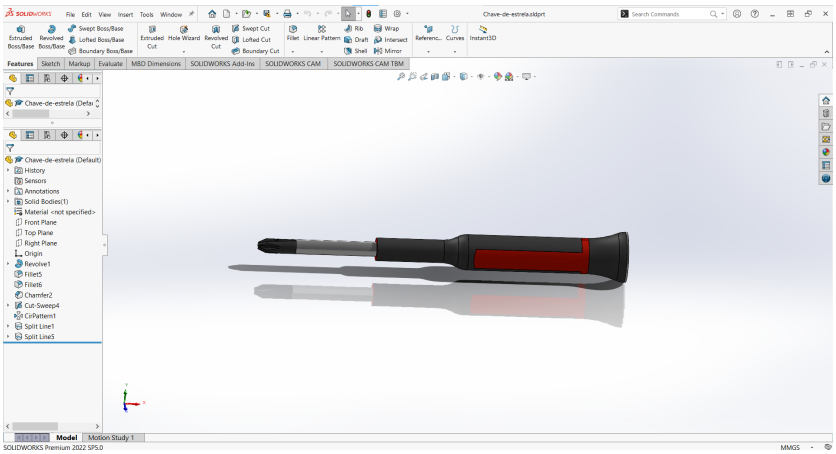


Figura 3.2: Ambiente de trabalho do Solidworks.

O resultado final das ferramentas modeladas é apresentado na Figura 3.3.

3.2 Descrição do conjunto de dados

A geração de dados foi dividida em dois domínios principais, o domínio de treino e o domínio de validação. Os diferentes conjuntos de dados que pertencem ao domínio de treino, como o nome indica, são as imagens que vão alimentar o modelo no processo de aprendizagem. O conjunto de dados de validação vai ser utilizado para testar o funcionamento dos modelos e avaliar o seu desempenho durante o treino, para saber se o treino deve, ou não, terminar. O padrão no treino de redes neuronais é criar um terceiro conjunto de dados denominado de conjunto de dados de teste para, quando uma rede neuronal chegar ao término do processo de aprendizagem, testar uma última vez a rede neuronal no seu estado final. Contudo, dado que neste projeto o objetivo é o desenvolvimento de um sistema *pick and place*, os testes e a avaliação serão feitos em tempo real onde uma ferramenta será posta em frente ao sistema. Desta forma, é possível estudar o funcionamento de cada rede neuronal treinada de uma forma mais precisa, uma vez que a rede neuronal estará a funcionar em tempo real e será possível estudar também se o braço mecânico consegue deslocar-se corretamente até à peça.

A aquisição dos diferentes conjuntos de dados de treino pode ser dividida em três partes: na aquisição do conjunto de dados inicial, na anotação manual de cada imagem e na aplicação de técnicas de *image augmentation* para gerar conjuntos de dados mais completos. Com o intuito de homogeneizar a orientação das ferramentas no conjunto de dados de treino, para permitir a um modelo treinado com este

(a) Chave de Estrela

(b) Chave Inglesa

(c) Chave de Sextavado

(d) Roquete

(e) Alicate

Figura 3.3: Imagens obtidas a partir das ferramentas modeladas.

conjunto a identificação das ferramentas em diferentes posições, variou-se, metodicamente, a orientação das ferramentas durante a aquisição de todos os conjuntos de dados (para efeitos de simplificar a explicação, designar-se-á estas imagens por imagens originais). Para um conjunto de dados, define-se o número de imagens originais que este irá possuir e divide-se esse conjunto em duas partes. De seguida, divide-se 360 pela primeira metade do número total de imagens originais com o intuito de saber o ângulo de rotação entre duas imagens consecutivas. Por exemplo, se forem definidas 50 imagens originais, o ângulo de rotação é $360 \div 25 = 14,4$. Este processo é parcialmente ilustrado na Figura 3.4.

Na segunda metade do número total de imagens originais, a posição da ferramenta é invertida e repete-se o processo, ilustrado na Figura 3.5. No caso do alicate, na primeira metade ele encontra-se fechado e na segunda encontra-se com diferentes

(a) Posição 1 (b) Posição 2 (c) Posição 3

Figura 3.4: Exemplo da sequência de posições da Chave de Sextavado na primeira metade do conjunto de dados.

níveis de abertura. Este processo foi realizado para todos os conjuntos de dados.

(a) Posição 1 (b) Posição 2 (c) Posição 3

Figura 3.5: Exemplo da sequência de posições da Chave de Sextavado na segunda metade do conjunto de dados.

Após a aquisição das imagens originais para os modelos, prossegue-se à anotação manual de cada imagem. O processo de anotação consiste na criação de um ficheiro XML, no qual estão contidas as coordenadas, na imagem, de uma caixa delimitadora onde está situada a região de interesse que, neste caso, são as ferramentas e o seu rótulo, como ilustrado na Figura 3.6. O *software* usado para a anotação foi o LabelImg [74].

Com as imagens originais criadas e anotadas, é possível aplicar o algoritmo de *image augmentation* abordado no capítulo anterior. Neste caso, variou-se a orientação, a rotação, a luminosidade e o ruído gaussiano. Na Figura 3.7, em (a) está representada uma imagem original e em (b) e (c) estão representadas duas imagens geradas com recurso a *image augmentation* [1].

Contudo, foi identificado um problema no algoritmo de *image augmentation*. Este recorre ao ficheiro XML para saber as coordenadas da caixa delimitadora e, quando aplica uma rotação à imagem, atualiza automaticamente as coordenadas. No entanto, o algoritmo não sabe a posição exata do objeto de interesse, então adiciona uma folga à caixa delimitadora para garantir que o objeto permaneça dentro dos

[1]Disponível em: https://github.com/EdjeElectronics/Image-Augmentation-Examples-for-Machine-Learning/blob/master/augment_with_KPs.py

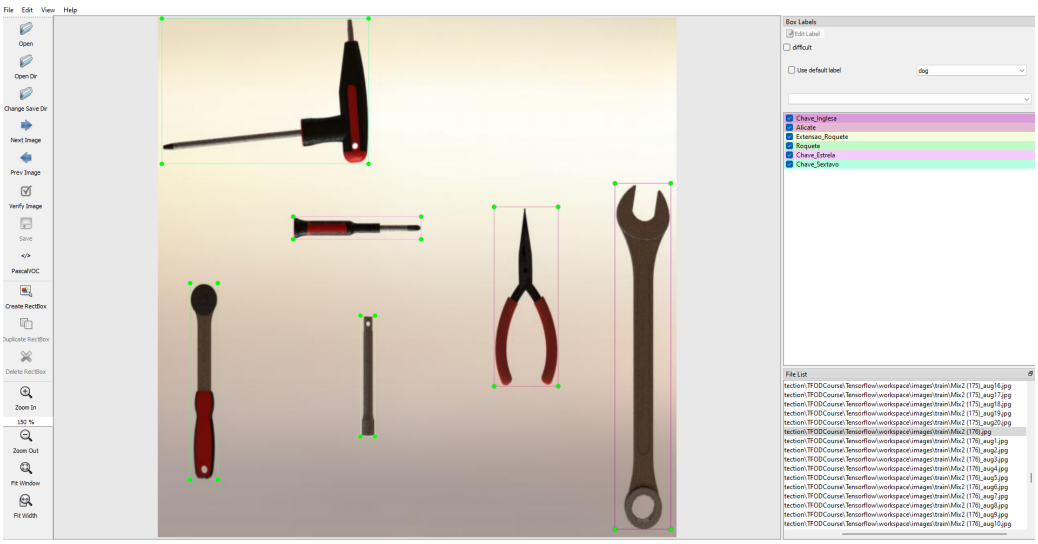


Figura 3.6: Exemplo de uma imagem anotada no LabelImg.

limites. Verificou-se que isto era um problema, uma vez que a rede neuronal ia ser treinada para deixar uma folga na caixa delimitadora imposta pela mesma. Para o ultrapassar adaptou-se o algoritmo original.

Assim, depois do algoritmo de *image augmentation* definir as novas coordenadas da caixa delimitadora, utilizam-se essas coordenadas para delimitar a região de interesse. De seguida, aplicou-se uma função *Canny* para detetar os contornos dos objetos dentro da caixa e extraiu-se o ponto mais à esquerda, o ponto mais à direita, o ponto mais acima e o ponto mais abaixo. Com isto, é possível remover toda e qualquer tipo de folga, garantindo que o objeto de interesse permaneça dentro da caixa delimitadora. Note-se que, este método só é viável uma vez que o fundo das imagens é completamente liso. Na Figura 3.8, apresenta-se à esquerda a caixa delimitadora sem a adaptação e à direita com a adaptação.

Nas próximas subsecções serão descritos cada um dos conjuntos de dados gerados.

3.2.1 Conjunto de dados 1

O conjunto de dados 1 possui uma particularidade quando comparado aos demais. Devido a uma limitação que será abordada em futuras secções, as imagens que constituem este conjunto de dados podem ser divididas em dois grupos: imagens individuais e imagens mistas. As imagens individuais possuem um única ferramenta enquanto as imagens mistas possuem cinco ferramentas numa mesma imagem. Desta forma, é possível obter a informação de cinco imagens individuais ocupando a memória de uma única imagem. Um exemplo de uma imagem mista encontra-se ilustrado na Figura 3.9.

O conjunto de dados 1 possui:

- 50 imagens do alicate;

(a) Imagem original de alicate

(b) Imagem gerada a partir da original 1

(c) Imagem gerada a partir da original 2

Figura 3.7: Exemplo de *image Augmentation*.

(a) Antes

(b) Depois

Figura 3.8: Antes e depois da adaptação do algoritmo de *image augmentation*.

- 50 imagens da chave de estrela;
- 73 imagens da chave inglesa;
- 50 imagens do roquete;
- 50 imagens da chave de sextavado;
- 154 imagens mistas.

Como tal, o conjunto de dados de dados 1 possuí 273 imagens individuais e 154 imagens mistas, o que resulta num total de 427 imagens originais. De seguida, foram geradas com recurso a *image augmentation* 70 imagens com cada imagem individual e 30 imagens com cada imagem mista, resultando em 19110 imagens

Figura 3.9: Exemplo de uma imagem mista.

individuais e 4620 imagens mistas geradas por *image augmentation*, totalizando em 23730 imagens.

Concluindo, o conjunto de dados 1 é constituído pela soma das imagens originais com as geradas por *image augmentation*, ou seja, 19383 imagens individuais e 4774 imagens mistas, resultando num total de 24157 imagens.

3.2.2 Conjunto de dados 2

A criação de um modelo 3D de raiz é um processo demorado e, uma vez que a velocidade da aquisição do modelo treinado é algo valorizado, identificou-se aqui uma oportunidade de otimização. Dado que existe uma vasta comunidade de modeladores que partilham, de forma gratuita, modelos 3D, é possível utilizar estes modelos para treinar uma rede neuronal.

Na Figura 3.10 estão representados os modelos 3D das ferramentas importados e utilizados no conjunto de dados 2. É de notar que se fez este estudo para apenas três ferramentas, uma vez que não foram encontrados modelos disponibilizados da chave de sextavado e, no caso da chave inglesa, não foram encontrados modelos significativamente diferentes do criado pelo autor. Para cada ferramenta foram usados três modelos 3D diferentes, com o intuito de obter uma representação mais abrangente de possíveis imagens a analisar [75].

De seguida foram obtidas 14 imagens por cada modelo, resultando em 42 imagens por cada classe e 126 imagens originais no total. Posteriormente, por cada imagem, foram geradas 100 imagens novas com recurso a *image augmentation* resultando num total de 4200 imagens por cada classe, 12600 no total.

Resumindo, o conjunto de dados 2 é constituído pelas imagens originais mais as imagens geradas por *image augmentation*, ou seja, 4242 por classe e 12726 imagens

(a) Alicates

(b) Chaves de Estrela

(c) Roquete

Figura 3.10: Modelos 3D importados.

no total.

3.2.3 Conjunto de dados 3

O conjunto de dados 3 é constituído por um subconjunto do conjunto de dados 1. Este conjunto de dados visa a comparação entre uma rede neuronal treinada com os modelos criados pelo autor e uma rede neuronal treinada com imagens dos modelos disponibilizados pela comunidade de modeladores (conjunto de dados 2). Desta forma é possível concluir se o tempo utilizado a criar os modelos específicos, se reflete nos resultados ou se é mais vantajoso utilizar modelos pré-existentes.

Para o conjunto de dados 3, foram usadas as imagens individuais do conjunto de dados 1 referentes ao alicate, à chave de estrela e ao roquete. Ou seja, 3550 imagens por ferramenta ou 10650 imagens no total.

3.2.4 Conjunto de dados 4

A combinação da abrangência procurada no conjunto 2, com a especialização do conjunto 3 gera o conjunto de dados 4. Ou seja, este conjunto visa a aglomeração entre a variabilidade de diferentes modelos das ferramentas apresentada no conjunto 2 com as ferramentas modeladas como réplicas das reais encontradas no conjunto 3.

O conjunto de dados 4 é composto pela totalidade do conjunto de dados 2 mais uma parte do conjunto de dados 3. Mais concretamente, as 50 imagens originais do conjunto de dados 3 por ferramenta. De seguida, com recurso a *image augmentation*, geraram-se 17 imagens com cada imagem original, resultando em 850 novas imagens por ferramenta ou 2550 no total.

Então, o conjunto de dados 4 é constituído pelas 4242 imagens por classe do conjunto 2 mais 900 imagens por classe do conjunto de dados 3, o que resulta em 5142 imagens por classe e num total de 15426 imagens.

3.2.5 Conjunto de dados 5

Por fim, o conjunto de dados 5 é constituído por imagens reais, ilustradas na Figura 3.11.

(a) Chave de Estrela (b) Roquete

Figura 3.11: Imagens pertencentes ao conjunto de dados 5.

Este conjunto foi criado com o intuito de comparar o funcionamento das redes neuronais treinadas com imagens geradas por computador e redes neuronais treinadas com imagens de objetos reais.

Para o conjunto de dados 5 foram obtidas 40 imagens de cada ferramenta, resultando num total de 200 imagens originais. Seguidamente, com recurso a *image augmentation* gerou-se 80 imagens novas a partir de cada imagem original, criando 3200 imagens por ferramenta e um total de 16000 imagens novas.

Resumindo, o conjunto de dados 5 possui 3240 imagens por ferramenta e um total de 16200 imagens.

3.2.6 Conjunto de dados de validação

O domínio de validação é constituído por um único conjunto de dados. Este, de forma análoga ao 5º conjunto de dados de treino, é composto por imagens reais das ferramentas. Para este conjunto, fotografou-se 10 vezes cada ferramenta. De seguida, por cada imagem foram geradas três novas imagens, onde numa se refletiu o eixo do x, noutra o eixo do y e na outra os eixos x e y, resultando num conjunto de dados com 40 imagens por ferramenta e um total de 200 imagens. Para as avaliações dos modelos treinados apenas com três classes remove-se as classes que não pertencem a esse modelo.

É de salientar que, independentemente da quantidade de modelos gerados a partir dos conjuntos de dados de treino, este será sempre o conjunto de dados de validação. Assim, é possível estabelecer uma comparação entre os diferentes modelos treinados.

3.3 Treino do modelo

3.3.1 Software

Relativamente ao *software* de aprendizagem de máquina e reconhecimento de imagem, existem diversas alternativas disponíveis. Algumas das mais comuns incluem:

- Tensorflow [76]: Uma biblioteca de aprendizagem de máquina de código aberto, desenvolvida pela equipa do Google Brain [77], amplamente utilizada para reconhecimento de imagem, processamento de linguagem natural e outras tarefas de aprendizagem de máquina;
- PyTorch [78]: Outra biblioteca de aprendizagem de máquina de código aberto, que ganhou popularidade nos últimos anos, especialmente entre pesquisadores e académicos;
- Keras [79]: Uma *Application Programming Interface* (API) de redes neuronais de alto nível que pode ser executada em cima do TensorFlow [76], do Microsoft Cognitive Toolkit (CNTK) [80] ou do Theano [81];
- Caffe [82]: Uma estrutura de aprendizagem profunda desenvolvida pelo Berkeley Vision and Learning Center [83];
- OpenCV [84]: Uma biblioteca de visão computacional de código aberto que inclui vários algoritmos para processamento de imagem e tarefas de visão computacional;
- Microsoft Cognitive Toolkit (CNTK) [80]: Um *kit* de ferramentas gratuito e de código aberto para aprendizagem profunda, que é usado por pesquisadores e profissionais da indústria.

Para o treino das RNC recorreu-se ao Tensorflow [76]. Este é considerado uma das bibliotecas de aprendizagem de máquina mais abrangentes e flexíveis, disponíveis atualmente. Fornece uma ampla gama de recursos para construir e treinar redes neuronais, incluindo RNC e redes neuronais recorrentes, que são comummente usadas em reconhecimento e rotulagem de imagem [85]. Para além disto, esta biblioteca inclui ferramentas de visualização e *debugging*, facilitando o processo de análise e entendimento dos resultados dos modelos de aprendizagem de máquina. Outra vantagem do Tensorflow [76] é a vasta comunidade de programadores que contribuíram com uma ampla gama de modelos pré-treinados e ferramentas que podem ser usadas para várias tarefas de aprendizagem de máquina [86].

Dado que, para a presente dissertação, a velocidade de treino da rede é algo importante, recorreu-se à API CUDA [87] e à biblioteca cuDNN [88] para efetuar aceleração do treino com recurso à GPU. O cuDNN é uma biblioteca de primitivas acelerada por GPU para redes neuronais profundas. Ele fornece implementações altamente eficientes de rotinas que surgem frequentemente neste tipo de redes. As versões utilizadas para cada um destes *softwares* foram as seguintes:

- CUDA toolkit: v11.0.2;
- Tensorflow-gpu: v2.7.0;
- cuDNN: v8.5.0.

3.3.2 Hardware

Como mencionado anteriormente, para o uso do *software* CUDA é necessária uma GPU da NVIDIA. O desempenho do código gerado depende da qualidade da GPU. Neste trabalho foi utilizada uma Nvidia Geforce GTX1650 [89], que possui 896 CUDA *Cores* e 4GB de Vram. A baixa memória Vram é a principal restrição encontrada neste projeto, uma vez que limita o número de imagens e as suas dimensões. Para além disso, o uso de modelos pré-treinados computacionalmente exigentes tornam-se inacessíveis e o *Batch size* definido para a fase de treino é limitado. Estas restrições comprometem, de certa forma, os resultados. O processador usado foi um Intel Core i5-10300H [90].

3.3.3 Modelo pré-treinado

O Tensorflow [76] fornece uma coleção de modelos de deteção pré-treinados no conjunto de dados *Common Objects in Context* (COCO) 2017 [91]. Estes modelos estão classificados segundo o tipo de identificação, precisão e velocidade, existindo um compromisso entre os últimos dois [92]. Estes modelos são baseados em diferentes arquiteturas, nomeadamente CenterNet [93], EfficientDet [94] e Resnet [95],

entre outras. Na Tabela 3.1 são apresentados alguns dos modelos disponíveis, assim como a sua velocidade de treino, a mAP e o tipo de deteção.

Tabela 3.1: Características de alguns modelos pré-treinados [91].

Nome do Modelo	Velocidade (Iteração/ms)	COCO mAP	Saída
SSD MobileNet V2 FPNLite 640x640	39	28.2	Caixas
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Caixas/Pontos Chave
EfficientDet D7 1536x1536	325	51.2	Caixas
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Caixas
Mask R-CNN Inception ResNet V2 1024x1024	301	39.0/34.6	Caixas/Máscara

Tendo em conta as limitações abordadas na subsecção anterior, a arquitetura selecionada foi a SSD MobileNet V2 FPNLite 640x640 [96], uma vez que trabalha com imagens de dimensões relativamente pequenas, isto é, 640x640 pixels, possui uma mAP de 28,2 e uma velocidade de treino de 39 ms. Por outras palavras, este modelo é rápido no momento de treino, o que, como mencionado anteriormente, é algo valorizado. No entanto, quando comparado a modelos mais complexos, pode apresentar uma precisão inferior no momento de identificação.

A arquitetura *Single-Shot Detector* (SSD) [97] juntamente com a combinação do extrator de recursos MobileNet V2 [43] e da fusão de recursos *Feature Pyramid Network* (FPN) [98] constituem o modelo pré-treinado SSD MobileNet V2 FPNLite 640x640. Este modelo é treinado no conjunto de dados COCO [99], é constituído por 118000 imagens e 80 categorias diferentes. Desta forma, as camadas de extração de recursos são pré-treinadas no conjunto de dados COCO, o que implica que estas camadas consigam reconhecer inúmeras características nas imagens de entrada. Contudo, as camadas de previsão, responsáveis por prever a caixa delimitadora dos objetos de interesse, são inicializadas aleatoriamente e afinadas durante o processo de treino. Este processo permite que o modelo se adapte às características das categorias pretendidas no novo conjunto de dados enquanto usa parte do conhecimento adquirido a partir do treino no conjunto de dados COCO.

3.3.4 Adaptações feitas para o modelo pré-treinado

Antes do início do treino é necessária a criação de dois ficheiros. O primeiro é o mapa de rótulos (*Label Map*). Este, consiste num ficheiro de texto simples, onde são definidos os nomes das diferentes classes. Os rótulos estabelecidos neste ficheiro e os atribuídos no momento das anotações devem ser iguais. O segundo, é um ficheiro TFRecord que contém as imagens e as anotações. O TFRecord é um formato simples utilizado para armazenar uma sequência de registos binários, que foi otimizado para o uso do Tensorflow. No caso de conjuntos de dados grandes, usar um ficheiro binário para armazenamento do conjunto, pode ter um impacto significativo no tempo de treino do modelo. Para além disso, dados binários ocupam menos memória. Foi criado um ficheiro TFRecord para cada conjunto de dados apresentado [100].

Por fim, é necessário atualizar um ficheiro com as configurações do sistema que está contido no ficheiro do modelo pré-treinado. Este ficheiro pode ser dividido em três partes, a configuração do modelo, a configuração do treino e a configuração de validação.

3.3.5 Configuração do modelo

A configuração do modelo especifica os detalhes do modelo de deteção de objetos que se pretende treinar. É de notar que a maioria dos parâmetros são *default* dos fornecidos pelo Tensorflow [76]. Os principais parâmetros são:

- num_classes: O número de classes que se pretende detetar. Para os modelos com cinco ferramentas diferentes é definido como cinco e para os que apenas possuem três ferramentas é definido como três;
- image_resizer: Uma vez que as redes neuronais são treinadas com um número fixo de entradas (capítulo 2), é preciso garantir que a dimensão das imagens do modelo é a correta. Uma vez que o modelo é treinado com imagens de dimensões 640x640, este parâmetro redimensiona as imagens para as dimensões corretas;
- feature_extractor: A configuração do extrator de recursos é baseada na arquitetura *Single Shot Multibox Detector* [97]. Os aspetos mais relevantes sobre esta configuração são:
  - Tipo: ssd_mobilenet_v2_fpn_keras, que especifica o tipo de extrator de características. Neste caso, é uma rede de características em pirâmide [98] baseada em MobileNetV2 [43] implementado em Keras [79];
- box_coder: Este módulo é responsável por codificar e descodificar as caixas delimitadoras (o usado é o faster_rcnn_box_coder [101]);
- anchor_generator: Este módulo é responsável por gerar caixas âncora;
- matcher: Este módulo é o módulo responsável por associar caixas de referência aos caixotes âncora;
- similarity_calculator: Este módulo é responsável por calcular a similaridade entre a caixa de referência e as caixas âncora;
- train_config: Aqui é definida a configuração do treino. Nomeadamente o *batch size*, o número de iterações de treino, o otimizador de treino, o número de iterações de treino de aquecimento, etc.

Tabela 3.2: Descrição dos modelos.

Modelo	Conjunto de dados	*Batch size*	Iterações de treino	Épocas	Taxa de aprendizagem
1	1	4	67k	11	0,08
2	2	4	26k	8	0,08
3	3	4	22k	8	0,08
4	4	4	31k	8	0,08
5	5	4	25k	6	0,08

3.3.6 Modelos

Foram treinados múltiplos modelos onde se alteraram vários parâmetros. No entanto, apenas serão abordados os modelos que obtiveram os melhores resultados. Na Tabela 3.2 são apresentados os diferentes modelos e os respetivos parâmetros - o conjunto de dados, o *Batch size*, o número de iterações de treino, o número de épocas e a taxa de aprendizagem. Inicialmente, foi estudado o impacto da variação do valor do *Batch size* e verificou-se que os *Batch size* possíveis estavam entre um e quatro. Como tal, alguns testes foram feitos para perceber o melhor valor e concluiu-se que com o valor de *Batch size* igual a quatro, as redes necessitavam de menos iterações de treino para completar a sua fase de treino. Então, todos os modelos aqui apresentados possuem o *Batch size* igual a quatro.

É possível observar na Tabela 3.2 que o modelo 1 possui um valor de iterações de treino significativamente superior aos demais. Isto pode ser justificado pela dimensão do conjunto de dados superior aos restantes e à necessidade de um treino mais longo para convergir os resultados. Os modelos 2, 3 e 4, devido a terem conjuntos de dados de dimensões semelhantes e tendo o objetivo de comparar os três, possuem o mesmo número de épocas e um número semelhante de iterações de treino. Por fim, o modelo 5, constituído por um conjunto de dados real, é aquele que se espera que tenha uma convergência mais rápida e, como tal, é o que teve menos iterações de treino.

Durante o processo de treino, a cada mil iterações, o modelo era avaliado no conjunto de dados de validação e era gravado um *checkpoint* com o estado atual do modelo. Desta forma, é possível ver dinamicamente a progressão de cada modelo para saber se o modelo está a exibir um comportamento de sobreajuste ou se ainda precisa de mais iterações de treino.

Transfer learning

Como abordado no Capítulo 2, o *transfer learning* traz um conjunto de vantagens para este tipo de sistemas, nomeadamente, melhores resultados, a necessidade de um conjunto de dados mais pequeno e, algo valorizado para este sistema, tempo

reduzido de treino. Combinando este conceito com o facto de que na literatura já foi explorada a adição de imagens geradas sinteticamente a conjuntos de dados reais [54, 60, 61] com resultados geralmente superiores a obtidos com conjuntos de dados puramente reais, gera-se aqui uma oportunidade de otimização.

A diferença entre o modelo 1 e o modelo 5 é que, neste último, os pesos e *bias* estarão melhor ajustados para a deteção das ferramentas, uma vez que este vai ser treinado com imagens reais. Contudo, se o treino do modelo 1 correr como expectável, no fim, os valores dos pesos e *bias* vão estar próximos dos valores do fim do treino do modelo 5. Como tal, se o modelo 1 continuar a ser treinado, mas desta vez com imagens reais, visto que os pesos e *bias* já estão relativamente otimizados para a tarefa de deteção e classificação, será necessário um treino menos extensivo do que quando comparado com o modelo 5, para alcançar os valores ótimos dos pesos e *bias*.

Uma vez que no *workflow* proposto, apresentado no capítulo 1, a rede neuronal real será sempre treinada depois da rede neuronal sintética, surge aqui uma oportunidade de otimização e a criação do modelo 6, que é na sua essência uma continuação do modelo 1. Quando este estiver treinado, será criada uma cópia dele e será feito um treino adicional com o conjunto de dados 5, com uma taxa de aprendizagem igual a 0,08, durante duas épocas com um *Batch size* de quatro, ou seja, durante 8100 iterações de treino.

O objetivo do modelo 6 é perceber se é melhor treinar, de "raiz"com o conjunto de dados reais (modelo 5) ou se se consegue alcançar o mesmo resultado com um treino com imagens sintéticas a finalizar com um treino com imagens reais.

3.4 Sistema *Pick and Place*

O sistema *Pick and Place* foi montado dentro dos laboratórios do INEGI em cima de uma mesa, de um lado foi colocada uma câmara WC001A-4 [102] no topo de um perfil de alumínio com a lente paralela à mesa e, no outro lado, diretamente à frente da câmara, foi colocado um robô KUKA LBR IIWA 14 R820 [103]. Ambos os equipamentos foram fornecidos pelo INEGI. Uma vez que o efetor final do robô não se encontrava disponível, apenas se estudou a deslocação do robô até à posição das ferramentas e não a capacidade de pegar nestas. Na Figura 3.12 é possível observar o *set-up* utilizado neste projeto, na imagem esquerda é apresentada uma visão de topo da mesa de trabalho e na imagem da direita uma visão lateral.

Este sistema encontra-se dividido em três partes: o sistema de visão, o sistema de controlo e a comunicação. O sistema de visão tem como função detetar e identificar a ferramenta, enquanto que o sistema de controlo é responsável pelo deslocamento do robô até à posição onde esta se encontra. Por fim, a comunicação consiste na

Figura 3.12: *Set-up* usado para o sistema o *pick and place*.

troca de informação entre o sistema de visão e o sistema de controlo. Nas próximas secções serão abordados cada um destes sistemas.

3.4.1 Sistema de visão

Como mencionado anteriormente, o objetivo do sistema de visão é identificar a ferramenta e a sua posição. Com o desenvolvimento das redes neuronais este sistema é simplificado. Para a identificação da ferramenta é necessário utilizar cada uma das imagens obtidas pela câmara e passá-las pelo modelo. Se a identificação for bem sucedida irá surgir, na imagem, uma caixa delimitadora contendo a ferramenta, a respetiva identificação textual e a probabilidade da deteção estar correta. Na Figura 3.13 é ilustrado o momento de deteção de uma ferramenta efetuada por um modelo. Neste caso, é identificado um Roquete e a probabilidade da identificação estar correta é de 100%.

Figura 3.13: Deteção de um Roquete feita por um modelo.

Quando é inserida uma imagem no modelo, este vai retornar uma matriz contendo todas as identificações que fez. Ou seja, frequentemente, esta matriz vem com múltiplas identificações incorretas com probabilidades de identificação baixas. Para contornar este problema foram tomadas duas decisões. A primeira é que o modelo

apenas irá fazer uma identificação por imagem. Desta forma, só será apresentado ao utilizador a identificação com a maior probabilidade de identificação. A segunda é que, se esta probabilidade de identificação for inferior a 50%, a identificação é ignorada. A posição da ferramenta é obtida através da caixa delimitadora, uma vez que o centro desta corresponde ao centro da ferramenta.

Desta forma, encontram-se reunidos três parâmetros importantes para o momento de recolha da ferramenta: o tipo de ferramenta e a sua posição na imagem dada pelas coordenadas espaciais (x, y). No entanto, se a ferramenta se encontrar inclinada, isto é, não completamente vertical, poderão existir problemas na recolha da ferramenta dado que é necessário que o efetor final se alinhe com ela. Para evitar esta situação, a inclinação da ferramenta será calculada para definir a rotação que o efetor final terá de realizar.

Uma vez que todas as ferramentas possuem uma forma relativamente simétrica, a inclinação da peça é calculada como o ângulo entre um plano vertical e o eixo de simetria da peça. Para encontrar o eixo de simetria da ferramenta, começa-se por extrair os seus contornos, recorrendo à função *Canny*. Para reduzir o custo computacional desta operação, define-se uma região de interesse, neste caso, a caixa delimitadora fornecida pela identificação do modelo. A Figura 3.14 ilustra este processo. A caixa delimitadora é também usada para identificar se a peça está orientada numa posição mais próxima da vertical ou da horizontal. Se a caixa for mais alta do que larga significa que a ferramenta se encontra na vertical, no entanto, se a caixa for mais larga do que alta significa que a ferramenta se encontra na horizontal.

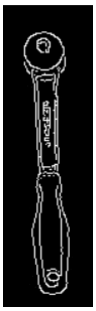


Figura 3.14: Contornos de um Roquete calculados pela função *Canny*.

Quando a ferramenta se encontra numa posição mais próxima da vertical, para calcular o ângulo entre um plano vertical e o eixo de simetria da peça, começa-se por encontrar os pontos extremos do contorno ao longo de uma linha horizontal na parte superior da ferramenta. De seguida, faz-se a média desses pontos e obtém-se um ponto do eixo de simetria da ferramenta. Este processo é repetido na parte inferior da ferramenta (representado pelas linhas brancas na parte inferior na Figura 3.15). São, assim, determinados dois pontos do eixo de simetria da ferramenta com coordenadas (x_1, y_1) e (x_2, y_2).

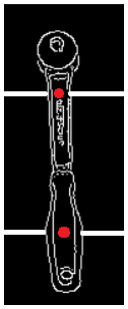


Figura 3.15: Pontos de interesse para o cálculo do declive quando a ferramenta se encontra mais próxima da vertical.

Se a ferramenta se encontrar numa posição mais próxima da horizontal, passa-se a fazer uma busca ao longo de duas linhas verticais, uma à esquerda e outra à direita, como ilustrado na Figura 3.16.

Figura 3.16: Pontos de interesse para o cálculo do declive quando a ferramenta se encontra na horizontal.

Com os pontos determinados, o declive pode ser dado por:

$$\theta = \tan^{-1}(\frac{y_2 - y_1}{x_2 - x_1}), \tag{3.1}$$

onde (x_1, y_1) é o ponto que está mais próximo do topo da imagem, ou mais à direita, e (x_2, y_2) é o mais próximo do fundo, ou mais à esquerda. O valor será usado para definir a rotação do efetor final de modo a garantir o alinhamento com a ferramenta.

Após a obtenção da posição, da rotação e do tipo de ferramenta, é necessário criar um servidor local OPC UA [104]. O OPC é um dos padrões de comunicação mais importantes para a Indústria 4.0 e para a *Internet of Things*. Com o OPC, o acesso a máquinas, dispositivos e outros sistemas no ambiente industrial é padronizado e permite a troca de dados de forma semelhante e independente do fabricante. Neste contexto, o UA refere-se à última especificação do padrão. Além de muitas outras melhorias, o OPC UA também suporta uma descrição semântica de dados. O Servidor OPC é a base da comunicação OPC. É um *software* que implementa o padrão OPC e, portanto, fornece as interfaces OPC padronizadas para o mundo exterior. Os Servidores OPC são fornecidos por diferentes partes [104]. São criadas cinco variáveis no servidor OPC UA:

- x: indica a coordenada x, do centro da ferramenta, na imagem;
- y: indica a coordenada y, do centro da ferramenta, na imagem;

- TipoFerr: indica o tipo da ferramenta (Alicate, Chave Inglesa, etc);
- rotação: indica a rotação que o efetor final irá fazer para recolher a ferramenta;
- status: indica que o robô está em missão (*True*) ou à espera de uma nova ferramenta (*False*).

Nesta aplicação, a profundidade não será calculada, uma vez que as ferramentas estão posicionadas em cima de uma mesa plana e, por isso, a profundidade permanecerá constante.

Na Figura 3.17 é apresentado o fluxograma do sistema visão.

3.4.2 Comunicação

Depois de adquirir a posição da ferramenta, é necessário que essa informação seja transferida para o robô. Como explicado na secção anterior, utilizou-se um servidor OPC UA para publicar a informação adquirida pelo sistema de visão. De seguida usou-se o Node-Red [105] para subscrever ao servidor, com o objetivo de ter acesso às variáveis publicadas. O Node-RED é uma ferramenta de programação para conectar dispositivos de *hardware*, API e serviços *online*. Este fornece um editor baseado em navegador que facilita a conexão de fluxos [105].

O Node-Red foi introduzido neste sistema por dois motivos. Apesar de não ter sido implementada uma base de dados, o Node-Red permitirá a sua futura inclusão. O outro motivo é o interesse do INEGI, uma vez que pretendiam estudar a ligação entre o Node-Red e o TwinCAT [106], um *software* de automação que será abordado mais à frente.

Na Figura 3.18 estão representadas as funções usadas para subscrever ao servidor OPC UA. Neste *software* foram usadas cinco funções:

- timestamp: esta função define o intervalo de tempo entre a atualização das variáveis. Foi definida como um segundo, o que implica que de um em um segundo este sistema vá buscar os valores publicados no servidor;
- bloco de variáveis: aqui é definido qual das variáveis é que está a ser atualizada. Para tal, é necessário introduzir neste bloco o nome definido no servidor e o id atribuído à dada variável;
- OPC UA *Client*: este bloco é o que vai interagir com o servidor. É necessário introduzir neste bloco o endereço do servidor e que tipo de interação vai ter com o sistema (ler, pesquisar, etc). A interação definida foi ler e é aqui que o valor da variável que é cedida a este bloco vai ser atualizada;
- bloco function: este bloco serve como ponte entre o bloco *client* e o bloco Ads;

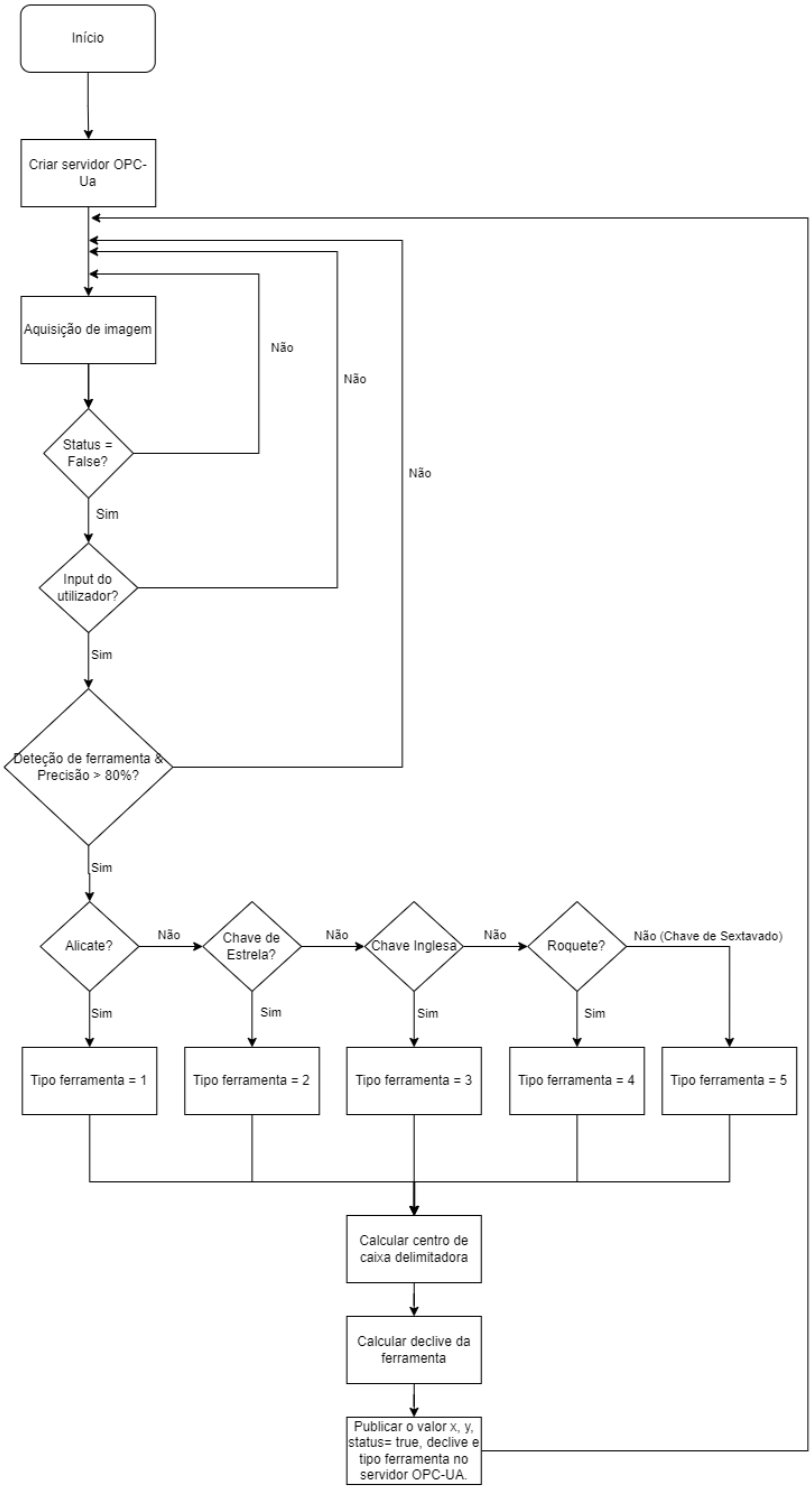


Figura 3.17: Fluxograma do sistema de visão.

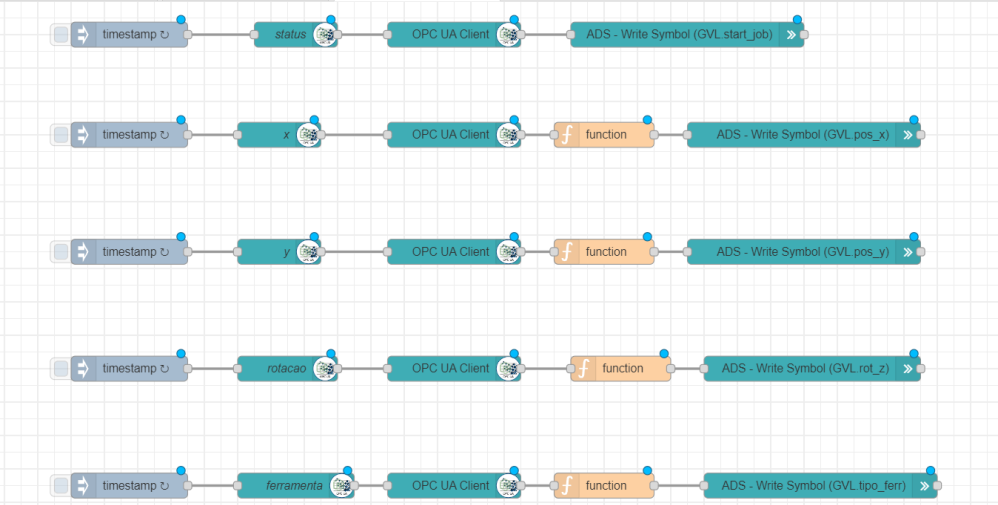


Figura 3.18: Funções usadas para subscrever ao servidor OPC UA.

- ADS write symbol: este bloco usa um nó ads-client para forçar os valores lidos do servidor OPC UA nas variáveis criadas no TwinCAT.

As variáveis adquiridas no sistema de visão são publicadas num servidor OPC UA, são lidas pelo nó criado no Node-Red e, de seguida, são enviadas para o TwinCAT que faz a ponte com o *software* do braço mecânico. O TwinCAT é um *software* de automação industrial desenvolvido pela empresa alemã Beckhoff Automation[2]. Ele é usado para controlar sistemas de produção em diversas indústrias incluindo automação, de alimentos e bebidas e de embalagens. O TwinCAT é uma solução flexível e escalável que permite o controlo de máquinas e processos de produção de forma eficiente e confiável. É amplamente utilizado e é uma escolha popular para aqueles que visam otimizar processos de produção [106]. Como mencionado, neste caso o TwinCAT irá servir apenas como ponte entre o sistema de visão e o sistema de controlo do robô, mais concretamente, irá receber as variáveis fornecidas pelo sistema de visão e fornecê-las ao sistema de controlo.

3.4.3 Sistema de controlo

Uma vez que o tipo de ferramenta e a sua posição são conhecidos pelo robô, é possível recolher a ferramenta de forma autónoma. No entanto, é necessário fazer algumas operações primeiro. A posição (x, y) enviada pelo sistema de visão, é relativa ao referencial da câmara. Contudo, para o robô ir buscar as ferramentas de forma correta, é necessário passar estas coordenadas para o referencial do robô. O primeiro passo consiste em mudar o referencial do robô de forma a que a posição (0, 0) em x, y coincida com o canto superior esquerdo da imagem capturada pela

[2]https://www.beckhoff.com/de-de/

câmara. Ou seja, a posição (0,0) do robô fica diretamente por cima da célula (0,0) da matriz imagem.

Como a câmara está paralela à mesa, é possível estabelecer uma relação aproximada entre uma posição na imagem e uma posição do robô. Para tal, é necessário saber a posição do robô quando está diretamente por cima da célula (639, 639) que corresponde ao pixel do canto inferior direito da imagem e será designada por (P_{x1}, P_{y1}). A posição (P_x, P_y) pode ser aproximada por:

$$P_x = \frac{x \times P_{x1}}{640} \tag{3.2}$$

$$P_y = \frac{y \times P_{y1}}{640} \tag{3.3}$$

onde, P_x e P_y é a posição da ferramenta no referencial do robô, x e y é a posição da ferramenta, na imagem, calculada pelo sistema de visão e 640 é a dimensão da imagem.

Com a posição da ferramenta calculada, o próximo passo consiste na definição de algumas posições intermédias entre a posição do robô e a ferramenta. Definiu-se que a posição do robô quando está à espera da informação relativa à posição da ferramenta é (0,0, 10), quando recebe a ordem para ir buscar a peça desloca-se até (x,y,10), onde roda o efetor final conforme a rotação recebida do sistema de visão e depois desce para recolher a ferramenta.

Neste capítulo foi apresentado o desenvolvimento do projeto. Inicialmente, foi definido o âmbito da dissertação, nomeadamente a criação de um sistema *pick and place* que, com recurso a uma rede neuronal treinada com imagens sintéticas, terá a capacidade de identificar objetos e deslocar-se até eles. De seguida, definiu-se os objetos a serem modelados e o *software* de modelação. O próximo passo consistiu na criação e descrição de diferentes conjuntos de dados, bem como o que se pretendia de cada um. Com os conjuntos de dados definidos, foi realizado um pequeno estudo de *softwares* de aprendizagem de máquina, com o intuito de escolher o mais vantajoso e, ultimamente, definiu-se o Tensorflow [76]. Apresentou-se também o *hardware* utilizado neste projeto e as restrições que este implicou. Também foi definida a utilização de modelos pré-treinados e da arquitetura SSD MobileNet V2 FPNLite 640x640 [43]. Foram apresentados os modelos utilizados e os parâmetros sob qual eles foram treinados. Por último, apresentou-se a arquitetura geral do sistema *pick and place* bem como as tecnologias usadas.

Capítulo 4

Resultados

Neste capítulo serão apresentados e analisados os resultados obtidos para os modelos treinados, comparando-os entre si. Para a análise dos resultados, serão realizadas três avaliações distintas por modelo, sendo duas delas com recurso ao Tensorboard [76]. Conforme mencionado no capítulo 2, o desempenho de cada classe será avaliado utilizando as métricas PASCAL VOC [107]. Cada classe terá a sua precisão média calculada com uma taxa de união de 50%, baseando-se no conjunto de dados de validação. Desta forma, é possível conhecer o desenvolvimento do modelo e identificar em qual *checkpoint* o seu desempenho é o melhor. De seguida, conhecido o *checkpoint* onde o desempenho do modelo é máximo, serão realizados mais dois testes. O primeiro será uma avaliação adicional com o Tensorboard [76], utilizando as métricas COCO [99], que avaliará a precisão média com índices de sobreposição de 50%, 75% e desde 50% até 95%, bem como a taxa de sensibilidade média. De seguida serão realizados 20 testes por ferramenta e por modelo. Cada teste consiste em posicionar uma determinada ferramenta numa posição aleatória em frente à câmara, verificando se a identificação foi realizada corretamente e se o braço mecânico se deslocou adequadamente até a ferramenta. Para a apresentação dos resultados recorrer-se-á a um código de cores com o intuito de facilitar a interpretação da mesma:

- Verde: A ferramenta foi devidamente identificada e o braço mecânico deslocou-se até ela;
- Amarelo: O sistema não conseguiu identificar a ferramenta;

- Vermelho: O sistema identificou incorretamente a ferramenta e o braço deslocou-se corretamente até ela.

É de notar que, no código de cores, não existe nenhuma cor para o caso do braço não se deslocar corretamente até a ferramenta, uma vez que esta situação nunca ocorreu nos testes.

4.1 Apresentação de resultados

4.1.1 Modelo 1

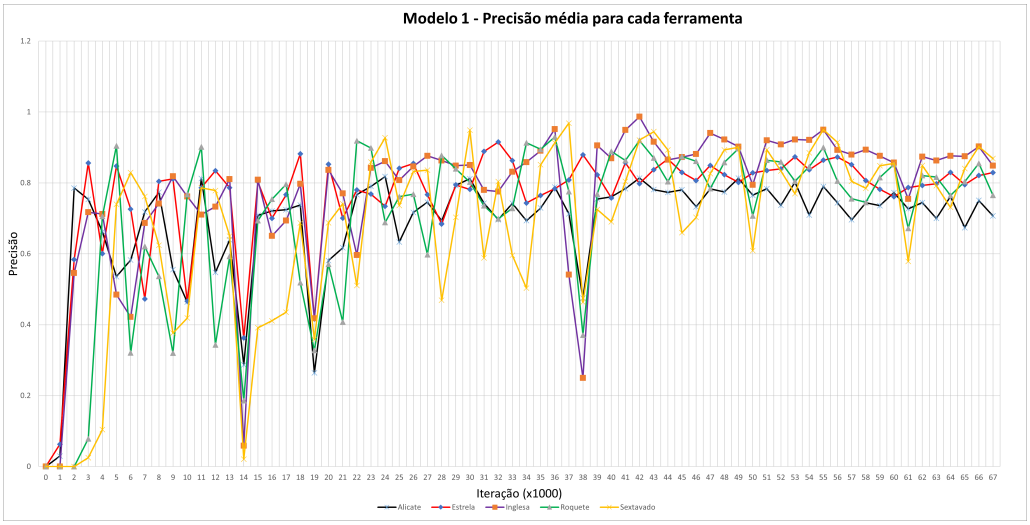


Figura 4.1: Resultados de precisão média de cada ferramenta ao longo do treino do modelo 1.

Analisando a Figura 4.1 é possível verificar um certo nível de convergência a partir das 42000 iterações, contudo, devido à quantidade de dados apresentados no gráfico, esta perceção torna-se difícil. Como tal, ir-se-á recorrer à precisão média para as diferentes ferramentas em cada iteração.

Na Figura 4.2 está representada a precisão média ao longo do treino do modelo. Reforça-se a ideia de que o modelo apresenta uma convergência de resultados a partir das 42000 iterações. Verifica-se também um ligeiro pico de desempenho na iteração 55000 sendo neste *checkpoint* que se irão realizar os restantes testes. Neste *checkpoint* o Alicate apresenta uma precisão de 0,79, a Chave de Estrela de 0,86, a Chave Inglesa de 0,95, o Roquete de 0,9 e a Chave de Sextavado de 0,95.

Realizando uma nova avaliação, mas desta vez com as métricas COCO , a precisão média com um índice de sobreposição de 50% é de 0,89, com 75% de 0,857 e com 50% até 95% de 0,749, a taxa de sensibilidade é 0,846 e a pontuação F1 é 0,867.

Analisando a Tabela 4.1, dos 100 testes efetuados ao modelo, houve cinco deteções incorretas, das quais três se verificaram na identificação do Alicate. Analisando

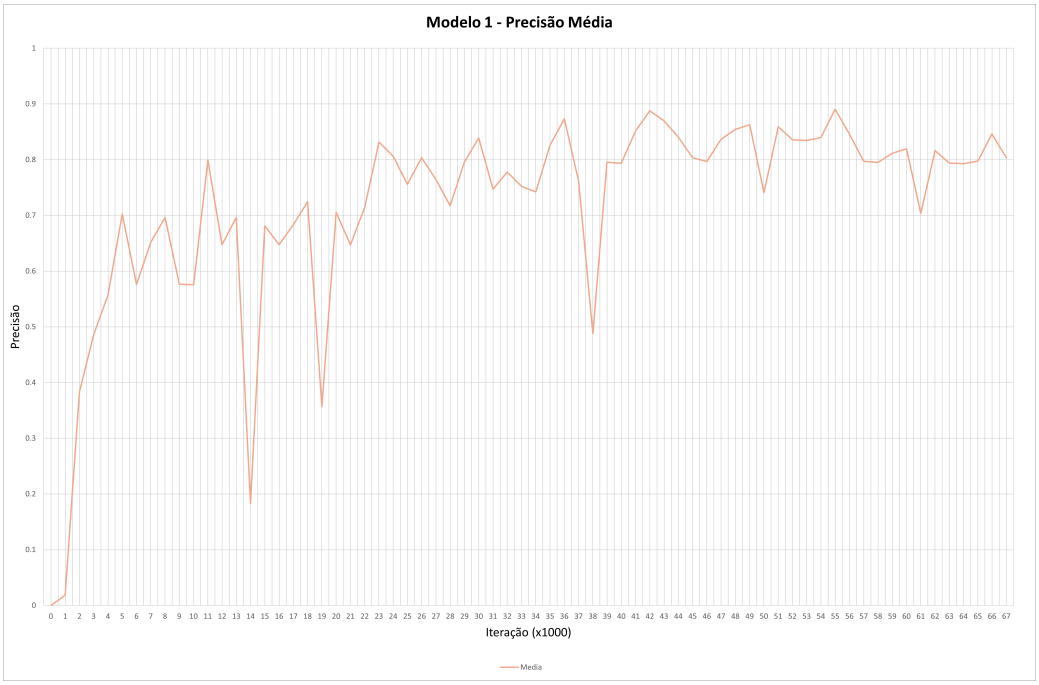


Figura 4.2: Média da precisão ao longo do treino do modelo 1.

a tabela e o gráfico da precisão média apresentado na Figura 4.1, é possível concluir que este modelo apresenta o pior desempenho para o Alicate e apresenta o melhor resultado para o Roquete e para a Chave de Sextavado. Relativamente ao sistema *Pick and Place*, em todos os testes conseguiu deslocar-se com sucesso até às ferramentas.

4.1.2 Modelo 2

Na Figura 4.3 está ilustrada a evolução da precisão média para cada ferramenta durante o processo de treino do modelo 2. A linha preta representa a precisão média para o Alicate, a vermelha para a Chave de Estrela e a verde a para o Roquete. Analisando o gráfico é percetível que o modelo apresenta comportamento semelhante para as três ferramentas. O início do treino é marcado por uma subida acentuada na precisão alcançada por cada classe, a meio do gráfico existe uma oscilação nos valores da precisão e no fim há uma convergência para um valor, indicando que, a partir deste ponto, o modelo não vai melhorar. É percetível que a convergência é mais rápida para o Alicate, e que a Chave de Estrela é a que a leva uma convergência mais lenta do modelo, com um comportamento mais oscilatório durante a fase intermédia do treino. As precisões médias convergem a partir das 20000 iterações de treino. Na iteração 26000, o modelo apresenta uma precisão de 0,8231 para o Alicate, de 0,8232 para a Chave de Estrela e de 0,8256 para o Roquete, sendo o modelo obtido nesta iteração o escolhido para os restantes testes.

Tabela 4.1: Modelo 1 - Resultados obtidos na fase de testes (o valor apresentado para cada teste indica o índice de confiança na identificação).

Teste	Alicate	Estrela	Inglesa	Roquete	Sextavado
1	100	100	95	100	96
2	100	100	100	100	97
3	100	100	100	100	98
4	100	100	100	100	96
5	100	100	Sextavado 87	100	96
6	100	100	99	100	96
7	100	100	98	100	93
8	100	100	99	99	94
9	Sextavado 98	100	97	100	91
10	66	100	100	100	93
11	77	100	100	100	94
12	Roquete 97	100	100	100	89
13	99	100	100	100	87
14	100	100	100	100	88
15	100	Roquete 88	100	100	88
16	100	100	100	100	95
17	87	100	100	100	95
18	65	100	100	100	97
19	Roquete 99	100	99	100	97
20	100	100	99	100	96

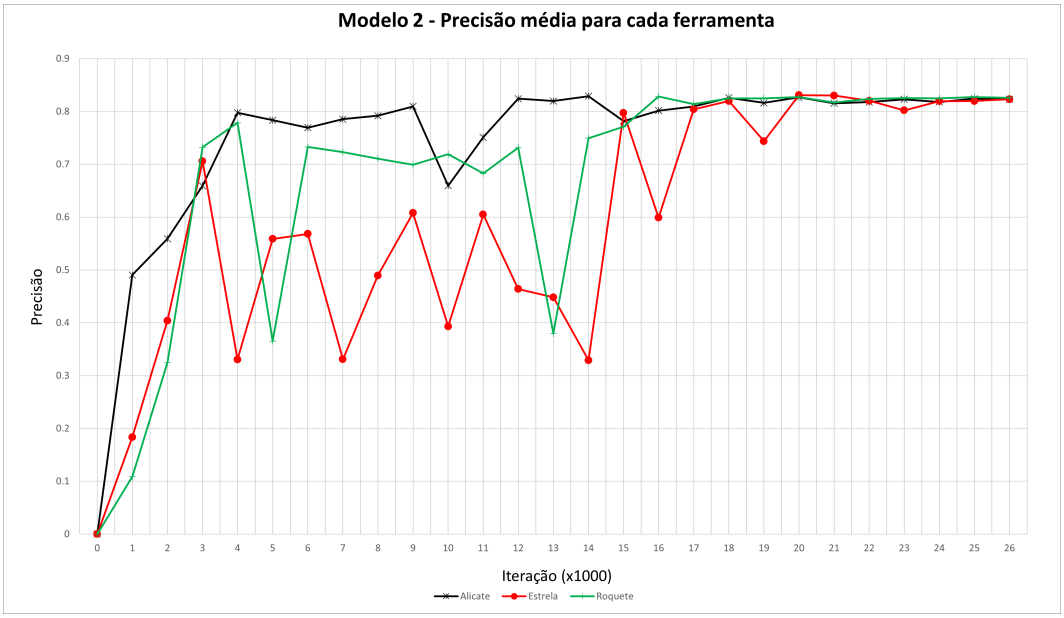


Figura 4.3: Resultados de precisão média para cada ferramenta ao longo do treino do modelo 2.

Tabela 4.2: Modelo 2 - Resultados obtidos na fase de testes (o valor apresentado para cada teste indica o índice de confiança na identificação).

Teste	Alicate	Estrela	Roquete
1	77	52	96
2	92	Alicate - 65	96
3	87	65	96
4	100	63	94
5	100	74	98
6	100	Roquete - 75	97
7	100	Roquete - 70	96
8	99	87	84
9	99	Roquete 62	98
10	98	76	100
11	97	76	99
12	100	74	100
13	100	87	99
14	100	70	99
15	100	68	99
16	100	80	98
17	100	78	98
18	100	58	98
19	100	70	99
20	82	93	99

Utilizando as métricas de avaliação COCO, nesta iteração, a precisão média com um índice de sobreposição de 50% é de 0,8239, com 75% de 0,7523 e com 50 até 95% de 0,6832, a taxa de sensibilidade é de 0,7833 e a pontuação F1 é 0,8031.

Na Tabela 4.2 estão apresentados os resultados dos 20 testes por ferramenta. Analisando a tabela é visível que o pior desempenho ocorre com a Chave de Estrela. Não só é a única ferramenta que resultou em identificações erradas, como também é o caso onde, para as identificações corretas, o índice de confiança é significativamente mais baixo. Já o Alicate e o Roquete foram identificados corretamente nos 20 testes com um índice de confiança elevado. Combinando o conhecimento adquirido pelos testes feitos e com o gráfico apresentado na Figura 4.3, é visível que para a Chave de Estrela o modelo teve, no momento de treino, dificuldade em convergir e, no momento de teste, o pior desempenho comparativamente com as outras ferramentas. O sistema *Pick and Place* não apresentou falhas conseguindo-se deslocar, com sucesso, até às ferramentas em todos os testes.

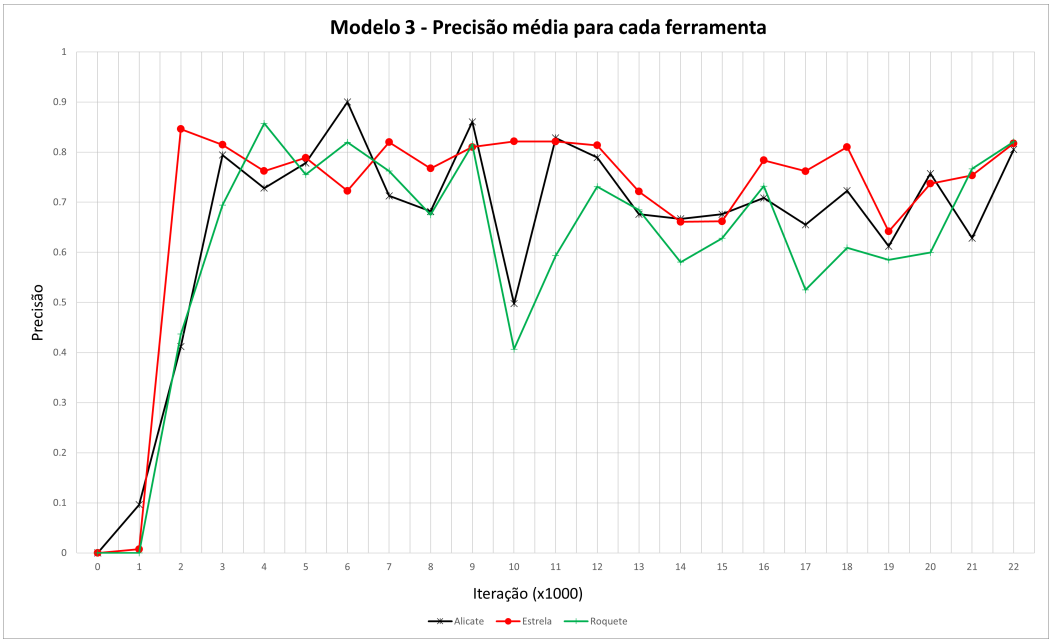


Figura 4.4: Resultados de precisão média para cada ferramenta ao longo do treino do modelo 3.

4.1.3 Modelo 3

O gráfico na Figura 4.4, relativo ao treino do modelo 3, apresenta um comportamento ligeiramente distinto dos anteriores. É visível que não existe um estado de convergência como nos outros modelos. Contudo, dado que a avaliação do desempenho na iteração 22000 é satisfatório, com a precisão do Alicate em 0,8052, da Chave de Estrela em 0,8176 e a do Roquete em 0,8205, não se treinou um modelo novo.

Com recurso à avaliação com as métricas COCO, a precisão média com um índice de sobreposição de 50% é de 0,8144, com 75% de 0,511 e com 50% até 95% de 0,499, uma taxa de sensibilidade de 0,721 e uma pontuação F1 de 0,7493. A queda entre a precisão média com um índice de sobreposição média de 50% e com um índice de sobreposição de 70% é mais significativa do que em outros modelos.

Analisando a Tabela 4.3, verifica-se que, mais uma vez, o sistema *pick and place* voltou a deslocar-se, corretamente, até as peças em todos os testes. Nos 60 testes efetuados, sete não foram adequadamente identificados, sendo das quais quatro vezes na identificação da ferramenta Chave de Estrela. Enquanto a Chave de Estrela apresenta o pior desempenho, nenhuma das restantes ferramentas se destaca pelo seu bom desempenho, tendo tanto o Alicate como o Roquete desempenhos semelhantes.

Tabela 4.3: Modelo 3 - Resultados obtidos na fase de testes (o valor apresentado para cada teste indica o índice de confiança na identificação).

Teste	Alicate	Estrela	Roquete
1	90	100	100
2	Roquete 77	Roquete - 88	100
3	94	100	100
4	83	100	100
5	100	10	100
6	100	Roquete - 75	100
7	100	Roquete - 87	99
8	100	87	100
9	100	100	100
10	100	100	99
11	100	100	100
12	100	99	Estrela 99
13	100	100	100
14	100	100	100
15	100	100	96
16	100	100	97
17	99	100	98
18	Roquete 85	100	100
19	100	Roquete 88	98
20	82	100	99

4.1.4 Modelo 4

O modelo 4 é uma combinação entre o modelo 2 e o modelo 3. Na Figura 4.5 está representada a evolução da precisão média, por cada 1000 iterações de treino, para cada ferramenta. No início verifica-se uma grande oscilação até que o modelo começa a convergir. A convergência dá-se a partir da iteração 22000, sensivelmente. Como tal, irá ser usado o *checkpoint* criado na iteração 27000, onde para o Alicate a precisão média é 0,95, para a Chave de Estrela é 0,9445 e para o Roquete é 0,9934.

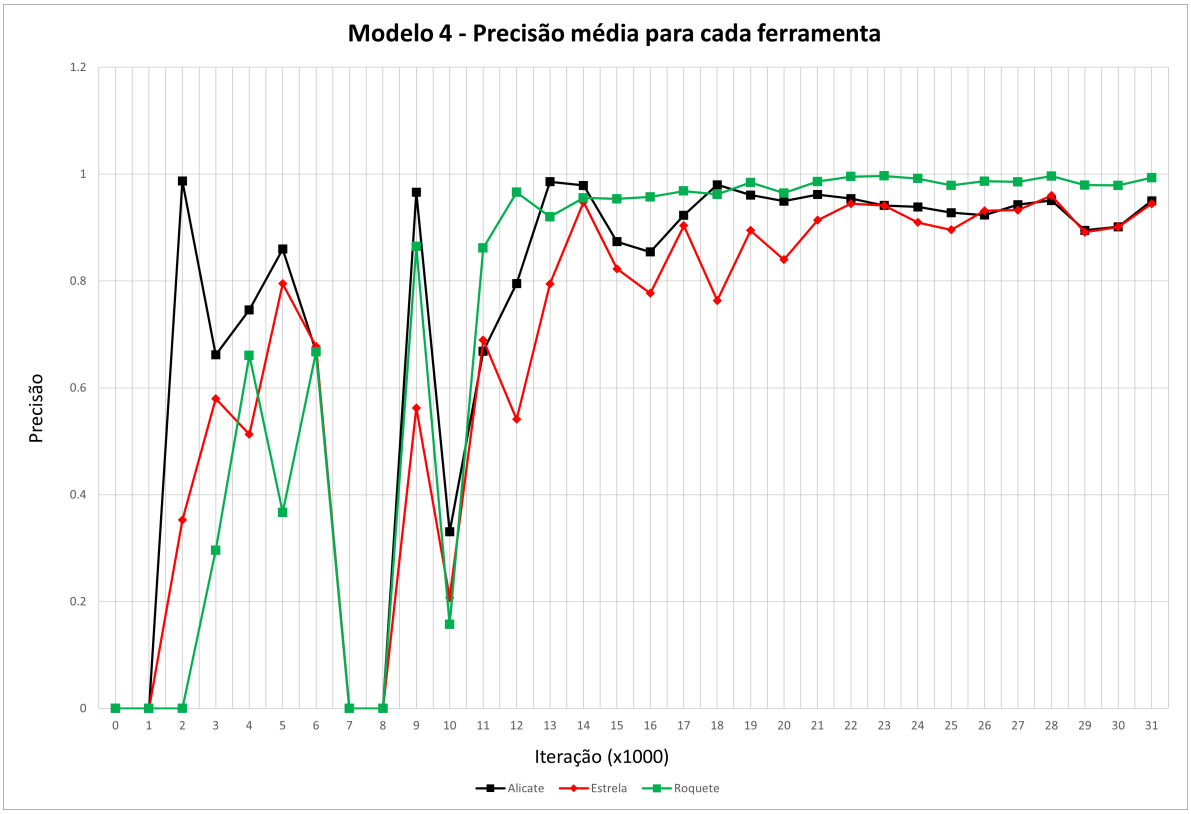


Figura 4.5: Resultados de precisão média para cada ferramenta ao longo do treino do modelo 4.

Com recurso às métricas de avaliação COCO, a precisão média com um índice de sobreposição de 50% é de 0,96, com 75% é de 0,789 e com 50% até 95% é de 0,747, a taxa de sensibilidade de 0,804 e a pontuação F1 de 0,875.

O sistema *pick and place* voltou a conseguir deslocar-se, de forma correta, até às ferramentas em todos os testes. Relativamente à classificação dos objetos, analisando a Tabela 4.4 verifica-se que em todos os testes ocorreu uma identificação correta.

4.1.5 Modelo 5

O modelo 5, treinado com imagens reais, servirá de base de comparação com os restantes modelos. Na Figura 4.6 está ilustrada a forma como a precisão para cada ferramenta evoluiu.

Tabela 4.4: Modelo 4 - Resultados obtidos na fase de testes (o valor apresentado para cada teste indica o índice de confiança na identificação).

Teste	Alicate	Estrela	Roquete
1	99	92	100
2	98	93	99
3	97	91	100
4	97	95	99
5	100	83	100
6	100	97	100
7	100	95	100
8	100	97	100
9	100	97	99
10	100	95	100
11	100	89	99
12	100	98	100
13	100	87	99
14	100	88	100
15	100	98	99
16	100	100	100
17	100	100	100
18	100	100	100
19	100	99	99
20	100	95	100

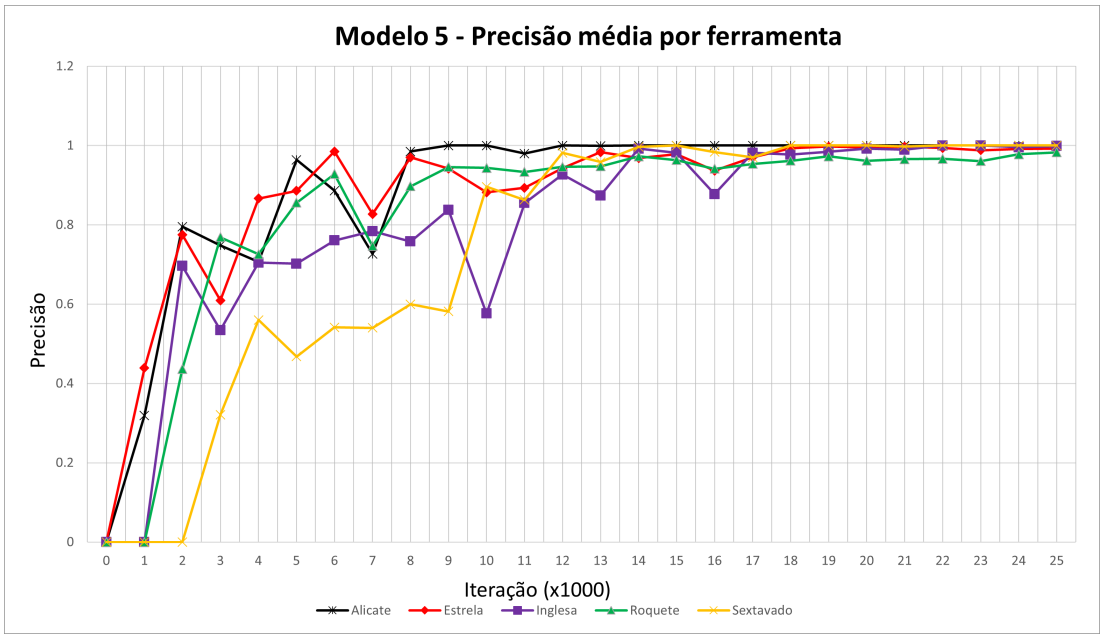


Figura 4.6: Resultados de precisão média para cada ferramenta ao longo do treino do modelo 5.

Tabela 4.5: Modelo 5 - Resultados obtidos na fase de testes (o valor apresentado para cada teste indica o índice de confiança na identificação).

Teste	Alicate	Estrela	Inglesa	Roquete	Sextavado
1	97	100	82	99	98
2	96	100	79	99	97
3	97	100	92	99	98
4	97	100	81	99	97
5	100	100	99	100	98
6	100	100	100	100	97
7	100	100	100	100	98
8	100	100	99	100	97
9	99	100	93	100	97
10	100	100	95	99	97
11	99	100	93	99	95
12	99	100	93	99	96
13	100	100	95	99	94
14	100	100	100	99	88
15	100	100	100	99	92
16	100	100	100	99	88
17	100	100	100	100	98
18	66	100	71	100	97
19	100	100	79	100	97
20	100	100	97	100	96

É possível, novamente, verificar o comportamento esperado da precisão do modelo durante o treino, sendo que a convergência acontece, sensivelmente, nas 17000 iterações. Os restantes testes foram feitos com o *checkpoint* correspondente às 25000 iterações, onde a precisão para o Alicate é de 1,0, para a Chave de Estrela é de 0,9921, para a Chave Inglesa é de 0,9988, para o Roquete é de 0,9829 e para a Chave de Sextavado é de 1,0. Para todas as classes, verifica-se que a precisão converge relativamente cedo, exceto no caso da Chave Inglesa que converge a partir da iteração 17000.

Analisando com as métricas COCO, a precisão média com um índice de sobreposição de 50% é de 0,995, com 75% é de 0,956 e com 50% até 95% é de 0,864, a taxa de sensibilidade é de 0,92 e a pontuação F1 é de 0,956.

Observando a Tabela 4.5, o comportamento obtido é o esperado dado que o modelo foi treinado com imagens reais das ferramentas. O comportamento observado neste teste também vai de encontro às métricas calculadas pelo *Tensorboard*. Mais uma vez o sistema *Pick and Place* não apresentou qualquer tipo de falha.

Para o modelo 5, foi realizado um teste adicional. Uma vez que o objetivo é comprovar que uma rede neuronal treinada com imagens geradas por computador

Tabela 4.6: Modelo 5 - Resultados obtidos no teste com imagens geradas por computador.

Teste	Alicate	Estrela	Inglesa	Roquete	Sextavado
1	99	68	NI	NI	89
2	99	NI	NI	NI	96
3	99	62	Sextavado 75	NI	95
4	62	88	94	Inglesa 94	93
5	91	NI	94	Inglesa 94	Alicate 79
6	98	NI	70	NI	86
7	99	NI	74	Inglesa 97	90
8	97	NI	96	Inglesa 80	78
9	91	62	97	96	62
10	98	88	NI	92	85
11	99	62	Sextavado 70	Alicate 92	97
12	97	88	76	Inglesa 98	92
13	99	NI	95	61	88
14	99	78	94	Inglesa 95	78
15	97	NI	75	NI	Alicate 97
16	99	NI	70	85	92
17	99	NI	93	Inglesa 85	89
18	62	52	96	Inglesa 79	78
19	85	55	79	Inglesa 98	88
20	72	68	58	91	89

consegue funcionar no mundo real, seria interessante verificar se o oposto também é verdade. Isto é, se uma rede neuronal treinada com imagens reais consegue identificar imagens dos objetos gerados por computador. Como tal, alimentou-se o modelo 5 com 20 imagens que constituíam parte do conjunto de dados de treino do modelo 1. Na Tabela 4.6 estão ilustrados os resultados obtidos.

Analisando a Tabela 4.6 é visível que a Chave de Estrela e o Roquete apresentam os piores resultados. Dos 40 testes feitos entre as duas, 10 geraram identificações incorretas, em 14 não foram identificadas e apenas 16 resultaram em identificações corretas.

4.1.6 Modelo 6

O modelo 6 resultou da continuação do treino do modelo 1. No fim do treino do modelo 1 treinado apenas com imagens sintéticas, cujos resultados estão apresentados numa subsecção anterior, continuou-se a treinar o modelo, mas agora com imagens reais. Na Figura 4.7 está ilustrada a forma como a precisão para cada ferramenta evoluiu, a partir da iteração 68000, altura em que o modelo previamente treinado com imagens sintéticas começa a ser treinado com imagens reais.

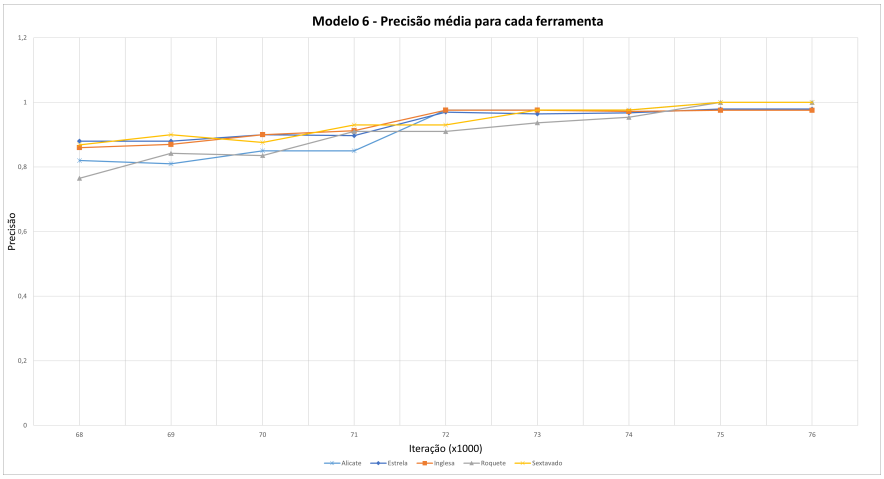


Figura 4.7: Resultados de precisão média para cada ferramenta ao longo do treino do modelo 6.

Os valores das precisões médias para cada ferramenta não começa em zero, uma vez que o modelo já foi treinado. O treino em imagens reais aparenta ter tido um impacto positivo, uma vez que a precisão de cada ferramenta aumentou significativamente. A média destas precisões atinge o pico no fim do treino, isto é, na iteração 76000 onde a precisão para o Alicate é de 1,0, para a Chave de Estrela é de 0,9795, para a Chave Inglesa é de 0,9756, para o Roquete é de 1,0 e para a Chave de Sextavado é de 1,0. Fazendo a análise com as métricas COCO, a precisão média com um índice de sobreposição de 50% é de 0,991, com 75% é de 0,9796 e com 50% até 95% é de 0,889, a taxa de sensibilidade é de 0,9125 e a pontuação F1 é de 0,9501.

Na Tabela 4.7 é visível que, todas as identificações foram feitas corretamente. Mais uma vez, o sistema *pick and place* foi capaz de se deslocar corretamente até à peça em todos os testes.

Neste capítulo foram apresentados os resultados para cada modelo treinado. Durante o processo de aprendizagem de cada rede, recorreu-se ao conjunto de dados de validação para identificar o estado da rede a cada 1000 iterações, com recurso às métricas PASCAL VOC [107]. No fim do treino, é identificada a iteração com melhor desempenho e é novamente avaliada no conjunto de dados de validação usando as métricas COCO [99], uma vez que estas fornecem informação complementar. Por fim, utiliza-se o modelo obtido na iteração da rede neuronal com melhor desempenho e integra-se no sistema *pick and place*, onde são feitos 20 testes por ferramenta e avaliando-se tanto o funcionamento da rede neuronal como o do sistema *pick and place*.

Tabela 4.7: Modelo 6 - Resultados obtidos na fase de testes (o valor apresentado para cada teste indica o índice de confiança na identificação).

Teste	Alicate	Estrela	Inglesa	Roquete	Sextavado
1	94	82	96	100	87
2	100	84	96	100	88
3	100	80	87	100	89
4	99	79	82	100	89
5	99	87	89	100	91
6	99	81	87	100	95
7	99	84	93	100	89
8	100	79	96	100	99
9	100	92	97	100	89
10	100	84	95	100	85
11	99	82	89	100	97
12	99	88	82	100	92
13	99	100	82	100	88
14	99	100	93	100	87
15	100	100	100	100	97
16	99	100	100	100	92
17	99	100	96	100	91
18	100	100	85	100	95
19	100	100	85	100	97
20	100	100	84	100	98

Capítulo 5

Discussão

O foco deste capítulo é a análise dos resultados obtidos, sendo feita uma comparação detalhada entre os modelos 1, 5 e 6 e entre os modelos 2, 3 e 4. Por fim, são abordados os resultados por classe.

5.1 Comparação dos modelos

5.1.1 Modelos 1, 5 e 6

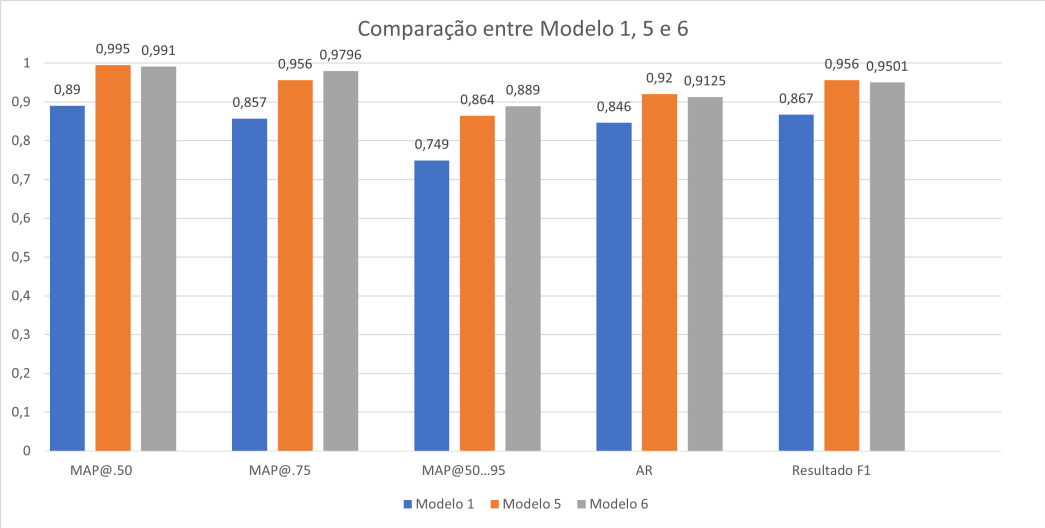


Figura 5.1: Avaliação dos modelos 1, 5 e 6.

A Figura 5.1 apresenta uma comparação dos resultados obtidos com o modelo 1, treinado exclusivamente com imagens sintéticas, com o modelo 5, treinado exclusivamente com imagens reais e com o modelo 6, que é a continuação do treino do modelo 1 mas desta vez com imagens reais, usando as métricas PASCAL VOC [107] e COCO [99].

Analisando a figura, verifica-se que o modelo treinado com imagens sintéticas (modelo 1) apresenta um desempenho inferior ao do modelo treinado com imagens puramente reais (modelo 5). Concretamente, para o $MAP_{@0.5}$ são 11,85% piores, para $MAP_{@0.75}$ são 11,55%, para o $MAP_{@0.5-0.95}$ são 15,35%, para a taxa de sensibilidade 8,75% e para a pontuação F1 10,27%. Analisando as Figuras 4.1 e 4.6, apresentadas no capítulo anterior, é visível que o modelo treinado com imagens exclusivamente reais não só necessitou de menos tempo de treino, como também convergiu de uma forma muito mais suave. Pode-se verificar também que existe uma queda progressiva no desempenho com o aumento da área de sobreposição, sendo esta queda mais acentuada no modelo 1. Relativamente aos resultados apresentados nas Tabelas 4.1 e 4.5, mais uma vez se verifica este comportamento, uma vez que o modelo 5 completou com 100% de sucesso todos os testes, enquanto que o modelo 1 apenas apresentou um comportamento correto em 95% das experiências e, no pior dos casos, apenas em 85% dos testes para o Alicate.

Os resultados destes testes indicam que a rede neuronal treinada com imagens geradas por computador não superou a rede treinada com imagens reais. Este resultado pode ser atribuído a uma série de fatores. Em primeiro lugar, as imagens reais tendem a ter mais variabilidade, complexidade e diversidade, o que permite que a rede aprenda uma gama mais ampla de recursos. Em comparação, as imagens geradas por computador podem ser limitadas na sua variabilidade e carecer das características únicas das imagens reais, levando ao sobreajuste e a uma menor capacidade de generalização. Em segundo lugar, a qualidade das imagens geradas por computador também pode afetar o desempenho. Se as imagens não forem geradas com detalhe suficiente ou contiverem artefactos, a rede pode não ser capaz de aprender características importantes. Além disso, as imagens reais geralmente contêm elementos complexos e desafiadores, como oclusões, mudanças de iluminação, iluminação não uniforme e ruído, que podem ajudar a rede a desenvolver uma representação robusta da tarefa. No geral, estes resultados sugerem que, para determinadas tarefas, imagens reais podem ser dados de treino superiores quando comparados com imagens geradas por computador, o que se alinha com os resultados de outras pesquisas [25, 62, 64].

Comparando o modelo 1 e o modelo 6, vemos que o segundo é melhor em todos os campos. Este comportamento é o expectável uma vez que o modelo 6 é a continuação do treino do modelo 1. No fim do treino do modelo 1, apresentado na Figura 4.1, continuou-se o treino durante 8000 iterações com o conjunto de dados reais, criando

desta forma o modelo 6. Apesar do treino ter sido breve, o modelo 6 melhorou, relativamente ao seu antecessor, o $MAP_{@0.50}$ em 11,35%, o $MAP_{@0.75}$ em 14,31%, o $MAP_{@0.50-0.95}$ em 18,69%, a taxa de sensibilidade em 7,86% e a pontuação F1 em 9,58%.

Por fim, analisando a Figura 5.1, a Tabela 4.5 e a Tabela 4.7 fica evidente que o desempenho do modelo 5 e o modelo 6 são bastante semelhantes. Contudo, este último apresenta um desempenho superior nas métricas $MAP_{@0.75}$ e $MAP_{@0.50-0.95}$ o que significa que as caixas delimitadoras geradas pelo modelo 6 são mais próximas do *Ground Truth* do que as geradas pelo modelo 5. Outra vantagem do modelo 6 é que, uma vez que os parâmetros internos, ou seja os pesos e *bias*, já estavam relativamente otimizados para a deteção de ferramentas pelo treino em imagens sintéticas, quando comparando com o número de iterações de treino do modelo 5, o modelo 6 necessitou de um treino significativamente menor e alcançou alguns resultados superiores ao modelo 5. Analisando as Figuras 4.6 e 4.7, verifica-se que o modelo 5 treinou durante 25000 iterações, ou seja aproximadamente seis épocas enquanto o modelo 6 necessitou apenas de duas. Este resultado vai de acordo com outras pesquisas [54, 60, 61], onde se afirma que conjuntos de dados que possuem a combinação de imagens reais e imagens sintéticas melhoram o desempenho do modelo treinado.

5.1.2 Modelos 2, 3 e 4

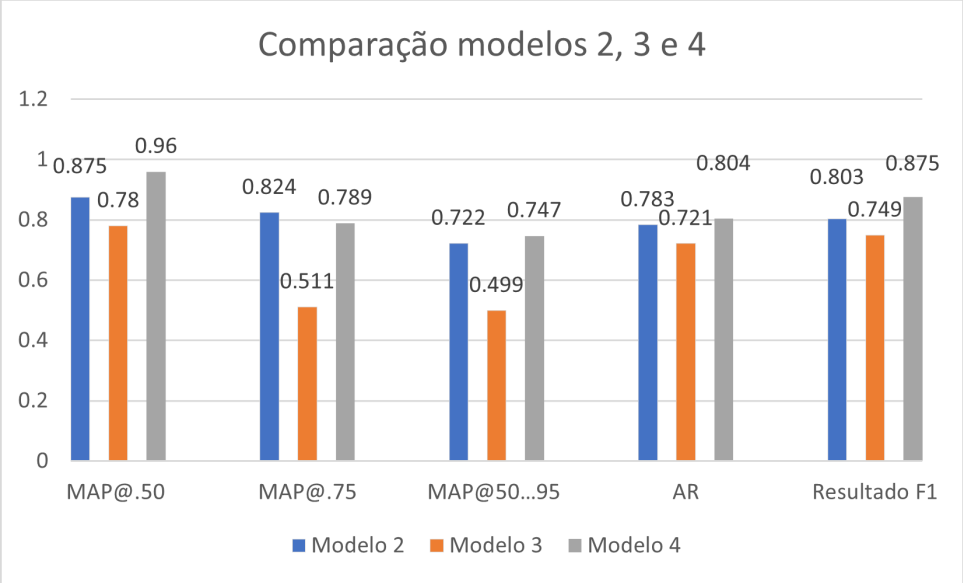


Figura 5.2: Avaliação dos modelos 2, 3 e 4.

Na Figura 5.2 comparam-se os resultados obtidos com o modelo 2, treinado com imagens sintéticas de ferramentas modeladas por outros, com o modelo 3, treinado a partir de imagens das ferramentas modeladas pelo autor e com o modelo 4, treinado

com uma combinação de imagens de ferramentas modeladas por outros e pelo autor. É evidente que o modelo 3 apresenta resultados significativamente inferiores aos restantes modelos. Em relação aos modelos 2 e 4 não é tão visível qual é o modelo superior. Enquanto o modelo 4 apresenta um desempenho superior em 9,71% em relação ao $MAP_{@0.50}$ do modelo 2, o mesmo não se verifica para o $MAP_{@0.75}$ onde o modelo 4 apresenta uma queda de 17,8% quando comparado com o $MAP_{@0.50}$ e um desempenho inferior em 4,4% quando comparado com o $MAP_{@0.75}$ do modelo 2. Na prática, isto significa que o modelo 4 apresenta um melhor desempenho na identificação da posição e do tipo de classe do objeto, contudo o modelo 2 consegue apresentar, por vezes, uma identificação ligeiramente mais ajustada.

Comparando os resultados dos testes, o mesmo comportamento é observável. O modelo 3 apresenta um comportamento significativamente inferior tendo gerado resultados corretos em apenas 88,33% dos testes, enquanto o modelo 2 gerou resultados corretos em 93,3% dos testes e o modelo 4 em 100% dos testes. Relativamente ao sistema *pick and place*, o braço conseguiu deslocar-se corretamente em todos os testes.

Em suma, estes resultados indicam que o modelo 4 apresenta um desempenho superior aos demais. Este comportamento é algo esperado dado que, no treino deste modelo, combinou-se a especialização dos objetos usados para o treino do modelo 3 com a generalização adquirida no modelo 2.

5.2 Análise por ferramenta

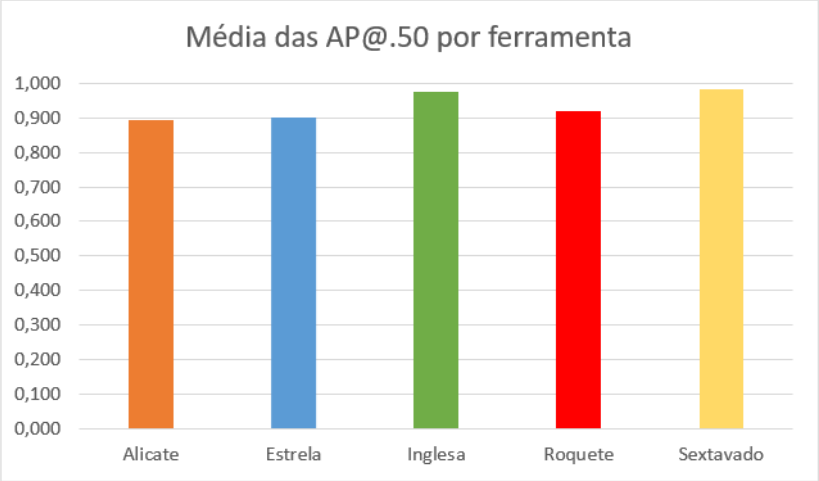


Figura 5.3: Avaliação do desempenho por ferramenta.

Na Figura 5.3 estão representadas as médias $AP_{@.50}$ por ferramenta de todos os modelos com o intuito de averiguar qual das ferramentas é de mais difícil identificação para os modelos. Como tal, é evidente que, no geral, a identificação da Chave Inglesa

e Chave de Sextavado é mais fácil, uma vez que as precisões médias de ambas em todos os modelos são as mais altas. Contudo, isto pode ser devido a estas ferramentas terem sido sujeitas a um menor número de testes quando comparado com as demais. Por outro lado, o Alicate é aquele que apresenta a precisão média mais baixa no geral.

Figura 5.4: Matriz de confusão.

Na Figura 5.4 está representada uma matriz de confusão de todos os testes efetuados. Uma matriz de confusão é usada para avaliar o desempenho de um modelo de classificação. Esta matriz mostra o número de verdadeiros positivos (valores das diagonais) e falsos positivos (restantes valores). Contudo, neste contexto, a matriz de confusão não vai ser usada para avaliar o desempenho de um modelo mas sim para perceber o comportamento das ferramentas em todos os testes feitos. É importante ressaltar que a matriz de confusão apresentada reflete os valores percentuais, representando a proporção de ocorrências relativas a cada classe em relação ao total de amostras avaliadas. Desta forma, os valores expressos na matriz de confusão estão em percentagem, permitindo uma análise mais precisa e comparativa das predições do modelo.

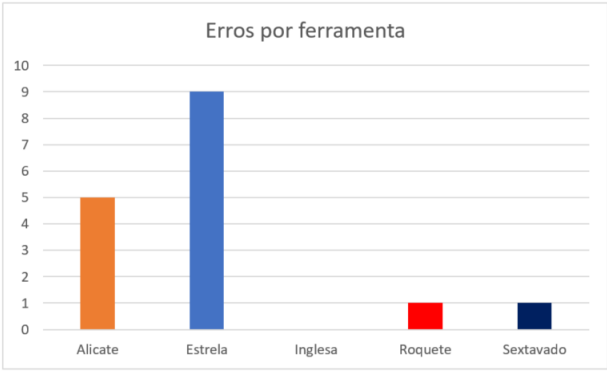


Figura 5.5: Falsas identificações por ferramenta.

Na Figura 5.5 estão as falsas identificações por ferramenta dos testes práticos feitos, extraídos da matriz de confusão apresentada na Figura 5.4. A informação extraída desta figura vai de encontro ao apresentado na Figura 5.3, isto é, o Alicate e a Chave de Estrela são mais difíceis de identificar enquanto a Chave Inglesa e a Chave de Sextavado são mais fáceis.

Por fim, o último estudo a ser feito é, no caso das identificações erradas, isto é, onde um objeto pertence a uma classe e é identificado como outra, quais são as classes que são identificadas. Na Figura 5.6 é apresentada a informação contida na matriz de confusão da Figura 5.4 mais sumarizada.

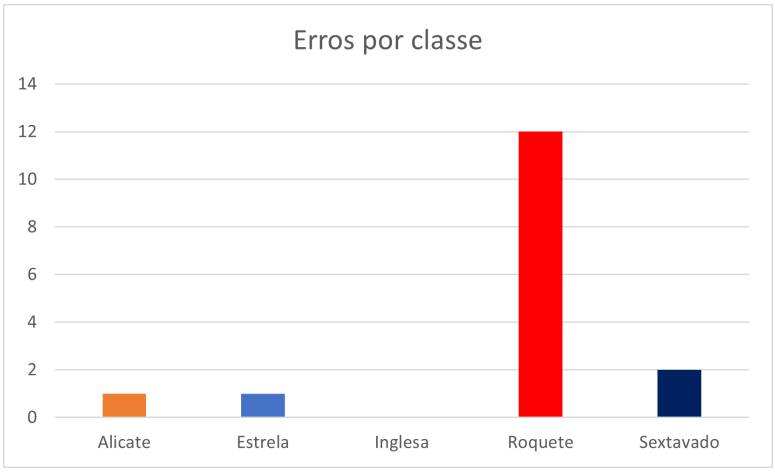


Figura 5.6: Classes identificadas nas falsas identificações.

Analisando a figura verifica-se que em 75% dos casos em que existe uma identificação errada, essa identificação é do tipo Roquete. Combinando este conhecimento, com o da Figura 5.5 e com o da Tabela 4.6 verifica-se que um modelo treinado com imagens reais tem dificuldade em identificar o modelo 3D do Roquete e da Chave de Estrela. Para além disso, segundo os testes práticos realizados, a Chave de Estrela é a ferramenta mais difícil de identificar quando os modelos são treinados com as imagens sintéticas criadas pelo autor e, quando existe uma falsa identificação, em 75% dos casos o objeto é identificado como Roquete. Apesar de ser difícil identificar a causa exata destes factos, uma possível explicação é a má modelação 3D destas ferramentas. Uma vez que uma rede treinada com imagens reais das ferramentas tem dificuldade a identificar uma réplica criada em simulação destes objetos, isto pode significar que existe um problema na modelação das mesmas. Como tal, o uso destes modelos para treino de redes neuronais pode causar entropia dentro da rede, uma vez que a mesma não vai conseguir captar características importantes do modelo podendo não só ter mais dificuldade a identificar as ferramentas (como no caso da Chave de Estrela) como também interferir na identificação das outras ferramentas (como no caso do Roquete). Contudo, mais testes práticos teriam de ser realizados para confirmar este fenómeno.

5.3 Automatização do processo de aquisição do conjunto de dados

Como referido no capítulo 1, no *workflow* proposto irá existir uma versão alfa do sistema que evoluirá para um sistema final. O sistema alfa consiste no sistema físico a trabalhar com uma rede neuronal treinada com dados sintéticos. Este sistema poderá ter limitações mas será funcional. Quando a máquina estiver a funcionar com os objetos reais, será adquirido o conjunto de dados dos objetos reais para o treino da rede neuronal final que, quando substituir a rede neuronal treinada com dados sintéticos, formará o sistema final.

O conjunto de dados real consiste em imagens dos objetos com as correspondentes anotações. As anotações são feitas manualmente e o processo de criação de uma anotação consiste na criação de um ficheiro que contém as coordenadas do objeto de interesse e a classe a que este pertence, sendo um processo lento e tedioso. Contudo, durante o funcionamento do sistema alfa, os objetos reais estarão a ser identificados pela rede neuronal treinada com dados sintéticos e a identificação consta na previsão da classe e das coordenadas do objeto de interesse, exatamente a mesma informação que o utilizador tem de inserir na anotação manual de uma imagem. Como tal, verifica-se aqui uma oportunidade de automatização da aquisição do conjunto de dados para o treino da rede do sistema final. Durante o funcionamento alfa do sistema, sempre que houver uma identificação de um objeto, o sistema vai continuar o funcionamento normal e, em paralelo, será guardada uma imagem e criada uma anotação dessa imagem com os dados de saída da rede neuronal treinada com dados sintéticos. Desta forma, posteriormente, é necessário apenas uma revisão manual onde o utilizador faz uma triagem entre imagens bem classificadas e imagens mal classificadas, sendo este processo significativamente mais rápido e simples do que o processo tradicional. Na Figura 5.7 estão, à esquerda, exemplos de uma anotação manual e à direita anotação autónoma.

Nesta figura é possível observar três situações diferentes, em (b) a classe atribuída é a correta e a caixa delimitadora é semelhante a caixa em (a), o que significa que o utilizador iria usar a anotação gerada automaticamente no conjunto de dados. Em (d) a classe atribuída é errada, contudo a caixa delimitadora está bastante semelhante a apresentada em (c). Nesta situação, o utilizador poderia fazer a alteração da classe para a correta ou simplesmente não utilizar esta amostra no conjunto de dados. Por fim, em (f) a classe é a correta contudo, a caixa delimitadora está significativamente maior do que a ideal. Existem três formas de enfrentar este problema: a primeira, seria o utilizador ajustar manualmente a caixa delimitadora; a segunda, seria o utilizador não usar esta amostra para o conjunto de dados; a terceira, e a ideal, seria usar técnicas de processamento de imagem para ajustar a dimensão da caixa delimitadora, tal como o processo abordado no capítulo 3.2.

(a) Anotação manual de uma imagem de um Alicate

(b) Anotação autónoma de uma imagem de um Alicate

(c) Anotação manual de uma imagem de uma Chave de Estrela

(d) Anotação autónoma de uma imagem de uma Chave de Estrela

(e) Anotação manual de uma imagem de uma Chave Inglesa

(f) Anotação autónoma de uma imagem de uma Chave Inglesa

Figura 5.7: Comparação entre imagens anotadas manualmente pelo autor e imagens anotadas autonomamente pelo modelo 1.

5.4 Redução do tempo de desenvolvimento do projeto

Uma das vantagens do *workflow* apresentado no capítulo 1 é possibilitar uma resposta mais rápida às necessidades do cliente. Como tal, três medidas permitem acelerar o processo desde a aquisição do conjunto de dados sintéticos até a rede neuronal final estar pronta.

A primeira é o uso da ferramenta CUDA [87] para a aceleração do processo de treino. Com recurso à computação paralela acelerada pela GPU uma iteração de treino do modelo pré-treinado SSD MobileNet V2 FPNLite 640x640 [96] com um *batch size* igual a quatro demora aproximadamente 0,503 segundos. Sem a aceleração pela GPU, uma iteração nas mesmas condições demora aproximadamente 3,14

segundos, ou seja, sem a aceleração, o processo de treino é 524,25% mais lento. Recorrendo ao modelo 1 como exemplo, este foi treinado durante 67000 mil iterações. Se a aceleração do processo de treino não fosse utilizada, o treino deste modelo demoraria aproximadamente 58 horas, ou seja dois dias e meio sensivelmente. Contudo, com a aceleração, este treino foi realizado em apenas nove horas.

A segunda é usar técnicas de *transfer learning* para acelerar o processo de treino. Uma vez que se recorreu a modelos pré-treinados, as camadas de extração de recursos já estavam treinadas e, como tal, não precisaram de ser ajustadas. Isto significa que apenas as camadas de previsão tenham de ser ajustadas, o que resulta em treinos menos extensivos. Para além disso, especificamente no modelo 6, utiliza-se *transfer learning* para melhorar um modelo treinado num conjunto de dados sintéticos (modelo 1) com recurso a imagens reais. Uma vez que o modelo 6 alcançou resultados idênticos ao modelo 5 treinado apenas com imagens reais, com menos 17000 iterações de treino, pode-se considerar que se otimizou ainda mais o processo.

Por fim, a última medida é a criação de um sistema de automatização da aquisição do conjunto de dados. O processo de anotação manual é lento e sujeito a erros devido a fadiga. Este sistema permite que os engenheiros do INEGI se foquem noutras tarefas durante a aquisição do conjunto de dados real. Quando este possuir um número de amostras suficiente, apenas é preciso rever o conjunto de dados manualmente. Esta revisão consta em eliminar as imagens mal anotadas e manter as anotadas corretamente.

Resumindo, após análise dos resultados obtidos com os diferentes modelos treinados, concluiu-se que, apesar dos modelos treinados com imagens geradas sinteticamente não possuírem o mesmo desempenho do modelo treinado com imagens reais, é possível garantir um certo nível de confiança nos resultados obtidos. Para além disso, especulou-se que as ferramentas Roquete e Chave de Estrela foram modeladas incorretamente o que afetou o desempenho dos modelos. De seguida, apresentou-se uma forma funcional de automatizar o processo de aquisição e anotação de conjunto de dados resultando numa redução significativa do tempo necessário para ter o sistema final funcional.

Capítulo 6

Conclusão

No presente estudo foi proposto um novo *workflow* para os projetos do INEGI que combinam visão computacional e aprendizagem profunda. Para validar este novo modelo de trabalho, aplicaram-se estes conceitos a um sistema *pick and place*, uma vez que a visão artificial desempenha um papel crucial na deteção e reconhecimento de objetos. Os métodos mais recentes, baseados em redes neuronais profundas, no geral, apresentam um bom desempenho, mas envolvem uma fase de treino que exige uma quantidade elevada de dados de treino previamente disponíveis e é, do ponto de vista computacional, muito exigente. Para ultrapassar esta limitação foi proposto o recurso a dados de treino gerados sinteticamente, que permite fazer o treino de uma versão preliminar do sistema, que poderá ser refinada quando os dados reais estiverem disponíveis, permitindo assim acelerar todo o processo desde a conceção do sistema até à sua implementação final.

Inicialmente, foram definidos os objetos de interesse para localização e reconhecimento, nomeadamente, ferramentas de oficina e prosseguiu-se com a modelação 3D das ferramentas escolhidas com recurso ao Solidworks [51]. Com as ferramentas já modeladas, passou-se para a geração do conjunto de dados, tendo sido criados cinco conjuntos de dados distintos, com propriedades únicas e com recurso a técnicas de *image augmentation*. De seguida, utilizou-se o modelo pré-treinado SSD MobileNet V2 FPNLite 640x640 [43] para treinar diversos modelos com base nos diferentes conjuntos de dados previamente criados. Estes modelos permitiram estimar a posição e classificar o tipo de objeto a ser movimentado pelo sistema *pick and place*.

Para a implementação física do sistema *pick and place*, foi utilizado um braço

mecânico KUKA LBR IIWA 14 R820 [103]. Abordaram-se os sistemas de comunicação industrial que permitiram fazer a ligação entre o momento em que a ferramenta é reconhecida e localizada no espaço, até ao momento em que o braço mecânico se desloca até a ferramenta.

Finalmente, testou-se o desempenho dos modelos de deteção e reconhecimento das ferramentas e o funcionamento do sistema *pick and place*. Comparou-se um modelo treinado com imagens puramente sintéticas, com um modelo treinado com imagens puramente reais, tendo-se concluído, como esperado, que o modelo treinado com imagens sintéticas é inferior em todos os aspetos, possuindo no melhor dos casos um mAP 11,55% pior e no pior dos casos 15,35% inferior. Enquanto que, com o modelo treinado com imagens reais, em todos os testes os objetos foram bem classificados, usando o modelo treinado com imagens sintéticas, verificou-se que, em 100 testes realizados, cinco resultaram em classificações erradas. Por outro lado, independentemente das classificações estarem corretas ou não, em todos os testes, o braço mecânico conseguiu deslocar-se com sucesso até ao objeto.

Também se concluiu que, no que toca a modelos treinados com dados gerados sinteticamente, o melhor desempenho é alcançado quando o conjunto de dados de treino possui a combinação de imagens especificamente criadas para o projeto e diversas imagens que sejam idênticas ao objeto de interesse mas com algumas particularidades.

6.1 Resultados Alcançados

Para a presente dissertação foram definidos três objetivos. O primeiro era a criação de um sistema *pick and place*. Uma vez que, só considerando os testes documentados, em 600 experiências, o sistema conseguiu deslocar-se corretamente até à ferramenta, sem falhas, pode-se concluir que este objetivo foi atingido.

O segundo objetivo consistia na criação de um modelo fiável treinado puramente com imagens sintéticas para ser usado enquanto o modelo treinado com imagens reais não está disponível. Um vez que, nos 400 testes feitos com os quatro modelos treinados puramente com dados sintéticos, em apenas 16 experiências é que os modelos não identificaram a ferramenta corretamente, também se pode concluir que este resultado foi alcançado.

Por fim, para diminuir o intervalo de tempo entre a criação do conjunto de dados sintético e o sistema final, foram tomadas três medidas. A primeira é o recurso às ferramentas CUDA [87] e cuDNN [88] que tornaram o processo de treino das redes neuronais quase sete vezes mais rápido. A segunda é a automatização do processo de criação e anotação do conjunto de dados real. E a última medida é a utilização de *transfer learning*. Uma vez que os modelos já estão pré-treinados, é necessário

um número significativamente menor de iterações de treino, conseguindo-se alcançar um desempenho superior.

6.2 Trabalho Futuro

Devido ao *hardware* usado durante o desenvolvimento deste projeto, foram impostas algumas limitações que afetaram o rendimento do treino das redes neuronais. No futuro, seria interessante treinar modelos com conjuntos de dados significativamente maiores. Para isso, poderiam ser aplicadas técnicas de aleatorização de domínio. Outra vantagem que o uso de um *hardware* superior proporcionaria, seria a capacidade de utilização de modelos pré-treinados computacionalmente mais exigentes e o uso de *batch sizes* maiores. Desta forma, apesar do treino teoricamente ser mais demorado por a rede ser mais complexa, poderia ser compensado com um *batch size* maior.

Uma alternativa a considerar também é a criação de uma rede neuronal de raiz, com o intuito de perceber se, para a identificação de objetos reais, tendo sido treinada com conjuntos de dados sintéticos, possui um desempenho melhor do que os modelos pré-treinados.

Algo interessante a estudar também seria a alteração do tipo de identificação dos modelos usados. Nas redes neuronais apresentadas, todas as identificações eram feitas por caixas delimitadoras, contudo, a identificação por máscara pode provar ser mais eficaz.

Uma vez que o Node-Red [105] foi implementado, numa versão futura deste projeto, a integração de uma base de dados está facilitada.

Por último, e relativamente ao sistema *pick and place*, seria necessário refazer os testes práticos, mas desta vez com um efetor final do tipo pinça equipado, com o intuito de perceber se a forma de estimação da posição da ferramenta é efetiva, ou se é preciso implementar melhores técnicas para o sistema conseguir recolher a peça. Numa futura versão deste projeto, a coordenada z, isto é, a profundidade da ferramenta, também poderá não permanecer constante e, como tal, será preciso aplicar técnicas de estimação de profundidade.

Referências

[1] R. S. Peres, X. Jia, J. Lee, K. Sun, A. W. Colombo, and J. Barata, "Industrial artificial intelligence in industry 4.0 - systematic review, challenges and outlook," *IEEE Access*, vol. 8, p. 220121–220139, 2020. [Citado nas páginas 1 e 2]

[2] M. Bertolini, D. Mezzogori, M. Neroni, and F. Zammori, "Machine learning for industrial applications: A comprehensive literature review," *Expert Systems with Applications*, vol. 175, p. 114820, 2021. [Citado na página 1]

[3] Y. Lu, H. Wang, and W. Wei, "Machine learning for synthetic data generation: A review," *arXiv:2302.04062v4*, 2023. [Citado na página 2]

[4] R. I. Mukhamediev, Y. Popova, Y. Kuchin, E. Zaitseva, A. Kalimoldayev, A. Symagulov, V. Levashenko, F. Abdoldina, V. Gopejenko, K. Yakunin, E. Muhamedijeva, and M. Yelis, "Review of artificial intelligence and machine learning technologies: Classification, restrictions, opportunities and challenges," *Mathematics*, vol. 10, no. 15, 2022. [Citado na página 7]

[5] "An introduction to deep learning." `https://developer.ibm.com/articles/an-introduction-to-deep-learning/`. "(Acedido em 02/05/2023)". [Citado nas páginas ix e 8]

[6] V. Wiley and T. Lucas, "Computer vision and image processing: A paper review," *International Journal of Artificial Intelligence Research*, vol. 2, p. 22, 02 2018. [Citado nas páginas 8 e 10]

[7] P. Gavali and J. S. Banu, "Deep convolutional neural network for image classification on cuda platform," *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, p. 99–122, 2019. [Citado na página 8]

[8] "Image processing, part 1: What is an image classifier & what to do with it." `https://levity.ai/blog/what-is-an-image-classifier`. "(Acedido em 17/06/2023)". [Citado nas páginas ix, 8 e 9]

[9] Z.-Q. Zhao, P. Zheng, S. tao Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, 2019. [Citado na página 8]

[10] R. Kaur and A. Doegar, "Brain tumor segmentation using deep learning: Taxonomy, survey and challenges," *Brain Tumor MRI Image Segmentation Using Deep Learning Techniques*, p. 225–238, 2022. [Citado na página 9]

[11] M. Thoma, "A survey of semantic segmentation," *arXiv:1602.06541*, 2016. [Citado na página 9]

[12] S. Malgonde, "Using convolutional neural networks for image segmentation - a quick intro." `https://blog.goodaudience.com/using-convolutional-neural-networks-for-image-segmentation-a\discretionary{-}{}{}quick-intro-75bd68779225`, February 2018. Acedido em 31/08/2022. [Citado nas páginas ix e 10]

[13] X. Liu, Z. Deng, and Y. Yang, "Recent progress in semantic image segmentation," *Artificial Intelligence Review*, vol. 52, no. 2, p. 1089–1106, 2018. [Citado na página 9]

[14] "Python opencv - estimativa de pose." `https://acervolima.com/python-opencv-estimativa-de-pose/`. "(Acedido em 24/05/2023)". [Citado nas páginas ix e 10]

[15] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, 2018. [Citado na página 11]

[16] "What is machine learning? definition, types, applications, and trends for 2022." `https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/`. "(Acedido em 04/05/2023)". [Citado nas páginas ix e 11]

[17] Q. Liu and Y. Wu, "Supervised learning," *Encyclopedia of the Sciences of Learning*, pp. 3243–3245, 01 2012. [Citado na página 11]

[18] S. Dridi, "A systematic review of unsupervised learning techniques for software defect prediction," *Information and Software Technology*, vol. 122, 12 2021. [Citado na página 11]

[19] Y. Ouali, C. Hudelot, and M. Tami, "An overview of deep semi-supervised learning," *arXiv:2006.05278v*, 2020. [Citado na página 11]

[20] Y. Li, "Deep reinforcement learning: An overview," *arXiv:1701.07274v6*, 2018. [Citado na página 11]

[21] J. Zou, Y. Han, and S.-S. So, "Overview of artificial neural networks," *Methods in molecular biology (Clifton, N.J.)*, vol. 458, pp. 14–22, 01 2009. [Citado na página 12]

[22] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, p. 92–108, 2022. [Citado nas páginas 12 e 13]

[23] P. Haddad, "How many neurons does a perceptron have?," *Stack Overflow*, May 1965. [Citado nas páginas ix e 13]

[24] "A gentle introduction to deep learning in r using keras." `https://lnalborczyk.github.io/slides/vendredi_quanti_2021/vendredi_quantis#12`. "(Acedido em 09/05/2023)". [Citado nas páginas ix e 14]

[25] C. S. Arcidiacono, "An empirical study on synthetic image generation techniques for object detectors," *KTH ROYAL INSTITUTE OF TECHNOLOGY SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE*, p. 1–91, Tese de Mestrado, 2018. [Citado nas páginas ix, x, 14, 18, 21, 27, 34, 36, 41, 42, 43, 44, 52 e 90]

[26] P. Geraldes, "In situ real-time zooplankton detection and classification," *Instituto Superior de Engenharia do Porto*, p. 1–108, Tese de Mestrado, 2021. [Citado nas páginas ix, x, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 31, 32, 33, 34 e 35]

[27] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, jan. [Citado na página 16]

[28] P. Guillou, "Qual É o principio de funcionamento de um algoritmo de inteligência artificial ?." Disponível em `https://medium.com/@pierre_guillou/qual-%C3%A9-o-principio-de-funcionamento-de-um-algoritmo-de-intelig%C3%AAncia-artificial-d68619ce2b4`, May 2018. (Acedido em 02/10/2022). [Citado nas páginas ix e 16]

[29] R. Aburasain, "Application of convolutional neural networks in object detection, reidentification and recognition," *Loughborough University*, p. 14–30, Tese de Mestrado, 2020. [Citado nas páginas 16, 17 e 26]

[30] M. Mano, "Understanding regularization in deep neural networks: a metalearning approach," *Faculdade de Engenharia do Porto*, p. 1–20, Tese de Mestrado, 2020. [Citado nas páginas 16, 17, 27 e 33]

[31] M. C. Vera, "Interpreting deep neural networks through backpropagation," *Instituto Superior Técnico da Universidade de Lisboa*, p. 1–35, Tese de Mestrado, 2019. [Citado nas páginas 18, 22 e 26]

[32] R. Kidambi and S. Kakade, *Stochastic gradient descent for Modern Machine Learning: Theory, algorithms and applications*. PhD thesis, University of Washington, 2019. [Citado nas páginas 19 e 26]

[33] A. P. O. fonso, "A comparative study of machine learning techniques for underwater visual object recognition," *Faculdade de Engenharia Da Universidade do Porto*, p. 1–59, Tese de Mestrado, 2019. [Citado nas páginas 20, 21 e 44]

[34] Hecht-Nielsen, "Theory of the backpropagation neural network," in *International 1989 Joint Conference on Neural Networks*, pp. 593–605 vol.1, 1989. [Citado nas páginas 24, 25 e 26]

[35] "What is a convolutional neural network?." Disponível em `https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html`. (Acedido em 31/08/2022). [Citado nas páginas x e 29]

[36] Y. Jeon and J. Kim, "Active convolution: Learning the shape of convolution for image classification," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1846–1854, 2017. [Citado na página 29]

[37] M. Basavarajaiah, "6 basic things to know about convolution." Disponível em `https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411`, Mar 2022. (Acedido em 31/08/2022). [Citado nas páginas x e 30]

[38] H. Wang, C. Ma, J. Zhang, and G. Carneiro, "Kernel adversarial learning for real-world image super-resolution," *arXiv:2104.09008v2*, 2022. [Citado na página 29]

[39] C. Kiourt, G. Pavlidis, and S. Markantonatou, "Deep learning approaches in food recognition," *Learning and Analytics in Intelligent Systems*, p. 83–108, Apr 2020. [Citado nas páginas x e 30]

[40] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv:1511.08458v2*, 2015. [Citado nas páginas 31 e 34]

[41] "What is a convolutional neural network (cnn) and how does it work?." `https://mriquestions.com/convolutional-network.html`. "(Acedido em 24/05/2023)". [Citado nas páginas x e 33]

[42] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. [Citado nas páginas 36 e 37]

[43] 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," *CoRR*, vol. abs/1704.04861, 2017. [Citado nas páginas 36, 38, 63, 64, 73 e 99]

[44] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv:1905.11946v5*, 2019. [Citado nas páginas x, 36, 37 e 38]

[45] N. AlDahoul, H. A. Karim, A. De Castro, and M. J. Tan, "Localization and classification of space objects using efficientdet detector for space situational awareness," *Scientific Reports*, vol. 12, no. 1, 2022. [Citado na página 37]

[46] P. Ganesh, "Types of convolution kernels: Simplified." Disponível em `https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37`, Oct 2019. (Acedido em 26/09/2022). [Citado nas páginas x e 38]

[47] I. Kandel, *Deep Learning Techniques for Medical Image Classification*. PhD thesis, Universidade Nova de Lisboa, May 2021. [Citado na página 39]

[48] E. Cengil, A. Cinar, and Z. Guler, "A gpu-based convolutional neural network approach for image classification," *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, 2017. [Citado nas páginas x e 40]

[49] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, p. 303–338, 2009. [Citado na página 41]

[50] "Blender." `https://www.blender.org/`. "(Acedido em 02/05/2023)". [Citado na página 45]

[51] "Soldiworks." `https://www.solidworks.com/`. "(Acedido em 05/05/2023)". [Citado nas páginas 45, 52 e 99]

[52] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. [Citado nas páginas 45 e 52]

[53] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtualworlds as proxy for multi-object tracking analysis," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. [Citado nas páginas 45 e 52]

[54] Y. Tian, X. Li, K. Wang, and F.-Y. Wang, "Training and testing object detectors with virtual images," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 2, p. 539–546, 2018. [Citado nas páginas 45, 52, 66 e 91]

[55] A. Shafaei, J. Little, and M. Schmidt, "Play and learn: Using video games to train computer vision models," *Procedings of the British Machine Vision Conference 2016*, 2016. [Citado na página 45]

[56] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, S. Birchfield, and et al., "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018. [Citado nas páginas 45 e 46]

[57] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. [Citado na página 45]

[58] H. Tjaden, U. Schwanecke, and E. Schomer, "Real-time monocular pose estimation of 3d objects using temporally consistent local color histograms," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017. [Citado na página 45]

[59] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. [Citado na página 45]

[60] A. Rozantsev, V. Lepetit, and P. Fua, "On rendering synthetic images for training an object detector," *Computer Vision and Image Understanding*, vol. 137, p. 24–37, 2015. [Citado nas páginas 45, 46, 66 e 91]

[61] M. Goyal, P. Rajpura, H. Bojinov, and R. Hegde, "Dataset augmentation with synthetic images improves semantic segmentation," *Communications in Computer and Information Science*, p. 348–359, 2018. [Citado nas páginas 45, 66 e 91]

[62] X. Peng, B. Sun, K. Ali, and K. Saenko, "Learning deep object detectors from 3d models," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. [Citado nas páginas 46 e 90]

[63] B. Sun and K. Saenko, "From virtual to reality: Fast adaptation of virtual object detectors to real domains," *Proceedings of the British Machine Vision Conference 2014*, 2014. [Citado na página 46]

[64] A. Jabbar, L. Farrawell, J. Fountain, and S. K. Chalup, "Training deep neural networks for detecting drinking glasses using synthetic images," *Neural Information Processing*, p. 354–363, 2017. [Citado nas páginas 46 e 90]

[65] X. Peng and K. Saenko, "Synthetic to real adaptation with generative correlation alignment networks," *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018. [Citado na página 46]

[66] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv:1712.04621v1*, 2017. [Citado na página 47]

[67] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," pp. 117–122, 05 2018. [Citado nas páginas 47 e 49]

[68] "Albumentations documentation - what is image augmentation." `https://albumentations.ai/docs/introduction/image_augmentation/`. "(Acedido em 06/03/2023)". [Citado nas páginas x, 47, 48 e 49]

[69] A. Galdran, A. Alvarez-Gila, M. I. Meyer, C. L. Saratxaga, T. Araújo, E. Garrote, G. Aresta, P. Costa, A. M. Mendonça, and A. Campilho, "Data-driven color augmentation techniques for deep skin image analysis," 2017. [Citado na página 49]

[70] A. Kwasigroch, A. Mikołajczyk, and M. Grochowski, "Deep convolutional neural networks as a decision support tool in medical problems – malignant melanoma case study," in *Trends in Advanced Intelligent Control, Optimization and Automation* (W. Mitkowski, J. Kacprzyk, K. Oprzędkiewicz, and P. Skruch, eds.), (Cham), pp. 848–856, Springer International Publishing, 2017. [Citado na página 49]

[71] A. Kwasigroch, A. Mikołajczyk, and M. Grochowski, "Deep neural networks approach to skin lesions classification — a comparative analysis," pp. 1069–1074, 08 2017. [Citado na página 49]

[72] M. Wasowicz, M. Grochowski, M. Kulka, A. Mikołajczyk, M. Ficek, K. Karpieńko, and M. Cićkiewicz, "Computed aided system for separation and classification of the abnormal erythrocytes in human blood," in *The Second International Conference "Biophotonics-Riga 2017*, 2017. [Citado na página 49]

[73] X. Peng, B. Usman, N. Kaushik, D. Wang, J. Hoffman, and K. Saenko, "Visda: A synthetic-to-real benchmark for visual domain adaptation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018. [Citado na página 52]

[74] Tzutalin, "Labelimg. git code." Disponível em `https://github.com/heartexlabs/labelImg`, 2015. [Citado na página 55]

[75] S. Z. Valtchev and J. Wu, "Domain randomization for neural network classification," *Journal of Big Data*, vol. 8, no. 1, 2021. [Citado na página 58]

[76] "Tensorflow." Disponível em `https://www.tensorflow.org/`. [Citado nas páginas 61, 62, 64, 73 e 75]

[77] "Brain team." https://research.google/teams/brain/. "(Acedido em 06/03/2023)". [Citado na página 61]

[78] "Pytorch." https://pytorch.org/. "(Acedido em 06/03/2023)". [Citado na página 61]

[79] "Keras." https://keras.io/. "(Acedido em 06/03/2023)". [Citado nas páginas 61 e 64]

[80] "Microsoft cognitive toolkit." https://learn.microsoft.com/en-us/cognitive-toolkit/. "(Acedido em 06/03/2023)". [Citado na página 61]

[81] "Theano." https://theano-pymc.readthedocs.io/en/latest/. "(Acedido em 06/03/2023)". [Citado na página 61]

[82] "Caffe." https://caffe.berkeleyvision.org/. "(Acedido em 06/03/2023)". [Citado na página 61]

[83] "Berkeley vision and learning center." https://dpd.berkeleyvision.org/. "(Acedido em 06/03/2023)". [Citado na página 61]

[84] "Opencv." https://opencv.org/. "(Acedido em 06/03/2023)". [Citado na página 61]

[85] H. Dai, X. Peng, X. Shi, L. He, Q. Xiong, and H. Jin, "Reveal training performance mystery between tensorflow and pytorch in the single gpu environment," *Science China Information Sciences*, vol. 65, no. 1, 2021. [Citado na página 62]

[86] M. C. Chirodea, O. C. Novac, C. M. Novac, N. Bizon, M. Oproescu, and C. E. Gordan, "Comparison of tensorflow and pytorch in convolutional neural network - based applications," in *2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pp. 1–6, 2021. [Citado na página 62]

[87] "Cuda." Disponível em https://developer.nvidia.com/cuda-zone. [Citado nas páginas 62, 96 e 100]

[88] "Cuda deep neural network." Disponível em https://developer.nvidia.com/cudnn. [Citado nas páginas 62 e 100]

[89] "Nvidia geforce gtx 1650 datasheet." https://www.techpowerup.com/gpu-specs/geforce-gtx-1650.c3366. "(Acedido em 06/03/2023)". [Citado na página 62]

[90] "Intel® core™ i5-10300h processor datasheet." Disponível em https://www.intel.com/content/www/us/en/products/sku/201839/

`intel-core-i510300h-processor-8m-cache-up-to-4-50-ghz/specifications.html`. [Citado na página 62]

[91] "tensorflow/models." Disponível em `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`. [Citado nas páginas xiii, 62 e 63]

[92] S. A. Sanchez, H. J. Romero, and A. D. Morales, "A review: Comparison of performance metrics of pretrained models for object detection using the tensorflow framework," *IOP Conference Series: Materials Science and Engineering*, vol. 844, p. 012024, 2020. [Citado na página 62]

[93] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. [Citado na página 62]

[94] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv:1905.11946v5*, 2019. [Citado na página 62]

[95] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. [Citado na página 62]

[96] Y.-C. Chiu, C.-Y. Tsai, M.-D. Ruan, G.-Y. Shen, and T.-T. Lee, "Mobilenet-ssdv2: An improved object detection model for embedded systems," pp. 1–5, 08 2020. [Citado nas páginas 63 e 96]

[97] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV 2016*, pp. 21–37, Springer International Publishing, 2016. [Citado nas páginas 63 e 64]

[98] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944, 2017. [Citado nas páginas 63 e 64]

[99] "Métricas coco." `https://cocodataset.org/#detection-eval`. "(Acedido em 17/03/2023)". [Citado nas páginas 63, 75, 86 e 90]

[100] "Tfrecord." `https://www.tensorflow.org/tutorials/load_data/tfrecord?hl=pt-br`. "(Acedido em 17/06/2023)". [Citado na página 63]

[101] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137–1149, 2017. [Citado na página 64]

[102] "Wc001a-4." https://www.orbitadigital.com/pt/cctv/ip/webcam/20079-wc001a-4-camara-web-webcam-resolucao-4mpx-angulo-de.html?srsltid=Ad5pg_Hqq16QMBrb_Qzr-691JMlLlssNMhvmMGeSrCxhy7-BTmfa_ru0QBk. "(Acedido em 09/03/2023)". [Citado na página 66]

[103] "Lbr iiwa 14 r820 datasheet." https://www.kuka.com/-/media/kuka-downloads/imported/8350ff3ca11642998dbdc81dcc2ed44c/0000246833_en.pdf?rev=59bb52b77ea24187a00e205042b0114a&hash=DE684100A66E9D16E4777318889B67E4. "(Acedido em 09/03/2023)". [Citado nas páginas 66 e 100]

[104] B. Krause, "What is opc ua? a practical introduction." Disponível em https://www.opc-router.com/what-is-opc-ua/, Aug 2022. (Acedido em 16/10/2022). [Citado na página 69]

[105] "Node-red." Disponível em https://nodered.org/. (Acedido em 16/10/2022). [Citado nas páginas 70 e 101]

[106] "Node-red." https://www.beckhoff.com/en-en/products/automation/twincat/. "(Acedido em 16/10/2022)". [Citado nas páginas 70 e 72]

[107] "Métricas pascal voc." http://host.robots.ox.ac.uk/pascal/VOC/. "(Acedido em 06/04/2023)". [Citado nas páginas 75, 86 e 90]