Universidade da Beira Interior
Departamento de Informática

Nº 6 - 2019: *Desenvolvimento de Sistemas Inteligentes para Registo Não-Cooperativo de Assiduidade em Ambientes de Sala de Aula*

Elaborado por:

Nuno Miguel da Silva Salvado, Nº 37575

Orientador:

Professor Doutor Hugo Proença

5 de Setembro de 2020

Agradecimentos

A conclusão deste trabalho não teria sido possível sem a imprescindível ajuda do meu orientador de projeto, o Professor Doutor Hugo Proença. A sua disponibilidade de tempo, os seus conselhos e ensinamentos foram cruciais para a realização deste projeto.

De seguida, quero manifestar o meu apreço pelo apoio e companheirismo demonstrado pelos meus colegas de curso, amigos próximos, elementos do grupo *Soft Computing and Image Analysis Laboratory* (SOCIALAB).

Quero também agradecer aos meus colegas de curso e amigos João Brito e Bruno Degardin por toda a ajuda que me disponibilizaram, quer na minha integração no grupo do SOCIALAB quer em dificuldades que eu fui encontrando no decorrer do trabalho.

Gostaria, ainda, de agradecer a todos os meus amigos e colegas que dispensaram um pouco do seu tempo para se deslocarem à universidade para me ajudarem numa fase crucial do meu projeto, a captura de vídeos dentro de uma sala do Departamento de Informática da Universidade da Beira Interior.

Por último, mas não menos importante, gostaria de agradecer à minha família e à minha namorada pelo apoio incondicional durante o meu percurso académico, e em especial na realização e conclusão deste trabalho.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

API	*Application Programming Interface*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
CVAT	*Computer Vision Annotation Tool*
DL	*Deep Learning*
fps	*frames per second*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
IoU	*Intersection over Union*
OpenCV	*Open Source Computer Vision Library*
PIL	*Python Imaging Library*
R-CNN	*Region based-Convolutional Neural Network*
ReLu	*Rectified Linear Unit*
ResNet	*Residual Neural Network*
RGB	Red, Green, Blue
RNA	Redes Neuronais Artificiais
RPN	*Region Prediction Network*
SOCIALAB	*Soft Computing and Image Analysis Laboratory*
SSD	*Single Shot MultiBox Detector*
YOLO	*You Only Look Once*

Capítulo

1

Introdução

1.1 Enquadramento

O presente documento, associado ao trabalho intitulado **Desenvolvimento de Sistemas Inteligentes para Registo Não-Cooperativo de Assiduidade em Ambientes de Sala de Aula**, foi realizado no âmbito da Unidade Curricular de Projeto que se enquadra no terceiro ano da Licenciatura em Engenharia Informática na Universidade da Beira Interior.

1.2 Motivação

Hoje em dia é cada vez mais importante encontrar tecnologias que nos permitam perder/dispender menos tempo em atividades que podem ser facilmente executadas por máquinas e poder concentrar a nossa atenção e aplicar o tempo noutras atividades que ainda não podem ser realizadas por elas. Desta forma é de extrema importância a constante evolução que a Inteligência Artificial (IA) e o *Machine Learning* têm apresentado ao longo dos últimos anos. O registo de presenças através de reconhecimento facial é uma das atividades em que a IA pode ser aplicada. Para isso ser possível é necessário o desenvolvimento de tecnologias que realizem a deteção e *tracking* de faces e/ou silhuetas humanas e a respetiva identificação biométrica. Além da questão da poupança de tempo, de recursos humanos e recursos materiais, esta forma de registo de presenças causa menos distúrbios no decorrer da aula e impede a fraude ao nível da presença nas aulas.

1.3 Objetivos

Este projeto visa desenvolver uma solução de registo de assiduidade, quer de alunos quer de docentes, dentro de uma sala de aula, sem intervenção ativa dos mesmos, sendo, apenas, necessária a sua presença na sala de aula. Para esse fim, é necessário completar as seguintes fases:

1. Deteção e *tracking* de faces/silhuetas humanas.
2. Identificação biométrica em ambientes não-cooperativos ou de video-vigilância.

No âmbito deste projeto apenas é abordada a primeira fase.

1.4 Organização do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. No capítulo 1 – **Introdução** – é apresentada uma descrição geral do projeto, com os respetivos enquadramento e objetivos, bem como a motivação para a realização do mesmo. Apresenta-se, ainda, a organização do presente documento.
2. No capítulo 2 – **Técnicas de Aprendizagem Profunda** – faz uma introdução à Visão Computacional, IA e *Machine Learning*. Adicionalmente, descreve as bases teóricas das Redes Neuronais Artificiais (RNA) e das *Convolutional Neural Network* (CNN). É, ainda, descrita a forma como é feito o reconhecimento de objetos e de faces humanas.
3. No capítulo 3 – **Abordagem Proposta** – apresenta uma descrição das tecnologias utilizadas neste projeto, a seleção da câmara e lente utilizadas e os dados que foram gerados da configuração (câmara + lente).
4. No capítulo 4 – **Experiências e Resultados** – são abordados a primeira fase deste projeto, a forma como foi concretizada e os resultados obtidos.
5. No capítulo 5 – **Conclusões e Trabalho Futuro** – está incluída uma revisão do documento, uma análise do projeto realizado, comparando os objetivos iniciais com os finais, e ainda uma discussão de possíveis propostas de trabalhos futuros ao nível do desenvolvimento deste projeto.

Capítulo

2

Técnicas de Aprendizagem Profunda

2.1 Introdução

A IA aplicada à deteção e ao reconhecimento de objetos e faces humanas já existe há décadas mas tem vindo a ser aperfeiçoada e atualmente está facilmente acessível nomeadamente ao nível das aplicações pessoais de fotografia e autenticação secundária para dispositivos móveis. Assim, hoje em dia, já é possível recorrer ao enorme conjunto de imagens disponíveis digitalmente para a identificação de criminosos ou pessoas desaparecidas. Tudo isto só se torna possível graças à utilização de algoritmos desenvolvidos a partir de IA e treinados através da tecnologia de *Deep Learning* (DL) baseada na utilização de RNA.

Assim, ao longo do presente capítulo serão explicitados alguns conceitos essenciais para a deteção e reconhecimento facial e a forma como estes são conseguidos.

2.2 Visão Computacional

A Visão Computacional é um ramo da IA que permite que os computadores interpretem e entendam o mundo visual através de imagens digitais, vídeo ou outras entradas visuais. Pode dizer-se que a IA permite que os computadores "pensem" e a Visão Computacional permite que os computadores "vejam e entendam o que estão a ver". A Visão Computacional treina as máquinas para desempenhar funções semelhantes às da visão humana através da utilização

de câmaras, dados e algoritmos, em vez de retinas, nervos óticos e córtex visual [1] [2] [3].

Na década de 50 do século passado foram realizadas as primeiras experiências no campo da Visão Computacional. Nestas experiências foram utilizadas algumas das primeiras redes neuronais para detetar os contornos de um objeto e colocar objetos em categorias como círculos ou quadrados. Mais tarde, na década de 70, a Visão Computacional foi aplicada na interpretação de texto digitado ou manuscrito através do reconhecimento ótico de caracteres, o que permitiu a interpretação de texto escrito para cegos. Na década de 80, o neurocientista britânico David Marr introduziu algoritmos que permitiram que as máquinas detetassem arestas, cantos, curvas e formas básicas semelhantes. Quase ao mesmo tempo, Kunihiko Fukushima, um cientista japonês, desenvolveu uma rede de células que permitia o reconhecimento de padrões. A evolução da internet a partir dos anos 90, disponibilizando inúmeros conjuntos de imagens *on-line* para análise, levou, em 2001, ao desenvolvimento dos primeiros modelos de reconhecimento facial em tempo real [1] [2] [3].

Atualmente, a Visão Computacional tem vindo a ser usada em várias áreas, nomeadamente a deteção e reconhecimento facial e de objetos, análise de imagens médicas, processamento da ação ao vivo de um jogo de futebol, entre outras utilizações. Para o desenvolvimento destas aplicações é necessário apresentar uma quantidade elevada de dados ao computador para que este identifique padrões. Para isso são necessárias duas tecnologias essenciais: o DL e as CNN [4].

2.3 Inteligência Artificial

A IA, cuja descoberta remonta aos anos 50 do século passado, é uma disciplina da Engenharia da Computação que se refere a uma tecnologia que tem como objetivo a simulação da inteligência humana em máquinas programadas para pensar e imitar ações características dos seres humanos, como a aprendizagem e resolução de problemas. Pode, também, ser definida como a capacidade do sistema interpretar corretamente dados externos, aprender com eles e usá-los para alcançar objetivos e tarefas específicos por meio de adaptação flexível. A IA engloba aprendizagem, raciocínio e percepção, ou seja, permite que as máquinas aprendam com a experiência e que, através da utilização de ferramentas como DL, *Machine Learning* e processamento de linguagem, realizem tarefas específicas, processando grandes quantidades de dados e reconhecendo padrões nestes [5] [6] [7] [8].

2.3.1 *Machine Learning*

Machine Learning é um ramo da IA que se baseia na ideia de que os sistemas podem aprender com dados, identificar padrões e tomar decisões com o mínimo de intervenção humana, ou seja, permite que as aplicações de *software* sejam bastante precisas na previsão de resultados, mesmo sem serem programadas para tal. Este tipo de "aprendizagem" consiste na execução de algoritmos que criam automaticamente modelos de representação de conhecimento com base num conjunto de dados. As bases do *Machine Learning* indicam que devemos treinar as máquinas, dando-lhes acesso a dados históricos e deixar que o algoritmo "aprenda", melhorando o seu desempenho. Após o treino, o modelo consegue realizar previsões em situações futuras e que estejam relacionadas com padrões históricos [9] [10].

O *Machine Learning* engloba vários algoritmos de aprendizagem, os quais variam de acordo com a sua abordagem, o tipo de dados que utilizam e produzem e o tipo de tarefa ou problema que pretendem resolver. Estes algoritmos classificam-se como:

- **Aprendizagem Supervisionada** - "É o termo usado sempre que o programa é "treinado" com um conjunto de dados pré-definido" [11]. Com base no treino com estes dados, "o programa pode tomar decisões precisas quando recebe novos dados" [11]. Designa-se supervisionado porque o processo de aprendizagem é controlado. De acordo com o tipo de resultado do algoritmo, podemos classificá-lo entre algoritmo de classificação ou algoritmo de regressão. Os algoritmos de classificação são usados quando as saídas se cingem a um conjunto limitado de valores, enquanto os algoritmos de regressão são utilizados quando as saídas podem ter um valor numérico qualquer dentro de um intervalo [12] [13] [14].

- **Aprendizagem Não-Supervisionada** - "Termo usado quando um programa pode, automaticamente, encontrar padrões e relações num conjunto de dados" [14]. Exemplo: Analisar um conjunto de *e-mails* e agrupá-los automaticamente de acordo com o tema, sem que o programa possua qualquer conhecimento prévio sobre os dados [12] [13] [14].

- **Aprendizagem por Reforço** - Termo usado quando se treina um programa para se comportar de uma certa forma num determinado ambiente, recorrendo à atribuição de recompensas sempre que atua da forma pretendida [10] [12].

Resumindo, trata-se de um método de análise de dados muito utilizado atualmente apesar de quase não nos apercebermos disso por nos ser muito familiar. É utilizado em situações tão banais como anúncios em tempo real nos *websites*, filtragem de *spam* no *e-mail* ou traduções automáticas do Google.

2.3.2 *Deep Learning*

DL é um subconjunto de *Machine Learning* baseado do princípio das RNA com várias camadas, semelhante ao cérebro humano. O DL utiliza um elevado número de dados como entrada e toma decisões intuitivas e inteligentes, através do reconhecimento de padrões e da utilização de muitas camadas de processamento. É de salientar que, quanto mais dados, melhor o desempenho [12] [15].

As redes neuronais surgiram nos anos 50 do século passado, mas só recentemente o poder computacional e as capacidades de armazenamento de dados evoluíram o suficiente para que o DL pudesse ser usado para criar novas tecnologias. Atualmente é utilizado em várias áreas, nomeadamente no reconhecimento da fala (ex: *Siri* da Apple), reconhecimento de imagens (ex: carros autónomos) e processamento de linguagem natural [12].

2.4 Deteção e Reconhecimento de Objetos

A deteção de objetos é um dos problemas clássicos da Visão Computacional e é, também, um passo essencial no processo de reconhecimento de objetos, sobretudo se houver vários objetos numa mesma imagem e/ou vídeo. As técnicas de deteção de objetos conseguem localizá-los numa imagem, mostrando as coordenadas da *bounding box* correspondente a cada um dos objetos da imagem. De todas as técnicas possíveis para realizar a deteção de um objeto, uma das mais comuns é apresentá-la como um problema de classificação, sendo que, nesta técnica são usadas janelas móveis que são colocadas sobre certas representações inseridas numa imagem para classificar determinadas regiões.

A classificação de objetos engloba a deteção e o reconhecimento dos mesmos, sendo que estes são modelados de forma a que:

- Cada tipo de objeto seja uma classe.
- Uma imagem pode conter uma ou várias classes.

- Exista um conjunto finito de tipos de objetos.

Um classificador que seja complexo e avançado, tal como uma CNN, é necessário para obter um alto rigor nas deteções, mas, ao mesmo tempo, diminui-se consideravelmente a velocidade da avaliação, devido à elevada necessidade de recursos computacionais que este tipo de modelos acarreta. Por outro lado, métodos para detetar e reconhecer objetos que requerem menos recursos computacionais são importantes para conseguir a deteção rápida de objetos em dispositivos de baixo desempenho, consumo e custo [16].

2.4.1 Reconhecimento Facial

O reconhecimento facial é um sistema desenvolvido para identificar uma ou mais pessoas por meio de uma imagem ou vídeo. É uma tecnologia que já existe há bastante tempo mas o seu uso tornou-se mais acessível e disseminado nos últimos anos, devido ao facto de, atualmente, ser usada em soluções inovadoras (por exemplo, autenticação secundária para dispositivos móveis e aplicações pessoais de fotografia) [17].

Para a identificação de características faciais exclusivas, o *software* de reconhecimento facial atravessa várias etapas:

- **Deteção de rosto** - o sistema identifica na imagem e/ou vídeo a parte que representa a face.

- **Pré-processamento (normalização de recursos)** - os dados são transformados num formato monolítico normalizado, ou seja, as imagens têm que possuir a mesma resolução, brilho, orientação e níveis de *zoom*.

- **Extração de recursos (codificação)** - o sistema tem que extrair dados das imagens faciais fazendo uma pequena seleção, onde escolhe os *bits* de dados mais relevantes e ignora tudo o que for "ruído".

- **Reconhecimento facial** - o sistema faz corresponder recursos de dados exclusivos a cada indivíduo [18].

Ao longo destas etapas podem surgir outros problemas relativamente ao tipo de dados usados. Esses problemas adaptam o fluxo de trabalho para reconhecimento facial e para as várias etapas de deteção/pré-processamento de faces que estejam envolvidas. Alguns desses problemas são os seguintes: iluminação, posição, idade, oclusão e resolução. Para resolver estes problemas

podem ser usadas técnicas e algoritmos de normalização de dados e de melhoramento de recursos de reconhecimento. Existem vários algoritmos de reconhecimento facial, mas a sua eficiência depende do tipo de dados/imagens [18].

2.5 Redes Neuronais Artificiais

RNA são sistemas de computação inspirados no cérebro humano. Foram desenvolvidas na década de 50 do século passado como tentativa de simular a rede de neurónios que compõem o cérebro humano e fazer com que o computador aprenda e resolva problemas de forma semelhante a um ser humano [19] [20] [21].

As RNA são compostas por diferentes camadas de neurónios artificiais, designados unidades, interconectados entre si, tais como os verdadeiros neurónios. Atualmente as RNA podem ter desde milhares a milhões de unidades e milhões de conexões. Cada neurónio só se pode conectar com neurónios da camada imediatamente anterior e neurónios da camada imediatamente seguinte, não se conectando com neurónios da mesma camada (figura 2.1). Os sinais (informação do mundo exterior) são recebidos pela primeira camada de neurónios artificiais e viajam através de várias camadas intermédias (camadas ocultas) até à última camada (camada de saída) [19] [20] [21].

As RNA são muito versáteis, recebendo dados na forma de texto, imagem, vídeo ou som, e podem ser usadas em várias tarefas, entre as quais, reconhecimento da fala, reconhecimento facial, tradução automática, videojogos, bioinformática e diagnóstico médico [20].

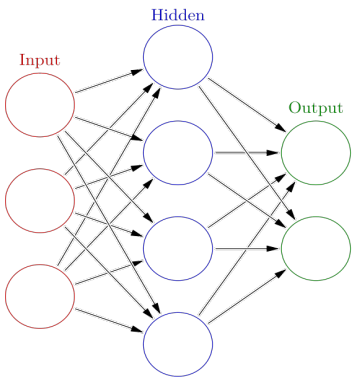

Figura 2.1: Arquitetura de uma RNA.

2.5.1 SSD

O *Single Shot MultiBox Detector* (SSD) é um algoritmo baseado numa CNN *feed-forward* muito usado na deteção de objetos [22]. Tem como vantagem o facto de ser mais rápido que o Faster R-CNN pois apenas necessita de tirar uma fotografia para detetar vários objetos na imagem, enquanto o Faster R-CNN precisa de duas e é, também, mais preciso que o *You Only Look Once* (YOLO) [22] [23] [24]. Por ser mais rápida que as redes *Region based-Convolutional Neural Network* (R-CNN) permite aplicação em vídeo em tempo real com 20 a 25 *frames* por segundo.

O SSD é constituído por um modelo *backbone* (caixas brancas na figura 2.2) e uma SSD *head* (caixas azuis na figura 2.2). O primeiro é uma rede de classificação de imagens treinada como um extrator de recursos, como a *Residual Neural Network* (ResNet), à qual foi retirada a camada final de classificação, ou seja, a camada completamente ligada. A SSD *head* corresponde a uma ou mais camadas convolucionais anexadas ao *backbone*, sendo as saídas interpretadas como *bounding boxes* e classes de objetos [25].

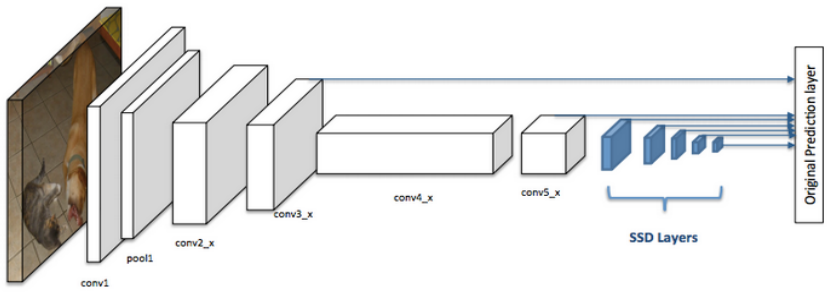

Figura 2.2: Arquitetura de uma CNN com um detetor SSD.

Para perceber melhor o funcionamento do SSD é necessário esclarecer alguns conceitos, como *Gride cell* e *Prior boxes*:

1. ***Gride cell*** **-** No caso do SSD, a imagem de entrada é dividida usando uma grelha, sendo cada célula dessa grelha (*gride cell*) responsável por detetar (classificar e localizar) os objetos nessa região da imagem [25].

2. ***Prior boxes*** **(ou** ***Anchor boxes*****) -** Caso existam vários objetos na mesma *gride cell* ou se se pretender detetar vários objetos de formas diferentes é necessário recorrer a *prior boxes*, sendo que cada *gride cell* pode conter várias *prior boxes* (figura 2.3). As *prior boxes* são predefinidas correspondendo cada uma a um tamanho e forma dentro de uma *gride*

cell. A *prior box* que melhor se ajustar a um objeto (ou que faça uma sobreposição deste com grau mais elevado) tem como função prever a classe e localização desse mesmo objeto [25].

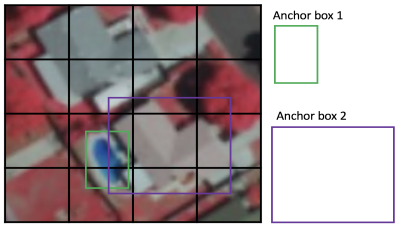

Figura 2.3: Exemplo de *prior boxes* ou *anchor boxes*.

2.5.2 YOLO

O YOLO é um algoritmo de deteção de objetos que funciona de forma semelhante ao SSD e, tal como este, apenas necessita de *"one-shot"* [26].

O YOLO deteta objetos através da aplicação de uma CNN à imagem completa, dividindo-a, em seguida, em regiões. Posteriormente prevê *bounding boxes* e probabilidades para cada região. Trata-se de um algoritmo muito usado porque é muito preciso e pode ser usado em tempo real. Graças a uma função *non-max supression* é garantido que o algoritmo deteta cada objeto apenas uma vez [27]. A arquitetura do YOLO é inspirada no GoogLeNet, algoritmo usado para classificação de imagens, e é constituída por 24 camadas convolucionais seguida de 2 camadas completamente ligadas (figura 2.4) [28].

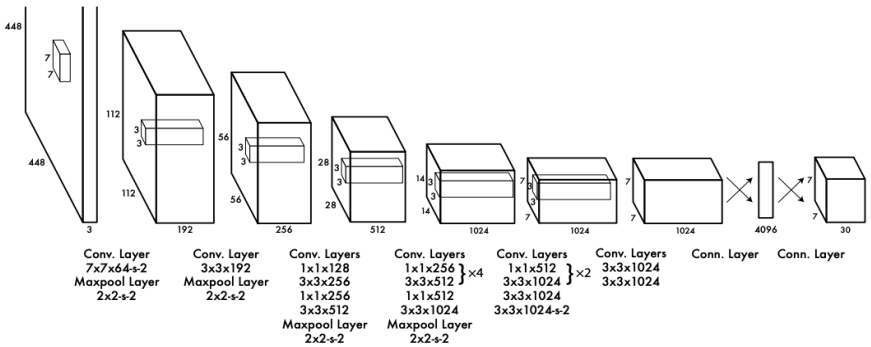

Figura 2.4: Arquitetura do YOLO.

No YOLO, a saída é um mapa de recursos e, como são utilizadas convoluções 1x1, o tamanho do mapa de previsão é igual ao tamanho do mapa de recursos anterior. Quanto à profundidade, as entradas no mapa de recursos são do tipo (Bx(5+C)), onde B representa o número de *bounding boxes* que cada célula prevê. Cada *bounding box* pode especializar-se na deteção de determinado objeto e possui atributos 5+C, os quais descrevem as coordenadas do centro, as dimensões, a pontuação do objeto e o nível de confiança C. Ao utilizar o YOLO espera-se que cada célula do mapa de recursos preveja um objeto utilizando uma das suas *bounding boxes*, o que acontece se o centro do objeto se localizar no campo recetivo dessa célula. Entenda-se como campo recetivo a região da imagem de entrada visível para a célula. Na figura 2.5 é a célula a vermelho que é responsável pela deteção do objeto, neste caso, o cão [29].

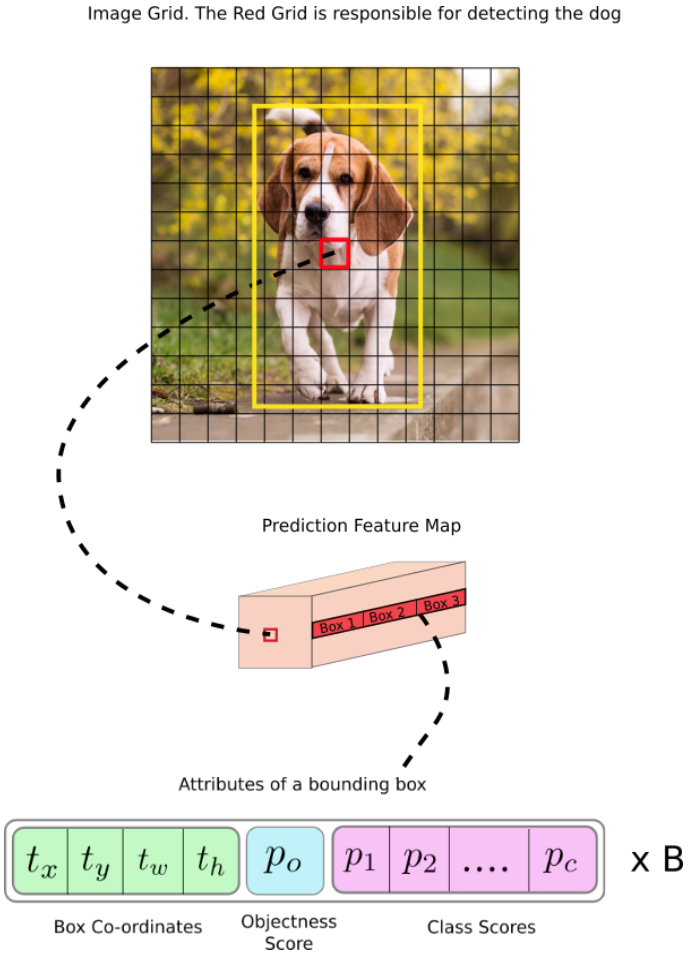

Figura 2.5: Representação da célula responsável pela deteção do objeto.

Existem mais duas versões do YOLO, o YOLOv2 e o YOLOv3. O YOLOv2, lançado em 2016, é capaz de detetar mais de 9000 classes de objetos, daí também poder ser designado YOLO9000. Este consegue prever deteções para classes de objetos sem dados de deteção rotulados. Embora consiga detetar 9000 classes de objetos, o YOLOv2 tem a desvantagem de ser pouco preciso [26] [27]. O YOLOv2 é constituído por 30 camadas, 19 camadas pertencentes à Darknet-19 e 11 camadas para deteção de objetos, mas tinha problemas na deteção de pequenos objetos. Em abril de 2018 foi lançado o YOLOv3 que tem como vantagens o facto de ser mais rápido e melhor na deteção de objetos pequenos [27] [29]. O YOLOv3 usa a Darknet-53, constituída por 53 camadas, e adiciona-lhe mais 53 camadas para deteção, formando uma arquitetura totalmente convolucional de 106 camadas (figura 2.6). Além disso, outra característica importante do YOLOv3 é a capacidade de fazer deteções em três escalas diferentes. Comparativamente ao YOLOv2, o YOLOv3 é mais lento pois prevê mais *bounding boxes* para uma imagem de entrada do mesmo tamanho [30].

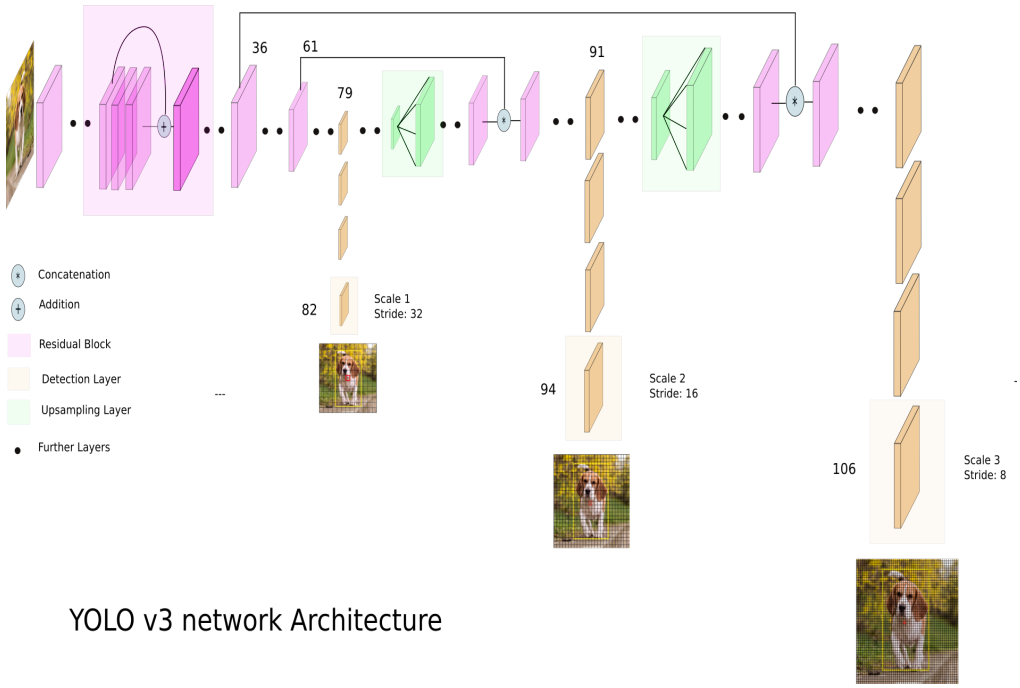

Figura 2.6: Arquitetura do YOLOv3.

2.5.3 Faster R-CNN

O Faster R-CNN, lançado em 2015, é a versão mais avançada da família R-CNN. Esta é uma família de algoritmos de deteção e classificação de objetos numa

imagem. No geral, as redes R-CNN fazem a deteção de objetos através de *bounding boxes*. De seguida são utilizadas uma CNN para extrair recursos desses objetos e uma camada de classificação para prever a que classe pertence o objeto. Por fim, é aplicada uma regressão para determinar, de forma mais precisa, as coordenadas da *bounding box* do objeto [31] [32] [33] [34].

Um dos problemas do R-CNN e do Fast R-CNN é o facto do seu algoritmo de pesquisa seletiva propôr muitas regiões possíveis e de exigir a classificação de todas elas, tornando este processo muito lento, o que impede a sua utilização em processamento de imagem em tempo real. O Faster R-CNN veio resolver este problema através da eliminação completa da pesquisa seletiva e da utilização de *Region Prediction Network* (RPN). O Faster R-CNN é constituído por dois módulos: o primeiro é o RPN e o segundo é um detetor Fast R-CNN. Assim, a imagem de origem é inserida na RPN, a qual considera um número relativamente grande de regiões possíveis e, de seguida, prevê quais apresentam maior probabilidade de ter objetos de interesse. Estas regiões previstas são submetidas a uma camada de *pooling* de *Region of Interest* que é usada para classificar as imagens de cada região e prever os valores de deslocamento das *bounding boxes* (figura 2.7) [31] [33] [34].

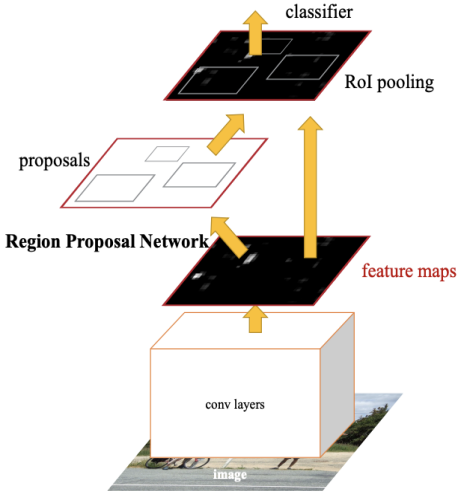

Figura 2.7: Arquitetura do Faster R-CNN.

As âncoras ou *bounding boxes* do Faster R-CNN podem ser ajustadas de acordo com o problema em estudo, ou seja, se pretendermos identificar pessoas num vídeo de uma câmara de vigilância, ajustamos a âncora a tamanhos menores. As âncoras servem de entrada para a RPN e esta classifica-as, com a ajuda de

uma função de ativação de regressão logística como a *Softmax*, em primeiro plano ou plano de fundo. As âncoras rotuladas como primeiro plano passam para a etapa seguinte do algoritmo, ou seja, a camada de *pooling* de *Region of Interest*. Nesta etapa, as regiões, de diferentes tamanhos, propostas pela RPN são reduzidas a mapas de recursos todos com o mesmo tamanho. O mapa de recursos de cada proposta de região passa por duas camadas completamente ligadas e pela função de ativação *Rectified Linear Unit* (ReLu) para gerar uma previsão para cada um dos objetos [33].

2.5.4 CNN

Uma CNN, também designada ConvNet, é um tipo de RNA cuja construção é inspirada na organização do córtex visual dos animais e que tem vindo a ser muito utilizada no processamento e análise de imagens digitais [35]. A origem da CNN remonta à década de 70 do século passado, mas a primeira arquitetura só foi proposta em 1998 por Yan LeCun, a designada LeNet [36].

A CNN é um algoritmo de DL que capta uma imagem de entrada, atribui importâncias (pesos e vieses) a aspetos e/ou objetos da imagem, conseguindo diferenciá-las umas das outras. A ideia principal do seu desenvolvimento é que a CNN consiga "filtrar linhas, curvas e bordas e em cada camada acrescida transformar essa filtragem em uma imagem mais complexa" [36]. Tem como vantagem o facto de o pré-processamento exigido ser muito menor comparativamente a outros algoritmos de classificação [37] [38].

Uma rede CNN reduz de forma gradual o tamanho do mapa de recursos, aumentando a profundidade enquanto avança para as camadas mais profundas. Estas últimas cobrem campos recetivos maiores, construindo uma representação mais abstrata, ao contrário das camadas mais superficiais. Desta forma, podem ser usadas camadas mais superficiais para objetos mais pequenos pois necessitam, apenas, de campos recetivos menores, e camadas mais profundas para objetos maiores [39]. Geralmente uma CNN é constituída pelas seguintes camadas: Camada de Entrada, Camada Convolucional, Camada de *Pooling* e Camada Completamente Ligada (ou Camada de Classificação) e Camada de Saída [36] [38] [40].

2.5.4.1 Camada de Entrada

É nesta camada que é colocada a imagem, a qual pode ser uma imagem 2D de camada única (escala de cinza), imagem 2D de 3 canais (cor Red, Green, Blue (RGB)) (figura 2.8) ou 3D [40].

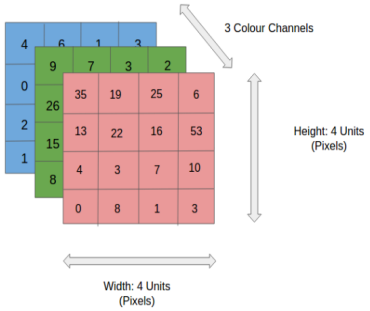

Figura 2.8: Imagem RGB 4x4x3.

2.5.4.2 Camada Convolucional

Esta camada é a primeira a extrair recursos de uma imagem de entrada, como arestas e curvas, sendo para isso necessário utilizar um *kernel* (ou seja, um filtro). Cada imagem tem o formato NxMxC, onde N é o número de linhas, M o número de colunas e C o número de canais de cor. Algumas imagens podem ter um tamanho inviável de *input*, o que faz com que seja utilizado um *kernel* para diminuir esse tamanho. Assim, o *kernel* passa pela imagem um número permitido de vezes, como por exemplo: numa matriz 5x5 com uma passada ou *stride* (o *kernel* é movido por um *pixel* e esse processo é repetido até que todos os locais possíveis na imagem sejam filtrados) de 1 (ou seja, movendo o filtro 1 *pixel* de cada vez), o *kernel* consegue cobri-la 9 vezes, criando um mapa de ativação de 3x3 (figura 2.9) [40]. A figura 2.9 representa uma situação simples mas podem existir situações mais complexas, como por exemplo uma imagem a cor RGB, ou seja com 3 canais. Neste caso a profundidade da matriz de entrada é 3. Assim, o *kernel* será aplicado a cada canal e os três valores obtidos são somados dando origem ao valor da matriz das *convolved features* [40].

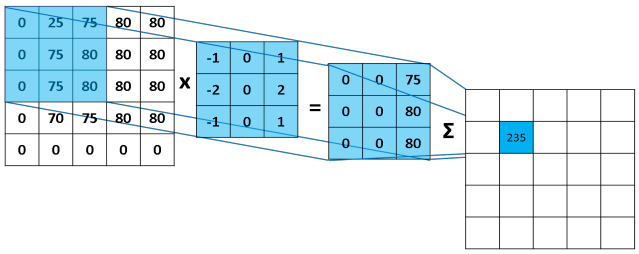

Figura 2.9: Convolução com uma matriz de entrada de 5x5 e um *kernel* de 3x3.

Ao *output* da Camada Convolucional pode ser adicionada uma função de ativação como a ReLu que tem como objetivo introduzir não-linearidade, o que tem bastante interesse no caso de se estar a trabalhar com imagens, por si só, não-lineares [40] [41].

2.5.4.3 Camada de *Pooling*

Tal como a camada descrita anteriormente, a Camada de *Pooling* é responsável pela diminuição do tamanho do *output* da camada anterior sem perda de informação, diminuindo, desta forma, a potência computacional necessária para processar os dados. Para além disto, esta camada é fundamental para que as camadas seguintes consigam extrair/captar detalhes que não sejam apenas bordas e curvas [38] [40].

Existem dois tipos de *pooling*: o *max pooling* e o *average pooling*, sendo o primeiro o mais popular pois tem melhor desempenho. Através do *max pooling* é escolhido o maior valor possível a partir de uma determinada região da matriz das *convolved features*. Além disso, o *max pooling* tem como vantagem o facto de suprimir o ruído e de evitar o *overfitting*, que se trata de um problema do *Machine Learning* que ocorre quando se constroem modelos que explicam de perto um conjunto de dados de treino "mas não generalizam quando aplicados a outros conjuntos de dados" [38] [41] [42]. Por outro lado, o *average pooling* faz a média aritmética dos valores da parte da imagem coberta pelo *kernel* (figuras 2.10 e 2.11) [38] [41].

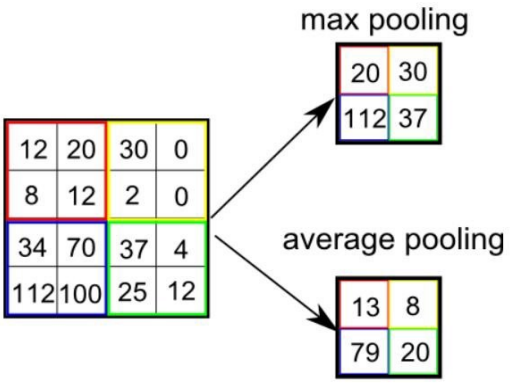

Figura 2.10: Tipos de *pooling*.

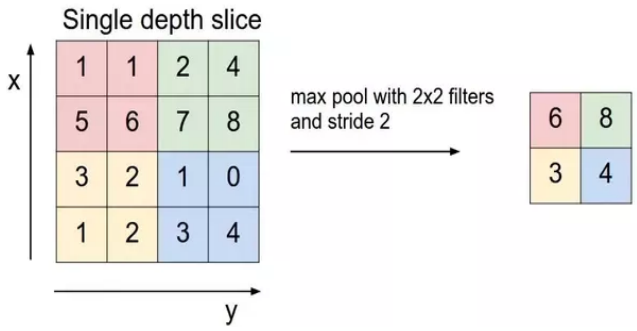

Figura 2.11: *Max pooling*.

2.5.4.4 Camada Completamente Ligada (*Fully Connected Layer*)

A camada completamente ligada liga todos os neurónios da camada anterior com os neurónios da camada seguinte, sendo que cada neurónio está ligado a todos os neurónios da camada seguinte. Esta camada combina todas as *features* extraídas da imagem de entrada pelas camadas anteriores e consegue comprimi-las num vetor coluna que servirá de *input* à camada de saída, a qual irá classificar a imagem utilizando, normalmente, a função *softmax* (função de ativação). A função *softmax* permite obter a probabilidade de uma dada imagem pertencer a qualquer uma das possíveis classes incluídas no problema em estudo [38] [40] [41] [43].

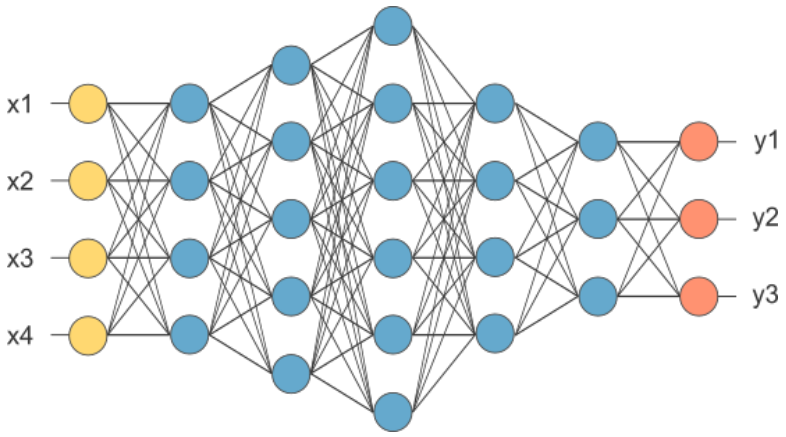

Figura 2.12: Últimas camadas de uma CNN.

Na figura 2.12, o resultado da interpolação das camadas de convolução e *pooling* é representada pelos círculos amarelos. Os círculos azuis representam

uma rede neuronal tipo *Multi-Layer Perceptron*. Esta possui o número de camadas escondidas necessário para a resolução do problema em estudo. A última camada, representada pelos círculos vermelhos, é constituída pelos neurónios ligados às classes existentes no problema [41]. Na figura 2.13 está representada a arquitetura completa de uma CNN.

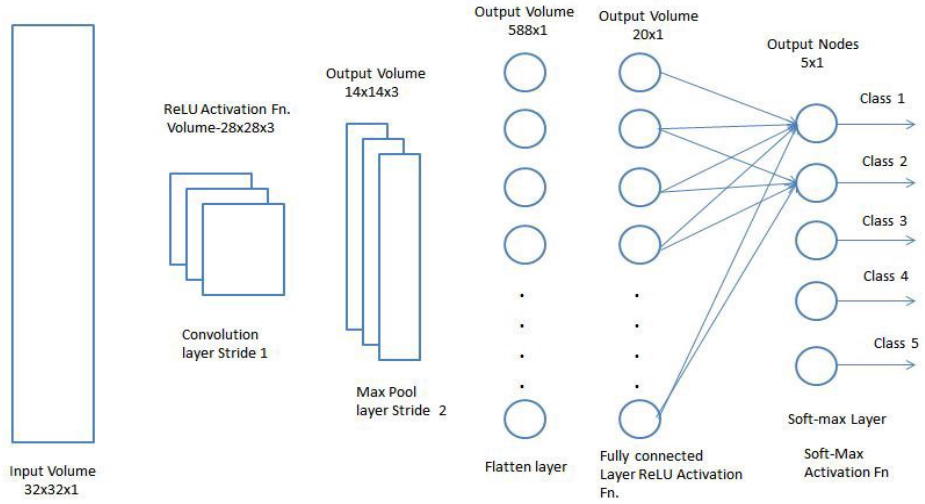

Figura 2.13: Arquitetura completa de uma CNN.

2.5.4.5 VGGNet

A VGGNet foi criada, em 2014, pelo *Visual Geometry Group* da Universidade de Oxford [44] [45]. Esta rede tem uma vantagem relativamente à AlexNet pois substitui os filtros grandes usados na AlexNet por vários filtros 3x3, um após o outro [44] [46]. Como os filtros de convolução são mais pequenos, a VGGNet pode ter um grande número de camadas de peso, o que lhe permite ter um melhor desempenho [47].

Tal como a AlexNet e a LeNet, a VGGNet é dividida em duas partes, sendo a primeira composta por camadas de convolução e *pooling* e a segunda por camadas completamente ligadas, como representado na figura 2.14. As camadas convolucionais nesta rede usam campos recetivos muito pequenos e podem, também, utilizar filtros de convolução 1x1 que são seguidos por uma função de ativação ReLu. O VGGNet possui três camadas completamente ligadas. As duas primeiras camadas possuem, cada uma, 4096 canais e a terceira 1000 canais, sendo 1 para cada classe. Relativamente às camadas ocultas, na VGGNet todas usam a função de ativação ReLu, o que permite, tal como na AlexNet, reduzir o tempo de treino [47] [48].

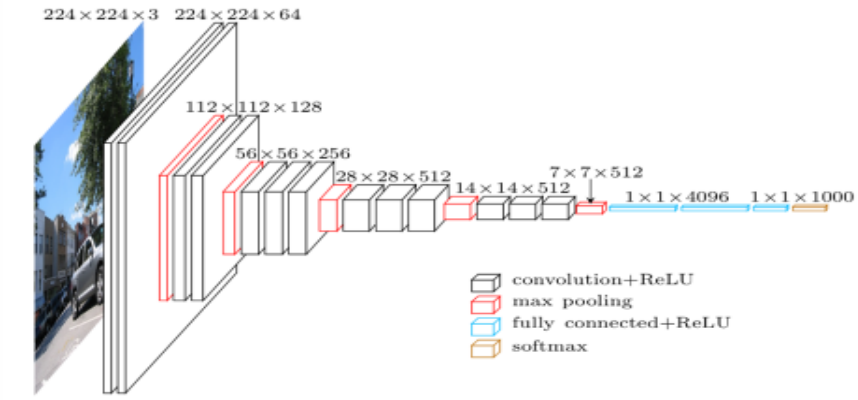

Figura 2.14: Exemplo de uma arquitetura de VGGNet.

2.5.4.6 ResNet

A ResNet, lançada em 2015, é uma RNA que se baseia na arquitetura do cérebro humano e tem a particularidade de utilizar a função ReLu, bem como "atalhos" para saltar algumas camadas [49]. Ao contrário da AlexNet e da VGGNet, que possuem arquitetura de redes sequenciais tradicionais, a ResNet possui uma arquitetura diferente composta por módulos de microarquitetura, ou seja, componentes básicos que no conjunto formam a macroarquitetura, isto é, a rede final [44]. A ResNet veio trazer uma grande vantagem ao nível do treino de redes profundas pois permite treinar redes com centenas a milhares de camadas com um bom desempenho. Tudo isto se deve à introdução de uma nova camada de rede neuronal, o bloco residual (figura 2.15) [50] [51]. O bloco residual possui duas camadas convolucionais, sendo cada uma seguida pela função de ativação ReLu. Utilizando o "atalho" é possível saltar essas duas operações de convolução e adiciona-se a entrada imediatamente antes da função de ativação ReLu final, como está demonstrado na figura 2.15 [48].

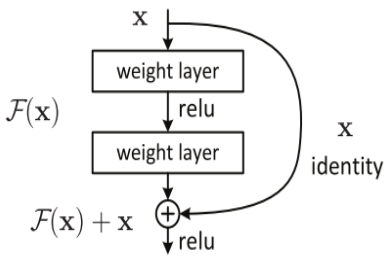

Figura 2.15: Bloco residual.

2.5.4.7 Outras Arquiteturas

Além das duas arquiteturas descritas anteriormente (VGGNet e ResNet), existem, ainda, as seguintes:

- **LeNet** - É uma CNN simples, ou seja, com muito menos camadas do que as atuais CNN profundas. Foi desenvolvida na década de 90 do século passado por Yann LeCunn com o objetivo de reconhecer dígitos manuscritos, utilizando imagens de tamanho 32x32. Como se trata da primeira CNN, a LeNet possui as camadas básicas de uma CNN, ou seja, camada convolucional, camada de *pooling* e camada completamente ligada [52] [53] [48] [54].

- **AlexNet** - É uma CNN projetada por Alex Krizhevsky em 2012 que é composta por 8 camadas, sendo 5 convolucionais e 3 completamente ligadas, e utiliza a função de ativação ReLu. A sua arquitetura é semelhante à da LeNet mas é mais profunda. Esta rede demonstrou que a profundidade é essencial para o bom desempenho [55].

- **GoogLeNet** - É a CNN vencedora do *ImageNet Large Scale Visual Recognition Competition* de 2014. É constituída por 22 camadas e vários módulos *Inception*. Foi desenvolvida pela Google com o objetivo de também poder ser usada num *smartphone*. Possui 12 vezes menos parâmetros que o seu antecessor AlexNet mas é igualmente rápida e muito mais precisa. Além disso é menos profunda que a ResNet, lançada posteriormente, e apresenta uma taxa de erro inferior à da VGGNet [56] [57].

2.6 *Intersection over Union* (IoU)

O IoU, também denominado como Índice de Jaccard, é uma métrica de avaliação que permite medir o índice de precisão de um detetor de objetos num conjunto de dados específico, ou seja, quantifica a semelhança entre a *ground truth bounding box* (desenhada a verde) e a *predicted bounding box* (desenhada a vermelho) para avaliar o quão boa é a *bounding box* (figura 2.16) [58] [59].

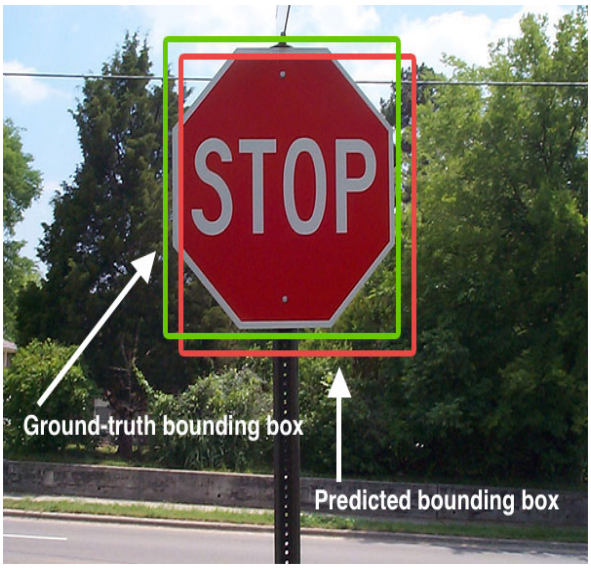

Figura 2.16: Exemplo de deteção de um sinal de trânsito, onde a *predicted bounding box* está desenhada a vermelho e a *ground truth bounding box* está desenhada a verde.

O IoU calcula-se através de uma equação, que é simplesmente uma razão. Esta razão mede a área de sobreposição entre a *predicted bounding box* e a *ground truth bounding box* sobre o domínio da união das duas *bounding boxes* juntas, ou seja, o numerador representa a área de sobreposição das duas *bounding boxes* e o denominador representa a área total abrangida por estas, como demonstrado na fórmula 2.1 e na figura 2.17, respetivamente. Desta forma obtém-se o valor de IoU, o qual pode variar entre 0 e 1, sendo que, quanto maior a semelhança entre as duas caixas, mais perto do 1 será a pontuação (uma pontuação IoU > 0,5 permite-nos inferir que o detetor de objetos utilizado apresenta uma boa previsão) [58] [59].

$$IoU(truth, pred) = \frac{Areaof Overlap}{Areaof Union} = \frac{truth \cap pred}{truth \cup pred} \tag{2.1}$$

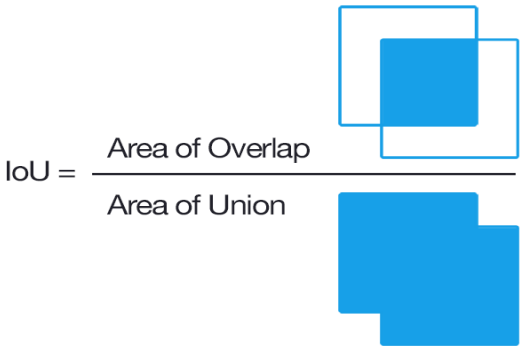

Figura 2.17: Equação para calcular o IoU.

2.6.1 Previsões: TP - FP - FN

A pontuação IoU permite classificar as previsões em:

- **Verdadeiros Positivos (TP)** - o objeto existe e o modelo consegue detetá-lo.
- **Falsos Positivos (FP)** - o objeto existe mas o modelo não coloca a *predicted bounding box* sobre o mesmo, isto é, deteta outra área na imagem que não a do objeto, ou então, o objeto não existe na realidade mas o modelo deteta um.
- **Falsos Negativos (FN)** - o objeto existe mas o modelo não o consegue detetar (figura 2.18) [58].

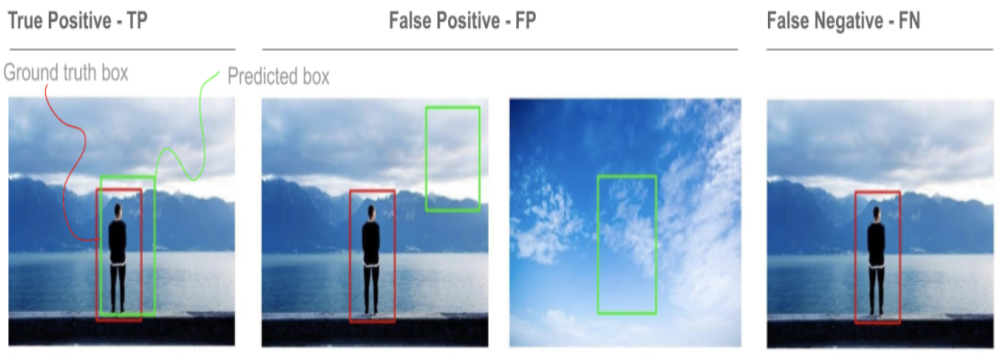

Figura 2.18: Exemplo de previsões TP - FP - FN.

2.6.2 Precisão e Sensibilidade (*Recall*)

As classificações anteriores permitem avaliar a precisão e sensibilidade do detetor de objetos. Precisão, ou valor preditivo positivo, representa a probabilidade das *predicted bounding boxes* corresponderem às *ground truth bounding boxes*, como representado na fórmula 2.2. O seu valor varia entre 0 e 1, sendo que quanto mais próximo de 1, mais preciso é o modelo. Um detetor de objetos com precisão de 0,7 indica que, quando um objeto é detetado, em 70% das vezes o detetor está correto.

$$Precision = \frac{TP}{TP+FP} = \frac{trueobjectdetection}{alldetectedboxes} \tag{2.2}$$

Sensibilidade corresponde à probabilidade dos objetos realmente presentes na imagem serem corretamente detetados pelas *ground truth bounding boxes*, como representado na fórmula 2.3. Tal como a precisão, o valor da sensibilidade também varia entre 0 e 1, sendo que, quanto mais próximo de 1 fôr o valor, mais sensível é o detetor. Portanto, um detetor com sensibilidade de 0,8 indica que o modelo deteta corretamente 80% dos objetos.

$$Recall = \frac{TP}{TP+FN} = \frac{trueobjectdetection}{allgroundtruthboxes} \tag{2.3}$$

Assim, uma sensibilidade alta e uma precisão baixa, indica que o modelo deteta todos os objetos realmente existentes na imagem, mas a maioria das deteções estão incorretas, ou seja, existem muitos Falsos Positivos. Uma baixa sensibilidade e uma precisão alta mostra que todas as *predicted bounding boxes* estão corretas, mas a maioria dos objetos realmente existentes na imagem não foi detetada, isto, é, o detetor apresenta muitos Falsos Negativos. Resumindo, um modelo de deteção ideal é aquele que consegue detetar corretamente a maioria dos objetos realmente existentes na imagem [58].

2.7 Conclusões

Este capítulo teve como principal foco a Visão Computacional e a IA, assim como as áreas que as rodeiam. É um capítulo essencialmente teórico onde foi explicado o que é e como funcionam os processos de deteção e reconhecimento de objetos, assim como o reconhecimento facial. Foram, ainda, mencionadas as várias redes artificias e CNN e foi explicada a arquitetura, o algoritmo e o funcionamento de cada uma de modo a ficar com uma ideia de quais as mais vantajosas ou quais as mais indicadas para cada situação.

Capítulo

3

Abordagem Proposta

3.1 Introdução

A deteção e o reconhecimento de faces humanas em imagens e/ou vídeos é possível com o uso da linguagem *Python* e de várias bibliotecas relacionadas com a mesma para *Machine Learning*.

O projeto consiste na deteção facial de alunos e docentes em sala de aula através do uso de uma câmara e de uma lente. Assim, o presente capítulo faz ênfase na seleção da câmara e da lente que vão ser utilizadas para a deteção de faces e descreve as tecnologias utilizadas que permitem também que esse feito seja possível.

3.2 *TensorFlow*

O *TensorFlow* é uma biblioteca de *software* livre, de arquitetura flexível e de código aberto para *Machine Learning* que pode ser aplicado a várias tarefas [60] [61]. Trata-se de um sistema para criação e treino de redes neuronais que permite detetar e decifrar padrões e correlações de forma semelhante à aprendizagem e raciocínio do ser humano [62].

O *TensorFlow* foi desenvolvido e lançado pela Google Brain em 2015 e veio substituir o *DistBelief*. O seu nome deriva das operações que essas redes neuronais executam em matrizes de dados multidimensionais, designadas como *Tensores*. Tem como vantagem o facto de poder ser executado em várias *Central Processing Unit* (CPU) e *Graphics Processing Unit* (GPU), bem como em dispositivos móveis, nomeadamente *Android* e *iOS* [61] [62].

3.3 Linguagem *Python*

Python é uma linguagem de programação interpretada, de alto nível, orientada a objetos, funcional, com uma semântica dinâmica, muito popular e utilizada em várias aplicações [63] [64] [65].

O *Python* foi criado no final dos anos 80 por Guido Van Rossum no Centrum Wiskunde & Informatica, nos Países Baixos, sendo considerado o sucessor da linguagem ABC. O nome tem como inspiração o grupo humorístico britânico Monty Python. Desde a sua criação têm vindo a ser desenvolvidas várias versões, sendo a última o *Python 3.0* que foi lançado no final de 2008. "Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos *Python Software Foundation*" [65].

Esta linguagem tem como "filosofia enfatizar a importância do esforço do programador sobre o esforço computacional", "prioriza a legibilidade do código sobre a velocidade ou expressividade" e "combina uma sintaxe concisa e clara com os recursos poderosos da sua biblioteca padrão e por módulos e *frameworks* desenvolvidos por terceiros" [65]. Trata-se de uma linguagem de propósito geral de alto nível, multiparadigma, funcional, suporta vários paradigmas de programação, incluindo a programação procedural, e tem a vantagem de permitir a fácil leitura do código, exigindo poucas linhas de código comparativamente a outras linguagens [65]. As suas "construções de linguagem e abordagem orientada a objetos têm como objetivo ajudar os programadores a escrever um código lógico e claro para projetos de pequena e grande escala. Dadas as suas características, tem como principal utilidade o processamento de textos, dados científicos e criação de *Common Gateways Interfaces* para páginas *Web*. O *Python* é, atualmente, utilizado em várias áreas e por várias empresas ou instituições, tais como Google, Youtube, NASA, Netflix e Air Canada [63] [64] [65] [66].

3.3.1 *Keras*

Keras é uma biblioteca de redes neuronais escrita em *Python*, que pode utilizar o *TensorFlow*, o *Microsoft Cognitive Toolkit* e o *Theano* como *backends*. Foi desenvolvida por François Chollet, engenheiro da Google, e lançada em 2015. O *Keras* tem sofrido várias atualizações, sendo a versão mais recente o *Keras 2.3.0*. O objetivo principal é permitir experimentação rápida e foi projetada de forma a ser modular e extensível, rápida e fácil de utilizar. Tem, também, como vantagens o facto de suportar redes convolucionais, redes recorrentes e a combinação de ambas e de funcionar em CPU e GPU [67] [68] [69] [70].

3.3.2 *NumPy*

O *NumPy*, criado em 2005 por Travis Oliphant com base em recursos do *Numarray*, é uma biblioteca utilizada para a linguagem *Python*. Esta "suporta *arrays* e matrizes multi-dimensionais" e possui um vasto conjunto de funções matemáticas para trabalhar com estas estruturas.

O *NumPy* e o *MATLAB* têm como vantagem o facto de permitirem escrever programas rápidos se a maioria das operações utilizar *arrays* multidimensionais ou matrizes em vez de escalares [71].

3.3.3 *Matplotlib*

Matplotlib é uma biblioteca de *software* feita para e a partir do *Python* e do *NumPy* com o objetivo de criar gráficos e visualizar dados. Esta fornece uma *Application Programming Interface* (API) orientada a objetos para incluir gráficos em aplicações usando *toolkits* de interface gráfica como *Tkinder*, *WxPython*, *Qt* ou GTK.

A biblioteca *Matplotlib* foi, originalmente, criada pelo biólogo e neurocientista americano John D. Hunter e, desde então, tem uma comunidade de desenvolvimento ativa, sendo distribuída sob uma licença BSD. Um pouco antes da morte do criador John Hunter em agosto de 2012, o programador Michael Droetboom foi nomeado o líder do projeto, juntando-se depois a ele o cientista Thomas Caswell.

O *Matplotlib* possui o módulo *Pyplot* com uma interface semelhante ao *MATLAB*. O *Matplotlib* é projetado e desenvolvido para ter o mesmo uso do *MATLAB*, mas com a flexibilidade da linguagem *Python*, e com as vantagens de ser código aberto e totalmente gratuito [72].

3.3.4 *Python Imaging Library* (PIL)

PIL é mais uma das bibliotecas da linguagem *Python*. Esta dá apoio à abertura e gravação de imensos formatos de imagens diferentes, tais como: PNG, BMP, EPS, TIFF e GIF.

Existem vários pacotes que usam o PIL para fazer manipulação de imagens, como é o caso das bibliotecas *Numeric Python* e *SciPy*, sendo que a primeira dá apoio a matrizes e *arrays* grandes e multi-dimensionais e, a segunda é uma biblioteca de rotinas numéricas e científicas [73].

3.4 *Open Source Computer Vision Library* (OpenCV)

O OpenCV foi desenvolvido pela Intel, em 2000, e é uma biblioteca multiplataforma, totalmente livre para utilização académica e comercial com o objetivo de o desenvolvimento de aplicações na área da Visão Computacional, seguindo apenas o modelo de licença BSD Intel.

A biblioteca OpenCV foi desenvolvida nas linguagens de programação C/C++, e dá apoio a programadores que utilizem *Python*, *Java* e *Visual Basic*. Esta possui módulos de processamento de imagens e vídeo I/O, álgebra linear, entre outros, e, ainda, centenas de algoritmos de Visão Computacional, entre os quais: reconhecimento de objetos, filtros de imagem e calibração de câmara [74].

3.5 Seleção da Câmara e da Lente e Captação de Vídeo

Este projeto tem como objetivo o registo de assiduidade de alunos e docentes em sala de aula, sendo apenas necessária a sua presença na mesma, ou seja, não haverá intervenção ativa destes. Esse registo de assiduidade é feito através de um vídeo captado por uma câmara, instalada na sala de aula, adaptada com uma lente compatível e, posteriormente, será feito a deteção facial de cada um dos presentes.

3.5.1 Seleção da Configuração Final da Câmara e da Lente

Previamente à fase de instalação da câmara na sala de aula, foi realizado um inventário de todas as câmaras e lentes existentes e disponíveis no *Soft Computing and Image Analysis Laboratory* (SOCIALAB), o que permitiu gerar um conjunto de possíveis casos, alternando as câmaras e as lentes para cada configuração. Nas tabelas 3.1 e 3.2, são apresentadas as câmaras e as lentes disponíveis no SOCIALAB.

Fabricante	**Modelo**
Canon	EOS 5D
JAI/Infaimon	BM-141GE
Stingray F504B ASG	073661
JAI/Infaimon	AD-080GE
Infaimon/IDS	5481VSE-C-SD32
IDS	UI-1240ML-C-HQ
Protos	NCL580
Canon	XM2
uEye	UI-1225LE-M / 080056

Tabela 3.1: Câmaras Disponíveis.

Fabricante	**Modelo**
Nikon	Nikkor/GMZ3D85900MCN
Infaimon	GMTHR33520MCN
Infaimon	OPT-MHR47518MCN
Infaimon	HR F2.8/50mm
Infaimon	OPT-MHR38014MCN
Infaimon	302298
Infaimon	OPT-MN38014MCN-1
Infaimon	11-1248
Canon	TV3514
Canon	ET-83C

Tabela 3.2: Lentes Disponíveis.

Após a realização do inventário, selecionaram-se as câmaras e as lentes que podiam ser usadas para possíveis testes iniciais. Desta seleção resultou um conjunto de 6 câmaras e 9 lentes, como demonstrado nas tabelas 3.3 e 3.4.

Fabricante	**Modelo**
Canon	EOS 5D
JAI/Infaimon	BM-141GE
Stingray F504B ASG	073661
JAI/Infaimon	AD-080GE
IDS	UI-1240ML-C-HQ
uEye	UI-1225LE-M / 080056

Tabela 3.3: Câmaras Selecionadas.

Fabricante	**Modelo**
Infaimon	GMTHR33520MCN
Infaimon	OPT-MHR47518MCN
Infaimon	HR F2.8/50mm
Infaimon	OPT-MHR38014MCN
Infaimon	302298
Infaimon	OPT-MN38014MCN-1
Infaimon	11-1248
Canon	TV3514
Canon	ET-83C

Tabela 3.4: Lentes Selecionadas.

A compatibilidade entre câmaras e lentes deu origem a um total de 37 casos possíveis. Seguidamente, foram realizados testes com os 37 casos possíveis, primeiro no SOCIALAB e, depois, na sala 6.03, o que permitiu reduzir o número de casos para um total de 5. A maioria dos casos foi excluída devido ao facto de a câmara e a lente não reproduzirem boas imagens, porque estas ou eram muito escuras, ou muito desfocadas ou demasiado próximas.

Os 5 casos que restaram resultam da utilização de apenas uma câmara e duas lentes, alternando-se só o tamanho do tripé que suportava a câmara. Para cada um destes casos foi feito um vídeo na sala 6.03 com cerca de 1 minuto e, no final, apenas foi escolhida uma configuração entre câmara e lente. Esta configuração foi a melhor em termos de resolução de imagem, de cor e de espaço na sala. A configuração final junta a Câmara IDS (UI-1240ML-C-HQ) com a Lente Infaimon (302298), colocando-se o tripé a metade da sua capacidade em altura (com 2 espaços).

3.5.2 Captação de Vídeo e Geração de *Frames*

Posteriormente, utilizando a configuração final referida anteriormente, realizou-se um vídeo, com cerca de 40 minutos, no qual participaram 10 pessoas (alunos). As propriedades do vídeo são as que se seguem:

- **Tempo:** 41 minutos e 35 segundos.
- **Tamanho de cada *frame*:** 1280x1024 px.
- **Taxa de *frames per second* (fps):** 21.32.
- **Pessoas presentes no vídeo:** 10 pessoas.

Foi a partir deste vídeo que se geraram as *frames* necessárias para a realização da etapa seguinte do presente projeto. Para isso, criou-se um pequeno *script* escrito em *Python*, que permitiu reduzir, de forma aleatória, o número de *frames*, das cerca de 53000 iniciais para as 1000 utilizadas no projeto. As últimas encontram-se guardadas numa pasta específica, com o formato *"frame%d.jpg"*, cujo o nome é *FrameExtraction*. Um exemplo desse conjunto de 1000 *frames* geradas está demonstrado na figura 3.1.

Figura 3.1: Exemplo de uma *frame* gerada aleatoriamente (*frame0.jpg*).

3.6 *Computer Vision Annotation Tool* (CVAT)

O CVAT é uma ferramenta *online* e gratuita de anotação de imagem e vídeo que é usada para classificar dados para algoritmos de Visão Computacional. Foi desenvolvida pela Intel com o objetivo de anotar dados, com propriedades diferentes, para as principais tarefas de *Machine Learning* supervisionado, sendo elas: deteção de objetos, classificação de imagens e segmentação das mesmas. Tem como características vantajosas: a interpolação de *bounding boxes* entre *frames* específicas (figura 3.2), a anotação semiautomática, através da utilização de modelos de DL nos formatos de API de deteção de objetos Intel OpenVINO e *TensorFlow*, um painel com uma lista de tarefas de anota-

ção, autenticação de acesso básico, entre outros [75] [76] [77].

Esta ferramenta é escrita em *JavaScript*, HTML, CSS, *Django* e *Python*. O seu código-fonte está disponível no *GitHub* e é distribuída sob a licença do MIT [75] [76].

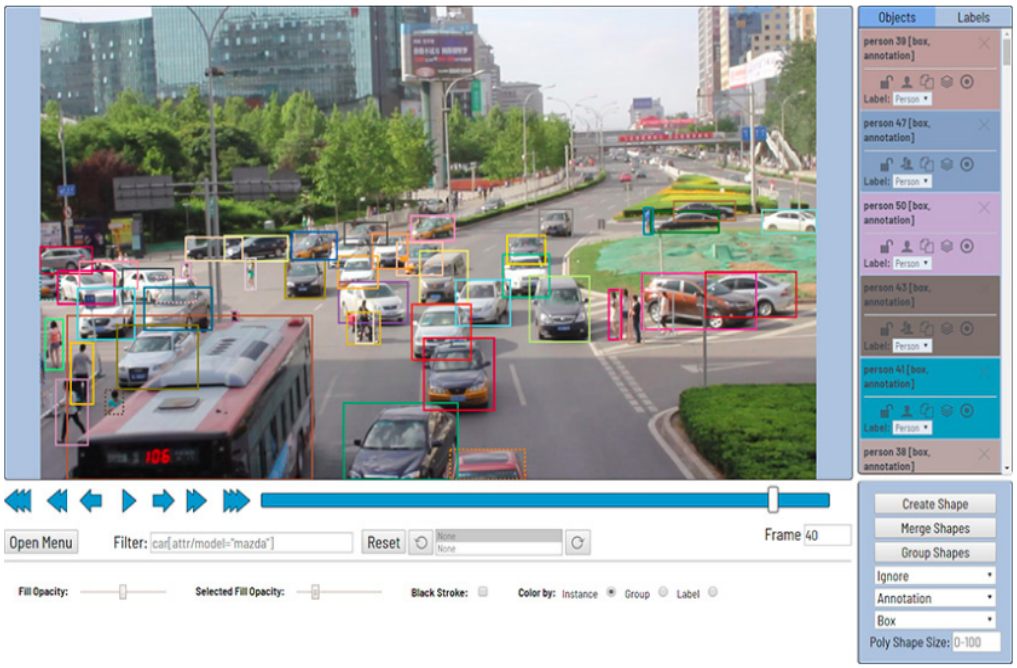

Figura 3.2: Exemplo de funcionamento da CVAT usando *bounding boxes*.

3.7 Conclusões

Este capítulo teve como foco a explicação da linguagem utilizada e das bibliotecas que permitem fazer uma deteção e um reconhecimento de uma face humana. Demonstrou-se, ainda, a variedade de câmaras e lentes disponíveis, através da apresentação de um inventário, e a forma como foi realizada a seleção da câmara e da lente para gerar os dados para a realização do presente projeto. Por fim, foi feita uma breve apresentação sobre as características do vídeo, a forma como foi captado, as *frames* geradas a partir deste e a aplicação que tratou de fazer as anotações manuais em cada *frame*.

Capítulo

4

Experiências e Resultados

4.1 Introdução

Das duas fases que o presente projeto englobava, a fase 1 ficou concluída, que correspondia à deteção e *tracking* de faces/silhuetas humanas. No que a um processo de *tracking* diz respeito, quando se está perante um processo deste tipo para qualquer objeto, alguns erros podem acontecer. No caso específico abordado por este projeto, um vídeo foi gravado através de uma câmara e de uma lente numa sala de aula. Este vídeo deu origem a um conjunto elevado de *frames* que, posteriormente, foram reduzidas para um número consideravelmente mais pequeno. Neste número reduzido de *frames*, *bounding boxes* foram colocadas ao redor das faces das pessoas que estiveram presentes no dia da captação dos dados por parte da câmara. Foram atribuídos IDs únicos de forma igual a cada um dos indivíduos, retratando, assim, em jeito de teoria, a ação de *tracking* à medida que se ia avançando nas anotações das *bounding boxes*.

O desafio proposto nesta primeira fase passou por detetar as faces das pessoas presentes na sala de aula, fazendo uma comparação entre as *ground-truth bounding boxes* (*bounding boxes* feitas à mão usando a aplicação CVAT) e as *predicted bounding boxes* do modelo de rede neuronal que se usou (YOLOv3), avaliando, assim, a performance do modelo de rede utilizado [1].

[1] https://github.com/sthanhng/yoloface

4.2 Etapas de Desenvolvimento

Inicialmente, foi captado um vídeo numa sala de aula da Universidade da Beira Interior, a partir do qual foi gerado um vasto conjunto de *frames*, tal como descrito anteriormente. De seguida, reduziu-se o número de *frames* para as 1000 usadas no presente projeto. Na figura 4.1 está representado um exemplo de uma dessas *frames*.

Figura 4.1: *Frame* nº 27000 original (gerada do vídeo capturado).

Posteriormente, foi realizada a anotação manual das *bounding boxes* que delimitam as faces das pessoas presentes na sala, usando a ferramenta CVAT (figura 4.2). Neste processo foi atribuído um ID diferente para cada pessoa, que se manteve ao longo das *frames*. No final desta etapa, as anotações foram extraídas desta ferramenta *online* num formato específico (Pascal VOC), criando, assim, um ficheiro *.xml* para cada *frame*. Estes ficheiros foram, depois, enviados para um mesmo ficheiro *.txt*, com o nome *Coordinates_annotations.txt*, que contém o nome de cada *frame* e os vetores de coordenadas de todas as faces identificadas por *frame*, no formato *"frame%d.jpg, [[xmin, ymin, xmax, ymax], [xmin, ymin, xmax, ymax]...]"*.

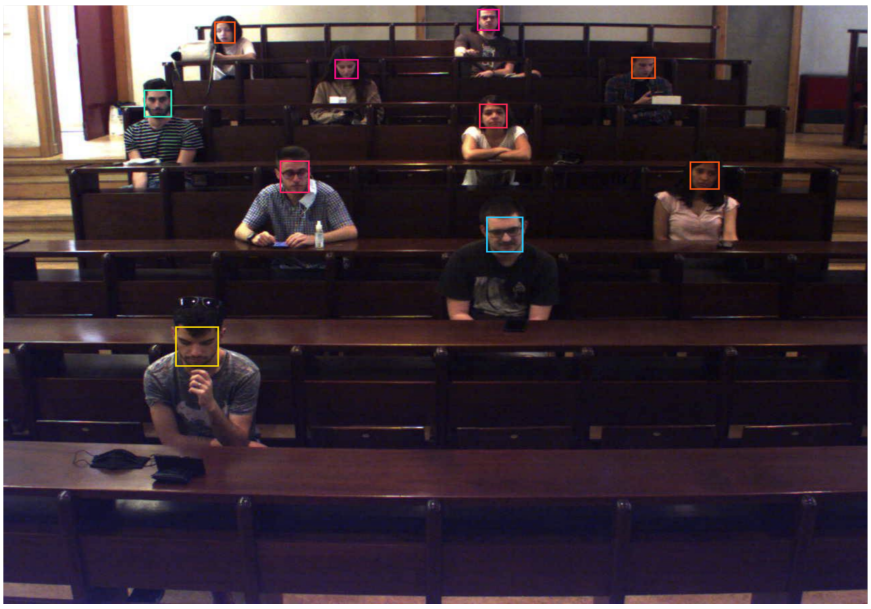

Figura 4.2: *Frame* nº 27000 com *bounding boxes* marcadas à mão (gerada da CVAT).

Seguidamente, entra-se na fase em que se trabalha o modelo de deteção de faces, um algoritmo do YOLOv3, que foi utilizado para deteção de faces em tempo real. Este algoritmo já se encontrava pré-treinado para a função de deteção facial, sendo necessário fornecer-lhe acesso às *frames* originais para que ele possa desenhar *bounding boxes* naquilo que ele identifica como sendo faces humanas (figura 4.3). Da mesma forma que para as anotações manuais, também para as observações do modelo foi criado um ficheiro *.txt* denominado *Coordinates_yoloface.txt*, que contém o nome de cada *frame* e os vetores de coordenadas de todas as faces identificadas por *frame*, no formato *"frame%d.jpg, [[left, top, right, bottom], [left, top, right, bottom]...]"*. Dado que, o ficheiro *Coordinates_yoloface.txt* não apresentava as coordenadas com a mesma ordem do ficheiro *Coordinates_annotations.txt*, foi necessário criar um terceiro ficheiro *.txt*, com o nome *Coordinates_yoloface_ordered.txt*, para ordenar as coordenadas do modelo segundo o ordem das coordenadas das anotações manuais.

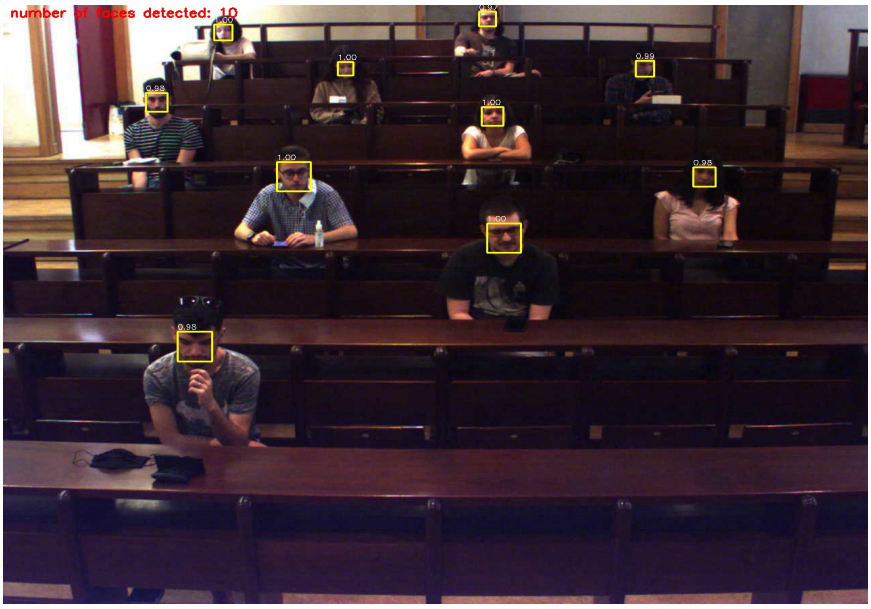

Figura 4.3: *Frame* nº 27000 com o número de faces detetadas e as *bounding boxes* que o modelo desenhou (gerada do *output* do modelo).

Por fim, é cálculado o valor do IoU para o modelo utilizado, comparando as *ground truth bounding boxes* (desenhadas a verde) com as *predicted bounding boxes* (desenhadas a vermelho), com o objetivo de avaliar a performance do mesmo (figura 4.4). Para isso, é necessário calcular o IoU de cada face em cada *frame*. Depois, calcula-se a média do IoU para cada *frame* (obtida em *MediaFinal_IoU.txt*), executando o ficheiro *Intersection_Over_Union.py* e, finalmente, é calculada a média final de todas as *frames*, o que corresponde ao IoU do modelo, obtido na linha de comandos ao executar o ficheiro *MediaIoU.py*.

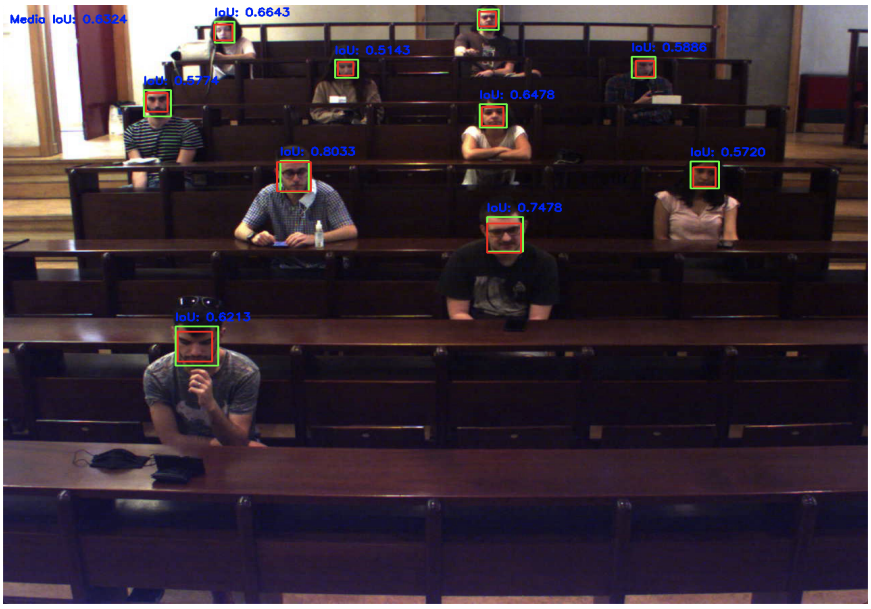

Figura 4.4: *Frame* nº 27000 com o valor do IoU para cada face, a respetiva média da *frame* e as *ground truth bounding boxes* (desenhadas a verde) juntamente com as *predicted bounding boxes* (desenhadas a vermelho).

O *Pipeline* seguinte resume todos os procedimentos descritos anteriormente (figura 4.5).

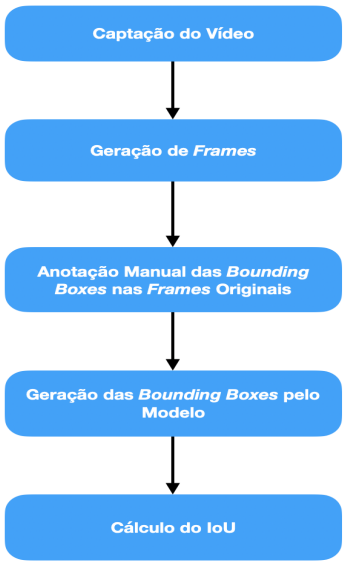

Figura 4.5: *Pipeline* das etapas de desenvolvimento.

4.3 Discussão e Resultados

A partir de uma análise geral das imagens enquanto o modelo está a executar, pode verificar-se que na maioria dos casos o modelo deteta as faces presentes na imagem, sendo coincidente com as *ground truth bounding boxes*, mas, em algumas situações não consegue detetá-las, nomeadamente nos momentos em que as pessoas estão em posições onde a face não aparece na totalidade e/ou em casos em que as pessoas se encontram de costas para a câmara. Aos casos em que a face aparece na *frame*, mas o modelo não a deteta chamam-se Falsos Negativos. Quanto maior o número de Falsos Negativos, menor a sensibilidade do modelo.

Analisando as *frames* verifica-se que não há nenhum caso em que o modelo coloque uma *bounding box* numa região da *frame* sem faces. Posto isto, pode afirmar-se que não há Falsos Positivos, portanto, a precisão do modelo não é afetada.

O *output* do modelo apresenta o valor final do IoU, bem como o melhor e pior valores de IoU de todas as *frames*. Para o melhor resultado numa *frame* o valor foi de 0.75, e para o pior o valor foi de 0.45. O cálculo final do IoU deu um valor de, aproximadamente, **0.62**. Deste valor pode inferir-se que, em **62%** dos casos, o modelo deteta corretamente as faces, ou seja, em **62%** dos casos as *predicted bounding boxes* são coincidentes com as *ground truth bounding boxes*. O facto do valor do IoU se afastar de 1 deve-se a que, tal como descrito anteriormente, em algumas situações a face está presente na *frame* e é assinalada pela *ground truth bounding box*, mas o modelo não consegue detetá-a, atribuindo-lhe um valor de 0 (Falsos Negativos). Isto implica que, no numerador do IoU nessa *frame* sejam adicionados valores de 0, não alterando o valor do numerador, mas o valor do denominador aumenta, o que leva a que a média dessas *frames* seja baixa. Por fim, estas falhas na deteção facial fazem com que o modelo tenha uma performance inferiror à que seria desejável, embora, no geral, apresente uma **boa previsão** (> 0.50).

Capítulo

5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

Este trabalho permitiu dar um salto na área das redes neuronais, nomeadamente ao nível da teoria e das respetivas implementações na prática. Ao realizar este trabalho adquiriu-se conhecimento acerca das redes neuronais e onde estas são aplicadas. Ao longo do projeto procuraram-se soluções que dessem resposta aos problemas e desafios que iam surgindo. Espera-se que, após a conclusão deste trabalho, o conhecimento foi adquirido possa vir a ser expandido e complementado.

5.2 Trabalho Futuro

Com vista à complementação do trabalho apresentado, algumas melhorias e/ou enriquecimentos poderiam ser feitos. Destacam-se os seguintes pontos:

- **Utilizar outros modelos de deteção de faces** - tendo em conta que estão disponíveis vários modelos de deteção de faces (por exemplo, SSD, Faster R-CNN) com funcionamento, vantagens e desvantagens diferentes, seria interessante e importante aplicar, também, esses modelos.

- **Comparar a performance de cada modelo de deteção de faces** - tal como referido acima, cada modelo tem o seu respetivo funcionamento e as suas vantagens/desvantagens. Assim, estes podem ser aplicados e a sua performance avaliada de forma a concluir qual será mais indicado para cada situação.

- **Utilizar um grupo maior de pessoas para a recolha de dados** - considerando a situação de pandemia que estamos a enfrentar não foi possível captar o vídeo com um número maior de pessoas na mesma sala, o que seria de valor para verificar como os modelos se comportam na deteção de um maior conjunto de faces.
- **Aplicar a fase 2 do presente trabalho (Identificação biométrica em ambientes não-cooperativos ou de video-vigilância)** - com a realização desta fase dar-se-ia por cumprido o objetivo geral do trabalho que seria o registo da assiduidade dos alunos e docentes sem a intervenção direta dos mesmos.

Bibliografia

[1] Wikipedia. Computer vision, 2020. [Online] https://en.wikipedia.org/wiki/Computer_vision. Último acesso a 9 de Abril de 2020.

[2] SAS Institute. Computer vision- What it is and why it matters, 2020. [Online] https://www.sas.com/en_us/insights/analytics/computer-vision.html. Último acesso a 9 de Abril de 2020.

[3] IBM. Computer vision – use machine learning and neural networks to teach computers to see. [Online] https://www.ibm.com/topics/computer-vision. Último acesso a 9 de Abril de 2020.

[4] Medium (Ilija Mihajlovic). Everything You Ever Wanted To Know About Computer Vision, 2019. [Online] https://towardsdatascience.com/everything-you-ever-wanted-to-know-aboutcomputer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e. Último acesso a 9 de Abril de 2020.

[5] Wikipedia. Artificial intelligence, 2020. [Online] https://en.wikipedia.org/wiki/Artificial_intelligence Último acesso a 10 de Abril de 2020.

[6] SAS Institute. Artificial intelligence – what it is and why it matters, 2020. [Online] https://www.sas.com/en_us/insights/analytics/what-is-artificial-intelligence.html Último acesso a 10 de Abril de 2020.

[7] Investopedia. Artificial intelligence (AI). [Online] https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp Último acesso a 10 de Abril de 2020.

[8] Runrun.it. O que é inteligência artificial e suas aplicações no presente (e no futuro!) dos negócios. [Online] https://blog.runrun.it/o-que-e-inteligencia-artificial/ Último acesso a 10 de Abril de 2020.

[9] SAS Institute. Machine Learning – o que é e qual a sua importância?, 2019. [Online] https://www.sas.com/pt_br/insights/analytics/machine-learning.html Último acesso a 20 de Abril de 2020.

[10] Wikipedia. Machine Learning, 2020. [Online] https://en.wikipedia.org/wiki/Machine_learning Último acesso a 20 de Abril de 2020.

[11] DSPA (Data Science Portuguese Association. Aprendizagem Supervisionada (Supervised Learning), 2019. [Online] https://dspa.pt/glossary/aprendizagem-supervisionada-supervised-learning/ Último acesso a 20 de Abril de 2020.

[12] Medium (Gabi Viana). Diferença entre Aprendizado de Máquina (Machine Learning) & Aprendizagem Profunda (Deep Learning), 2018. [Online] https://medium.com/@gabi.viana11/diferen%C3%A7a-entre-aprendizado-de-m%C3%A1quina-machine-learning-aprendizagem-profunda-deep-learning-3035e95ba1d1 Último acesso a 20 de Abril de 2020.

[13] Wlademir Ribeiro Prates. Aprendizado de máquina: supervisionado e não supervisionado, 2018. [Online] https://www.wrprates.com/aprendizado-de-maquina-supervisionado-e-nao-supervisionado/ Último acesso a 20 de Abril de 2020.

[14] Ciência e Dados (David Matos). Conceitos fundamentais de Machine Learning. [Online] https://www.cienciaedados.com/conceitos-fundamentais-de-machine-learning/ Último acesso a 20 de Abril de 2020.

[15] Investopedia. Deep Learning, 2019. [Online] https://www.investopedia.com/terms/d/deep-learning.asp Último acesso a 20 de Abril de 2020.

[16] F. M. F. Ferreira e C. A. P. S. Martins B. A. G. de Oliveira. FLOD-Net - Detecção e reconhecimento de objetos em dispositivos de baixa especificação: um estudo de caso em classificação de alimentos. [Online] https://seer.ufrgs.br/index.php/reic/article/viewFile/83477/48325 Último acesso a 23 de Abril de 2020.

[17] Amazon Web Services. Os fatos sobre a tecnologia de reconhecimento facial com inteligência artificial, 2020. [Online] https://aws.amazon.com/pt/rekognition/the-facts-on-facial-

recognition-with-artificial-intelligence/ Último acesso a 23 de Abril de 2020.

[18] Usemobile (Luís Otávio). Reconhecimento Facial: como funciona, 2019. [Online] https://usemobile.com.br/reconhecimento-facial-como-funciona/ Último acesso a 23 de Abril de 2020.

[19] Forbes (Bernard Marr). What Are Artificial Neural Networks - A Simple Explanation For Absolutely Anyone, 2018. [Online] https://www.forbes.com/sites/bernardmarr/2018/09/24/what-are-artificial-neural-networks-a-simple-explanation-for-absolutely-anyone/#3070de2f1245 Último acesso a 28 de Abril de 2020.

[20] Wikipedia. Artificial Neural Network, 2020. [Online] https://https://en.wikipedia.org/wiki/Artificial_neural_network Último acesso a 28 de Abril de 2020.

[21] Medium (Nagesh Singh Chauhan). Introduction to Artificial Neural Networks(ANN), 2019. [Online] https://towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9 Último acesso a 28 de Abril de 2020.

[22] Dragomir Anguelov *et al* Wei Liu. SSD: Single Shot MultiBox Detector, 2016. [Online] https://arxiv.org/pdf/1512.02325.pdf Último acesso a 2 de Maio de 2020.

[23] Medium (Sik-Ho Tsang). Review: SSD — Single Shot Detector (Object Detection), 2018. [Online] https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11 Último acesso a 2 de Maio de 2020.

[24] Medium (Hao Gao). Understand Single Shot MultiBox Detector (SSD) and Implement It in Pytorch, 2018. [Online] https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad Último acesso a 2 de Maio de 2020.

[25] ArcGIS. How single-shot detector (SSD) works?, 2020. [Online] https://developers.arcgis.com/python/guide/how-ssd-works/ Último acesso a 2 de Maio de 2020.

[26] Pyimagemsearch (Adrian Rosebrock). YOLO object detection with OpenCV, 2018. [Online] https://www.pyimagesearch.com/2018/11/

12/yolo-object-detection-with-opencv/ Último acesso a 2 de Maio de 2020.

[27] Medium (Open Data Science). Overview of the YOLO Object Detection Algorithm, 2018. [Online] https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0 Último acesso a 2 de Maio de 2020.

[28] Santosh Divvala *et al* Joseph Redmon. You Only Look Once: Unified, Real-Time Object Detection. [Online] https://pjreddie.com/media/files/papers/yolo_1.pdf Último acesso a 2 de Maio de 2020.

[29] Paperspace Blog (Ayoosh Kathuria). How to implement a YOLO (v3) object detector from scratch in PyTorch: Part 1. [Online] https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/ Último acesso a 2 de Maio de 2020.

[30] Medium (Ayoosh Kathuria). What's new in YOLO v3?, 2018. [Online] https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b Último acesso a 2 de Maio de 2020.

[31] Medium (Shilpa Ananth). Faster R-CNN for object detection, 2019. [Online] https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46 Último acesso a 2 de Maio de 2020.

[32] Medium (Yinghan Xu). Faster R-CNN (object detection) implemented by Keras for custom data from Google's Open Images Dataset V4, 2018. [Online] https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a Último acesso a 2 de Maio de 2020.

[33] Missinglink.ai. Faster R-CNN: Detecting Objects Without the Wait. [Online] https://missinglink.ai/guides/convolutional-neural-networks/faster-r-cnn-detecting-objects-without-wait/ Último acesso a 2 de Maio de 2020.

[34] Kaiming He *et al* Shaoqing Ren. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016. [Online] https://arxiv.org/abs/1506.01497f62b9141a Último acesso a 2 de Maio de 2020.

[35] Wikipedia. Rede Neuronal Convolucional, 2020. [Online] https://pt.wikipedia.org/wiki/Rede_neural_convolucional Último acesso a 4 de Maio de 2020.

[36] Medium (Giseli Alves). Entendendo Redes Convolucionais (CNNs), 2018. [Online] https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184 Último acesso a 4 de Maio de 2020.

[37] Aliger. As Redes Neurais Convolucionais no Deep Learning, 2019. [Online] https://www.aliger.com.br/blog/as-redes-neuronais-convolutivas-no-deep-learning Último acesso a 4 de Maio de 2020.

[38] Medium (Sumit Saha). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, 2018. [Online] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 Último acesso a 4 de Maio de 2020.

[39] Medium (Hao Gao). Understand Single Shot MultiBox Detector (SSD) and Implement It in Pytorch, 2018. [Online] https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad Último acesso a 4 de Maio de 2020.

[40] MLNotebook. Convolutional Neural Networks - Basics, 2017. [Online] https://mlnotebook.github.io/post/CNN1/ Último acesso a 4 de Maio de 2020.

[41] Medium (Prabhu). Understanding of Convolutional Neural Network (CNN) — Deep Learning, 2018. [Online] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 Último acesso a 4 de Maio de 2020.

[42] Data Science Bowl (Mike Kim). Overfitting, 2015. [Online] https://datasciencebowl.com/overfitting/ Último acesso a 4 de Maio de 2020.

[43] LinkedIn (Alex Fernandes Mansano). O que é uma rede neuronal convolucional, 2017. [Online] https://www.linkedin.com/pulse/o-que-%25C3%25A9-um-rede-neural-convolucional-alex-fernandes-mansano Último acesso a 4 de Maio de 2020.

[44] Pyimagesearch (Adrian Rosebrock). ImageNet: VGGNet, ResNet, Inception and Xception with Keras I, 2017. [Online] https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/ Último acesso a 4 de Maio de 2020.

[45] Medium (Edward Ma). What is the VGG neural network?, 2017. [Online] https://becominghuman.ai/what-is-the-vgg-neural-network-a590caa72643 Último acesso a 4 de Maio de 2020.

[46] Neurohive (Muneeb ul Hassan). VGG16 – Convolutional Network for Classification and Detection, 2018. [Online] https://neurohive.io/en/popular-networks/vgg16/ Último acesso a 4 de Maio de 2020.

[47] Medium (Jerry Wei). VGG Neural Networks: The Next Step After AlexNet, 2019. [Online] https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c Último acesso a 4 de Maio de 2020.

[48] Zachary C. Lipton *et al* Aston Zhang. Dive Into Deep Learning, 2020. [Online] https://d2l.ai/d2l-en.pdf Último acesso a 4 de Maio de 2020.

[49] Wikipedia. Residual Neural Network, 2020. [Online] https://en.wikipedia.org/wiki/Residual_neural_network Último acesso a 4 de Maio de 2020.

[50] Medium (Connor Shorten). Introduction to ResNet, 2019. [Online] https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4 Último acesso a 4 de Maio de 2020.

[51] Medium (Vincent Fung). An Overview of ResNet and its Variants, 2017. [Online] https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035 Último acesso a 4 de Maio de 2020.

[52] Wikipedia. LeNet, 2020. [Online] https://en.wikipedia.org/wiki/LeNet Último acesso a 4 de Maio de 2020.

[53] Muhammad Rizwan. LeNet-5 – A Classic CNN Architecture, 2018. [Online] https://engmrk.com/lenet-5-a-classic-cnn-architecture Último acesso a 4 de Maio de 2020.

[54] Sovit Ranjan Rath. LeNet-5: A Practical Approach, 2019. [Online] https://debuggercafe.com/lenet-5-a-practical-approach/ Último acesso a 4 de Maio de 2020.

[55] Wikipedia. AlexNet, 2020. [Online] https://en.wikipedia.org/wiki/AlexNet Último acesso a 4 de Maio de 2020.

[56] Medium (Sik-Ho Tsung). Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification), 2018. [Online] https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7 Último acesso a 4 de Maio de 2020.

[57] Leonardo Araújo dos Santos. Artificial inetlligence, 2020. [Online] https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/googlenet Último acesso a 4 de Maio de 2020.

[58] Manal El Aidouni. Evaluating Object Detection Models: Guide to Performance Metrics, 2019. [Online] https://manalelaidouni.github.io/manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html?fbclid=IwAR1TyofiYG4zTYGeOPCMFq5LSF37VgRb-0yw2lpk81eawwjcBNOmgu2Kyi4#intersection-over-union-iou Último acesso a 5 de Maio de 2020.

[59] Pyimageserach (Adrian Rosebrock). Interection over Union (IoU) for object detection, 2016. [Online] https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ Último acesso a 5 de Maio de 2020.

[60] Tensor Flow. Porque usar o Tensor Flow? [Online] https://www.tensorflow.org/ Último acesso a 5 de Maio de 2020.

[61] Wikipedia. Tensorflow, 2020. [Online] https://en.wikipedia.org/wiki/TensorFlow Último acesso a 5 de Maio de 2020.

[62] Wikipedia. Tensorflow, 2020. [Online] https://pt.wikipedia.org/wiki/TensorFlow Último acesso a 5 de Maio de 2020.

[63] Opensource. What is Python?, 2020. [Online] https://opensource.com/resources/python Último acesso a 5 de Maio de 2020.

[64] Python. What is Python? Executive Summary, 2020. [Online] https://www.python.org/doc/essays/blurb/ Último acesso a 5 de Maio de 2020.

[65] Wikipedia. Python (programming language), 2020. [Online] https://en.wikipedia.org/wiki/Python_(programming_language) Último acesso a 5 de Maio de 2020.

[66] Wikipedia. Python, 2020. [Online] https://pt.wikipedia.org/wiki/Python Último acesso a 5 de Maio de 2020.

[67] Wikipedia. Keras, 2020. [Online] https://en.wikipedia.org/wiki/Keras Último acesso a 10 de Maio de 2020.

[68] TensorFlow. Keras, 2020. [Online] https://www.tensorflow.org/guide/keras Último acesso a 10 de Maio de 2020.

[69] Guru99. Keras Tutorial for Beginners with Python: Deep Learning EXAMPLE, 2020. [Online] https://www.guru99.com/keras-tutorial.html#6 Último acesso a 10 de Maio de 2020.

[70] Machine Learning Mastery (Jason Brownlee). Your First Deep Learning Project in Python with Keras Step-By-Step, 2020. [Online] https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/ Último acesso a 10 de Maio de 2020.

[71] Wikipedia. NumPy, 2020. [Online] https://en.wikipedia.org/wiki/NumPy Último acesso a 10 de Maio de 2020.

[72] Wikipedia. Matplotlib, 2020. [Online] https://n.wikipedia.org/wiki/Matplotlib Último acesso a 10 de Maio de 2020.

[73] Wikipedia. Python Imaging Library, 2020. [Online] https://pt.wikipedia.org/wiki/Python_Imaging_Library Último acesso a 10 de Maio de 2020.

[74] Wikipedia. OpenCV, 2019. [Online] https://pt.wikipedia.org/wiki/OpenCV Último acesso a 10 de Maio de 2020.

[75] Wikipedia. Computer Vision Annotation Tool, 2020. [Online] https://en.wikipedia.org/wiki/Computer_Vision_Annotation_Tool Último acesso a 10 de Maio de 2020.

[76] GitHub. Computer Vision Annotation Tool (CVAT), 2020. [Online] https://github.com/opencv/cvat Último acesso a 10 de Maio de 2020.

[77] Medium (Internet of Wasted Things — UVA). CVAT and More Video Annotation Tools, 2019. [Online] `https://medium.com/@iowt.uva/cvat-and-more-video-annotation-tools-85e93ddcd0a4` Último acesso a 10 de Maio de 2020.