

# **Universidade da Beira Interior**

## Departamento de Informática



**Departamento de  
Informática**

**Nº 93 - 2020: *Sistema de detecção e tracking de objectos em movimento com base em análise de imagem***

Elaborado por:

**David Miguel Martins Pires**

Orientador:

**Professor Doutor Pedro Domingues de Almeida**

7 de Setembro de 2020



# ***Agradecimentos***

A conclusão deste trabalho, bem como da grande maior parte da minha vida académica não seria possível sem a ajuda da minha família, principalmente os meus pais e irmão, por todo o apoio, incentivo e educação dada, pois sem eles não teria concluído esta licenciatura.

Agradeço também ao meu orientador, Professor Pedro Domingues de Almeida por toda a orientação e ajuda providenciada durante a elaboração deste projeto.

Estes agradecimentos são extensíveis à minha namorada, amigos, colegas e docentes por toda a ajuda e incentivo dados ao longo de todo o percurso académico. Por me terem ajudado nos momentos difíceis, por me acompanharem nas noites de estudo e por todos os momentos de alegria e descontração, que carregarei para sempre.



# **Conteúdo**

<b>Conteúdo</b>	<b>iii</b>
<b>Lista de Figuras</b>	<b>v</b>
<b>Lista de Tabelas</b>	<b>vii</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Enquadramento . . . . .	1
1.2 Motivação . . . . .	2
1.3 Objetivos . . . . .	2
1.4 Organização do Documento . . . . .	3
<b>2 Estado da Arte</b>	<b>5</b>
2.1 Introdução . . . . .	5
2.2 Algoritmos de reconhecimento de objetos . . . . .	6
2.2.1 Redes Convolucionais . . . . .	7
2.2.1.1 Convolução . . . . .	7
2.2.1.2 Pooling . . . . .	9
2.2.2 Overfeat . . . . .	9
2.2.3 R-CNN . . . . .	10
2.2.4 Fast R-CNN . . . . .	11
2.2.5 Faster R-CNN . . . . .	12
2.2.6 YOLO . . . . .	13
2.2.7 SSD . . . . .	13
2.3 Algoritmos de rastreamento de objetos . . . . .	14
2.3.1 GOTURN . . . . .	16
2.3.2 ROLO . . . . .	16
2.3.3 KCF . . . . .	18
2.3.4 Centroid . . . . .	18
2.4 Conclusões . . . . .	18
<b>3 Tecnologias e Ferramentas Utilizadas</b>	<b>21</b>
3.1 Introdução . . . . .	21

3.2	Hardware . . . . .	21
3.2.1	Raspberry Pi . . . . .	22
3.2.2	Câmera . . . . .	23
3.3	Software . . . . .	24
3.3.1	Raspberry Pi OS . . . . .	24
3.3.2	Python . . . . .	25
3.3.3	OpenCV . . . . .	25
3.3.4	TensorFlow . . . . .	26
3.4	Conclusões . . . . .	26
<b>4</b>	<b>Implementação e Testes</b>	<b>27</b>
4.1	Introdução . . . . .	27
4.2	Estudo e investigação . . . . .	27
4.2.1	Reconhecimento de Objetos . . . . .	28
4.2.1.1	MobileNet . . . . .	28
4.2.1.2	SSD . . . . .	29
4.2.1.3	Mobilenet-SSD . . . . .	30
4.2.2	Rastreamento de objetos . . . . .	31
4.2.2.1	MOSSE . . . . .	31
4.2.2.2	Pré-processamento . . . . .	31
4.2.2.3	Alvo sintético . . . . .	32
4.2.2.4	Etapas do rastreador . . . . .	32
4.3	Programa realizado . . . . .	34
4.4	Testes finais . . . . .	40
4.5	Conclusões . . . . .	42
<b>5</b>	<b>Conclusões e Trabalho Futuro</b>	<b>43</b>
5.1	Conclusões Principais . . . . .	43
5.2	Trabalho Futuro . . . . .	44
<b>Bibliografia</b>		<b>45</b>

# ***Lista de Figuras***

2.1	Ilustração da classificação e localização de um objeto numa imagem.	6
2.2	Exemplo de uma convolução numa imagem.	8
2.3	Efeitos de uma convolução com diferentes núcleos numa imagem.	8
2.4	Convolução em volume.	9
2.5	Método de <i>pooling</i> utilizando a função máximo e núcleo $2 \times 2$ .	10
2.6	Ilustração de uma janela deslizante (esquerda) com diferentes proporções (direita).	10
2.7	A arquitetura padrão de uma rede <i>Region-based Convolutional Neural Network</i> (R-CNN) consiste num método de propostas regionais. A classificação final é feita com uma <i>Support Vector Machine</i> (SVM) e uma regressão. Esta imagem foi retirada de [8].	11
2.8	A arquitetura padrão da rede Fast-R-CNN. As propostas regionais são geradas usando pesquisa seletiva, mas agrupadas diretamente num mapa de características, seguindo de várias camadas totalmente conectadas para a classificação final e regressão da caixa delimitadora. Esta imagem foi retirada de [8].	12
2.9	A arquitetura padrão da rede Faster-R-CNN, onde as propostas de região são geradas usando uma <i>Region Proposal Network</i> (RPN), que trabalha directamente com o mapa de características. As últimas camadas estão totalmente conectadas para a classificação e regressão das caixas delimitadoras. Esta imagem foi retirada de [8].	12
2.10	Exemplo do funcionamento da arquitetura <i>You Only Look Once</i> (YOLO).	13
2.11	Exemplo do funcionamento da arquitetura <i>Recurrent YOLO</i> (ROLO)	17
3.1	Fotografia do <i>Raspberry Pi</i> utilizado.	22
3.2	Fotografia da câmara utilizada.	24
4.1	Arquitetura do modelo <i>Single Shot Detector</i> (SSD).	29
4.2	Exemplo das camadas finais de uma rede <i>MobileNet</i> .	30

4.3	Exemplo das camadas finais de uma rede <i>MobileNet</i> alimentando uma rede SSD. . . . .	30
4.4	Cenário onde foram realizados os testes finais. . . . .	41

# ***Listas de Tabelas***

3.1 Tabela das características do <i>Raspberry Pi</i> utilizado. . . . .	23
--	----



# ***Listas de Excertos de Código***

3.1	Comandos utilizados para atualizar pacotes no sistema operativo <i>Raspberry Pi OS</i> . . . . .	25
4.1	Inicialização do <i>TensorFlow Lite</i> . . . . .	34
4.2	Alocar <i>threads</i> para o <i>OpenCV</i> . . . . .	34
4.3	Classe dedicada ao trabalho da câmara . . . . .	34
4.4	Função de recuperação de objetos perdidos . . . . .	36
4.5	Funcionalidades da deteção de objetos . . . . .	38



# **Acrónimos**

<b>API</b>	<i>Application Programming Interface</i>
<b>CNN</b>	<i>Convolutional Neural Network</i>
<b>COCO</b>	<i>Common Objects In Context</i>
<b>FPS</b>	<i>Frames Per Second</i>
<b>GOTURN</b>	<i>Generic Object Tracking Using Regression Networks</i>
<b>HDMI</b>	<i>High-Definition Multimedia Interface</i>
<b>IA</b>	<i>Inteligência Artificial</i>
<b>ID</b>	<i>Identification (Identificação)</i>
<b>KCF</b>	<i>Kernelized Correlation Filters</i>
<b>LSTM</b>	<i>Long Short Term Memory</i>
<b>MOSSE</b>	<i>Minimum Output Sum of Squared Error</i>
<b>OpenCV</b>	<i>Open Source Computer Vision Library</i>
<b>R-CNN</b>	<i>Region-based Convolutional Neural Network</i>
<b>RGB</b>	<i>Red Green Blue</i>
<b>RoI</b>	<i>Region of Interest</i>
<b>ROLO</b>	<i>Recurrent YOLO</i>
<b>RPN</b>	<i>Region Proposal Network</i>
<b>SSD</b>	<i>Single Shot Detector</i>
<b>SSH</b>	<i>Secure Socket Shell</i>
<b>SVM</b>	<i>Support Vector Machine</i>
<b>VNC</b>	<i>Virtual Network Computing</i>

**USB**      *Universal Serial Bus*

**YOLO**      *You Only Look Once*

## *Capítulo*

# 1

## *Introdução*

### **1.1 Enquadramento**

Este projeto, titulado de sistema de deteção e tracking de objetos em movimento com base em análise de imagem, impõe duas tarefas principais, o reconhecimento de objetos e o rastreamento de objetos. Ambas as tarefas fazem parte da área da Visão Computacional.

Estas tarefas consistem a encontrar objetos num fluxo de vídeo, identificar esses objetos e reconhecer esses objetos (rastreamento). Este será um fluxo de vídeo em tempo real, ou seja, serão realizadas estas tarefas ao mesmo tempo que serão capturadas as imagens de uma câmara *Universal Serial Bus* (USB). Esta será uma câmara de vídeo comum, e as imagens capturadas serão num ambiente de luz natural.

Nós humanos, conseguimos reconhecer múltiplos objetos em diferentes imagens, e muitas vezes objetos obstruídos por outros objetos, em diferentes disposições e segundos, sem qualquer dificuldade. Isto para a Visão Computacional é ainda um desafio, e é muito mais complexo reconhecer um objeto em uma imagem do que pensamos à primeira vista.

A acrescentar a estas tarefas, todos os programas desenvolvidos neste projeto, deverão ser executados num pequeno computador, chamado *Raspberry Pi*. Este é um computador com baixa capacidade de processamento, o que acrescenta um trabalho de otimização e eficiência. Isto acrescenta a este projeto um grande desafio de obter a melhor relação entre:

- A qualidade na deteção e rastreamento de um objeto;

- A velocidade com que realiza esta tarefa.

Esta situação acontece pois partimos do princípio em que uma maior qualidade na deteção e rastreamento de um objeto, envolve uma maior capacidade de processamento.

## **1.2 Motivação**

O tema deste projeto é um tema atual e muito importante para a área de Visão Computacional e de outras áreas como a da Robótica. Estas são também áreas em constante crescimento e que apresentam ter um grande futuro, daí ser uma grande motivação na escolha deste projeto.

Mais especificamente, o enquadramento deste projeto, as tarefas e os desafios que este projeto envolve, são uma grande motivação. São tarefas que desde há algum tempo tenho bastante interesse em aprender.

O equipamento que este projeto envolve (Raspberry Pi), apenas acrescenta o desafio de conseguir realizar estas tarefas de forma eficiente/otimizada, o que acrescenta ainda mais a motivação para a escolha deste projeto. O Raspberry Pi é um computador ligado ao ensino, e é mais presente nas áreas da robótica.

É também uma grande motivação na escolha deste projeto o Professor orientador, Pedro Almeida.

## **1.3 Objetivos**

O objetivo deste trabalho é o estudo e aprendizagem das principais tecnologias já existentes capazes de lidar com estas tarefas, como o *Tensorflow*, *Yolo* e *OpenCV*. Com isto o principal objetivo é o desenvolvimento de um código na linguagem *Python* que implemente algumas das ferramentas estudadas.

As bibliotecas utilizadas terão de ser adaptadas, bem como o código desenvolvido, para que possa ser executado num computador com baixa capacidade computacional.

Também será usada uma câmara comum através de uma porta USB.

## 1.4 Organização do Documento

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

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objetivos e a respetiva organização do documento.
2. O segundo capítulo – **Estado da arte** – apresenta o estado atual do reconhecimento de objetos e o seu funcionamento. Também é descrito e explicado os algoritmos estudados.
3. O terceiro capítulo – **Tecnologias Utilizadas** – descreve os conceitos mais importantes no âmbito deste projeto, bem como as tecnologias utilizadas durante o desenvolvimento do projeto.
4. O quarto capítulo – **Implementação e Testes** – apresenta o estudo e investigação feitos ao longo do desenvolvimento deste projeto. Descreve também o programa desenvolvido neste projeto. Por fim são apresentados e descritos os testes finais.
5. O quinto capítulo – **Conclusões e Trabalho Futuro** – apresenta as conclusões do projeto, bem como o trabalho futuro.



## *Capítulo*

# 2

## ***Estado da Arte***

### **2.1 Introdução**

O reconhecimento e rastreamento de objetos é um dos campos mais atuais da visão computacional.

É possível reconhecer diferentes objetos numa cena através das suas formas, cor e diferentes características do objeto. É também possível rastrear um objeto numa sucessão de imagens de vídeo através do cálculo de distâncias, comparação de formas, e outras características.

Neste capítulo será abordado o estado atual do reconhecimento e rastreamento de objetos, e os diferentes algoritmos que existem para o fazer. Isto com especial atenção ao meio de desenvolvimento que será um *Raspberry Pi*, um pequeno computador com capacidades limitadas.

**Este capítulo está estruturado da seguinte forma:**

1. Secção 2.2 - **Algoritmos de reconhecimento de objetos** - Esta secção apresenta vários algoritmos estudados sobre a deteção/reconhecimento de objetos e uma breve explicação do seu funcionamento.
2. Secção 2.3 - **Algoritmos de rastreamento de objetos** - Esta secção apresenta vários algoritmos estudados sobre o rastreamento de objetos, e uma breve explicação do seu funcionamento.
3. Secção 2.4 - **Conclusões** - Esta secção apresentará uma breve conclusão sobre este capítulo.

## 2.2 Algoritmos de reconhecimento de objetos

A deteção de objetos combina as tarefas de localização e classificação de objetos. Corresponde a localizar um objeto numa imagem e adivinhar que objeto é esse, como vemos na figura 2.1. Para além de estimar a classe de um ou vários objetos presentes numa imagem também preciso estimar as caixas delimitadoras que contém estes objetos.

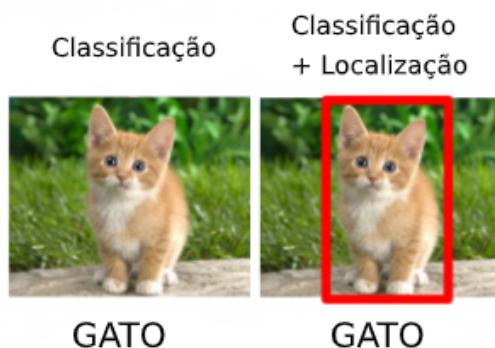


Figura 2.1: Ilustração da classificação e localização de um objeto numa imagem.

São necessárias 4 variáveis para identificar exclusivamente um retângulo. Portanto, para cada instância de um objeto numa imagem, será necessário calcular as seguintes variáveis:

- nome\_da\_classe;
- caixa\_delimitadora\_topo\_esquerda\_coordenada\_x;
- caixa\_delimitadora\_topo\_esquerda\_coordenada\_y;
- caixa\_delimitadora\_altura;
- caixa\_delimitadora\_largura.

Nas subsecções abaixo irei explicar um pouco mais sobre os conceitos mais comuns dos algoritmos de reconhecimento de objetos, como é o caso do *Convolutional Neural Network* (CNN).

**CNN** é uma rede neuronal que consiste em várias camadas diferentes, como a camada de entrada (*input*), pelo menos uma camada escondida e uma camada de saída (*output*). Esta rede é usada para reconhecer várias características numa imagem como é o caso de padrões, formas, cores, texturas

e outras.

Irei também mencionar alguns dos algoritmos de deteção de objetos, focando-me nos de ponta, que usam Redes Neuronais e Aprendizagem Profunda.

Os algoritmos de deteção de objetos atuais podem ser divididos em duas categorias:

- Redes que separam as tarefas de determinar a localização dos objetos e sua classificação, onde *Faster Region-based Convolutional Neural Network* (R-CNN) é uma das mais famosas;
- Redes que prevêem caixas delimitadoras e classificações de classes de uma só vez, onde *You Only Look Once* (YOLO) e *Single Shot Detector* (SSD) são as mais famosas.

## 2.2.1 Redes Convolucionais

Como mencionado, neste capítulo irei explicar os conceitos necessários para a compreensão destas redes. Estas redes são também chamadas de CNN.

### 2.2.1.1 Convolução

A convolução de imagens é uma operação matemática que consiste no produto interno entre a matriz dos pixels da imagem e uma matriz núcleo. O núcleo percorre toda a imagem de entrada e cada produto interno, obtido da sobreposição de uma porção da imagem e o núcleo, torna-se um pixel na imagem de saída, como podemos ver na figura 2.2.

Como podemos ver na figura 2.2, o núcleo é uma matriz  $3 \times 3$  e cada pixel do resultado é o resultado do produto interno de uma porção correspondente a uma matriz  $3 \times 3$  da imagem de entrada e o núcleo. A matriz resultado é também chamada de **mapa de características**.

Núcleos diferentes produzem um efeito diferente na imagem de saída e realçam características diferentes da imagem de entrada. Podemos ver na figura 2.3 o efeito de diferentes núcleos numa imagem.

É comum trabalharmos com imagens coloridas, no formato *Red Green Blue* (RGB). Neste caso imagem de entrada corresponde não só a uma matriz, mas sim três, uma para cada cor fundamental (Vermelho, Verde e Azul). Na figura 2.4 podemos ver o processo de convolução de uma imagem de tamanho  $6 \times 6$  pixels. Pode-se interpretar a imagem como um cubo de dimensões

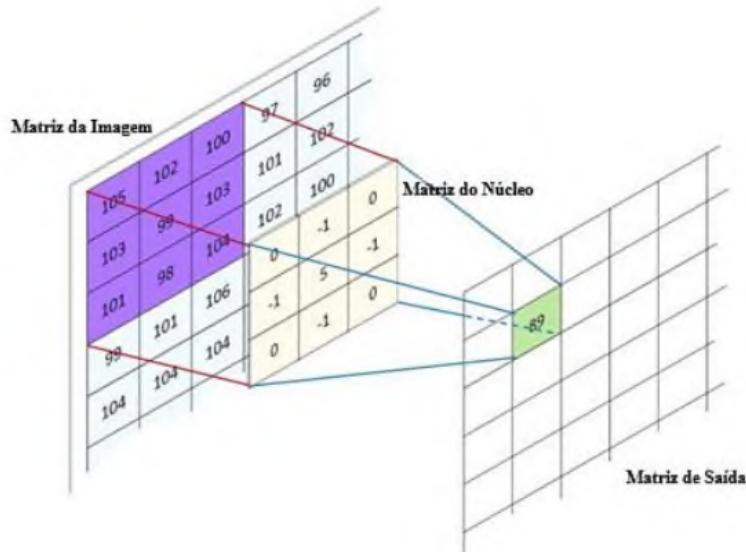


Figura 2.2: Exemplo de uma convolução numa imagem.



Figura 2.3: Efeitos de uma convolução com diferentes núcleos numa imagem.

$h \times w \times c$ , onde  $h$  é a altura da imagem em pixeis,  $w$  é a largura da imagem em pixeis e  $c$  é o numero de canais da imagem (que neste caso são 3).

Nesta imagem podemos observar que a sobreposição do núcleo com a primeira porção da imagem de entrada, gerará o primeiro pixel da imagem de saída. Podemos também observar que o processo de convolução além de des-

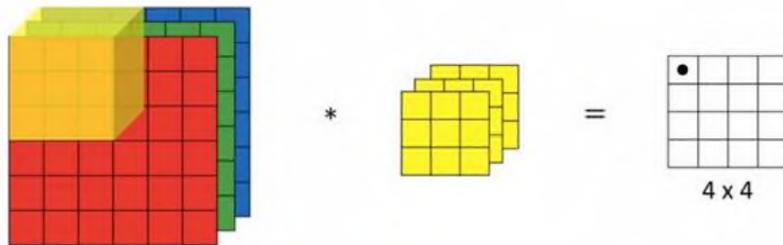


Figura 2.4: Convolução em volume.

tacar características da imagem, reduz a quantidade de pixels e consequentemente a quantidade de dados na rede.

### 2.2.1.2 Pooling

Ao processar grandes imagens com muitos pixels, é comum reduzir o numero de neurónios no sistema, com o intuito de diminuir o tempo de processamento, o numero de parâmetros a serem aprendidos pela rede e o poder de processamento requerido. Para isto utilizamos métodos *pooling*, também chamados de agregação.

A forma como o *pooling* funciona é muito parecida com a convolução, pois também percorre a imagem de entrada em grupos de neurónios, porem a principal diferença é que os neurónios de saída aqui são resultado de uma função de *pooling*, que pode ser máximo, mínimo, média ou outras.

A figura 2.5 demonstra o processo de *pooling* utilizando o método de máximo num núcleo  $2 \times 2$  numa imagem de entrada de 16 neurónios. Podemos ver que o núcleo percorre a imagem de entrada andando de 2 em 2 neurónios e seleccionando o valor máximo da área como valor do neurónio de saída.

### 2.2.2 Overfeat

A primeira rede neuronal profunda para deteção de objetos foi **Overfeat**. Esta introduziu uma abordagem de janela deslizante de várias proporções, como mostra na figura 2.6, e usando uma CNN, mostraram que a deteção de objetos também melhorou a classificação de imagens.

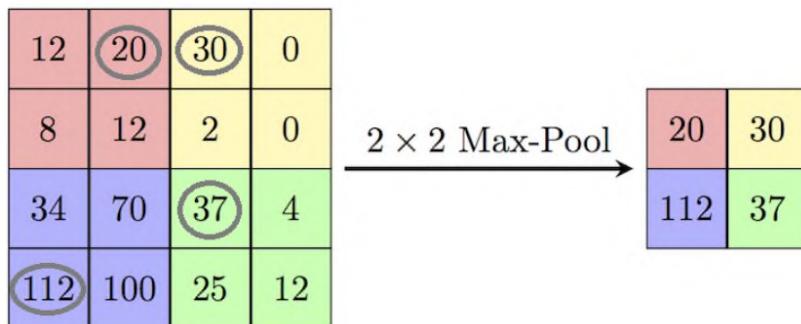


Figura 2.5: Método de *pooling* utilizando a função máximo e núcleo  $2 \times 2$ .

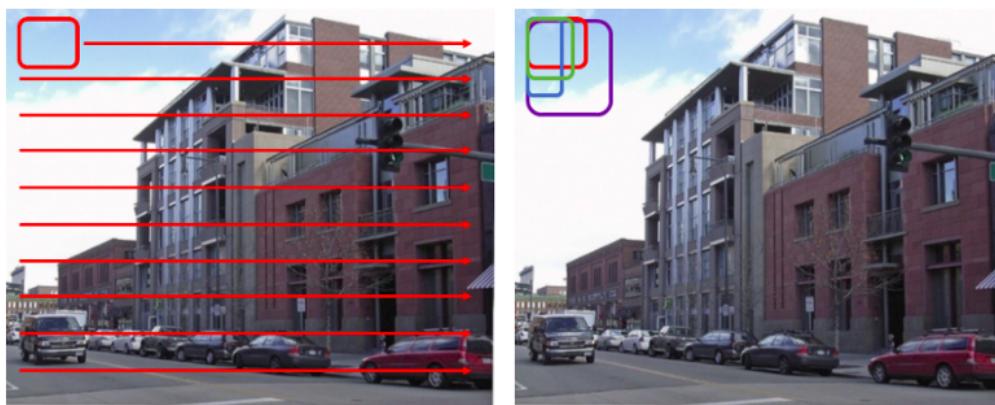


Figura 2.6: Ilustração de uma janela deslizante (esquerda) com diferentes proporções (direita).

### 2.2.3 R-CNN

O modelo anterior foi logo seguido pelo modelo **R-CNN**: Regiões com características CNN. As CNNs são demasiado lentas e dispendiosas, por isso é impossível correr uma CNN em tantos caminhos gerados por uma janela deslizante.

A R-CNN resolve este problema usando um algoritmo de propostas de regiões de objetos chamado de **Pesquisa seletiva** que reduz o número de caixas a serem alimentadas no classificador. A Pesquisa seletiva usa dicas locais como textura, intensidade, cor e/ou a medida de incidências, etc, para gerar todas as propostas de localização do objeto. Cada região é alimentada numa CNN, que produz um vetor de características de grande dimensão. Este vetor é depois usado para a classificação final e regressão da caixa delimitadora,

como mostra na figura 2.7.

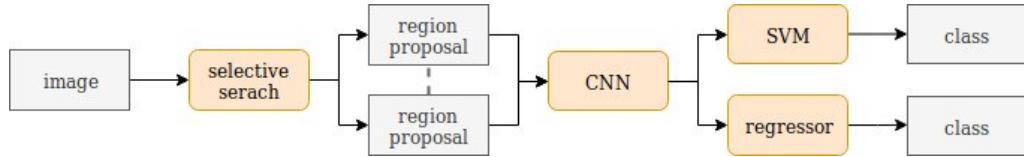


Figura 2.7: A arquitetura padrão de uma rede R-CNN consiste num método de propostas regionais. A classificação final é feita com uma SVM e uma regressão.

Esta imagem foi retirada de [8].

**SVM** é um algoritmo de aprendizagem supervisionada, em que o objetivo é classificar um determinado conjunto de pontos de dados que são mapeados para um espaço de características multidimensional.

Existem 3 importantes etapas da R-CNN:

1. Realizar pesquisa seletiva para gerar prováveis objetos.
2. Alimentar esses caminhos numa CNN, seguidos de uma SVM para prever a classe de cada caminho.
3. Optimizar os caminhos, treinando as caixas delimitadoras separadamente.

#### 2.2.4 Fast R-CNN

Uma abordagem mais sofisticada, o **Fast R-CNN** também gera proposições regionais com pesquisa seletiva, mas alimenta a imagem inteira através de uma CNN para gerar um mapa de características.

Esta supera o desempenho da rede *Overfeat* por uma grande margem, mas ainda assim é muito lenta, porque a geração de propostas regionais usando pesquisa seletiva é muito demorada, tal como a necessidade de alimentar cada proposta através de uma CNN.

As propostas regionais são agrupadas directamente num mapa de características através do método *Region of Interest (RoI) pooling*. Os vetores característicos agrupados são alimentados numa rede totalmente conectada para classificação e regressão, como retratado na figura 2.8.

**RoI pooling**, ou agrupamento da Região de Interesse é o processo de converter uma região de interesse da imagem original numa imagem de tamanho fixo para que se possa passar à próxima etapa da deteção do objeto.

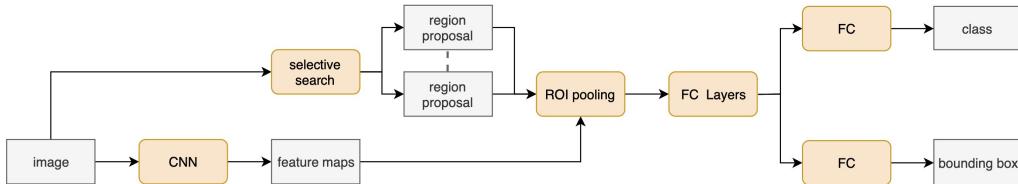


Figura 2.8: A arquitetura padrão da rede Fast-R-CNN. As propostas regionais são geradas usando pesquisa seletiva, mas agrupadas diretamente num mapa de características, seguindo de várias camadas totalmente conectadas para a classificação final e regressão da caixa delimitadora.

Esta imagem foi retirada de [8].

## 2.2.5 Faster R-CNN

A parte mais lenta dos algoritmos anteriores é a pesquisa selectiva. Então a arquitetura **Faster R-CNN** substitui a pesquisa seletiva por uma pequena rede convolucional chamada rede de propostas por região (*Region Proposal Network* (RPN)) para gerar regiões de interesse. A restante rede é semelhante à arquitetura Fast R-CNN.

O objetivo da RPN é gerar um conjunto de propostas de caixas delimitadoras, cada uma probabilidade de ser um objeto e também a classe. A RPN pode receber imagens de entradas de qualquer tamanho para realizar essa tarefa.

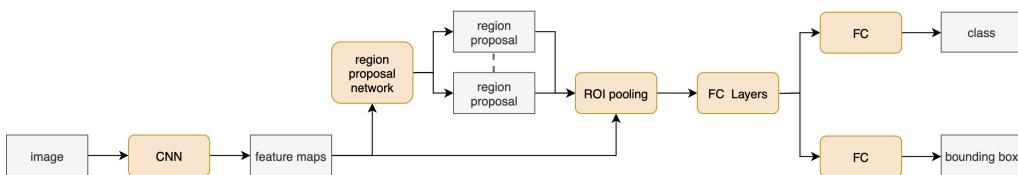


Figura 2.9: A arquitetura padrão da rede Faster-R-CNN, onde as propostas de região são geradas usando uma RPN, que trabalha directamente com o mapa de características. As últimas camadas estão totalmente conectadas para a classificação e regressão das caixas delimitadoras.

Esta imagem foi retirada de [8].

### 2.2.6 YOLO

O design das redes de deteção de objetos foi revolucionado pela rede **YOLO**. Esta segue uma abordagem completamente diferente à dos algoritmos mencionados anteriormente e é capaz de prever classificações de classes e caixas delimitadoras de uma vez.

Este modelo divide a imagem numa grelha  $S \times S$ , e cada célula prevê  $N$  caixas delimitadoras e a sua taxa confiança. A confiança reflete a precisão da caixa delimitadora e se a caixa delimitadora realmente contém o objeto (independente da classe).

Portanto, são previstas  $S \times S \times N$  caixas, no entanto, muitas dessas caixas tem pontuações baixas de confiança e se definirmos um limite, digamos que a 30% de confiança, podemos remover a maioria delas, como é mostrado na figura 2.10.

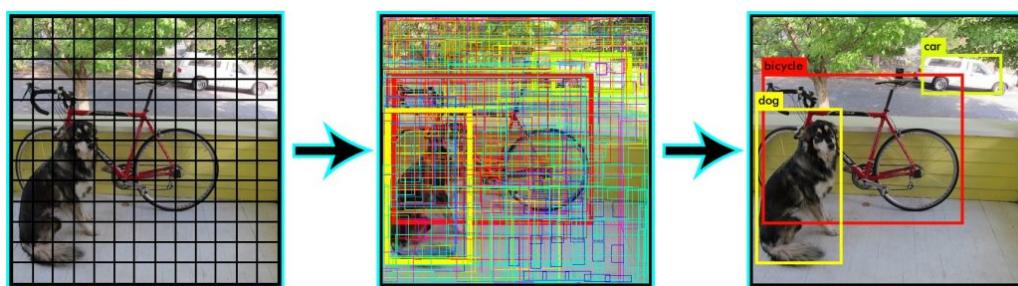


Figura 2.10: Exemplo do funcionamento da arquitetura YOLO.

Os autores anunciaram mais três versões desta arquitetura:

- YOLO9000, que é capaz de prever até 9000 categorias de objetos;
- YOLOv2, que é capaz de processar imagens maiores;
- YOLOv3, que é mais preciso na deteção.

### 2.2.7 SSD

Outra rede que prevê classes e caixas delimitadoras de uma vez é a **SSD**. Esta executa uma rede convolucional em apenas uma imagem de entrada e calcula um mapa de recursos. Aqui executamos um pequeno núcleo convolucional de tamanho  $3 \times 3$  para prever as caixas delimitadoras e a taxa de confiança da

classificação.

Para lidar com a escala, a rede SSD prevê caixas delimitadoras após várias camadas convolucionais. Como cada camada convolucional opera numa diferente escala, esta é capaz de detetar objetos de várias escalas.

É comparável com a rede YOLO, mas esta usa várias proporções por cada célula da grelha e mais camadas convolucionais para melhorar a precisão.

## 2.3 Algoritmos de rastreamento de objetos

O rastreamento de objetos é o processo de identificar objetos detetados num vídeo ao longo do tempo. Este é um problema antigo e difícil da visão computacional.

Existem várias técnicas e algoritmos que tentam resolver esse problema de várias maneiras diferentes. No entanto, a maioria baseia-se em dois pontos principais:

- **Modelo de movimento**

Um dos principais componentes de um bom rastreador é a capacidade de entender e modelar o movimento do objeto. Assim, é desenvolvido um modelo de movimento que captura o movimento dinâmico de um objeto. Este prevê a posição potencial dos objetos nos quadros de vídeo (imagens de um vídeo) seguintes, reduzindo assim o espaço de pesquisa. No entanto, o modelo de movimento só por si pode falhar em cenários em que o movimento é causado por coisas que não estão no vídeo ou direcção abrupta e mudança de velocidade.

- **Modelo de Aparência Visual**

A maioria dos rastreadores mais precisos precisam perceber a aparência do objeto que estão a rastrear e precisam aprender a discriminar o objeto do plano de fundo. Nos rastreadores de um único objeto, apenas a aparência visual pode ser suficiente para rastrear o objeto ao longo dos quadros de vídeo, enquanto nos rastreadores de vários objetos, apenas a aparência visual não é suficiente.

Existem várias características para classificar um rastreador de objetos, como por exemplo:

- **Rastreadores com base na deteção**

Os quadros de vídeo consecutivos são dados a um detetor de objetos pré

treinado que fornece uma hipótese de deteção, que depois é usada para formar trajetórias de rastreamento. Novos objetos são detetados e objetos que desaparecem são finalizados automaticamente. Nesta abordagem, o rastreador é usado para os casos em que a deteção de objetos falha. Em outra abordagem, o detetor de objetos é executado para nada  $N$  quadros de vídeo e as previsões restantes são feitas usando o rastreador.

- **Rastreamento sem deteção**

O rastreamento sem deteção requer inicialização manual de um número fixo de objetos no primeiro quadro de vídeo. De seguida, localiza esses objetos nos quadros de vídeo subsequentes. Esta abordagem não pode lidar com o caso em que novos objetos aparecem nos quadros de vídeo subsequentes.

- **Acompanhamento de um único objeto**

Apenas um único objeto é rastreado, mesmo que o ambiente tenha vários objetos. O objeto a ser rastreado é determinado pela inicialização do primeiro quadro de vídeo.

- **Rastreamento de Objetos múltiplos**

Todos os objetos presentes são rastreados ao longo do tempo. Se um rastreador baseado em deteção for usado, este poderá rastrear novos objetos que apareçam no meio do vídeo.

- **Rastreamento offline**

Os rastreadores offline são usados quando se precisa rastrear um objeto numa sucessão de imagens de vídeo gravadas. Neste caso, podemos não apenas usar os quadros de vídeo passados, mas também os futuros, para fazer previsões de rastreamento mais precisas.

- **Rastreamento online**

Rastreadores online são usados onde as previsões estão disponíveis imediatamente, e portanto, estes não podem usar quadros de vídeo futuros para melhorar os resultados.

- **Rastreamento de aprendizagem online**

Estes rastreadores geralmente aprendem sobre o objeto a ser rastreado usando o quadro de vídeo de inicialização e alguns subsequentes. Portanto estes rastreadores são mais gerais, porque podemos simplesmente desenhar uma caixa delimitadora em torno de um objeto e rastreá-lo.

- **Rastreamento de aprendizagem offline**

O treinamento destes rastreadores acontece apenas offline. Ao contrário dos rastreadores de aprendizagem online, estes rastreadores não aprendem nada durante o tempo de execução. Estes rastreadores aprendem sobre os conceitos completos offline, ou seja, podemos treinar um rastreador para identificar um objeto e este poderá ser usado para rastrear continuamente todas as instâncias desse objeto num fluxo de vídeo.

Muitos algoritmos de rastreamento tradicionais (não baseados em aprendizagem profunda) estão integrados na *Application Programming Interface* (API) do *Open Source Computer Vision Library* (OpenCV). A maioria destes algoritmos não é muito precisa comparativamente com os restantes algoritmos baseados em Aprendizagem Profunda. No entanto estes podem vir a ser úteis para executar num ambiente com restrições na capacidade de processamento, como é o caso do Raspberry Pi. No entanto, os rastreadores baseados em aprendizagem profunda estão agora muito à frente dos rastreadores tradicionais em termos de precisão.

Nos pontos a seguir serão mencionados alguns dos algoritmos mais populares no rastreamento de objetos, baseados em *Inteligência Artificial* (IA), como é o caso do *Generic Object Tracking Using Regression Networks* (GOTURN) e do *Recurrent YOLO* (ROLO), bem como dos tradicionais, como é o caso do *Kernelized Correlation Filters* (KCF) e do *Centroid*.

### 2.3.1 GOTURN

O **GOTURN** é um rastreador de aprendizagem offline com base numa CNN. Este rastreador é treinado usando vídeos de rastreamento geral de objetos. Este rastreador pode ser usado para rastrear a maioria dos objetos sem nenhum problema, mesmo que esses objetos não fizessem parte do conjunto de objetos com que foi treinado. Este algoritmo está integrado na biblioteca OpenCV.

### 2.3.2 ROLO

O **ROLO** é um algoritmo de rastreamento baseado em deteção, online, e com um único objeto. Este usa a rede YOLO para deteção de objetos e um rede *Long Short Term Memory* (LSTM) para encontrar a trajectória do objeto de destino. Há duas razões pelas quais LSTM com CNN é uma excelente combinação:

- As redes LSTM são particularmente boas em aprender padrões históricos, sendo particularmente adequadas para o rastreamento de objetos visuais.
- As redes LSTM não são muito caras em termos computacionais, por isso é possível criar rastreadores do mundo real muito rápidos.

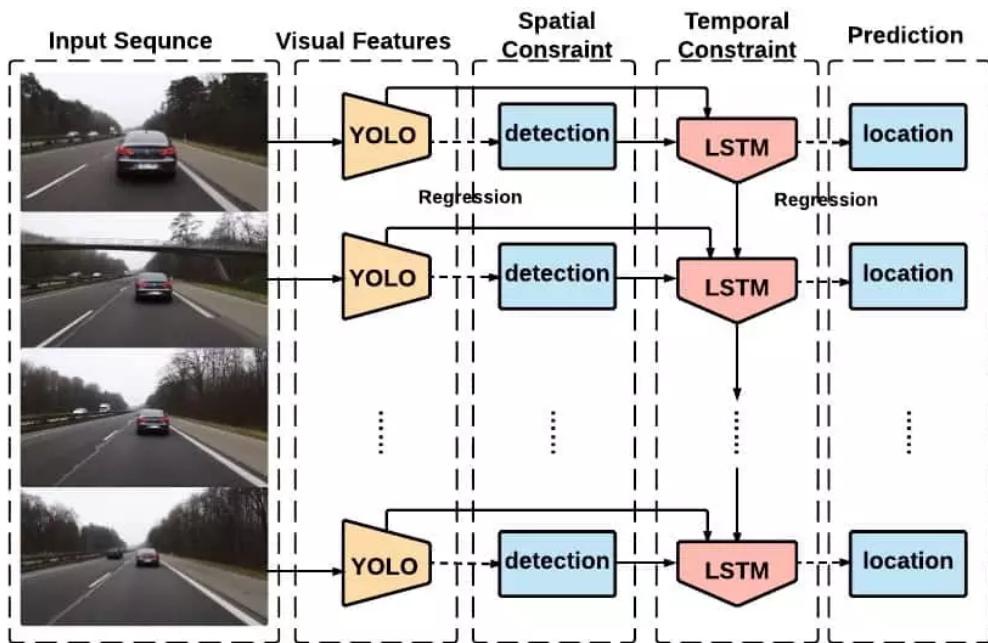


Figura 2.11: Exemplo do funcionamento da arquitetura ROLO

A figura 2.11 fornece-nos o seguinte entendimento.

- **YOLO INPUT:** Quadros de vídeo de entrada;
- **YOLO OUTPUT:** Vetor de recursos dos quadros de vídeo de entrada e coordenadas da caixa delimitadora;
- **LSTM INPUT:** Junção dos recursos da imagem e coordenadas da caixa delimitadora;
- **LSTM OUTPUT:** Coordenadas da caixa delimitadora do objeto a ser rastreado.

Os quadros de vídeo de entrada passam pela rede YOLO. Nesta rede, são obtidas duas saídas diferentes (Recursos de imagem e coordenadas da caixa

delimitadora). Estas duas saídas são fornecidas à rede LSTM. O LSTM gera trajetórias, isto é, a caixa delimitadora do objeto a ser rastreado. A inferência preliminar de localização (do YOLO) ajuda o LSTM a prestar atenção a certos elementos visuais. O ROLO explora a história espaço-temporal, ou seja, juntamente com o histórico de localização, o ROLO também explora a história dos recursos visuais. Mesmo quando a deteção do YOLO falha devido a desfoque de movimento, o rastreamento de ROLO permanece estável.

### 2.3.3 KCF

A ideia básica do **KCF** é estimar um filtro de imagem ideal, de modo a que a filtragem com a imagem de entrada produza a resposta desejada. Dado um conjunto inicial de pontos, o KCF tenta calcular o movimento destes pontos observando a direcção da mudança no próximo frame. Por isso este é um rastreador offline. Em cada frame consecutivo, tentamos procurar o mesmo conjunto de pontos na vizinhança. Depois que as novas posições desses pontos são identificadas, podemos mover a caixa delimitadora sobre o novo conjunto de pontos. Há matemática envolvida para tornar a pesquisa mais rápida e eficiente.

### 2.3.4 Centroid

O **rastreamento de centroides** é um algoritmo presente na biblioteca OpenCV, que depende da distância euclidiana<sup>1</sup> entre centroides de objetos (objetos que o rastreador de centroide já viu antes) e novos centroides de objetos entre quadros de vídeo subsequentes. Neste algoritmo temos de usar um detetor de objetos em cada frame de vídeo e determinar os centroides de cada objeto. Assumimos aqui que no próximo frame o centroide do objeto estará a uma curta distância no centroide no frame anterior, e assim assumimos que seja o mesmo objeto através da distância. Este algoritmo não é muito eficiente em ambientes com mais que um objeto.

## 2.4 Conclusões

Neste capítulo foram descritos alguns dos algoritmos de deteção e rastreamento mais populares. Podemos concluir que apesar da deteção e rastreamento de objetos ser uma ser ainda uma temática bastante recente teve já um grande avanço. Apesar da sua dificuldade e do seu nível computacional,

---

<sup>1</sup>Distância entre dois pontos

cada vez existem mais algoritmos para melhorarem e tornarem o reconhecimento e rastreamento de objetos mais preciso e rápido.

Existem inúmeros algoritmos de detetores de objetos em imagens e de rastreadores de objetos em imagem. Em este capítulo foquei-me apenas em alguns dos mais populares e em versões mais atuais, sendo a maioria baseadas em Aprendizagem Profunda e Redes Neuronais.



## *Capítulo*

# 3

## ***Tecnologias e Ferramentas Utilizadas***

### **3.1 Introdução**

Para o desenvolvimento deste projecto foram utilizadas diversas tecnologias e ferramentas. Devido à complexidade do reconhecimento e rastreamento de objetos, é necessário recorrer a várias ferramentas que nos ajudam a obter estes resultados. Este projeto foca-se na implementação destas ferramentas em hardware com baixa capacidade computacional. Neste capítulo vamos falar sobre estas tecnologias e ferramentas, nomeadamente do Hardware e Software utilizado.

**Este capítulo está estruturado da seguinte forma:**

1. Secção 3.2 - **Hardware** - Esta secção apresenta todo o hardware utilizado neste projeto.
2. Secção 3.3 - **Software** - Esta secção apresenta todo o software utilizado no reconhecimento e rastreamento de objetos.
3. Secção 3.4 - **Conclusões** - Esta secção apresentará uma breve conclusão sobre este capítulo.

### **3.2 Hardware**

Para este projeto foram utilizados vários equipamentos de Hardware, porém vou apenas mencionar os principais, que é um *Raspberry Pi* e uma câmara

USB. Foram também utilizadas mais tecnologias ou ferramentas, para o manuseamento e iteração com o *Raspberry Pi*, como o rato, teclado, ecrã, porem estas poderiam ser quaisquer uma.

### 3.2.1 Raspberry Pi

O *Raspberry Pi* é um computador de tamanho reduzido. Este apresenta todo o seu hardware integrado numa única placa, como mostra na imagem 3.1, que mostra o *Raspberry Pi* usado. Podemos também notar na imagem que foi usada uma caixa com duas ventoinhas para uma melhor dissipação de calor. O principal objetivo deste computador é promover o ensino na Ciência da Computação.



Figura 3.1: Fotografia do *Raspberry Pi* utilizado.

Para a realização deste projeto é nos pedido a elaboração do mesmo num *Raspberry Pi* e eu escolhi utilizar o ***Raspberry Pi 4 Model B 4Gb***. Este pequeno computador conta com o hardware disposto na tabela 3.1.

	<b>Hardware</b>
Tipo Chip	Broadcom BCM2711
Tipo Core	Cortex-A72 (ARM v8) 64-bit
Nº de cores	4
Clock CPU	1.5GHz
GPU	VideoCore VI
RAM	4GB

Tabela 3.1: Tabela das características do *Raspberry Pi* utilizado.

Foi utilizado também, junto deste *Raspberry Pi*, um cartão *microSD* de 64GB da *SanDisk* com velocidades de leitura de 160MB/s e de escrita de 90MB/s, onde foi instalado todo o software utilizado e desenvolvido este projeto.

Este equipamento foi adquirido por mim, no inicio da realização deste projeto, devido ao meu interesse no aprendizado das funcionalidades e capacidades deste equipamento. Foi também disponibilizado pelo professor um *Raspberry Pi 3*, porém decidi adquirir e utilizar esta versão mais recente deste computador.

O manuseamento e iteração com este computador pode ser feito de várias formas. Durante a realização deste trabalho, eu fiz isto de duas formas:

- Através de um rato, teclado e monitor comum. Estes podem ser quais queres uns, que liguem ao *Raspberry Pi* através de portas USB e *Micro High-Definition Multimedia Interface* (HDMI) (No caso do *Raspberry Pi 4*).
- Através de um cabo *Ethernet*, ligado a outro computador. Este pode ser feito através de uma conexão *Secure Socket Shell* (SSH) e o serviço *Virtual Network Computing* (VNC), com o *Raspberry Pi*. Para isto segui um tutorial presente na página oficial do *Raspberry Pi* [9].

### 3.2.2 Câmara

Sendo um dos objetos deste trabalho a deteção e rastreamento de objetos em tempo real, é necessário a utilização de uma câmara onde pudesse obter imagens num fluxo de vídeo em tempo real para serem analisadas e processadas. Esta tarefa poderá ser feita utilizando uma câmara qualquer conectada através de USB ao *Raspberry Pi*.

Para este efeito, utilizei a câmara **Logitech Webcam C270 HD** presente na imagem 3.2, por ser uma câmara que já tinha. Não foi necessário a instalação de qualquer driver adicional para a utilização desta câmara no *Raspberry Pi*.



Figura 3.2: Fotografia da câmara utilizada.

### 3.3 Software

Para a realização deste projeto foi também pedido para utilizar um conjunto de ferramentas de programação e de análise de imagem padrão, de utilização legal gratuita.

Para isto foram utilizadas ferramentas como o *OpenCV*, para a análise de imagens e o *Tensorflow* para a deteção de objetos.

#### 3.3.1 Raspberry Pi OS

*Raspberry Pi OS* (anteriormente chamado de *Raspbian*) é o Sistema Operativo oficialmente suportado pelo *Raspberry Pi*. Existem outros sistemas operativos suportados pelo *Raspberry Pi*, como o *Ubuntu*, porém optei pela utilização do sistema operativo oficial, devido a ser desenvolvido com foco no *Raspberry Pi*.

Este pode ser obtido através de uma imagem de disco, disponível no website oficial do *Raspberry Pi* [7], e terá de ser inserido num cartão *microSD*.

### 3.3.2 Python

Python é uma linguagem de programação de alto nível imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Esta foi a linguagem proposta para a realização deste projeto e todo o software desenvolvido neste projeto foi escrito nesta linguagem.

Esta linguagem já vêm instalada no sistema operativo do *Raspberry Pi*, porém é recomendada a verificação por novas actualizações do pacote desta linguagem e de outros pacotes relacionados com esta linguagem, através dos seguintes comandos.

```
sudo apt-get update  
sudo apt-get upgrade
```

Exerto de Código 3.1: Comandos utilizados para atualizar pacotes no sistema operativo *Raspberry Pi OS*

### 3.3.3 OpenCV

OpenCV é uma biblioteca de software de Computação Visual e Aprendizado de máquina aberta. Esta foi construída para dispor uma estrutura comum para aplicações de computação visual. Baseado no seu website oficial [5], esta conta com mais de 47 mil contribuições e contém mais de 2500 algoritmos optimizados implementados.

Entre estes algoritmos, neste projeto utilizei várias ferramentas disponibilizadas no OpenCV, como:

- Ferramentas de captura e transformação de uma imagem através de uma câmara USB;
- Algoritmos de rastreamento de objetos;
- Desenho de rectângulos e texto numa imagem.

Optei pela utilização desta biblioteca por ser uma das mais populares e abrangentes na área da Visão Computacional, e satisfazer algumas das necessidades deste projeto.

Devido a esta biblioteca não ser focada em sistemas como o *Raspberry Pi*, esta deverá ser compilada com recurso a um conjunto de tarefas. Para a instalação desta biblioteca recorri a um tutorial presente na website *pyimagesearch* [1], instalando assim a versão completa do *OpenCV*.

### 3.3.4 TensorFlow

*TensorFlow* é uma biblioteca de código aberto para aprendizado de máquina aplicável a uma grande variedade de tarefas. É um sistema para criação e treinamento de redes neurais para detetar e decifrar padrões e correlações. Esta biblioteca é usada como base para várias aplicações, como por exemplo:

- Classificação de texto
- Classificação de vídeo
- Classificação de imagem

Em este projeto utilizei esta biblioteca para a classificação de imagem. Existem outras bibliotecas capazes de lidar com esta tarefa, como o Darknet [10], que deixei de utilizar em este projeto após algumas implementações e testes ineficientes. Em este projeto optámos pela variante *lite* da biblioteca do *Tensorflow*. Esta variante é desenvolvida para dispositivos com baixa capacidade computacional, como é o nosso caso. A principal diferença entre esta variante e a normal é que a versão normal pode ser usada para treinamento em rede e inferência, enquanto a versão *lite* apenas pode ser usada para inferência.

Esta biblioteca pode ser obtida através do website oficial do *TensorFlow* [6] e instalada através de um comando pip (Comando de instalação de pacotes para o Python).

## 3.4 Conclusões

Em suma estas foram as tecnologias e ferramentas utilizadas, que permitiram e facilitaram o desenvolvimento do trabalho referido no próximo capítulo. Foram também utilizadas outras ferramentas, porém abandonadas devido à sua ineficiência para o contexto do trabalho. A área de Computação Visual é uma área ainda um pouco recente, e a capacidade computacional dos equipamentos de ponta é cada vez maior, o que leva a que ainda não existam muitas ferramentas optimizadas para este tipo de equipamento.

Apesar destas ferramentas não parecerem simples à primeira vista, depois de exploradas e estudadas o seu potencial é enorme e tornam-se mais fáceis de utilizar.

## *Capítulo*

# 4

## ***Implementação e Testes***

### **4.1 Introdução**

Neste capítulo é descrito o trabalho realizado ao longo das semanas, as implementações e testes realizados. Em primeiro lugar foram estudadas as bibliotecas envolventes, e só depois se partiu para o programa e testes.

**Este capítulo está estruturado da seguinte forma:**

1. Secção 4.2 - **Estudo e investigação** - Esta secção descreve o estudo e investigação feito para o reconhecimento e rastreamento de objetos.
2. Secção 4.3 - **Programa realizado** - Esta secção descreve a aplicação realizada neste projeto ao longo das semanas.
3. Secção 4.4 - **Testes finais** - Esta secção descreve os testes finais e os seus resultados.
4. Secção 4.5 - **Conclusões** - Esta secção apresentará uma breve conclusão sobre o trabalho realizado.

### **4.2 Estudo e investigação**

A primeira fase de deste projeto foi o estudo e investigação das ferramentas disponíveis para o reconhecimento e rastreamento de objetos. Com isto o trabalho final resultou no uso de dois algoritmos específicos. Em esta secção irei explicar estes algoritmos.

### 4.2.1 Reconhecimento de Objetos

No inicio do trabalho comecei por implementar os algoritmos YOLO e Tiny-YOLO (Variante do YOLO, porem mais rápida e menos precisa) através da biblioteca *Darkflow* [2], que é basicamente a biblioteca *Darknet* em formato *Tensorflow*, adaptada para a linguagem *Python*. Para isto usei modelos pré-treinados das implementações do algoritmo YOLO e Tiny-YOLO, ou seja, usam os pesos (*weights*) de modelos treinados por outras pessoas para realizar as previsões.

Após alguns testes, no melhor dos casos obtive uma média de 3 a 4 *Frames Per Second* (FPS), apenas realizando deteções de objetos a cada iteração num fluxo de vídeo em tempo real. Isto levou me a abandonar esta implementação e procurar por outras soluções mais eficientes.

Depois de um longo estudo e vários testes práticos, acabei por escolher o modelo *MobileNet SSD*, visto a ser já um modelo feito a pensar em dispositivos com baixa capacidade computacional como dispositivos moveis. Ainda assim é um modelo que não abdica de muita precisão para obter um bom desempenho de FPS em estes dispositivos.

Este modelo pode ser visto como a composição de 2 modelos, o *MobileNet* e o *SSD*. Antes de explicar com mais detalhe este modelo, explicarei um pouco destes dois modelos originários nas secções abaixo.

#### 4.2.1.1 MobileNet

A *MobileNet* é um modelo de uma CNN profunda, desenvolvida por pesquisadores da *Google*, cujo o objetivo principal é a eficiência, baixa latência e pequeno tamanho, para que possa ser usado em dispositivos com baixa capacidade computacional.

O modelo serve para aplicações de classificação de imagem. O *textitMobileNet* padrão utiliza o *textitdataset* (conjunto de dados) *ImageNet* [4] com mais de 150000 imagens divididas em 1000 categorias. Ou seja, este é capaz de classificar imagens em 1000 classes pré-definidas na construção do modelo e no treinamento. Mais especificamente este modelo receberá uma imagem, que alimentará na rede e gerará um vetor de 1000 elementos em que cada um deles é a probabilidade de uma respectiva classe estar presente na imagem.

Grande parte da rapidez da rede deve-se à forma com a qual ela calcula as convoluções na imagem. O modelo utiliza a chamada *depthwise separable*

*convolution* e utilizando núcleos de tamanho  $3 \times 3$  utiliza entre 8 e 9 vezes menos poder de processamento que as convoluções comuns.

#### 4.2.1.2 SSD

O Modelo SSD é usado para a deteção. Isto significa que as imagens inseridas terão como saída não apenas a probabilidade de a classe ser encontrada mas também a sua localização. Por isso as imagens usadas para treinamento da rede precisam de conter duas informações: a classificação dos objetos que estão contidos na imagem e as suas *ground truth boxes*, caixas delimitadoras que delimitam o objeto classificado na imagem.

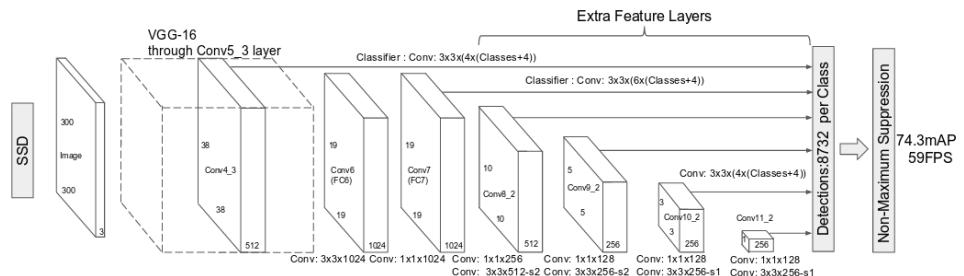


Figura 4.1: Arquitetura do modelo SSD.

Como podemos ver na figura 4.1, este modelo contém uma CNN treinada para a classificação de imagens (VGG-16), com as camadas de classificação cortadas e então alimenta o resto do modelo. Este modelo contém 6 camadas de convolução de dimensões decrescentes. A ultima camada na rede é uma camada de supressão de não-máximo.

Cada célula do mapa de característica possui caixas de deteção padrão, e a rede calcula para cada caixa de deteção padrão de cada célula, a probabilidade de estar contida naquela caixa cada uma das classes definidas no modelo e a variação de posição, largura e altura que a caixa padrão terá de sofrer para que a classe detetada esteja contida correctamente e completamente no interior.

As deteções de todas as células das camadas de convolução são alimentadas numa camada de supressão de não-máximo que remove todas as deteções sobrepostas a fim de manter apenas uma caixa delimitadora por objeto contido na imagem.

#### 4.2.1.3 Mobilenet-SSD

Este foi o modelo utilizado para o reconhecimento de objetos neste projeto. Este modelo consiste basicamente, num modelo SSD cuja a rede que alimenta as camadas anteriores às de localização são retiradas de um modelo textitMobileNet, cortado antes das camadas de classificação.

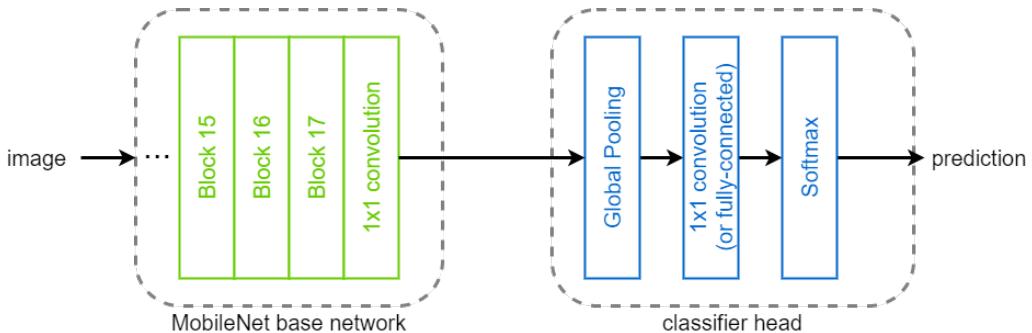


Figura 4.2: Exemplo das camadas finais de uma rede *MobileNet*.

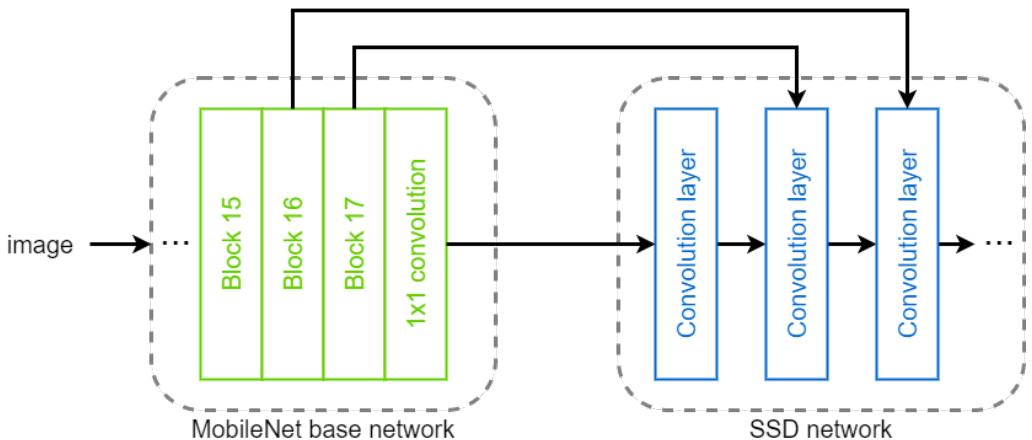


Figura 4.3: Exemplo das camadas finais de uma rede *MobileNet* alimentando uma rede SSD.

As figuras 4.2 e 4.3 retratam as diferenças básicas nas camadas finais da arquitetura *MobileNet* utilizada para a classificação de imagens. Percebe-se que a rede *MobileNet* serve de base para o modelo SSD provendo o processo de extração de características.

Em este projeto usei este modelo citado pré-treinado pelos seus desenvolvedores no *dataset Common Objects In Context* (COCO), um *dataset* com

mais de 200 mil imagens devidamente classificadas. Este encontra-se disponível para download através da página oficial do *Tensorflow* [3]. Este modelo em específico foi feito para detetar até 10 objetos numa imagem e é capaz de reconhecer 80 classes de objetos.

### 4.2.2 Rastreamento de objetos

Após testar na prática alguns dos algoritmos estudados para o rastreamento de objetos, acabei por escolher o algoritmo *Minimum Output Sum of Squared Error* (MOSSE), para a continuação da realização deste projeto. A escolha deste algoritmo deve-se principalmente à sua performance perante a baixa capacidade computacional.

Nas subsecções abaixo irei explicar um pouco mais sobre o funcionamento deste algoritmo.

#### 4.2.2.1 MOSSE

Este é um algoritmo robusto a variações de iluminação, escala, pose e deformações não rígidas. Este é um algoritmo baseado em fórmulas matemáticas e usa correlação adaptativa. Neste capítulo podemos ver uma breve explicação sobre o funcionamento deste rastreador. Em primeiro lugar serão explicadas algumas das etapas mais complexas deste rastreador e posteriormente enumerados os passos dados pelo rastreador durante o seu trabalho.

#### 4.2.2.2 Pré-processamento

O pré-processamento é uma etapa realizada a cada quadro do vídeo a ser rastreado. Esta etapa transforma uma imagem num domínio espacial numa imagem no domínio de *Fourier*. Aqui podemos ver as etapas deste pré-processamento:

1. Em primeiro lugar recebemos uma imagem recortada centrada no objeto.
2. A imagem dada é convertida para uma imagem em escala de cinzentos.
3. É realizada uma transformação logarítmica na imagem anterior, através da seguinte equação:

$$x = \ln(y + 1)$$

Nesta equação, x e y representam as coordenadas do pixel x e y da imagem de entrada e de saída. Esta função é aplicada para reduzir os efeitos

da iluminação e melhorar o contraste, disponibilizando recursos com alto contraste para o inicializar o filtro.

4. Os valores obtidos na etapa anterior são normalizados para obter uma média de zero e uma normal de um. A normalização ajuda a diminuir os efeitos da mudança na iluminação.
5. O modelo obtido na etapa anterior é convertido do domínio espacial para o domínio de Fourier. A transformada de Fourier é usada para decompor um sinal em seus componentes seno e cosseno. Uma imagem não é um sinal analógico, portanto usamos a transformada discreta de *Fourier* (DFT) para uma imagem. O output dessa transformação representa a imagem no domínio de *Fourier* ou de frequência. A seguinte equação é usada para o DFT numa imagem bidimensional:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-2\pi i (\frac{xy}{M} + \frac{yy}{N})} \quad u = 0, \dots, M-1; \quad v = 0, \dots, N-1$$

Nesta equação  $F(u, v)$  representa a imagem no domínio das frequências e  $f(x, y)$  representa a imagem no domínio espacial.

#### 4.2.2.3 Alvo sintético

O alvo sintético é um *output* (saída) de correlação desejado gerado sinteticamente. Uma imagem do alvo sintético contém um pico gaussiano centrado no objeto. O alvo sintético é usado na inicialização do filtro e para atualizar o mesmo durante o rastreamento, mapeando o modelo com o output desejado. A seguinte equação é usada para gerar sinteticamente a imagem do alvo sintético:

$$g_i = \sum e^{-\frac{(x-x_j)^2 + (y-y_j)^2}{\sigma^2}}$$

Nesta equação,  $g_i$  é o alvo gerado sinteticamente.  $x$  e  $y$  representam a localização dos pixels na imagem.  $x_j$  e  $y_j$  representam a localização do centro do objeto, o centroide da caixa delimitadora. O raio do pico é representado por  $\sigma$ .

#### 4.2.2.4 Etapas do rastreador

Para melhor entender este algoritmo irei enumerar os passos dados pelo mesmo abaixo, para rastrear um objeto:

1. Para que este rastreador possa ser inicializado, são dadas a este as coordenadas da caixa delimitadora deste objeto.
2. É feito um modelo do objeto, através do recorte da caixa delimitadora na imagem original.
3. Este modelo é pré-processado e transformado para o domínio de Fourier, obtendo  $F$ , como explicado na secção 4.2.2.2.
4. É criado um alvo sintético  $G$ , como explicado na secção 4.2.2.3.
5. Os valores de  $F$  e  $G$  dos passos anteriores são substituídos pelos valores de  $F_1$  e  $G_1$  na seguinte equação para calcular  $N_1$  e  $D_1$  para o primeiro quadro do vídeo.

$$N_1 = (G_1 \odot F_1^*)$$

$$D_1 = (F_1 \odot F_1^* + \epsilon)$$

Nas expressões anteriores  $\odot$  significa produto de *Hadamard*. O símbolo  $*$  representa o conjugado de um numero complexo.

6. O passo anterior pode ser repetido várias vezes, o que faz com que o objeto se torne mais distinto.
7. Os valores obtidos para  $N$  e  $D$  nos passos anteriores são usados para calcular o filtro usando a seguinte equação:

$$H_i^* = \frac{N_i}{D_i}$$

Esta equação dá nos o conjugado complexo do filtro,  $H^*$ .

8. Aqui começa o rastreamento do objeto. O quadro de vídeo é pré-processado e convertido para o domínio de *Fourier*,  $F$ . A imagem pré-processada no domínio de *Fourier* é correlacionada com o filtro usando a seguinte equação:

$$G = H^* \odot F$$

A localização do pico em  $G$  diz nos a nova localização do objeto.

9. O passo anterior é repetido nos restantes quadros do vídeo para rastrear o objeto.

Através destes passos conseguimos rastrear objetos de uma forma simples, precisa e rápida.

### 4.3 Programa realizado

A implementação destes algoritmos resultou num programa capaz de reconhecer e rastrear objetos num ambiente de luz natural. Em esta secção iremos explicar os detalhes do código desenvolvido.

Isto resultou em programa com um ciclo infinito, dividido em duas partes, reconhecimento e rastreamento. Antes de iniciar este ciclo, e com o objeto de aumentar a eficácia em FPS será iniciado o interpretador *TensorFlow*, com 2 *Threads* alocados para o trabalho deste. Isto foi feito através do trecho de código seguinte:

```
# Carrega o modelo do TensorFlow Lite.
interpreter = Interpreter(model_path=PATH_TO_CKPT)

# Alocação de mais Threads para o trabalho do TensorFlow
interpreter.set_num_threads(2)
interpreter.allocate_tensors()
```

Exerto de Código 4.1: Inicialização do *TensorFlow Lite*

Reservei também 1 *Thread* para o trabalho do *OpenCV*, através da seguinte chamada:

```
cv2.setNumThreads(1)
```

Exerto de Código 4.2: Alocar *threads* para o *OpenCV*

Para a captura de vídeo em tempo real através da câmara USB, desenvolvi uma classe dedica a esta tarefa. Esta dispõem também um *Thread* dedicado. Podemos ter uma visualização desta classe no seguinte trecho de código:

```
# Classe VideoStream para lidar com o streaming de video através de uma
# webcam num processo diferente
class VideoStream:
    def __init__(self, resolution=(640,480), framerate=30):
        # Inicializa a câmara e a stream de imagens
        # 0 corresponde à câmara 0 numa lista de câmeras conectadas ao
        # Raspberry Pi. Se apenas uma câmara conectada, selecionará
        # sempre essa.
        self.stream = cv2.VideoCapture(0)
        ret = self.stream.set(3, resolution[0])
        ret = self.stream.set(4, resolution[1])

        # Lê o primeiro frame da stream
        (self.grabbed, self.frame) = self.stream.read()

        # Variável para controlar a câmara quando esta for parada
        self.stopped = False
```

```

def start(self):
    # Inicia o Thread que irá ler quadros de vídeo da stream de vídeo
    Thread(target=self.update, args=()).start()
    return self

def update(self):
    # Mantém um loop infinito até o Thread ser parado
    while True:
        # Se a câmara parar, para o Thread
        if self.stopped:
            # Fecha os recursos da câmara
            self.stream.release()
            return

        # Obtém o frame seguinte da stream de video
        (self.grabbed, self.frame) = self.stream.read()

def read(self):
    # Retorna o frame mais recente
    return self.frame

def stop(self):
    # Indica que a câmara e o Thread devem parar
    self.stopped = True

```

Exerto de Código 4.3: Classe dedicada ao trabalho da câmara

Uma das características deste programa é a capacidade de recuperar objetos perdidos durante o rastreamento ou reconhecimento, para isto implementei uma lista que guardará objetos que num determinado fotograma deixem de ser detetados, para que possa recuperá-los no futuro. Esta lista guardará várias informações para cada objeto, para que possam ser usadas no futuro, como, caixa de delimitadora, classe e um tempo correspondente ao tempo em que deixou de detetar este objeto.

Implementei um método auxiliar para a recuperação destes objetos. Este método será responsável também pela eliminação de antigas perdas. Isto servirá para:

- Reduzir o numero de ciclos ao percorrer a lista nas próximas chamadas à função, aumentando a eficiência do programa;
- Previr que novos objetos que apareçam na cena em determinado fotograma, possam ser confundidos com antigos objetos.

O factor responsável pela eliminação de antigas perdas, será o tempo. Isto é uma variável ajustável, dependendo da cena em que este programa fosse

usado e dos objetivos da deteção, como por exemplo, em cenários onde a câmara esteja fixa:

- Cenas em que os objetos a ser detetados, estejam próximos da câmara (Por exemplo: Uma divisão numa casa): Objetos iriam provavelmente manter-se na cena durante mais tempo, e seriam os mesmos, podendo assim aumentar-se este tempo;
- Cenas em que os objetos a ser detetados, estejam longe da câmara (numa estrada por exemplo, em que o objetivo seja detetar carros): Objetos iriam variar mais e manter-se na cena durante pouco tempo, podendo diminuir-se este tempo.

Como padrão defini este tempo de 30 segundos, e este será verificado, comparando o tempo atual com o antigo tempo guardado na lista de objetos perdidos.

Para verificar se uma nova deteção realizada, corresponde com uma antiga deteção perdida, usamos a distância entre centroides e a classe para nos referirmos ao mesmo objeto. Para isto uso também uma distância ajustável, podendo ser ajustada dependendo da cena em que o programa fosse usado. Em cenas em que a câmara estive-se perto dos objetos a ser detetados, esta distância poderia ser maior do que em cenas em que os objetos estivessem mais distantes da câmara.

Podemos visualizar esta lista e função no seguinte trecho de código:

```
# Lista de rastreamentos falhados, para que possam ser recuperados
# Composta por tuplos (tuplo caixa, tuplo cor, tuplo predição, tempo em
# milissegundos de quando falhou)
failed_detections = []

def getOldObject(new_box, object_name):
    # Retorna um antigo objeto se o centroide da caixa e nome de objeto
    # coincidirem

    global failed_detections

    new_failed_detections = failed_detections.copy()
    element_to_return = None
    removed = 0

    atual_time = int(round(time.time() * 1000))

    for i, obj in enumerate(failed_detections):
        """
        0 Caixa delimitadora
        1 Cor
        2 Predição
        3 Tempo em milissegundos
        """
        if obj[0] == new_box and obj[1] == object_name:
            element_to_return = obj
            removed += 1
            break

    if element_to_return != None:
        failed_detections.remove(element_to_return)

    if removed > 0:
        failed_detections = new_failed_detections
```

```
1 Cor
2 Predicção
3 Tempo de quando deixou de ser detectado
"""

# Vamos verificar o tempo dos rastreamentos falhados (menos
# interacções na próxima vez)
if (obj[3] + time_to_lose_object < atual_time):
    # Já passou 30 segundos!
    # Vamos remover da lista
    del new_failed_detections[i-removed]
    removed += 1
    print("Perdemos o objeto [" + str(obj[2][0]) + "] " + obj
          [2][1] + "\nTempo passado: " + str((atual_time - obj
          [3])))
    continue
else:
    # Ainda não passou 30 segundos!

    # É o mesmo objeto ?
    if obj[2][1] == object_name:

        box1 = obj[0]

        box_centroid = (int(new_box[0]+(new_box[2]/2)), int(
            new_box[1]+(new_box[3]/2)))
        box1_centroid = (int(box1[0]+(box1[2]/2)), int(box1[1]+(
            box1[3]/2)))

        # Vamos comparar os centroids com uma distância de
        # max_distance_same_object pixels
        if box_centroid[0] - box1_centroid[0] <
           max_distance_same_object and box_centroid[0] -
           box1_centroid[0] > -max_distance_same_object:
            if box_centroid[1] - box1_centroid[1] <
               max_distance_same_object and box_centroid[1] -
               box1_centroid[1] > -max_distance_same_object:

                # Os centroides estão próximos!

                # Vamos prever que seja o mesmo objeto, retornar
                # :
                element_to_return = obj

                # Como vamos retornar, removemos também da lista
                !
                del new_failed_detections[i-removed]

                # Já temos o que queríamos. Vamos embora!
```

```

    Analisamos os restantes tempos superiores a
    30 segundos para a próxima.
break

failed_detections = new_failed_detections

return element_to_return

```

Exerto de Código 4.4: Função de recuperação de objetos perdidos

Passando ao ciclo principal, este é dividido em duas partes que são executadas alternadamente. Uma parte para a deteção de objetos e outra para o rastreamento dos objetos já detetados. A maneira em que alternamos entre estes estados é:

- De  $k$  em  $k$  quadros de vídeo, sendo  $k$  também uma variável ajustável;
- Sempre que uma deteção não detete nenhum objeto;
- Sempre que o rastreamento de objetos falhe.

Sempre que uma destas ações acontecer, o programa volta a executar a deteção de objetos.

Durante a deteção de objetos, os novos objetos detetados serão comparados com objetos antigos através do rastreamento de centroides, para que possamos recuperar os objetos dos quadros de vídeo anteriores.

É também aqui que verificamos se existem antigos objetos perdidos durante alguma deteção e que possam ser recuperados. O seguinte trecho de código, contém algumas das partes da deteção faladas aqui.

```

# Executa a deteção, correndo o modelo com a imagem (frame) de input
interpreter.set_tensor(input_details[0][ 'index' ],input_data)
interpreter.invoke()

# Recebe os resultados da deteção
boxes = interpreter.get_tensor(output_details[0][ 'index' ]) [0] # 
    Coordenadas das caixas delimitadoras do objetos encontrados
classes = interpreter.get_tensor(output_details[1][ 'index' ]) [0] # Index
    das classes dos objetos detetados
scores = interpreter.get_tensor(output_details[2][ 'index' ]) [0] # Confian
    ça dos objetos detetados
# num = interpreter.get_tensor(output_details[3][ 'index' ]) [0] # Numero
    total de objetos detetados (Impreciso e não preciso)

```

```
# Corre por todas as deteções
for i in range(len(scores)):

    # Verifica se a percentagem de certeza é maior que a minima indicada
    if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

        # Pega as coordenadas da caixa delimitadora
        # O interpretador pode devolver coordenadas fora das dimenções
        # da imagem, vamos força las a pertencerem à imagem usando o
        # max() e min()
        ymin = int(max(1,(boxes[i][0] * imH)))
        xmin = int(max(1,(boxes[i][1] * imW)))
        ymax = int(min(imH,(boxes[i][2] * imH)))
        xmax = int(min(imW,(boxes[i][3] * imW)))

        object_name = labels[int(classes[i])]

        # Vamos verificar se este já existia
        # Posição de objeto na lista (Centroid Traking)
        original_i = getData((xmin, ymin, xmax-xmin, ymax-ymin),
                             object_name)

        # Atualizamos o ID do ultimo objeto
        last_id += 1

        new_box = (xmin, ymin, xmax-xmin, ymax-ymin)
        new_color = (randint(64, 255), randint(64, 255), randint(64,
                                                               255))
        # Atribui novo id, porém poderá ser reposado por um já existente
        new_predict = (last_id, object_name, int(scores[i]*100))

        # Objeto já existe
        if original_i != -1:

            last_id -= 1

            if original_i not in checked:

                new_color = colors[original_i]
                new_predict = (predictions[original_i][0], predictions[
                               original_i][1], int(scores[i]*100)) # Atualiza
                # percentagem

                checked.append(original_i)

        else:

            # Objeto existe, porém já está a ser usado
            # Provavelmente existem 2 objetos do mesmo tipo muito
```

```

        perto um do outro -> Vou supor que seja o mesmo
        objeto ->
    continue

else:
    # Vamos verificar se existem objeto perdidos anteriormente

    old_obj = getOldObject(new_box, object_name)

    if (old_obj != None):

        new_color = old_obj[1]
        new_predict = (old_obj[2][0], old_obj[2][1], int(scores[
            i]*100)) # Atualiza percentagem
        last_id -= 1

```

Excerto de Código 4.5: Funcionalidades da deteção de objetos

Como resultado final foi obtido um programa que junta dois algoritmos de rastreamento, o algoritmo MOSSE e o algoritmo de rastreamento de centróides. Usa também as bibliotecas *OpenCV* para a implementação do algoritmo *MOSSE* e a biblioteca *Tensorflow Lite* para o uso do modelo de deteção *MobileNet-SSD*. O *TensorFlow* Lite é iniciado quando o programa é iniciado e corre durante todo o programa em segundo plano para que não haja perdas de FPS ao iniciar este interpretador.

## 4.4 Testes finais

Durante a elaboração deste projeto foram feitos imensos testes, porém estes apenas serviram para melhorar o desempenho e qualidade do programa.

No final foi obtido um programa capaz de executar com uma média de 130 FPS, este resultado deve-se principalmente ao rastreador de objetos. Este foi projectado para executar a uma elevada quantidade de FPS, segundo os autores a mais de 400 FPS. Foi conseguido um pouco menos no *Raspberry Pi*, porém não foram visualizadas perdas significativas no rastreamento de objetos. Se só executado o detetor de objetos em um fluxo de vídeo em tempo real, este deteta objetos com uma média de 6 FPS, e quando executado em conjunto com o rastreador, nota-se uma pequena perda no contador de FPS nos quadros de vídeo seleccionados para a deteção de objetos, porém quase imperceptível ao visualizar o vídeo.

Este programa conta com dois problemas:

- Por vezes este detetor de objetos não deteta todos os objetos presentes na imagem, o que leva a que estes fiquem sem a correspondente caixa delimitadora durante alguns quadros de vídeo. Para isto foi implementado um sistema de recuperação dos objetos que em antigos quadros de vídeo não tenham sido detetados. Assim, mesmo que não sejam detetados durante um curto tempo, se num frame futuro voltem a ser detetados na mesma área, será recuperado o *Identification (Identificação)* (ID) do mesmo;
- O rastreador MOSSE não lida muito bem com a oclusão e se um objeto desaparecer por completo, este também termina o rastreamento do mesmo. O sistema que implementei para a recuperação de antigos objetos, é também usado para estes casos. Se o objeto voltar a aparecer na mesma área ou perto deste será recuperado, porém se reaparecer num sítio distante da última aparição, este será interpretado como um novo objeto.

Ainda assim, de forma geral conseguiu-se resultados bastante aceitáveis. Estes FPS foram obtidos através do registo dos mesmos e do cálculo no final do programa, após 5 minutos de execução. O cenário é o presente na figura 4.4.



Figura 4.4: Cenário onde foram realizados os testes finais.

## **4.5 Conclusões**

Todo o estudo e investigação feitos, foram sempre feitos alternadamente com o desenvolvimento de software. Apesar das categorias de estudo e de implementação serem apresentadas em separado, à medida que o estudo foi feito, vários programas e tutoriais foram desenvolvidos.

## *Capítulo*

# 5

## ***Conclusões e Trabalho Futuro***

### **5.1 Conclusões Principais**

O reconhecimento de objetos bem como o rastreamento são temas da inteligência artificial que cada vez estão a evoluir mais. É impressionante como hoje em dia as máquinas são capazes de reconhecer objetos e rastreá-los, e ainda mais, fazendo isso de um modo em que não consome uma quantidade enorme de processamento. Se ainda "ontem" eram precisos super computadores para realizar estas tarefas, cada vez mais os computadores de bolso, como *smartphones* são capazes de lidar com estas tarefas com maior facilidade.

Esta temática está bastante mais avançada do que pensava, porém ainda não é perfeita. Principalmente a parte do rastreamento, o desafio de trabalhar com a baixa capacidade de processamento, levou me ao estudo de vários algoritmos mais optimizados ou menos eficientes, no entanto, a tarefa de rastreamento é a tarefa menos à altura do atual estado da arte de um detector de objetos mais avançado.

Este foi um projeto que adorei fazer, e que aumentou significativamente o meu interesse e gosto pelas áreas presentes, como a Visão Computacional. Levou também a aprender os principais conceitos e estado da arte desta área. Entre o aprendido, gostei bastante de aprender como funciona uma CNN, e os algoritmos baseados nesta arquitetura. Gostei também de aprender o funcionamento do algoritmo MOSSE, bem como todos os restantes estudados.

O programa a desenvolver foi um desafio em obter um resultado aceitá-

vel entre a confiança das deteções/rastreamentos e desempenho a nível de FPS, que considero alcançado. Foi uma tarefa que requereu também algum conhecimento sobre Algoritmos e a programação em si.

## 5.2 Trabalho Futuro

Os objetivos propostos no projeto foram compridos. Como objetivos extra, poderiam ter sido feitas várias versões do programa, já adaptadas para vários cenários e propósitos diferentes. Com isto poderiam ter sido feitos algoritmos e optimizações mais específicas para cada cenário, e assim obter melhores resultados em cada cenário e objetivo.

Também não sendo o objetivo do trabalho, poderia ter sido feito o treinamento de uma rede convolucional profunda, projectada especificamente para os objetos com os quais queríamos trabalhar. Com isto poderíamos talvez obter melhores resultados.

Outras aplicações interessantes, porém não era o objetivo deste projeto, poderá ser a implementação deste programa em outros dispositivos de baixa capacidade computacional, como smartphones comuns.

## **Bibliografia**

- [1] Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster, 2019. [Online] <https://www.pyimagesearch.com/2019/09/16/install-opencv-4-on-raspberry-pi-4-and-raspbian-buster/>. Último acesso a 25 de Abril de 2020.
- [2] darkflow, 2020. [Online] <https://www.tensorflow.org/lite/guide/python>. Último acesso a 11 de Maio de 2020.
- [3] Deteção de objetos | tensorflow lite, 2020. [Online] [https://www.tensorflow.org/lite/models/object\\_detection/overview](https://www.tensorflow.org/lite/models/object_detection/overview). Último acesso a 10 de Junho de 2020.
- [4] Image-net, 2020. [Online] <http://www.image-net.org/about-overview>. Último acesso a 4 de Junho de 2020.
- [5] OpenCV About, 2020. [Online] <https://opencv.org/about/>. Último acesso a 20 de Junho de 2020.
- [6] Python quickstart, 2020. [Online] <https://www.tensorflow.org/lite/guide/python>. Último acesso a 10 de Maio de 2020.
- [7] Raspberry Pi OS (previously called Raspbian), 2020. [Online] <https://www.raspberrypi.org/downloads/raspberry-pi-os/>. Último acesso a 20 de Abril de 2020.
- [8] Understanding object detection, 2020. [Online] <https://towardsdatascience.com/understanding-object-detection-9ba089154df8>. Último acesso a 21 de Abril de 2020.
- [9] VNC (Virtual Network Computing), 2020. [Online] <https://www.raspberrypi.org/documentation/remote-access/vnc/>. Último acesso a 7 de Maio de 2020.
- [10] Joseph Redmon. Darknet: Open source neural networks in c, 2013–2016. [Online] <http://pjreddie.com/darknet/>. Último acesso a 10 de Maio de 2020.