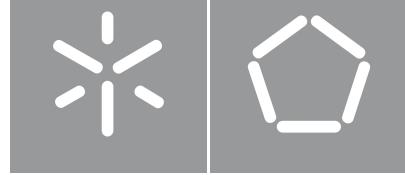


Universidade do Minho
Escola de Engenharia

Catarina Freitas da Cruz

**Framework para análise de comportamentos
de objetos interativos em vídeo jogos**



Universidade do Minho
Escola de Engenharia

Catarina Freitas da Cruz

**Framework para análise de comportamentos
de objetos interativos em vídeo jogos**

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação de
Paulo Novais
André Pimenta

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho:



CC BY

<https://creativecommons.org/licenses/by/4.0/>

Agradecimentos

No decorrer do último ano, o trabalho que tive de desenvolver não teria sido o mesmo sem o contributo de algumas pessoas muito especiais para mim, que sempre me apoiaram nos momentos críticos. Deste modo, quero agradecer e dedicar este documento a todas elas como forma de gratidão.

Em primeiro, ao meu orientador Professor Paulo Novais e co-orientador André Pimenta que me deram a oportunidade e liberdade de trabalhar num tema do meu interesse e que me chama muito a atenção. Agradeço também a todos os funcionários da empresa Anybrain que me ajudaram, deram sugestões e partilharam comigo conhecimentos que me fizeram prosseguir os meus interesses.

De seguida, quero agradecer aos meus pais, Alice Freitas e António Cruz, por tudo o que fizeram e continuam a fazer por mim. Obrigada por sempre acreditarem em mim e por todas as condições e oportunidades que me proporcionaram. Agradeço também aos meus irmãos, Célia e Luís Cruz e ao meu primo Paulo Freitas por sempre me apoiarem e aturarem e acima de tudo por todos os conselhos que só irmãos sabem dar.

Agradeço aos meus amigos e colegas de curso por todos os momentos de diversão e descontração que passamos juntos e acima de tudo por partilharem e passarem comigo alguns dos momentos mais difíceis, é sempre mais fácil percorrer uma estrada com uma amigo do que sozinha.

E como não podia deixar de ser, um especial agradecimento ao meu namorado Rodrigo Pereira, por toda a ajuda, paciência e motivação sem a qual era impossível ter acabado este projeto e ajudaram mais do que posso expressar. Obrigada por todos os momentos de suporte e apoio que sabias que precisava mesmo sem ter de dizer. Agradeço também à mãe dele, professora Natália Pereira, pela ajuda na revisão final relativamente à escrita desta dissertação.

Por fim, queria agradecer a todos os que me ajudaram direta ou indiretamente ao longo de todo o percurso académico.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, Braga, novembro 2022

Catarina Freitas da Cruz

Resumo

Com um crescimento exponencial tanto na área da Inteligência Artificial como dos vídeo jogos, a criação de plataformas que auxiliam os jogadores passou a ser fundamental. A criação de uma ferramenta analítica que estude detalhadamente o comportamento humano, abre portas a jogos mais dinâmicos, competitivos e justos.

A análise do ecrã de um jogador permite identificar, detetar e rastrear movimentos de determinados objetos, em tempo real, podendo ter o intuito de o ajudar ou de o vigiar. Seja qual for o caso, é necessário, primeiro, identificar e detetar os objetos visualizados, através de algoritmos de *Object Detection*. Depois, já identificado o objeto, é possível prever a sua próxima localização, bem como rastrear o seu movimento, utilizando algoritmos de *Object Tracking*.

Intercalando o rastreamento com a deteção de objetos, quer quando este desaparece de vista, quer para obter confirmação que se está a seguir o objeto correto, é possível assim analisar o ecrã do jogador para o poder ajudar.

Esta dissertação tem como objetivo desenvolver um modelo capaz de identificar o movimento de um determinado objeto, em tempo real, no ambiente de um jogo, utilizando para isso técnicas de *Machine Learning* e *Computer Vision*, mais especificamente métodos de *Object Detection* e *Object Tracking*.

O ambiente prático será desenvolvido utilizando a biblioteca OpenCV para Python, que tem ao dispor um diverso leque de algoritmos de *Computer Vision* e ainda permite a utilização paralela de CPU e GPU para a otimização destes mesmos algoritmos.

Palavras-chave Inteligência Artificial, Machine Learning, Computer Vision, CNN, Object Detection, Object Tracking, OpenCV

Abstract

With exponential growth both in the area of Artificial Intelligence and videogames, the creation of platforms that help players has become fundamental. The creation of an analytical tool that analyzes human behavior, opens the door to more dynamic, competitive and fair games.

The analysis of a player's screen allows identifying, detecting and tracking movements of certain objects, in real time, and can help or monitor them. Whatever the case, it is first necessary to identify and detect the objects visualized, through Object Detection algorithms. Then, once the object has been identified, it is possible to predict its next location, as well as track its movement, using Object Tracking algorithms.

It is possible to analyze the player's screen by interleaving object tracking with object detection, either when it disappears from view, or to obtain confirmation that the correct object is being followed, thus helping the player.

This dissertation aims to develop a model capable of identifying the movement of a given object, in real time, in a game environment, using Machine Learning and Computer Vision techniques, more specifically methods of Object Detection and Object Tracking.

The practical environment will be developed using the OpenCV python library, which has a diverse range of computer vision algorithms available and also allows the parallel use of CPU and GPU for the optimization of these same algorithms.

Keywords Artificial Intelligence, Machine Learning, Computer Vision, CNN, Object Detection, Object Tracking, OpenCV

Conteúdo

1	Introdução	1
1.1	Contextualização	1
1.2	Motivação	2
1.3	Objetivos	3
1.4	Estrutura do documento	4
2	Estado da arte	5
2.1	Inteligência Artificial	5
2.2	Visão por Computador	6
2.2.1	Desafios da Visão por Computador	7
2.2.2	Aplicações existentes no mercado	9
2.2.3	OpenCV	12
2.3	Deteção e Reconhecimento de Objetos	12
2.3.1	Imagens e as suas características	12
2.3.2	Convolutional Neural Network (CNN)	14
2.3.3	Algoritmos de Deteção de Objetos	18
2.3.4	Template Matching	21
2.4	Análise de movimento e rastreamento de objetos	28
2.4.1	Deteção de Objetos vs Rastreamento de Objetos	28
2.4.2	Desafios no rastreamento de objetos	29
2.4.3	Técnicas de deteção de objetos para o Rastreamento	29
2.4.4	Métodos de Rastreamento de Objetos	31
2.4.5	Rastreamento de Objetos no OpenCV	32
2.4.6	Comparação entre Algoritmos de Rastreamento de Objetos no OpenCV	33
2.5	Síntese	35

3 Detect & Track	37
3.1 Levantamento de Requisitos	37
3.2 Detetar objetos (<i>Object Detection</i>)	39
3.2.1 YOLO	39
3.2.2 Template Matching	42
3.2.3 Capturar Janela	46
3.3 Rastrear objeto (<i>Object Tracking</i>)	48
3.3.1 GOTURN	48
3.3.2 Problemas e limitações	51
3.4 Plataforma e API	52
3.4.1 Protótipo	52
3.4.2 Tecnologias usadas	54
3.4.3 Pedidos à API	56
3.4.4 Interface Final	58
3.4.5 Análise de usabilidade	61
4 Testes e análise de resultados	67
4.1 Especificações da máquina	67
4.2 Jogos	68
4.3 Imagens	68
4.4 Threshold	69
4.5 Resultados obtidos e a sua análise	71
4.6 Conclusão dos resultados obtidos	80
5 Conclusões e trabalho futuro	81
5.1 Conclusões	81
5.2 Perspetiva de trabalho futuro	83

Lista de Figuras

1	Ilusão: vaso ou rosto? Castelli [2019]	8
2	Diversas figuras de cães	9
3	Matriz de uma imagem em tons de cinza Elgendi [2020]	13
4	Junção das 3 matrizes RGB Visalpara et al. [2020]	13
5	<i>Artificial Neural Network</i> Medisetti [2021]	15
6	<i>Convolutional layer Education</i> [2020a]	16
7	Tipos de <i>Pooling</i> Saha [2018]	17
8	<i>Convolutional Neural Network</i> Swapna	17
9	<i>R-CNN: Regions with CNN features</i> Gandhi [2018]	18
10	You Only Look Once (YOLO) Gandhi [2018]	20
11	Real-Time Object Detection on COCO coc [a]	21
12	Exemplo ilustrativo de <i>Template Matching</i> Rosebrock [2021a]	22
13	Exemplo da deslocação da imagem modelo Rosebrock [2021a]	23
14	Matriz Resultante da semelhança com a imagem modelo Rosebrock [2021a]	23
15	Fórmula de cálculo e Resultado do método CV_TM_SQDIFF ope [e]	25
16	Fórmula de cálculo e Resultado do método CV_TM_SQDIFF_NORMED ope [e]	25
17	Fórmula de cálculo e Resultado do método CV_TM_CCORR ope [e]	25
18	Fórmula de cálculo e Resultado do método CV_TM_CCORR_NORMED ope [e]	26
19	Fórmula de cálculo e Resultado do método CV_TM_CCOEFF ope [e]	26
20	Fórmula de cálculo e Resultado do método CV_TM_CCORFF_NORMED ope [e]	26
21	Valores de entrada para Multi-Template Matching Kang and Atuk [2020]	27
22	Deteção de múltiplos objetos em simultâneo (<i>multi-template Matching</i>) Kang and Atuk [2020]	27
23	Taxa de sucesso dos algoritmos de rastreamento do OpenCV Janku et al. [2016]	33
24	Exatidão Média dos algoritmos de rastreamento do OpenCV Dardagan et al. [2021]	34

25	Comparação dos vários algoritmos de rastreamento relativamente a exatidão e robustez Held et al. [2016]	34
26	Fluxo de dados	36
27	Imagens de pessoas reais a passear na rua	41
28	Imagens do jogo Brawlhalla	41
29	Imagens do jogo de League of Legends	41
30	Imagens do jogo CS-GO	42
31	Imagens de input e output da função <code>matchTemplate()</code> com as diferentes métricas normalizadas possíveis	44
32	Deteção de múltiplos objetos através do <i>Template Matching</i>	46
33	Arquitetura do rastreamento do algoritmo GOTURN	49
34	Fluxograma do programa	51
35	Protótipo da Plataforma	53
36	Stack tecnológica utilizada	54
37	Fluxograma do <i>endpoint</i> RUN	57
38	Fluxograma do <i>endpoint</i> STOP	58
39	Interface final da plataforma desenvolvida	61
40	Especificações da máquina	67
41	Regra de escolha de imagem modelo	69
42	Objetos detetados com diferentes valores de <i>threshold</i>	70
43	Resultados obtidos da deteção de um único objeto em jogos 2D	73
44	Deteção dos jogadores nos diversos jogos	74
45	Deteção de objeto parcialmente oculto e a rodar	75
46	Resultados obtidos da deteção de vários objetos em jogos 2D	76
47	Deteção de um Objeto vs Deteção de vários objetos	77
48	Deteção de vários objetos com deslocação	78
49	Resultados obtidos da deteção objetos num jogo 3D	79
50	Deteção de multi-objetos com rotação	80

Siglas e Acrônimos

AI - Artificial Intelligence

ANN - Artificial Neural Networks

API - Application Programming Interface

Caffe - Convolutional Architecture for Fast Feature Embedding

CNN - Convolutional Neural Network

CPU - Central Processing Unit

CSRT - Channel and Spatial Reliability Tracking

CV - Computer Vision

DCF-CSR - Discriminative Correlation Filter with Channel and Spatial Reliability

FPS - Frames Per Second

GIL - Global Interpreter Lock

GOTURN - Generic Object Tracking Using Regression Networks

GPU - Graphics Processing Unit

IA - Inteligência Artificial

ID - Identity Document

KCF - Kernelized Correlation Filters

KNN - K-Nearest Neighbors

LSTM - Long short-term memory

MAP - Mean Average Precision

MIL - Multiple Instance Learning

ML - Machine Learning

MOOSE - Minimum Output Sum of Squared Error

PLN - Processamento de Linguagem Natural

RCNN - Region-based Convolutional Neural Networks

ReLU - Rectified Linear Unit

RGB - Red Green Blue

RNN - Recurrent Neural Network

ROLO - Recurrent YOLO

SORT - Simple Online Real-time Tracker

SSD - Single Shot Detector

SVM - Support Vector Machines

TLD - Tracking, Learning, and Detection

VC - Visão por Computadores

YOLO - You Only Look Once

Capítulo 1

Introdução

1.1 Contextualização

Desde a sua criação, a indústria dos jogos continua a ser uma área em expansão e, nos últimos anos, tem-se cruzado cada vez mais com a área de Inteligência Artificial (IA), quer para auxiliar o jogador sugerindo dicas ou recomendações baseadas nas suas jogadas, quer para tornar o ambiente mais competitivo através da criação de *bots*.

No entanto, *computer vision* (CV) tem sido pouco explorada nesta área. Muitos jogos parecem ter uma IA muito avançada, contudo, na maioria das vezes, é apenas uma ilusão. Normalmente, a IA embutida nos jogos tem uma vantagem relativamente aos jogadores, uma vez que conseguem aceder a todos os estados de jogo possíveis, incluindo aqueles indisponíveis ao oponente humano. CV tem como objetivo analisar e interpretar imagens, e, no mundo dos vídeo jogos, pode analisar e explorar vídeos ou imagens constantes de um determinado jogo e aprender as mecânicas do mesmo, sem qualquer acesso a dados confidenciais do jogo, tal como qualquer outro jogador.

Este projeto teve início com a empresa [Anybrain](#), criada por alumni da Universidade do Minho em 2015. Desde então, utilizam inteligência artificial em prol da sociedade, em especial da indústria de vídeo jogos. É clara a finalidade da empresa no que respeita a produção e análise de vídeo jogos; facto esse que é facilmente perceptível na sua própria dinâmica. Sabe que estes só fazem sentido quando são divertidos, competitivos e, acima de tudo, justos. Acreditam que qualquer pessoa pode jogar, que as comunidades *online* e de *eSports* merecem um bom ambiente e fazem de tudo para o poder melhorar. [Pinto et al. \[2021\]](#)

Os desportos eletrónicos (*eSports*) estão em crescimento a nível mundial, e devido à pandemia provocada pela covid-19, ganharam ainda mais destaque. Existem cada vez mais jogos e torneios, e cada vez mais pessoas aderem a plataformas relacionadas, seja para visualizar torneios, seja para participar neles. Por isso, é fundamental melhorar a experiência tanto de quem participa como de quem assiste.

O principal produto da Anybrain consiste numa plataforma, desenvolvida com técnicas de *machine learning*, que aprende automaticamente, em tempo real, sem qualquer fricção e com mais de 99% de precisão, a maneira como um jogador interage com um jogo. Esta plataforma adapta-se automaticamente a diferentes plataformas e a qualquer jogo e deteta proativamente fraudes e comportamentos anormais num jogador.

Assim sendo, este tema surge com o objetivo de desenvolver um modelo capaz de identificar, em tempo real, o movimento de um determinado objeto num ambiente de um jogo sem acesso a *in-game data*, ou seja, informação do jogo que o utilizador não consegue aceder normalmente. Desta forma, o modelo será totalmente independente do seu desenvolvedor e poderá ser usado para qualquer jogo ou aplicação. Para o seu desenvolvimento serão usadas técnicas de *machine learning* e *computer vision*. Este modelo poderá, no futuro, integrar a plataforma *Anybrain* servindo de base para auxílio de um jogador, deteção de alguma infração cometida pelo mesmo, ou até facilitar no treino de um jogo, visto que permite a monitorização, em tempo real, de jogos de esports.

1.2 Motivação

Durante muito tempo, o Homem sonhou em criar máquinas com características idênticas às humanas, nomeadamente com inteligência para que estas pudessem pensar e agir como humanos. Uma das ideias mais fascinantes era a de dar visão aos computadores, para que estes pudessem interpretar o mundo à sua volta. Todavia, construir máquinas que possam ver e interpretar é um desafio interessante, mas notoriamente complexo de resolver. Para o ser humano, que está constantemente a receber informação visual, reconhecer um rosto ou um determinado objeto parece uma tarefa fácil, contudo, para uma máquina, que tem dados limitados, a simples catalogação de uma imagem é uma tarefa bastante complexa.

Graças aos avanços no poder computacional e nas áreas de inteligência artificial (IA) a ficção tornou-se finalmente realidade. *Computer vision* (CV) tornou-se, finalmente, um sub-campo de destaque de inteligência artificial e, dentro desta área, também a deteção e o rastreamento de objetos têm evoluído constantemente. Para além de detetar um objeto numa imagem estática, rastrear o seu movimento e prever a sua trajetória, em tempo real, são conhecimentos cada vez mais aplicados nas diversas áreas, desde detetar automóveis que tenham cometido infrações, ao auxílio na medicina, passando também pelos vídeo jogos.

Também os vídeo jogos sofreram uma grande popularização nos últimos anos, principalmente com

o auxílio do *streaming*, atraindo milhares de visualizadores a um novo mundo. Passou a ser ainda mais fundamental o conforto tanto para os jogadores como para os visualizadores, havendo um crescimento exponencial de ferramentas ligadas a este fim, ainda mais em conjunto com a inteligência artificial e *machine learning*. Criaram-se *bots*, ferramentas analíticas, plataformas de *streaming*, entre muitas outras aplicações destinadas a jogos.

Com o auxílio de *computer vision* e *machine learning* é possível desenvolver algoritmos que integrem plataformas que ajudem a melhorar a experiência de vídeo jogos, bem como a analisar comportamentos humanos. No final, o mais desafiante é integrar aplicações ou produtos já existentes com tecnologias que recorrem a *computer vision* e, consequentemente, melhorá-los.

A motivação com este trabalho está ligada à possibilidade de desenhar um programa, em parceria com uma empresa que permitiu o envolvimento das minhas capacidades académicas dentro do que era possível. A disponibilidade de todos os recursos que foram facultados pela empresa para o cumprimento da tarefa associado às aprendizagens feitas em contexto académico, sob a orientação de dois formadores que orientaram todo o trabalho no sentido da pesquisa e da auto formação. O desenvolvimento obtido ao longo deste período permitiu perceber que as necessidades do mercado são voláteis e por isso de constante adaptação face aos programas de base existentes nesta mercado. Espero no final desta pesquisa que o trabalho/programa seja concretizável e útil à comunidade dos jogos online.

1.3 Objetivos

O principal propósito desta dissertação é desenvolver um modelo capaz de identificar, em tempo real, o movimento de um determinado objeto num ambiente de um jogo, sem acesso a *in-game data*¹.

Para o seu desenvolvimento, a primeira tarefa a realizar é analisar ferramentas existentes de identificação, deteção e rastreamento de objetos. De seguida, é fundamental perceber e explorar modelos de *machine learning* e *computer vision* passíveis a ser usados.

Após uma primeira fase de análise e exploração dos conceitos base, a tarefa seguinte passa por desenvolver uma aplicação/agente capaz de analisar e recolher a informação do ecrã de um jogador. Seguidamente, através da informação obtida, pretende-se detetar no ecrã do jogador um determinado objeto. Detetado o objeto, é necessário fazer o seu rastreamento de forma a seguir o seu movimento. Estas tarefas devem ser todas feitas em tempo real. Por último, devem-se agregar estes agentes numa só plataforma para facilitar o seu uso a qualquer utilizador. Desta forma será criada uma plataforma onde

¹ informação do jogo que o utilizador não consegue aceder normalmente

seja possível selecionar o objeto que se pretende que seja detetado e rastreado.

Desta forma, os objetivos definidos são:

- Criar uma aplicação/agente que recolha informação contida no ecrã de um jogador em tempo real;
- Criar uma aplicação/agente que detete, em tempo real, um objeto no ecrã do jogador, utilizando a informação recolhida previamente;
- Criar uma aplicação/agente que rastreie o objeto detetado previamente, em tempo real;
- Criar uma plataforma para a exibição da deteção e rastreamento dos objetos.

1.4 Estrutura do documento

O presente documento é relativo à dissertação e encontra-se dividido em cinco capítulos. No primeiro, e atual capítulo, encontra-se a introdução, onde são abordados a contextualização, motivação e objetivos da dissertação realizada.

O segundo capítulo é referente ao estado da arte, no qual são apresentados os conceitos base e essenciais ao desenvolvimento do projeto. Em particular, são abordados os temas de inteligência artificial, *machine learning* e *computer vision*, explorando ainda a biblioteca OpenCV. São ainda referenciados os principais algoritmos de *object detection*, *template matching* e *object tracking*, bem como é feita uma avaliação comparativa da performance destes.

No terceiro capítulo aborda-se todo o processo de desenvolvimento do projeto, começa-se pela realização do levantamento de requisitos da plataforma a desenvolver, prossegue-se para a sua implementação e design. São expostas todas as dificuldades passadas e explicadas detalhadamente todas as soluções encontradas e implementadas para ultrapassar esses problemas. São ainda exploradas todas as ferramentas e tecnologias usadas ao longo do projeto. Por fim, é exibido o protótipo da plataforma, bem como a interface final, onde é feita uma análise de usabilidade desta.

O quarto capítulo é referente aos testes realizados à plataforma e consequentemente análise dos resultados obtidos. Aborda-se um pouco a performance do programa e sugerem-se algumas melhorias possíveis como trabalho futuro que poderiam ser feitas no projeto.

No último capítulo, é efetuado um ponto de situação onde se apresentam as conclusões sobre o desenvolvimento do problema em estudo. Ainda nesse capítulo, é delineado um possível trabalho futuro para o projeto realizado.

Capítulo 2

Estado da arte

2.1 Inteligência Artificial

Artificial Intelligence, ou em português Inteligência Artificial (IA), é o resultado de uma procura contínua com o objetivo de criar máquinas que façam tudo o que os humanos conseguem, desde ouvir, ver, perceber, pensar e até exibir emoções.

Esta ideia surgiu em 1950, quando o cientista da computação Alan Turing propôs um teste para avaliar a inteligência de uma máquina: "*If a machine can trick humans into thinking it is human, so then it has intelligence.*", sugeriu Alan Turing [Madjour and Madjour \[2018\]](#). Esta ideia manteve-se até 1995, altura em que se realizou a primeira conferência sobre *Artificial Intelligence*, onde se introduziu pela primeira vez este termo. McCarthy, juntamente com Alan Turing, Allen Newell, Herbert A. Simon e Marvin Minsky, ficaram assim conhecidos como os pais fundadores da AI [Ertel \[2018\]](#). Desde aí desenvolveram-se máquinas quer com o intuito de competição, começando com o Deep Blue, que em 1997 derrotou o campeão mundial de xadrez Garry Kasparov até 2017, quando o AlphaGo venceu o campeão mundial Ke Jie no complexo jogo de tabuleiro Go, quer com o intuito de auxiliar o ser humano nas tarefas mais mundanas, desde a Roomba que foi o primeiro aspirador robótico que aprendeu a navegar e limpar casas, até à Siri e Alexa, assistentes virtuais com uma interface de voz. [Righetti et al. \[2020\]](#)

A Inteligência Artificial tem sub-campos em várias áreas como processamento de linguagem natural (PLN), robótica, aprendizagem máquina e aprendizagem profunda (Machine Learning (ML) e Deep Learning (DL)), sistemas especialistas, fala ou reconhecimento de voz, automação inteligente e visão por computadores (VC). [Gollapudi \[2019\]](#)

Machine Learning (ML) é um tipo de inteligência artificial que permite criar máquinas capazes de desenvolver inteligência, uma vez que, utilizando apenas a sua própria experiência, isto é, dados históricos, são capazes de desenvolver um algoritmo de previsão de resultados sem serem explicitamente programa-

das para tal. Quanto maior os dados que o algoritmo conseguir ter acesso, melhor é a certeza e precisão destes algoritmos. [Monteiro et al. \[2021\]](#)

O objetivo destes algoritmos é produzir um resultado na forma de uma regra que possa ser generalizado. *Machine Learning* é caracterizada pela forma como os algoritmos aprendem e prevêem, sendo normalmente subdivididos em quatro abordagens básicas [Ayodele \[2010\]](#):

- aprendizagem supervisionada/*supervised learning*, visto que usa dados catalogados para a previsão dos resultados;
- aprendizagem não supervisionada/*unsupervised learning*, uma vez que não é fornecido ao algoritmo qualquer dado catalogado ou rotulado;
- aprendizagem semi-supervisionada/*semi-supervised learning*, combinação das duas primeiras abordagens, ou seja, apenas parte dos dados estão catalogados;
- aprendizagem por reforços/*reinforcement learning*, baseado em recompensar comportamentos desejados e/ou punir comportamentos indesejados.

Alguns exemplos bastante comuns da utilização de algoritmos de *Machine Learning* são mecanismos de recomendação, deteção de fraude, filtragem de spam, deteção de ameaças de *malware*, entre outros. Há estudos que vão mais longe e implementam estas técnicas no auxílio da autenticação contínua de dispositivos eletrônicos [Rocha et al. \[2021\]](#), ou ainda na análise da performance de um trabalhador na sua área de trabalho [Carneiro et al. \[2017\]](#), através da análise da fatiga deste [Pimenta et al. \[2016\]](#).

2.2 Visão por Computador

Computer vision (CV) é o campo de estudo que procura desenvolver técnicas que permite computadores, dispositivos ou máquinas em geral verem, compreenderem e interpretarem conteúdo de dados visuais digitais, como fotografias, vídeos ou até mesmo uma representação gráfica de um local ou de um mapa de intensidade de calor.

Embora o auge da visão por computador tenha sido recente, não é uma nova área científica. Um dos primeiros e mais importantes artigos que influenciaram *computer vision* foi publicado no final dos anos 1950 por dois neuro-fisiologistas. O artigo intitulado "Receptive fields of single neurons in the cat's striate cortex"([Hubel and Wiesel \[1959\]](#)) descreve as propriedade da resposta dos neurónios do córtex visual de um gato, chegando à conclusão que existem neurónios simples e complexos no córtex visual primário e que o processamento visual começa com estruturas simples como bordas e linhas.

Apesar deste artigo não ter nada a ver com a área de engenharia ou teste de softwares, foi essencial para se criar a estrutura base dos algoritmos, uma vez que *computer vision* funciona da mesma maneira que a visão humana. Similarmente a um humano a tentar decifrar uma imagem distante, CV primeiro discerne arestas e formas simples e, só de seguida, é que completa a informação das suas previsões depois de várias iterações. [Demush \[2019\]](#)

Após a publicação deste artigo, os próximos marcos na história da CV foram a invenção de um scanner digital de imagens, que permitiu aos computadores digitalizarem e adquirirem imagens, e ainda a transformação de imagens em matrizes de números.

Só após estas ferramentas base terem sido descobertas e desenvolvidas é que se pôde avançar com a implementação de algoritmos de *computer vision*. Esta tecnologia necessita de analisar inúmeros dados repetidamente, para conseguir gradualmente distinguir pequenas diferenças até conseguir reconhecer imagens. *Computer vision* implementa técnicas de *machine learning*, *deep learning*, *convolutional neural network* (CNN) ou *recurrent neural network* (RNN) e, nalguns casos específicos, implementa técnicas de processamento de linguagem natural para a análise de texto extraído de imagens. A CNN é usada para interpretar uma só imagem, enquanto que a RNN é usada, de maneira semelhante, para relacionar imagens em vídeo.

Com os avanços na área de *deep learning*, a construção de métodos para classificar imagens, detectar objetos, fazer rastreamento e manipulação de imagens, tornou-se cada vez mais simples e precisa. Hoje em dia, existem inúmeras aplicações que utilizam *computer vision*, desde aplicações que permitem encontrar um objeto ou uma pessoa num vídeo, compreender movimento e padrões num vídeo, até aumentar ou diminuir o tamanho, brilho ou nitidez de uma imagem. [Demush \[2019\]](#)

2.2.1 Desafios da Visão por Computador

Computer vision parece um problema fácil de resolver, uma vez que até crianças e mesmo animais conseguem ver e interpretar o mundo que os rodeia. Todavia, permanece um problema por resolver, em parte devido à limitada compreensão da visão biológica e em função da complexidade da percepção da visão relativamente a um mundo físico dinâmico e infinitamente variável.

Analisar uma simples imagem digital pode ser bastante complicado, uma vez que esta pode apresentar diversa informação e conteúdo. Uma imagem pode retratar um simples objeto, possuir uma descrição, relatar minuciosamente um modelo multidimensional, entre outros. A interpretação destas imagens influencia a precisão dos mecanismos de *computer vision*. [Brownlee \[2019\]](#)

Para além disso, os dados visuais digitais podem ser adquiridos através de várias fontes, como web-

cams, câmaras fotográficas ou de filmagem, gravadores de vídeo, scanners, entre outros.

De seguida, é exibida uma série de imagens problemáticas que tornam o processo da interpretação de imagens complexo:

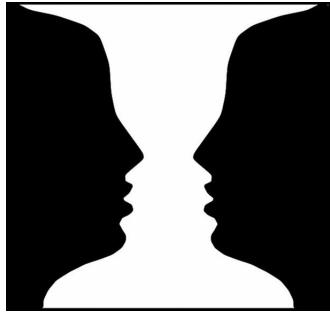


Figura 1: Ilusão: vaso ou rosto? [Castelli \[2019\]](#)

Tal como se pode observar na figura 1, existem imagens que até para o olho humano podem trazer problemas, visto que depende da perspetiva da pessoa. A imagem tanto pode representada um vaso como duas caras de perfil.

Podem ainda existir problemas com a resolução e qualidade de uma imagem, dificultando o foco de um objeto e a sua interpretação. Este problema surge principalmente em imagens com um fundo pouco contrastante em relação ao objeto principal, como por exemplo, em imagens com pouca iluminação, onde se torna difícil de visualizar o pretendido.

Por último, as imagens da figura 2 são todas relativas a cães, todavia parecem totalmente diferente umas das outras, isto deve-se ao facto de haver várias raças de animais (sub-figura 2(a)), serem imagens de diferentes ângulos (sub-figuras 2(b), 2(c)) e caso um objeto esteja em movimento, pode apresentar uma forma ligeiramente diferente, como é o caso do cão a correr (sub-figura 2(d)).



(a) Diferentes raças de cães [Lesser \[2022\]](#)



(b) Cão de costas [Teiga \[2016\]](#)



(c) Cão de pernas para o ar [White \[2021\]](#)



(d) Cão a correr [blo \[2021\]](#)

Figura 2: Diversas figuras de cães

Para o ser humano, que cresceu a ver estas diferentes variações e está constantemente a receber informação visual, a catalogação destas imagens parece trivial, contudo para uma máquina, que tem dados limitados e pontuais, esta catalogação é bastante complexa.

2.2.2 Aplicações existentes no mercado

Computer vision está em constante uso e progresso e, como tal, existem cada vez mais áreas e aplicações que utilizam esta tecnologia. Desde o retail até à segurança, saúde, agricultura e automação industrial, são algumas das áreas industriais que mais têm beneficiado com os avanços tecnológicos. [Meel \[2021\]](#)

Indústria automóvel

O recente desenvolvimento dos carros autónomos trouxe uma revolução na forma como a inteligência pode ser incorporada nos automóveis para gerir congestionamentos, acidentes rodoviários e manutenção pró-ativa dos veículos. Os veículos são equipados com sensores e câmaras que conseguem recolher imagens e dados sobre o ambiente envolvente. Com isto, os veículos tornam-se capazes de detetar sinalização na via como sinais de limite de velocidade, avisar o condutor quando este se encontra estacionado numa zona proibida, encontrar um lugar de estacionamento livre e até dar indicações ao condutor para

chegar a um determinado local ou até identificar obstáculos perigosos no meio da estrada.

Para além disso, tem havido um uso intensivo de tecnologias de *computer vision* em cidades inteligentes para a identificação de veículos que cometem infrações como excesso de velocidade, desobedecer um sinal vermelho num semáforo ou um sinal de STOP, andar em contra-mão, entre outros. Esta tecnologia também pode ser utilizada para detetar a ocupação de um parque de estacionamento utilizando apenas as câmaras de vigilância. [Meel \[2021\]](#)

Indústria biomédica e da saúde

Similarmente à indústria automóvel, a indústria da saúde tem adotado cada vez mais técnicas de *machine learning* e *computer vision* para auxiliar e facilitar muitos dos processos envolventes na análise de um paciente. Através do reconhecimento de imagens é possível detetar ligeiras diferenças entre imagens cancerígenas e não cancerígenas, auxiliando o médico a diagnosticar precocemente tumores e cancro através de dados de exames de ressonância magnética. É ainda possível distinguir se o cancro é maligno ou benigno permitindo um tratamento muito mais personalizado e eficiente. Da mesma forma, armazenar os dados de análise e obter percepções de relatórios digitais de saúde de pacientes pode melhorar significativamente a precisão e eficácia dos tratamentos.

Ademais, existem robôs capazes de realizar cirurgias complexas com precisão e eficiência. Em casos de cirurgias, é ainda possível prever a quantidade de perda sanguínea, evitando transfusões de sangue desnecessárias. [Meel \[2021\]](#)

Com a situação pandémica devida ao coronavírus, o reconhecimento do uso de máscara e equipamento de proteção tornou-se essencial e indispensável para limitar a disseminação do vírus. Por esse motivo, empresas privadas como a Uber implementaram um mecanismo de reconhecimento facial especialmente para os motoristas e estafetas. Este mecanismo obrigou os trabalhadores a tirarem uma selfie com a máscara antes de começarem o seu percurso. Assim, é confirmado que os todos os funcionários cumprem as normas de segurança tornando o transporte muito mais seguro nesses tempos pandémicos. [Nunes \[2020\]](#).

Agricultura

A monitorização dos animais com *computer vision* e *machine learning* é fundamental para a agricultura. Os sistemas de visão inteligente têm como objetivo analisar o comportamento animal e aumentar a produtividade, saúde e bem-estar dos animais e, deste modo, influenciar a produção e os benefícios económicos da indústria. Para tal, são utilizadas câmaras para vigiar e monitorizar a saúde de animais específicos, isto é, porcos, gado ou aves, conseguindo assim uma análise muito mais personalizada.

Por outro lado, aplicações de *computer vision* permitem ainda acompanhar o crescimento de plantas

de forma contínua e não destrutiva, possibilitando o exame detalhado das necessidades nutricionais das plantas. Em comparação com as operações manuais, é possível detetar mudanças subtils devido à desnutrição e muito mais precocemente, fornecendo uma base confiável e precisa para uma regulação atempada.

É possível ainda identificar áreas menos férteis em termos de crescimento e áreas áridas como também reconhecer rapidamente e precisamente insetos e contá-los com o objetivo de controlar e prevenir eventuais pragas. [Meel \[2021\]](#)

Indústria do mercado de revenda

Tanto lojas físicas como virtuais conseguem utilizar técnicas de *machine learning* e *computer vision* para facilitar e melhorar a experiência dos clientes. Um exemplo bastante conhecido é a Amazon, que implementou *computer vision* para identificar produtos visualmente semelhantes com preços ligeiramente diferentes e sugere artigos baseados com a comparação entre gostos de diferentes clientes. Para além disso, aconselha os vendedores acerca do posicionamento de um determinado produto.

Relativamente a lojas físicas é possível, utilizando câmaras de vigilância previamente instaladas, contar pessoas anonimamente e analisar o tempo médio gasto destas em diferentes áreas, o tempo de espera em filas e avaliar metodicamente a qualidade do serviço. Esta análise tem como objetivo melhorar a distribuição e organização da loja, detetar e evitar longas filas e contar o número de clientes na loja, para impedir que o limite máximo seja atingido. Como não podia deixar de ser, com a pandemia covid-19, foi ainda necessário ter em conta o distanciamento entre pessoas. Para isso, foram utilizados detetores de distância para medir o distanciamento entre clientes e funcionários. [Meel \[2021\]](#)

Desporto

Por último, sendo a área desportiva bastante popular a nível mundial é também uma área de interesse para os algoritmos de *computer vision*. Através de vídeos de jogos de equipas é possível calcular e determinar a pose e o movimento dos diversos jogadores, bem como analisar o desempenho, postura, identificar pontos fracos e melhorar os potenciais de cada jogador. Este estudo personalizado permite auxiliar os treinadores e os jogadores a analisarem os jogos de forma rápida e concisa, permitindo realizar esta análise entre jogos ou partes de um mesmo jogo.

Em contrapartida, os sistemas de auto-treinamento também ganharam destaque. Sendo possível corrigir pequenos defeitos de postura e desempenho, é possível criar um plano de treino personalizado e progredir até certo ponto sem a ajuda de outrem. [Meel \[2021\]](#)

2.2.3 OpenCV

OpenCV significa "open source computer vision" e é uma biblioteca construída para fornecer suporte a aplicações de *computer vision* e *machine learning*, usado tanto a nível académico como comercial. Contém um leque de funções de processamento de imagens de baixo nível e algoritmos de alto nível, como deteção de rosto, deteção e correspondência de características e rastreamento de objetos. [ope \[a,b\]](#)

Foi uma iniciativa que começou no *Intel Research Lab* com o intuito de agilizar todo o processo de percepção da máquina e possibilitar o uso intensivo de CPU (Central Processing Unit). Foi concebido como uma forma de disponibilizar universalmente a infraestrutura de *computer vision* e como tal possui mais de 2500 algoritmos otimizados, com um conjunto abrangente dos mais recentes algoritmos.

Em 2010 foi adicionado ao OpenCV um novo módulo que ainda se encontra em desenvolvimento. Este módulo permite a utilização do GPU (Graphics Processing Unit), sem a necessidade de programação direta a nível do GPU. Os utilizadores podem assim tentar combinar simultaneamente o processamento do GPU com os cálculos do CPU, com o objetivo de minimizar as sobrecargas de transferência de dados e aumentar o desempenho geral. [ope \[d\]](#)

OpenCV suporta as linguagens de C/C++, Python, MATLAB e Java e pode ser usada para construir aplicações de *computer vision* para qualquer sistema operativo tanto para computadores como para dispositivos móveis, isto é, suporta Windows, Linux, macOS, Android e iOS. [ope \[b\]](#)

2.3 Deteção e Reconhecimento de Objetos

A deteção de objetos é um dos problemas fundamentais de *computer vision*, servindo de base para muitas áreas como *image captioning*, *instance segmentation*, *object tracking* entre outros. A deteção de objetos, em conjunto com classificação e localização de objetos, serve para determinar a posição de um ou mais objetos numa determinada imagem digital e, seguidamente, catalogar cada objeto encontrado, permitindo responder a questões como "Que objetos estão onde?".

2.3.1 Imagens e as suas características

Ao contrário de nós, humanos, os computadores vêem uma imagem como uma matriz 2D, ou seja, as imagens digitais são constituídas por uma matriz de pixels. O Pixel é a menor unidade que compõe uma imagem e cada imagem é composta por um conjunto de pixels, onde o valor de cada um representa

a intensidade da luz naquele determinado local da imagem.

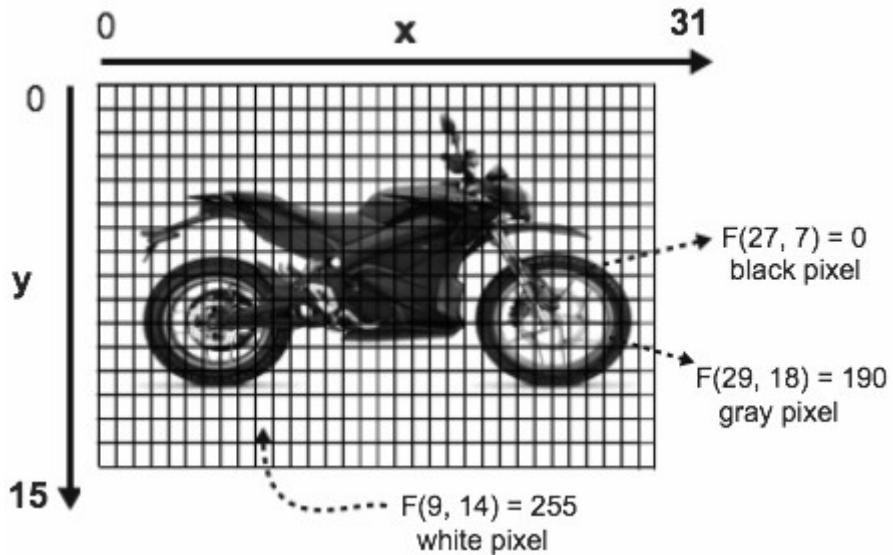


Figura 3: Matriz de uma imagem em tons de cinza [Elgendi \[2020\]](#)

A imagem 3 representada acima tem um tamanho de 32×16 , o que significa que as dimensões da imagem são 32 pixels de largura e 16 pixels de altura. No total, a imagem possui $32 \times 16 = 512$ pixels. Nesta imagem a preto e branco, cada pixel contém um valor que, como já foi referido, representa a intensidade da luz no pixel específico. Este valor pode variar entre 0 e 255, onde o 0 representa a cor preta e o 255 a cor branca. Todos os valores entre estes dois números representam um tom de cinza, onde, quanto mais baixo for o valor, mais escuro é a sua tonalidade. Todas as imagens que são guardadas neste formato também são chamadas de canal (*channel*). [Elgendi \[2020\]](#)

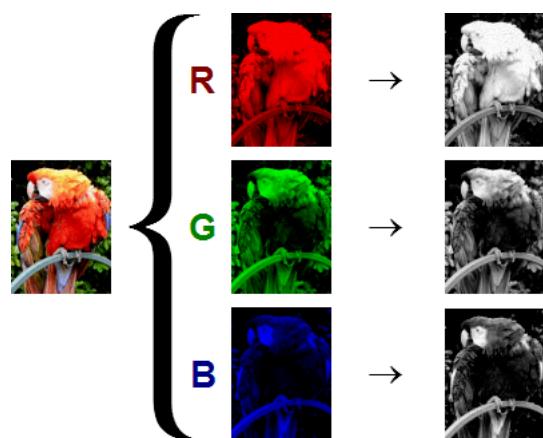


Figura 4: Junção das 3 matrizes RGB [Visalpara et al. \[2020\]](#)

Para imagens a cores, em vez de estas serem representadas por uma única matriz, são representadas por três matrizes: uma que representa a intensidade do vermelho, uma para o verde e outra para o azul,

ou seja, uma por cada tonalidade RGB (red, green blue), tal como se pode ver na imagem 4. Cada uma destas matrizes têm novamente valores que variam de 0 a 255, onde cada um desses números representa a intensidade dos pixels, isto é, os tons de vermelho, verde e azul, respetivamente. No final, estas matrizes são sobrepostas para que a imagem apresente todas as cores possíveis.

Assim, uma imagem digital a cores, quando vista por um computador, tem a forma $N \times M \times 3$, em que N é o número de pixels de altura, M é o número de pixels de largura e 3 representa o número de canais, sendo cada um dos canais para cada uma das cores vermelho, verde e azul. Desta forma, de modo similar às imagens com tons de cinza, as imagens coloridas guardadas neste formato são chamadas de três canais (*Three-Channel*). [Singh \[2021\]](#)

2.3.2 Convolutional Neural Network (CNN)

Convolutional neural networks (CNN) são um tipo de *artificial neural networks* (ANN) e são constituídas por neurónios, pesos e *biases*. Fornecem uma abordagem escalável para a classificação de imagens e tarefas de reconhecimento de objetos, aproveitando os princípios da álgebra linear, especificamente, a multiplicação de matrizes, para identificar padrões dentro de uma imagem. Todavia, requerem poder computacional bastante elevado, sendo muitas vezes necessário treinar modelos utilizando processamento gráfico (GPUs).

Uma ANN contém uma camada de input, uma ou mais camadas ocultas (*hidden layers*) e uma camada de output, como se pode ver na imagem 5. Cada camada é constituída por neurónios, e cada neurónio liga-se a pelo menos um da camada seguinte, passando-lhe informação. Cada neurónio possui ainda um peso associado que varia consoante a informação recebida. Se este peso estiver acima de um determinado valor de *threshold*, então o neurónio será ativado e enviará informação para a camada seguinte. Caso contrário, nenhuma informação será enviada para a próxima camada. [Education \[2020b\]](#)

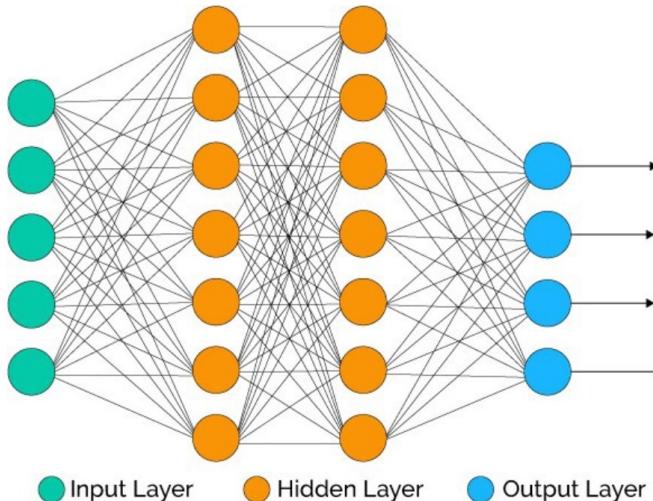


Figura 5: Artificial Neural Network Medisetti [2021]

A principal diferença entre uma CNN e uma ANN é que uma CNN recebe uma imagem como input. Sabendo o seu input é possível otimizar a rede. Assim, uma CNN apresenta três camadas principais: *convolutional layer*, *pooling layer* e *fully-connected layer*.

A **camada convolucional** (*convolutional layer*) é a camada principal que dá o nome à rede e que requer maior poder computacional. Recebe como parâmetros os dados relativos a uma imagem e um filtro, que consiste normalmente numa matriz 3X3, e devolve como output final um mapa de ativação (também chamado de *feature map*, *activation map* ou *convolved feature*). Nesta camada, o filtro é então aplicado a uma região da imagem e é calculado o produto escalar entre os pixels da imagem recebida no input e o filtro. O resultado final é exibido no output, tal como sugere a imagem abaixo 6. Depois, o filtro desloca-se, normalmente uma unidade, repetindo o processo até que toda a imagem seja percorrida. O output final é então o conjunto dos produtos escalares da imagem e do filtro.

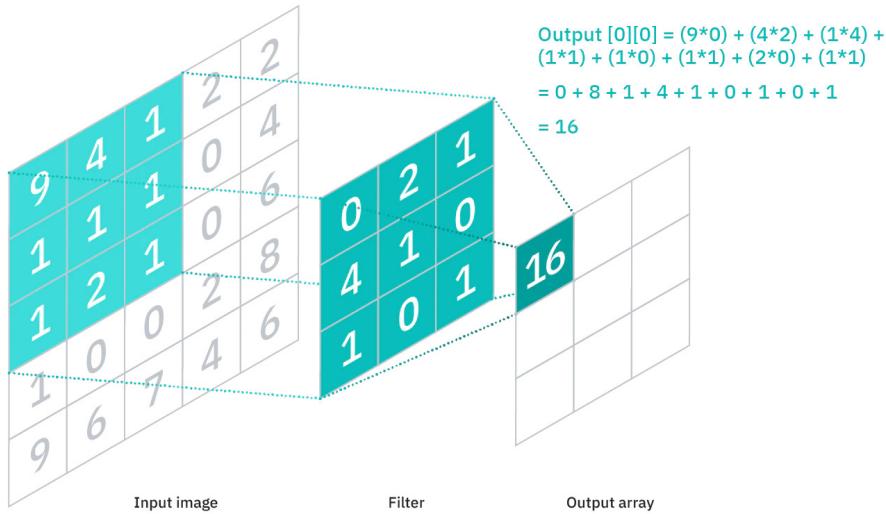


Figura 6: *Convolutional layer* Education [2020a]

Após cada operação de convolução, a CNN aplica uma transformação de Unidade Linear Retificada (*Rectified Linear Unit - ReLU*) ao mapa de ativação, para introduzir não linearidade no modelo. O número de camadas de convolução não é limitado, podendo seguir outra camada após a camada de convolução inicial. Em cada operação de convolução são aplicados diferentes filtro, dependendo do que se pretende reconhecer.

Na **camada de Pooling** (agrupamento) é realizada uma redução ao número de parâmetros de input, sendo este normalmente o mapa de ativação devolvido pela camada de convolução. Nesta camada é aplicada uma função de agregação aos valores recebidos apresentando os resultados na forma de matriz. Há dois tipos principais de *pooling* (comprovado pela figura 7):

- Max pooling: seleciona o pixel com o maior valor (mais usado)
- Average pooling: calcula o valor médio dos pixels dentro de cada campo

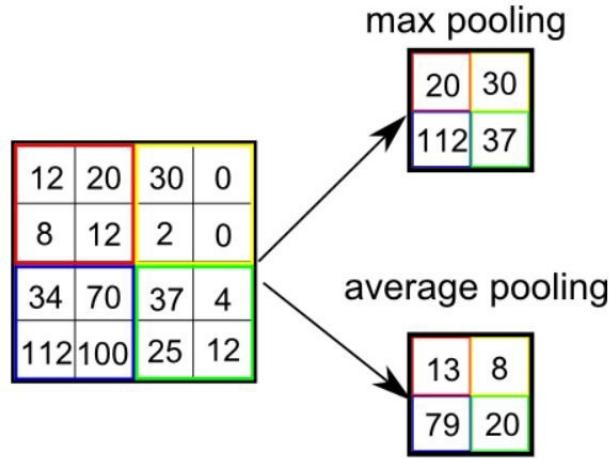


Figura 7: Tipos de Pooling [Saha \[2018\]](#)

Apesar de se perder muita informação nesta camada, os benefícios compensam, pois permite reduzir a complexidade, melhorar a eficiência e limita o risco de *overfitting* (sobreajuste).

Na **full-connected layer**, tal como o nome indica, os neurónios estão totalmente ligados, ou seja, cada neurónio na camada de output está diretamente ligado a todos os neurónios da camada anterior. É nesta camada que se realiza a tarefa de classificação com base nas características extraídas das camadas anteriores. O resultado final apresentado é uma probabilidade entre 0 e 1 relativo à certeza que o algoritmo tem sobre a classificação da imagem.

Na figura 8 é possível visualizar a junção das várias camadas. Assim, para a análise de uma só imagem utilizam-se várias camadas de convolução e *pooling* consecutivamente de modo a extrair informação da imagem. E, por último, na camada totalmente conectada, é que se faz a classificação da imagem, devolvendo, como output, a probabilidade para cada uma das possíveis opções.

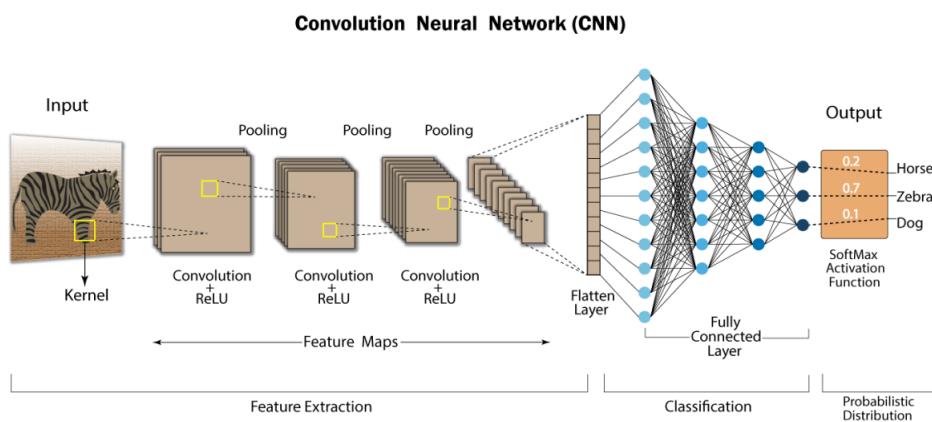


Figura 8: *Convolutional Neural Network* [Swapna](#)

2.3.3 Algoritmos de Detecção de Objetos

Um algoritmo de detecção de objetos tem como principal objetivo analisar uma imagem e determinar a área onde um objeto se encontra e classificá-lo. Assim, após detetada a região onde o objeto está, é possível, através da utilização de uma CNN, classificar o objeto detetado.

Existem alguns algoritmos de detecção de objetos que utilizam uma rede CNN para analisarem as imagens recebidas, sendo apresentados de seguida os principais.

Region-based Convolutional Neural Networks (R-CNN)

Nos modelos de redes neuronais convolucionais baseadas em região (R-CNN), a imagem é primeiro dividida em regiões (cerca de duas mil regiões) com o objetivo de extrair os recursos essenciais. Este processo de seleção das características mais significativas pode ser computado com a ajuda de um algoritmo de busca seletiva que alcança os recursos regionais mais importantes, de forma a garantir que as sub-segmentações escolhidas sejam as ideais. Após a conclusão do algoritmo de busca seletiva e dividida a imagem em regiões, são extraídos os recursos e realizadas previsões, aplicando uma rede neuronal convolucional a cada uma dessas regiões. A etapa final do R-CNN é fazer as previsões apropriadas para a imagem total e catalogar cada região delimitadora detetada. [K \[2021\]](#) Este processo é apresentado na imagem 9.

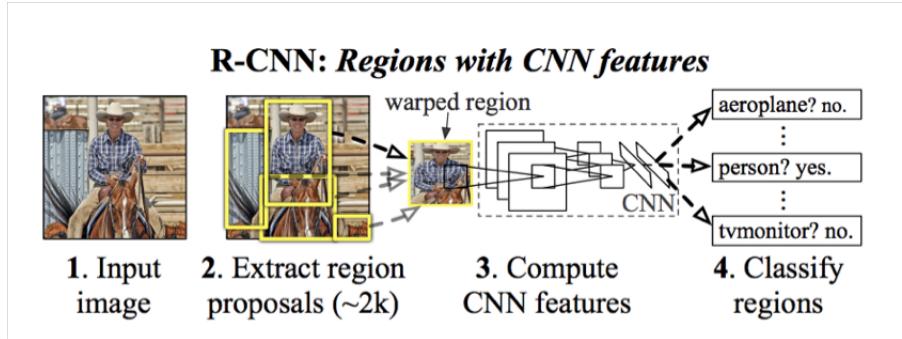


Figura 9: *R-CNN: Regions with CNN features* [Gandhi \[2018\]](#)

Apesar de produzir resultados corretos, o procedimento geral de extração de todas as regiões propostas é extremamente lento. Na globalidade, o modelo R-CNN apresenta não só uma baixa taxa de treino, como também um elevado tempo de previsão.

Em 2015, o *Fast R-CNN* foi desenvolvido com a intenção de reduzir significativamente este tempo de treino. Enquanto o R-CNN original calcula independentemente os recursos da rede neuronal em cada uma das regiões, o *Fast R-CNN* executa a rede neuronal uma única vez para toda a imagem. Existem outras

versões do modelo R-CNN e, a que apresenta melhor desempenho a nível de velocidade e *accuracy*, é o modelo *Faster-RCNN*, que devolve o resultado ótimo em cerca de 0.2 segundos. [Girshick \[2015\]](#)

Single Shot Detector (SSD)

Single Shot Detector (SSD) é um método que deteta e classifica múltiplos objetos numa imagem usando apenas uma única rede neuronal profunda. Começa-se por diferenciar o output através de caixas delimitadoras *default* de diferentes tamanhos e proporções. De seguida, prevê-se a categoria do objeto, combinando múltiplas características para permitir a catalogação de objetos de vários tamanhos.

Este método é relativamente fácil de treinar e integrar em sistemas que requerem uma componente de deteção. Comparativamente a outros métodos de previsão de objetos com caixas delimitadoras, o SSD apresenta melhor valores de *accuracy* mesmo para imagens menores. [Liu et al. \[2016\]](#)

You Only Look Once (YOLO)

Ao contrário da maior parte dos algoritmos de deteção de objetos, que utiliza regiões para localizar um objeto dentro de uma imagem, You Only Look Once (YOLO) é um sistema que, em tempo real, utiliza uma única rede neuronal para analisar a imagem toda de uma única vez, similarmente a SSD.

YOLO utiliza uma única rede convolucional para simultaneamente prever e classificar múltiplas caixas delimitadoras, como se vê na figura 10. E, uma vez que este processo é realizado num único passo, é cerca de mil vezes mais rápido que R-CNN e cem vezes que o Fast R-CNN. Comparativamente a SSD, apresenta valores exatos ligeiramente piores, mas ultrapassa na velocidade. O modelo base processa imagens, em tempo real, a 45 fps (frames por segundo), enquanto que as versões mais recentes ultrapassam os 155 fps com uma exatidão bastante elevada comparativamente a outros métodos. [Brownlee \[2021\]](#)

A maior desvantagem do algoritmo YOLO é que este exibe algumas dificuldades a detetar objetos relativamente pequenos, devido às restrições espaciais do algoritmo. [Redmon et al. \[2016\]](#)

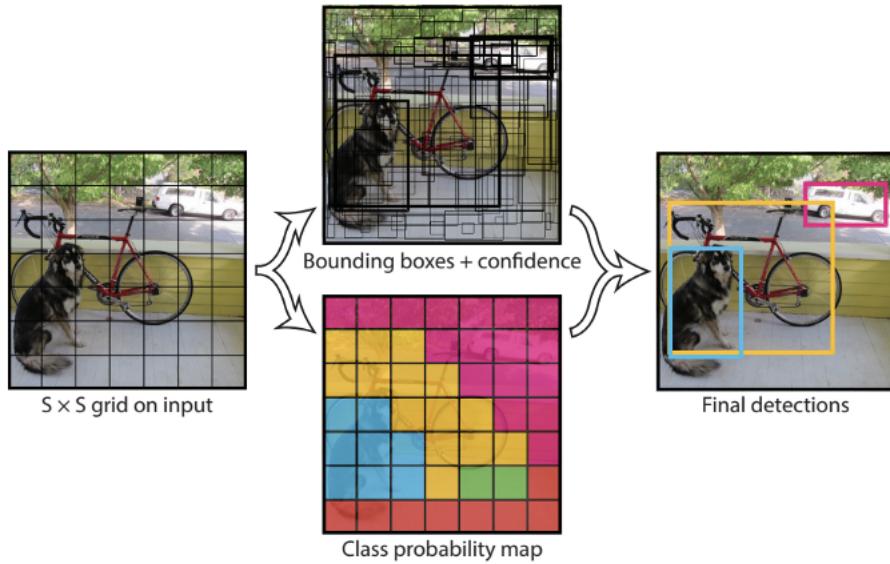


Figura 10: You Only Look Once (YOLO) [Gandhi \[2018\]](#)

Comparação entre os vários algoritmos

Microsoft's Common Objects in Context (COCO) coc [b] é um dataset usado para deteção, segmentação e catalogação de objetos em grande escala. É utilizado para avaliar o desempenho de métodos de *computer vision* e é interpretado automaticamente em tempo real.

O *dataset* apresenta um formato específico em JSON que dita como é que os rótulos e os metadados devem ser guardados numa imagem. Contém mais de 80 objetos de diferentes categorias, com as respetivas caixas delimitadoras.

A métrica de avaliação primária para modelos de deteção de objetos é o valor médio das precisões médias de cada classe (mean average precision - MAP). Esta avaliação tem em conta caixas delimitadoras pequenas, médias e grandes, bem como limites variados para o número de deteções por imagem. É possível ainda analisar os vários modelos tendo em conta os fps e o tempo que demoraram a detetar os vários objetos. [Yohanandan \[2020\]](#)

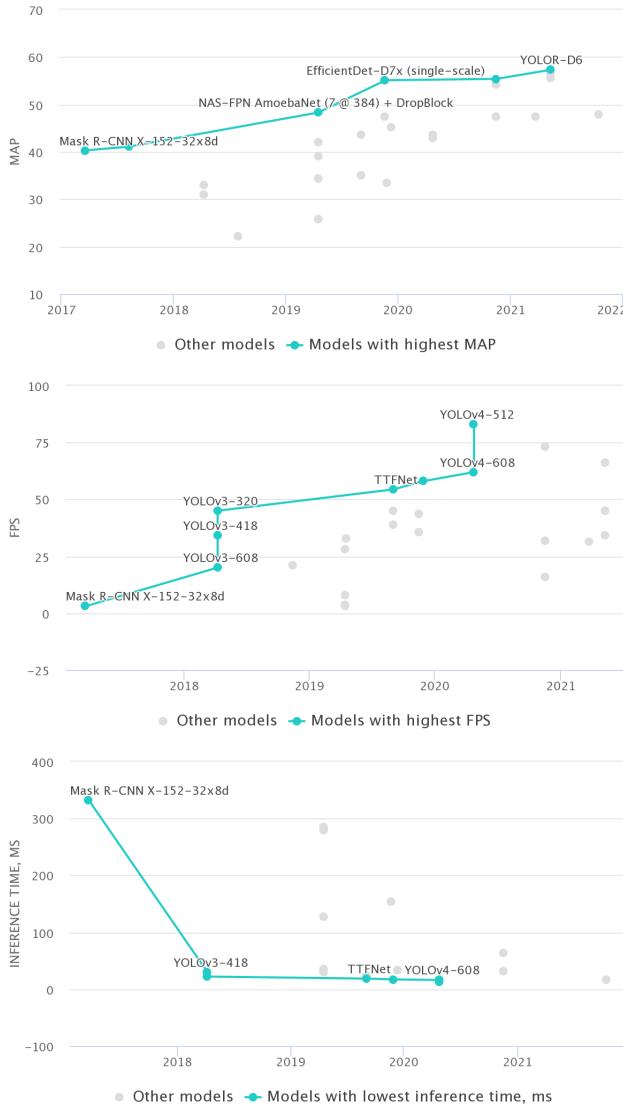


Figura 11: Real-Time Object Detection on COCO [coc \[a\]](#)

Segundo a plataforma COCO, dos algoritmos apresentados anteriormente e tal como se pode ver nos gráficos da figura 11, YOLO é o que apresenta melhor performance. Existem várias versões do mesmo algoritmo, e o melhor modelo, relativamente a MAP, é YOLOR-D6 com 57.3 MAP para 34 fps. O que apresenta maiores valores de fps, bem como menor tempo de inferência é o modelo YOLOv4-512 com 43 MAP, 83 fps e 12ms. Contudo, temos de ter em conta, que este modelo apresenta valor de treino extra comparativamente aos outros algoritmos e daí obter valores melhorados.

2.3.4 Template Matching

Template Matching é um método de procura cujo objetivo é encontrar a localização de uma imagem modelo numa imagem maior. É considerado um método básico de deteção de objetos, uma vez que é

capaz de detetar a localização de um objeto específico fornecido através de uma imagem como input. Para isso, é usado um processo de mover a imagem modelo sobre toda a imagem de forma a calcular a semelhança entre o imagem modelo e a área coberta da imagem.

Template Matching é implementado por meio de convolução bidimensional. Na convolução, o valor de um pixel de saída é calculado multiplicando os elementos de duas matrizes e somando os resultados. No *template matching*, uma dessas matrizes representa a própria imagem, enquanto a outra matriz, que seria o filtro, é a imagem modelo. Assim, se na área selecionada houver correspondência com o filtro da imagem modelo, significa que foi detetado o objeto/padrão pretendido. [X. Binjie \[2008\]](#), [Woodham et al.](#)

Para a utilização deste método são necessários dois componentes principais:

- **Imagen original:** imagem na qual se pretende encontrar a imagem modelo;
- **Imagen modelo:** imagem que serve de comparação (filtro) para procurar um padrão na imagem original.

Após percorrida a imagem toda, consegue-se detetar a área da imagem com maior correspondência com a imagem modelo, como acontece na figura 12.

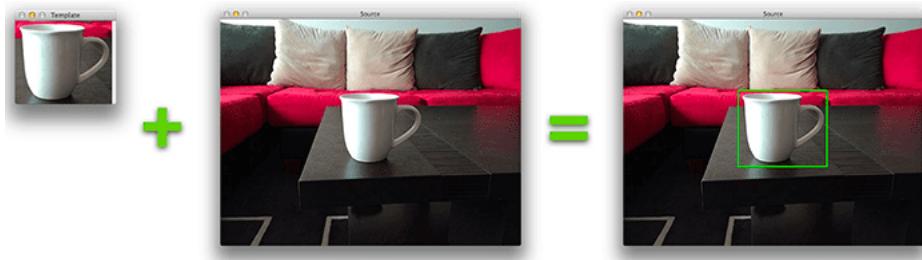


Figura 12: Exemplo ilustrativo de *Template Matching* [Rosebrock \[2021a\]](#)

Para identificar a área correspondente, é necessário comparar a imagem modelo com a imagem original deslizando-a ao longo desta, ou seja, move-se a imagem modelo um pixel de cada vez, da esquerda para a direita, de cima para baixo, como ilustra a imagem 13.

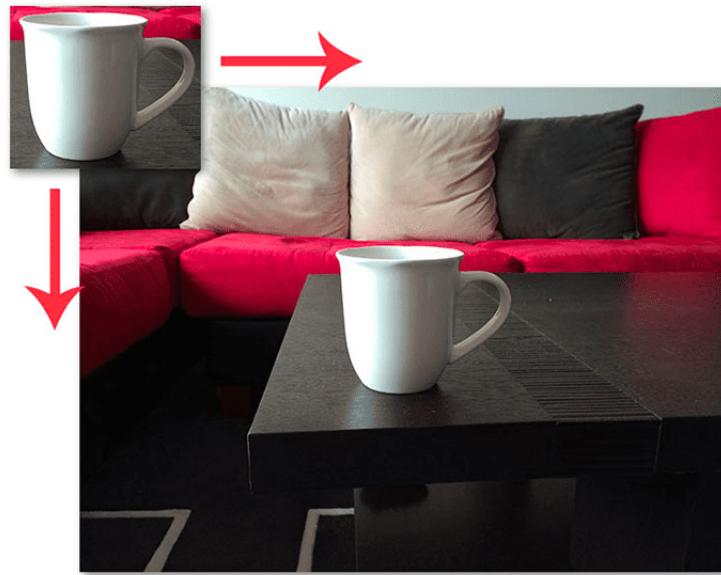


Figura 13: Exemplo da deslocação da imagem modelo [Rosebrock \[2021a\]](#)

Em cada local, é calculada um mapa de correlação que representa a semelhança entre a área específica da imagem original com a imagem modelo. Este mapa é uma imagem a preto e branco que simboliza a semelhança entre as duas imagens, como se visualiza na imagem 14. Quanto mais elevada for a correspondência com a imagem modelo, mais claro será o local na imagem resultante. Esta imagem é guardada em forma de matriz de maneira a facilmente se detetarem os principais pontos de semelhança. [Rosebrock \[2021a\]](#)

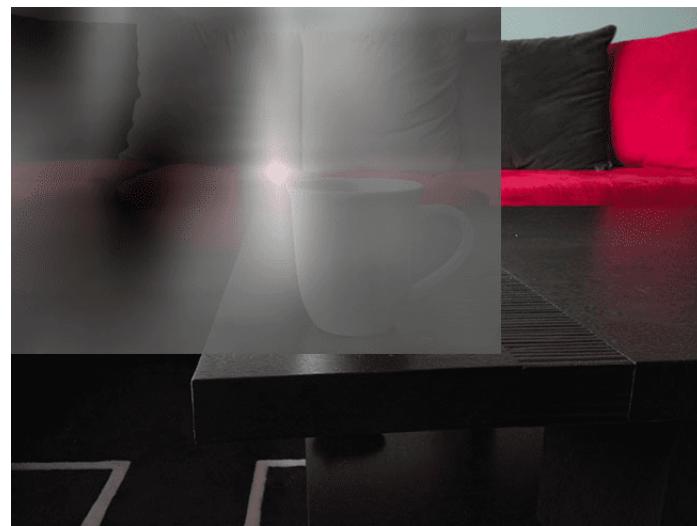


Figura 14: Matriz Resultante da semelhança com a imagem modelo [Rosebrock \[2021a\]](#)

A localização marcada pelo ponto branco representa o local com maior correspondência, de modo que o retângulo formado a partir desse ponto (canto superior esquerdo do retângulo) e dimensões iguais

à imagem modelo será considerado o local de maior semelhança com a imagem modelo.

Template matching é conhecido por ser uma operação cara no que toca à classificação de grandes conjuntos de imagens, uma vez que, para cada uma dessas imagens, é necessário percorrer toda a imagem original, levando a que o tempo de execução seja proporcional ao tamanho do conjunto das imagens. De forma idêntica, detetar numa só imagem vários objetos diferentes também exige elevado poder computacional, pois para cada imagem modelo diferente é necessário percorrer novamente a imagem original.

Funções disponíveis no OpenCV

O OpenCV é dotado de uma função específica para *template matching* `cv.matchTemplate()`. Como se explicou anteriormente, esta função apenas faz deslizar a imagem do modelo sobre a imagem de entrada e compara a área da imagem sobreposta com a imagem modelo. O OpenCV permite ainda escolher diferentes métricas de comparação entre as duas imagens, dando, para cada uma delas, maior atenção a diferentes características. O resultado é uma imagem em tons de cinza, onde cada pixel representa a semelhança que a vizinhança do pixel apresenta com o modelo. Dependendo da métrica usada, a região com mais parecenças pode variar. [ope \[e\]](#)

Uma vez que o modelo necessita de caber dentro da imagem original para que a correlação seja calculada, a imagem do resultado final não será do mesmo tamanho que o modelo original. Se a imagem de entrada for de tamanho ($W \times H$) e a imagem de modelo for de tamanho ($w \times h$), a imagem de saída terá um tamanho de ($W-w+1, H-h+1$).

Após obtida a imagem resultante, é necessário utilizar a função `cv.minMaxLoc()` para descobrir o valor máximo/mínimo, que identifica o ponto mais parecido com a imagem modelo. Descoberto o ponto extremo, fixa-se este como canto superior esquerdo do retângulo e toma-se (w, h) (dimensões da imagem modelo) como largura e altura do retângulo. Esse retângulo é a região do modelo, isto é, a área da imagem original que mais semelhança apresenta com a imagem modelo.

Na função do openCV `cv.matchTemplate()` existem seis métricas de comparação diferentes disponíveis, sendo estas exibidas de seguida com a respetiva fórmula de cálculo e o exemplo de uma imagem resultante para a métrica apresentada. [ope \[e\]](#)

- **CV_TM_SQDIFF**

$$\text{Fórmula: } R(x, y) = \sum_{x',y'} (T(x', y') - I(x + x', y + y'))^2$$

Resultado:



Figura 15: Fórmula de cálculo e Resultado do método CV_TM_SQDIFF ope [e]

- **CV_TM_SQDIFF_NORMED**

$$\text{Fórmula: } R(x, y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Resultado:



Figura 16: Fórmula de cálculo e Resultado do método CV_TM_SQDIFF_NORMED ope [e]

- **CV_TM_CCORR**

$$\text{Fórmula: } R(x, y) = \sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))$$

Resultado:



Figura 17: Fórmula de cálculo e Resultado do método CV_TM_CCORR ope [e]

- **CV_TM_CCORR_NORMED**

$$\text{Fórmula: } R(x, y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Resultado:



Figura 18: Fórmula de cálculo e Resultado do método CV_TM_CCORR_NORMED ope [e]

- **CV_TM_CCOEFF**

$$\text{Fórmula: } R(x, y) = \sum_{x',y'} (T'(x', y') \cdot I(x + x', y + y'))$$

where

$$T'(x', y') = T(x', y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'', y'')$$

$$I'(x + x', y + y') = I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x + x'', y + y'')$$

Resultado:



Figura 19: Fórmula de cálculo e Resultado do método CV_TM_CCOEFF ope [e]

- **CV_TM_CCOEFF_NORMED**

$$\text{Fórmula: } R(x, y) = \frac{\sum_{x',y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x',y'} T'(x', y')^2 \cdot \sum_{x',y'} I'(x + x', y + y')^2}}$$

Resultado:



Figura 20: Fórmula de cálculo e Resultado do método CV_TM_CCORFF_NORMED ope [e]

Multi-template matching

Aplicar *template matching* normal apenas resulta na deteção de um único objeto, isto é, a área (objeto) que apresenta maior semelhança com a imagem modelo.

Como se apresenta na figura 21, a imagem de entrada é uma imagem de uma carta 10 de copas (que possui 10 símbolos de copas), enquanto que a imagem modelo é um coração (símbolo de copas).

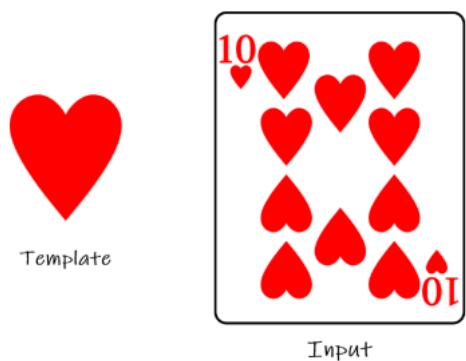


Figura 21: Valores de entrada para Multi-Template Matching [Kang and Atuk \[2020\]](#)

Aplicando o *template matching* normal, apenas será detetado um único coração na imagem original. Para serem reconhecidos todos os símbolos, é necessário filtrar a matriz resultante da função `cv2.matchTemplate()` e aplicar uma *non-maxima suppression*. Isto significa que, como queremos detectar múltiplos objetos, necessitamos de um valor de *threshold*, e todas as coordenadas da matriz resultante maiores que esse valor de *thershold* serão consideradas parecidas o suficiente da imagem modelo e o retângulo com vértice superior esquerdo nesse ponto será exibido. Isto significa que, para todas as regiões acima do valor de *threshold* considera-se que foi detectado um objeto. [Rosebrock \[2021b\]](#) Aplicando este método será possível detectar múltiplos objetos, tal como se mostra na imagem 22, onde mais símbolos de coração são detectados.

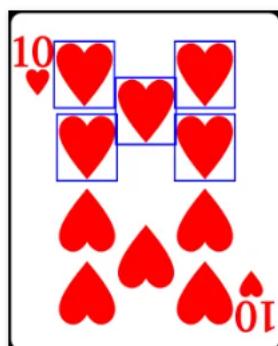


Figura 22: Detecção de múltiplos objetos em simultâneo (*multi-template Matching*) [Kang and Atuk \[2020\]](#)

Contudo, nessa mesma imagem [22](#), é possível ainda deduzir um problema. Claramente, o *template matching* é invariável a rotações e escalas, pois, pela figura, é perceptível que os corações que estão invertidos não foram detetados, bem como os de menor tamanho. Este problema deve-se ao *template matching* comparar exatamente a forma destes com a da imagem modelo. Uma possível solução seria correr duas vezes o template matching, uma para o coração direito e outra para o coração invertido. Todavia, exigiria um maior poder computacional. Outros potenciais problemas que o *template matching* pode ter são relativos à oclusão, falta de iluminação, mudanças de fundos, entre outros. [Kang and Atuk \[2020\]](#)

2.4 Análise de movimento e rastreamento de objetos

O rastreamento de um objeto é o processo de localizar e descobrir a posição exata de um objeto enquanto este está em movimento, isto é, em frames consecutivos de um vídeo.

O primeiro passo no rastreamento de um objeto é detetá-lo, criando uma caixa delimitadora à volta do objeto e posteriormente catalogá-lo. De seguida, é atribuído um ID único a cada objeto encontrado. Para cada frame de um vídeo é necessário detetar novamente os objetos presentes na imagem e de seguida compará-los com as informações previamente guardadas para saber se se tratam de um objeto novo ou um já antes detetado.

2.4.1 Deteção de Objetos vs Rastreamento de Objetos

Para a análise de um vídeo em tempo real, normalmente, os algoritmos de rastreamento são mais rápidos que simplesmente a utilização de algoritmos de deteção. Isto deve-se ao facto de que, quando se realiza um rastreamento de um objeto, este já é conhecido nos frames anteriores. Desta forma a aparência já é conhecida, bem como a sua localização, direção e velocidade do movimento que apresenta. Esta informação facilita a procura de um objeto num novo frame de um vídeo, enquanto que um algoritmo de deteção começa do zero a cada novo frame.

Apesar dos algoritmos de rastreamento beneficiarem de informações extras, também podem perder o controlo de um objeto quando este se desloca rapidamente ou quando o objeto fica atrás de um obstáculo por longos períodos de tempo. Nestes casos, o algoritmo de rastreamento acumula sucessivamente mais erros e a caixa delimitadora do objeto afasta-se cada vez mais do mesmo. Nestes casos, é necessário que um algoritmo de deteção seja executado para corrigir os erros para reencontrar o objeto perdido. [Maiya \[2020\]](#)

2.4.2 Desafios no rastreamento de objetos

O rastreamento de objetos é um método bastante desejado e é esperado apresentar resultados precisos num curto espaço de tempo. Todavia, existem alguns casos em que o rastreamento de objetos encontra algumas entraves na resolução do seu trabalho. [Gollapudi \[2019\]](#)

- **Oclusão do Objeto:** Em muitos casos, o objeto a seguir desaparece por breves períodos de tempo, ou porque fica ofuscado por outro objeto ou simplesmente porque desaparece do alcance da câmara. Nestes casos é difícil detetar e atualizar o objeto nas imagens futuras;
- **Velocidade:** Quando o movimento de um objeto é demasiado rápido, a câmara pode não detetar o objeto com nitidez suficiente para que o algoritmo consiga detetar o objeto e perseguí-lo;
- **Forma:** Caso o objeto a ser detetado não possua uma forma rígida e constante, isto é, se apresentar mudanças na sua forma, haverá uma falha na deteção e consecutivamente no rastreamento do objeto;
- **Falsos Positivos:** quando há vários objetos semelhantes é difícil distinguir com elevada certeza os vários objetos relativamente ao seu ID. Ou seja, o algoritmo pode perder o objeto atual e começar a rastrear um novo objeto semelhante.

Estes desafios podem interromper repentinamente o programa ou dar uma previsão completamente errada da localização do objeto a perseguir.

2.4.3 Técnicas de deteção de objetos para o Rastreamento

O rastreamento de objetos usa técnicas de deteção de objetos que são aplicadas a frames consecutivos de um vídeo. Visto que um vídeo é constituído por um conjunto de frames consecutivos, as caixas delimitadoras de cada objeto necessitaram de ser ajustadas em cada um desses frames.

Uma informação a ter em conta, é que, para o rastreamento mostrar resultados, assume-se que o objeto a ser rastreado está presente em todos os frames do vídeo. Isto é essencial para que se possa comparar a posição do objeto nos vários frames consecutivos.

Diferenciação de Frames

Frames são imagens de um vídeo capturadas a intervalos regulares e cada frame é constituído por um conjunto de pixels. A mínima variação num píxel indica a variação num frame.

A diferenciação de frames é uma técnica usada para medir a diferença entre dois frames de um vídeo, tendo em conta a posição de um ou mais objetos. Para determinar o movimento de um objeto, primeiro é calculada a diferença entre pixels em imagens consecutivas.

Visto que nem todas as alterações nos pixels são devido à movimentação de objetos, é implementado um limite. Ou seja, para diferenciar ruído ou até mudanças na luminosidade é imposto um valor mínimo de mudança, ignorando todos os valores inferiores a esse.

A diferenciação de frames é um método que funciona relativamente bem, sobretudo com imagens de fundo estático. Apresenta valores de *accuracy* elevados e necessita de pouco poder computacional.

[Gollapudi \[2019\]](#)

Subtração de Fundo

A subtração de plano de fundo é uma técnica de pré-processamento importante. Tem como principal objetivo separar o plano de fundo do primeiro plano num vídeo. Uma vez que a maior parte dos objetos se encontram no primeiro plano, a localização destes torna-se muito mais rápida, principalmente em câmaras estacionárias.

Todavia, no mundo real, este método torna-se um pouco mais complexo, devido à iluminação e mudança de luz, podem surgir sombras e focos de luz. Podem ainda existir objetos que estejam incluídos no plano de fundo e sejam removidos, tornando a sua localização muito mais difícil e muitas vezes incorreta. O método de subtração de fundo é mediano relativamente a *accuracy* e tempo computacional.

[Gollapudi \[2019\]](#)

Optical Flow

Optical flow corresponde à movimentação de objetos entre imagens consecutivas, quer devido à movimentação da câmara ou do objeto na imagem em si. Este método cria um vetor para cada pixel que se movimenta de um frame para outro.

Para esta técnica funcionar assume-se que a intensidade dos pixels entre frames apenas se altera se houver movimento e que existe movimentação semelhante entre pixels adjacentes. Para usar *optical flow* é preciso poder computacional relativamente alto e os resultados obtidos apenas conseguem *accuracy* moderada. [Gollapudi \[2019\]](#)

2.4.4 Métodos de Rastreamento de Objetos

O rastreamento de objetos começou com alguns algoritmos clássicos de *machine learning*, como KNN (K-Nearest Neighbors) e SVM(Support Vector Machines). Estas abordagens são relativamente boas para a previsão de objetos mas necessitam muitas vezes de informações extraídas manualmente, por exemplo, é necessário muitas vezes as imagens estarem manualmente catalogadas para estes algoritmos serem mais eficientes.

Desta maneira, os algoritmos destinados ao rastreamento de objetos foram introduzidos com o objetivo de melhorar a *accuracy* e a eficiência dos algoritmos já existentes. Assim, estes algoritmos realizam já a extração das características da imagem que necessitam.

GOTURN

Generic Object Tracking Using Regression Networks ou GOTURN utiliza uma abordagem baseada em regressão para rastrear objetos. O modelo regride diretamente para localizar o objeto com apenas uma única passagem *feed-forward*¹ pela rede.

A rede recebe dois inputs, recebe o objeto encontrado no frame anterior e consecutivamente a região de procura no frame atual. Desta forma, a rede compara as duas imagens e encontra facilmente o objeto na imagem atual.

É um modelo que treina de forma offline, ou seja, apenas é utilizado depois de completar o treino. Este tipo de algoritmos apresentam melhores resultados do que os online, uma vez que podem ser treinados para lidar com rotações, mudanças do ponto de vista, mudanças de iluminação entre outros desafios complexos. Assim, consegue correlacionar o movimento e a aparência de objetos e podem ser usados para rastrear objetos para os quais não foram treinados.

ROLO - Recurrent YOLO

ROLO é uma combinação de redes neurais recorrentes e do método de deteção de objetos YOLO. ROLO combina dois tipos de redes neurais, uma é CNN, usada para extrair informações espaciais, e outra é uma rede LSTM (Long Short-Term Memory), que calcula a trajetória do objeto.

A cada iteração, as informações espaciais e as caixas delimitadoras dos objetos, extraídas pela CNN, são enviadas ao LSTM.

¹ as ligações entre os nodos da rede não formam um ciclo

DeepSORT

DeepSORT é uma junção do *Simple Online Real-time Tracker* (SORT) com *deep learning network*. Apesar de ser um algoritmo online, o que o torna lento e pouco adequado para devolver resultados em tempo real, é bastante eficaz contra oclusão de objetos.

O algoritmo SORT é composto por três fases principais. Primeiramente, na fase inicial é detetado o objeto. De seguida, é feita uma previsão da localização deste. Por último, é feita uma otimização de maneira a calcular a posição correta.

O DeepSORT agrupa SORT com técnicas de *deep learning*, permitindo estimar a localização de um objeto e rastreá-lo com maior precisão.

2.4.5 Rastreamento de Objetos no OpenCV

OpenCV oferece um conjunto de algoritmos pré-construídos desenvolvidos explicitamente para fins de rastreamento de objetos. De seguida apresentam-se alguns dos rastreadores disponíveis no OpenCV [Rosebrock \[2018a\]](#), [Deshpande et al. \[2020\]](#):

- **BOOSTING**: é baseado no algoritmo de *machine learning* AdaBoost e existe há mais de 10 anos. É treinado em tempo de execução aprendendo os exemplos positivos e negativos do objeto a ser rastreado. É lento e não apresenta resultados muito confiáveis, mesmo para alguns dados mais superficiais;
- **Multiple Instance Learning (MIL)**: semelhante ao BOOSTING, todavia, para além de utilizar a localização do objeto para o rastrear, também utiliza parte da vizinhança deste. Apresenta melhor precisão que o BOOSTING;
- **Kernelized Correlation Filters (KCF)**: baseia-se no conceito de que vários exemplos positivos têm grandes regiões sobrepostas;
- **Discriminative Correlation Filter with Channel and Spatial Reliability (DCF-CSR) ou Channel and Spatial Reliability Tracking (CSRT)**: usa um mapa de confiança espacial para auxiliar na localização de objetos. Proporciona alta precisão para fps baixos.
- **MedianFlow**: calcula os deslocamentos do objeto em tempo real e mede o erro e a diferença entre as duas posições. O principal objetivo é minimizar o erro e detetar falhas no rastreamento e para poder selecionar as trajetórias mais confiáveis.

- Tracking, Learning, and Detection (TLD) : este algoritmo segue o objeto frame a frame e localiza em todos os frames a posição do objeto, tendo por base o que aprendeu no frame anterior, corrigindo simultaneamente se necessário.
- Minimum Output Sum of Squared Error (MOOSE) : Suporta escalas, deformações não rígidas, mudanças de iluminação e oclusões e consegue ainda reencontrar o objeto assim que este reaparece em cena. Relativamente a desempenho, é um pouco pior que GOTURN.
- GOTURN : é baseado numa abordagem *deep learning*, visto que utiliza *convolutional neural networks*. Apresenta *accuracy* elevada e suporta deformações, mudanças de iluminação e de ponto de visto, contudo não lida bem com oclusões.

2.4.6 Comparação entre Algoritmos de Rastreamento de Objetos no OpenCV

Estudos como [Dardagan et al. \[2021\]](#) e [Janku et al. \[2016\]](#) avaliam os diferentes algoritmos de rastreamento de objetos disponíveis na biblioteca OpenCV. Estes estudos comparam os diferentes algoritmos tendo em conta o nível médio de exatidão e precisão, bem como o sucesso a analisar características mais concretas como iluminação, oclusão, deformação de objetos entre outros.

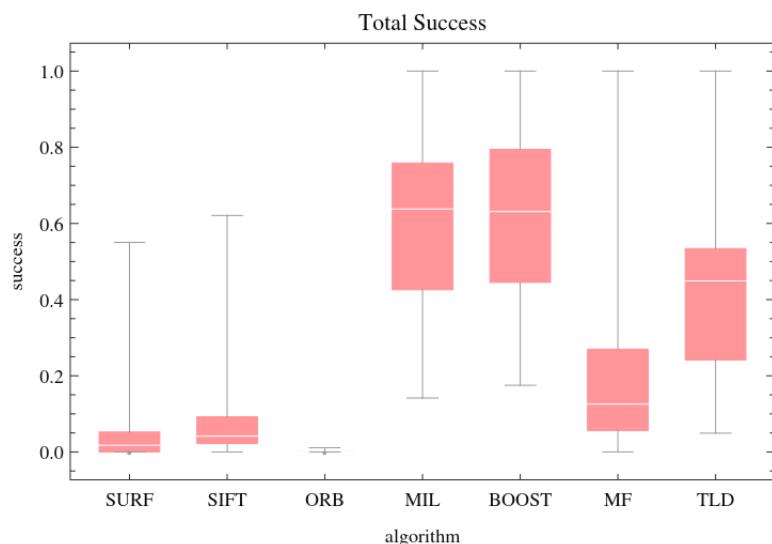


Figura 23: Taxa de sucesso dos algoritmos de rastreamento do OpenCV [Janku et al. \[2016\]](#)

Video/Tracker	Boosting	CSRT	KCF	Medianflow	MIL	MOSSE	TLD
MOT20-01	0.3955	0.4179	0.3852	0.3382	0.4079	0.3803	0.1771
MOT20-02	0.3564	0.4057	0.3193	0.2783	0.3406	0.3118	0.1888
MOT20-03	0.2888	0.4755	0.1667	0.0984	0.3843	0.2983	0.0614
MOT20-05	0.2995	0.3240	0.2480	0.2385	0.3186	0.1901	0.0895

Figura 24: Exatidão Média dos algoritmos de rastreamento do OpenCV [Dardagan et al. \[2021\]](#)

Esses estudos concluíram que MOSSE e Medianflow são os melhores algoritmos para o rastreamento de objetos em tempo real (gráfico da figura 23) e conseguem rastrear mais de 100 objetos simultaneamente. Relativamente à exatidão e precisão (tabela da figura 24), CSRT é o melhor algoritmo seguido pelo MIL e Boosting. KCF, Medianflow, MOOSE e TLD apresentam os piores resultados. Deduz-se que Boosting ou MIL são os algoritmos que balanceiam ambos os aspectos de precisão e rapidez.

Apesar de não ser realizado em ambiente OpenCV, [Held et al. \[2016\]](#) é um estudo que propõe um novo método para treino *offline* de redes neurais. Este algoritmo, intitulado GOTURN, foi o primeiro a conseguir atingir os 100 fps. Nos testes realizados que comparam este novo algoritmo com os já existentes (gráfico da figura 25) é possível concluir que este *tracker* supera, de forma geral, todos os algoritmos anteriores.

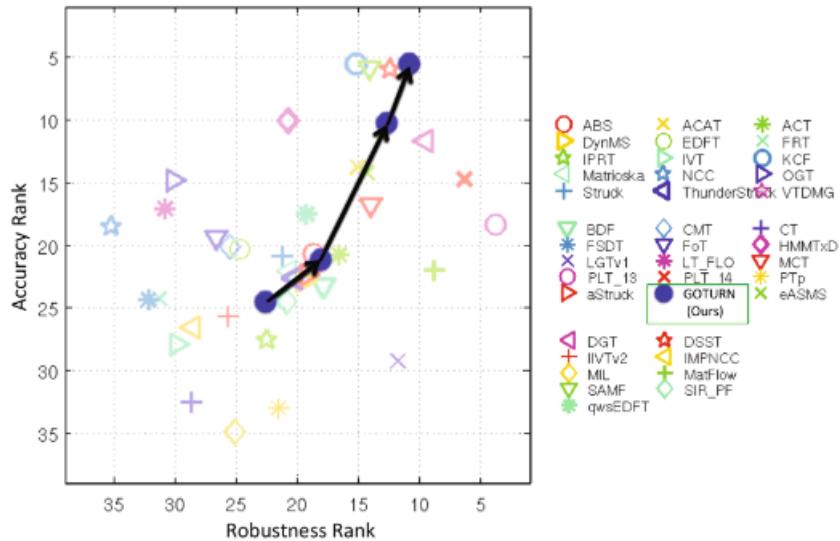


Figura 25: Comparação dos vários algoritmos de rastreamento relativamente a exatidão e robustez [Held et al. \[2016\]](#)

Deve-se ter em especial atenção o facto de que o OpenCV continua em progresso e, como tal, já existem versões mais recentes, com novos algoritmos, desde que estes estudos foram realizados. A versão mais recente de OpenCV, atualmente, é 4.6.0 lançada a 12 de Junho de 2022 [ope \[d\]](#). A partir

da versão 4.5.5 passou a existir o algoritmo GOTURN, que possui potencial para ser ainda melhor que os outros algoritmos já existentes e analisados até ao momento. Contudo ainda não foram realizados estudos sobre o mesmo no ambiente de OpenCV. Para além disso, OpenCV permite a utilização paralela do GPU com o CPU, possibilitando otimizar qualquer algoritmo.

2.5 Síntese

Nesta secção foram identificados os conceitos base e essenciais ao desenvolvimento do projeto. Foram abordadas as principais áreas de interesse como inteligência artificial, *machine learning* e *computer vision*. Foram ainda identificadas as principais dificuldades relativas à análise de imagens e vídeos, sendo necessário realizar uma análise cuidada para ultrapassar estes problemas.

Exploraram-se também as áreas de *object detection*, *template matching* e *object tracking*, analisando em concreto alguns dos seus algoritmos. Feita uma análise comparativa entre os diferentes algoritmos, foi possível retirar as vantagens e desvantagens de cada um, de forma a escolher o mais adequado a usar.

Para o projeto em mente, é necessária a utilização conjunta de um algoritmo de deteção e um de rastreamento de objetos. Como tal, começou-se pela escolha do algoritmo de deteção. Numa fase inicial, foi escolhido o algoritmo YOLO, pois entre os algoritmos de *object detection* é o que apresenta melhor performance. Após a sua implementação, explicada à frente na secção 3.2.1, reparou-se que este algoritmo, bem como todos os outros algoritmos de *object detection*, não se adaptam bem a ambientes de jogos, pois os *datasets* existentes são relativos a objetos do quotidiano. Mesmo que se arranjasse um *dataset* de objetos de jogos, este seria limitado, pois nunca se conseguiria treinar um elevado número de objetos de forma ao utilizador não ter limite de escolha. E, tendo em conta o número de jogos existentes no mercado e o número de objetos diferentes que cada jogo possui, nunca seria possível criar um *dataset* suficientemente abrangente que englobasse todos os jogos.

Desta forma, após alguma investigação, optou-se pela escolha do *template matching* para deteção de um objeto. Esta técnica permite detetar uma imagem, representante de qualquer objeto, numa outra imagem de maiores dimensões. Assim, o utilizador pode optar por detetar qualquer objeto, desde que possua uma imagem deste. A maior desvantagem é que se o objeto não tiver exatamente a mesma forma da imagem fornecida, então o objeto não será detetado.

Escolhidos a técnica e algoritmo de deteção de objeto, foi eleito o algoritmo GOTURN para rastrear um objeto. Esta seleção foi feita devido ao facto de ser um dos poucos algoritmo existentes no OpenCV

que treina de forma *offline* e ser um dos mais recentes. Por ser dos mais novos, acredita-se que seja um dos mais eficientes e o facto de treinar em modo *offline* é uma grande valia para a deteção em tempo real, fundamental para o projeto a desenvolver.

No diagrama da figura 26 é representado o fluxo de dados do projeto a realizar. Nesse diagrama é perceptível o uso dos algoritmos até agora apresentados. Primeiramente é recolhida e processada a informação do ecrã do utilizador. Nessa informação serão aplicados os algoritmos escolhidos, Template Matching e GOTURN para deteção e rastreamento do objeto, respetivamente. Por último será apresentada uma caixa delimitadora a identificar o objeto em tempo real.

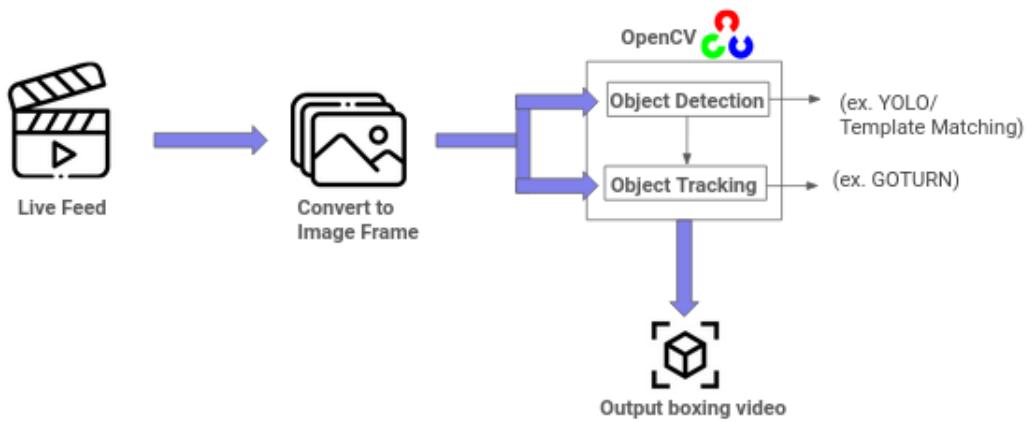


Figura 26: Fluxo de dados

Capítulo 3

Detect & Track

Neste capítulo será apresentado o trabalho desenvolvido ao longo da dissertação e serão descritas detalhadamente as diferentes fases da sua implementação.

Sucintamente, o trabalho consistiu no desenvolvimento de uma plataforma capaz de detetar e rastrear uma imagem de um objeto fornecida pelo utilizador. Esta plataforma foi desenvolvida tendo em mente que o objeto estaria presente num ambiente de um vídeo jogo. Como tal, o principal objetivo seria detetar este em tempo real sem a utilização de ferramentas computacionalmente pesadas que possam comprometer o jogo do utilizador.

Antes de mais, foi necessário realizar um levantamento dos requisitos necessários para o desenvolvimento do projeto e estabelecer, desde o início, as condições mínimas necessárias. Em segundo, foi necessário detetar o objeto no ecrã do jogador. Para isso, inicialmente, foi experimentado o algoritmo de *object detection* YOLO, e depois alterou-se para um algoritmo de *template matching*. Esta alteração deveu-se ao YOLO não conseguir detetar qualquer objeto que o utilizador quisesse.

Em terceiro lugar, para rastrear o objeto após a sua deteção, foi utilizado um algoritmo de *object tracking*, mais concretamente GOTURN. Por último, após estes passos estarem concluídos, é que se desenvolveu a plataforma com a qual o utilizador irá interagir, intitulada Detect&Track,

Seguidamente são explicados todos os passos e decisões tomadas para a realização da plataforma.

3.1 Levantamento de Requisitos

Nesta secção pretende-se recolher todos os requisitos à qual a aplicação desenvolvida deve atender. Desta maneira, os requisitos estabelecidos pretendem definir a estrutura e o comportamento do software criado. Desta forma, primeiramente são apresentados os requisitos funcionais, e de seguida são apresentados os requisitos não funcionais.

Os requisitos funcionais representam as características e funcionalidades que a aplicação deve pos-

uir para permitir que os utilizadores cumpram as tarefas definidas inicialmente, sendo assim importante ter a certeza que estes ficam clarificados. Em geral, requisitos funcionais descrevem o comportamento do sistema em certas circunstâncias ou condições. Desta forma, foram definidos os seguintes requisitos funcionais:

- o utilizador deverá ser capaz de escolher um janela para o programa detetar;
- o utilizador deverá ser capaz de fazer upload de uma imagem do objeto que pretende que seja detetado;
- o utilizador deverá ser capaz de escolher parâmetros, como *threshold*, para ajustar a precisão de deteção do programa;
- o sistema deverá apresentar explicações breves de todas as decisões que o utilizador possa tomar;
- o utilizador deverá conseguir começar a correr o programa com sucesso.
- o sistema deverá detetar e rastrear o objeto escolhido pelo utilizador, na janela escolhida, com sucesso.

Os requisitos não funcionais são relacionados com o uso da aplicação em termos de desempenho, usabilidade, confiabilidade, disponibilidade, segurança e tecnologias envolvidas. Muitas vezes são considerados como restrições aos requisitos funcionais, visto que determinam todas as necessidades que um requisito funcional não pode cumprir.

- a versão disponibilizada da plataforma para download tem de estar operacional;
- o produto deve zelar pelo aspeto simplista;
- o produto deverá ser desenvolvido em tons modernos;
- o produto deve ter um tema académico que remeta para e-sports;
- o produto deve apresentar eficiência perante a deteção de um objeto num jogo, sem que o tempo de resposta aumente exponencialmente;
- o produto deve operar sempre em modo *offline*, de modo a que não sobrecarregue o computador do utilizador, estando os modelos necessários já previamente treinados;
- o produto deve rejeitar a introdução de dados incorretos;

- o produto deve estar totalmente em inglês, de forma a que qualquer utilizador independentemente da sua nacionalidade e língua materna consiga entender e utilizar a plataforma com facilidade e sucesso.

3.2 Detetar objetos (Object Detection)

O primeiro passo no desenvolvimento deste projeto foi a deteção de um objeto, sendo esta a base de todo o projeto a desenvolver. Caso o programa não seja capaz de detetar corretamente o objeto, então será impossível realizar o seu rastreamento. Para além disso, é fundamental que o programa confirme, de vez em quando, se o objeto detetado continua a ser o desejado ou se, por algum motivo, o programa cometeu algum erro e detetou outro objeto por engano. Desta forma, é essencial que o *object detection* corra o mais rápido e corretamente possível.

3.2.1 YOLO

O algoritmo elegido para o *object detection* foi o YOLO, já analisado na secção anterior 2.3.3. Este algoritmo, à semelhança do SSD, é um detetor de estágio único que, apesar de normalmente serem menos precisos, são significativamente mais rápidos, fator essencial para o programa a desenvolver. Para além disso, tal como analisado na secção 2.3.3, comparativamente aos outros *object detection* analisados, YOLO é o que apresenta melhor performance e maiores valores de FPS.

De seguida é explica detalhadamente todo o processo de implementação do *object detection* YOLO no OpenCV Rosebrock [2018b].

Antes de mais, para a utilização de algoritmos de *objet detection*, é necessário utilizar *datasets* previamente treinados. Assim, foi utilizado o *COCO dataset*, treinado pela equipa DarKnet ([Redmon](#), [git](#)). Este dataset possui 80 classes de objetos já treinadas, sendo algumas destas classes pessoas, bicicletas, carros e camiões, aviões, animais como cães, gatos, pássaros, cavalos, entre outros, sinais de stop e até objetos de cozinha como copos, garfos, facas, colheres, entre muitos outros objetos.

Assim sendo, iniciou-se a utilização de YOLO primeiro numa só imagem. Para organização e maior compreensão do projeto, dividiu-se este em duas pastas:

- *yolo-coco*: onde se encontram os ficheiros do modelo treinado com o *dataset COCO*.
- *images*: esta pasta contém as imagens utilizadas para treinar e testar o algoritmo implementado.

Para além destas pastas, existe um *script* de Python com o código necessário para a utilização do

algoritmo YOLO. O programa pode receber quatro parâmetros iniciais, sendo estes:

- `---image` (obrigatório): o caminho para a imagem na qual se quer detetar objetos;
- `---yolo` (obrigatório): o caminho base para a pasta YOLO onde se encontram os ficheiros do modelo treinado;
- `---confidence` (opcional): a probabilidade mínima para filtrar deteções fracas (apresenta como valor *default* 50%);
- `---threshold` (opcional): o limite de supressão não máximo para eliminar caixas delimitadoras sobrepostas presente como valor *default* 0.3).

Com os argumentos fornecidos, são obtidas as configurações e pesos do modelo treinado, bem como todas as informações da imagem (como as dimensões e os dados correspondentes à imagem em binário).

Após extraída toda a informação necessária dos argumentos fornecidos, são inicializadas três listas que irão conter a informação final a ser exibida. As listas criadas são:

- `boxes`: que irá conter as caixas delimitadoras (*bounding boxes*) à volta do objeto;
- `confidences`: que irá conter os valores de confiança que o YOLO atribui a cada um dos objetos detetado. Valores de confiança inferiores ao valor passado como argumento inicialmente serão ignorados e o objeto não será considerado;
- `classIDs`: o *id* da classe do objeto detetado.

Assim, para cada objeto detetado serão extraídas as informações supracitadas, e, caso este apresente uma confiança superior à passada inicialmente como argumento, então as informações serão adicionadas às respetivas listas.

Uma vez que o YOLO não apresenta supressão não máxima (*non-maximum suppression*), é necessário aplicá-lo explicitamente com a intenção de remover todas as caixas delimitadoras sobrepostas e apenas deixar aquelas que apresentam maior confiança.

Por último, é necessário desenhar, na imagem fornecida, as caixas delimitadoras dos objetos encontrados e escrever as respetivas classes e confiança para cada um dos objetos.

De seguida são exibidos os resultados obtidos das quatro imagens testadas, apresentando o input (imagem fornecida ao algoritmo) e o output respetivo (imagem devolvida pelo programa), destacando os objetos destacados com a respetiva certeza:



(a) input Thomas [2020]

(b) output

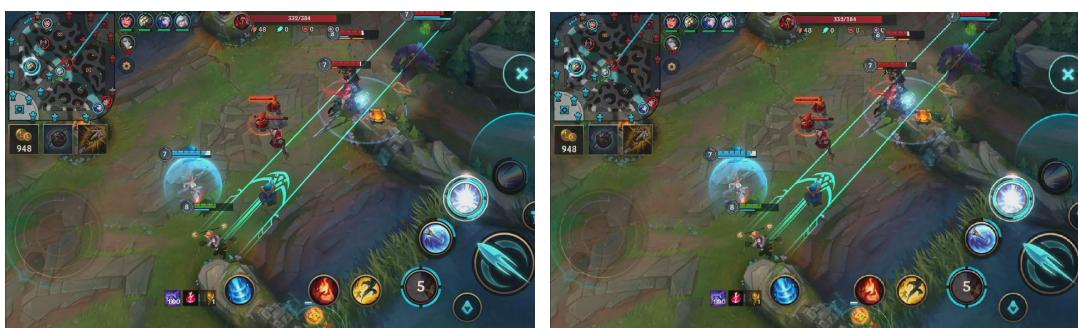
Figura 27: Imagens de pessoas reais a passear na rua



(a) input

(b) output

Figura 28: Imagens do jogo Brawlhalla



(a) input Neto [2019]

(b) output

Figura 29: Imagens do jogo de League of Legends



(a) input *Rahman [2021]*

(b) output

Figura 30: Imagens do jogo CS-GO

A partir dos testes realizados ao algoritmo YOLO, como se pode comprovar nos resultados exibidos, rapidamente se percebe que os objetos que o *dataset* possui são mais associados a objetos do quotidiano e não tanto a objetos com temática de jogos, isto é, que aparecem frequentemente ou só aparecem no mundo dos jogos. Tanto no jogo Brawlhalla (imagem 28) como no CS-GO (imagem 30), o YOLO não detetou nenhum objeto, e no jogo de LoL (imagem 29) só detetou dois dos três jogadores como pessoas, não detetando mais nenhum objeto. Na primeira imagem 27, mais associado ao mundo real (com pessoas a passear na rua) o YOLO conseguiu corretamente detetar todas as pessoas e carros, bem como o cão que aparece na imagem.

Desta forma, para se poder utilizar o algoritmo YOLO seria necessário criar um *dataset* próprio com múltiplos objetos de vários jogos, para que o utilizador pudesse escolher um objeto que estivesse presente no *dataset*. Seria necessário juntar várias imagens diferentes de cada objeto que se gostasse de adicionar ao *dataset*. Porventura, esta tarefa revelou-se bastante custosa e demorada e, para além disso, o utilizador teria sempre um leque limitado dos objetos possíveis a detetar. Como tal, optou-se por procurar alternativas a este problema, optando-se utilizar *template matching* em vez do algoritmos de *object detection* YOLO.

3.2.2 Template Matching

Após explorado o algoritmo YOLO, chegou-se à conclusão que este não era o pretendido para o projeto a desenvolver. Deste modo, procurou-se alternativas, sendo a escolhida o *Template Matching*.

Template matching, já explicado anteriormente na secção 2.3.4, permite detetar uma imagem modelo numa imagem original ou até mesmo num vídeo (sequência de imagens). Assim, para detetar um determinado objeto apenas é necessário uma imagem modelo do objeto pretendido, não sendo necessário treinar previamente nenhum algoritmo. Isto apresenta uma grande vantagem, pois para treinar um

algoritmo é necessário um diverso leque de imagens de cada objeto que se pretende treinar e obviamente nunca seria possível treinar todos os objetos existentes.

Desta forma, optou-se por implementar este método em vez dos algoritmos de *object detection*, uma vez que o utilizador apenas necessita de uma imagem do objeto que pretende que a plataforma detete e não está limitado ao número restrito de objetos que o algoritmo estaria treinado a reconhecer. Todavia, comparativamente aos outros algoritmos de *object detection*, o *template matching* possui uma grande desvantagem. Caso o objeto que o utilizador queira que seja detetado possua diferentes formas, o *template matching* apenas será capaz de reconhecer uma destas formas, isto é, a forma representada na imagem modelo. Por exemplo, caso o objeto gire, o *template matching* não será capaz de detetar o objeto na forma invertida, apenas na forma que é apresentada na imagem modelo.

Este problema resolver-se-ia facilmente correndo o *template matching* com mais do que uma imagem modelo, isto é, uma para cada forma diferente do objeto. Contudo, esta opção não foi posta em prática, por uma questão de performance, no entanto seria algo a considerar como trabalho futuro.

Para implementar o *template matching*, utilizou-se a biblioteca OpenCV, que possui centenas de funções para processamento e compreensão de imagens.

A função do OpenCV mais relevante para o projeto a desenvolver é a `matchTemplate()`. Esta função recebe como argumentos:

- uma imagem modelo, que representa o objeto a detetar;
- uma imagem global onde se irá procurar a imagem do objeto;
- um método de comparação (métrica) entre as duas imagens.

O resultado final é uma imagem a preto e branco da imagem global fornecida. A imagem resultante dependerá da métrica fornecida, onde os pontos a branco representam os pontos com maior semelhança à imagem modelo.

Assim, o primeiro passo a seguir é processar as imagens fornecidas, utilizando o método `imread()` que recebe dois parâmetros de entrada. O primeiro é o caminho do ficheiro da imagem, o segundo é um sinalizador para pré-processar a imagem. Optou-se por deixar a imagem inalterada (`UNCHANGED`), visto que processar a imagem depende muito da imagem escolhida e do ambiente em que esta se encontra, podendo melhorar o reconhecimento da imagem alguns jogos e piorar para outros.

Tendo as imagens carregadas, é só necessário invocar a função `matchTemplate()`. Inicialmente foi testado com uma imagem original para além da imagem modelo, mas rapidamente se substitui esta imagem original por uma captura da janela do jogador (explicado à frente na sub-secção 3.2.3 o processo

para obter esta captura de ecrã). Relativamente ao método de comparação utilizado, foram analisados as métricas normalizadas existentes, já mencionadas anteriormente [2.3.4](#). Estes métodos podem ser divididos em dois grupos, os normalizados e os não normalizados. O objetivo da normalização é fornecer uma métrica que possa comparar modelos de tamanhos diferentes. Por exemplo, duas imagens modelos do mesmo objeto mas com dimensões diferentes deveriam detetar o mesmo objeto numa outra imagem. Desta forma, como não sabemos as dimensões, intensidade das cores ou saturação da imagem modelo que o utilizador pretende escolher, optou-se por escolher a versão normalizada do método. [Reynolds \[2019\]](#)

Para a mesma imagem (modelo e original), invocou-se a função `matchTemplate()` variando apenas a métrica. Desta forma, foi possível avaliar as diferentes métricas normalizadas existentes, obtendo-se os seguintes resultados:

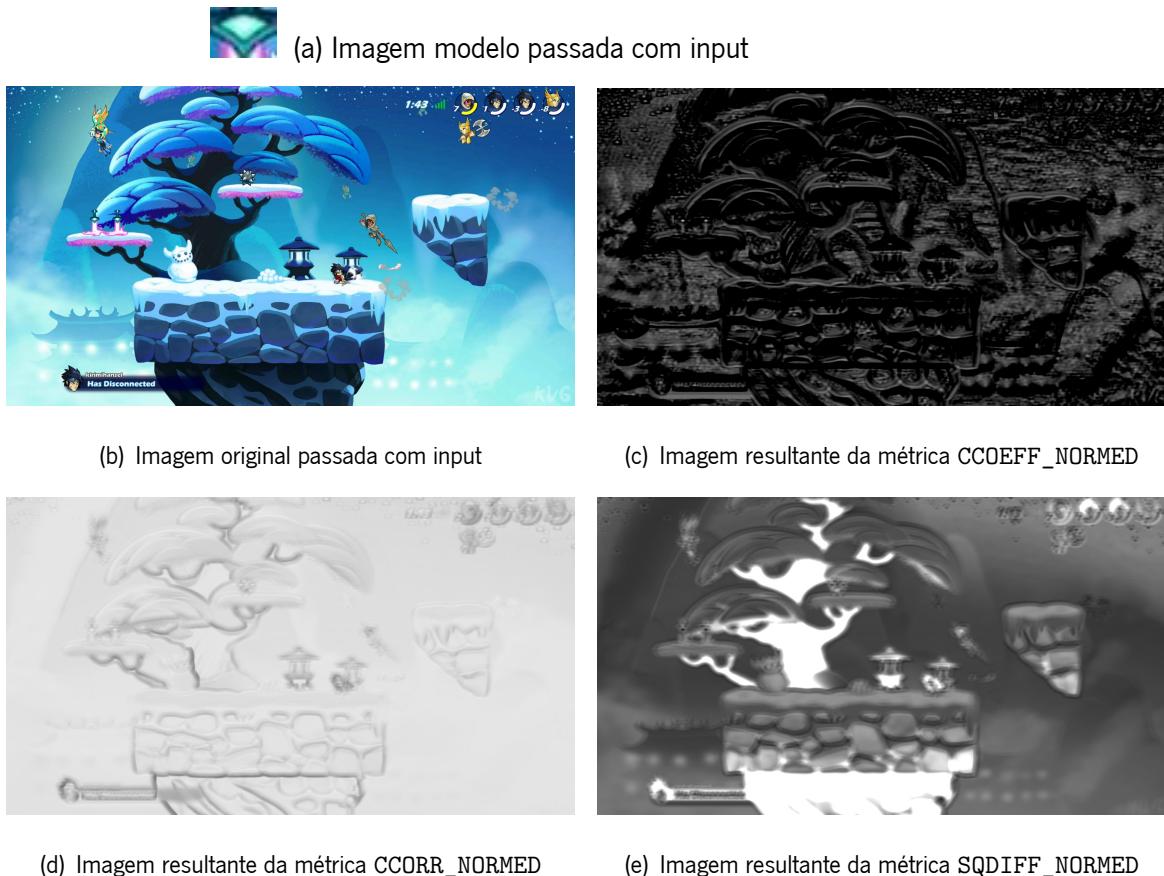


Figura 31: Imagens de input e output da função `matchTemplate()` com as diferentes métricas normalizadas possíveis

Para as métricas CCOEFF_NORMED ([31\(c\)](#)) e CCORR_NORMED ([31\(d\)](#)), o pixel mais claro (valor máximo) representa a melhor correspondência com a imagem modelo, sendo esse ponto obtido através da

função `minMaxLoc()` que devolve o mínimo e o máximo obtido, ou seja, o pior e o melhor valores e respetivas localizações. Para a métrica `SQDIFF_NORMED` (31(e)) é necessário realizar o inverso de todas as ações, por exemplo, a melhor correspondência seria o ponto mais escuro que é o valor mínimo obtido na mesma pela função `minMaxLoc()`.

A melhor posição obtida, em qualquer um dos casos, corresponde ao canto superior esquerdo da imagem modelo e, consecutivamente, da caixa delimitadora que rodeia o objeto, que tem as mesmas dimensões que a imagem modelo.

Uma possibilidade ponderada foi permitir o utilizador escolher a métrica que mais preferia, podendo ajustar consoante o ambiente do jogo escolhido. Todavia, à primeira vista, seria algo novo e completamente diferente do que o utilizador está habituado e, como tal, seria necessário dar muita informação ao utilizador para ele entender, na totalidade, o que teria de escolher. Por isso, optou-se por não se expor esta opção ao utilizador e utilizar sempre a mesma métrica, até porque os resultados entre as métricas não apresentam grandes variações.

Assim, para o programa a desenvolver, como métrica, foi escolhida a `CCOEFF_NORMED`, uma vez que segundo os testes realizados, esta é a métrica que apresenta melhores resultados para a maior parte dos casos. No entanto, `SQDIFF_NORMED` também poderia ser uma opções a considerar, visto que apresenta valores bastante precisos na maior parte dos casos com valores de confiança elevados, porém para imagens com algum ruído não é o mais indicado.

Escolhida a métrica ideal, já se consegue detetar a imagem modelo na imagem global fornecida. Porventura, se quisermos detetar mais que um objeto, em vez de simplesmente obtermos a melhor correspondência, é necessário obter todos os pontos acima de um determinado valor limite (*threshold*), passado inicialmente como parâmetro. Quanto menor o valor de *threshold*, mais objetos serão detetados. No entanto, se este valor for demasiado baixo, poderão ser detetados falsos positivos, isto é, pode ser detetado um objeto que não coincide com o pretendido.

Todavia, detetar mais que um objeto não é assim tão simples. O *template matching*, para um mesmo objeto, pode detetar e desenhar mais que um retângulo à volta de um objeto, parecendo que uns objetos tem um retângulo com espessura maior do que outros. Desta forma, para uniformizar todos os retângulos, o OpenCV disponibiliza de uma função de agrupar `groupRectangles()`, que recebe como parâmetros:

- lista de retângulos: contém as dimensões e as coordenadas do canto superior esquerdo de todos os retângulos
- groupThreshold: indica o número de retângulos que têm de estar sobrepostos para serem agrupados. Foi-lhe atribuído o valor de 1. Se este parâmetro tomar o valor de 0, então nenhum re-

tângulo seria agrupado e se tornar um valor superior a 1, então era necessário que mais retângulos estivessem sobrepostos para serem agrupados.

- valor indicativo da proximidade dos retângulos: indica a proximidade que os retângulos têm de ter para serem agrupados. Valores mais baixos requerem que retângulos necessitem de estar mais próximos para serem agrupados, enquanto valores elevados agrupa retângulos mais distantes. Foi-lhe atribuído o valor de 0.5.

Finalmente agrupados os retângulos e obtidos os valores e posições dos pixels que representam os objetos encontrados, é necessário desenhar um retângulo à volta do(s) objeto(s) e mostrar ao utilizador todos os objetos detetados na imagem original. O resultado esperado será do género da figura 32, que recebeu como input as imagens apresentadas em 31(a) e 31(b). Nesta imagem é possível visualizar que o algoritmo detetou com sucesso os dois "spawn weapons" existentes.



Figura 32: Deteção de múltiplos objetos através do *Template Matching*

3.2.3 Capturar Janela

Um dos principais objetivos é a detetar as imagens do ecrã do jogador em tempo real. Dito isto, é fundamental que o programa seja capaz de tirar *screenshots* constates ao ecrã do utilizador, para, posteriormente, conseguir detetar e rastrear o objeto no jogo. Assim, inicialmente, optou-se por utilizar a biblioteca de *python* pyautogui.

Pyautogui é uma biblioteca *cross-platform* que permite automatizar interações com interfaces gráficas através de código. Também permite controlar o rato e o teclado de um computador. Fornece algumas funcionalidades tais como:

- mover o rato e clicar em janelas de outras aplicações
- executar cliques de teclas noutras aplicações que estejam abertas
- localizar, mover, redimensionar, maximizar e fechar uma janela

Esta biblioteca foi desenhada para ser simples de utilizar em qualquer máquina independentemente do seu sistema operativo. Desta forma, não necessita de drivers específicos para Windows, porém não consegue atingir performances elevadas uma vez que não apresenta software específico, ou seja, como funciona em vários sistemas operativos com métodos de funcionamento diferente, não apresenta tanta performance.

Isto causa uma penalização de desempenho bastante elevada que impede que o resto das operações necessárias à plataforma desempenhem o seu trabalho corretamente.

Como tal, e como a plataforma é para ser utilizada somente em Windows, uma vez que a maior parte dos jogos também são quase só para este sistema operativo, tomou-se a decisão de mudar a biblioteca para uma específica a Windows. A biblioteca intitula-se Win32GUI e esta fornece uma interface direta em *python* da API nativa de janelas do Windows, a Win32. A Win32GUI em *python* permite realizar as mesmas operações que Pyautogui, mas, como é nativo, estas operações são executadas com maior performance.

Para além de detetar o ecrã todo do utilizador, com esta biblioteca, é possível ainda detetar apenas uma janela em específico, sendo necessário indicar o nome exato da janela desejada. Caso o nome da janela esteja incorreto ou a janela se encontre minimizada, o programa não conseguirá detetar corretamente a janela.

Assim, foi criada uma função que exibe ao utilizador todas as janelas que ele tem de momento abertas para que este apenas tenha de escolher a opção que deseja, tentado ao máximo evitar erros por parte do utilizador.

Após capturada a janela, esta é passado ao *template matching* para ser analisada e tentar encontrar os objetos modelo. Caso o encontre, fará o rastreamento deste com o algoritmo de *object tracking* GOTURN.

3.3 Rastrear objeto (Object Tracking)

3.3.1 GOTURN

Já detetado os objetos pretendidos passou-se à implementação de um algoritmo de rastreamento. No rastreamento, pretende-se encontrar um objeto no frame atual, desde que se tenha rastreado o objeto com sucesso em todos (ou quase todos) os frames anteriores. Uma alternativa possível ao algoritmo de rastreamento seria correr constantemente o algoritmo de deteção de objetos, ou seja, para cada um dos frames capturados seriam detetados novamente todos os objetos contidos na imagem. Contudo, os algoritmos de rastreamento tendem a ser mais eficientes, uma vez que tiram partido de informação contida nos frames anteriores, nomeadamente a localização, direção e velocidade do movimento que o objeto apresenta [Maiya \[2020\]](#).

Desta forma, e, até para corrigir algumas das dificuldades dos algoritmos de object tracking, optou-se por correr o algoritmo de deteção no início, sempre que o objeto deixa de ser detetado e de 10 em 10 iterações, como forma de validar se o objeto seguido continua a ser o correto.

Como já mencionado na secção [2.4.6](#) foi eleito o algoritmo GOTURN como algoritmo de rastreamento, principalmente por ser um dos mais recentes algoritmos com melhor performance e mais importante ainda, ser um algoritmo de treino offline. Significa que o algoritmo é previamente treinado e como tal, sempre que é utilizado não atualiza o modelo, tornando-o mais rápido e mais adequado para programas em tempo real.

O OpenCV 4 dispõe de uma API de rastreamento que contém implementações de muitos algoritmos de rastreamento de um único objeto, incluindo o GOTURN. A versão do OpenCV 3.2 já possuía algumas destas implementações, contudo na versão OpenCV 3.3 a API de rastreamento mudou.

De todos os algoritmos de rastreamento, GOTURN é o único baseado numa Rede Neuronal Convolucionial (CNN). Segundo [ope \[c\]](#) e [Held et al.](#), GOTURN é muito mais rápido devido ao treino offline, para além disso é bastante robusto a mudanças do ponto de vista (variações na câmara), mudanças de iluminação e deformações. Todavia, apresenta como principal ponto fraco as oclusões.

Conforme ilustrado na imagem seguinte, GOTURN é treinado usando milhares de pares de imagens cortadas, uma do momento exatamente anterior e outra do momento atual.

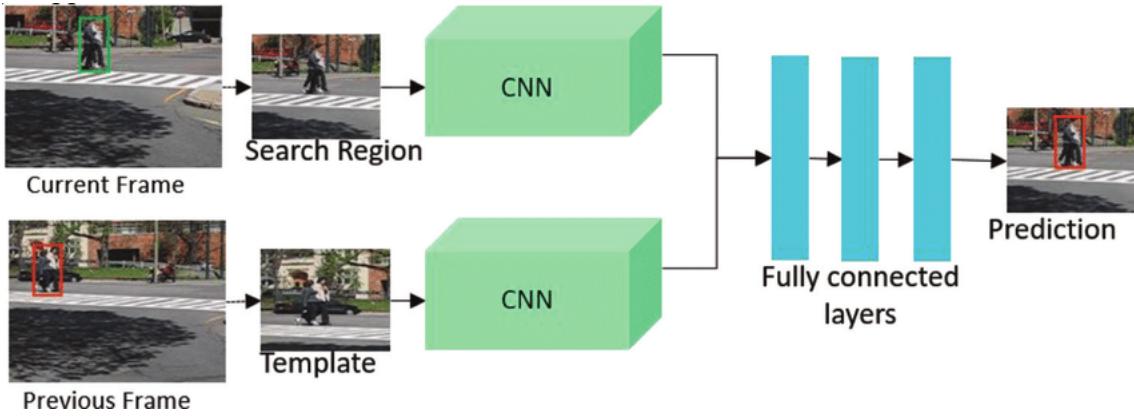


Figura 33: Arquitetura do rastreamento do algoritmo GOTURN

Como na imagem referente ao momento anterior é conhecida a localização do objeto, recorta-se esta de modo a que o objeto fique centrado, passando a ter duas vezes o tamanho da caixa delimitadora à volta do objeto.

Conhecida a localização do objeto no momento anterior, recorta-se também a imagem referente ao momento atual, de igual forma, de modo a ambos os frames possuírem a mesma dimensão. Contudo, caso o objeto se tenha movido, este não estará centrado na imagem recortada. Assim, será necessário prever a sua localização e confirmar a nova posição deste.

Sendo um rastreador baseado em CNN, o GOTURN necessita de um modelo Caffe (Convolutional Architecture for Fast Feature Embedding) para realizar o rastreamento.

Caffe é uma framework de deep learning que permite criar modelos de classificação e segmentação de imagens. Os ficheiros caffemodel são modelos em binário e, como tal, não podem ser abertos, examinados e editados manualmente. Estes modelos são criados e guardados através de ficheiros prototxt em texto simples. Deste modo, os modelos podem ser treinados e ajustados com a framework Caffe, e no final, o programa guarda o modelo treinado num ficheiro caffemodel.

Assim, para o projeto a desenvolver, foi utilizado um modelo previamente treinado. Para a sua utilização, tanto os modelos Caffe como o ficheiro prototxt devem estar presentes na pasta onde o algoritmo é utilizado.

Para fins de testes exemplificativos, e numa tentativa de perceber melhor o funcionamento do algoritmo GOTURN, começou-se por aplicar o algoritmo de rastreamento a um vídeo simples e concreto passado como argumento. Antes do vídeo começar, era exibido o primeiro frame e definia-se manualmente a caixa delimitadora. Obtidas as coordenadas desta, inicializou-se o tracker, que recebe como input o primeiro frame do vídeo bem como as coordenadas da caixa. Uma vez que o tracker é inicializado, é necessário atualizá-lo para todos os frames do vídeo. Desta forma, é criado um ciclo que itera entre

todos os frame, alimentando o tracker com cada novo frame. O tracker irá então prever as coordenadas da caixa delimitadora nesse frame que posteriormente será desenhada e exibida no ecrã.

Após se ter testado separadamente o object tracking, juntou-se com o object detection, havendo assim três grandes alterações. Primeiramente, em vez de o object tracker analisar um vídeo passa a capturar o ecrã, em tempo real, do utilizador. Assim, o ciclo, em vez de iterar ao longo do vídeo, captura o frame atual e passa-o ao tracker que deteta o objeto em tempo real.

A segunda alteração consistiu em, no primeiro frame, a caixa delimitadora já não ser fornecida pelo utilizador, mas sim pelo object detection que deteta inicialmente o objeto no frame atual e passa as coordenadas da caixa delimitadora ao object tracking.

Por último, já que o object detection deteta mais que um objeto em simultâneo, o tracker também necessita de rastrear todos os objetos encontrados. Desta forma, é necessário criar um tracker para cada objeto e atualizar todos eles a cada novo frame. É criada assim uma lista com todas as caixas delimitadoras previstas por cada um dos trackers, de forma a ser possível desenhá-las e exibi-las simultaneamente.

Para além destas modificações, foi adicionada uma condição como forma de verificação. Esta condição faz com que a cada dez iterações se valide, com o object detection, se o objeto rastreado pelo object tracking ainda é o certo.

Após estas mudanças, detetou-se que a performance obtida não era nem perto da desejada e como tal, implementaram-se threads numa tentativa de melhorar a performance e a rapidez do programa. Uma vez que já era criado um novo tracker para cada novo objeto detetado, optou-se por inicializar e correr cada tracker em threads distintas. Desta forma, todos os objetos detetados estão a ser processados e rastreados concorrentemente, em vez de sequencialmente. De forma análoga ao processo sem threads, é necessário atualizar os trackers com os novos frames obtidos e estes irão devolver as coordenadas da caixa delimitadora do objeto que estão a rastrear. Estas coordenadas serão adicionadas a uma lista para serem desenhadas e apresentadas ao utilizador.

De seguida é apresentado um fluxograma do programa onde é possível perceber com mais clareza a interação entre o template matching e o algoritmo de rastreamento (GOTURN). Percebe-se com mais clareza todo o processo explicado acima da criação e atualização dos trackers.

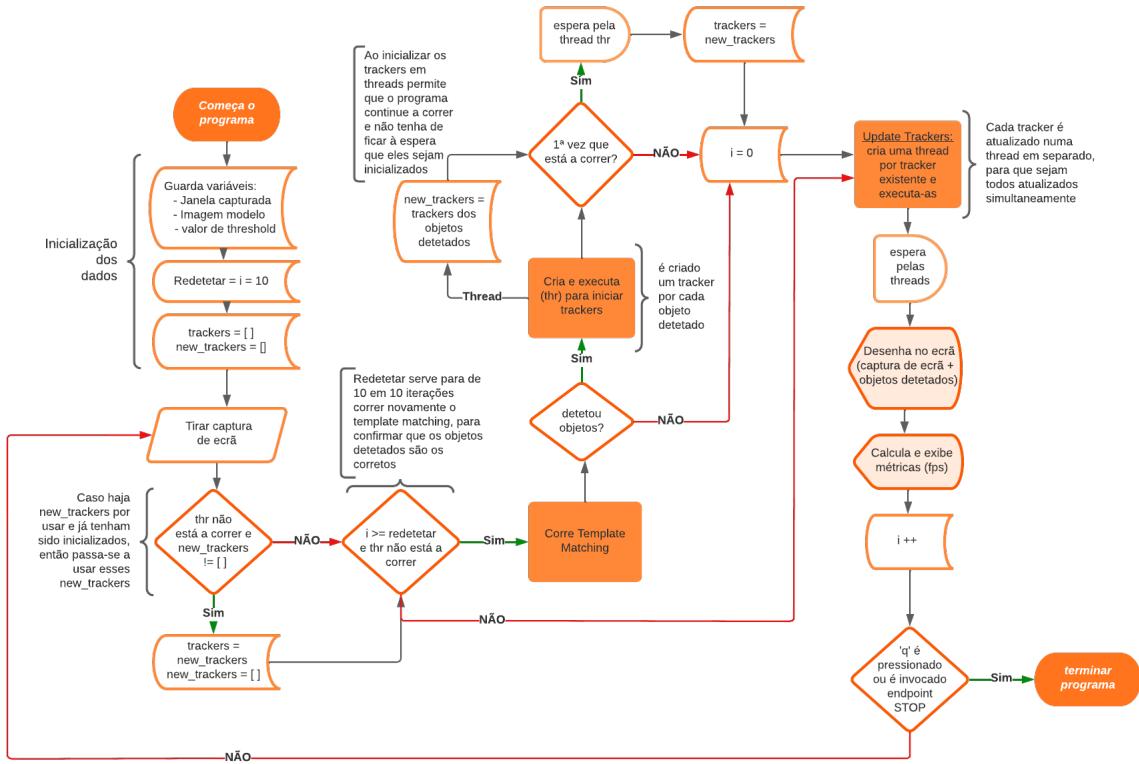


Figura 34: Fluxograma do programa

3.3.2 Problemas e limitações

Ao longo do desenvolvimento e implementação do algoritmo GOTURN, facilmente se percebe que este não é perfeito e que por vezes ocorrem algumas falhas na deteção e rastreamento de objetos.

Um dos principais problemas é que não existe informação de movimento. Desta forma, se o objeto se deslocar rapidamente e no frame atual não se encontrar perto da posição em que estava no frame anterior, então o GOTURN não irá detetar corretamente a nova posição do objeto, podendo, por engano, começar a detetar um outro objeto incorretamente que se encontre nas redondezas.

Num caso específico em que o utilizador opte por se detetar apenas uma parte específica de um objeto, o algoritmo pode acabar, ao fim de algum tempo, por detetar o objeto na sua totalidade em vez da parte específica pedida. Provavelmente isto acontece, devido ao facto de ele ser treinado a detetar o objeto como um todo e não por partes, conseguindo mais facilmente distinguir o objeto do plano de fundo e não pelas suas diferentes partes. Por este facto, caso as caixas delimitadoras fornecidas ao algoritmo apresentem demasiado ruído de fundo, a probabilidade desse ruído aumentar ao longo das iterações é elevado. Levando à acumulação sucessiva de erros e consequentemente uma deteção errada do objeto.

3.4 Plataforma e API

3.4.1 Protótipo

Com o objetivo de testar a usabilidade e a existência de potenciais problemas, começou-se por prototipar a plataforma a desenvolver. Assim, foi possível realizar um protótipo rápido e simples para testar o bom funcionamento da aplicação ainda antes de se começar a codificar.

Para tal utilizou-se a plataforma *Figma*, uma ferramenta de design para interfaces, que oferece a possibilidade de trabalho colaborativo num regime completamente online. Permite o uso simultâneo, num mesmo projeto, de diferentes membros sem necessitar de instalar nenhum programa ou aplicação. Funciona em múltiplas plataformas e não depende do sistema operativo. [dig \[2021\]](#)

Figma, lançado em 2016, veio facilitar o trabalhado dos web designers e revolucionar as ferramentas de design de UI existentes até então. Tem ganho cada vez mais popularidade no mercado e, como tal, tem havido um aumento na comunidade de desenvolvedores, que disponibilizaram, em 2019, uma biblioteca de *plugins*.

Apresenta inúmeras vantagens como uma curva de aprendizagem relativamente simples, manipulação vetorial flexível e colaboração em tempo real. Permite ainda a criação de comentários e anotações, que é extremamente útil quando se lida com clientes.

Todavia, apesar das grandes vantagens que possui, está longe de ser perfeita. Relativamente a outras plataformas existentes no mercado continua a ser mais nova, faltando bibliotecas de integração e ajuda da comunidade. Para além disso, não funciona em modo *offline*, sendo necessária uma conexão estável à Internet. Também não existe nenhuma versão mobile, que poderia ser extremamente útil para testar os protótipos nos dispositivos móveis. [Duò \[2022\]](#)

Para a plataforma a desenvolver considerou-se que apenas seria necessário possuir uma página com toda a informação, como é visível na imagem [35](#).

Optou-se por separar a página em duas secções distintas. A parte esquerda, apresenta uma breve explicação do trabalho desenvolvido, possuindo no final botões para informação relevante como o ficheiro final da dissertação desenvolvida e o *link do github* do projeto com acesso a todo o código desenvolvido. Por último, o logo mais à direita é o da empresa Anybrain, que auxiliou e encaminhou o desenvolvimento do projeto, e redireciona o utilizador para a página principal deles.

Na parte direita encontram-se as configurações que o utilizador pode alterar. À frente de cada opção existe um ícone de informação  que, caso se passe o rato por cima, exibe uma breve descrição da opção

bem como atenções a ter. Uma vez que o objetivo do projeto não era enfadar o utilizador com muitas opções, foram escolhidas as consideradas como principais. No futuro é sempre possível adicionar-se outras opções que se achem relevantes. Porventura, o objetivo é desenvolver uma plataforma simples e de fácil utilização, que não obrigue o utilizador a um processo maçador e demorado.

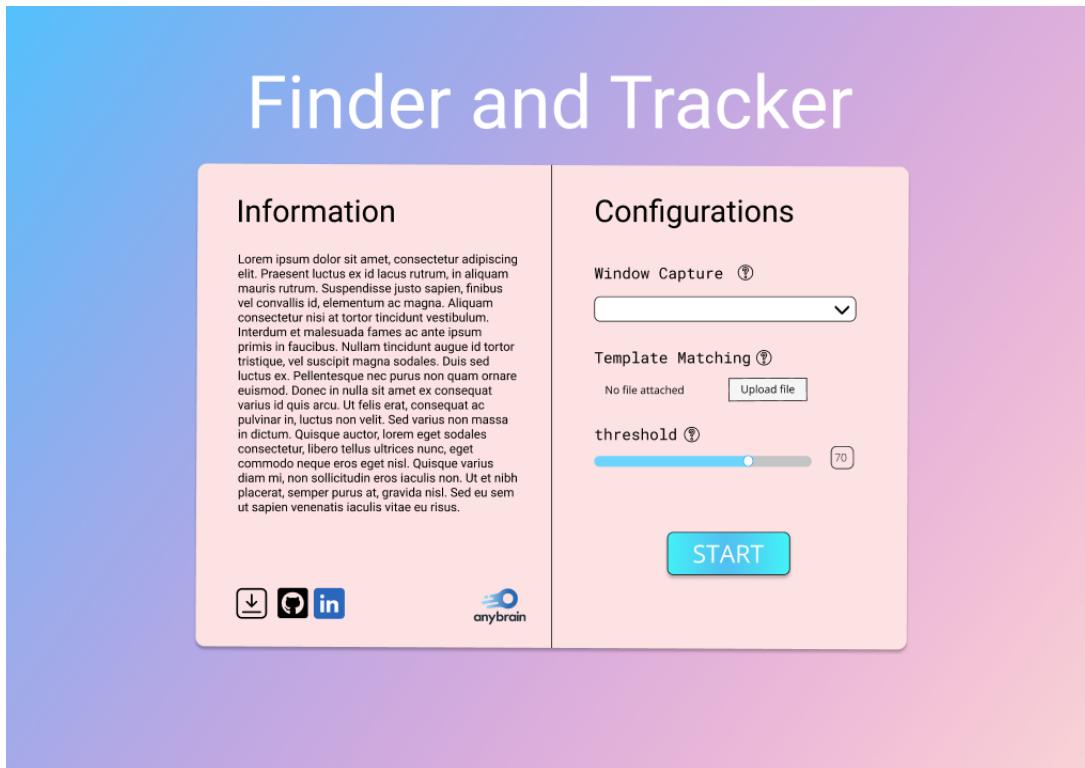


Figura 35: Protótipo da Plataforma

As opções disponíveis para o utilizador alterar são:

- **Window Capture** - é exibida uma lista com as janelas que o utilizador tem abertas no momento. o utilizador seleciona qual a que pretende que o programa capture informação;
- **Template Matching** - o utilizador faz upload de uma imagem de um objeto que pretende que o programa reconheça;
- **Threshold** - valor de threshold (limite mínimo para reconhecimento da imagem) que pode ser alterado num slide ou na caixa que exibe o valor definido atualmente (valor default 70).

Escolhida a configuração pretendida o utilizador apenas tem de clicar no botão 'START' para iniciar o processo de deteção e rastreamento do objeto inserido.

3.4.2 Tecnologias usadas

A escolha da arquitetura certa para o desenvolvimento de uma aplicação é o primeiro passo fundamental para todo o ciclo de vida do projeto. Permite alinhar as necessidades do produto ainda antes do começo do projeto, evitando problemas futuros relacionados com a correção ou atualização de casos. Estas decisões podem levar a uma economia tanto de tempo como de orçamento bastante significativas. Para a escolha da arquitetura teve-se em conta vários fatores como tamanho e complexidade do projeto, bem como a duração do mesmo, funcionalidade, velocidade e desempenho de um produto e flexibilidade e escalabilidade da *stack* tecnológica, entre outros.

Para a realização da aplicação, utilizou-se, como já referido anteriormente, *Python* e OpenCV para o desenvolvimento e aplicação dos algoritmos de *machine learning* e *computer vision*. Estas ferramentas constituem a camada lógica de negócios, camada encarregue de realizar o processamento por trás da aplicação. Esta camada comunica com a camada de apresentação (camada visível para o utilizador) através de uma API desenvolvida com o auxílio da *framework* Flask. Para a camada de apresentação foi utilizado Vue.js em conjunto com Electron. De seguida, na figura 36 é apresentada a *stack* tecnológica utilizada para a realização da aplicação.

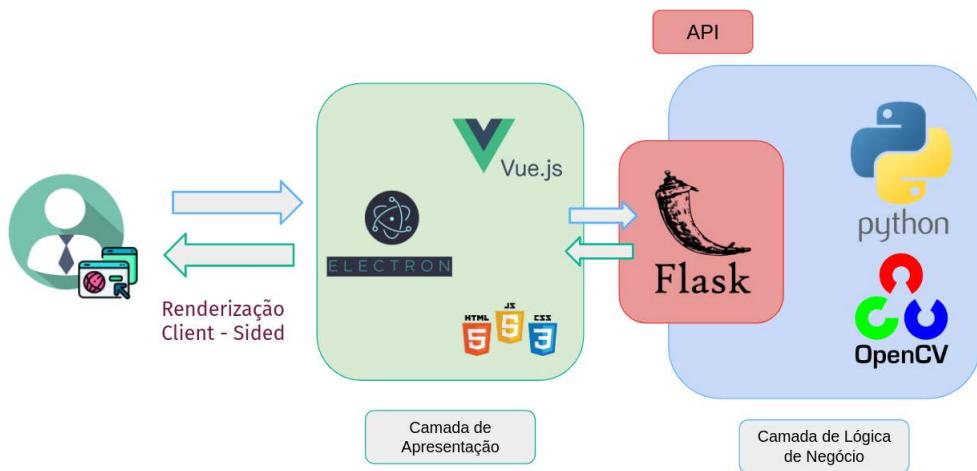


Figura 36: Stack tecnológica utilizada

Vue

A camada de apresentação é aquela com a qual o utilizador interage e serve de intermediário na troca de informações entre o utilizador e a aplicação. É nesta camada que os dados são tratados e convertidos para o formato desejado. Para o seu desenvolvimento optou-se pela utilização da *framework*

Vue.js. [Grankin \[2020\]](#) É uma *framework* que costuma a ser escolhida principalmente para a criação de projetos simples, perfeita para a construção de interfaces e aplicações de uma só página, como é o caso.

É uma ferramenta reativa relativamente recente, desenhada para melhorar a performance e fornecer tempos de downloads rápidos, conseguindo recarregar elementos individuais numa página evitando recarregar desnecessariamente a página inteira. É bastante flexível, trabalhando de igual forma para o desenvolvimento de aplicações ou de páginas web. É desenvolvida em JavaScript, uma das linguagens mais populares e de fácil aprendizagem no mundo do *web development*. É ainda uma aplicação facilmente adaptável que permite abranger rapidamente a novas necessidades que surjam ao longo do desenvolvimento do projeto.

Para além dos pontos supracitados possui ainda mais vantagens como ser leve (acelerando a instalação e carregamento de bibliotecas necessárias para a execução da aplicação), amigável ao utilizador e principalmente aos iniciantes (fácil acesso, utilização, aprendizagem e dispões de diversos *templates*), permite a reutilização de código, entre outras.

Todavia, apresenta como desvantagens limitações na utilização da comunidade (muitos dos elementos da *framework* encontram-se disponíveis em línguas que não o inglês, tornando difícil o uso desses elementos). Como é uma *framework* relativamente nova, não existem tantos *plugins* nem utilizadores experientes como comparando com outras *frameworks* tipo Angular e React. Para além disso, possui uma comunidade pequena e, como tal, não é viável a sua utilização para grandes projetos, uma vez que pode não se obter assistência imediata na resolução de problemas.

Electron

Ainda na camada de apresentação, foi utilizada a *framework* **Electron**, que permite a construção de programas de *desktop* tirando partido de tecnologias web como HTML, CSS e JS. Assim, construída a plataforma com o auxílio da ferramenta Vue, foi só utilizar Electron para converter a página criada numa plataforma aplicacional, de maneira a que seja gerado um executável de forma a que o utilizador apenas necessite de fazer o download deste para utilizar a plataforma. [Nalegave \[2018\]](#)

Uma das grandes vantagens de Electron, é que não é necessário aprender novas ferramentas e tecnologias para desenvolver uma aplicação. As aplicações criadas pelo Electron comportam-se de forma idêntica a aplicações Web e são capazes de correr em diferentes plataformas como Windows, macOS, Linux, Android e iOS, desde que a aplicação desenvolvida o permita.

Todavia, Electron corre através do *Chromium*, o que significa que, apesar de poder utilizar os seus benefícios (como ferramentas de *developer* e possuir acesso ao armazenamento), cada aplicação criada

possuiu a sua versão própria versão de *Chromium*, necessitando de instalar a aplicação em cima do *Chromium*, o que leva a que a aplicação seja demasiado pesada, desnecessariamente. [Erfan \[2021\]](#)

Flask

Para integrar a página desenvolvida com a camada lógica de negócios (*backend*), foi criada uma API com vários *endpoints* (pontos de acesso) que processam os pedidos feitos pela página e executam funções do *backend* (onde corre o software de processamento de imagem do OpenCV). Assim, a camada de interface faz chamadas aos *endpoints* através de pedidos HTTP, que enviam a resposta do pedido enviado. Os *endpoints*, por sua vez, comunicam com o *backend*, e correm os comandos necessários para a execução do programa. Os *endpoints* foram desenvolvidos com o auxílio da *microframework* Flask, que oferece todos os recursos básicos de uma aplicação web e trata de todos os pedidos HTTP recebidos.

A *framework*, por si só, tem recursos limitados, dado que apenas permite a criação de *endpoints* e é necessário, para a sua utilização, adicionar funcionalidades extras com tecnologias complementares. No presente caso, utilizaram-se outros *packages* de Python, como numpy e OpenCV para a execução do programa. [Semik and Stempniak \[2021\]](#)

Com Flask, é simples diversificar a estrutura do projeto de software com *microframeworks*, visto que permite separar o projeto em diferentes serviços e organizá-lo mais facilmente. A flexibilidade é o principal recurso desta estrutura de código *open-source*.

Flask é uma boa escolha quando existe uma multiplicidade de funcionalidades pequenas e dispersas, quando existe um número desconhecido de (micro)serviços altamente especializados e/ou se quer experimentar diferentes arquiteturas, bibliotecas e tecnologias mais recentes.

Para além disso, Flask também funciona com aplicações Web que são divididos em pequenos serviços que não exigem muito código para atingir os objetivos definidos. Um exemplo seriam as aplicações Web que se enquadram em sistemas maiores usados em DevOps, como monitoramento ou soluções de *logs* em grande escala.

3.4.3 Pedidos à API

API (Application Programming Interface) é um intermediário de software que permite que duas aplicações comuniquem entre si. Sempre que o utilizador realiza uma ação na plataforma é necessário comunicar com o servidor do *backend* e informar que ação foi executada, de forma a que este consiga processar a informação e executar a tarefa corretamente, enviando de volta a resposta ao pedido feito.

Para efetuar os pedidos à API utilizou-se, como já mencionado, a *framework* Flask. Esta permite configurar, para cada *endpoint* da API, os *headers* dos pacotes, o tipo de método HTTP (POST, GET, PUT, ...) e também receber e processar os dados enviados pela API.

De seguida listam-se os diferentes pedidos que estabelecem comunicação com a API utilizados pela camada de apresentação:

- **getWindows**: pedido GET que lista todas as janelas abertas, no momento, pelo utilizador;
- **run**: pedido POST para correr o programa de deteção e rastreamento;
- **stop**: pedido POST para parar de correr o programa.

São exibidos dois fluxogramas correspondentes aos últimos dois *endpoints*, referentes aos pedidos POST do programa. Assim é possível perceber os possíveis erros e diferentes caminhos que o programa pode percorrer.

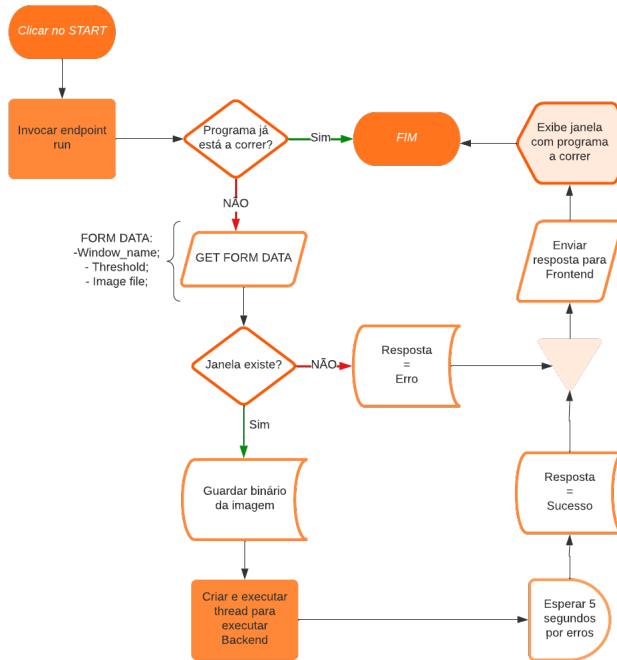


Figura 37: Fluxograma do *endpoint* RUN

Relativamente ao endpoint RUN [37](#), referente ao início do programa, o primeiro passo que este realiza é confirmar se o programa já está inicializado ou não. Como este demora algum tempo a abrir e não é realizado de forma instantânea, adicionou-se esta condição para impedir que o utilizador sobrecregasse o sistema com inundações do mesmo pedido repetidamente.

Caso ainda não esteja a correr, então irá obter a processar a informação fornecida pelo Form, isto é, os dados preenchidos na plataforma pelo utilizador (nome da janela, imagem modelo e valor de threshold). Caso a janela não exista, será devolvido um erro. Caso exista, irá guardar a imagem modelo do objeto e começará a executar o *backend* de forma a inicializar a deteção e rastreamento do objeto. Se ao fim de cinco segundos não for devolvido nenhum erro, assume-se que o programa foi inicializado e será enviada como resposta o valor de 'Sucesso'.

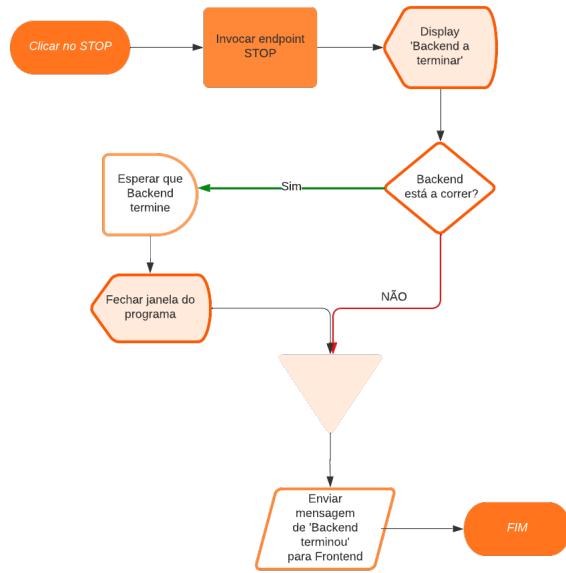


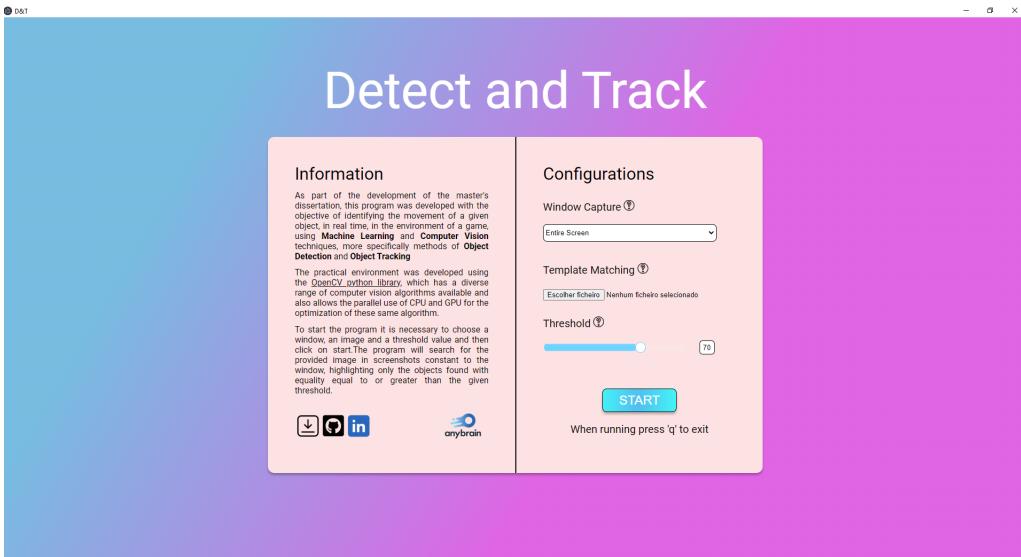
Figura 38: Fluxograma do endpoint STOP

O endpoint STOP [38](#) foi necessário criar para o utilizador conseguir parar a aplicação pela plataforma, sem usar atalhos ou ter de ir ao *backend* parar os processos ativos. Este endpoint apenas confirma se o *backend* está a correr. Em caso afirmativo fecha a janela do programa e termina todos processos ativos. Em caso negativo, não realiza nenhuma ação.

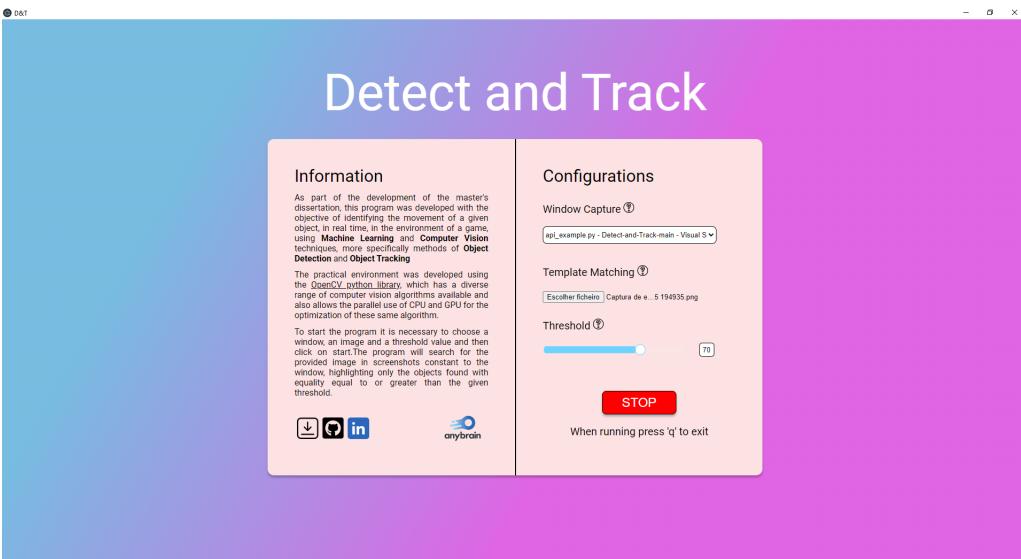
3.4.4 Interface Final

Visto que o protótipo criado é simples e só se trata de uma única página, a plataforma final é relativamente idêntica não sofrendo muitas mudanças. Com efeito, as principais alterações foram no texto apresentado, passando de um rascunho para a informação realmente pretendida. Para além disso, foi adicionada uma linha por baixo do botão de 'START' com informação relativa a como parar o programa.

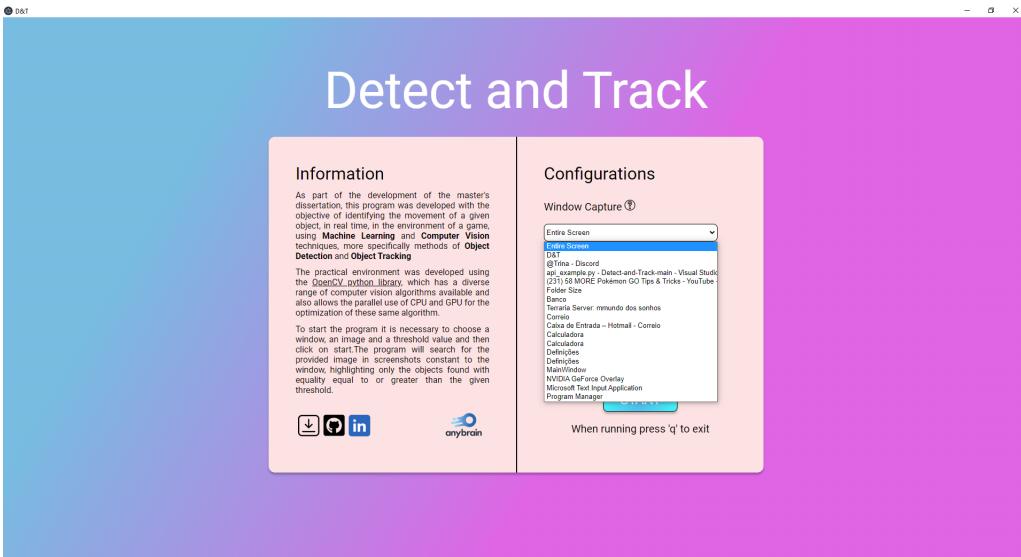
De seguida apresenta-se a interface final obtida.



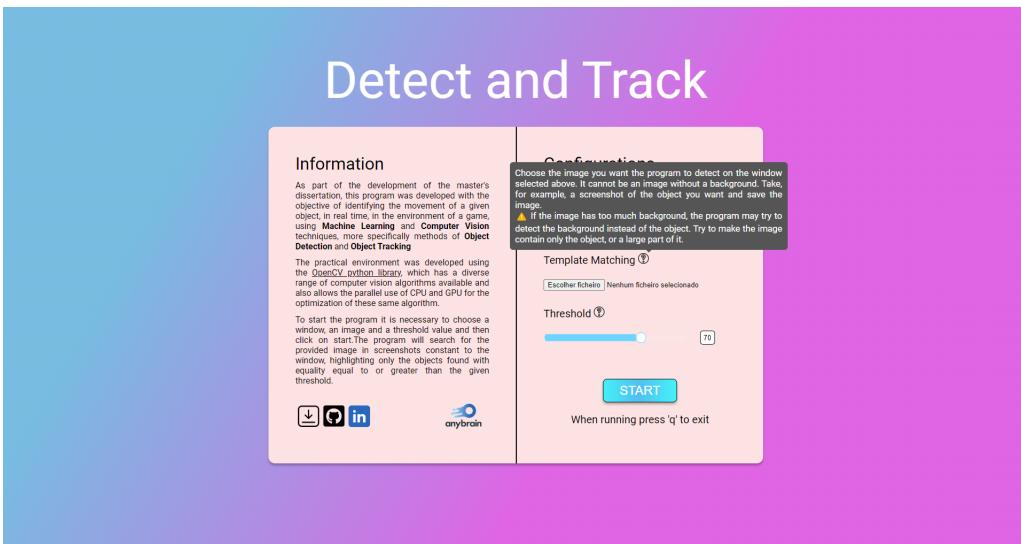
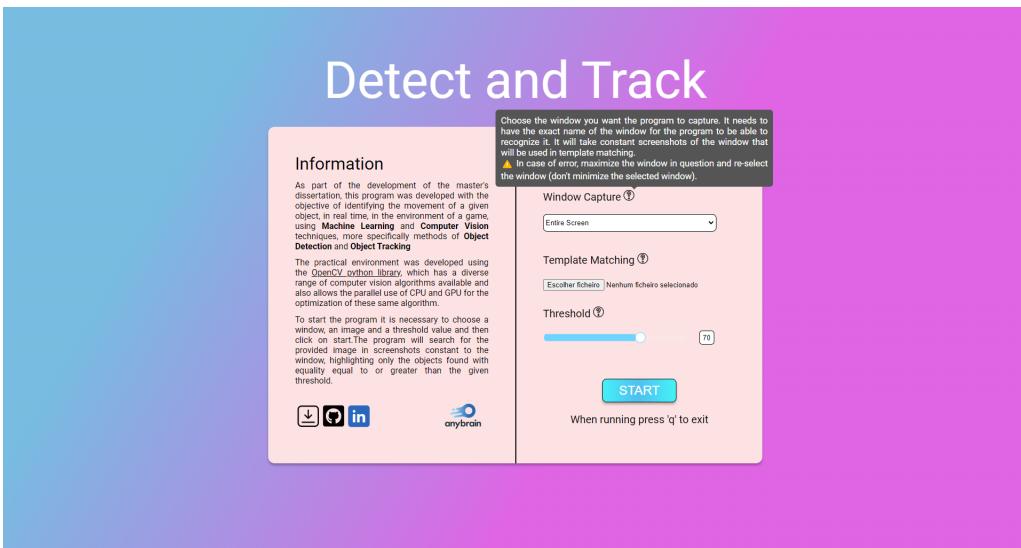
Assim que o utilizador começa a correr o programa, o botão de 'START' muda para botão de 'STOP', de forma a que o utilizador consiga facilmente perceber que o programa está a correr e parar quando quiser.



Como previsto no protótipo, a opção das configurações relativa a Window Capture exibe uma lista das janelas abertas no momento na máquina do utilizador.



Para cada opção existe uma breve descrição desta, bem como atenção a ter.



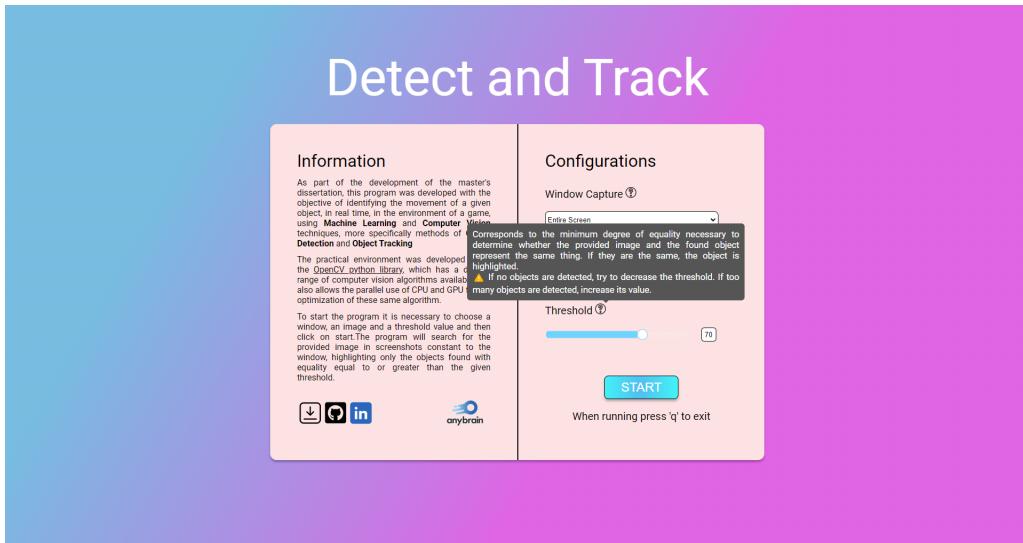


Figura 39: Interface final da plataforma desenvolvida

3.4.5 Análise de usabilidade

A qualidade de um sistema está diretamente relacionada com a forma como este é utilizado. Assim, para além do sistema responder de forma eficiente a todos os requisitos funcionais e não funcionais precisa ainda de ser usável. Desta forma é fundamental que um utilizador, especialmente um que não conheça a plataforma, consiga usar o sistema e atingir os seus objetivos com eficiência, eficácia e satisfação.

Assim, de maneira a analisar a usabilidade do sistema construído, teve-se em conta os princípios de usabilidade apresentados de seguida. Por último, de forma a avaliar a usabilidade do sistema, foram utilizadas as Heurísticas de *Nielsen*, dez heurísticas fundamentais para o desenvolvimento de uma interface gráfica.

Princípios de Usabilidade

Aprendizagem (Learnability)

Este princípio baseia-se na facilidade com que um novo utilizador consegue efetivamente começar a interagir com o sistema e atingir máximo desempenho, tendo em conta as seguintes propriedades:

- **Previsibilidade/Predictability** (determinar o efeito de uma ação no sistema): todos os botões existentes na plataforma têm o ícone mais comum associado à ação ou a sua descrição, por exemplo, o ícone do github reencaminha o utilizador para a página do *github* do projeto ou o botão 'START' começa a correr o programa.

- **Sintetizabilidade/Synthesizability** (avaliar o efeito de ações passadas no estado atual): assim que o programa inicia é possível observar que teve em conta as opção selecionadas previamente na plataforma.
- **Familiaridade/Familiarity** (maneira como conhecimento prévio (do mundo ou outros sistemas) se aplicam ao novo sistema): é normal, no final da página, haver uma zona de contacto através das diferentes redes sociais, para caso o utilizador queira contactar o desenvolvedor da plataforma.
- **Generalização/Generalizability** (estender o conhecimento relativo a uma interação específica a novas situações): pode se considerar que o preenchimento da plataforma é sempre o mesmo, não havendo alterações no seu preenchimento. Desta forma, assim que o utilizador adquire o conhecimento necessário para o preenchimento da plataforma pode generalizá-lo, uma vez que será sempre idêntico.
- **Consistência/Consistency** (parecença no comportamento de input/output em situações semelhantes): escolhendo sempre as mesmas opções nas configurações e exibindo o mesmo conteúdo, será de esperar que o programa se comporte da mesma forma e detete sempre os mesmos objetos, caso a situação seja idêntica.

Flexibilidade (Flexibility)

O princípio da flexibilidade diz respeito à multiplicidade de formas que o utilizador e o sistema trocam informação, tendo em mente as seguintes propriedades:

- **Iniciativa de Diálogo/Dialogue Initiative** (maximizar o controlo do utilizador): é o utilizador que possui o controlo total sobre a plataforma, sendo que o sistema apenas responde às suas ações.
- **Multithreading** (suportar mais que uma tarefa simultaneamente do mesmo utilizador): o utilizador consegue iniciar o programa, e, estando este a correr, consegue, na plataforma, aceder aos links lá disponíveis.
- **Migração de Tarefas/Task Migrability** (passar a responsabilidade do controlo de tarefas entre o utilizador e o sistema): após escolhida a configuração ideal e inicializado o programa passa a ser o sistema a possuir o controlo. Caso o utilizador pretenda alterar alguma configuração necessita de parar e reiniciar o programa.

- **Substituitividade/Substitutivity** (permitir substituição entre valores equivalentes): por exemplo, para a escolha do *threshold*, o utilizador pode utilizar o *slider* para indicar o valor pretendido ou escrever um número na caixa ao lado.
- **Personalização/Customizability** (permitir o utilizador modificar a interface): não aplicável, visto que a aplicação tem uma dimensão reduzida não é possível alterar a interface a gosto pessoal do utilizador.

Robustez (Robustness)

A robustez refere-se ao nível de suporte fornecido ao utilizador relativamente à avaliação do comportamento e ao sucesso alcançado de modo a atingir um objetivo. Foram tidas em conta as seguintes propriedades:

- **Observabilidade/Observability** (avaliar o estado interno do sistema): pelo botão 'START' e 'STOP' é possível avaliar o estado interno do sistema.
- **Recuperação/Recoverability** (habilidade de tomar medidas corretivas): inicializado o programa, não é possível alterar as configurações, sendo necessário reiniciar o programa com as novas configurações pretendidas. Todavia, até este ser começado é possível alterar as opções infinitas vezes.
- **Capacidade de Resposta/Responsiveness** (como os utilizadores percebem a taxa de comunicação do sistema): o sistema informa o utilizador caso haja algum erro.
- **Conformidade da tarefa/Task Conformance** (grau em que os serviços do sistema suportam as tarefas dos utilizadores): caso uma tarefa não seja fazível, então o sistema não irá permitir que esta seja realizada. Assim, todas as tarefas possíveis de realizar pelo utilizador são suportadas pelo sistema.

Heurísticas de Nielsen

1. Visibility of system status

A interface deve manter os utilizadores informados sobre o que se passa, através de *feedback* apropriado. Considera-se que a interface cumpre esta heurística uma vez que o utilizador é informado através do botão de 'START' E 'STOP' se o programa está ou não a correr. Para além disso, facilmente se observa se o programa está de facto a correr corretamente, uma vez que se abre uma janela nova com a informação captura da janela escolhida previamente nas configurações.

2. Match between system and the real world

Apresentar a plataforma com uma linguagem natural, lógica e familiar aos utilizadores, torna-se um requisito obrigatório, já que os utilizadores devem ser capazes de interpretar as informações que lhes são fornecidas. Esta correspondência entre o mundo real e o sistema está presente na aplicação, uma vez que utiliza conceitos básicos e internos à compreensão de utilizadores com um vocabulário menos extenso. Os vocabulários mais técnicos possuem uma pequena explicação para o utilizador comum poder entender o seu significado.

3. User control and freedom

Uma vez que os utilizadores executam frequentemente ações por engano, é necessário fornecer "saídas de emergência", claramente marcadas, de forma a que estes possam, tanto corrigir os seus erros, como ter a liberdade de navegar pela plataforma, tendo sempre controlo sobre decisões que executa.

Visto que a plataforma desenvolvida é de apenas uma página, não é necessário ter em conta a navegação entre páginas, que seria onde o utilizador teria mais dificuldade. Estando sempre na mesma página é apenas preciso ter em conta, que as ações que o utilizador faz podem ser desfeitas.

Desta forma, escolhendo uma opção, o utilizador pode sempre alterá-la até iniciar o programa. Assim que o inicia, pode pará-lo a qualquer momento e reiniciá-lo novamente com as novas configurações que pretender.

4. Consistency and standards

A interface deve seguir as convenções da própria plataforma e padrões utilizados na indústria. A plataforma respeita os princípios desta heurística uma vez que, por exemplo, as designações aplicadas aos elementos presentes na interface são mantidas constantes, isto é, designações diferentes não possuem significados iguais.

5. Error prevention

As melhores interfaces previnem o erro, impedindo ações destinadas a causarem conflitos no sistema de acontecerem. Neste sentido, o utilizador tem a opção de preencher um conjunto de opções para poder iniciar o programa. Estas opções para além de apresentam um valor default, para caso o utilizador não escolha nenhuma opção, apenas aceitam determinados valores para que o sistema não falhe.

6. Recognition rather than recall

Esta heurística define que o utilizador não deve ser obrigado a lembrar-se da informação de uma parte da interface para a outra, desta forma a aplicação deve ajudá-lo a lembrar-se exibindo toda a informação necessária.

Esta heurística acaba por não ser cumprida na sua totalidade uma vez que, quando o programa é iniciado não são exibidas as configurações escolhidas pelo utilizador, tendo de voltar à plataforma para ver as opções selecionadas. Todavia, considerou-se que esta era a melhor escolha, uma vez que se queria aproveitar o ecrã todo e não apresentar informação redundante.

7. Flexibility and efficiency of use

Para os utilizadores mais experientes, a interface deve fornecer atalhos e acesso direto aos itens utilizados mais frequentemente, garantindo uma maior flexibilidade e eficiência.

Devido à pequena dimensão da plataforma, não foi possível a realização de inúmeros atalhos, até porque não existem assim tantas ações a executar. Relativamente à escolha das opções e início do programa não existem atalhos disponíveis, uma vez que os recursos são de fácil acesso não existindo qualquer dificuldade. Quanto a parar o programa é possível para além do botão 'STOP', usar a tecla 'q' (de 'quit') para interromper o programa e voltar à página inicial da plataforma.

8. Aesthetic and minimalist design

Em termos da interface gráfica, esta deve possuir um aspeto minimalista, intuitivo e agradável à vista, com diálogos simples e sem conteúdo irrelevante.

Esta heurística foi tida em conta ao longo do desenvolvimento da plataforma, tentando-se ao máximo desenvolver uma plataforma simples, apenas com as opções essenciais e explicações de fácil compreensão.

9. Help users recognize and recover from errors

Outro aspeto bastante importante é ajudar os utilizadores a perceberem e recuperarem de erros, deste modo a aplicação deve possuir mensagens de erro informativas, simples, e caso possível, devem sugerir uma solução ao problema.

À frente de cada opção, para além de informação adicional sobre essa configuração, o sistema exibe ainda cuidados a ter na escolha dessa opção, de forma a explicar ao utilizador o motivo dos erros que possa ter. Para além disso, se ocorrer algum erro será exibida uma mensagem de erro alertando o utilizador do erro em questão.

10. Help and documentation

Finalmente, a última heurística de *Nielsen* especifica que uma boa aplicação deve ser utilizável sem documentação, no entanto esta deve estar disponível caso o utilizador este possua qualquer dúvida sobre o seu funcionamento.

Apesar da simplicidade da aplicação criada, e ao reduzido número de funcionalidades que possui, existe mesmo assim documentação. Para cada uma das opções que o utilizador tem de escolher, existe um ícone de ⓘ, que exibe uma pequena explicação sobre aquela opção, bem como o que deve selecionar e cuidados a ter.

Capítulo 4

Testes e análise de resultados

4.1 Especificações da máquina

Para avaliar a performance da plataforma desenvolvida, correu-se o programa para diferentes jogos com diferentes imagens modelo de forma a detetar e rastrear diferentes objetos. Para se poder comparar os vários jogos, optou-se por deixar o valor *default* de *threshold*, sendo este valor 70. Alterando, no final, este valor num só jogo de forma a avaliar a importância deste valor no programa.

Para correr a plataforma com sucesso é aconselhável o uso de um monitor externo, para se poder colocar o jogo num ecrã e o programa noutra, de forma a visualizar o programa a correr e o jogo em simultâneo. Para além disso, apesar do programa correr em qualquer computador, certas funcionalidades, como a captura de ecrã, só estão adaptadas para Windows, por isso é fundamental a utilização deste sistema operativo. Por último, quanto melhor for o computador, melhor será a deteção e rastreamento dos objetos, dado que consegue fornecer com maior frequência capturas de ecrã ao programa. Assim, o objeto não se desloca com tanta rapidez (já que o programa recebe *screenshots* constante com menores intervalos de tempo entre eles) e, para além disso, o programa consegue ser executado muito mais rapidamente, melhorando obviamente a sua performance.

Para além de um monitor externo, foi utilizado uma máquina com as seguintes especificações:

Nome do equipamento	OMEN by HP-12-an022np
Microprocessador	Intel® Core™ i7-7700HQ (frequência base de 2,8 GHz, até 3,8 GHz com Tecnologia Intel® Turbo Boost, 6 MB de cache, 4 núcleos)
Memória, padrão	SDRAM DDR4-2400 de 16 GB (2 x 8 GB)
Placa de vídeo gráfica	NVIDIA® GeForce® GTX 1050 Ti (GDDR5 dedicada de 4 GB)
Sistema Operativo	Windows 10 Home 64

Figura 40: Especificações da máquina

4.2 Jogos

Para testar a performance do programa a correr optou-se por selecionar diferentes jogos e comparar os resultados entre eles. Escolheram-se jogos leves e com gráficos simples, de modo a que a máquina não ficasse sobrecarregada e fosse possível correr corretamente o programa sem interferir com a usabilidade do jogo. Os jogos eleitos foram PokeMMO, Terraria, Club Penguin e Old School RuneScape.

Inicialmente ponderou-se no jogo Brawlhalla, mas, ao fim de alguns testes, excluiu-se este jogo, uma vez que a câmara deste jogo movimenta-se muito rápido e frequentemente, o que torna difícil o rastreamento de um objeto, uma vez que o tamanho deste está constantemente a variar bem como a sua localização, em adição ao movimento do utilizador.

Todos os jogos escolhidos, exceto Old School RuneScape, são jogos 2D, o que, para além de os tornar mais leves e com gráficos mais simples, diminui a variedade de formas que um objeto pode ter.

4.3 Imagens

A escolha da imagem modelo é fundamental, pois, caso esta não seja a mais indicada, pode influenciar bastante a deteção do objeto. A imagem modelo deve ser uma captura de ecrã do objeto, para que este esteja no tamanho e ângulo correto, pois caso este mude de tamanho ou direção o *template matching* terá mais dificuldades a detetar.

Esta imagem deve conter o menor possível de ruído de *background*, isto é, não deve possuir fundo para além do objeto. É preferível que se corte um pouco ao objeto para que este não apresente fundo que confunda o *template matching*. Caso a imagem apresente demasiado ruído de fundo, o *template matching* tentará detetar o fundo em vez do objeto. Também não é possível remover o fundo para só se ter o objeto, pois, neste caso, o *template matching* tentará encontrar no jogo pixéis a `null`, que claramente não existe, e não encontrará o objeto.

De seguida apresenta-se uma série de imagens inadequadas como imagens modelos passadas ao programa, bem como a imagem ideal que deveria ser escolhida para o objeto em causa.



Figura 41: Regra de escolha de imagem modelo

Na figura 41 é demonstrado um objeto do jogo Terraria e é referente a uma fogueira com efeito, isto é, a chama oscila e tem fumo em cima. Contudo, se se quiser detetar este objeto não convém selecionar a parte que oscila e varia de posição, pois o *template matching* terá dificuldades em detetar este objeto. Desta forma, o procedimento correto a seguir é selecionar a parte central da fogueira, que será a mesma em ambos os casos. Também, de forma alguma, se deve eliminar o *background*.

Em princípio, para objetos de maior dimensão, que consequentemente possuem imagens modelos maiores, o programa apresenta melhor performance ao nível do *template matching*, dado que o número de imagens modelo que cabem dentro de uma captura de ecrã é menor. Desta forma, o *template matching* terá de analisar menos imagens de cada vez que é corrido, conseguindo obter tempos e consecutivamente FPS melhores.

4.4 Threshold

O valor de *threshold* é referente à precisão que o *template matching* tem sobre os objetos encontrados. Se a precisão for inferior ao valor de *threshold*, então esse objeto será descartado. Assim sendo, este valor é o limite máximo inferior de precisão que o algoritmo pode ter relativamente a uma imagem. Por um lado, se este valor for demasiado alto, então o algoritmo, pode não ser capaz de reconhecer todos os objetos no ecrã. Por outro, se for demasiado pequeno, então o algoritmo pode reconhecer 'objetos' que não são o indicado.



Figura 42: Objetos detetados com diferentes valores de *threshold*

Testou-se para um mesmo jogo com a mesma imagem modelo, diferentes valores de *threshold* (97, 70 e 30), como se pode ver na figura 42. A imagem modelo era referente a umas flores que apareciam repetidas no jogo, de forma ao *template matching* poder reconhecer vários objetos em simultâneo. Para o valor de 97 apenas foi detetado apenas uma das flores. Para o valor de 70 foram detetadas as quatro flores que apareciam no ecrã. E, para o valor de 30, para além das 4 flores que existiam, foram detetadas erradamente mais objetos.

O valor *default* considerado foi 0.7, pois foi o valor que apresentou os resultados pretendidos. Apesar de, na maior parte dos jogos, uma variação no *threshold* não apresentar mudanças significativas, existem outros que podem afetar o número de objetos detetados, principalmente em jogos que possuam objetos parecidos mas não idênticos.

Um outro fator que o *threshold* influencia é a performance do programa geral. Caso o *threshold* seja demasiado baixo, então o *template matching* detetará mais objetos. Consequentemente o *object tracking* terá de rastrear mais objetos simultaneamente e obviamente será um processo mais demorado e exigente em termos de performance. Assim, o programa será mais lento e vai correr com menos FPS. Assim sendo, não é aconselhado diminuir muito o valores de *threshold*, pois a performance do programa pode ficar comprometida.

4.5 Resultados obtidos e a sua análise

Para analisar os resultados obtidos, foram gravados uma série de pequenos episódios (cerca de um minuto) de vários jogos, com várias imagens modelos. Estes episódios serviram para poder avaliar com maior precisão a performance do programa. Apesar de a maior parte da avaliação ser um pouco subjetiva, visto não haver muitas métricas para comparação, foram tidas em conta três métricas distintas para uma melhor análise dos resultados:

- `Total time ran (s)`: Tempo total da gravação de cada episódio;
- `Average FPS`: FPS médios que o programa obteve (independente dos FPS do jogo);
- `Average detection precision`: Precisão média dos objetos detetados (apenas é referente ao *template matching*, independente do *object tracking*).

O tempo serve para avaliar que a duração dos episódios. É um parâmetro fundamental para a análise das médias e é essencial que os episódios possuam tempos relativamente parecidos entre si, de forma a que o facto do programa correr mais ou menos tempo não influenciar os resultados nem a performance da plataforma.

Um dos parâmetros mais fundamentais é a medição dos FPS, pois, o objetivo seria obter o maior valor possível. Após alguns ajustes feitos, como a adição de *threads* para a execução do GOTURN, já explicados anteriormente na secção 3.3.1, conseguiu-se ter valores razoáveis de FPS no decorrer do programa. Este parâmetro não corresponde aos FPS do jogo, corresponde aos *frames* por segundo que o programa consegue analisar e exibir ao jogador.

Obviamente este valor varia consoante o poder computacional (*single-core*) da máquina usada. Principalmente porque em *Python* existe o *Global Interpreter Lock* (GIL) que permite que apenas uma só *thread* consiga aceder ao interpretador. Isto serve para impedir que problemas de multi-processamento aconteçam, como por exemplo *data races* (ocasião em que duas ou mais *threads* mudam o valor de uma variável ao mesmo tempo e, no entanto, apenas uma das alterações persiste).

Desta forma, quando existe código *multi-threaded*, na verdade, o código será executado concorrentemente devido a só possuir acesso a um core. Todavia, existem na mesma casos em que compensa a utilização de *threads*. Por exemplo, se uma *thread* tiver de aceder ao disco para transferir dados para memória RAM, então este processo pode passar para *background* e o processador pode prosseguir para a próxima *thread*, retomando à anterior quando esta já possuir a informação necessária em RAM. Este

caso em concreto acontece no programa desenvolvido quando os *trackers* necessitam de carregar do disco o modelo Caffe (modelo de tamanho considerável) para poderem rastrear o objeto.

Em suma, independentemente do número de cores que o computador tenha, o programa apenas usará um deles e a performance será dependente apenas do poder computacional *single-core* da máquina. Sabendo isto, optou-se pela utilização de uma máquina com poder computacional intermédio, pois será o computador médio que os jogadores comuns possuem.

Por último, foi adicionada a métrica que calcula a precisão média da deteção do objeto por parte do *template matching*. Assim que se corre o *template matching*, para cada possível objeto detetado é associado um valor de certeza que o algoritmo tem relativamente ao objeto. Caso esse valor seja inferior ao *threshold*, então o objeto será descartado e não será detetado nem rastreado. Caso seja superior, será passada a sua posição ao algoritmo de rastreamento para seguir a sua posição. Esta métrica é então uma média dos valores de certeza obtidos pelo *template matching* quando o objeto é detetado com sucesso, ou seja, quando esse valor é superior ao valor de *threshold*. Como o valor de *threshold* usado foi o *default* (0.7), então será de esperar que todos os valores de deteção sejam superiores a este valor, porque os valores inferiores a 0.7 são descartados. Assim, esta métrica é correspondente à precisão média dos objetos detetados com sucesso.

Infelizmente, não foi possível avaliar a precisão do rastreamento, já que o algoritmo usado não dispunha de nenhuma métrica de análise nem exibe a certeza e precisão do rastreamento dos objetos.

Primeiramente realizou-se para cada um dos jogos selecionados, a deteção de um único objeto em simultâneo. Para isso, escolheram-se objetos que não apareceriam repetidos para avaliar a performance do programa na deteção e rastreamento de um único objeto. Para cada um dos objetos fez-se uma avaliação separada para caso o utilizador esteja parado (não há qualquer movimentação no ecrã) e quando este se desloca.

PokeMMO	Entrada casa	Parado	Total time ran	52.68949818611145
			Average FPS	8.623833750628359
		Com deslocação	Average detection precision	0.7937082280790019
			Total time ran	63.173726320266724
	NPC	Parado	Average FPS	8.546318740463136
			Average detection precision	0.7945850585953217
		Com deslocação	Total time ran	25.57032322883606
			Average FPS	9.276277418506359
	Terraria	Parado	Average detection precision	0.8323273181915282
			Total time ran	44.99509620666504
		Com deslocação	Average FPS	9.320239438915547
			Average detection precision	0.8267340768467297
Terraria	Fogueira	Parado	Total time ran (s)	61.72688412666321
			Average FPS	9.513317718253393
		Com deslocação	Average detection precision	0.7707452003161112
			Total time ran (s)	64.82143139839172
	Player	Parado	Average FPS	9.120779094596946
			Average detection precision	0.7667447709861925
		Com deslocação	Total time ran (s)	61.189706563949585
			Average FPS	9.233965243386528
	Club Penguin	Parado	Average detection precision	0.791940254589607
			Total time ran (s)	82.39102506637573
		Com deslocação	Average FPS	9.421452551820618
			Average detection precision	0.7922628465536479
Club Penguin	Tronco	Sem oclusão	Total time ran	78.62544083595276
			Average FPS	6.754300506451045
		Com oclusão	Average detection precision	0.8053013704157647
			Total time ran	81.31485033035278
	Player	Parado	Average FPS	6.507609808025265
			Average detection precision	0.7987072923026249
		A rodar	Total time ran	64.13719463348389
			Average FPS	6.933456623682058
	Andar várias dimensões	Parado	Average detection precision	0.7693688631057738
			Total time ran	76.59201669692993
		A rodar	Average FPS	6.980802261999501
			Average detection precision	0.7915436781786537
		Andar várias dimensões	Total time ran	58.75576138496399
			Average FPS	7.140852307473007
			Average detection precision	0.7668212992804391

Figura 43: Resultados obtidos da deteção de um único objeto em jogos 2D

Pelos resultados obtidos, foi possível comprovar que as gravações apresentam tempos de duração semelhantes e os FPS rondam os 8 FPS, dependendo mais do jogo e da imagem modelo do que propriamente se o jogador estava parado ou a deslocar-se. A precisão média, tal como referido anteriormente, encontra-se acima do valor de *threshold* definido, rondando os 0.8. Foi possível compreender que imagens com demasiado ruído de fundo, tamanho demasiado reduzido ou abstratas (sem contraste com *background* ou que possua formas estranhas difíceis de serem reconhecidos como objetos pelo GOTURN) não são boas

escolhas para imagem modelo, pois não se consegue realizar o rastreamento do objeto.

Para além dos valores apresentados, após algumas observações do programa em execução foi possível tirar mais conclusões que forneceram mais informação sobre os problemas e limitações do programa. Em todos os casos foi possível perceber que a deteção do objeto é realizada com sucesso. Como se pode ver na figura 44, os objetos são delimitados corretamente e a maior parte das falhas ocorriam ao nível do rastreamento.

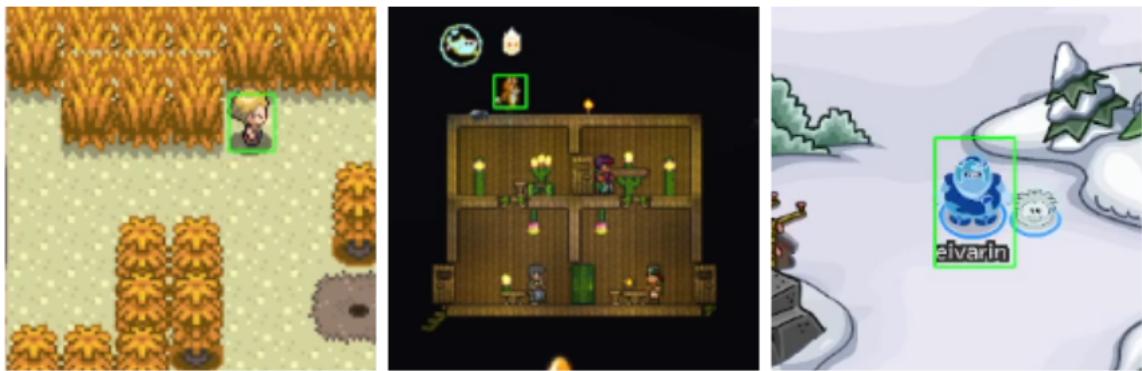


Figura 44: Deteção dos jogadores nos diversos jogos

Quando tanto o utilizador como o objeto estão parados, é possível confirmar que tanto a deteção como o rastreamento funcionam corretamente. O rastreamento nunca é tão preciso como a deteção e oscila sempre um pouco ao longo da gravação, mesmo sem haver nenhum movimento, o que seria de esperar. Quando de facto existe movimento, se este for demasiado rápido, o *tracker*, por vezes, começa, por engano, a rastrear um objeto parecido ou que se encontra na antiga posição do objeto a rastrear inicialmente. Isto deve-se ao facto de o programa não conseguir processar o ecrã do jogador à velocidade que o jogo corre. Passa a receber informação atrasada, que leva a que o *tracker* receba como informação a de rastrear o objeto numa determinada posição quando este já se encontra noutra. Para evitar estes erros, o programa de 10 em 10 iterações confirma através do *template matching* que está a detetar o objeto correto. É possível confirmar que de facto existe esta correção, pois sempre que o *tracker* começa a rastrear o objeto incorreto, rapidamente o programa corrige o problema.

Uma das vantagens do *tracker* relativamente à deteção do objeto, é percetível quando o objeto roda ou fica oculto, como se visualiza na figura 45. Quando o objeto roda e deixa de estar no mesmo ângulo em que estava quando a imagem modelo foi tirada ou quando fica oculto por outro objeto, seria de esperar que o objeto deixasse de ser detetado. Porventura, se já tiver sido detetado e for o *tracker* a fazer o seu rastreamento, este consegue lidar consideravelmente bem com oclusões parciais e rotações do objeto em causa, desde que estas rotações sejam feitas progressivamente. Caso o objeto seja oculto na totalidade,

por vezes o *tracker* começa a perseguir o objeto que o ocultou e pode demorar a detetar novamente o objeto pretendido se este continuar escondido.



Figura 45: Detecção de objeto parcialmente oculto e a rodar

Após realizados os testes com deteção de um único objeto, prosseguiu-se à realização de testes com deteção de vários objetos iguais simultaneamente. Para isso escolheram-se objetos que se sabia que iam aparecer repetidos no ambiente do jogo testado. Os resultados obtidos foram os seguintes:

2D - Multi Object				
PokeMMO	Entrada casa	Parado	Total time ran (s)	53.92001724243164
			Average FPS	5.462278717509264
			Average detection precision	0.7910907319820292
	Buraco	Parado	Total time ran (s)	64.83453869819641
			Average FPS	5.5884727294514756
			Average detection precision	0.7887142776884002
	Fogueira	Parado	Total time ran (s)	86.25428462028503
			Average FPS	3.366850575823551
			Average detection precision	0.7953701844747119
	Banco	Parado	Total time ran (s)	85.86637902259827
			Average FPS	3.2749663227379955
			Average detection precision	0.7947127002177469
Terraria	Fogueira	Parado	Total time ran (s)	65.93006873130798
			Average FPS	5.985481984085379
			Average detection precision	0.765900571194906
	Com deslocação	Parado	Total time ran (s)	93.84486746788025
			Average FPS	5.410573101205185
			Average detection precision	0.7663050793832348
	Banco	Parado	Total time ran (s)	62.803534269332886
			Average FPS	4.69372997578765
			Average detection precision	0.914859899216228
	Com deslocação	Parado	Total time ran (s)	62.11498308181763
			Average FPS	4.633953952909821
			Average detection precision	0.8321289351353278

Figura 46: Resultados obtidos da deteção de vários objetos em jogos 2D

A partir dos resultados apresentados e visualizando o programa a correr para os referidos objetos é possível comprovar que os FPS diminuem quando o programa deteta mais do que um objeto. Isto já seria de prever, pois o rastreamento é um processo complexo e a existência de vários objetos obviamente demora mais e necessita de maior poder computacional. Apesar desta diminuição já ser considerável, é ainda maior nos casos em que a imagem é pequena, como no Buraco do PokeMMO e o banco de Terraria. Nestes casos, para além de haver uma diminuição mais acentuada ao nível dos FPS, também a deteção e rastreamento ficam comprometidas.



Figura 47: Deteção de um Objeto vs Deteção de vários objetos

No caso do buraco, como é uma imagem relativamente pequena e abstrata e existem vários buracos seguidos, como se pode ver na figura 48, o *tracker* não consegue detetar quando acabava um buraco e começava outro. Nestes casos, o *tracker* começa a abranger mais do que um buraco e perdendo o objeto original de vista.

Se já existe alguma dificuldade no rastreamento destes objetos quando não existe movimento, então quando o jogador se desloca, o rastreamento ainda se torna mais lento e com mais incorreções. O programa deixa de ter FPS suficientes para detetar corretamente a deslocação do objeto e como este é pequeno, a sua deslocação será ainda maior. O *tracker* começa a procurar objetos na antiga posição do objeto e as caixas começam a aumentar o seu tamanho.

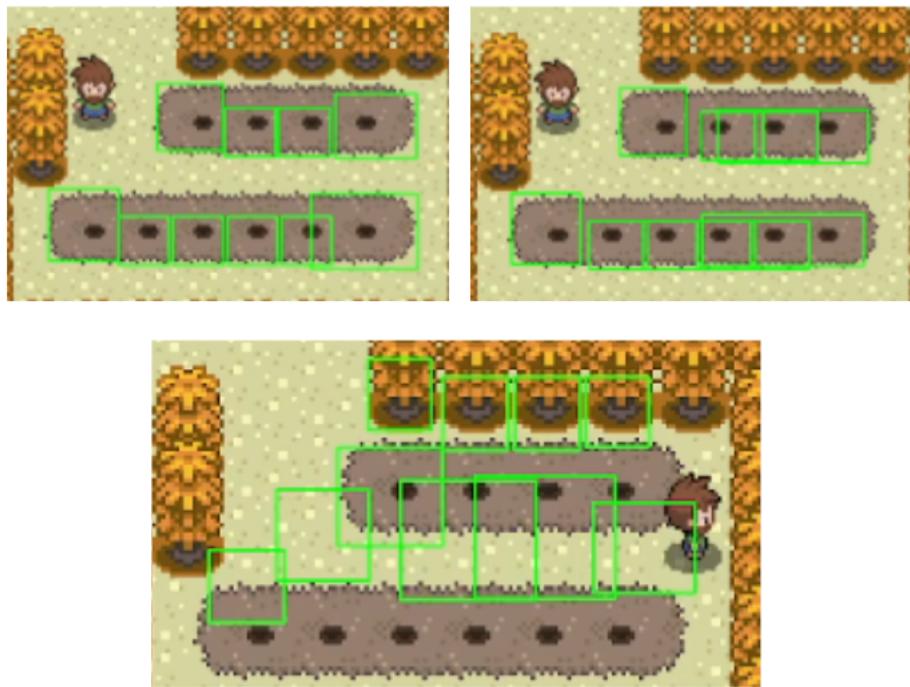


Figura 48: Deteção de vários objetos com deslocação

É de salientar que a precisão de deteção é apenas referente ao *template matching* e não ao algoritmo de rastreamento. Apesar desta precisão poder ser elevada em objeto pequenos, quando o *tracker* tenta seguir o movimento destes objetos, a precisão diminui e o objeto deixa de ser detetado.

Por último, testou-se a deteção e rastreamento de objetos num jogo em 3D, com o intuito de perceber a performance do programa comparativamente aos jogos 2D. Escolheu-se para isso um jogo leve com gráficos simples de fácil deteção. De seguida apresenta-se os resultados obtidos tanto para a deteção de um só objeto como a deteção de vários objetos em simultâneo.

3D - Old School RuneScape				
Single Object Detection	Plantinha	Parado	Total time ran (s)	69.02053570747375
			Average FPS	7.740475441602711
			Average detection precision	0.7876371870560178
	Com deslocação	Rodar a câmara	Total time ran (s)	131.48871207237244
			Average FPS	8.74953795061742
			Average detection precision	0.7261663639090122
	NPC	Parado	Total time ran (s)	53.93157982826233
			Average FPS	8.600921779277108
			Average detection precision	0.7372377660241065
	Com deslocação	Rodar a câmara	Total time ran (s)	67.5277967453003
			Average FPS	9.088873660054604
			Average detection precision	0.7684832044384066
Multi Object Detection	Cactus	Parado	Total time ran (s)	114.1725697517395
			Average FPS	9.205990828720354
			Average detection precision	0.7118100023069305
	Rodar a câmara	Rodar a câmara	Total time ran (s)	82.06568670272827
			Average FPS	8.961611265868878
			Average detection precision	0.7347990935425868
Multi Object Detection	Cactus	Parado	Total time ran (s)	112.75975346565247
			Average FPS	4.943304341057386
			Average detection precision	0.7613714887620581
	Rodar a câmara	Rodar a câmara	Total time ran (s)	97.5064857368469
			Average FPS	5.271991861169463
			Average detection precision	0.737730607500783

Figura 49: Resultados obtidos da deteção objetos num jogo 3D

Em geral, para este jogo 3D, os FPS do programa são tão bons ou melhores que os jogos 2D testados.

Mesmo quando o utilizador se desloca, como o faz lentamente, o programa consegue rastrear o objeto relativamente bem. Porventura, como é possível rodar a câmara, se o *tracker* perder o objeto de vista, é muito difícil o programa voltar a detetá-lo, pois é necessário que o objeto esteja num ângulo próximo ao da imagem modelo. Isto ainda se torna mais evidente quando se tenta rastrear mais do que um objeto simultaneamente. Como se verifica na imagem 50, se inicialmente se detetarem as diferentes plantas, o *tracker* ainda consegue manter o rastreamento da planta quando a câmara roda. Contudo, se perder o objeto de vista é bastante difícil de o voltar a detetar, principalmente porque o objeto em questão muda bastante de forma quando rodado. Apesar de existirem algumas plantas, quando estas não estão na mesma posição/ângulo que a da imagem modelo, o algoritmo não as reconhece como sendo o mesmo objeto.

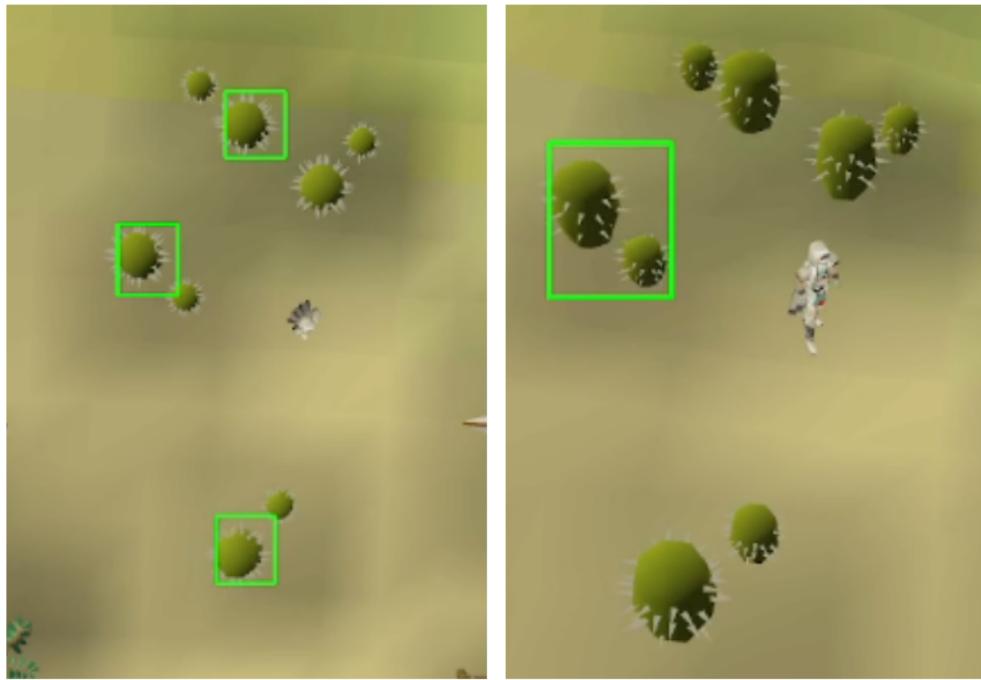


Figura 50: Deteção de multi-objetos com rotação

4.6 Conclusão dos resultados obtidos

Em suma, pelos testes efetuados foi possível comprovar que o algoritmo de *template matching* consegue detetar os objetos pretendidos com sucesso. O programa funciona corretamente tanto para jogos 2D como para jogos 3D e para a deteção de um ou vários objetos em simultâneo. Para além disso, consegue ainda rastrear objetos mesmo em jogos 3D com mudanças de câmara (variações de ângulo) desde que a imagem já tenha sido previamente detetada. De igual forma, se já tiver sido detetado o objeto, consegue-se manter o rastreamento do objeto mesmo que este fique parcialmente oculto. Foi ainda possível comprovar que a alteração de *threshold* influêncie de facto o número de objetos detetados.

Por outro lado, a performance da plataforma ficou um pouco aquém do esperado, principalmente no rastreamento de múltiplos objetos. A pouca performance influencia o funcionamento do *tracker* e, uma vez que esta é baixa, o *tracker* nem sempre funciona da melhor forma. Por último, se os objetos não estiverem no ângulo certo, o *template matching* não será capaz de detetar o objeto pretendido.

Capítulo 5

Conclusões e trabalho futuro

Neste capítulo é feita uma análise de todo o trabalho desenvolvido. São expostos para cada objetivo definido inicialmente, os principais prontos, problemas e dificuldades obtidas e respetivas soluções aos problemas encontrados.

Por último, são sugeridas algumas melhorias e informação a acrescentar que poderiam ser feitas num trabalho futuro.

5.1 Conclusões

A presente dissertação de mestrado permitiu a criação de uma plataforma capaz de detetar e rastrear um objeto presente no ecrã de um jogador, utilizando para isso algoritmos de *machine learning* e *computer vision*. Os objetivos que foram estipulados inicialmente foram cumpridos na sua totalidade, apesar de possuírem algumas limitações.

O primeiro objetivo definido é referente à criação de uma aplicação/agente que recolhe informação contida no ecrã do jogador, em tempo real. Este ponto foi facilmente alcançado e para a sua realização inicialmente foi utilizada a biblioteca `pyautogui`, mas como esta não apresentava performance suficiente, foi alterada para `Win32Gui`. Nenhuma das implementações apresentou grandes dificuldades. A maior limitação existente neste patamar foi que a biblioteca final utilizada não permite interagir com a janela selecionada, apenas recolher a informação lá contida. Para o projeto desenvolvido este fator não é uma entrave, mas para possíveis trabalhos futuros, sugeridos à frente, pode ser um problema.

O segundo objetivo é a criação de uma aplicação/agente que detete, em tempo real, um objeto no ecrã do jogador, utilizando a informação contida previamente. Apesar de cumprido o objetivo, este foi um dos pontos mais problemáticos, pois foi necessária a alteração do algoritmo inicialmente proposto. Começou-se pela implementação do algoritmo de *object detection*, mais especificamente o YOLO, escondido previamente por apresentar uma performance elevada. Após a concretização da sua implementação,

apercebeu-se que este algoritmo apenas detetava cerca de 80 objetos e a maior parte deles não eram reconhecidos em ambientes de jogos que era o principal alvo da plataforma a desenvolver.

Desta forma, exploraram-se novas alternativas a este problema e a solução encontrada foi a utilização do *template matching*. O *template matching* é um algoritmo que encontra uma imagem numa outra imagem maior. Como não necessita de treinar objetos para a sua utilização, permite detetar qualquer objeto independentemente do seu conhecimento sobre ele. Assim, enquanto que no YOLO o utilizador estaria limitado aos objetos treinados pelo algoritmo, com a implementação desta técnica o jogador pode escolher qualquer objeto para detetar. Todavia, apresenta algumas desvantagens, no YOLO era possível detetar vários objetos diferentes numa mesma imagem, enquanto que no *template matching* apenas se pode reconhecer um tipo de objetos (apesar de se detetar múltiplos objetos idênticos). Para além disso, se o objeto não estiver no mesmo ângulo da imagem modelo, então a probabilidade do algoritmo reconhecer o objeto será menor. Mesmo assim, preferiu-se a utilização desta técnica, pois considerou-se mais importante abranger todos os jogos, do que arranjar ou treinar um *dataset* para um jogo em específico.

O terceiro objetivo estipulado é a criação de uma aplicação/agente que rastreie o objeto detetado previamente em tempo real. Já implementado o algoritmo que deteta o objeto, passou-se à fase de rastrear o objeto detetado. Geralmente, rastrear um objeto é mais eficiente que detetar, em todos os frames, pelo mesmo objeto. À partida, o algoritmo de rastreamento já tem informação da localização do objeto no frame anterior. Eleito o algoritmo GOTURN, existente na biblioteca de OpenCV, realizou-se a sua execução e agregou-se ao *template matching* já aplicado.

Posto em execução facilmente se concluiu que a performance não era a esperada, pois, mesmo sem a realização de testes, o programa encrava bastante, rondando os 1 e 2 fps, chegando mesmo a atingir valor de 0 fps. Como tal, procurou-se melhorias possíveis ao programa e optou-se por implementar *threads*, de maneira os *trackers* concorrentemente. Esta vantagem seria benigna principalmente para o rastreamento de múltiplos objetos, contudo viu-se uma aumento significativo da performance, para cerca de 5 fps. Apesar de não ser um número elevado, é mais do dobro do máximo obtido anteriormente. Mesmo assim existem alguns problemas e limitações, caso o objeto se movimento rapidamente, o GOTURN continua sem o conseguir rastrear corretamente. De igual forma, se a imagem modelo fornecida não for a mais indicada (apresentar ruído) a deteção e o rastreamento podem ser incorretos. Conclui-se que este objetivo foi cumprido, apesar de apresentar várias limitações e ser a principal restrição do programa.

O último objetivo delimitado é a criação de uma plataforma para facilitar o uso da aplicação criada até então. Desta forma, qualquer utilizador pode desfrutar da deteção e rastreamento de um objeto através

de uma interface de fácil comunicação e simples de usar. Para a realização desta tarefa começou-se pela prototipagem da plataforma, de modo a testar o bom funcionamento da aplicação e potenciais problemas ainda antes de a codificar. O próximo passo seguiu para o desenvolvimento da plataforma, tendo por base o protótipo criado. A maior dificuldade sentida nesta fase foi a comunicação entre a camada de apresentação(*frontend*) e a camada de lógica de negócios(*backend*), pois foi necessário desenvolver uma API para fazer a ligação entre elas.

Em suma, o principal objetivo foi cumprido, a plataforma consegue detetar e rastrear com sucesso um ou vários objetos. Caso o objeto já tenha sido previamente detetado consegue manter o rastreamento mesmo que o objeto fique parcialmente oculto, mude de forma ou de ângulo. No final, a performance da plataforma ficou um pouco aquém do esperado, principalmente no rastreamento de múltiplos objetos. O facto da performance geral influenciar o funcionamento do *tracker*, impede que este funcione da melhor forma, levando a que por vezes não consiga rastrear o objeto corretamente.

Foi de facto interessante a realização deste projeto, pois não existia no mercado nenhuma plataforma deste estilo disponível à comunidade. Esta plataforma permite a qualquer utilizador, jogador ou não jogador, detetar um determinado objeto à sua escolha sem a necessidade de treinarem previamente um algoritmo para o objeto em questão. Com a informação obtida a partir da plataforma, o utilizador pode realizar alguma medida ou ação relativamente ao objeto ou a algo que interaja com ele. É deixado à imaginação do utilizador todas as possíveis utilizações da plataforma ou do programa realizado.

5.2 Perspetiva de trabalho futuro

Concluído o presente trabalho de dissertação surgiram algumas melhorias que poderiam ser feitas como trabalho futuro.

Antes de mais poder-se-ia adicionar mais configurações à plataforma. Foram escolhidas as essenciais, contudo poder-se ia adicionar opções como a escolha da métrica do *template matching* ou até o número máximo de objetos que o utilizador pretende que a plataforma detete.

Poder-se-ia ainda permitir que o utilizador escolhesse mais que uma imagem diferente. Desta forma, o programa podia detetar e rastrear diferentes objetos em simultâneo. Esta opção não foi considerada para o projeto atual, pois, já que o projeto a detetar só uma imagem tinha pouca performance, adicionar ainda mais imagens para detetar e rastrear diminuiria ainda mais o desempenho do programa. Porém, se um utilizador possuir uma máquina boa, pode conseguir correr o programa com as duas imagens distintas.

Como um dos principais problemas é o desempenho do programa, seria um fator a melhorar. Existem algumas opções encontradas ao fim de alguma pesquisa, porventura, como foram feitas numa fase tardia foi impossível a implementação destas no projeto.

Uma delas seria a implementação dos algoritmos na gráfica. É possível usar o GPU para otimizar os algoritmos de *computer vision*, no entanto, o uso deste requerer a instalação de software da nvidia e recompilação do opencv para python com essas configurações.

Em segundo, podia-se colocar o backend a correr em paralelo. Uma vez que o GIL de python limita os benefícios de usar threads, podia-se em vez de ter um tracker por thread ter um tracker por processo. Esta sugestão seria possível através da utilização de um módulo de multiprocessing de python, que basicamente permite a criação de vários processos. Cada um destes processos tem o seu próprio GIL e pode correr paralelamente em núcleos diferentes do processador.

Por último, como forma de tornar este projeto mais útil e integrá-lo com outras plataformas podia-se adicionar alguma ação a fazer caso se encontrasse um objeto, como por exemplo, clicar nele ou fazer com que o jogador se deslocasse para lá. Esta opção exige a exploração de bibliotecas de python que permitam interagir com o jogo, algo que de momento não é possível com a plataforma.

Bibliografia

Papers with code - coco benchmark (real-time object detection), a. URL <https://paperswithcode.com/sota/real-time-object-detection-on-coco>. accessed on 29/01/2022.

Common objects in context, b. URL <https://cocodataset.org/#home>.

Darknet: Yolo. URL <https://github.com/pjreddie/darknet>.

Opencv, a. URL <https://github.com/opencv>.

About. OpenCV, b. URL <https://opencv.org/about/>. accessed on 20/01/2022.

Cv::trackergoturn class reference, c. URL https://docs.opencv.org/3.4/d7/d4c/classcv_1_1TrackerGOTURN.html.

Releases. OpenCV, d. URL <https://opencv.org/releases/>. accessed on 10/01/2022.

Template matching, e. URL https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html.

Cachorro mais rápido do mundo: Saiba mais!: Petz. Blog Petz, Sep 2021. URL <https://www.petz.com.br/blog/cachorros/cachorro-mais-rapido-do-mundo/>. accessed on 19/01/2022.

Aprenda o que É figma e vantagens para design de interfaces, May 2021. URL <https://www.digitalhouse.com.br/blog/o-que-e-figma/>. accessed on 08/05/2022.

Taiwo Oladipupo Ayodele. Types of machine learning algorithms. *New advances in machine learning*, 3: 19–48, 2010.

Jason Brownlee. A gentle introduction to computer vision. Machine Learning Mastery, Mar 2019. URL <https://machinelearningmastery.com/what-is-computer-vision/>. accessed on 10/01/2022.

Jason Brownlee. A gentle introduction to object recognition with deep learning, Jan 2021. URL <https://machinelearningmastery.com/object-recognition-with-deep-learning/>.

Davide Carneiro, André Pimenta, José Neves, and Paulo Novais. A multi-modal architecture for non-intrusive analysis of performance in the workplace. *Neurocomputing*, 231:41–46, 2017.

Ian Castelli. Mais 5 ilusões de ótica que vão confundir o seu cérebro, Feb 2019. URL <https://www.megacurioso.com.br/ilusao-de-optica/69419-mais-5-ilusoes-de-optica-que-vao-confundir-o-seu-cerebro.htm>. accessed on 19/01/2022.

Nađa Dardagan, Adnan Brđanin, Džemil Džigal, and Amila Akagic. Multiple object trackers in opencv: A benchmark. In *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, pages 1–6. IEEE, 2021.

Rostislav Demush. A brief history of computer vision (and convolutional neural networks). HackerNoon, Feb 2019. URL <https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3>. accessed on 10/01/2022.

H Deshpande, A Singh, and H Herunde. Comparative analysis on yolo object detection with opencv. *International Journal of Research in Industrial Engineering*, 9(1):46–64, 2020.

Matteo Duò. Figma vs sketch: Uma comparação de funcionalidades de ambas ferramentas de design, Sep 2022. URL <https://kinsta.com/pt/blog/figma-vs-sketch/>.

IBM Cloud Education. What are convolutional neural networks? IBM, Oct 2020a. URL <https://www.ibm.com/cloud/learn/convolutional-neural-networks>. accessed on 20/01/2022.

IBM Cloud Education. Neural networks. IBM, Aug 2020b. URL <https://www.ibm.com/cloud/learn/neural-networks>. accessed on 10/01/2022.

Mohamed Elgendi. *Deep learning for vision systems*. Simon and Schuster, 2020.

Erfan. Electron.js framework, its advantages and disadvantages - framework, Jun 2021. URL <https://ded9.com/introducing-the-electron-js-framework-its-advantages-and-disadvantages/>. accessed on 06/06/2022.

Wolfgang Ertel. *Introduction to artificial intelligence*. Springer, 2018.

Rohith Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo - object detection algorithms. Medium, Jul 2018. URL <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>. accessed on 29/01/2022.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Sunila Gollapudi. *Learn computer vision using OpenCV: with deep learning CNNs and RNNs*. Apress, 2019.

Denis Grankin. Pros and cons of vue.js framework programming, Dec 2020. URL <https://ddi-dev.com/blog/programming/the-good-and-the-bad-of-vue-js-framework-programming/>. accessed on 08/05/2022.

David Held, Sebastian Thrun, and Silvio Savarese.

David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. pages 749–765, 2016.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959.

Peter Janku, Karel Koplik, Tomas Dulik, and Istvan Szabo. Comparison of tracking algorithms implemented in opencv. In *MATEC Web of Conferences*, volume 76, page 04031. EDP Sciences, 2016.

Bharanth K. Object detection algorithms and libraries, Aug 2021. URL <https://neptune.ai/blog/object-detection-algorithms-and-libraries>. accessed on 30/01/2022.

Kang and Atuk. Template matching using opencv, Dec 2020. URL <https://theailearner.com/2020/12/12/template-matching-using-opencv/>. accessed on 12/06/2022.

Jennifer Lesser. The 7 types of dog breeds, Jan 2022. URL <https://www.thesprucepets.com/types-of-dog-breeds-4688776>. accessed on 19/01/2022.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

Thomas Madjour and Mickael Madjour. Artificial intelligence – part 1: Couple of definitions, Oct 2018. URL <https://athis-technologies.com/news/innovation/ai-big-data/2018/artificial-intelligence-part-1-couple-of-definitions/>. accessed on 19/01/2022.

Shishira R Maiya. Deepsort: Deep learning to track custom objects in a video. Nanonets, Apr 2020. URL <https://nanonets.com/blog/object-tracking-deepsort/>. accessed on 30/01/2022.

Yashwanth Medisetti. Neural networks in everyday life. Medium, Feb 2021. URL <https://medium.com/analytics-vidhya/neural-networks-in-everyday-life-ca2b7cb37052>. accessed on 10/01/2022.

Vidushi Meel. 83 most popular computer vision applications in 2022. viso.ai, Mar 2021. URL <https://viso.ai/applications/computer-vision-applications/>. accessed on 15/01/2022.

José Pedro Monteiro, Diogo Ramos, Davide Carneiro, Francisco Duarte, João M. Fernandes, and Paulo Novais. Meta-learning and the new challenges of machine learning. *International Journal of Intelligent Systems*, 36(11):6240–6272, 2021.

Shardul Nalegave. Electron: Pros and cons, Feb 2018. URL <https://medium.com/@nalegaveshardul40/electron-pros-and-cons-8f58fd6313d5>. accessed on 06/06/2022.

Guilherme Neto. Coluna - criadora de league of legends, riot anuncia sete novos games, Oct 2019. URL <https://agenciabrasil.ebc.com.br/esportes/noticia/2019-10/coluna-criadora-de-league-legends-riot-anuncia-sete-novos-games>. accessed on 05/04/2022.

Diogo Ferreira Nunes. Uber obriga motoristas, passageiros e estafetas a usarem máscara. *Diário de Notícias*, May 2020. URL <https://www.dn.pt/dinheiro/uber-obriga-motoristas-passageiros-e-estafetas-a-usarem-mascara-12193002.html>. accessed on 16/01/2022.

André Pimenta, Davide Carneiro, José Neves, and Paulo Novais. A neural network to classify fatigue from human-computer interaction. *Neurocomputing*, 172:413–426, 2016.

José Pedro Pinto, André Pimenta, and Paulo Novais. Deep learning and multivariate time series for cheat detection in video games. *Machine Learning*, 110(11):3037–3057, 2021.

Sharil Abdul Rahman. Cs:go dead? not really, Jul 2021. URL <https://www.gosugamers.net/counterstrike/features/54659-cs-go-dead-not-really>. accessed on 05/04/2022.

Joseph Redmon. Yolo: Real-time object detection. URL <https://pjreddie.com/darknet/yolo/>. accessed on 02/04/2022.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Alexander Reynolds. Understanding and evaluating template matching methods, Sep 2019. URL <https://stackoverflow.com/questions/58158129/understanding-and-evaluating-template-matching-methods>. accessed on 21/04/2022.

Claudio Righetti, Mariela Fiorenzo, Omar Hurtado, and Gabriel Carro. Augmented intelligence - next level network and services intelligence. *CABLE-TEC EXPO Virtual Experience Fall Techinal Forum SCTE, ISBE and NCTA.*, page 12–16, Oct 2020. URL https://wagtail-prod-storage.s3.amazonaws.com/documents/1790_Righetti_3259_paper.pdf.

Rodrigo Rocha, Davide Carneiro, and Paulo Novais. Continuous authentication with a focus on explainability. *Neurocomputing*, 423:697–702, 2021.

Adrian Rosebrock. Opencv object tracking. PyImageSearch, Jul 2018a. URL <https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/>. accessed on 22/01/2022.

Adrian Rosebrock. Yolo object detection with opencv, Nov 2018b. URL <https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/>. accessed on 02/04/2022.

Adrian Rosebrock. Opencv template matching, Apr 2021a. URL <https://pyimagesearch.com/2021/03/22/opencv-template-matching-cv2-matchtemplate/>. accessed on 12/06/2022.

Adrian Rosebrock. Multi-template matching with opencv, Mar 2021b. URL <https://pyimagesearch.com/2021/03/29/multi-template-matching-with-opencv/>. accessed on 12/06/2022.

Sumit Saha. A comprehensive guide to convolutional neural networks. Medium, Dec 2018. URL <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. accessed on 20/01/2022.

Wojciech Semik and Adam Stempniak. Flask vs. django: Which python framework is better for your web development?, Nov 2021. URL <https://www.stxnext.com/blog/flask-vs-django-comparison/>. accessed on 21/05/2022.

Himanshi Singh. How are images stored on a computer? Analytics Vidhya, Mar 2021. URL <https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/>. accessed on 22/01/2022.

KE Swapna. Convolutional neural network: Deep learning. Developers Breach. URL <https://developersbreach.com/convolution-neural-network-deep-learning/>. accessed on 25/01/2022.

Nídia Teiga. O seu cão vira-lhe as costas? saiba porquê! Love Pet Food, Mar 2016. URL <https://lovepetfood.com/blog/o-seu-cao-vira-lhe-as-costas-saiba-porque/>. accessed on 19/01/2022.

Chris Thomas. Walking in the street to maintain a safe distance? it could be against the law, Apr 2020. URL <https://bikeportland.org/2020/04/23/walking-in-the-street-to-maintain-a-safe-distance-its-against-the-law-313944>. accessed on 05/04/2022.

Savan Visalpara, Anant Jain, and Keshav Dhandhania. How do computers see an image? CommonLounge, 2020. URL <https://www.commonlounge.com/discussion/244616b76d3d40f88e8f12103a22743d>. accessed on 22/01/2022.

Alice White. The dog and its body language: What humans need to understand. PetTime, Oct 2021. URL <https://pettime.net/the-dog-and-its-body-language-what-humans-need-to-understand/>. accessed on 19/01/2022.

Bob Woodham, Jim Little, and Fred Tung. Computer vision. *The University of British Columbia*, Lecture 6: Template Matching. URL https://www.cs.ubc.ca/~lsigal/425_2019W2/Lecture6b.pdf.

HuJ. X. Binjie. Fabric testing - template matching. *Fabric appearance testing*, pages 148–188, 2008.

Shivy Yohanandan. Map (mean average precision) might confuse you!, Jun 2020. URL <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2>. accessed on 29/01/2022.