INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
Departamento de Engenharia Eletrotécnica Energia e Automação

Separação de lixo por processamento de imagem

João Centúrio de Almeida Bernardo

(Licenciado em Engenharia Eletrotécnica)

Dissertação para a obtenção do grau de Mestre em Engenharia Eletrotécnica – ramo de Automação e Eletrónica Industrial

Orientador:
Doutora Maria da Graça Vieira de Brito Almeida

Júri:

Presidente:
Doutor Filipe André de Sousa Figueira Barata

Vogais:
Doutora Mafalda Maria Morais Seixas
Doutora Maria da Graça Vieira de Brito Almeida

Junho 2023

Consagre ao Senhor
tudo o que você faz,
e os seus planos serão bem-sucedidos.
Provérbios 16:3

RESUMO

Esta dissertação tem como objetivo o desenvolvimento de uma solução para categorização e separação de resíduos urbanos, utilizando técnicas de computação e processamento de imagem. Para isso, foi necessário adquirir conhecimentos em programação Python, desenvolver um algoritmo de reconhecimento e categorização de lixo urbano e projetar um sistema físico com equipamentos que permitam a separação de resíduos urbanos, utilizando o algoritmo de categorização desenvolvido.

O trabalho teve início com a realização de um estudo sobre técnicas de aquisição e processamento de imagem, seguido de uma análise da realidade portuguesa em relação à recolha de resíduos urbanos. Em seguida, foi desenvolvido um algoritmo de caracterização de símbolos, escrito em Python, utilizando a biblioteca OpenCV, e contendo um modelo de reconhecimento de imagem, o YOLO. Para este modelo ser utilizado para este propósito, teve de ser treinado com imagens categorizadas.

Após treinar o modelo com um conjunto de 500 imagens com 3 categorizações (VERDE, AMARELO e AZUL), o mesmo modelo foi treinado mais duas vezes com parâmetros de treino diferentes. O algoritmo foi colocado em testes com imagens que seriam de um esquema real, e foram obtidos resultados promissores. O terceiro modelo treinado destacou-se por apresentar resultados mais consistentes, mas ainda há espaço para melhoria com o uso de uma base de dados maior.

Embora o trabalho não tenha cumprido todos os objetivos, já que o objetivo do sistema físico foi abandonado, o algoritmo desenvolvido apresenta resultados promissores e pode ser utilizado em outras aplicações.

PALAVRAS-CHAVE

Automação, Processamento e Tratamento de Imagem, Tecnologias de Reconhecimento, Python, Ambiente

ABSTRACT

This dissertation aims to develop a solution for the categorization and separation of urban waste, using computer and image processing techniques. To achieve this, it was necessary to acquire knowledge in Python programming, develop an algorithm for recognition and categorization of urban waste. It would also be necessary to project a physical system with equipment that allows for the separation of urban waste using the developed categorization algorithm.

The work began with a study on image acquisition and processing techniques, followed by an analysis of the Portuguese reality regarding the collection of urban waste. Next, a symbol characterization algorithm was developed, written in Python using the OpenCV library and containing an image recognition model, YOLO. To use this model for this purpose, it had to be trained with categorized images.

After training the model with a set of 500 images with 3 categories (GREEN, YELLOW, and BLUE), the same model was trained twice more with different training parameters. The algorithm was tested with images that would be from a real scheme, and promising results were obtained. The third trained model stood out for presenting more consistent results, but there is still room for improvement with the use of a larger database.

Although the work did not fulfill all the objectives, as the physical system objective was abandoned, the developed algorithm shows promising results and can be used in other applications.

KEYWORDS

Automation, Image Processing and Treatment, Recognition Technologies, Python, Environment.

AGRADECIMENTOS

O presente trabalho só foi possível com o contributo de algumas pessoas que de alguma forma ajudaram-me neste percurso académico.

Em primeiro lugar agradeço à minha orientadora, Doutora Maria da Graça Vieira de Brito Almeida por toda a disponibilidade, apoio, motivação e orientação. A escolha do presente tema deve-se à sua capacidade de abraçar novos desafios.

À minha namorada, Glória Vicente, que caminhou ao meu lado em todo o processo, agradeço com um carinho muito especial a presença, a partilha, a compreensão e o incentivo fundamentais no desenvolvimento deste trabalho.

À minha irmã Joana Bernardo e ao amigo e namorado Nuno Lopes, que me provocavam para eu continuar, ao irmão Bruno Andrade, que rapidamente se disponibilizou nos momentos de dificuldade e dúvida para estar ao meu lado, à irmã Teresa Fernandes que com as suas palavras motivou-me a avançar, ao irmão Pedro Ferreira que orou por mim em todo o percurso e a todos os meus amigos que me foram acompanhando ao longo desta jornada académica e sem os quais chegar aqui teria sido muito mais difícil, deixo palavras de agradecimento e um abraço.

Por último, dirijo um agradecimento especial aos meus pais, por serem modelos de coragem, pelo incentivo, amizade, preocupação e paciência demonstrada, por me terem dado a oportunidade de fazer este caminho académico. A eles dedico este trabalho!

ABREVIATURAS

A/D – Analógico Digital

BSD – Licença de Código Aberto (de Berkeley Software Distribution)

CAN – Controller Area Network

CCD – Charge-Coupled Device

CFA – Color Filter Array

CMOS - Complementary Metal-Oxide-Semiconductor

ECAL – Embalagens de Cartão para Alimentos Líquidos

EPS – Poliestireno expandido

GPU – Graphic Processing Unit

HART – Highway Addressable Remote Transducer Protocol

HDR – High Dynamic Range

HSV – Hue Saturation Value

I/O – Entradas e saídas (de Inputs and Outputs)

IA – Inteligência Artificial

IoT – Internet das Coisas (de Internet of Things)

IOU – Intersecção sobre União

Modelo OSI – Open System Interconnection

MOS – Metal-Oxide-Semiconductor

OPC – Open Platform Communications

PE – Polietileno

PEAD – Polietileno de Alta Densidade

PET – Politereflalato de Etileno

QoS – Qualidade de Serviço (de Quality of Service)

RGB – Red Green Blue

RU – Resíduos Urbanos

SIFT – Scale-Invariant Feature Transform

SURF – Speeded Up Robust Features

YOLO – You Olny Look Once

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

Capítulo 1

INTRODUÇÃO

Neste capítulo é apresentada a Motivação do trabalho, os Objetivos, o Enquadramento do trabalho e a Estrutura da tese.

1.1 – Motivação do trabalho

O estudo e as análises que são feitas ao planeta terra, à biodiversidade e ao ambiente, realizados neste século, chega-se à conclusão que é aconselhável a mudança de hábitos para um futuro melhor. Cada vez mais as palavras tecnologia verde, ambiente e reciclar, estão no nosso vocabulário.

A motivação para esta dissertação prende-se com a ideia de desenvolver um mecanismo que permite reciclar sem intervenção humana, de forma que o lixo colocado nos caixotes seja processado nas centrais sem que as pessoas o separem. Deste modo o processo tornar-se-á mais rápido e sem sujeitar o humano a um conjunto de cheiros desagradáveis.

Com a cuidada seleção e separação das substâncias que são consideradas lixo, consegue-se de uma forma mais eficaz e em maior quantidade reciclar, reutilizando a matéria mais do que uma vez, levando a uma menor exploração ambiental e maior sustentabilidade.

1.2 – Objetivos

O objetivo principal do estudo realizado, é o projeto de desenvolvimento de um sistema automático, programado em Python, de identificação, categorização e separação de um resíduo urbano, substância colocado no contentor do lixo.

A ideia deste sistema contempla um automatismo, que iria receber de um depósito de lixo indiferenciado objetos que necessitariam de ser categorizados, para serem separados por 3 categorias. Para isso recorre-se a um tapete rolante, equipamento de iluminação e captura de imagem, e os 4 depósitos de lixo categorizado (VERDE, AMARELO, AZUL e resíduos não recicláveis).

Para o desenvolvimento desta dissertação são necessários cumprir os seguintes objetivos:

- Desenvolver conhecimentos na linguagem de programação Python.
- Criar um algoritmo de reconhecimento e categorização de lixo urbano.
- Desenvolver um sistema físico com equipamentos que permitam a utilização do algoritmo acima descrito e a separação dos objetos reconhecidos.

1.3 – Enquadramento do trabalho

Este trabalho enquadra-se no curso de mestrado em Engenharia Eletrotécnica, envolvendo conceitos interdisciplinares abrangendo as temáticas de programação, processamento de sinal e automação.

1.4 – Estrutura da tese

Esta dissertação está dividida em seis capítulos:

Capítulo 1: É o capítulo em que se insere este ponto. É feita uma apresentação do tema da dissertação, é descrita a motivação, objetivos e organização desta dissertação.

Capítulo 2: Este capítulo tem o objetivo de sensibilizar o leitor para o problema existente com os resíduos urbanos, explica as metas que Portugal tem de cumprir nos próximos anos, explica ao leitor a caracterização de material reciclável e termina com a sinalética nas embalagens.

Capítulo 3: Neste capítulo é descrito o estado de arte, em que se dá a conhecer a história da automação até aos dias de hoje de seguida aborda-se a temática do processamento e tratamento de imagem que irão ser necessários para a criação do projeto.

Capítulo 4: É apresentado o modelo do projeto e os ensaios que foram feitos.

Capítulo 5: São discutidos os resultados obtidos para cada um dos modelos treinados.

Capítulo 6: São apresentadas as conclusões gerais em função dos objetivos propostos e faz-se a perspetiva de trabalhos futuros.

A dissertação contém além dos capítulos referidos: o índice de capítulos, índice de figuras, abreviaturas, bibliografia e anexo.

Capítulo 2

ECOLOGIA VERDE

Neste capítulo é apresentado o tema e problemática que levou ao tema da tese. A ideia é consciencializar o leitor para as regras e metas que haveriam de ser cumpridas, bem como os trâmites legais associados.

2.1 – Tomada de consciência

Produzem-se anualmente centenas de milhões de toneladas de resíduos, muitos deles não biodegradáveis: resíduos domésticos e comerciais, detritos de demolições, resíduos clínicos, eletrónicos e industriais, resíduos altamente tóxicos e radioativos. A poluição produzida pelos resíduos engloba perigos presentes em variados ambientes. A Terra, nossa casa, aparenta transformar-se cada vez mais num imenso depósito de lixo. Em muitos lugares do planeta, os idosos recordam com saudade as paisagens de antigamente, que agora veem submersas de lixo. Tanto os resíduos industriais como os produtos químicos utilizados nas cidades e nos campos podem produzir um efeito de bioacumulação nos organismos dos moradores nas áreas limítrofes, que se verifica mesmo quando é baixo o nível de presença dum elemento tóxico num lugar. Muitas vezes só se adotam medidas quando já se produziram efeitos irreversíveis na saúde das pessoas. [1]

Infelizmente só quando as situações se tornam demasiado graves é que o publico, em geral, toma consciência do mal que se está a fazer no planeta Terra. Felizmente cada vez mais gente está atenta a esta situação, e deseja-se que cada um seja de uma forma consciente um agente na melhoria do ambiente.

2.2 – Relatório Agência Portuguesa Ambiente

Com a apresentação de alguns dados deste relatório [2] pretende-se salientar o estado dos resíduos em Portugal continental.

2.2.1 – **Recolha de Resíduos Urbanos**

A Figura 1 demonstra a produção de resíduos urbanos ao longo dos anos, desde 2014 a 2021, em Portugal Continental. Em 2021 foram produzidas em Portugal **5,311 milhões de toneladas** (t) de resíduos urbanos (RU), mais 1% do que em 2020, verificando-se um ligeiro aumento na produção, quando comparado com o ano anterior.

Região	2014	2015	2016	2017	2018	2019	2020	2021
PT Continental	4 474	4 523	4 640	4 745	4 945	5 007	5 014	5 043
RA Madeira	110	110	119	124	126	129	123	118
RA Açores	136	132	132	137	142	146	142	150
TOTAL	**4 719**	**4 765**	**4 891**	**5 007**	**5 213**	**5 281**	**5 279**	**5 311**
Variação face ao ano anterior	***↑2%***	***↑1%***	***↑3%***	***↑2%***	***↑4%***	***↑1%***	***↓0,05%***	***↑1%***

Figura 1 - Recolha de RU (10^3 t) em Portugal Continental, entre 2014 e 2021

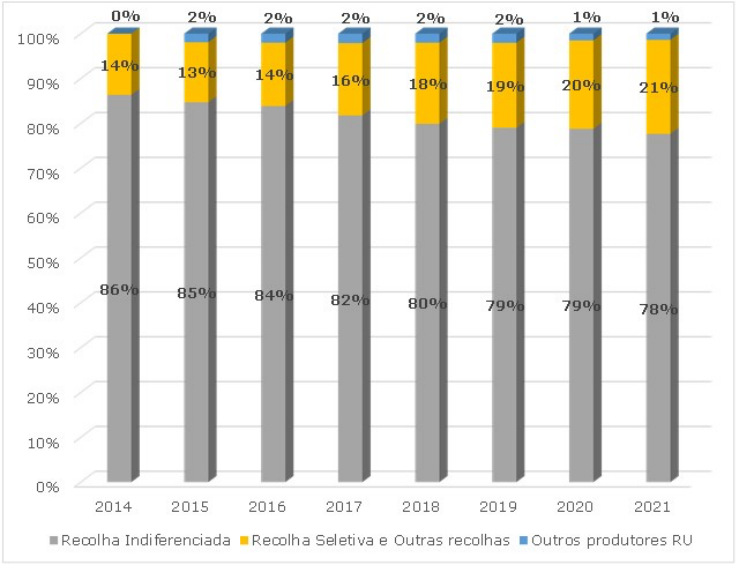


Figura 2 - Evolução da recolha de RU entre 2014 e 2021

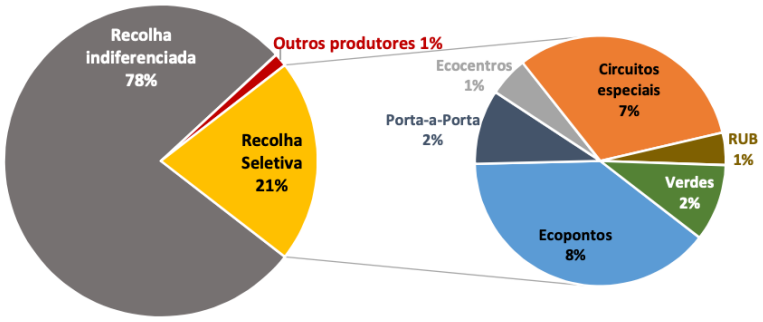


Figura 3 - Resíduos Urbanos por origem, em 2021

As Figura 2 e a Figura 3 permitem aferir uma melhoria anual muito ténue, no que respeita aos quantitativos de RU recolhidos seletivamente, o que representa um incremento de 7 pontos

percentuais de 2014 até 2021. O aumento verificado parece resultar de uma diminuição da recolha indiferenciada de RU, apesar da taxa de 21% consubstanciar uma percentagem muito inferior ao desejado. Estes resultados cruzados com os previstos na Figura 2 demonstram que, cerca de metade dos quantitativos da recolha seletiva advêm dos ecopontos e circuitos especiais, sendo o contributo do circuito porta-a-porta e a recolha de verdes pouco significativo.

Apesar das melhorias verificadas, a taxa de recolha indiferenciada mantém-se muito elevada quando comparada com a da recolha seletiva, desígnio que é crucial mudar na presente década.

O relatório acrescenta que no fluxo indiferenciado existe ainda uma quantidade muito significativa de resíduos com um enorme potencial de retoma, os quais deverão ser desviados para a recolha seletiva.

2.2.2 – **Destinos dos Resíduos Urbanos**

No que diz respeito ao encaminhamento de RU para as operações de gestão a distribuição dos mesmos é feita da seguinte forma: colocação em aterro, valorização orgânica (tratamento dos resíduos orgânicos, com vista a transformá-los em produtos úteis), tratamento mecânico (triagem e separação dos diferentes tipos de materiais presentes nos RUs distinguindo e aproveitando os materiais recicláveis) ou tratamento mecânico e biológico (uso de microorganismos para decompor a fração orgânica dos resíduos), bem como valorização energética ou material (recuperação dos materiais em forma de energia, como a inceneração, ou em materiais como combustível sólido) (Figura 4).

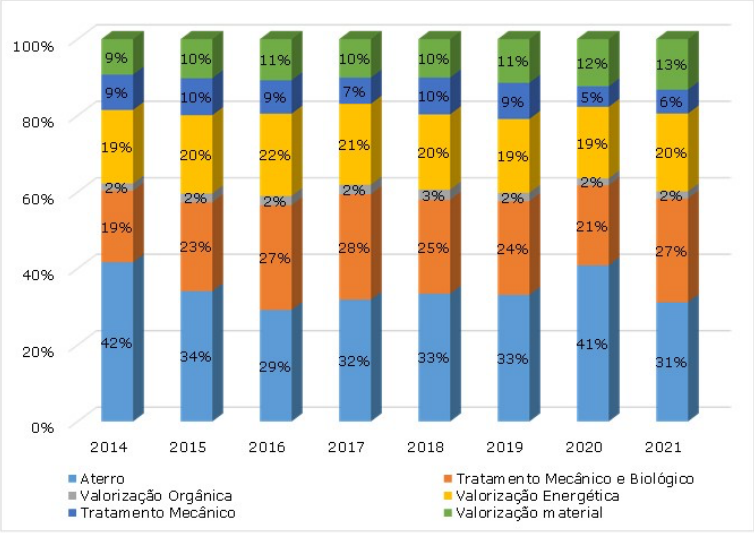


Figura 4 - Destino dos RU em 2021

Em 2021 verifica-se uma redução significativa da deposição de resíduos em aterros, contrariando a situação verificada em 2020, em parte justificada pelas Orientações e Recomendações para a gestão de resíduos em situação de pandemia de COVID-19. A acrescentar, a percentagem de resíduos direcionada diretamente para valorização energética foi de 20%, resultando assim num total de 52% de RU encaminhados para os níveis mais baixos da hierarquia de resíduos. A restante fatia de RU (48%) teve como destino o tratamento mecânico e biológico, tratamento mecânico, valorização orgânica e valorização material. Em termos percentuais o aterro continua a ser o destino mais optado (31%).

A Figura 5 mostra o destino final dos RU produzidos em Portugal no ano 2021. Embora os valores apresentados sejam uma indicação do esperado, muitas vezes estes resíduos não chegam às estações de tratamento alocadas, assim sendo os dados não apresentam uma consequência direta sobre o destino final efetivo dos mesmos.

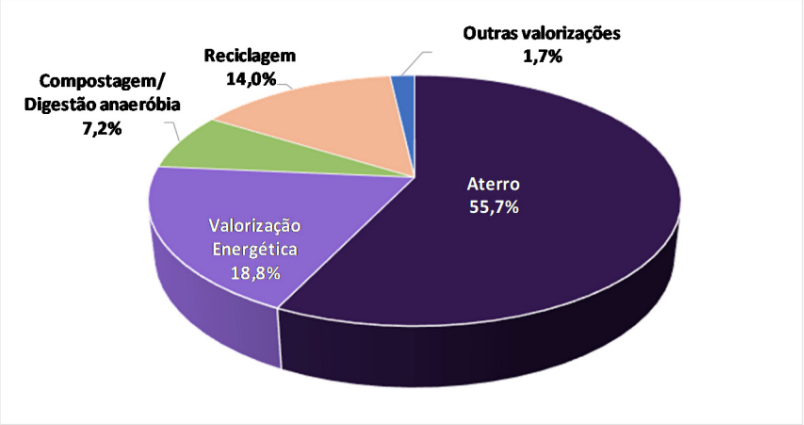


Figura 5 - Destinos finais dos RU produzidos, em Portugal Continental, em 2021

2.2.3 – **Metas dos Resíduos Urbanos**

Em termos de destino final, em 2021, 56% dos resíduos produzidos em Portugal Continental foram depositados em aterro. Porém, em termos de cálculo da meta de deposição em aterro, e conforme exposto na Figura 6, o indicador situou-se em 54%. Esta diferença verifica-se porque o peso dos resíduos produzidos durante operações de valorização dos RU, que subsequentemente são depositados em aterro, não é incluso no peso dos RU comunicados como depositados em aterro. Também em termos de destino final, 19% dos RU foram encaminhados para valorização energética, recorrendo à sua queima foi produzida energia elétrica.

Indicador		Unidade	Referência (2019)	Situação em 2021	Meta a alcançar		
					2025	2030	2035
Prevenção [Quantidade de resíduos produzidos]	Nacional	kg/hab.ano	513 kg/hab.ano	513 kg/hab.ano	-5% Face a 2019	-15% Face a 2019	-
	Portugal Continental		511 kg/hab.ano	511 kg/hab.ano			
Preparação para reutilização e reciclagem	Nacional	% de RU recicláveis	-	32%	55%	60%	65%
	Portugal Continental			33%			
Deposição em aterro	Nacional	% de RU depositados em aterro	-	53%	-	-	10%
	Portugal Continental			54%			

Figura 6 - Resumo com o ponto de situação dos indicadores em função das metas a alcançar

É possível apurar que Portugal ainda está longe de alcançar a meta de "Preparação para reutilização e reciclagem" que é 55% de materiais recicláveis.

2.3 – Caracterização Material reciclável

A Portaria n.º 851/2009, de 7 de agosto aprova o regime geral da gestão de resíduos, exigindo uma caracterização antecipada dos resíduos produzidos que possibilite identificar e quantificar aqueles que, embora suscetíveis de reciclagem, são encaminhados para aterro, incineração ou coincineração [3]

Considerado como material disponível o resíduo embalagem e resíduo não embalagem, nos seguintes temos:

Categorias	Subcategorias
Finos < 20 mm.	
Bio-resíduos (*)	Resíduos alimentares (restos de cozinha).
	Resíduos de jardim.
	Outros resíduos putrescíveis.
Papel/cartão	Resíduos de embalagens de papel/cartão.
	Jornais e revistas.
	Outros resíduos de papel/cartão.
Plástico	Resíduos de embalagens em filme de PE.
	Resíduos de embalagens rígidas em PET.
	Resíduos de embalagens rígidas em PEAD.
	Resíduos de embalagens rígidas em EPS.
	Outros resíduos de embalagens de plástico.
	Outros resíduos de plástico.
Vidro	Resíduos de embalagens de vidro.
	Outros resíduos de vidro.
Compósitos	Resíduos de embalagens de cartão para alimentos líquidos (ECAL).
	Outros resíduos de embalagens compósitas.
	Pequenos aparelhos electrodomésticos.
	Outros resíduos compósitos.
Têxteis	Resíduos de embalagens têxteis.
	Outros resíduos têxteis.
Têxteis sanitários.	
Metais	Resíduos de embalagens ferrosas.
	Resíduos de embalagens não ferrosas.
	Outros resíduos ferrosos.
	Outros resíduos metálicos.
Madeira	Resíduos de embalagens de madeira.
	Outros resíduos de madeira.
Resíduos perigosos	Produtos químicos.
	Tubos fluorescentes e lâmpadas de baixo consumo.
	Pilhas e acumuladores.
	Outros resíduos perigosos.
Outros resíduos	Outros resíduos de embalagens.
	Outros resíduos não embalagem.
Resíduos verdes (recolhidos em separado).	
Resíduos volumosos.	

Figura 7 - Grelha de Análise dos Tipos de RU Produzidos [3]

2.4 – Embalagens e Resíduos de Embalagens

São embalagens todos e quaisquer produtos feitos de materiais de qualquer natureza utilizados para conter, proteger, movimentar, manusear, entregar e apresentar mercadorias, tanto matérias-primas como produtos transformados, desde o produtor ao utilizador ou consumidor, incluindo todos os artigos "descartáveis" utilizados para os mesmos fins.

Os princípios e normas aplicáveis à gestão de embalagens e resíduos de embalagens em Portugal, encontram-se estabelecidos no Decreto-Lei n.º 152-D/2017, de 11 de dezembro, que

transpõe para ordem jurídica nacional as diretivas n.º 94/62/CE e 2004/12/CE, do Parlamento Europeu e do Conselho, relativas a embalagens e resíduos de embalagens.

A aplicação das medidas e ações recomendadas na legislação portuguesa que regula a gestão do fluxo das embalagens e resíduos de embalagens concretizou-se através do licenciamento da entidade gestora Sociedade Ponto Verde. [4]

2.5 – Sinalética nas Embalagens

Quando uma embalagem tem o símbolo ponto verde impresso significa que contribui financeiramente para a Sociedade Ponto Verde (Figura 8), uma vez que, em Portugal, a Sociedade Ponto Verde detém os direitos exclusivos de utilização deste símbolo, permitindo a sua utilização de acordo com condições previamente definidas. Os clientes da Sociedade Ponto verde podem utilizar o símbolo ponto verde nas suas embalagens.

Figura 8 - Símbolo Ponto Verde [5]

Para facilitar ao máximo a separação doméstica de resíduos de embalagens, a sociedade Ponto Verde criou um conjunto de ícones que, quando colocados na embalagem, indicam aos consumidores as regras básicas da deposição seletiva e os ajudam nessa tarefa. Para que esta informação chegue ao consumidor final, é necessário que as empresas responsáveis pela colocação de embalagens não-reutilizáveis no mercado nacional apliquem esta mesma sinalética nas suas embalagens. [5]

A Figura 9 e a Figura 10 estão representadas as sinaléticas em vigor em Portugal.

Figura 9 – Selos Iconográficos a Preto [5]

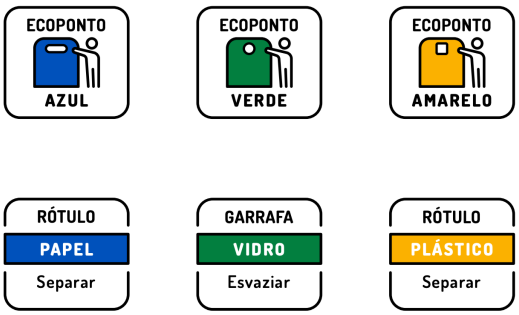


Figura 10 - Selos Iconográficos e Selos com Legenda [5]

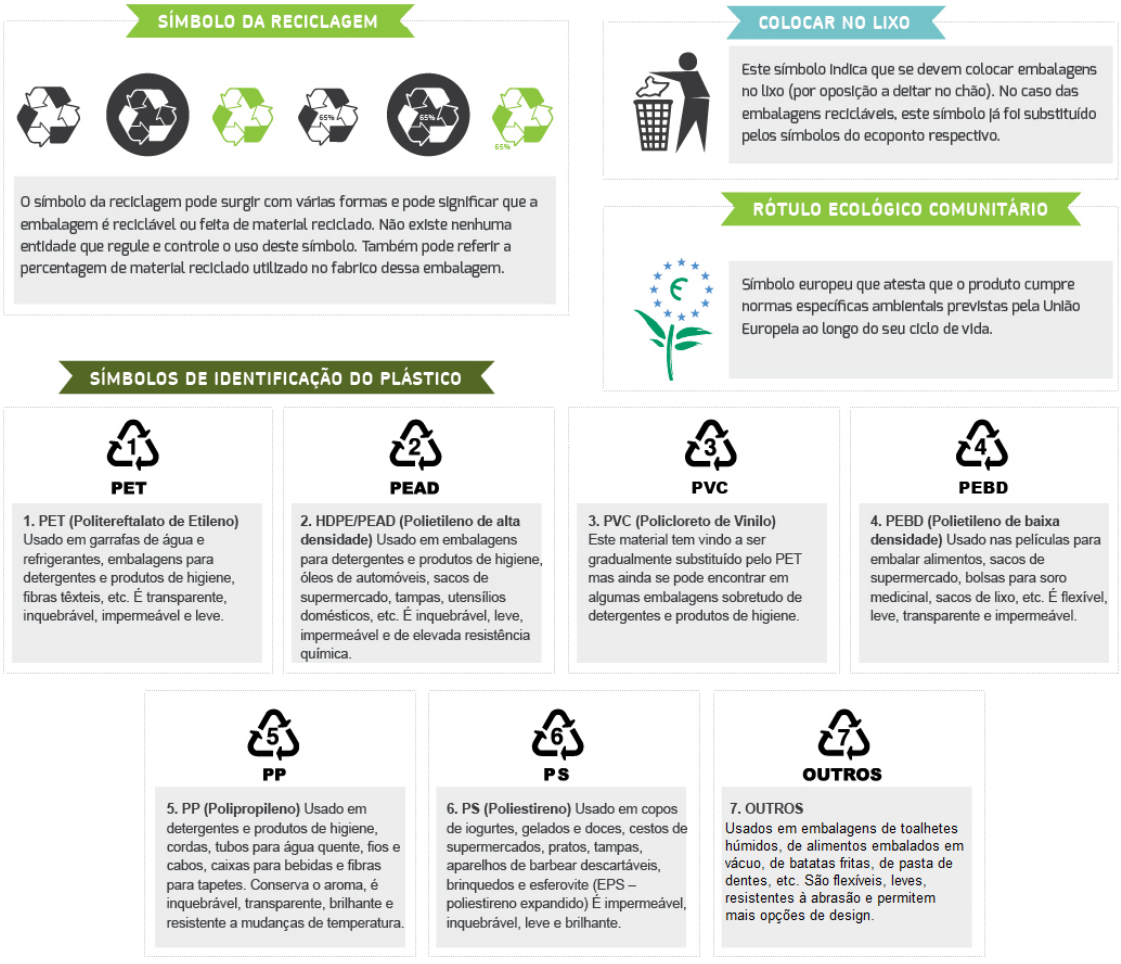


Figura 11 - Outros Símbolos e Ícones [5]

São muitos os símbolos presentes nas embalagens que, com significados e importância variável, contribuem para o esclarecimento do consumidor (Figura 11). Neles pode encontrar-se informação sobre a conceção das embalagens e dos produtos, identificar normas ambientais cumpridas pelo fabricante, encontrar uma garantia de respeito pela natureza ou ainda a indicação do local onde devemos depositar a embalagem depois de consumido o produto.

Neste capítulo foi possível perceber que ainda estamos longe de alcançar a meta de "Preparação para reutilização e reciclagem" que se estabelece nos 55% de RU recicláveis, estando apenas em 33%.

Foi possível perceber a caracterização do material reciclável e quais as sinaléticas presentes nas embalagens. Estes permitem aos utilizadores destes RU fazerem a distinção do tipo de resíduo e onde o colocar no seu fim de vida, aumentado o valor de recolha seletiva, contribuindo para o aumento da reciclagem.

Esta simbologia será utilizada no algoritmo desenvolvido neste projeto.

Capítulo 3

ESTADO DA ARTE

Neste capítulo são apresentados os temas principais que serviram de base para a elaboração da presente dissertação a fim de permitir ao leitor um melhor enquadramento com os sistemas implementados no projeto.

Inicialmente é feita uma pequena viagem pelo mundo da automação industrial, passando por alguns dos pontos mais marcantes da história da evolução industrial, de seguida são conhecidas técnicas de Processamento de Imagem e Tratamento de imagem, finalizando com Tecnologias de Reconhecimento de Imagem.

3.1 – Automação Industrial

Todas as espécies têm como objetivo facilitar a sua vida, o homem não é exceção. Desde há muito tempo que se preocupou em facilitar o seu trabalho, inventou a roda, descobriu o fogo. Com o nascimento da automação o homem criou equipamentos e técnicas que o fez produzir em maior quantidade e melhor qualidade aquilo que necessitava. O processo da automação não é confinado à realidade industrial, onde começou, hoje a automação está em todos os processos, desde a indústria, de onde surgiu, às finanças, engenharia e ciências.

A automação tem desempenhado um papel fundamental no avanço destas realidades, com impacto substancial na corrida espacial, que fez mudar o mundo, na aviação comercial que permite que o mundo funcione como hoje o conhecemos, e à guerra como muitas vezes gostávamos que não tivesse ajudado.

No Século II a.C. surgiu um dos primeiros dispositivos automáticos da humanidade, o relógio de água, um equipamento que veio melhorar as condições do ser humano facilitando a medição do tempo. Muito mais tarde, devido à não conformidade do Homem, aconteceu a denominada revolução industrial, que trouxe a máquina a vapor. Com esta, James Watt, desenvolveu o primeiro controlador automático para um processo industrial. Este controlador baseia-se na força centrífuga exercida em duas esferas. Com o aumento da velocidade do motor, maior era a forca centrifuga que deslocava as esferas para fora assim, fechava-se o mecanismo de válvulas deixando menor passagem de vapor, se a velocidade diminuísse a forca centrifuga reduzia, as bolas ficariam mais próximas do centro, abrindo a válvula, permitindo uma maior passagem do vapor. Deste modo era possível controlar a velocidade da máquina a vapor. [6]

A história da automação tem muitos colaboradores, desde Isaac Newton, que desenvolveu os fundamentos da modelagem matemática e da análise, até aos pesquisadores de hoje em dia, que conseguem enviar um míssil, controlado por satélites, a quilómetros de distância, acertando no alvo, com um erro de poucos centímetros. As guerras são os maiores patrocinadores dos desenvolvimentos de tecnologias em todas as áreas, e a automação não fica de fora. [7]

Com a eletrificação da indústria foi-se tornando imprescindível o desenvolvimento tecnológico, visto que os processos de fabrico, a complexidade e a velocidade de operação dos equipamentos

continuam a aumentar. Naturalmente começou-se a sonhar com a possibilidade de as máquinas trabalharem de forma independente do ser humano. Em 1830 dá-se a invenção do relé eletromagnético, dando um grande passo na autonomia das máquinas. Anos depois uma nova evolução estaria a acontecer graças ao transístor, em 1968 a Modicon apresenta o primeiro PLC (*Programmable Logical Controller)* e introduz a lógica programada, que veio gradualmente substituir a lógica cablada até então utilizada. O seu tamanho muito inferior aos longos painéis de lógica cablada existentes até à altura, aliado à possibilidade de se poder alterar a sua programação com recurso a diferentes métodos de controlo, permitiu que a indústria evoluísse a uma velocidade superior com uma melhor e mais fácil interligação entre os diferentes equipamentos existentes. Com a larga difusão do controlo moderno existiu a necessidade de outros tipos de algoritmo de controlo como o PID.

Redes de campo industriais (fieldbus) para comunicação entre PLCs foram o passo seguinte, criando uma rede de comunicação industrial. Com características diferentes das transmissões já existentes, estes tinham como objetivo a possibilidade de efetuar o controlo de sistemas de produção garantindo sempre uma elevada qualidade de serviço (QoS). Redes tais como Modbus (1979), CAN Bus e HART (1986), e Profibus (1989) foram surgindo [8] possibilitando à indústria ter linhas de produção completas a trabalhar devidamente organizadas, maximizando a sua produtividade. Um passo importante para o avanço no desenvolvimento não só́ das redes industriais, mas de todos os tipos de comunicação foi a criação pela ISO (International Organization for Standardization) em 1970 do Modelo OSI (Open System Interconnection) [9]. Ainda servindo de referência nos dias de hoje, é uma hierarquia que define por camadas a organização e o funcionamento dos componentes existentes em redes de comunicação por forma a manter a interoperacionalidade entre diferentes sistemas.

A evolução tecnológica prosseguiu a todo o vapor, de forma exponencial, havendo evoluções tanto em hardware como software. Estas evoluções foram de tal forma significantes que na realidade presente o controlo é aplicado a tudo e é maioritariamente informático. Atualmente é impossível fazer-se parte de uma sociedade moderna sem ter qualquer contacto com aparelhos que estabelecem algum tipo de comunicação. Tendo-se passado o mesmo no meio industrial. Com o aparecimento de sistemas mais robustos, começou-se a explorar novos conceitos de produção com vista a melhorar tanto o produto final como o próprio processo de fabrico. E aqui entra a Indústria 4.0.

Em 2011 surge este novo conceito, em que a comunicação era o pilar, tanto homem-máquina como máquina-máquina. Soluções como o Modelo OSI, e os sistemas OPC, foram grandes impulsionadores de uma indústria mais flexível, organizada e com menos barreiras. Passou-se a ter linhas de produção com máquinas com maiores capacidades de interligação entre si, possibilitando que os processos se tornassem mais unificados, mais fluidos, e com menos tempos de paragem. Mais tarde. protocolos de comunicação como o ModBus TCP/IP e o Profinet começaram a utilizar a rede de Ethernet, tendo possibilidade de estar conectados a internet.

Outros dois grandes passos tecnológicos foram o surgimento da Internet Of Things (IoT) e da inteligência artificial (IA).

3.2 – Processamento e Tratamento de Imagem

O ser humano naturalmente consegue distinguir cores, formas, coisas, sem ter que utilizar os seus 5 sentidos, mas apenas um, a visão. O facto de um sistema de automação receber dados de múltiplos sensores, podendo estes ser de vários tipos, nada significa se o equipamento não estiver preparado para processar e analisar a informação. O olho humano, o sensor, de nada serve de forma isolada, contudo, ligado a uma vasta rede de neurotransmissores, permite que o cérebro possa cruzar a informação processando-a, permitindo adquirir múltiplos pontos de referência de um dado objeto.

É frequente fazerem-se comparações diretas entre o olho humano (Figura 12) e uma câmara de captura de imagem, e de facto as suas estruturas assentam em princípios de funcionamento semelhantes. A luz ao chegar ao olho atravessa em primeiro lugar a córnea que se comporta como uma lente fixa, orientando a luz para o Cristalino (em inglês *lens* = lente) que tem a função de focagem da imagem (lente móvel). A pupila que se localiza sobre o cristalino, é adaptada pela íris de acordo com a intensidade de luz recebida, trabalhando como a abertura de uma "lente". Por fim surge a retina que funciona como elemento sensor, dividida em dois tipos de células: os "bastonetes" que funcionam como recetores de luz monocromática, e os "cones" que são responsáveis pela identificação das cores.

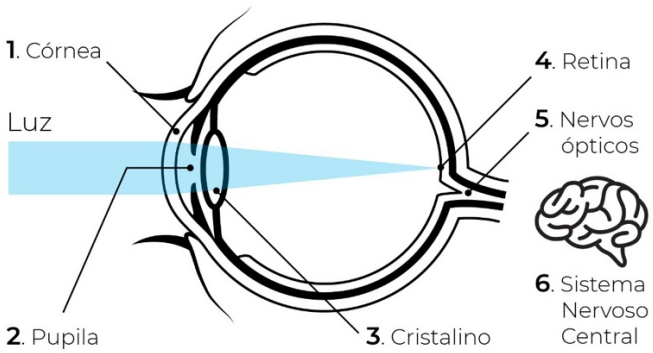


Figura 12 - Olho Humano [10]

A digitalização das câmaras, a partir de 1980, criou diversas áreas de desenvolvimento. Imagens que até então eram apenas impressas/reveladas, passaram a ser formatos digitais, abrindo lugar a um novo tipo de sensor. Foi necessário a criação de técnicas de processamento de imagem para que a câmara começasse a surgir nos processos de automação e controlo. São várias as áreas em que foi possível passar a utilizar este equipamento, tais como:

- Indústria:
  - Controlo de qualidade
  - Identificação de códigos
  - Deteção de objetos
  - Guiamento de robôs
- Automobilismo:
  - Controlo de trajetórias (auto-piloto)
  - Previsão de colisões
  - Transporte autónomo
- Segurança:
  - Deteção de pessoas
  - Controlo de acessos
  - Previsão de movimentos
  - Análise facial
  - Deteção de Incêndios (chamas e fumo)
- Agricultura:
  - Deteção de pragas
  - Deteção de alimentos estragados
  - Apoio na programação de cultivos

- Medicina:
    - Identificação de células
    - Análise de tumores
    - Modelação 3D de órgãos

A capacidade de processamento de informação da visão computacional, quando comparada a sinais digitais ou analógicos é muito elevada, sendo assim, a sua evolução tem estado diretamente relacionada com o avanço tecnológico, tanto a nível eletrónico como de computação. Ainda assim, embora grande parte da investigação sobre o tema, resulte em algoritmos complexos, muitas vezes com uma componente relativamente grande de matemática, difícil de processar, têm-se conseguido desenvolver métodos que simplifiquem o esforço dos controladores em aplicar estes mesmos algoritmos. As abordagens de tratamento matricial das imagens, trabalhando-as como agrupamentos lógicos de dados, foi um passo importante neste campo, visto facilitar a relação com a organização das memórias computacionais convencionais. O processamento de imagem envolve o tratamento de grandes quantidades de dados a alta velocidade (aplicação de inúmeros processos por cada *frame* de vídeo), e sendo esta uma necessidade comum à inteligência artificial, a evolução de algoritmos mais complexos tem passado por tirar o melhor partido de cada uma das duas áreas, de modo a permitir abordagens mais preditivas, e rápidas no momento de trabalhar e analisar os dados.

No mercado existem diversas soluções comerciais tanto para captura como para processamento de imagem. As empresas que se dedicam à produção de elementos sensores, focam os seus esforços no desenvolvimento de elementos cada vez mais compactos, mas em simultâneo com melhores resultados, procurando ir de encontro às necessidades de cada público. Uma solução industrial por norma não tem os mesmos requisitos que um fotógrafo profissional. Enquanto numa máquina com um ambiente relativamente controlado, são valorizados a capacidade de atingir grandes detalhes geométricos a grande velocidade, uma fotografia num ambiente aberto é favorecida por uma melhor composição das cores e maior variação de contrastes, e embora o tempo de captura seja sempre importante, comparativamente ao primeiro caso, o tempo de pós-processamento pode até ser alongado em favor de um melhor resultado. As soluções mais típicas na indústria passam por câmaras com funções integradas, controladores dedicados e software de processamento e análise.

Na implementação da visão computacional, para além do dispositivo de aquisição de imagem propriamente dito (câmara) outros elementos são também essenciais, nomeadamente: o sistema de iluminação, a interface com a unidade de processamento digital e o *software* de processamento de imagem. Dependendo da aplicação em questão, o sistema de aquisição de imagem pode ter especificações exigentes, o mesmo sucede com o processamento computacional, ao qual se pode exigir uma grande capacidade de memória e elevada velocidade de processamento.

Estes requisitos retardaram muito a utilização da visão na robótica há algumas décadas atrás como uma alternativa tecnicamente possível e economicamente viável. Atualmente, com o grande desenvolvimento tecnológico na área dos microcomputadores, em que oferecem capacidades computacionais cada vez mais elevadas a preços cada vez mais baixos, e a banalização da câmara de vídeo, transformada num produto de eletrónica de consumo para entretenimento com qualidade aceitável, tornaram a visão computacional um recurso perfeitamente acessível à robótica industrial e com custos baixos a moderados [10].

Os procedimentos para aplicação de técnicas de visão computacional encontram-se normalmente divididos em seis etapas sequenciais (ver Figura 13) [11]:

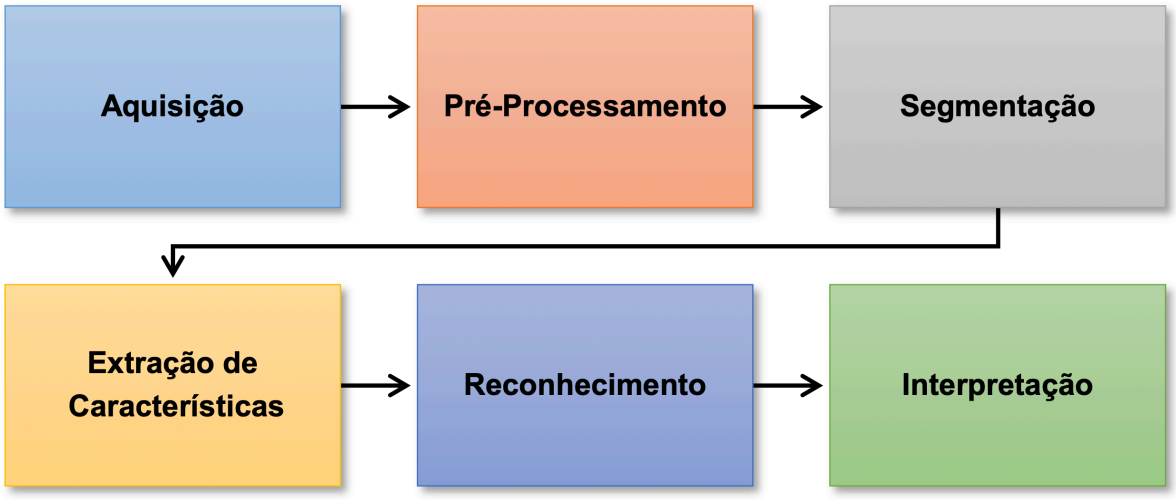


Figura 13 - Etapas de um processo de visão computacional.

Analisando de um modo geral cada uma das etapas dos procedimentos de visão computacional, temos [11]:

- **Aquisição:** aquisição da imagem captada com recurso ao dispositivo de visão (normalmente câmara digital). São utilizadas técnicas especiais de iluminação para

obter imagem com contraste suficiente para processamento posterior, pois na imagem adquirida existe uma considerável quantidade de informações que devem ser realçadas, entre as quais: dimensão e representação espacial/geométrica, cor e textura;

- **Pré-Processamento:** compensa deficiências especificas, geradas no momento da aquisição da imagem. Destaca os detalhes da imagem que são de interesse para análise ou que tenham sofrido alguma deterioração;
- **Segmentação:** tem como objetivo isolar regiões de pontos da imagem pertencentes a objetos para posterior extração de atributos e cálculo de parâmetros descritivos;
- **Extração de caraterísticas:** a partir de imagens já segmentadas, esta etapa procura obter dados específicos ou padrões relevantes das regiões ou objetos destacados. Os tipos de dados, padrões ou características mais comuns são o número total de objetos, suas dimensões e geometria;
- **Reconhecimento:** o objetivo desta etapa é reconhecer objetos na imagem, agrupando parâmetros de acordo com a sua semelhança para cada região de pixéis encontrada;
- **Interpretação:** O objetivo fulcral de um sistema de visão computacional culmina nesta etapa, em que o sistema consegue interpretar a imagem que captou.

É importante agrupar em categorias estas etapas que constituem um sistema de visão computacional, de acordo com a sua sofisticação e implementação. Considera-se então três níveis de processamento: baixo, médio e alto nível [11].

- Associa-se a processamento de **baixo nível** todos os processos que apenas utilizam as etapas de aquisição e pré-processamento (técnicas de limpeza e aprimoramento de imagem);
- O processamento de **nível médio** diz respeito a todos os processos que extraem, caracterizam e identificam componentes numa imagem resultante do processamento de baixo nível. Este nível de processamento utiliza as etapas de segmentação (técnica de agrupar áreas de uma imagem que têm características semelhantes em entidades distintas representando parte da imagem), extração de características (técnica que lida com as características adequadas para distinguir diferentes objetos, como por exemplo, tamanho e forma) e reconhecimento (técnica que identifica objetos, como por exemplo, chave e parafuso);

- Por fim, um processamento de **alto nível** refere-se a um processo relacionado com as tarefas de cognição associadas à visão humana, onde se aplicam técnicas de controlo baseadas em algoritmos de inteligência artificial. Enquanto os algoritmos desenvolvidos em baixo e médio abrangem um espetro razoavelmente bem definidos de atividades, o processamento de alto nível é consideravelmente mais vago e especulativo.

A realização desta sequência de etapas é alcançada através da complexidade multidisciplinar que a visão computacional normalmente apresenta (ver Figura 14).

Figura 14 - Multidisciplinaridade em visão computacional

A visão computacional abrange diversas áreas cientifico-tecnológicas que atravessam a Mecânica, a Eletrotécnica e a Informática.

Desta forma, o sistema consegue tomar decisões a partir da extração de informações do mundo real através de imagem. A decisão pode ser feita a partir de verificações simples a respeito de parâmetros extraídos dos objetos ou de algoritmos mais complexos de inteligência artificial. Contudo, para que seja possível "educar" um sistema a interpretar imagens, é necessário que previamente haja um estudo acerca da imagem em si, desde a sua captação até à sua interpretação.

3.2.1 – Aquisição de Imagem

A radiação eletromagnética está presente na nossa vida e é a que nos permite ver. A luz visível, ou seja, identificável pelo olho humano, por definição, trata-se da radiação eletromagnética que se encontra no respetivo espectro, na gama de largura de bandas entre 400nm e 700nm [12], indo desde o violeta até ao vermelho respetivamente (Figura 15). É frequente fazer-se referência também à "luz ultravioleta" e à "luz infravermelha". Embora a retina não receba a radiação que se encontra na gama ultravioleta (10nm a 400nm) visto ser filtrada pela córnea, é conhecido que entre os 310nm e os 400nm a radiação reage com diversos materiais gerando luz visível (violeta). Já na gama infravermelha, apenas é visível a luz vermelha no limiar dos 700nm, largura de banda a partir da qual são utilizados por exemplo, equipamentos para deteção de radiação térmica.

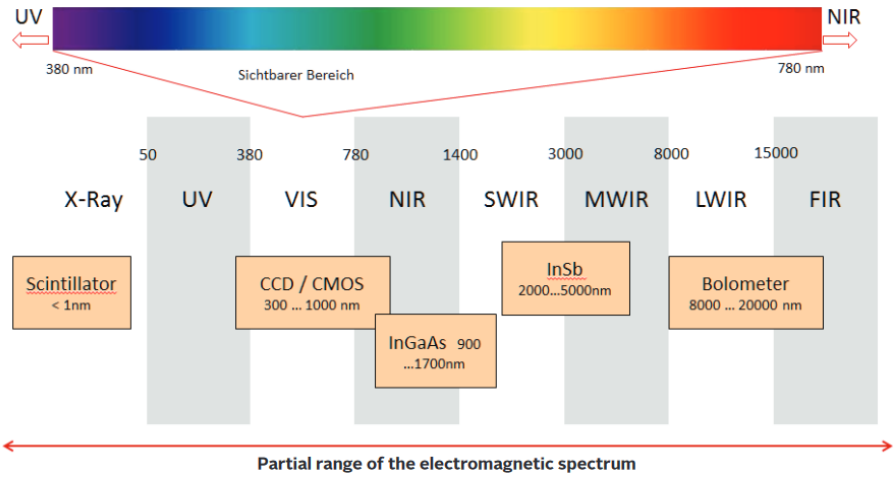


Figura 15 - Diversas tecnologias de sensores para captação do espectro luminoso [14]

Enquanto o olho humano tem as limitações referidas, numa câmara digital é possível ir além do espectro de luz visível. Tal deve-se aos elementos sensores capturarem apenas a energia emitida pelas fontes de luz sem qualquer diferenciação (Figura 15).

Normalmente o elemento sensor de uma câmara é formado por uma matriz de condensadores MOS (*Metal-Oxide-Semiconductor*), que armazenam carga ao serem projetados pela radiação luminosa. No entanto toda a matriz é uniforme, sendo necessário isolar cada uma das cores para obter imagens dentro do espectro acima referido. Esta operação pode ser feita tanto através de

matrizes de filtros de cores visíveis ou não visíveis, colocadas sobre a matriz de condensadores (CFA - *Color Filter Array*), como através de lentes prismáticas que separem naturalmente a luz e a encaminham para elementos sensores separados (Figura 16**).**

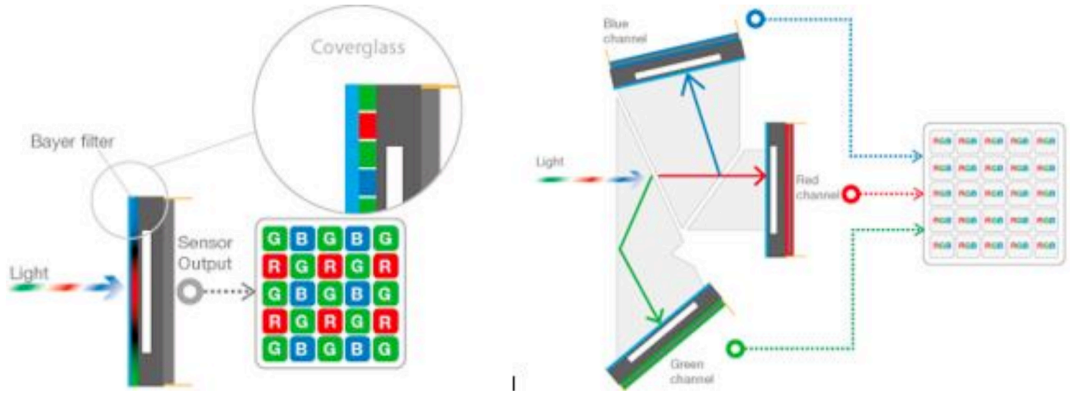


Figura 16 - Métodos de separação de cores: Filtro matricial (esquerda), Lente prismática (direita) [14]

A forma como essa carga armazenada é tratada depende da tecnologia em uso, sendo as duas principais a CCD e a CMOS (ver Figura 16**)**:

- **CCD (*Charge-Coupled Device*):** Constituído por um grupo de condensadores ligados entre si e organizados em linhas verticais. Ao se aplicar uma tensão entre os mesmos, é possível transmitir a carga acumulada ao longo dessas linhas. Apos a captura, esta ação é realizada repetidamente até se transmitir toda a carga existente na matriz a uma zona responsável pela conversão A/D (Analógico-Digital) dos sinais recebidos, transformando-os depois em dados digitais. A grande vantagem dos sensores CCD está na qualidade de imagem devido ao facto de a carga ser capturada de forma uniforme ao longo de toda a matriz, não existindo variações na captura de carga entre cada célula.
- **CMOS (*Complementary Metal-Oxide-Semiconductor*):** Um sensor CMOS, ao contrário de um sensor tipo CCD tem a matriz convertida localmente, acelerando o processo de captura e reduzindo o custo energético do sensor. É também possível realizar algumas correções em pré-processamento recorrendo à eletrónica para limitar a quantidade de carga tratada em cada célula como no caso do processamento HDR (*High Dynamic Range*) [13]. Este processo tem o inconveniente de poder provocar disparidades entre a carga capturada e o sinal de saída do conversor A/D devido a diferenças existentes entre os componentes ao longo da matriz. Apresenta também outro

inconveniente, visto que o sensor tem de ser ocupado com elementos conversores a matriz de captura consequentemente apresenta uma menor área.

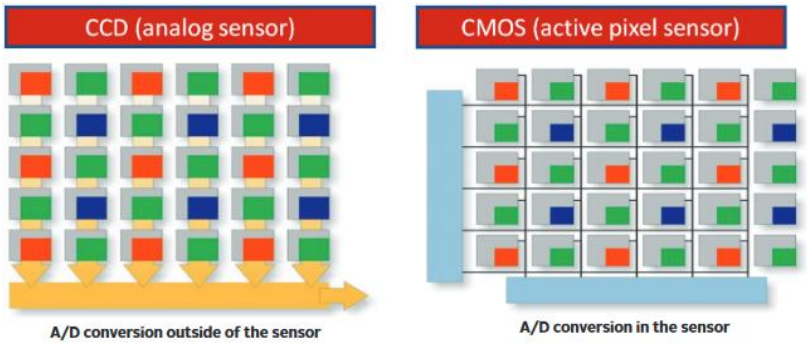


Figura 17 - Comparação entre as tecnologias de sensores CCD e CMOS [14]

3.2.2 – Processamento digital de sinal

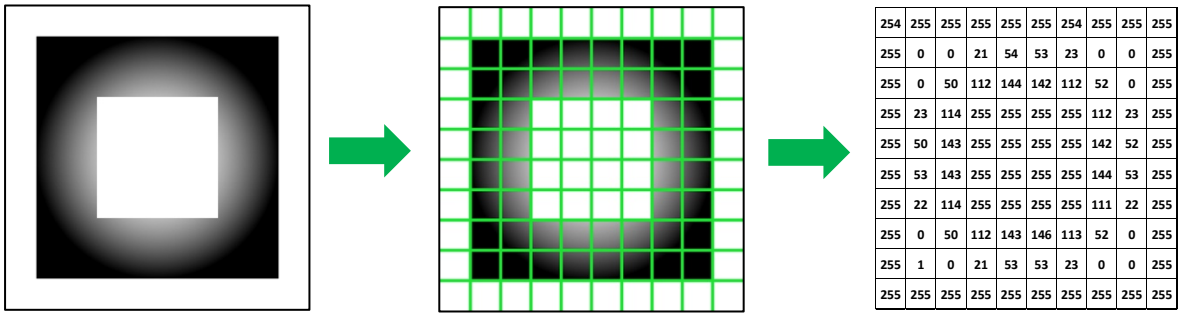
3.2.2.1 – Digitalização

Através do processo de digitalização, uma imagem "real", capturada por um sensor (exemplo: câmara), é discretizada, ou seja, digitalizada. A imagem digital obtida é resultado da aplicação de uma função que transforma a imagem numa ou várias matrizes, dentro do respetivo espaço de cor, ou seja, cada ponto da imagem (pixel) será o resultado de uma função de duas variáveis, que representam as coordenadas do ponto codificado:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & . & f(0,n) \\ f(1,0) & f(1,1) & . & . \\ . & . & . & f(m-1,n) \\ f(m,0) & . & f(m,n-1) & f(m,n) \end{bmatrix} \quad \text{(eq. 1)}$$

Onde m x n são a dimensão da imagem em pixéis.

Pode considerar-se uma imagem digital monocromática, com 256 tons de cinzento, como uma matriz de n linhas por m colunas, em que cada pixel assume um valor entre 0 e 255 representando um tom de cinzento. O branco puro terá um valor de 255 e o preto puro terá um valor de 0. Para esta resolução, cada pixel será representado por um byte (8 bits) - ver Figura 18.



254	255	255	255	255	255	254	255	255	255
255	0	0	21	54	53	23	0	0	255
255	0	50	112	144	142	112	52	0	255
255	23	114	255	255	255	255	112	23	255
255	50	143	255	255	255	255	142	52	255
255	53	143	255	255	255	255	144	53	255
255	22	114	255	255	255	255	111	22	255
255	0	50	112	143	146	113	52	0	255
255	1	0	21	53	53	23	0	0	255
255	255	255	255	255	255	255	255	255	255

Figura 18 - Digitalização de uma imagem.

3.2.2.2 – Segmentação

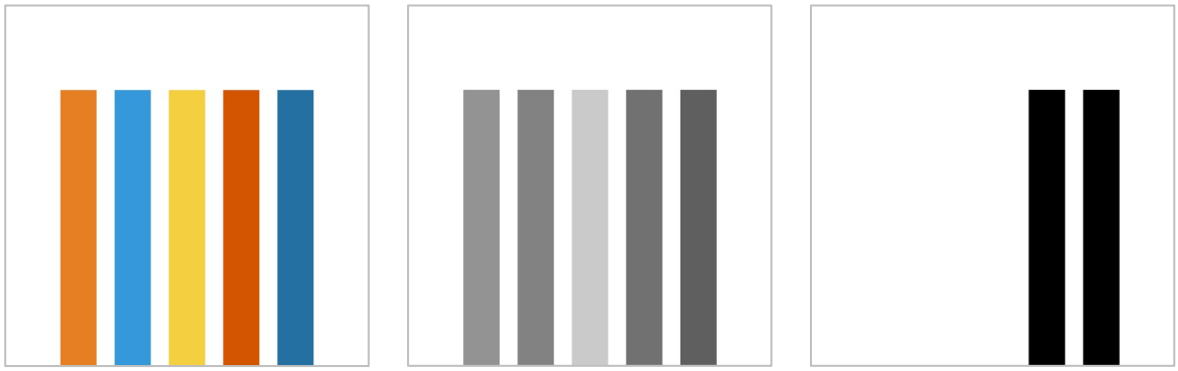
A segmentação de imagens é a técnica que subdivide uma imagem em partes, ou objetos, que a constituem. Quando o olho humano capta uma imagem e a analisa, o cérebro humano divide a cena em elementos, que poderão ser um único objeto ou um elemento mais complexo visualmente (por exemplo uma árvore num bosque).

Se os pontos das áreas em que foi subdividida a imagem partilharem uma ou mais características comuns (por exemplo a cor), as utilizações de algoritmos de reconhecimento de padrões poderão ser utilizadas para as funções de análise, em processamento de imagem.

Na análise por segmentação, o número de objetos que resultam da mesma, depende do nível de detalhe ou da abstração do processo. Por exemplo, na análise de uma imagem com blocos habitacionais, num primeiro nível de abstração poderemos ter só as casas e as ruas, num nível mais profundo, poderemos identificar as portas e as janelas de cada casa. Logo, o algoritmo de segmentação não resulta numa só solução, dependendo sim do objetivo do processamento da imagem.

3.2.2.3 – Imagens Binárias

As imagens binarias caracterizam-se por serem constituídas por pontos pretos e brancos puros. Nestas imagens não há tons de cinzento intermédios. Esta é uma das formas utilizadas por muitos programas de computador para armazenarem grandes imagens, em ficheiros de reduzida dimensão, uma vez que para guardar a informação relativa a cada pixel basta 1 bit em vez dos 8 bits necessários para os 256 tons de cinzento (ver Figura 19).



a) b) c)

Figura 19 – a) Imagem a cores, b) imagens a 256 tons de cinzento e c) imagem binarizada com o treshold a 111 (a preto e branco)

Na Figura 19 é possível ver em a) a imagem original com as suas diversas cores, em b) a imagem em tons de cinza, sendo que cada cor equivale a um valor de 256 tons de cinzento, e na imagem binarizada, c), de 4 ricas restam apenas 2.

3.2.2.4 – Espaços de Cor

A forma como a cor de cada ponto (pixel) da imagem é codificado remete-nos para os espaços de cor. Adicionalmente, no processamento pode ser necessário "condicionar" as cores originais da imagem tendo em conta o tipo de análise pretendida. O "condicionamento" das cores pode ser realizado por transferência entre espaços de cor, preservando os detalhes e a aparência natural, ou não, da imagem original. A consideração dos espaços de cor é também de extrema importância na aquisição das imagens, no seu armazenamento e ainda na capacidade de computação necessária para a realização dos algoritmos no processamento digital. A escolha do espaço de cor é muito importante tendo em conta os vários processos que se vão realizar no processamento de imagem, desde a aquisição, armazenamento, processamento e análise.

Os vários métodos de modelação reagem de formas diferentes com a mudança de espaço de cor. Seguidamente apresentam-se os alguns tipos de espaços de cor mais utilizados: RGB e HSV. Relativamente a canais de televisão existem outros espaços de cor que são mais apropriados.

RGB

Neste espaço de cor as cores são especificadas em termos das três cores primárias: vermelha (R-Red), verde (G-Green) e azul (B-Blue). É um modelo aditivo, porque todas as cores são criadas a partir das combinações das componentes primárias de cor (Figura 20 - Espaço de cor RGB). Se cada componente primário for codificado em 8 bits, teremos 24 bits para as três componentes primárias e com essa dimensão é possível representar até cerca de 16,7 milhões de cores (224 = 16777216). Por exemplo, para a cor vermelha temos o termo ordenado (255, 0, 0), para o verde (0,255, 0), e assim, combinado o R e G, em quantidades iguais, poderemos criar o amarelo, definido por (255, 255, 0). A quantificação de cada componente de cor primária dá a sua intensidade. Se os 3 componentes tiverem a intensidade máxima (255, 255, 255), teremos a cor branca. O RGB é um dos espaços de cor de maior utilização, sendo utilizado nas placas de vídeo dos computadores e na tecnologia de monitores. A sua utilização na deteção de pele humana é muito condicionada por alterações nas condições de luz (iluminação).

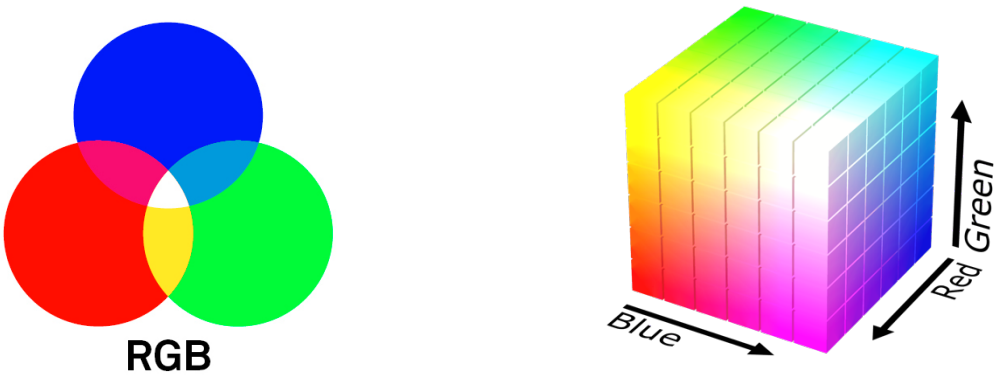


Figura 20 - Espaço de cor RGB

RGB Normalizado

A fim de se reduzir a dependência do espaço RGB em relação às condições de iluminação, os valores RGB podem ser normalizados, por um simples procedimento de normalização conforme seguidamente se apresenta, dando origem ao RGB normalizado.

$$r = \frac{R}{R + G + B} \qquad \text{(eq. 2)}$$

$$g = \frac{G}{R + G + B} \qquad \text{(eq. 3)}$$

$$b = \frac{B}{R + G + B} \quad \text{(eq. 4)}$$

A soma dos componentes normalizados do RGB é igual à unidade (r + g + b = 1), assim sendo, r e g são componentes suficientes para representar uma cor neste espaço de cor, pois a terceira componente, b, pode ser obtida diretamente dos valores de "r" e "g". O RGB e a sua versão normalizada estão entre os espaços de cor mais popularmente utilizados na deteção de pele humana.

HSV

O espaço de cor Hue-Saturation-Value foi introduzido pela necessidade de especificar as propriedades de cor de forma quantitativa. As componentes descrevem as cores com valores intuitivos, baseados na ideia "artística" da coloração, saturação e tom. O Hue (matiz) define a cor dominante de uma área (vermelho, verde, roxo e amarelo), a Saturation (saturação) mede o colorido de uma área, em relação ao contraste, e por fim o Value (valor) é a componente que quantifica a luminância da cor (Figura 21 - Espaço de cor HSV). O seguinte conjunto de equações é usado para transformar o espaço RGB no espaço HSV:

$$H = arccos \frac{\frac{1}{2}((R - G) + (R - B))}{\sqrt{((R - G)^2 + (R - B)(G - B))}} \quad \text{(eq. 5)}$$

$$S = 1 - 3\frac{min(R, G, B)}{R + G + B} \quad \text{(eq. 6)}$$

$$V(L) = \frac{1}{3}(R + G + B) \quad \text{(eq. 7)}$$

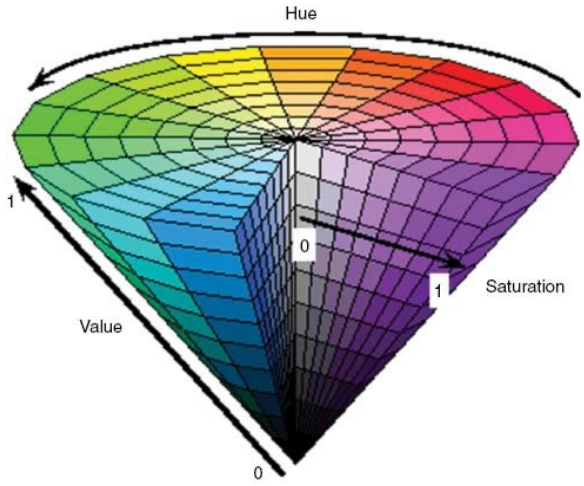


Figura 21 - Espaço de cor HSV [16]

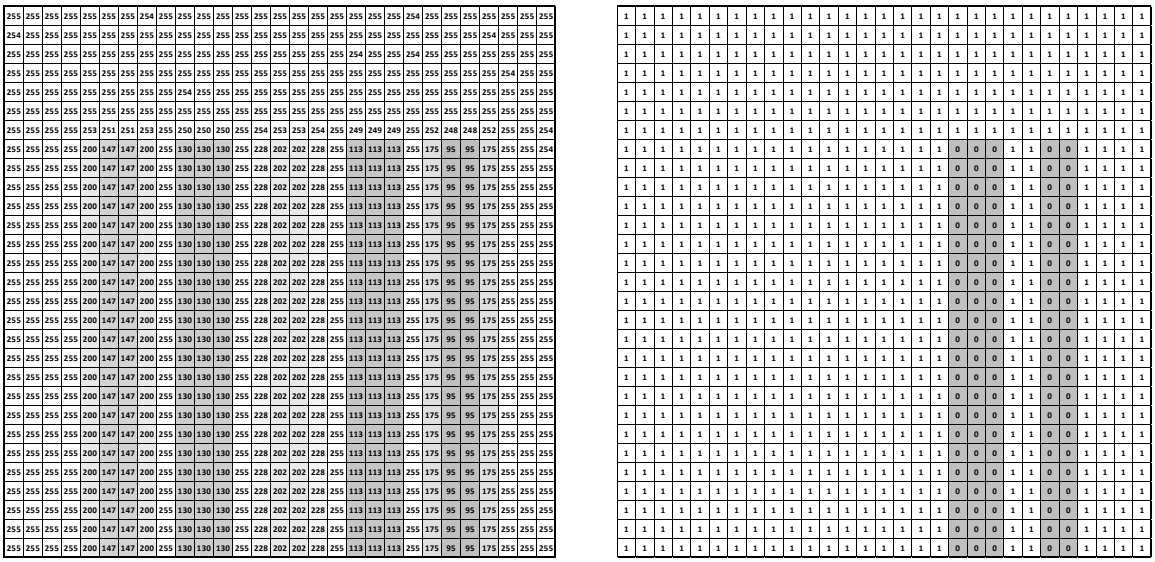
3.2.2.5 – Binarização (segmentação monocromática)

É frequente as imagens a processar estarem sobre um fundo de uma qualquer cor, o que dificulta o tratamento da imagem. A binarização é uma transformação de grande utilidade sempre que pretendemos localizar formas e contornos na imagem, uma vez que permite separar um objeto do fundo da imagem.

A binarização consiste em definir um valor como nível de separação (*threshold, T*), onde todos os pixéis, com valor inferior a este, assumem o valor 0 (preto), e todos os pixéis, com valor acima deste, assumem o 1 (branco).

$$g(x,y) = \begin{cases} 1 \Leftarrow f(x,y) > T \\ 0 \Leftarrow f(x,y) \leq T \end{cases} \qquad \text{(eq. 8)}$$

Definindo o nível de cinzento de separação (*threshold*) para um valor abaixo do fundo, todo o fundo ficará branco. Utilizando como exemplo a Figura 19, para um *threshold* de 111, a imagem b), com 256 tons de cinzento, apresenta uma matriz Figura 22 a), após a sua binarização, ficará conforme é apresentado na Figura 22 b). Ou seja, todos os valores inferiores a 111 ficaram representados a preto e os restantes a branco.



a) b)

Figura 22 - a) Matriz Original; b) Matriz depois da Binarização

Para efeito de aplicação prática observa-se o exemplo mostrados na Figura 23:

a) b)

Figura 23 - Binarização – a) Imagem de origem a 256 tons de cinzento e b) Imagem binarizada com treshold de 110

Como se pode observar o elemento que se pretende identificar, o selo iconográfico, ficou mais legível e a imagem ocupa menos espaço em memória.

3.2.2.6 – Definição de pixel vizinho

Para a quantificação de certas áreas das regiões da imagem, assim como para a aplicação de alguns algoritmos de processamento de imagem, torna-se necessário definir o conceito de pixel vizinho.

- Diz-se que 2 pixéis são 4-vizinhos se ao partilharem um limite, partilham também uma face (Figura 24 a)).
- Diz-se que 2 pixéis são 8-vizinhos se ao partilharem um limite, partilham também pelo menos um canto (Figura 24 b)).

Pixeis 4 - Vizinhos a)

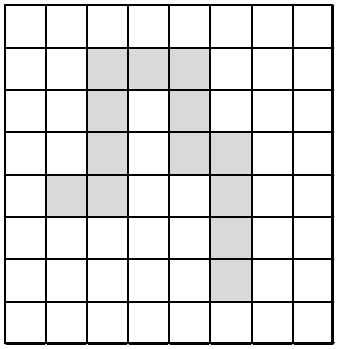
Pixeis 8 - Vizinhos b)

Figura 24 - Pixel vizinho

3.2.2.7 – Definição de Caminhos

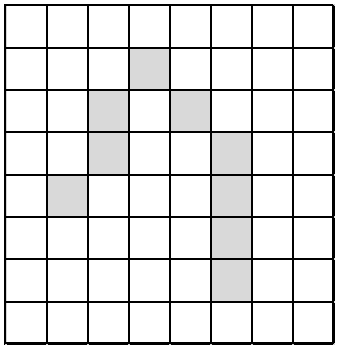
Para a quantificação de arestas e perímetros de certas regiões da imagem, bem como para a aplicação de alguns algoritmos de processamento de imagem, torna-se necessário definir o conceito de caminho.

- Diz-se que um caminho é um 4-Caminho se as ligações entre os pixéis que formam o caminho forem pixéis 4-Vizinhos (Figura 25 a)).
- Diz-se que um caminho é um 8-Caminho se houver ligações entre os pixéis que formam o caminho forem pixéis 8-Vizinhos., (Figura 25 b)).



4 - Caminho

a)



8 - Caminho

b)

Figura 25 - Caminhos: 4-Caminho e 8-Caminhos

3.2.2.8 – Definição de Fronteira

Para a definição de certas regiões na imagem, bem como para a aplicação de alguns algoritmos de processamento de imagem (por ex. morfológicos), que serão referidos mais à frente neste trabalho, torna-se necessário definir o conceito de fronteira. Define-se como fronteira de um objeto, numa imagem, o conjunto de pontos que estão no limite do objeto e que têm 4-Vizinhos ou 8-Vizinhos (Figura 26 b)).

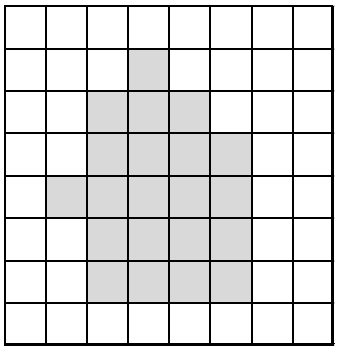


Imagem Original

a)

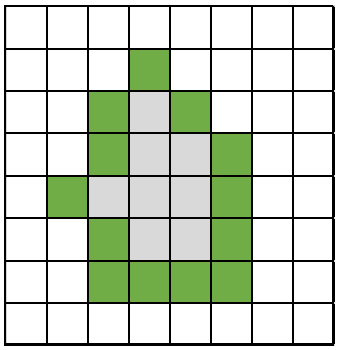


Imagem com Pixeis de fronteira

b)

Figura 26 - Pixéis Fronteira – a) Imagem Original, b) Imagem com Pixeis de fronteira a verde

O tamanho de uma área da imagem, representado em notação matricial, será uma matriz com a dimensão nxm, representada por:

$$A = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathrm{I[i,j]} \qquad \text{(eq. 9)}$$

3.2.2.9 – Definição de Distância

Para a quantificação da distância entre dois pontos de uma imagem torna-se necessário definir 3 tipos de distâncias:

- Distância Euclidiana

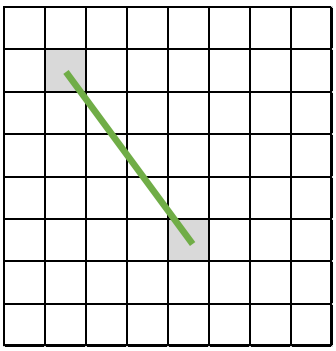


Figura 27 - Distância Euclidiana

$$d_{Euclidiana} = ([i_1, j_1], [i_2, j_2]) = \sqrt{(i_2 - i_1)^2 + (j_2 - j_1)^2} \qquad \text{(eq. 10)}$$

- Distância quarteirão (city block)

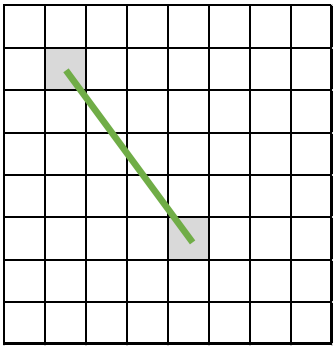


Figura 28 - Distância quarteirão (city block)

$$d_{quarteirão} = ([i_1, j_1], [i_2, j_2]) = |i_2 - i_1| + |j_2 - j_1| \quad \text{(eq. 11)}$$

- Distância tabuleiro de xadres (Chestboard)

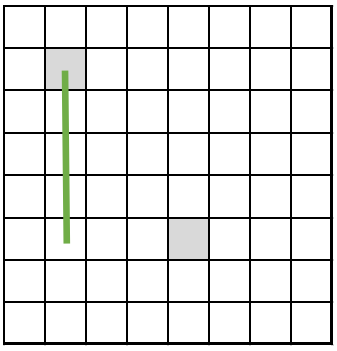


Figura 29 - Distância tabuleiro de xadres (Chestboard)

$$d_{xadrez} = ([i_1, j_1], [i_2, j_2]) = \max(|i_2 - i_1| + |j_2 - j_1|) \quad \text{(eq. 12)}$$

Em processamento de imagem não é habitual utilizar a distância euclidiana mas sim uma das outras duas, devido a tempo de cálculo computacional ser mais rápido.

3.2.2.10 – Filtros de deteção de objetos

No processo de deteção de objetos são usadas técnicas de segmentação de forma a selecionar na imagem possíveis pontos ou objetos de interesse. A segmentação é o processo pelo qual uma imagem digital é dividida em regiões, agrupando os pixéis segundo um determinado critério. A segmentação por cor, intensidade dos pixéis, textura ou algoritmos de deteção de contornos, vértices e descontinuidades são importantes no contexto da análise de imagem [14].

Tal como apresentado na figura que se seguem (Figura 30), o processo de segmentação de imagem pode ser efetuado com a aplicação de várias técnicas de deteção de contornos, tais como: *Laplacian of Guassian* (LoG), *Sobel*, *Prewitt*, *Canny*, *Otsu*, entre outros [15].

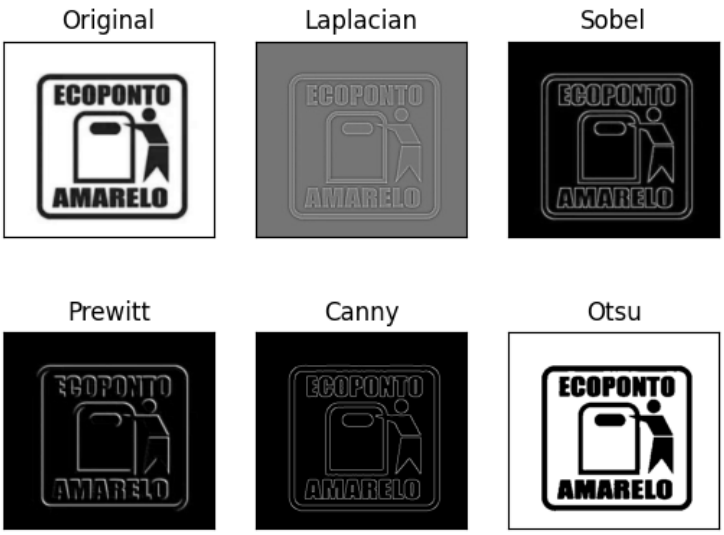


Figura 30 - Técnica de segmentação com recurso ao operador Sobel, Laplacian, Sobel, Prewitt, Canny e Otsu

O operador *Canny* é o que melhor define os contornos da imagem [15]. Este pode ser bom para certos tipos de análise de imagem, contudo em casos particulares pode fornecer informação a mais e dificultar a análise da imagem, o que não se verifica numa imagem gráfica como a da Figura 30 mas em imagens obtidas com a câmara poderá acontecer.

A **técnica de subtração do fundo** é uma abordagem amplamente utilizada na inspeção de placas de circuito impresso (PCB), que tem como intuito detetar ausência ou excesso de componentes nas PCBs. Esta técnica é baseada num processo de subtração de pixéis, em que se subtrai o valor de cada pixel da imagem de referência, pelo respetivo pixel da imagem inspecionada. Este processo de subtração tem como resultado uma terceira imagem, e pode ser deduzido pela seguinte expressão [16]:

$$g(i,j) = f_1(i,j) - f_2(i,j) \qquad \text{(eq. 13)}$$

Onde, g: imagem resultante; f_1: imagem de referência; f_2: imagem inspecionada

A Figura 31 ilustra um exemplo de aplicação do método de subtração do fundo em inspeção de PCB.

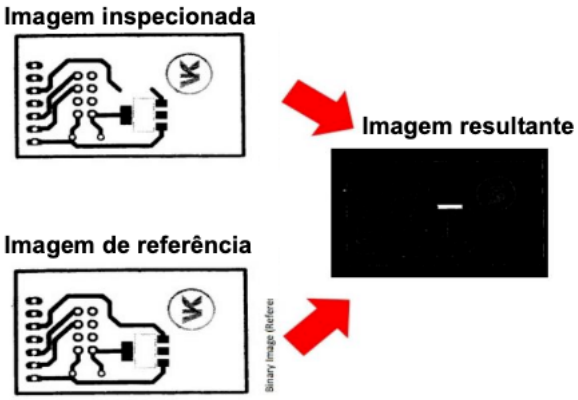


Figura 31 - Aplicação do método de subtração do fundo em inspeção de PCB

3.2.2.11 – Reconhecimento de Objetos

Após a segmentação da imagem, dependendo do objetivo que se pretende para o sistema, por vezes é necessário obter informações que classifiquem e descrevam os objetos. Características como a área, o diâmetro, o centro de gravidade e o perímetro são frequentemente utilizadas. Contudo, devido ao facto de serem sensíveis ao fator de escala, podem ser insuficientes para uma classificação correta e completa dos objetos em questão. Casos em que seja necessário distinguir objetos com a mesma forma é necessário usar outros algoritmos. Neste contexto, os histogramas de cor são bastante úteis. Comparando os histogramas de cor de cada objeto, pode-se facilmente obter uma descrição de cada um deles [14].

Com o intuito de tornar o reconhecimento imune às variações referidas anteriormente, existem algoritmos que se baseiam na extração de características específicas, normalmente pertencentes à sua superfície, cantos ou arestas. Dois dos algoritmos mais utilizados para a extração de características e reconhecimento de objetos são o Scale Invariant Feature Transform (SIFT) e Speed Up Robust Features (SURF). Devido à sua robustez e ao facto de se centrarem em características invariantes à escala e rotação (ver Figura 32), permite que o reconhecimento seja eficaz em diversos tipos de situações [17].

SIFT

O SIFT é um algoritmo de visão computacional desenvolvido por David G. Lowe que é utilizado frequentemente para o reconhecimento de objetos através da deteção e extração de descritores invariantes em escala e rotação. Este é bastante utilizado em aplicações envolvendo a correspondência de diferentes imagens de um objeto ou cena devido a serem razoavelmente invariantes a mudanças de iluminação, ruído na imagem, rotação e escala ([17], [18]). Este algoritmo é composto por dois elementos distintos, o detetor que é implementado com recurso a funções Difference of Gaussian (DoG) e o descritor, que se obtém com base em histogramas de orientações da vizinhança local dos pontos-chave. O DoG é um algoritmo que salienta características particulares de imagens com recurso a subtrações de imagens processadas por filtros gaussianos a diferentes escalas ([17], [18]).

A obtenção dos descritores de uma imagem pode-se resumir nas seguintes etapas ([17], [18]):

- Deteção de extremos no espaço-escala;
- Localização de pontos-chave;
- Determinação de orientação;
- Descritor de pontos-chave.

Após a obtenção dos descritores para cada ponto, a tarefa de encontrar a correspondência entre imagens resume-se a encontrar entre os descritores de uma imagem, os melhores candidatos a serem os seus equivalentes na outra imagem [17].

Figura 32 - Resultado da aplicação do algoritmo SIFT [17]

SURF

O SURF é um algoritmo de visão computacional desenvolvido por *Herbert Bay* inspirado no algoritmo SIFT, tendo, portanto, atributos idênticos ([17], [19]). Este foi desenvolvido com o intuito de tornar o processo de identificação de aspetos semelhantes entre imagens mais célere. A principal diferença face ao SIFT reside na fase de deteção dos pontos-chave, que neste é efetuada com recurso a um detetor *Hessian*. Este detetor é sustentado na matriz Hessiana (matriz composta por derivadas parciais de segunda ordem (eq. 14)) capaz de identificar zonas de imagem onde se verificam variações acentuadas de intensidade.

$$H[f(x_1, x_2, \dots, x_n)] = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial {x_1}^2 \partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial {x_1}^2 \partial {x_n}^2} \\ \frac{\partial^2 f}{\partial {x_2}^2 \partial {x_1}^2} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial {x_2}^2 \partial {x_n}^2} \\ \vdots & \vdots & \ddots & \\ \frac{\partial^2 f}{\partial {x_n}^2 \partial {x_1}^2} & \frac{\partial^2 f}{\partial {x_n}^2 \partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{bmatrix} \qquad \text{(eq. 14)}$$

A procura por correspondências segundo este algoritmo pode ser dividida essencialmente em três fases ([17], [20]):

- Encontrar pontos-chave na imagem, definindo-os como aqueles que sejam identificáveis em diferentes condições de imagem;
- Representar por um vetor de características ou descritores a vizinhança dos pontos chave. Devem ser distintivos e robustos ao ruído, às deformações geométricas da imagem e aos erros;
- Comparar os vetores descritores da imagem com o objeto. A comparação é geralmente baseada na distância entre vetores.

Figura 33 - Resultado da aplicação do algoritmo SURF [17]

Segundo os dados experimentais obtidos em [19], o algoritmo SURF possui uma maior precisão nos pontos-chave relativamente ao algoritmo SIFT.

3.2.3 – Tecnologias de Reconhecimento

Atualmente existem várias bibliotecas que permitem, recorrendo aos algoritmos referidos atrás, reconhecer e tratar imagens, algumas dessas bibliotecas são OpenCV, FastCV, Camellia Lybrary.

OpenCV

Open Source Computer Vision (OpenCV) é, como o nome indica, uma biblioteca de código aberto para processamento gráfico com uma componente de aprendizagem de sistemas. Com interfaces para um vasto conjunto de linguagens e bibliotecas de programação, o OpenCV é compatível com as plataformas desktop Windows, Linux e Mac, e com as plataformas móveis Android e iOS. Produzido com o intuito de servir de infraestrutura comum a diversas aplicações relacionadas com robótica e computação gráfica, e sempre com o objetivo de atingir a máxima eficácia computacional, esta biblioteca acelera a evolução de produtos comerciais capazes de tirar o máximo proveito do multiprocessamento em tempo real dos dispositivos atuais.

Escrita originalmente em C/C++, disponibiliza camadas de tradução das funções básicas para outras linguagens de programação como Python e Java, atingindo um universo de quarenta e sete mil pessoas ativas na comunidade do OpenCV, com o total de descargas da plataforma a exceder os sete milhões.

Com mais de dois mil e quinhentos algoritmos otimizados para servir áreas como arte interativa, inspeção de minas, informação geográfica ou robótica, esta plataforma implementa tanto técnicas clássicas de computação gráfica e aprendizagem de sistemas, como as soluções mais recentes propostas pela comunidade científica. Neste vasto conjunto podem ser encontrados algoritmos para detetar e reconhecer faces, identificar objetos, classificar ações humanas em vídeos, agregar imagens para produzir uma única imagem de alta resolução, encontrar imagens semelhantes numa base de dados, introduzir realidade aumentada numa cena de objetos, entre outros.

Distribuída sobre os termos da licença BSD (licença de código aberto), esta biblioteca é de livre utilização tanto para uso comercial como académico, permitindo que entidades acedam e alterem o código conforme necessário.

FastCV

O projeto FastCV é o resultado do esforço realizado pela empresa Qualcomm na construção de uma biblioteca e um conjunto de ferramentas de desenvolvimento para processamento de imagem, destinados exclusivamente para suportar o desenvolvimento de aplicações com capacidades gráficas avançadas em dispositivos móveis. Desenhada especificamente para ter uma performance eximia no sistema operativo Android. Esta biblioteca, ainda na sua versão 1.0, é distribuída sobre a forma de um único ficheiro binário, contendo código otimizado para processadores ARM e Snapdragon.

Apresentada como sendo a primeira biblioteca de processamento de gráficos e imagem dedicada para as arquiteturas dos dispositivos móveis, a biblioteca FastCV disponibiliza interfaces de programação com capacidades de aceleração por hardware, especialmente em processadores Snapdragon desenvolvidos pela empresa. Devido à sua vasta coleção de funções para processamento de imagem a baixo nível e interação com sensores e câmaras, este projeto constitui a base da solução da Qualcomm para realidade aumentada: Vuforia Augmented Reality. Segundo a empresa, o FastCV pode ser utilizado diretamente nas aplicações, ou utilizado por programadores de middleware (software de que fornece serviços para softwares e aplicativos além daqueles disponíveis pelo sistema operacional) no desenho e conceção de plataformas de reconhecimento de gestos, faces, e objetos, assim como a deteção de movimento, cálculo de profundidade de campo, identificação de caracteres, entre outros.

Assente numa arquitetura modular, o FastCV é composto por diversos componentes como operações matemáticas e vetores, processamento de imagem, transformação de imagem, deteção de áreas de interesse, deteção de objetos, reconstrução 3D, clustering e pesquisa e seguimento de objetos e movimento, mantendo uma documentação vasta sobre cada um destes módulos.

Com o conjunto de ferramentas de desenvolvimento do FastCV Android disponível para Windows, Mac e Linux sobre a licena FastCV SDK License (sem custos de utilização), a equipa da Qualcomm responsável pelo projeto prevê o lançamento desta solução para outros sistemas operativos móveis como iOS e Windows Phone.

Camellia Lybrary

Camellia é o resultado de uma colaboração entre parceiros como École des Mines de Paris (ENSMP), Philips, Universidade de Hannover, Universidade de Las Palmas e Renault, que se reuniram para o desenvolvimento de uma biblioteca de processamento de gráficos e imagem, com o intuito de ser multiplataforma, robusta e com a máxima performance. Totalmente escrita em C, esta biblioteca de código livre inclui funções para o processamento de imagem convencional como aplicar filtros nas imagens, distorção, manipulação de gráficos, e conversão de cores.

Encarado como o projeto substituto do Intel IPL, atualmente descontinuado, o Camellia mantém completa interoperabilidade com o OpenCV, embora adicionando camadas de tradução para facilitar a compreensão das funções. Possível de ser utilizado na linguagem C++, permitindo às equipas de desenvolvimento retirar todas as vantagens da programação orientada a objetos, assim como uma interface para a linguagem interpretada Ruby, que inclui o suporte a exceções e otimização de memória.

Suportando imagens até 16 bits de profundidade, processamento de regiões de interesse, e aplicação de máscaras de bits, contém algoritmos exclusivos para processamento de imagem como Watershed (técnica de segmentação de imagens, método conhecido como “Linhas Divisoras de Água”) e estimativa do movimento.

Distribuído sobre a licença BSD (licença de código aberto) e publicado através do sítio de alojamento de software Sourceforge, este projeto pretende crescer com a comunidade apelando

a contribuições para manter a biblioteca atualizada e proeminente na área do processamento de imagem.

3.3 – Linguagem de Programação – Python

A linguagem de programação Python foi a escolhida para ser utilizada neste projeto, esta é uma linguagem fácil de aprender e poderosa. Ela tem estruturas de dados de alto nível eficientes e uma abordagem simples, mas efetiva de programação orientada a objetos. [21]

O Python combina flexibilidade de alto nível, legibilidade e uma interface bem definida com capacidades de baixo nível, incluindo uma interface oficial em C que permite estender a linguagem com código em C e ligar a bibliotecas de terceiros em C, C++ e Fortran. Essa generalidade é benéfica para o cálculo científico moderno: é um ambiente produtivo de uso diário que também permite otimizar tarefas de desempenho crítico. Além disso, oferece flexibilidade para construir ferramentas com um equilíbrio preciso entre recursos de baixo e alto nível, para que se possa escolher adequadamente entre desempenho e facilidade de desenvolvimento ou uso. [22]

Figura 34 - Símbolo da Linguagem de Programação Python

Outra linguagem de programação possível de utilizar seria o Matlab (que foi utilizado em diversas Unidades Curriculares deste curso), mas sendo o Python uma linguagem de código aberto, tem a vantagem económica para além dos algoritmos desenvolvidos poderem ser utilizados em diversos dispositivos.

Neste capítulo foi possível perceber qual a história e o desenvolvimento tecnológico no âmbito industrial bem como no de processamento de imagem, conhecendo também as bibliotecas utilizadas no dia de hoje tratamento e categorização de imagens. Percebeu-se a linguagem de programação que será utilizada e o porque da sua massiva utilização hoje.

Capítulo 4

DESENVOLVIMENTO DO PROJETO

Neste capítulo serão apresentadas as soluções desenvolvidas para o controlo do sistema de resíduos recicláveis. Serão apresentados os *softwares* e bibliotecas utilizadas, o equipamento utilizado, os vários algoritmos implementados e a sua descrição funcional.

Atualmente, em aplicações industriais, são usualmente utilizados dois tipos de plataformas computacionais, os computadores convencionais e os chamados *single-board computers*.

Os computadores convencionais são comercializados numa das duas formas: *desktops* ou computadores portáteis. Os *desktops* são compostos por uma torre central que incorpora vários componentes no seu interior e uma série de periféricos acoplados (teclado, rato, colunas e monitor). Este tipo de solução só é apropriado para sistemas que suportem o seu volume estrutural. Já os computadores portáteis, possuindo um volume estrutural mais reduzido (já incluem vários periféricos, tais como teclado, rato, colunas e monitor no seu *chassi*), são normalmente utilizados no âmbito industrial, e na maior parte das vezes como ferramenta de programação/teste dos sistemas indústrias. Além do seu volume reduzido em comparação com os *desktops*, os computadores portáteis também incluem uma bateria, permitindo o seu uso durante um período considerável sem a conexão permanente a uma fonte de alimentação externa [23].

A outra opção de plataforma computacional são os *single-board computers*. São pequenas placas eletrónicas (do tamanho de um cartão de crédito) que partilham uma arquitetura comum com um computador convencional. Estes incluem um processador, memória RAM, unidade gráfica e unidade de *Inputs*/*Outputs* (entradas/saídas). Este tipo de computadores também possibilitam a conexão de periféricos usuais, tais como o teclado (conexão USB), rato (conexão USB) e monitor (conexão HDMI). Os *single-board computers* funcionam normalmente com recurso a um processador ARM (similar aos utilizados nos *smartphones*) [23].

Neste trabalho foi desenvolvida a solução para um portátil, o mesmo poderia ter sido colocado num single-board computer, visto que a linguagem de programação é nativa nestas plataformas.

O sistema idealizado seria composto como o da Figura 35. Constituído por um tapete rotativo onde iria ser depositado o lixo de forma individual, o movimento do tapete seria controlado por uma plataforma computacional baseada na leitura de uma câmara. Esta câmara estaria por cima do tapete e em tempo real enviaria informação para o controlador. Mediante a sua leitura o controlador faria avançar o tapete e consequentemente faria a separação do lixo recorrendo a três pistolas de ar comprimido, como se pode verificar na Figura 36.

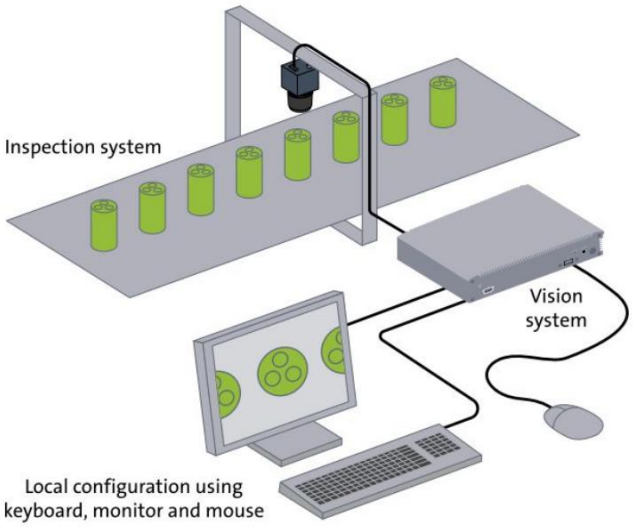


Figura 35 - Sistema Protótipo [27]

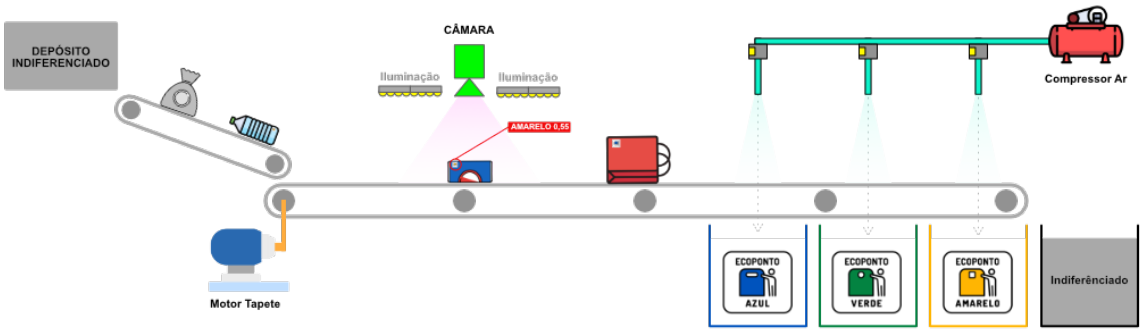


Figura 36 - Topologia Protótipo

A Figura 36 demonstra mais a pormenor o funcionamento do circuito. De um depósito indiferenciado o RU é filtrado indo para o tapete apenas um a um. O movimento do tapete é dado pelo motor tapete e controlado pela plataforma computacional (não apresentada na imagem). A câmara deteta a presença de um objeto e analisa-o à procura de um conjunto de selos iconográficos. Mediante estes selos decide se o objeto é para separar de forma diferenciada ou de forma indiferenciada. Tendo o resultado, o controlador faz rolar o tapete levando o objeto até ao grupo de pistolas de ar comprimido. Dependendo do depósito a colocar o RU, este permanece mais tempo no tapete até chegar à zona de disparo da pistola assignada

ao tipo de resíduo (azul, verde ou amarelo). Se o resíduo for indiferenciado continua no tapete até cair para um depósito de resíduos indiferenciados.

Sendo assim os materiais necessários para o mecanismo seriam:

- Depósito Indiferenciado Total;
- Motor dos tapetes e tapetes;
- Equipamento de iluminação da área de visualização da câmara;
- Câmara de vídeo;
- Compressor e pistola de ar;
- 4 depósitos final - azul, verde, amarelo e indiferenciado.

No âmbito do projeto, foi apenas desenvolvido o algoritmo de deteção de sinalética. Assim considera-se que os resíduos possuem a sinalética de acordo com a lei em vigor. Inicialmente a deteção iria recorrer a métodos tradicionais como falado anteriormente. Depois de uma investigação aprofundada decidiu-se que o algoritmo desenvolvido em Python utilizaria um modelo de deteção de objetos YOLO dada á sua versatilidade e rapidez.

4.1 – Algoritmo de Deteção de Sinalética

Como visto no capítulo Sinalética nas Embalagens, a sinalética em vigor em Portugal para a indicação de material reciclável e em que depósito o colocar está marcado com símbolos diferentes. O algoritmo desenvolvido foi treinado para fazer a deteção de 3 tipos de símbolos (azul, verde e amarelo), os restantes símbolos apresentados no Sinalética nas Embalagens não foram utilizados nesta primeira fase.

Figura 37 - Símbolos Detetados Algoritmo

4.2 – YOLO

YOLO (You Only Look Once) é um modelo popular de deteção de objetos e segmentação de imagens desenvolvido por Joseph Redmon e Ali Farhadi na Universidade de Washington. A primeira versão do YOLO foi lançada em 2015 e rapidamente ganhou popularidade devido à sua alta velocidade e precisão. Desde o lançamento até 2021 já foram feitas várias atualizações no software de modo a adicionar novos recursos.

O YOLO tem sido amplamente usado em uma variedade de aplicações, incluindo veículos autônomos, segurança e vigilância e imagens médicas. [24]

Neste trabalho foi utilizado a versão 5 do modelo YOLO, com nome YOLOv5.

Arquitetura

O modelo YOLO usa uma rede neural que faz previsões de caixas delimitadoras e probabilidades de classe de uma só vez. Isto difere de outros algoritmos de deteção de objetos, que reutilizam classificadores para realizar a deteção.

O algoritmo YOLO recebe como entrada uma imagem e, em seguida, utiliza uma rede neural de convolução profunda simples para detetar objetos na imagem. A arquitetura do modelo CNN que forma a base do YOLO é mostrada abaixo.

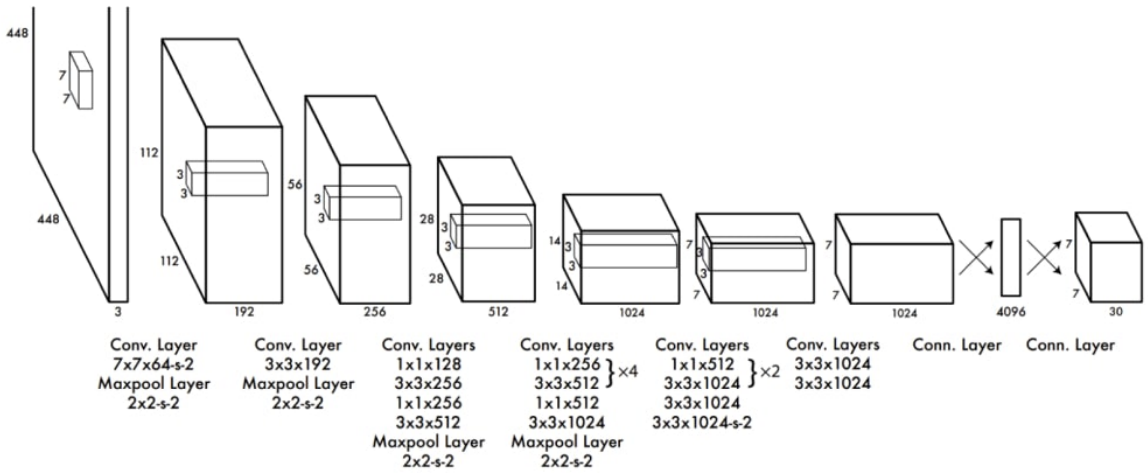


Figura 38 - Arquitetura YOLO [29]

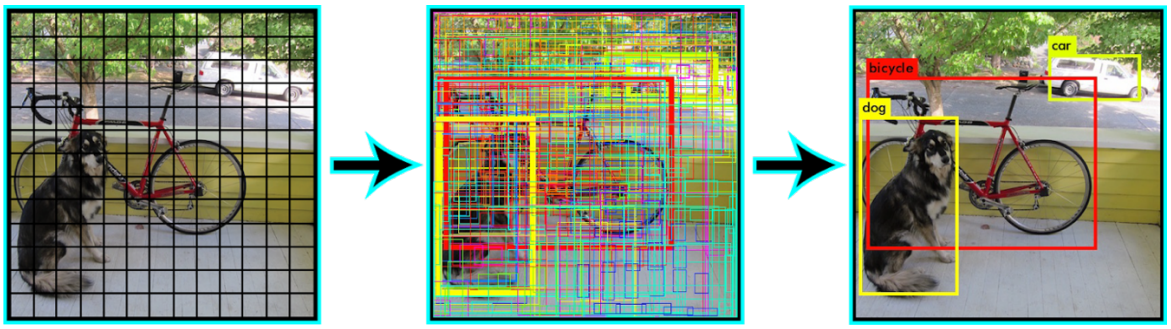
A Figura 38 mostra as primeiras 20 camadas de convolução do modelo são pré-treinados usando ImageNet (banco de dados de imagens utilizados em investigação). Em seguida, este modelo pré-treinado é convertido para realizar a deteção, pois pesquisas anteriores mostraram que adicionar camadas de convolução conectadas a uma rede pré-treinada melhora o desempenho. A camada totalmente conectada final do YOLO prevê tanto as probabilidades de cada categoria como as coordenadas da caixa delimitadora.

Tecnologia

O YOLO divide uma imagem de entrada em uma matriz S × S. Se o centro de um objeto cair em uma célula de grade, essa célula de matriz é responsável por detetar esse objeto. Cada célula da matriz prevê B caixas delimitadoras e pontuações de confiança para essas caixas. Essas pontuações de confiança refletem o quão confiante o modelo é de que a caixa contém um objeto e o quanto ele acha que a caixa prevista é precisa.

O YOLO prevê múltiplas caixas delimitadoras por célula de matriz. Na hora do treino, queremos que apenas um indicador de caixa delimitadora seja responsável por cada objeto. O YOLO atribui um indicador a ser "responsável" por prever um objeto com base na previsão que tem o IOU (Interseção sobre União, explicado mais à frente) atual mais alto. Isso leva a uma especialização entre os indicadores de caixa delimitadora. Cada indicador fica cada vez melhor em prever certos tamanhos, proporções ou classes de objetos, melhorando a sua pontuação geral.

Uma técnica chave usada nos modelos YOLO é a supressão não máxima (NMS). NMS é uma etapa de pós-processamento que é usada para melhorar a precisão e eficiência da deteção de objetos. Na deteção de objetos, é comum que múltiplas caixas delimitadoras sejam geradas para um único objeto em uma imagem. Estas caixas delimitadoras podem-se sobrepor ou estar em posições diferentes, mas todas representam o mesmo objeto. A técnica NMS é utilizada para identificar e remover caixas redundantes ou incorretas para cada objeto da imagem. [25]



a) b) c)

Figura 39 - Técnica NMS – a) Imagem de entrada com matriz SxS, b) Imagem com caixas delimitadores possíveis, c) Imagem com a previsão do modelo [31]

Redes Neuronais no YOLO

O modelo YOLO utiliza redes neuronais no seu algoritmo interno. Uma rede neuronal é um mapeador. Mapeia uma variável de entrada x, numa variável de saída y. A variável de entrada corresponde aos atributos ou características a analisar. A variável de saída corresponde ao que se pretende atingir. Pode ser discreta (classificação) ou contínua (regressão).

Num problema de classificação o objetivo é encontrar a fronteira de decisão que melhor permita distinguir as classes/ categorias a classificar.

As redes neurais artificiais são um conceito da computação que visa trabalhar no processamento de dados de maneira semelhante ao cérebro humano. O cérebro é tido como um processador altamente complexo e que realiza processamentos de maneira paralela. Para isso, ele organiza a sua estrutura, ou seja, os neurônios, de forma que eles realizem o processamento necessário. Isso é feito numa velocidade extremamente alta e não existe qualquer computador no mundo capaz de realizar o que o cérebro humano faz.

Nas redes neurais artificiais, a ideia é realizar o processamento de informações tendo como princípio a organização de neurônios do cérebro. Como o cérebro humano é capaz de aprender e tomar decisões baseadas na aprendizagem, as redes neurais artificiais devem fazer o mesmo. Assim, uma rede neural (Figura 40) pode ser interpretada como um esquema de processamento capaz de armazenar conhecimento baseado em aprendizagem (experiência) e disponibilizar este conhecimento para a aplicação em questão. [26]

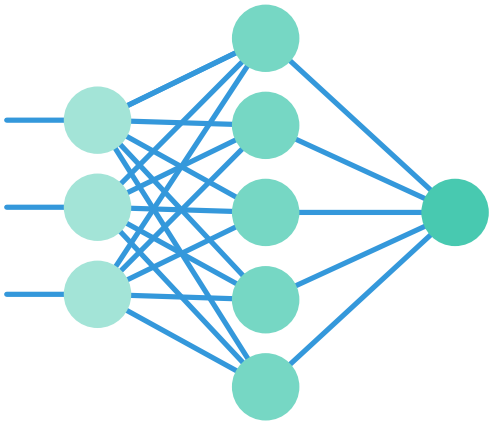


Figura 40 - Exemplo rede neuronal de 3 camadas

No exemplo da rede neuronal da Figura 41, pode-se verificar que a primeira camada, a camada de entrada, consiste em oito nós. Cada um dos oito nós nesta camada representa um recurso individual de uma amostra específica do conjunto de dados.

Assim uma única amostra do conjunto de dados consiste em oito dimensões. Quando se escolhe uma amostra do conjunto de dados e se passa essa amostra para o modelo, cada um dos oito valores contidos na amostra será fornecido a um nó correspondente na camada de entrada.

Cada um dos oito nós de entrada está conectado a todos os nós na próxima camada. Cada conexão entre a primeira e segunda camadas transfere a saída do nó anterior para a entrada do nó recetor (da esquerda para a direita). As duas camadas no meio que têm seis nós cada são camadas escondidas simplesmente porque estão posicionadas entre as camadas de entrada e saída. À conexão entre dois nós dá-se o nome de peso (weight) e o melhor resultado deste valor indicará o melhor modelo.

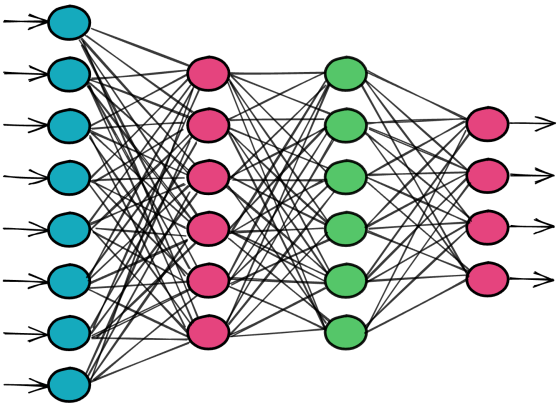


Figura 41 - Exemplo rede neuronal com 8 entradas

Stochastic gradient descent é um algoritmo de aprendizagem utilizado para treinar o YOLO que tem uma série de hiperparâmetros.

Dois hiperparâmetros importantes são o:

- batch size – tamanho do lote
- epochs – número de épocas.

Stochastic gradient descent é um algoritmo de aprendizagem iterativa que usa um conjunto de dados de treino para atualizar um modelo. O tamanho do lote é um hiperparâmetro do gradiente descente que controla o número de amostras de treino a serem trabalhadas antes que os parâmetros internos do modelo sejam atualizados. O número de épocas é um hiperparâmetro do gradiente descente que controla o número de passagens completas pelo conjunto de treino. [27]

Como exemplo, se a base de dados contiver 1000 imagens para treinar o modelo (rede), de modo a identificar diferentes categorias, então o tamanho do lote (batch) será 10. Isso significa que 10 imagens serão passadas como um grupo ou como um lote, pela rede, no momento do treino do modelo.

Dado que uma única época (epoche) é uma única passagem de todos os dados pela rede, levará 100 lotes para compor uma época completa. Tendo 1000 imagens divididas por um tamanho de lote de 10, o que equivale a 100 lotes totais.

$$batches\ in\ epoch\ =\ training\ set\ size\ /\ batch\_size \qquad \text{(eq. 15)}$$

Fases do Processo YOLO

Criar um modelo personalizado para detetar objetos é um processo iterativo de coleta e organização de imagens, rotulagem dos objetos de interesse, treino de um modelo, implantação no ambiente para fazer previsões e, em seguida, usar esse modelo treinado para recolher exemplos de casos para este repetir o treino e melhorar o modelo. A Figura 42, recolhida do website da YOLO demonstra o processo cíclico.

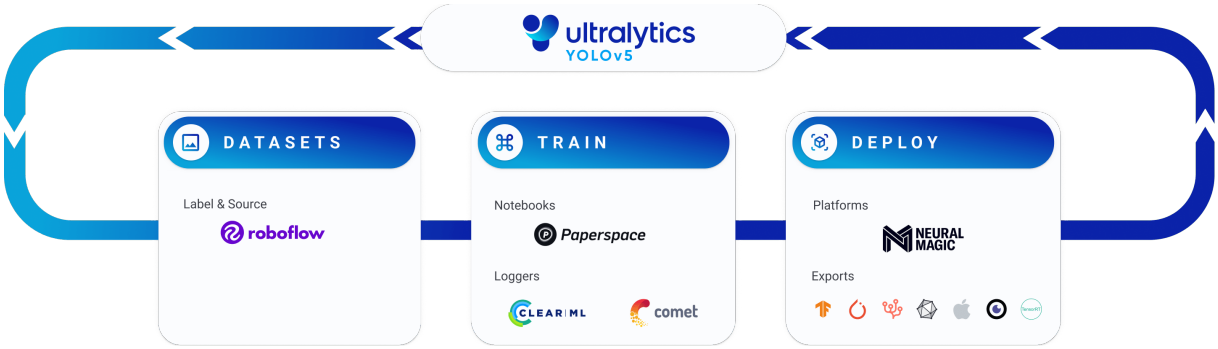


Figura 42 - Ciclo utilização YOLO [34]

Numa primeira fase é preciso criar a base de dados de imagem categorizada, de seguida treinar o modelo e por fim utilizar o modelo. Com a utilização do modelo podemos alimentar novamente a base de dados e assim sucessivamente.

A Figura 42 mostra diversos softwares onde as três tarefas podem ser realizadas, para a categorização da base de dados, neste projeto, será utilizada a aplicação roboflow. O sucesso do sistema consiste na seleção de imagens que irão ser apresentadas ao sistema que servirão de aprendizagem (treino).

Mediante a base de dados e a categorização das imagens o YOLO pode ser treinado para detetar diversos objetos, na Figura 43 o modelo YOLO foi utilizado para detetar aviões. Como se pode ver na Figura 43, após o treino, é possível identificar vários aviões, com tamanhos e posições diferentes.

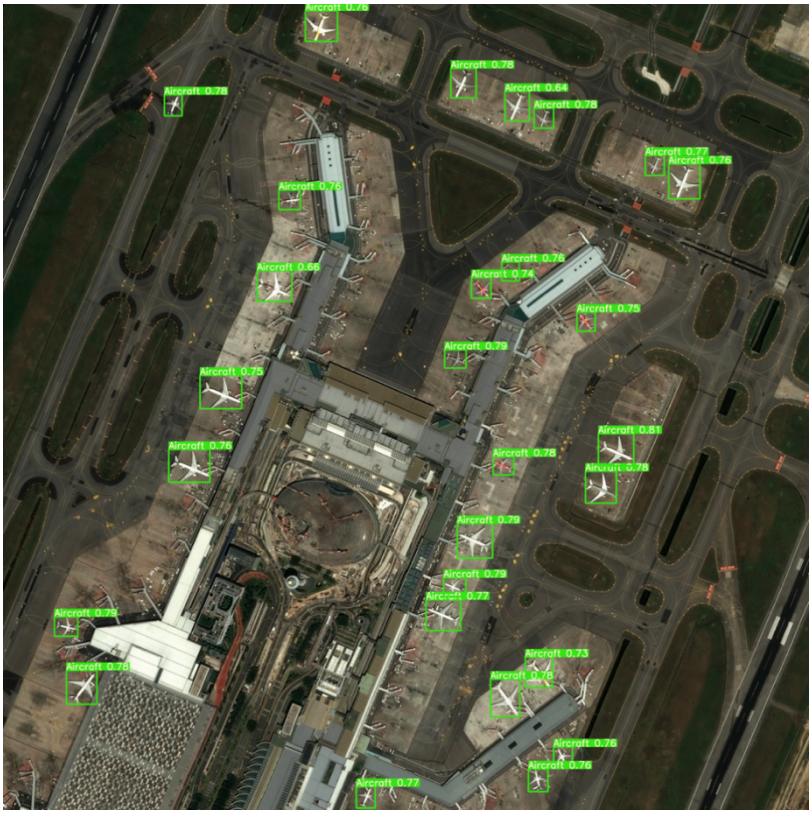


Figura 43 - Modelo YOLO para detetar aviões [35]

4.3 – Sistema Desenvolvido

Neste capítulo são descritas as etapas do sistema desenvolvido neste projeto e das suas fases de implementação: Banco de Imagens, Categorização das imagens, As 80 Categorias Pré-Treinadas YOLO, e por fim o Algoritmo Final de Deteção de Sinalética.

Conforme mencionado anteriormente, um sistema que utiliza processamento de imagens para classificação é altamente dependente das imagens utilizadas para treino do sistema e das categorias selecionadas para a sua utilização.

4.3.1 – Banco de Imagens

Para treinar o modelo de deteção de imagem é necessário fornecer imagens ao modelo de treino, nesse sentido foram recolhidas algumas amostras de RU e de seguida foram fotografadas, evidenciando o seu selo iconográfico, as fotografias foram tiradas com um fundo negro para simular o tapete rolante falado na Figura 36 - Topologia Protótipo.

Alguns exemplos são os que se apresentam na Figura 44:

Figura 44 - Exemplos de Fotografias de RU

O valor total do banco de imagens corresponde a 616 fotografias, com diversos resíduos urbanos, de diversas cores, formatos e de diferente caracterização de reciclagem. Nas fotografias existem também aberrações, elementos de RU urbanos colocados de forma propositada de forma a não se ver a simbologia, para criar entropia na imagem e no teste ao modelo.

4.3.2 – Categorização das imagens

Para treinar o modelo, é necessário escolher uma ferramenta de categorização adequada para categorizar o conjunto de dados personalizado. O modelo YOLO exige que os dados usados para o treino tenham cada uma das classificações desejadas rotuladas com precisão, geralmente manualmente. Para esta tarefa utilizou-se o roboflow. Este é ferramenta gratuita para uso online, rápida, pode realizar melhorias de imagem e transformações nos dados carregados para diversificar o conjunto de dados e pode até triplicar a quantidade de dados de treinamento com base nas configurações de entrada.

Para enriquecer o conjunto de dados, melhorando o modelo, foram colocadas imagens dos símbolos iconográficos, com o fundo branco.

Na plataforma roboflow o primeiro passo é carregar as imagens, como se vê na Figura 45:

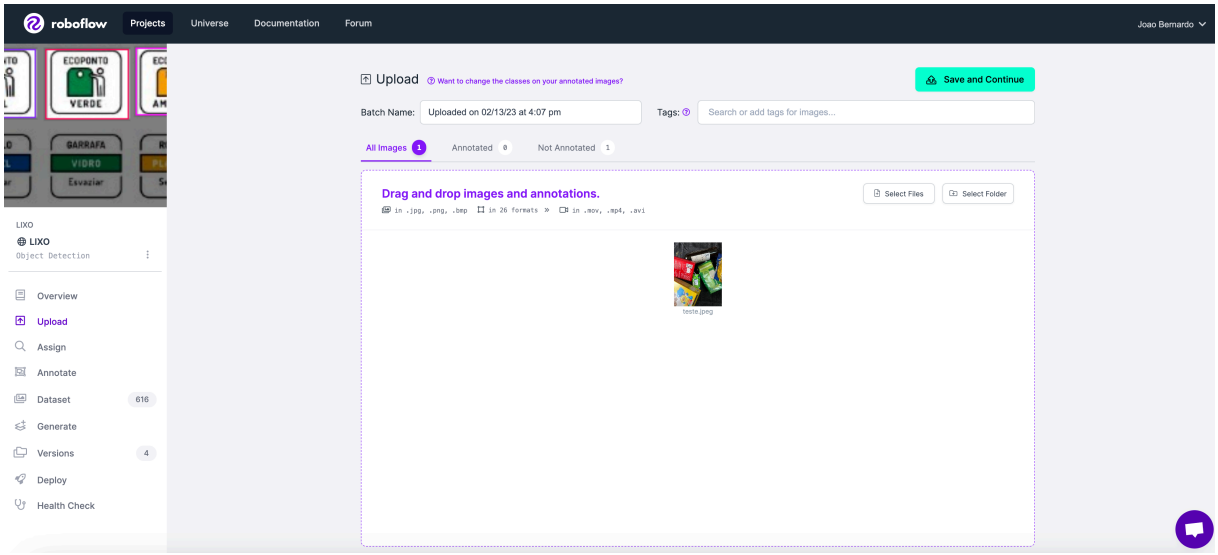


Figura 45 - Carregar imagens na plataforma roboflow

De seguida é necessário categorizar as imagens, neste passo escolhemos as categorias com que o nosso modelo será treinado e que irá utilizar na busca quando em utilização. Neste projeto foram criadas 3 categorias (Figura 46):

- AZUL
- VERDE
- AMARELO

Figura 46 - Categorizar imagens plataforma roboflow

Quando as imagens estiverem todas categorizadas teremos uma base de dados de imagens com as categorias necessárias para o treino do modelo (Figura 47).

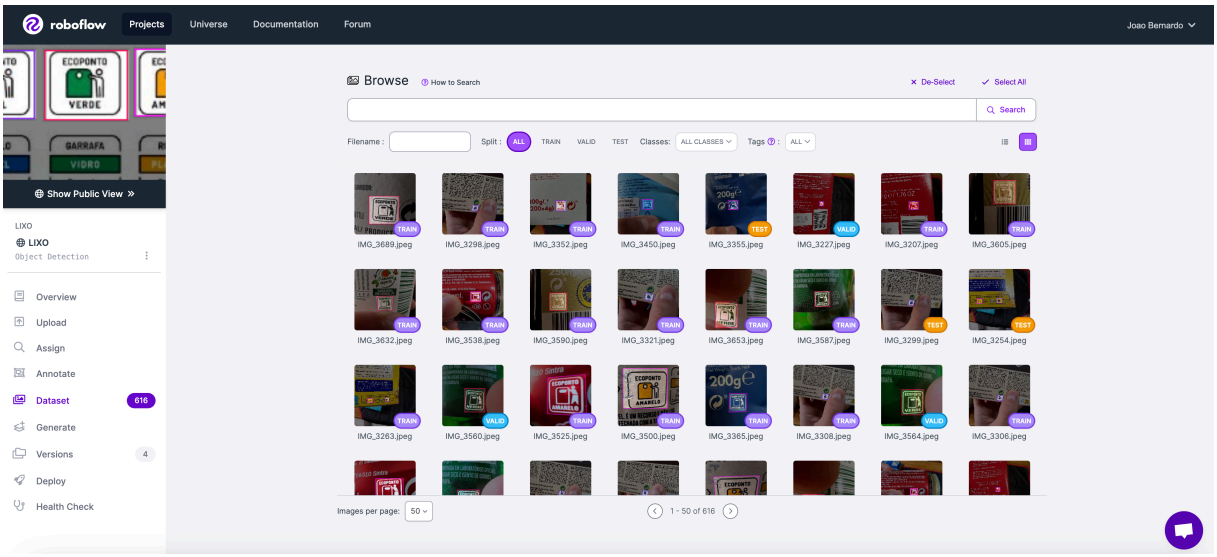


Figura 47 - Base de dados carregada e categorizada na plataforma roboflow

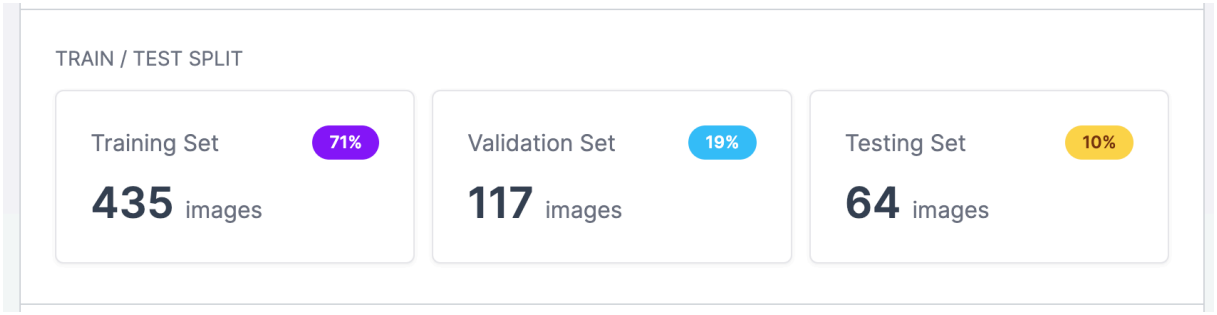


Figura 48 - Separação Banco de Imagens no roboflow

Em geral, é sugerido que o modelo seja treinado com 2000 imagens para cada categoria de classificação. Para o propósito desta aplicação foram utilizadas 616 imagens (294 – categoria AMARELO, 239 – categoria AZUL, 220 – categoria VERDE). Sabendo que é possível melhorar a capacidade deste modelo adicionando mais imagens, a etapa mais importante seria expô-lo a mais dados de treino e a um conjunto de validação mais robusto (Figura 48).

No treino de modelos de reconhecimento é aconselhado que o conjunto de dados se divida em 70% imagens de treino, 20% validação e 10% teste.

O **conjunto de imagens de treinamento** (Figura 48 – Training Set) é o conjunto de dados usado para treinar e fazer o modelo aprender as características/padrões ocultos nos dados. Em cada epoche (época), os mesmos dados de treino são alimentados repetidamente na arquitetura da rede neural, e o modelo continua a aprender as características dos dados. O conjunto de treino deve ter uma variedade diversificada de entradas para que o modelo seja treinado em todos os cenários e possa prever qualquer amostra de dados não vista que possa aparecer no futuro.

O **conjunto de validação** (Figura 48 – Validation Set) é um conjunto de dados, separado do conjunto de treinamento, usado para validar o desempenho do nosso modelo durante o treinamento. Esse processo de validação fornece informações que ajudam a ajustar os hiperparâmetros e configurações do modelo. Tem a função de um crítico dizendo se o treino está a seguir a direção correta ou não. O modelo é treinado no conjunto de treinos e, simultaneamente, a avaliação do modelo é realizada no conjunto de validação após cada época.

A ideia principal de dividir o conjunto de dados em um conjunto de validação é evitar que o modelo sofra overfitting (Sobre-ajuste), ou seja, o modelo torna-se muito bom em classificar as amostras no conjunto de treino, mas não pode generalizar e fazer classificações precisas em dados que não foram vistos antes.

O **conjunto de teste** (Figura 48 – Testing Set) é um conjunto separado de dados usado para testar o modelo após a conclusão do treinamento. Ele fornece uma métrica final de desempenho do modelo imparcial em termos de precisão. Em termos simples, este conjunto permite responder à pergunta de como o modelo se sai em termos de desempenho.

Tendo esta tarefa feita o roboflow permite a exportação de dados ou a implementação em plataformas computacionais de treino, o que vai ser visto mais à frente.

4.3.3 – As 80 Categorias Pré-Treinadas YOLO

Todos os modelos YOLO podem ser utilizados sem serem treinados, eles vêm com a capacidade de reconhecer 80 objetos. Com o propósito de perceber a utilização do modelo e fazer uma comparação futura com o modelo testado, foi criado um algoritmo para testar imagens com este modelo. Este algoritmo escrito na linguagem python, e com recurso à biblioteca opencv consta do Anexo 1a e Anexo 1b deste documento.

A versão do modelo escolhida para este teste é a 3. Para correr este modelo são necessários os ficheiros “yolov3.weights”, “yolov3.cfg” e “coco.names” que são a base do modelo.

As 80 categorias para as quais o modelo foi detetado são:

1. person
2. bicycle
3. car
4. motorbike
5. aeroplane
6. bus
7. train
8. truck
9. boat
10. traffic light
11. fire hydrant
12. stop sign
13. parking meter
14. bench
15. bird
16. cat
17. dog
18. horse
19. sheep
20. cow
21. elephant
22. bear
23. zebra
24. giraffe
25. backpack
26. umbrella
27. handbag
28. tie
29. suitcase
30. frisbee
31. skis
32. snowboard
33. sports ball
34. kite
35. baseball bat
36. baseball glove
37. skateboard
38. surfboard
39. tennis racket
40. bottle
41. wine glass
42. cup
43. fork
44. knife
45. spoon
46. bowl
47. banana
48. apple
49. sandwich
50. orange
51. broccoli
52. carrot
53. hot dog
54. pizza
55. donut
56. cake
57. chair
58. sofa
59. pottedplant
60. bed
61. diningtable
62. toilet
63. tvmonitor
64. laptop
65. mouse
66. remote
67. keyboard
68. cell phone
69. microwave
70. oven
71. toaster
72. sink
73. refrigerator
74. book
75. clock
76. vase
77. scissors
78. teddy bear
79. hair drier
80. toothbrush

Pelas categorias que o modelo consegue identificar rapidamente se percebe que sem treino este modelo não faria sentido para esta aplicação. Na Figura 49 estão os resultados para 3 imagens.

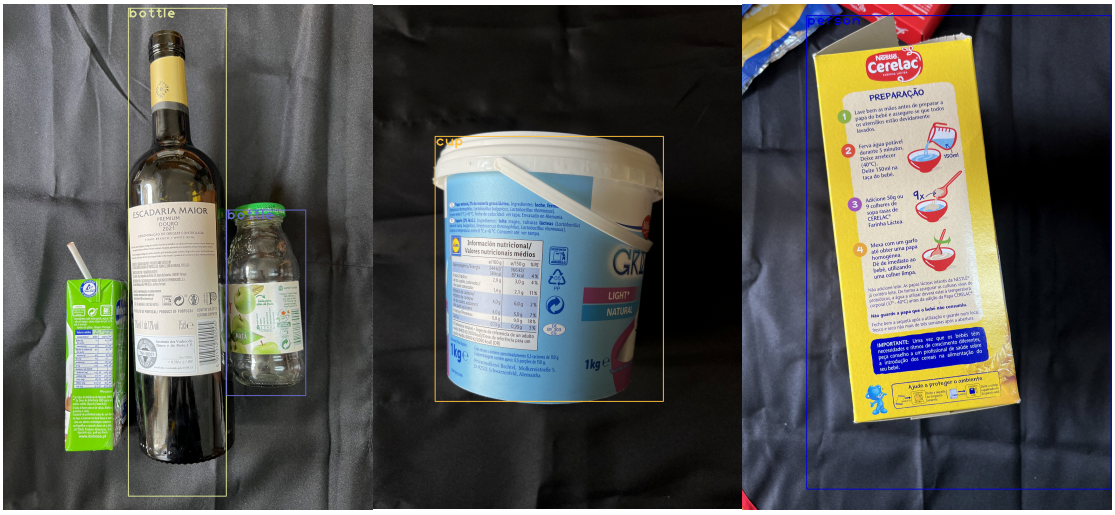


Figura 49 - Resultados modelo YOLO treino base

Como é comprovado pela Figura 49, apenas as garrafas têm um reconhecimento que faz sentido, a embalagem de iogurte, figura do meio, aparece como uma caneca e a embalagem na figura à esquerda aparece como uma pessoa. Isto acontece porque é o por defeito que o sistema reconhece. O modelo não foi previamente preparado para classificar diversos tipos de embalagens.

Assim sendo, apesar de se estar a utilizar o modelo YOLO, ele vai ter ser adaptado ao sistema de identificação de sinalética (nos RUs), ou seja, com imagens previamente selecionadas e classificadas para este objetivo concreto.

4.3.4 – Algoritmo Final de Deteção de Sinalética

O algoritmo final é um programa em Python que utiliza a biblioteca de imagem OpenCV, e dependendo da fonte da imagem (vídeo ou fotografia) categoriza os objetos recorrendo ao modelo treino para as 3 categorias (AZUL, VERDE, AMARELO), YOLO.

Para a criação do algoritmo com o modelo treinado é necessário treinar o modelo, para isso utilizou-se a plataforma Google Colab. A plataforma Colaboratory, ou "Colab" abreviado, é um produto da Google Research. O Colab permite que qualquer pessoa escreva e execute código python arbitrário através do navegador e é especialmente adequado para aprendizagem de máquinas, análise de dados e para educação. Mais tecnicamente, o Colab é um serviço que não

requer configuração para uso, oferecendo um acesso gratuito a recursos de computação, incluindo GPUs. [28]

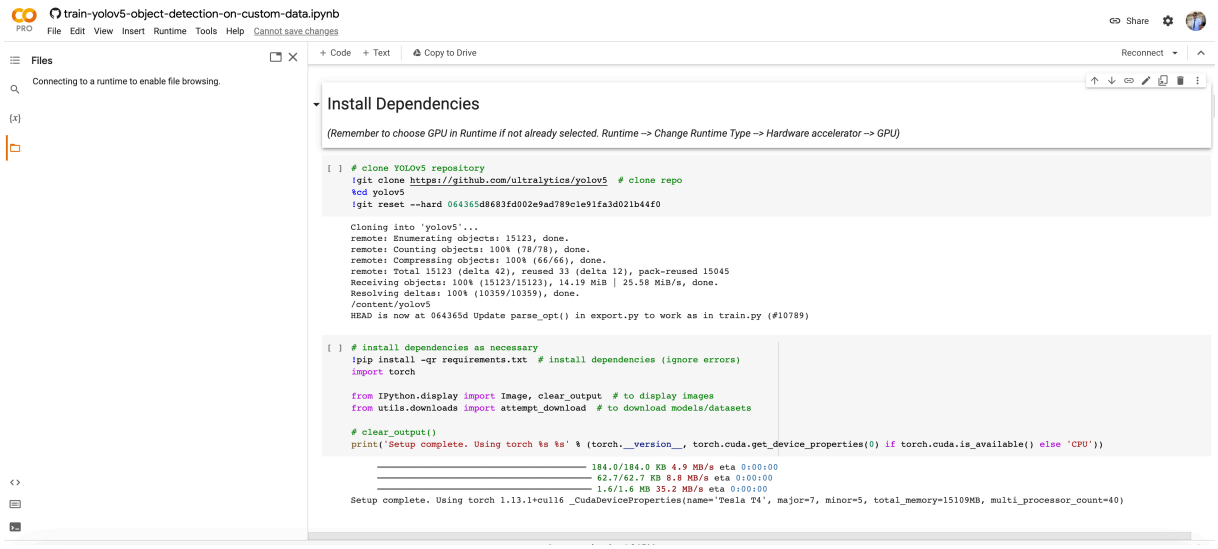


Figura 50 - Plataforma Google Colab

Esta plataforma (Figura 50) foi utilizada para treinar o modelo de Deteção de Sinalética. Para o treino do modelo é necessária uma grande capacidade computacional de processamento gráfico, esta plataforma permite que uma máquina virtual num servidor externo faça os cálculos em vez de ser uma máquina local a fazer, tendo o benefício de ser mais rápido e de custo gratuito. O código utilizado na máquina foi o python e a base da dados utilizada foi a treinada no roboflow, como indicado no capítulo Categorização das imagens. No Anexo 2 encontra-se o código utilizado nesta plataforma.

Treino Modelo YOLOv5 aplicado à Deteção de Sinalética

Para o treino do modelo foram escolhidas e classificadas diversas imagens. Assim a base de dados de fotografias existentes contempla 435 images de treino, 117 de validação e 64 imagens de teste. Foram treinados 3 modelos do YOLOv5, com valores diferentes de batch size e epoche.

Tabela 1 – Modelos Treinados

	Tamanho do Lote (batch size)	*Número épocas (epoche)*
Modelo 1	16	300
Modelo 2	16	500
Modelo 3	8	1000

O valor pré definido no Google Colab de batch size no algoritmo era de 16 e em 2 modelos foi o valor utilizado. Devido à capacidade computacional do Google Colab num terceiro teste tentou executar-se uma batch size de 8 com um epoche de 1000.

Na plataforma de treino é possível verificar como correu o treino do algoritmo, utilizando as imagens de treino do banco de dados, este teste tem como objetivo verificar se alguma das imagens do banco de testes, categorizadas no programa YOLO, não são categorizadas pelo modelo, este dado permite-nos ter uma aferição global do funcionamento do modelo.

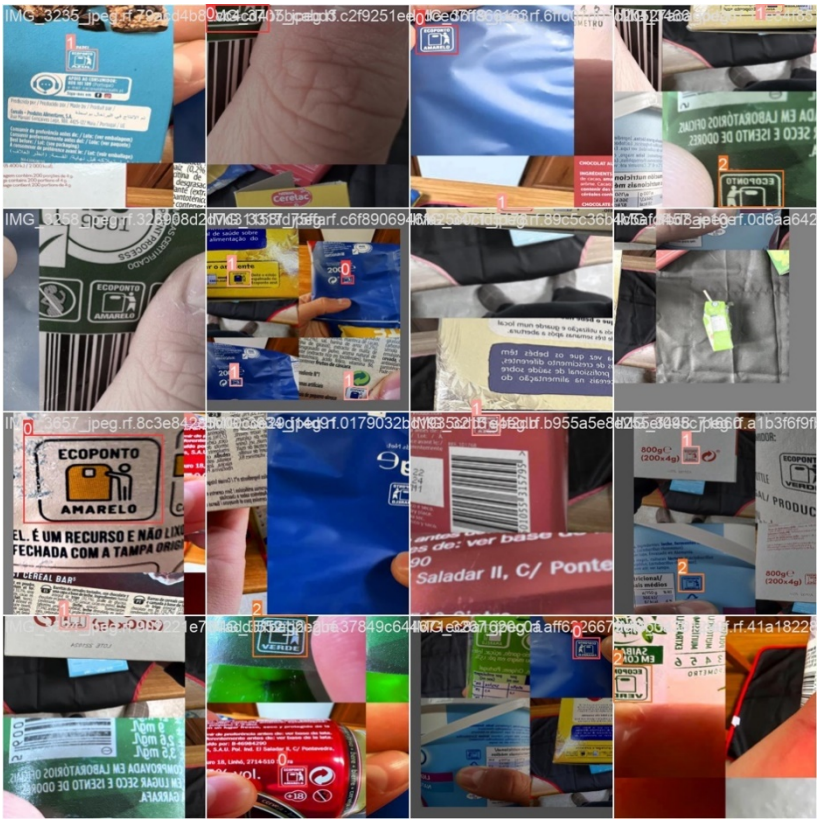


Figura 51 – Imagem de Treino do Algoritmo - Modelo 3

Na Figura 51 verificamos que o modelo conseguiu categorizar grande parte das imagens para o qual foi testado.

Assim se concluiu o capítulo do Sistema Desenvolvido no projeto da dissertação, sendo relevante a ligação deste capítulo com os capítulos do Estado da Arte e o da Ecologia Verde.

Demonstra-se a ideia do projeto e foca-se no algoritmo de reconhecimento e classificação das 3 categorias de produtos. Conhece-se a base do modelo utilizado no algoritmo e é explicada toda a fase de categorização de imagens e treino do modelo. Como foi devidamente explicado a fase de treino é de extrema importância para o correto funcionamento do sistema.

Para comprovar o sistema desenvolvido apresentam-se, no próximo capítulo, os resultados obtidos.

Capítulo 5

RESULTADOS

Neste capítulo são apresentados os resultados obtidos com cada um dos modelos.

5.1 – Interpretar os resultados de classificação

Para determinar e comparar o desempenho previsivo de diferentes modelos de deteção de objetos, são necessárias métricas quantitativas padronizadas. De seguida serão explicadas algumas destas métricas: IoU, Accuracy, Precision, Recall, AP e mAP.

Para entender estes parâmetros é necessário perceber a matriz confusão, esta é utilizada para verificar o desempenho de um algoritmo de classificação (Figura 52):

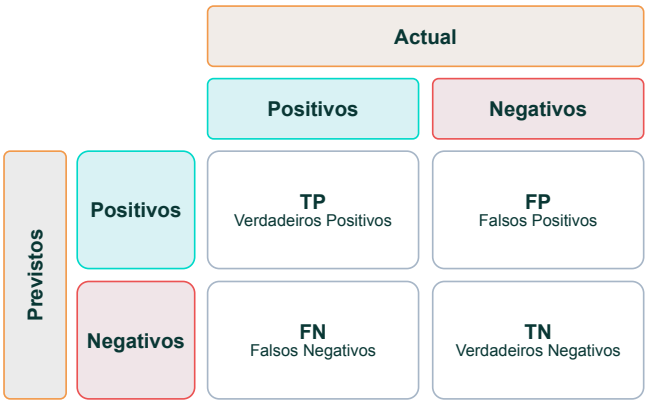


Figura 52 - Matriz Confusão

Esta matriz tem quatro valores possíveis:

- TP – Verdadeiros Positivos (O modelo previu uma categoria e acertou conforme as categorias existentes)
- TN – Verdadeiros Negativos (O modelo não prevê uma categoria e não faz parte das categorias existentes)
- FP – Falso Positivo (O modelo prevê uma categoria, mas não faz parte das categorias existentes)
- FN – Falsos Negativos (O modelo não prevê uma categoria, mas faz parte das categorias existentes)

IoU – Intersecção sobre União

A intersecção sobre união é uma métrica popular para medir a precisão de localização e calcular erros de localização em modelos de deteção de objetos.

Para calcular o IoU entre as caixas delimitadoras previstas e as caixas delimitadoras de referência (ground truth, categorias), primeiro é preciso determinar a área de interseção entre as duas caixas delimitadoras correspondentes para o mesmo objeto. Em seguida, é calculada a área total coberta pelas duas caixas delimitadoras, também conhecida como união, e a área de sobreposição entre elas, chamada de interseção.

A interseção dividida pela união dá a proporção de sobreposição em relação à área total, fornecendo uma estimativa de quão próxima a caixa delimitadora prevista está da caixa delimitadora original (Figura 53).

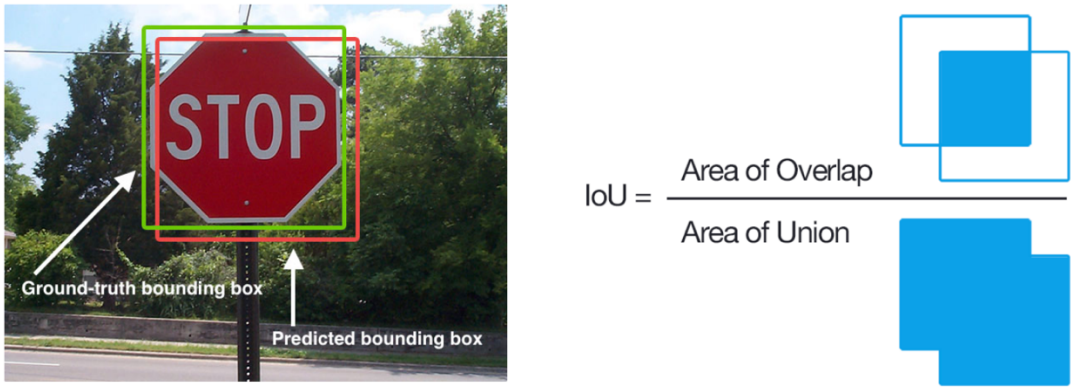


Figura 53 – Interseção sobre União (IoU) [37]

O valor de IoU é utilizado para definir os valores de Precision, Recall e AP. Normalmente é considerado um valor de IoU de 0,5 na classificação se a previsão é um verdadeiro positivo (TP) ou um falso positivo (FP).

O exemplo da Figura 54 mostra que se o limiar de IoU for 0,5 e o valor de IoU é 0,7 a classificação da previsão é verdadeiro positivo (TP). Por outro lado, se o IoU for 0,3, classificamos como falso positivo (FP).

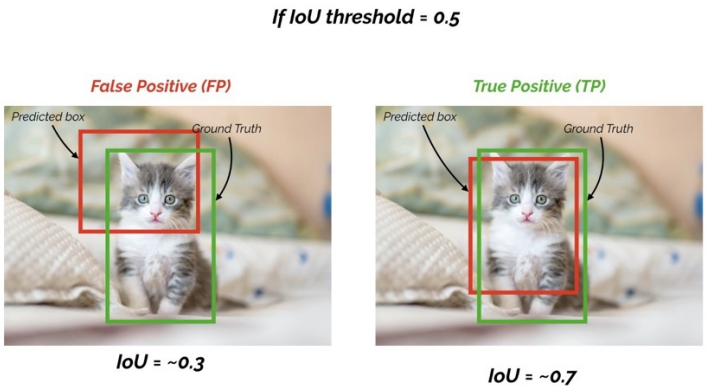


Figura 54 - Limiar de IoU [38]

Accuracy

É a métrica de avaliação mais usada para avaliar o resultado prático dos modelos, medida entre 0 e 1. Esta medida informa quantas vezes o modelo previu corretamente em relação ao número total de vezes que o modelo foi usado para fazer previsões, contudo não dá informações específicas em relação às categorias. Faz sentido usar esta métrica como indicador apenas quando o conjunto de dados é balanceado (todas as classes têm o mesmo número de amostras).

$$Accuracy = \frac{TP + TN}{TP + FP + FN + FP} = \frac{Nº\ Previsões\ Correctas}{Tamanho\ da\ Base\ de\ Dados} \qquad \text{(eq. 16)}$$

P – Precision

É uma medida entre 0 e 1 que indica o quanto podemos confiar nas previsões positivas do modelo, ou seja, indica a percentagem de previsões que estão corretas. O valor da precision altera com o valor do limiar de confiança (IoU).

$$Precision = \frac{TP}{TP + FP} \qquad \text{(eq. 17)}$$

R – Recall

É uma medida entre 0 e 1 que indica quão bem são encontradas as previsões positivas do modelo, ou seja, indica a percentagem de previsões corretas em todo o grupo de previsões. O valor da precision altera com o valor do limiar de confiança (IoU).

$$Recall = \frac{TP}{TP + FN} \quad \text{(eq. 18)}$$

AP – Average Precision

É uma medida entre 0 e 1, que indica a previsão média de um modelo, este valor não é calculado fazendo a média do Precision, mas sim, fazendo a área entre os dados do Precision e do Recall para cada elemento (Figura 55).

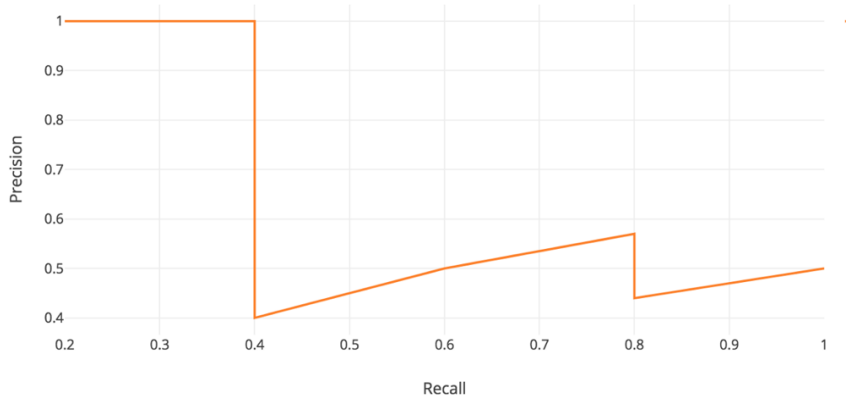


Figura 55 - Exemplo de Gráfico Precision vs Recall

$$AP = \int_0^1 p(r)dr \quad \text{(eq. 19)}$$

Dado que os valores de Precisão e Recall estão compreendidos entre 0 e 1 o valor de AP também estará entre 0 e 1.

mAP – mean Average Precision

É calculada encontrando a Precisão Média (AP) para cada classe e, em seguida, calculando a média sobre um número de classes.

A fórmula da média de precisão média (mAP) resulta da curva entre a precision e o recall e considera tanto falsos positivos (FP) quanto falsos negativos (FN). Esta propriedade torna a mAP uma métrica adequada para avaliar aplicações de deteção.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \qquad \text{(eq. 20)}$$

Visto que o limiar IoU altera os valores da matriz de Confusão e por consequências os valores de precision e recall, o valor de mAP será alterado com a alteração deste limiar. No treino do modelo foram considerados mAP50 e mAP50-95, os valores correspondem ao valor do limiar do IoU.

O valor de mAP50 foi calculado utilizando um IoU de 0,5, enquanto o mAP50-95 é calculado fazendo a média dos mAP calculados com IoU de 0,50, 0,55, 0,60, …, 0.95. Calculando o mAP desta segunda forma remove-se a ambiguidade de escolher um IoU ótimo para avaliar a performance de um modelo.

5.2 – Resultados Experimentais Obtidos com o Modelo 1

```
300 epochs completed in 0.409 hours.
Optimizer stripped from runs/train/yolov5s_results3/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results3/weights/best.pt, 14.8MB

Validating runs/train/yolov5s_results3/weights/best.pt...
Fusing layers...
custom_YOLOv5s summary: 182 layers, 7251912 parameters, 0 gradients
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 4/4 [00:01<00:00,  2.19it/s]
                   all        101        115      0.931      0.857      0.877      0.688
               AMARELO        101         43          1      0.883      0.904      0.738
                  AZUL        101         24      0.793      0.833      0.869      0.617
                 VERDE        101         48      0.999      0.854      0.859      0.708
Results saved to runs/train/yolov5s_results3
CPU times: user 18.2 s, sys: 1.99 s, total: 20.2 s
Wall time: 25min 5s
```

Figura 56- Modelo 1 – 16 bach size e 300 epoche

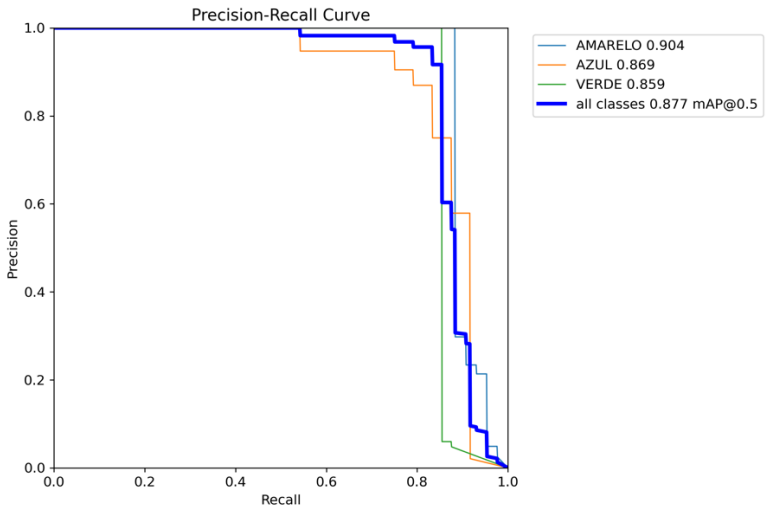


Figura 57 - Modelo 1 – 16 bach size e 300 epoche - Precision-Recall Curve

Para o treino do modelo 1 foram solicitados 300 epochs e o modelo concluiu o processamento dos mesmos em 25 minutos e 5 segundos. apresentado um valor geral (all) de Precision de 0,931, de Recall 0,857 e de mAP50 de 0,877, sendo estes valores perto da unidade, os dados à partida são positivos. Na amostra de treino o modelo conseguiu 87% de correspondências positivas. É de notar que o valor do Precision AMARELO é de um, indicando que todas as previsões feitas por este modelo para as imagens com a categoria AMARELO estavam corretas. Este valor faz aumentar o valor de mAP50 para a categoria AMARELA. É possível assim afirmar que a característica que o modelo melhor deteta é o AMARELO, sendo bastante percetível no gráfico, Figura 57 , representado com a cor azul-claro.

5.3 – Resultados Experimentais Obtidos com o Modelo 2

```
456 epochs completed in 0.631 hours.
Optimizer stripped from runs/train/yolov5s_results2/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results2/weights/best.pt, 14.8MB

Validating runs/train/yolov5s_results2/weights/best.pt...
Fusing layers...
custom_YOLOv5s summary: 182 layers, 7251912 parameters, 0 gradients
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 4/4 [00:01<00:00,  2.22it/s]
                   all        101        115      0.952      0.834      0.893        0.7
               AMARELO        101         43      0.974      0.856       0.93      0.732
                  AZUL        101         24      0.886      0.792       0.89      0.672
                 VERDE        101         48      0.998      0.854      0.859      0.697
Results saved to runs/train/yolov5s_results2
CPU times: user 28 s, sys: 3.09 s, total: 31.1 s
Wall time: 38min 23s
```

Figura 58 - Modelo 2 - 16 bach size e 500 epoche

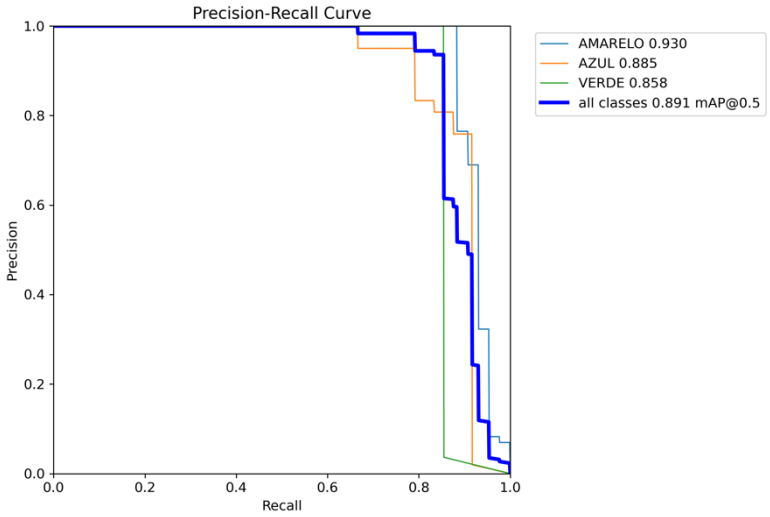


Figura 59 - Modelo 2 - 16 bach size e 500 epoche - Precision-Recall Curve

O modelo 2 apresentou um tempo de computação ligeiramente superior ao do modelo 1, explicado por ter de fazer mais 200 epochs de processamento. Foi pedido ao modelo este valor de epochs contudo o algoritmo de treino apercebeu-se que o melhor resultado possível de mAP50 ocorreu na epoch 355 e guardou esse modelo como o modelo final, fazendo com que na realidade o número de epoch realizado por este modelo sejam 355 e não os 500. Neste modelo verificou-se que o valor de Precision mais alto por categoria foi o VERDE, contudo não obteve o melhor valor de mAP50, tendo este valor sido atribuído ao AMARELO. Isto deve-se ao facto do valor Recall da categoria AMARELA ser superior. É de notar que a categoria AZUL tem os valores bastante inferiores em todos os parâmetros, esta ocorrência deve-se ao facto de o número de imagens de treino da categoria AZUL serem cerca de metade das outras

2 categorias. Como no modelo anterior a característica que o modelo melhor deteta é o AMARELO, sendo bastante percetível no gráfico, Figura 59 , representado com a cor azul-claro.

5.4 – Resultados Experimentais Obtidos com o Modelo 3

```
572 epochs completed in 1.171 hours.
Optimizer stripped from runs/train/yolov5s_results3/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results3/weights/best.pt, 14.8MB

Validating runs/train/yolov5s_results3/weights/best.pt...
Fusing layers...
custom_YOLOv5s summary: 182 layers, 7251912 parameters, 0 gradients
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 7/7 [00:02<00:00,  3.12it/s]
                   all        101        115      0.922       0.83      0.901      0.721
               AMARELO        101         43      0.911      0.907       0.96      0.744
                  AZUL        101         24      0.855      0.792      0.874      0.682
                 VERDE        101         48          1       0.79       0.87      0.737
Results saved to runs/train/yolov5s_results3
CPU times: user 58.3 s, sys: 6.48 s, total: 1min 4s
Wall time: 1h 10min 51s
```

Figura 60 - Modelo 3 - 8 bach size e 1000 epoche

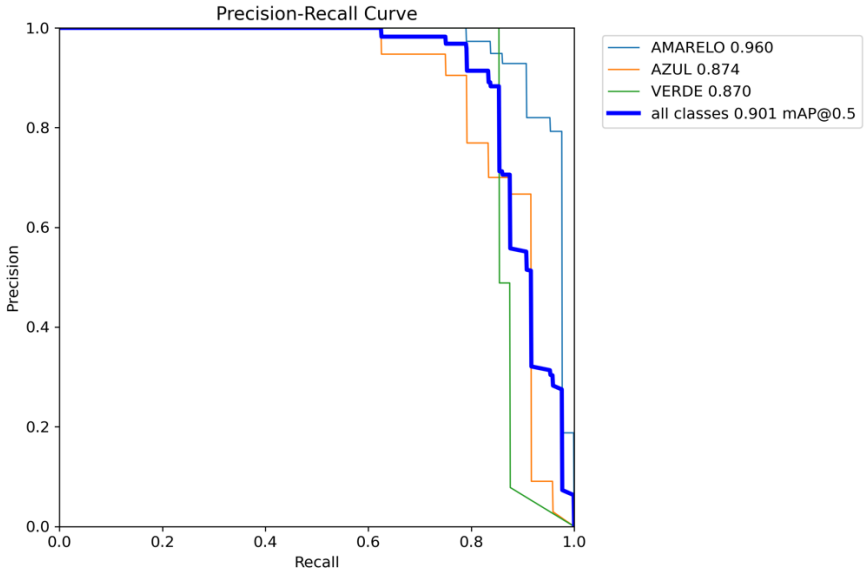


Figura 61 - Modelo 3 - 8 bach size e 1000 epoche Precision-Recall Curve

Com 1000 epochs programadas com um batch size de 8, o número de cálculos necessários aumentou, diminuindo o número de imagens a serem processadas por passagem e aumentando a quantidade de passagens, este modelo, mesmo sem verificar os resultados, espera-se ser mais preciso. Da mesma forma que o modelo 2, o algoritmo de processamento entendeu que a melhor

epoch se deu no valor 471. O volume de cálculos a fazer a rotina de treino demorou 1 hora e 10 min, sensivelmente o dobro do modelo 2, dado que o tempo necessário por cada um dos epochs é o dobro (58.3s). Seguindo a tendência anterior a categoria AMARELA teve o melhor mAP50. A categoria VERDE obteve o melhor valor de Precision, mas devido ao valor de Recall não obteve o melhor valor de mAP50, o algoritmo quando indica que um objeto na imagem pertence à categoria VERDE acerta 100% das vezes, porém só 79% das vezes é que reconhece os objetos com a categoria VERDE na imagem, fazendo com que a categoria VERDE seja a pior a ser analisada pelo modelo. É interessante perceber isto no gráfico, percebe-se que o valor de mAP50 é prejudicado pela categoria VERDE.

5.5 – Comentário aos Resultados

Nesta tabela resumo, Tabela 2, é possível verificar que o modelo 2 é o que tem o valor de Precision mais elevado (0,952), é o modelo que tem o segundo melhor valor de Recall (0,834), contudo não é o modelo com o melhor valor de mAP50 (o fator decisivo na escolha do modelo mais elevado).

O modelo 1 tem o melhor valor de Recall (0,857) e o segundo melhor valor de Precision (0,931), contudo, da mesma forma que o modelo 2, não é o modelo com o melhor valor de mAP50. O melhor valor de mAP50 encontra-se no modelo 3, apesar de ter o pior valor de Precision e Recall.

Tabela 2 - Tabela Resumo Modelos Gerados

	Precision	*Recall*	*mAP50*	*mAP50-95*
Modelo 1	0,931	**0,857**	0,877	0,688
Modelo 2	**0,952**	0,834	0,893	0,700
Modelo 3	0,922	0,83	**0,901**	**0,721**

Os valores gerais são calculados com os valores das médias de cada categoria. Fazendo com o que o modelo 3 tenha o melhor valor de mAP50 e consequentemente mAP50-95. Isto indica que o modelo 3, não é o melhor modelo a acertar que um objeto pertence a uma certa categoria (melhor foi modelo 2), nem a verificar todos os objetos de uma imagem (melhor foi modelo 1), mas numa performance geral o modelo acerta 90,1% das vezes.

Estes valores continuariam a subir se o número de epochs, batch size e imagens da base de dados aumentassem, contribuindo para um melhor modelo.

5.6 – Casos Práticos

Nesta secção serão demonstrados a aplicação dos modelos com recurso ao algoritmo criado para testar imagens estes modelos, este algoritmo escrito na linguagem python, e com recurso à biblioteca opencv consta do Anexo 3a e Anexo 3b deste documento.

Tabela 3 - Utilização dos Modelos Gerados Para Categorizar RUs

Id	*Modelo 1*	*Modelo 2*	*Modelo 3*
1			
2			

3	

A tabela 3 tem como objetivo fazer uma comparação das 3 categorias de deteção, estas imagens apenas contêm um objeto presente. Os valores apresentados nas figuras, a seguir à categoria (por exemplo "AZUL 0,63") correspondem ao valor de certeza (accuracy) que o modelo tem na afirmação da categoria.

Na linha com id1 da Tabela 3, o modelo que detetou com mais certeza a característica AZUL foi o modelo 1, interessante visto que o modelo com o melhor valor de mAP50 para esta característica é o modelo 2, por mais 2,1% em relação ao modelo 1.

Na linha com id2 da Tabela 3, o modelo que detetou com mais certeza a característica AMARELA foi o modelo 3 com uma diferença de certeza de 19% em relação ao segundo lugar, o modelo 2.

Na linha com id3 da Tabela 3, o modelo que detetou com mais certeza a característica VERDE foi o modelo 2. O modelo 1 não foi capaz de detetar nenhuma característica, e o modelo 3, o modelo que apresentou o valor de Precison 1 e o melhor valor de mAP50, detetou 2 objetos com característica VERDE, sendo o resultado inválido também.

Tabela 4 - Utilização dos Modelos Gerados Para Categorizar RUs Continuação

Id	*Modelo 1*	*Modelo 2*	*Modelo 3*
1			
2			
3			

4			

Numa segunda fase de testes foram utilizadas imagens que à partida dificultariam os testes, recorrendo a imagens com muitos objetos e/ou iluminadas de forma deficiente.

A linha com id1da Tabela 4, demonstra a capacidade do modelo 3, este que apresenta maior valor de mAP50 neste caso detetou 2 objetos, enquanto os outros modelos apenas um. O objeto detetado pela totalidade dos modelos, é apresentado com mais certeza no modelo 3 também.

Na linha com id2 da Tabela 4, todos os modelos acertam num objeto com a categoria AMARELA, com uma taxa de confiança na ordem dos 40%, contudo falham a acertar o objeto presente na garrafa, isto pode dever-se ao facto de o pacote azul estar focado e a garrafa não.

Na linha com id3 da Tabela 4, está presente um teste complicado para todos os grupos, múltiplos objetos presentes, mal iluminados, com transladações e alguns desfocados. Neste teste o modelo 3 teve bastantes dificuldades ficando apresentando o pior resultado do grupo. A imagem presente neste teste foi utilizada como imagem de base de dados, tem o objetivo de mostrar que com menos epoches para uma imagem semelhante é mais fácil obter bons resultados.

A linha com o id4 da Tabela 4, têm o objetivo de mostrar que nenhum dos modelos se comporta nas melhores condições quando a imagem está desfocada e mal iluminada.

Neste capítulo estão presentes os resultados obtidos do treino dos modelos. Foi feita uma análise a cada modelo seguida de uma comparação dos três modelos. De seguida mostrou-se o comportamento dos modelos numa situação que se assemelha à real. Permitindo aferir a qualidade de cada modelo.

Capítulo 6

CONCLUSÕES E TRABALHO FUTURO

Este capítulo é o culminar do trabalho de pesquisa e desenvolvimento que foi realizado nesta dissertação. Aqui são apresentadas as principais conclusões sobre os resultados obtidos, é feito um balanço das limitações da metodologia adotada e são avaliadas possíveis perspetivas de desenvolvimento futuro do trabalho realizado.

6.1 – Análise de resultados

O primeiro objetivo deste projeto era desenvolver os conhecimentos na linguagem de programação Python e aplicá-los a um processo. No processo de investigação e desenvolvimento, o treino neste tipo de linguagem foi imprescindível, foi necessário fazer alguns exercícios de programação para poder conhecer a linguagem e desenvolver um algoritmo capaz de realizar o pretendido, utilizando a linguagem proposta, o Python. Este foi desenvolvido com o auxílio da biblioteca OpenCV de tratamento de imagem e utilizando um modelo de reconhecimento de imagem o YOLO.

A abordagem de reconhecimento de resíduos sólidos teve como princípio a deteção de símbolos iconográficos desenhados pela Sociedade Ponto Verde. Visto que foi tomada esta abordagem o algoritmo não consegue detetar resíduos que não tenham estes símbolos ou não os tenham visíveis. O algoritmo também tem dificuldade com a captação da imagem se as condições ideais não existirem. Nestas situações o grau de certeza diminui ou torna-se existente, condições como iluminação, focagem e contraste na imagem são pontos bastante importantes.
É interessante mencionar que há objetos que trazem símbolos indicativos de reciclagem, mas os mesmos não cumprem as normas, o que faz com que por base não sejam detetados pelo algoritmo.
Foram apresentados três modelos diferentes. Estes foram treinados com a mesma base de dados (616 imagens: 294 – categoria AMARELO, 239 – categoria AZUL, 220 – categoria VERDE), fazendo apenas alterações aos parâmetros de treino. O modelo 3 destacou-se pela sua performance em relação aos outros, contudo com uma base de dados maior conseguiria atingir melhores resultados, e categorizar mais tipos de objetos. Para efeitos de comparação de valores utilizou-se sempre o valor limiar de 0,5.

O terceiro objetivo deste projeto consistia no desenvolvimento de um sistema físico com equipamentos que permitiriam a utilização do algoritmo criado para o objetivo 2, este teve a sua topologia pensada e desenhada, contudo não chegou a ser posto em prática, ficando apenas como modelo de um possível sistema de implementação.

O algoritmo desenvolvido tem um valor acrescido quando outras aplicações e cenários são pensados. Com recurso a uma biblioteca de texto para voz o algoritmo poderia ser adaptado e poder ajudar audiovisuais a separar o seu lixo, ou a crianças, no ciclo básico, que estejam a aprender a reciclar utilizando um jogo.

Numa outra perspetiva, este trabalho, também tentou sensibilizar o leitor para o problema ambiente e dos resíduos urbanos portugueses, podendo levar a uma mudança de hábitos.

6.2 – Trabalho futuro

Embora alguns objetivos deste trabalho não tenham sido completamente cumpridos, a solução apresentada é extremamente promissora. O algoritmo desenvolvido tem um grande potencial para permitir novas categorizações e aumentar a precisão, especialmente se a base de dados for atualizada e alimentada regularmente.

Uma das possibilidades para trabalho futuro seria a implementação prática do terceiro objetivo, que envolve o desenvolvimento de um sistema físico com recurso aos equipamentos propostos no protótipo inicial. Esse sistema seria particularmente útil na indústria de tratamento de resíduos, pois aumentaria significativamente a capacidade de categorização e separação de resíduos urbanos para posterior reciclagem.

Um outro investimento de interesse, seria desenvolver uma aplicação para dispositivos móveis com o objetivo de estender a categorização e a separação seletiva de resíduos urbanos para mais pessoas. Essa aplicação, especialmente direcionada para crianças em meio escolar, pessoas com limitação visual e outras futuros utilizadores, seria uma forma eficaz de aumentar a consciência ambiental e incentivar mais pessoas a adotar práticas de reciclagem.

BIBLIOGRAFIA

[1] P. Francisco, Laudato si, 2015.

[2] A. P. d. Ambiente, "Relatório Anual Resíduos Urbanos 2021," 2022.

[3] *Portaria n.º 851/2009 do Ministério do Ambiente, do Ordenamento do Território e do Desenvolvimento Regional.*

[4] A. p. d. ambiente, "Embalagens e Resíduos de Embalagens," 2021. [Online]. Available: https://apambiente.pt/residuos/embalagens-e-residuos-de-embalagens.

[5] S. Pontoverde, "Símbolos e Ícones," 2020. [Online]. Available: https://www.pontoverde.pt/1_4_simbolos_e_icones.php.

[6] A. A. D. Medeiros, Modelagem e Análise de Sistemas Dinâmicos, Apostila, 2003.

[7] F. S. Lima, A automação e a sua evolução, Brasil: Universidade Federal do Rio Grande do Norte, 2003.

[8] J. Jasperneite, T. Sauter e M. Wollschlaeger, The Future of Industrial Communication, IEEE Industrial Electronics Magazine, 2017.

[9] ISO. [Online]. Available: https://www.iso.org/ics/35.100/x/.

[10] marcelocreppe, "marcelocreppe.com.br," [Online]. Available: https://marcelocreppe.com.br/aprenda-como-funciona-a-capacidade-da-visao/.

[11] C. Couto, Introdução à Robótica Industrial, Universidade Aberta, 2000.

[12] K. S. Fu, R. C. Gonzalez e C. S. G. Lee, ROBOTICS - Control, Sensing, Vision and Intelligence, McGraw-Hill Book Co, 1987.

[13] "Light," [Online]. Available: https://en.wikipedia.org/wiki/Light.

[14] A. Vision, "CCD or CMOS: can CMOS sensors replace CCDs in all cases?".

[15] J. N. Burghart, H.-G. Graf e C. Hare, HDR CMOS Imagers and Their Applications, 2006.

[16] K. &. Y. Erdoğan, "Shifting Colors to Overcome not Realizing Objects Problem due to Color Vision Deficiency.".

[17] J. Rodrigues, Algoritmo de seguimento de complexidade variável, Porto: FEUP, 2012.

[18] B. Kalyani, B. Poornima, Y. Ramadevi e T. Sridevi, Segmentation and object recognition using edge detection techniques, International Journal of Computer Science & Information Technology (IJCSIT), 2010.

[19] A. Kaur, B. Kaur e G. Kaur, Detection and Classification of Printed Circuit Board Defects Using Image Subtraction Method, Patiala: IEEE, 2014.

[20] C. V. R. Coutinho, Robótica Móvel - Sistema de Condução Autónoma, Instituto Superior de Engenharia de Lisboa, 2014.

[21] Y. Sakai, T. Oda, M. Ikeda e L. Barolli, An Object Tracking System Based on SIFT and SURF Feature Extraction Methods, International Conference on Network-Based Information Systems, 2015.

[22] H. Bay, T. Tuytelaars, L. Van Gool, A. Leonardis, H. Bischof e A. Pinz, SURF: Speeded Up Robust Features, 2006.

[23] H. Bay, L. Van Gool e T. Tuytelaars, SURF: Speeded Up Robust Features, Lecture Notes in Computer Science, 2006.

[24] Python, "The Python Tutorial," 2023. [Online]. Available: https://docs.python.org/3/tutorial/.

[25] F. Pérez, B. E. Granger e J. D. Hunter, Python: An Ecosystem for Scientific Computing, IEEE, 2010.

[26] D. E. R. Gaspar, Raspberry Pi: a Smart Video Monitoring Platform, Lisboa: Instituto Superior Técnico, 2014.

[27] J. &. H. Demčák, "SMART Identification by Vision System.".

[28] J. Redmon, S. Divvala, R. Girshick e A. Farhadi, You Only Look Once: Unified, Real-Time Object Detection, IEEE, 2016.

[29] M. Chablani, "towardsdatascience.com," [Online]. Available: https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006.

[30] V7Labs, "YOLO: Algorithm for Object Detection Explained [+Examples]," 03 fevereiro 2023. [Online]. Available: https://www.v7labs.com/blog/yolo-object-detection.

[31] F. Belharar, "medium.com," [Online]. Available: https://medium.com/@fatimazahra.belharar/object-detection-using-cnn-an-introduction-to-the-yolo-algorithm-df0f7b173c6.

[32] E. Alecrim, *Redes Neurais Artificiais,* 2004.

[33] J. Brownlee, "Difference Between a Batch and an Epoch in a Neural Network," 10 agosto 2022. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

[34] ultralytics, "github.com," [Online]. Available: https://github.com/ultralytics/ultralytics.

[35] J. Faudi, "medium.com," [Online]. Available: https://medium.com/artificialis/detecting-aircrafts-on-airbus-pleiades-imagery-with-yolov5-5f3d464b75ad.

[36] Google, "Colaboratory," [Online]. Available: https://research.google.com/colaboratory/faq.html.

[37] J. Agrawal, "medium.com," [Online]. Available: https://medium.com/@jalajagr/mean-average-precision-map-explained-in-object-detection-fb61adf67ef4.

[38] D. Shah, "v7labs.com," [Online]. Available: https://www.v7labs.com/blog/mean-average-precision.

[39] S. Bennett, A Brief History of Automatic Control, IEEE Control Systems Magazine, 1996.

[40] M. Calonder, V. Lepetit, C. Strecha e P. Fua, BRIEF: Binary robust independent elementary features, In European Conference on Computer Vision, 2010.

[41] E. Karami, S. Prasad e S. M., Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images., 2017.

[42] J. Redmon e A. Farhadi, You only look once: Unified, real-time object detection, IEEE, 2015.

[43] J. Redmon e A. Farhadi, You only look once: Unified, real-time object detection, IEEE, 2015.

[44] J. M. Samtos, *Redes Neuronais Conceitos,* Porto.

[45] O. Kivrak e M. Z. Gürbüz, “Performance Comparison of YOLOv3, YOLOv4 and YOLOv5 Algorithms: A Case Study for Poultry Recognition,” 2022.

Anexo 1a – Algoritmo de Teste - Utilizando Imagens Carregadas

```
import cv2 as cv
import numpy as np
#from wandb import Classes

# load yolo
net = cv.dnn.readNet("yolov3.weights",
                     "yolov3.cfg")
classes = []
with open("coco.names", 'r') as f:
    classes = [line.strip() for line in f.readlines()]
# print(classes)
layer_name = net.getLayerNames()
output_layer = [layer_name[i - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))

# Load Image
img = cv.imread("TestV5 Real/imagesTeste/image17.jpeg")
img = cv.resize(img, None, fx=0.4, fy=0.4)
height, width, channel = img.shape

# Detect Objects
blob = cv.dnn.blobFromImage(
    img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layer)
# print(outs)

# Showing Information on the screen
class_ids = []
confidences = []
boxes = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            # Object detection
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            # cv.circle(img, (center_x, center_y), 10, (0, 255, 0), 2 )
            # Reactangle Cordinate
            x = int(center_x - w/2)
            y = int(center_y - h/2)
            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

# print(len(boxes))
# number_object_detection = len(boxes)

indexes = cv.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
print(indexes)
# TODO trabalho de casa para o menino Joao
```

```
font = cv.FONT_HERSHEY_PLAIN
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        # print(label)
        color = colors[i]
        cv.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv.putText(img, label, (x, y + 30), font, 3, color, 3)

cv.imshow("IMG", img)
cv.waitKey(0)
cv.destroyAllWindows()
```

ANEXO 1B – ALGORITMO DE TESTE - UTILIZANDO CÂMARA EM TEMPO REAL

```
import cv2
import sys
import numpy as np
#from wandb import Classes

classes = []
with open("coco.names", 'r') as f:
    classes = [line.strip() for line in f.readlines()]

s = 0
if len(sys.argv) > 1:
    s = sys.argv[1]

source = cv2.VideoCapture(s)

win_name = 'Camera Preview'
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)

net = cv2.dnn.readNet("yolov3.weights",
                      "yolov3.cfg")
layer_name = net.getLayerNames()
output_layer = [layer_name[i - 1] for i in net.getUnconnectedOutLayers()]

# Model parameters
in_width = 300
in_height = 300
mean = [104, 117, 123]
conf_threshold = 0.7

while cv2.waitKey(1) != 27:
    has_frame, frame = source.read()
    if not has_frame:
        break
    frame = cv2.flip(frame,1)
    frame_height = frame.shape[0]
    frame_width = frame.shape[1]

    # Create a 4D blob from a frame.
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (160, 160), (0, 0, 0),
swapRB=True, crop=False)
    # Run a model
    net.setInput(blob)
    outs = net.forward(output_layer)

    # Showing Information on the screen
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Object detection
```

```
                center_x = int(detection[0] * frame_width)
                center_y = int(detection[1] * frame_height)
                w = int(detection[2] * frame_width)
                h = int(detection[3] * frame_height)
                # cv.circle(img, (center_x, center_y), 10, (0, 255, 0), 2 )
                # Reactangle Cordinate
                x = int(center_x - w/2)
                y = int(center_y - h/2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            if (len(confidences) > 1):
                label = label + " %.2f" % confidences[1]
            # print(label)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 1)

            label_size, base_line = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 1, 3)
            x_left_bottom = x
            y_left_bottom = y - 10
            x_right_top = x + w
            y_right_top = y + h - 10
            cv2.rectangle(frame, (x_left_bottom, y_left_bottom - label_size[1]),
                                 (x_left_bottom + label_size[0], y_left_bottom +
base_line),
                                 (171, 71, 188), cv2.FILLED)

            cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 255, 255), 3)

    t, _ = net.getPerfProfile()
    label = 'Inference time: %.2f ms' % (t * 1000.0 / cv2.getTickFrequency())
    cv2.putText(frame, label, (0, 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0))
    cv2.imshow(win_name, frame)

source.release()
cv2.destroyWindow(win_name)
```

ANEXO 2 – ALGORITMO DE TREINO – PLATAFORMA GOOGLE COLABS

```
# clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
!git reset --hard 064365d8683fd002e9ad789c1e91fa3d021b44f0


# install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (ignore errors)
import torch

from IPython.display import Image, clear_output  # to display images
from utils.downloads import attempt_download  # to download models/datasets

# clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__,
torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))


#follow the link below to get your download code from from Roboflow
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="buIYqwI3RhhxrIXb1bDX")
project = rf.workspace("lixo").project("lixo")
dataset = project.version(3).download("yolov5")


%cd /content/yolov5
#after following the link above, recieve python code with these fields filled in
#from roboflow import Roboflow
#rf = Roboflow(api_key="YOUR API KEY HERE")
#project = rf.workspace().project("YOUR PROJECT")
#dataset = project.version("YOUR VERSION").download("yolov5")


# this is the YAML file Roboflow wrote for us that we're loading into this notebook
with our data
%cat LIXO-3/data.yaml


# define number of classes based on YAML
import yaml
print(dataset.location)
with open(dataset.location + "/data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])


#this is the model configuration we will use for our tutorial
%cat /content/yolov5/models/yolov5s.yaml


#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
```

```
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))


%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes}  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]


# train yolov5s on custom data for 100 epochs
# time its performance
%%time
```

```
%cd /content/yolov5/
!python train.py --img 416 --batch 8 --epochs 1000 --data
{dataset.location}/data.yaml --cfg ./models/custom_yolov5s.yaml --weights '' --name
yolov5s_results  --cache


# when we ran this, we saw .007 second inference time. That is 140 FPS on a TESLA
P100!
# use the best weights!
%cd /content/yolov5/
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --
conf 0.4 --source /content/yolov5/LIXO-3/test/images
```

ANEXO 3A – ALGORITMO DE FINAL - UTILIZANDO IMAGENS CARREGADAS

```
import cv2
import torch
from PIL import Image
# import yolov5
from os import listdir
from os.path import isfile, join
import numpy


'''
# load pretrained model
model = yolov5.load('yolov5s.pt')

# or load custom model
model = yolov5.load('train/best.pt')
'''

# Model
#model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
model = torch.hub.load('ultralytics/yolov5', 'custom', path='TestV5
Real/best_1000_v2_8b.pt')  # local model


# # Images
# im1 = Image.open(TESTE.jpg')  # PIL image
#im1 = cv2.imread('TestV5 Real/image.jpeg')[..., ::-1]  # OpenCV image (BGR to RGB)

mypath='TestV5 Real/imagesTeste'
onlyfiles = [ f for f in listdir(mypath) if isfile(join(mypath,f)) ]
print(onlyfiles)
#images = numpy.empty(len(onlyfiles), dtype=object)
images = []
for n in range(0, len(onlyfiles)):
  images.append(cv2.imread(join(mypath,onlyfiles[n]))[..., ::-1])

# Inference
results = model(images, size=640) # batch of images

# Results
results.print()
results.save()  # or .show()

results.xyxy[0]  # im1 predictions (tensor)
results.pandas().xyxy[0]  # im1 predictions (pandas)
```

ANEXO 3B – ALGORITMO DE FINAL - UTILIZANDO CÂMARA EM TEMPO REAL

```
from importlib.resources import path
from time import time
import torch
from matplotlib import pyplot as plt

import numpy as np
import cv2

model = torch.hub.load('ultralytics/yolov5', 'custom', path= 'TestV5
Real/best_1000_v2_8b.pt', force_reload=True)
#model = torch.hub.load('ultralytics/yolov5', 'custom', path='TestV5
Real/best_500.pt')  # local model

cap = cv2.VideoCapture(0)
while cap.isOpened():
  start = time()
  ret, frame = cap. read()
  result = model(frame)

  end = time ()
  fps = 1/(end - start)
  #print(fps)
  label = 'Inference time: %.2f ms' % fps
  cv2.putText(frame, label, (0, 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0))

  cv2.imshow('Screen', np. squeeze (result.render ()))

  if cv2.waitKey(10) & 0xff == ord('x'):
    break
  if cv2.getWindowProperty("Screen", cv2.WND_PROP_VISIBLE) <1:
    break
  #print(result[1])
cap.release()
cv2.destroyAllWindows()
```