Universidade da Beira Interior
Departamento de Informática

Nº 127 - 2022: *Object Sniffer: Procura Remota de Objetos a partir de Dispositivos Aéreos Não-Tripulados (UAVs)*

Elaborado por:

Fernando Manuel Nunes Gonçalves

Orientador:

Professor Doutor Hugo Proença

11 de julho de 2022

Agradecimentos

Agradeço em primeiro lugar à pessoa que permitiu que este projeto fosse possível e que sempre me orientou e me apoiou, ao meu orientador, ao professor doutor Hugo Proença. Obrigado por tudo o que me ensinou, ao tempo que me dedicou e a todos os conselhos que me deu. Gostaria de agradecer também a outra pessoa essencial para a realização deste trabalho, à minha namorada, que sempre me apoio e sempre acreditou nas minhas capacidades. Agradeço também à Universidade da Beira Interior, pela disponibilização do computador o acesso do laboratório essenciais para a realização deste trabalho. Por último, mas não menos importante agradeço à minha família por todo o suporte emocional e financeiro.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Acrónimos

CNN *Convolutional Neural Network*

GPU *Graphics Processing Unit*

RoI *Region of Interest*

SociaLab *Soft Computing and Image Analysis Laboratory*

UBI Universidade da Beira Interior

UAV's Dispositivos Aéreos Não-Tripulados

Capítulo

1

Introdução

1.1 Enquadramento

Este relatório foi feito no contexto da unidade curricular de projeto da Universidade da Beira Interior (UBI). Foi no *Soft Computing and Image Analysis Laboratory* (SociaLab), localizado na UBI, que a maioria do trabalho foi efetuado, tendo o *hardware* necessário para a sua elaboração fornecido pelos mesmos. O trabalho referido neste documento tem como base o desenvolvimento de um programa capaz de detetar objetos, pessoas e/ou veículos através da análise de vídeos gravados por Dispositivos Aéreos Não-Tripulados (UAV's).

1.2 Motivação UBI

Este projeto surge no final do percurso académico na Licenciatura de Engenharia Informática e, como tal, tem como objeto motivador o salto para o mercado de trabalho dentro da área e também o desafio de conviver com temas e tecnologias de muito interesse pessoal mas pouco referidas no decorrer do curso. O que promove um aumento das capacidades de adaptação e flexibilidade que o mesmo mercado procura nos trabalhadores.

1.3 Objetivos

Na vídeo-vigilância de espaços públicos, a utilização de UAV's surge como um alternativa relativamente barata e muito mais versátil que as ferramentas mais utilizadas, como câmaras fixas, já que estes têm a capacidade de cobrir áreas muito maiores e de difícil acesso. Surgem, neste contexto, os objetivos

propostos para este trabalho, que serão o desenvolvimento de uma aplicação que ao analisar imagens gravadas por UAV's seja capaz de detetar objetos, pessoas ou veículos especificados numa *query* fornecida pelo utilizador utilizando algoritmos de deteção de objetos em imagens e de estimação de poses.

1.4 Organização do Documento

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

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objetivos e a respetiva organização do documento.

2. O segundo capítulo – **Estado da Arte** – descreve todo o trabalho e revisão teórica que antecedeu a parte prática, apresentando diferentes algoritmos de deteção de objetos e estimação de poses, os benefícios de cada um, a definição das métricas utilizadas e também os *datasets* utilizados.

3. O terceiro capítulo – **Implementação Proposta** – mostra as ferramentas utilizadas e como estas foram aplicadas para o desenvolvimento do programa, explicando a utilidade das mesmas no contexto da aplicação.

4. O quarto capítulo – **Testes e Resultados** – demonstra o testes elaborados no programa desenvolvido, os resultados alcançados e faz-se uma análise dos mesmos.

5. O quinto capítulo – **Conclusões e Trabalho Futuro** – finaliza o trabalho efetuado, explicando os problemas encontrados, o que fica por fazer no futuro e sintetiza o que foi elaborado.

Capítulo

2

Estado da Arte

2.1 *Machine Learning*

Machine Learning é um ramo da inteligência artificial que usa dados e algoritmos de modo a imitar o modo de funcionamento do cérebro humano, tendo como objetivo dar um sistema a capacidade de, por exemplo, descobrir o significado de um texto, detetar objetos e anomalias em imagens, fazer o reconhecimento de vozes, entre outros.

2.1.1 *K-means Clustering*

K-means Clustering é um algoritmo de aprendizagem não supervisionada, sendo um dos mais simples e populares. No seu funcionamento, este algoritmo irá agrupar um número fixo de *clusters* referentes a um conjunto de dados agrupado por si devido a certas semelhanças, no centro de cada um irá existir o centroide e o *output* final é a média dos valores ligados a este.

2.2 *Deep Learning*

Deep Learning é um subcampo de *machine learning* que utiliza redes neuronais com três ou mais camadas para realizar as deteções. O seu funcionamento vai consistir na transição sequencial entre as diferentes camadas de modo a melhorar previsão, isto é chamado de *forward propagation*, a figura 2.1 representa uma *Deep Neural Network* (Rede Neuronal Profunda).

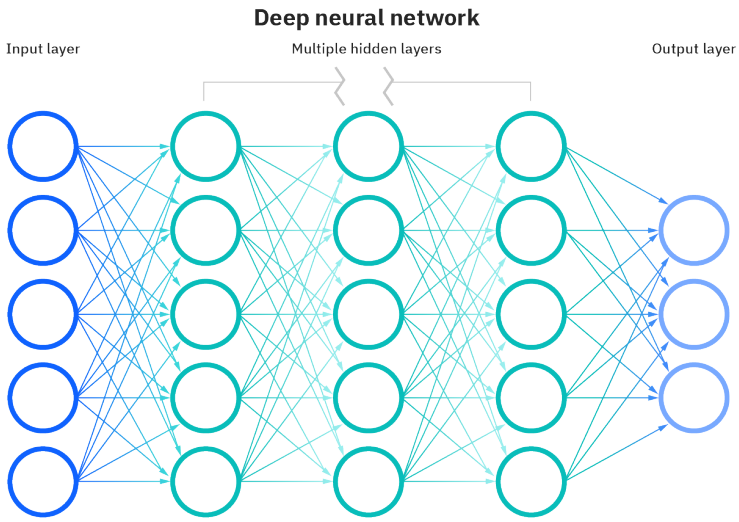


Figura 2.1: Esquematização de uma rede neuronal

2.3 Deteção de Objetos em Imagens

A deteção de objetos é a tarefa de detetar instâncias de objetos de várias classes dentro de uma imagem, bem como adquirir as localizações das mesmas (2.2), utilizando abordagens de *deep learning* que aplicam *Convolutional Neural Network* (CNN)2.3.1

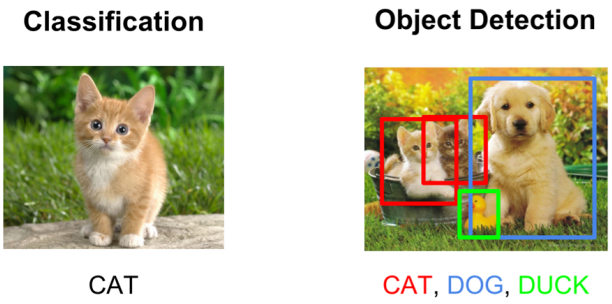


Figura 2.2: Comparação entre Deteção de objetos (direita) e Classificação de imagens (esquerda)

Como principais aplicações desta tecnologia, tem-se a capacidade de, por

exemplo, realizar contagens de uma certa ocorrência de um objeto, detetar anomalias em exames médicos, detetar obstáculos para veículos de condução autónoma e realizar vídeo-vigilância, entre outros.

2.3.1 *CNN*

CNN são algoritmos de *Deep Learning* que aceitam como input imagens e as envolve com filtros e *kernels* para extrair características das mesmas. Os objetos a encontrar não podem ser espacialmente dependentes, isto é, poderão localizar-se em qualquer parte da imagem e a sua deteção não pode ser afetada. Outro aspeto importante será que estes algoritmos devem encontrar características abstratas dos objetos mais complexas à medida que se chega a camadas mais profundas. Na prática, as CNN's consistem na *Input Layer*, *Convolutional Layer, Pooling Layer* e *Output Layer*. Sendo a *Input Layer* a imagem original, a *Convolutional Layer* aquela que aplica os filtros e *kernels*, a *Pooling Layer* que faz *down-sampling* de modo a diminuir o *feature map* (estas duas últimas surgem alternadamente uma à outra) e a *output layer* é uma *full-connected network* que será, no fim, usada numa função de ativação.[8]

2.3.2 *Two-Stage Networks*

Estas redes geram sub-regiões da imagem original, num primeiro estágio, e, de seguida, realizam a classificação dos objetos dentro dessas sub-regiões criadas. São as redes que atingem os melhores resultados quanto à precisão, mas são mais lentos que as *Single-Stage Networks*2.3.3

2.3.2.1 *R-CNN*

A *R-CNN (Region Based Convolutional Neural Network)* é constituída por três módulos distintos: o primeiro gera duas mil sub-regiões independentemente da possível categoria e combina-as em sub-regiões maiores, com base nas características e semelhanças das mesmas, o segundo é uma CNN que extrai um vetor de características de cada região e o último irá separar o objeto e o fundo usando *SVM's (Support vector machines* - métodos de aprendizagem supervisionada usados para classificação, regressão e deteção de *outliers*).

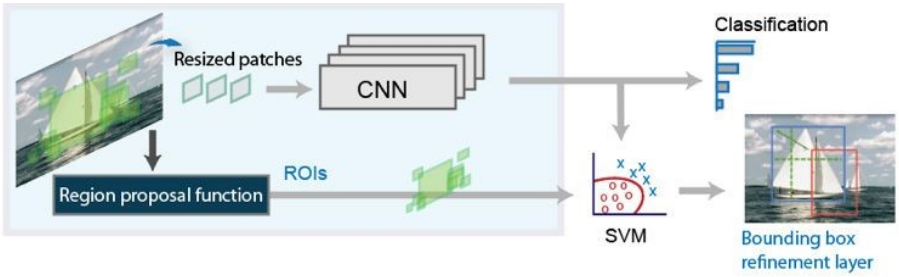


Figura 2.3: Representação da arquitetura da *R-CNN*

[9]

2.3.2.2 *Fast R-CNN*

A *Fast R-CNN (Fast Region Based Convolutional Neural Network)* foi criada para corrigir o problema que surge na R-CNN, no primeiro módulo da mesma. A diferença surge no fornecimento da imagem à *CNN*, em vez de se gerarem duas mil sub-regiões para classificação, fornece-se apenas a imagem original para que se gere um mapa de características. Identifica-se as regiões propostas e, de seguidas, usa-se uma *Region of Interest* (RoI) *pooling layer* para as remodelar para um tamanho fixo e adiciona-las a uma *fully connected layer*. A partir do RoI *feature vector*, usa-se uma camada *Softmax* para prever a classe da região e o valor do *offset* da *bouning box*.

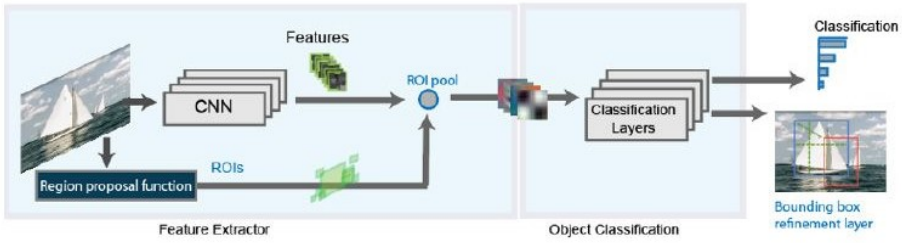


Figura 2.4: Representação da arquitetura da *Fast R-CNN*

[10]

2.3.2.3 *Faster R-CNN*

A *Faster R-CNN (Faster Region Based Convolutional Neural Network)* foi criada para lidar com o restante fator problemático, em termos do tempo de execução e teste, do *Fast R-CNN*, a procura de regiões de proposta. Que utiliza um

algoritmo de pesquisa seletiva que é lento e dispendioso. Para isto, a imagem original é enviada para uma rede convolucional que criará um *feature map* e, ao invés de lhe ser aplicado um algoritmo de pesquisa seletiva, é-lhe aplicada outra rede convolucional que irá prever as regiões propostas. O resto do funcionamento será igual ao do *Fast R-CNN*. A velocidade de teste deste algoritmo permite uma aproximação à deteção de objetos em tempo real. [11]

2.3.3 *Single-Stage Networks*

Nestas redes a *CNN* produz previsões para detetar regiões em toda a imagem usando *anchor boxes*, e as previsões são descodificadas para gerar as *bounding boxes* finais para os objetos. Estas podem ser muito mais rápidas do que as *Two-Stage Networks* 2.3.2, mas não atingem o mesmo nível de precisão, especialmente para cenas contendo objetos pequenos.

2.3.3.1 *YOLO*

O *YOLO* utiliza apenas uma rede convolucional e, para encontrar objetos, irá dividir a imagem numa grelha SxS e gerar m caixas por quadrado, calculando depois a probabilidade da mesma pertencer a uma classe e o seu *offset*, associando essa classe ao quadrado da grelha. Isto irá gerar uma grande quantidade de possíveis deteções que irão ser submetidas a um algoritmo *Non-max suppression* para eliminar as desnecessários e terminar com a *bounding box* ideal para a deteção, como representado na figura 2.5. [3]

Figura 2.5: Representação do resultado da aplicação do *Non-max supression*

2.3.3.1.1 *YOLOv2*

Nesta versão são introduzidas as *anchor boxes* que são idealizações da posição dos objetos a detetar. Estão são usados num cálculo do *IoU (Intersection over*

Union), figura 2.6 para obter o rácio de sobreposição da caixa detetado e da *anchor box*, isto para decidir se a deteção é forte o suficiente.

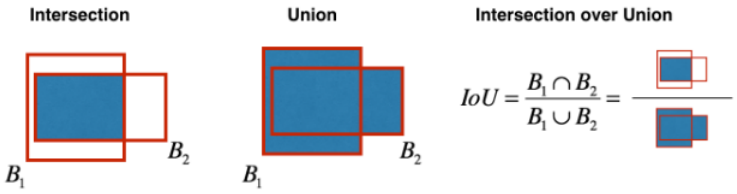


Figura 2.6: Esquema da operação *IoU*

2.3.3.1.2 *YOLOv3*

Esta versão conta com 75 camadas convolucionais o que aumenta a precisão comparativamente aos anteriores e também a torna mais pesada, mas sem perder velocidade, já que não contém *fully connected layers* nem *pooling layers*. Adiciona também *feature pyramid network (FPN)* que extrai características de uma imagem e concatena-as para que o modelo aprenda características locais e gerais.

2.3.3.1.3 *YOLOv4*

Aqui apresentam-se os conceitos de *Bag of Freebies* e *Bag of Specials*.
Sendo o *Bag of Freebies* um conjunto de técnicas e métodos que mudam a estratégia de treino ou custo para maximizar a precisão do modelo, como a *Data Augmentation* (processo que aumenta a variabilidade das imagens de entrada ao criar distorções nas mesmas, entre outros), o *Semantic Distribution Bias* (sendo este o problema de, por exemplo, num modelo que detete o género binário, este possa ser tendencioso para detetar pessoas do género feminino devido a um número e variedade de imagens superior para este género) e o *Objective Function of BBox Regression* (são as chamadas *loss functions* e servem para penalizar e guiar o modelo para uma convergência nas etapas de treino seguintes, temos a *MSE - Mean Square Error*, a *IoU - Intersection over Union*, a *GIoU - Generalized Intersection over Union*, a *DIoU - Distance-IoU* e a *CIoU - Complete IoU*) e sendo o *Bag os Specials* um conjunto de *plugins* e de módulos de pós-processamento que aumentam de forma reduzida o custo de inferência mas aumentas drasticamente a precisão do modelo, como os *Spatial Attention Modules* (que gera *feature maps* usando relações de características inter-espaciais, aumenta a precisão mas aumenta o tempo de treinos), os *Non-max suppression* (que atua no caso de haver objetos próximos com várias caixas, eliminando as falsas e as excessivas), as *Non-linear activation*

functions (usadas para reduzir a linearidade da rede neuronal, permitindo a esta aprender relações mais complexas entre o *input* e o *output*).

2.3.3.1.4 *YOLOv5*

Nesta versão (envolta em alguma controvérsia devido a ser considerada apenas uma implementação do *YOLOv3*2.3.3.1.2 em *PyTorch*) ganha-se a vantagem da implementação ser em *PyTorch* em vez de C (permitindo um ganho na *data augmentation* e no cálculo da perda e uma *framework* de testes e treino de mais fácil utilização, bem como uma instalação e uso muito acessíveis) e de aprender as *anchor boxes* de forma automática (sem ser necessário as adicionar manualmente).

2.3.3.1.5 *Performance*

Com a análise dos gráficos e tabelas A.1, A.2, A.3, A.4 e A.5 e sabendo que cada versão é uma melhoria da anterior, com exceção da quinta versão, os melhores algoritmos *YOLO* a utilizar irão ser os da quinta e quarta versão, visto terem tempo de inferência e *mAP* similares, mas bastante superiores a todas as outras versões.

2.3.3.2 *SSD*

Este modelo divide a imagem original em grelhas de vários tamanhos (sequencialmente mais pequenas) e realiza a deteção de várias classes em diferentes *aspect-ratios*, sendo atribuído um *score* para cada *bounding box*, no fim é aplicado *non maximum supression* naquelas que se sobrepõem para se ficar com a deteção final, representado na figura 2.7. [12]

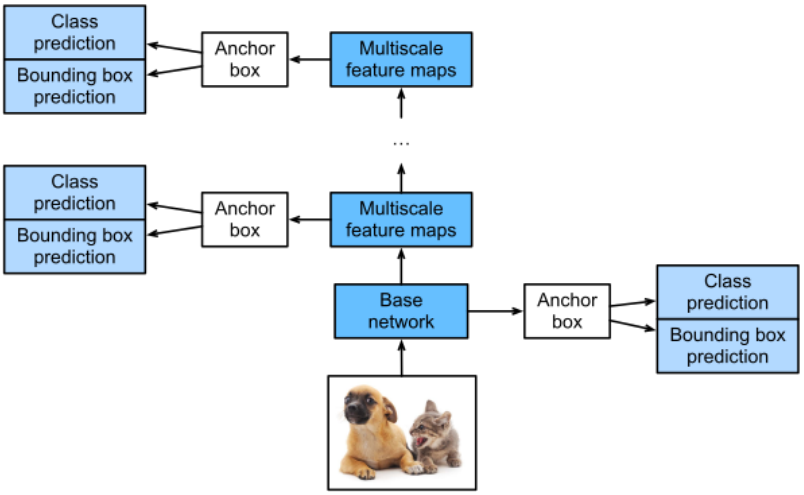


Figura 2.7: Esquema do funcionamento do *SSD*

2.3.4 Métricas na Deteção de Objetos

2.3.4.1 *Precision*

Calculado com a divisão dos positivos verdadeiros com a soma entre estes e os falsos positivos para detetar a percentagem de deteções realmente corretas.

2.3.4.2 *Recall*

Calculado com a divisão entre os positivos verdadeiros e soma entre estes e os falsos negativos para detetar a percentagem de positivos encontrados.

2.3.4.3 *IoU - Intersection over Union*

Esta métrica serve para averiguar o grau de sobreposição entre a previsão e o *ground-truth*. O cálculo é realizado calculando a divisão entre a área da interseção com a união, variando o resultado entre 0 e 1, significando o 1 que a previsão é perfeita. Um deteção será considerada válida quando o resultado é superior a um certo valor, geralmente superior a 0,86.

2.3.4.4 *PR Curve - Precision x Recall Curve*

Esta métrica coloca num gráfico a precisão no eixo y e o *recall* no eixo x. E a sua interpretação baseia-se na tentativa de ter a maior área possível debaixo da curva resultante, vistos serem ambos positivos e ser necessário haver um *trade-off* entre eles no modelo.

2.3.4.5 *AP - Average Precision*

Este métrica é nada mais do que a área por baixo da *PR Curve*. É acompanhada por um valor do *IoU*, como por exemplo AP50 ou AP75, que significa que a *AP* foi calculada usando esse valor.

2.3.4.6 *mAP - Mean Average Precision*

Havendo valor de *AP* por classe, a *mAP* agrega todos esses valores num único valor médio.

2.4 *Pose Estimation*

A *pose estimation* é o problema de estimar a pose de uma pessoa usando tecnologias de *machine learning* aplicadas a fotografias ou vídeos, detetando espacialmente as principais articulações do corpo humano. Os métodos que fazem estas deteções podem ser caracterizados como *Top-down* ou *Bottom-up*, sendo que o primeiro começa por detetar todas as pessoas do *input* e, de seguida, faz a estimação de cada parte do corpo das mesmas, já o segundo deteta primeiramente as diferentes partes do corpo encontradas numa imagem, agrupando-as, de seguida, por pessoa. Cada uma destas abordagens carece, no entanto, das suas desvantagens, como a baixa performance e a dificuldade em realizar deteções quando as poses são complexas ou quando existem um aglomerar de pessoas no *input*, no caso do *Top-down*, e quando a aparência de alguma parte do corpo é ambígua ou existe um aglomerar de pessoas, no caso do *Bottom-up*. [1]

2.4.1 *OpennPose*

Este método de deteção descreve uma abordagem que usa uma representação não paramétrica, chamada *Part Affinity Fields* (PAF), para associar partes do corpo com indivíduos. Alguns dos problemas resolvidos são a possibilidade de detetar articulações corporais individuais mesmo quando no *input* dado existe um aglomerado de pessoas, uma interação irregular entre as mesmas ou o tamanho destas na imagem são diferentes.

Primeiro, para um dado *input*, o conjunto dos mapas de confiança é previsto. Esses mapas representam a localização de cada articulação. Além disso, um conjunto de campos de afinidade parcial é previsto. Esses campos descrevem a localização e orientação das partes do corpo e são descrito como um conjunto de vetores 2D que indicam o grau de associação entre as partes do

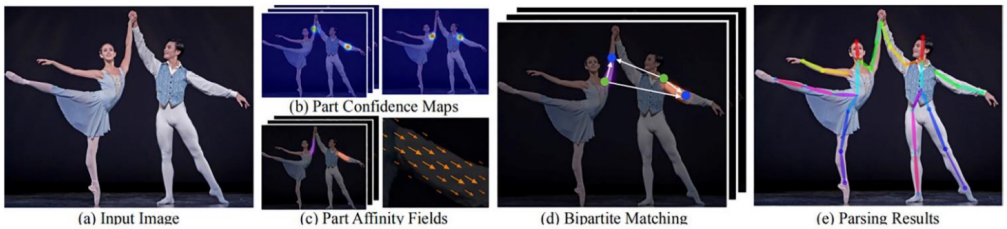


Figura 2.8: Funcionamento do modelo *OpenPose* [1]

corpo. Depois, é necessário um algoritmo que realize a associação entre articulações, isto é feito com *bipartite matching*. No final, todas as articulações terão de ser montadas numa pose de corpo inteiro para cada pessoa incluída no input dado. Esquematizado com um exemplo na figura 2.8 [13]

2.5 *Dataset*

Um *dataset* é uma coleção de dados que irá ser processado pelo algoritmo a treinar, de modo a que este seja capaz de detetar padrões e usar os mesmos para realizar deteções fora do conjunto de dados de treino.

2.5.1 *COCO - Common Objects in Context*

Contando com mais de 328 milhares de imagens e mais de 2,5 milhões de etiquetas entre 91 tipos de objetos diferentes, o *dataset COCO* é um dos mais completos existentes atualmente2.9 e dos que apresenta melhores resultados.

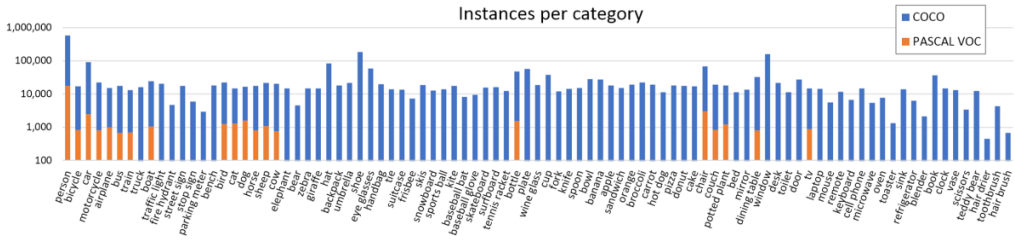


Figura 2.9: Comparação entre os números de instâncias por categoria entre o *dataset COCO* e o *Pascal VOC*[2]

Este *dataset* teve como principal objetivo resolver três grandes problemas na compreensão de cenas, os quais são detetar objetos em perspetivas não canónicas, o contexto e relação entre diferentes objetos e a localização precisa de objetos em duas dimensões.[2]

2.5.2 Custom Gender Dataset

Este *dataset* foi criado no âmbito deste projeto e conta com 107 imagens. Nestas, encontram-se diferentes poses e perspetivas de pessoas que têm a cara visível, com intuito de realizar a distinção entre os sexos masculino e feminino. Os dois sexos têm uma distribuição de cerca de 50%. Para a aquisição das imagens utilizou-se a base de dados *Pexels* (website que fornece milhares de imagens gratuitas e livre de direitos de autor) e para a criação das anotações usou-se a ferramenta online *makesense* [14]. O *download* pode ser feito através do seguinte link:
`https://ubipt-my.sharepoint.com/:u:/g/personal/fernando_goncalves_ubi_pt/EYAV84GiqgNLs3NMe1hGT9YBTSY43woUntnm1nJFXQgh2w?e=gr7ffe`

2.5.3 MPII Human Pose Dataset

Este é um *dataset* para avaliação de poses humanas e conta com mais de 25000 imagens que contêm mais de 40000 pessoas em 410 atividades diferentes, o que lhe proporciona uma grande variedade de poses para o modelo.[15]

Capítulo

3

Método Proposto

Com o objetivo final de se obter uma aplicação capaz de detetar, seguir e contabilizar as ocorrências de um objeto com certas características físicas em vídeos baseadas no *input* do utilizador, é-se proposta uma aplicação que usa um algoritmo de deteção de objetos para detetar o objeto procurado, que pode ser um veículo (carro, mota, camião, autocarro ou barco) ou uma pessoa, submete a deteção a outras funções que irão extrair as características (estas serão a cor, no caso dos veículos, e o género e a cor da roupa no tronco ou pernas, no caso das pessoas), certificando-se que corresponde ao pedido pelo utilizador, fazendo a contabilização estatística de cada uma, não duplicando a contabilização nos *frames* seguintes.

3.1 *Hardware*

Para a realização deste projeto, foi inicialmente fornecido pelo SociaLab um computador com uma *Graphics Processing Unit* (GPU) *NVIDIA® GeForce® RTX 2080 Ti* e 32GB de memória *RAM*. Infelizmente, por motivos alheios, a GPU deste foi trocada por uma *NVIDIA® GeForce® GTX 680* que se mostrou incompatível com a ferramenta *PyTorch* e a solução encontrada foi trocar de computador na sua totalidade, o qual conta com uma GPU *NVIDIA® Titan X Pascal* e 16GB de *RAM*. Este último tem especificações e *performances* teoricamente boas para o projeto em execução, mas mostrou ter problemas a nível de *software* ou até mesmo *hardware* que nunca foram identificados nem, portanto, resolvidos, o que resultou inicialmente na não deteção do *CUDA* pelo *PyTorch* e, ao fim de um certo tempo de atividade do computador (quer fosse ou não a realizar algum tipo de processamento, como um treino do *YOLO*) a perda da *interface* gráfica e consequente perda de conexão com o *AnyDesk*

(programa utilizado para acesso e trabalho remoto). Após uma reinstalação do sistema operativo *Pop!_OS*, resolveu-se o problema do *CUDA* não detetado, mas a perda da *interface* gráfica (possivelmente por um *overflow* - não comprovado - da *VRAM* da gráfica). Todos estes contratempos resultaram na incapacidade da realização de testes fortes e concisos do programa criado, bem como na incapacidade de um treino correto do *dataset* criado 2.5.2.

3.2 Tecnologias e Ferramentas

3.2.1 *Input*

O *input* pode ser feito de duas maneiras, quer por um ficheiro de texto como diretamente na linha de comandos, ao executar o programa. Este baseia-se num vetor com 5 parâmetros e estruturado da seguinte maneira : *Tipo Cor Género Roupa Vídeo*. Cada parâmetro tem o seguinte significado:

- Tipo:
  - 0 - pessoa
  - 1 - carro
  - 2 - camião
  - 3 - autocarro
  - 4 - mota
  - 5 - barco
- Cor:
  - 0 - marrom (castanho-avermelhado)
  - 1 - vermelho
  - 2 - laranja
  - 3 - amarelo
  - 4 - oliva (cor de azeitona)
  - 5 - verde
  - 6 - roxo
  - 7 - magenta
  - 8 - lima
  - 9 - verde-azulado

  - 10 - ciano
  - 11 - azul
  - 12 - naval (azul escuro)
  - 13 - preto
  - 14 - cinzento escuro
  - 15 - cinzento claro
  - 16 - branco

- Género:
  - -1 - caso *Tipo* seja diferente de pessoa (0)
  - 0 - masculino
  - 1 - feminino

- Roupa:
  - -1 - caso *Tipo* seja diferente de pessoa (0)
  - 0 - Parte de cima (casaco, t-shirt, etc...)
  - 1 - Parte de baixo (calças, calções, saia, etc...)

Exemplos:

- python3 YoloOpenPose.py 1 16 -1 -1 videoTeste.mp4 ——> Carro de cor branca
- python3 YoloOpenPose.py 0 11 0 0 videoTeste.mp4 ———-> Homem com roupa azul da cintura para cima
- python3 YoloOpenPose.py input.txt

3.2.2 Linguagem de Programação

A linguagem utilizada foi *Python* devido ao facto de ser uma linguagem amplamente utilizada e, devido a isso, contar com uma enorme variedade de *frameworks* e bibliotecas, bem como as suas documentações, que facilitam de uma forma enorme o ato de programar. Ao ser utilizada por tantos programadores, acaba também a ter uma comunidade enorme o que permite encontrar várias alternativas para o mesmo problema.

3.2.3 Processamento de Vídeo e de Imagens

Para o processamento de vídeo usou-se a biblioteca *CV2* e as suas funções *VideoCapture* (para abrir o vídeo de *input*), *read* (para ler os frames) e *VideoWriter_fourcc*, *VideoWriter* e *write* (para a gravação dos novos *frames* num novo ficheiro *.mp4*). Usou-se também o *Matoplotlib* para a adição das *labels* e *bounding boxes* das deteções.

3.3 *Framework*

Para a aplicação do método de deteção de objetos enunciado na secção seguinte, 3.4.1, utilizou-se a *framework open-source PyTorch*, desenvolvida pela *Meta AI* é baseada em *Python* e na biblioteca *Torch* tal como indica o nome, sendo uma das mais utilizadas para *deep learning*. Tem como principais características vantajosas a computação de *tensors* acelerada pela GPU e a diferenciação automática para a criação e treino de redes neuronais profundas.

3.4 Deteção de Objetos em Imagens

O algoritmo escolhido para realizar a deteção de pessoas e veículos no programa elaborado foi o *YOLOv3*, apresentado no capítulo 2.3.3.1.2. Os principais fatores na sua escolha foram a capacidade de realizar deteções muito perto do tempo real, o que será extremamente útil numa situação em que se estejam a receber imagens em *live-feed* de UAV's, e o valor do mAP-50 ser superior a 50% no *dataset* COCO, significando que este é rápido e preciso para a aplicação em questão. 2.5.1. Esta versão foi lançada no ano de 2018, tendo sido, em 2020, lançadas outras três mais recentes, rápidas e leves, mas devido a problemas de compatibilidade com o sistema utilizado, optou-se por usar esta já que satisfaz todas as necessidades do projeto. No capitulo 2.3.3.1.2 explica-se mais detalhadamente o funcionamento do *YOLOv3* e como se compara com as outras versões.

3.4.1 *PyTorch-YOLOv3*

Tal como descrito pelo autor (Erik Linder-Norén) esta é "uma implementação mínima do *YOLOv3* no *PyTorch*". Esta torna-se especialmente útil devido a dois factores, sendo eles:

- ao contrário do que acontece com o *YOLOv5*, não existe uma implementação oficial do *YOLOv3* em *PyTorch* que suporte que se treine modelos e realizem deteções de modo simples;

- pode ser instalado como uma *API*, permitindo importar os módulos e chamar funções diretamente.

Esta implementação conta com suporte para treino, deteção e avaliação, apresentando no *dataset COCO* resultados ligeiramente inferiores aos descritos no artigo oficial do *YOLOv3*[5] e aos apresentados pelo autor, apresentados nas tabelas 3.1, 3.2 e 3.3.

Index	Class	mAP-50
0	person	0.71421
...	...	...
2	car	0.57038
3	motorbike	0.67145
...	...	...
5	bus	0.83567
...	...	...
7	truck	0.54086
8	boat	0.41651
...	...	...
	mAP-50 final	0.53914

Tabela 3.1: Valor *mAP-50* obtido nas classes usadas e na combinação das usadas e não usadas

Modelo	mAP-50
YOLOv3 (publicação original)	55.3%
YOLOv3 (implementação em PyTorch segundo o autor da mesma)	55.5%
YOLOv3 (implementação deste trabalho)	53.9%

Tabela 3.2: Valores do *mAP-50* obtidos nesta implementação, comparativamente aos valores de referência do artigo original e da implementação em *PyTorch*

Modelo	GPU	FPS
YOLOv3 (publicação original)	TitanX	76
YOLOv3 (implementação em PyTorch segundo o autor da mesma)	1080ti	74
YOLOv3 (implementação deste trabalho)	TitanX	58

Tabela 3.3: Comparação da capacidade média de inferência por segundo

Neste programa, o algoritmo vai receber como *input* o *frame* atual, realizando as deteções e, caso estas coincidam com o pedido pelo utilizador, recorta esse *frame* em imagem mais pequenas e correspondentes às *bounding boxes*, como se vê no seguinte excerto de código:

```
def _draw_and_save_output_image(..., detections, ..., cocoClasses,...):
    ...
    for det_aux in detections:
        x1, y1, x2, y2, conf, cls_pred = det_aux
      if (frame-1)%5 == 0:
        ...
        if int(i[0]) == 0 and cocoClasses[int(cls_pred)] == "person": #--Person--
          ...
        elif vehicles[int(i[0])] == cocoClasses[int(cls_pred)]: #--Vehicle--
          ...
      ...
  ...
...
#Load the YOLO COCO model
modelCoco = models.load_model(
  "PyTorch-YOLOv3/config/yolov3.cfg",
  "PyTorch-YOLOv3/weights/yolov3.weights")
...
cocoClasses = load_classes("PyTorch-YOLOv3/data/coco.names")
...
while true:
...
    if (num_frames-1) %5 == 0: #make new detections in only 1 in 5 frames, to save
        resources
      ...
    boxes = detect.detect_image(modelCoco, img)
  ...
  ... = _draw_and_save_output_image(..., boxes, ..., cocoClasses,...)
```

Excerto de Código 3.1: Funcionamento do *YOLOv3* no programa.

3.5 *Pose Estimation*

Para calcular os *keypoints* das pessoas detetadas pelo *YOLOv3*, ou seja, realizar a *pose estimation*, utilizou-se o algoritmo *OpenPose*. Este vai receber como *input* o recorte da imagem do *frame inicial*, quando é detetada uma pessoa pelo *YOLO* com o modelo *COCO* e depois de passar pela classificação do género pelo modelo explicado na secção 2.5.2. De seguida, é feito o recorte da imagem usando como extremidades os *keypoints* do tronco ou das pernas, dependendo do pretendido pelo utilizador, e, por último, submete-se essa última imagem às funções da cor dominante 3.6.

Esta secção do trabalho apesar de implementada, nunca é atingida pelo programa, visto que o *dataset* dos géneros falha no treino e, portanto, nunca é detetado nenhum género. Provocando um *output* vazio, mesmo quando haveriam casos positivos.

3.6 Deteção da Cor Dominante

De modo a realizar a estimação da cor dominante das deteções provenientes do *YOLOv3* e do *OpenPose* utilizou-se um algoritmo *K-means clustering*2.1.1, utilizando-se a seguinte função:

```
def getDominantColor(img):
  clusters  = 6
  img = imutils.resize(img,height=200)
  flat_img = np.reshape(img,(-1,3))
  kmeans = KMeans(n_clusters=clusters,random_state=0)
  kmeans.fit(flat_img)
  dominant_colors = np.array(kmeans.cluster_centers_,dtype='uint')
  percentages = (np.unique(kmeans.labels_,return_counts=True)[1])/flat_img.shape[0]
  p_and_c = zip(percentages,dominant_colors)
  p_and_c = sorted(p_and_c,reverse=True,key=lambda x: x[0])

  color1 = p_and_c[0][1]
  color2 = p_and_c[1][1]
  color3 = p_and_c[2][1]
  colorsNames = []
  colorsNames.append(getColorName(color1[0]*1.20, color1[1]*1.20, color1[2]*1.20))
  colorsNames.append(getColorName(color2[0]*1.20, color2[1]*1.20, color2[2]*1.20))
  colorsNames.append(getColorName(color3[0]*1.20, color3[1]*1.20, color3[2]*1.20))
  return  colorsNames
```

Excerto de Código 3.2: Função utilizada para obter as cores dominantes de uma imagem utilizando *K-means*.

No excerto de código anterior, a função recebe um imagem que é inicialmente redimensionada, para aumentar a velocidade de execução, e que é transformada num *array* para ser então lida pelo *K-means*. Desta execução é retornado o valor dos 5 *clusters* definidos e estes irão corresponder às 5 cores dominantes. Este valor provém de testes, mais clusters e a execução seria demasiado lenta e menos clusters e a deteção da cor dominante é demasiado generalizada em imagens muito coloridas, por fim ordena-se os *clusters* por percentagem da cor correspondente, enviam-se para a função que transforma os valores rgb em linguagem natural e retornam-se os nomes, em inglês, das três cores mais comuns. Os nomes provêm da função seguinte:

```
from scipy.spatial import KDTree
from webcolors import (
    CSS21_HEX_TO_NAMES,
    hex_to_rgb,
)
def getColorName(b,g,r):
  css21_db = CSS21_HEX_TO_NAMES
  names = []
  rgb_values = []
  for  color_hex, color_name in css21_db.items():
    names.append(color_name)
    rgb_values.append(hex_to_rgb(color_hex))
  kdt_db = KDTree(rgb_values)
  distance, index = kdt_db.query((r,g,b))
  return names[index]
```

Excerto de Código 3.3: Função utilizada para transformar valores *RGB* nos nomes da cor mais próxima em linguagem natural e em inglês.

No excerto anterior, apenas se pretende obter nomes de cores mais simples, como vermelho, preto ou azul, para tal utilizou-se a biblioteca *WebColors* e as suas funções *CSS21_HEX_TO_NAMES* e *hex_to_rgb*, a primeira irá guardar num *array* os nomes das cores contidas no *CSS21* (sendo estas o preto, branco, vermelho, amarelo, verde, ciano, azul, magenta, cinzento escuro e claro, verde-azeitona, laranja, lima, verde-azulado, roxo, azul escuro e castanho). De seguida separam-se em duas listas os nomes em linguagem natural e o valor da cor correspondente em hexadecimal e por fim usa-se uma estrutura *KDTree* para encontrar o index do valor *RGB* da lista de cores mais próximo ao *RGB* de *input*, returnando-se o nome correspondente ao mesmo.

3.7 Estatísticas

As estatísticas serão feitas contabilizando o número de deteções em cada *frame*, evitando duplicar a contabilização do mesmo objeto em cada um usando uma função que irá comparar, com base nas coordenadas da deteção, se as mais recentes se encontram a menos de uma determinada distância limite em relação aos *frames* anteriores. Para tal, são guardados os centros das deteções nos 10 *frames* anteriores numa lista e estas serão comparadas a uma nova deteção na qual é medida a distância entre os dois centros. A cada deteção considerada nova é também gerado um *ID* que é mantido nas deteções seguintes para que sejam consideradas o mesmo objeto, permitindo haver, até certo ponto, o *tracking* do objeto.

3.8 *Output*

Este protótipo terá três *outputs* diferentes. Terá um ficheiro de vídeo com as *bounding boxes* e as *labels* da deteção, um ficheiro txt com o *ID*, tipo, localização e *frame* da deteção e o último será na linha de comandos à medida que as deteções são feitas, com a mesma informação que o *output* do ficheiro de texto.

Capítulo

4

Testes e Resultados

4.1 Deteção de Veículos e Pessoas

Utilizando o método exposto no capítulo 3.4, é realizada a deteção dos diferentes tipos de veículos e pessoas, apresentando-se os resultados apresentados na tabela 4.1 Mesmo aplicando o algoritmo a todos os *frames*, podemos comprovar que o mesmo tem a capacidade de correr em tempo real e com o valor da *mAP* acima dos 50%, 3.4

ID do vídeo	Características do vídeo	Número de deteções	Tempo de execução
1	(960×540) 30fps | 4,2s	379	4,3s

Tabela 4.1: Tempos de execução obtidos nas deteções de veículos e pessoas

4.2 Extração da Cor Dominante

Nesta fase do programa surgiu o problema de ,devido ao fundo das imagens, o valor de cor dominante ser quase sempre cinzento ou preto. Isto surge devido ao *output* do *YOLO* não ter incluído a segmentação da imagem, o que provoca a leitura de pixeis correspondentes, por exemplo, à estrada no calculo da cor dominante de um veículo. Na tentativa de colmatar esta falha, o programa irá aceitar como válida uma deteção em que a cor dominante não seja a pedida, mas em que a mesma esteja como segunda ou terceira cor.

Figura 4.1: Imagem usada nos testes apresentados na tabela 4.2

Número de *Clusters*	Cor Real	Cor dominante 1	Cor dominante 2	Cor dominante 3	Tempo de execução
6	Azul	Prata	Cinzento	Azul Marinho	0,95s
5	Azul	Prata	Cinzento	Preto	0,8s
4	Azul	Prata	Cinzento	Preto	0,6s
3	Azul	Prata	Cinzento	Preto	0,41s

Tabela 4.2: Resultados obtidos ao correr o código na imagem da figura 4.1 utilizando diferentes valores para o número de *Clusters*

Figura 4.2: Imagem usada nos testes apresentados na tabela 4.3

Número de *Clusters*	Cor Real	Cor dominante 1	Cor dominante 2	Cor dominante 3	Tempo de execução
6	Amarelo	Prata	Cinzento	Cinzento	0,98s
5	Amarelo	Amarelo	Prata	Cinzento	0,9s
4	Amarelo	Amarelo	Cinzento	Verde seco	0,8s
3	Amarelo	Cinzento	Amarelo	Verde seco	0,68s

Tabela 4.3: Resultados obtidos ao correr o código na imagem da figura 4.1 utilizando diferentes valores para o número de *clusters*

Ao analisar os resultados da tabela 4.2, consegue-se perceber o problema provocado pelo fundo. A cor da estrada, e também dos vidros e grelha do carro, torna o cálculo da cor dominante muito tendencioso para cores mais escuras e, portanto, acaba a ignorar o azul que, a olho humano, é claramente percetível. No segundo exemplo, tabela 4.3, como a *bounding box* da deteção é muito mais justa aos contornos do carro, o cálculo da cor dominante é totalmente correto para o valor 5 de número de *clusters*. Com base nestes testes, e outros que obtiveram resultados semelhantes, optou-se por se usar 5 *clusters* na versão final do protótipo da aplicação. No que toca à velocidade de execução, esta funcionalidade retira à aplicação, a capacidade de correr em tempo real, visto demorar, por deteção, sempre mais de meio segundo, sendo sempre muito mais lento que os mais comuns 30 *frames* por segundo de qualquer vídeo.

4.3 Cálculo das Deteções Únicas

Esta funcionalidade apresenta resultados satisfatórios e o consumo de recursos é muito reduzido. Vai-se, portanto, ser focado no casos testados em que o mesmo falha. O maior número de falhas surge quando o objeto a ser detetado fica obscuro do vídeo durante alguns *frames*, quando, por exemplo, um edifício ou uma árvore quebra a visão do objeto ou quando há um movimento mais repentino do drone durante a gravação que provoca um deslocamento do centro da deteção de forma exagerada. Outro motivador de falhas, é a presença de falsos negativos na deteção do objeto. Como foi visto nos capítulos anteriores, isto provém essencialmente da deteção incorreta da cor. Na figura 4.3 vemos um caso em que o objetivo será encontrar carros brancos e em que há uma falha provocada pela alteração do ângulo de gravação do drone de forma repentina, o *ID* da deteção passa de 0 para 1 enquanto que deveria de manter o mesmo.

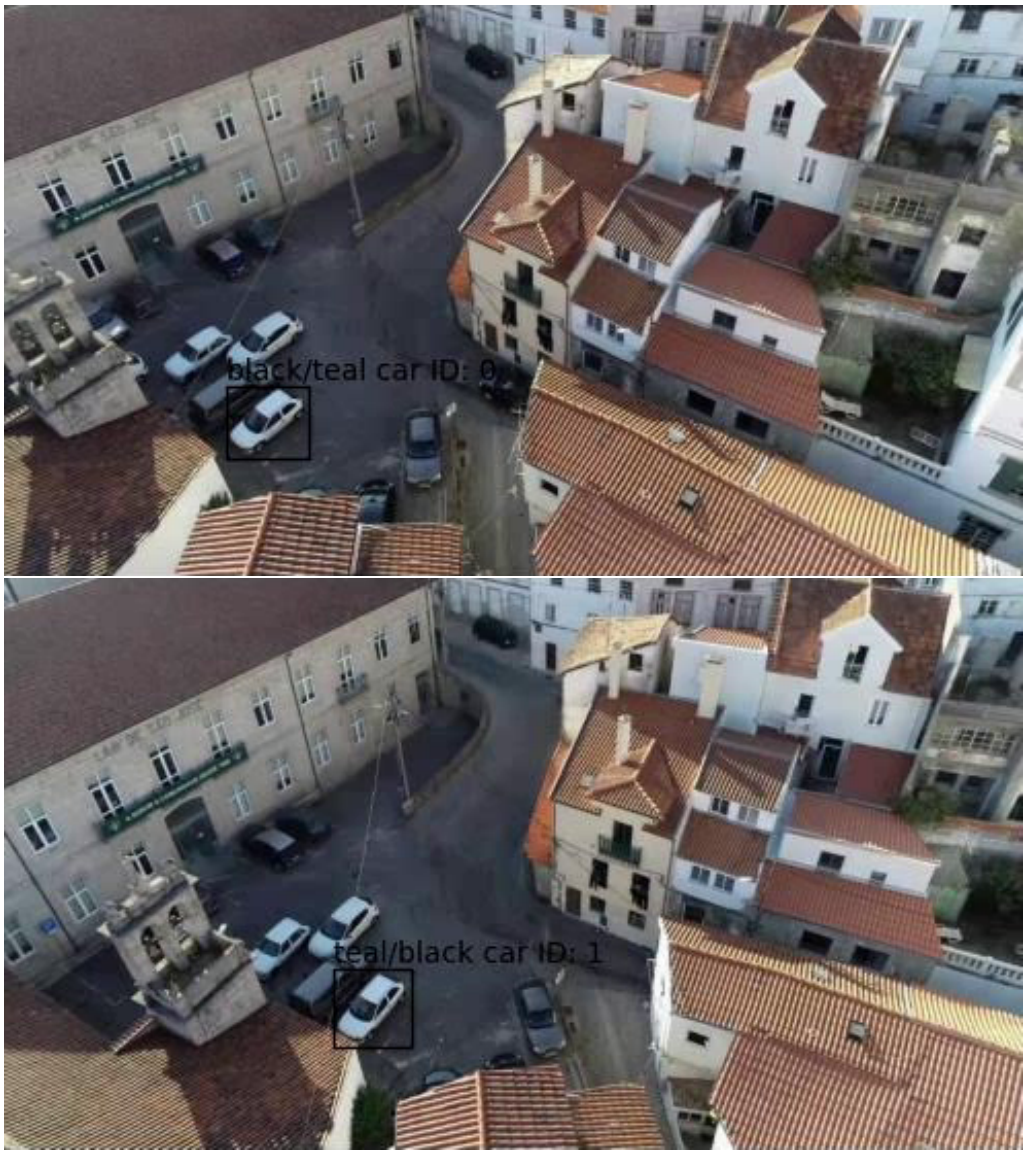


Figura 4.3: Exemplo de falha na contagem única

Na figura anterior, 4.3, vemos também outras duas falhas provocadas pelo cálculo da cor dominante 4.2, devido ao facto das *bounding boxes* conterem os carros numa posição diagonal, existe muito fundo dentro das mesmas provocando um cálculo incorreto da cor. Mesmo na deteção podemos ver que o *output* não indica a cor branca, isto deve-se ao facto dessa cor ser considerada a terceira mais dominante e, então, não aparece no ecrã para não o sobrecarregar de informação. Outro fator que estará a afetar este caso, é a falta de luminosidade, que irá também afastar o *output* da cor branca.

4.4 Treino *Dataset*

Como já foi referido no decorrer deste relatório, o treino do *dataset* enunciado na secção 2.5.2 decorreu de maneira deficiente. Na figuras seguinte comprava-se imediatamente que o modelo é inutilizável com base nos valores do *mAP* da validação do mesmo.

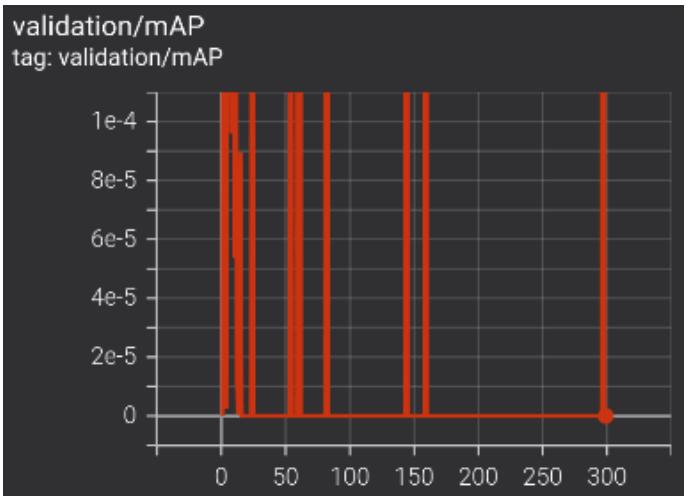


Figura 4.4: Valor do *mAP* obtido da validação do *dataset* de géneros

No treino em si, os problemas continuam e obtendo-se os valores para o *train loss* apresentados na seguinte imagem:

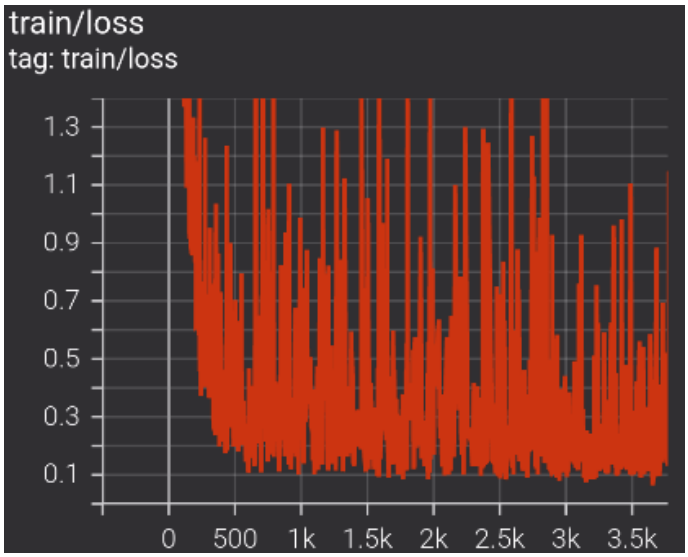


Figura 4.5: Valor do *train loss* obtido do treino do *dataset* de géneros

4.5 Conclusões

No resultado final, juntando todas as funcionalidades das secções anteriores, obtém-se um resultado satisfatório no que toca à deteção de veículos apesar de haver uma grande sensibilidade do programa quando o ambiente não é o ideal, como ambientes de pouca luminosidade ou com movimentos de câmara muito repentinos. De notar a ausência de testes na deteção de pessoas, devido à falha da implementação do *dataset* dos géneros, provocada pela incapacidade de treinar o modelo e ficando por realizar. O programa tem esta última parte implementada, mas o *output* irá ser sempre vazio, pois é incapaz de distinguir os géneros das pessoas detetadas.

Em baixo encontra-se um exemplo de execução utilizando o ficheiro "input.txt"com os parâmetro "1 16 -1 -1 video.mp4", ou seja, para detetar carros brancos:

- Comando a executar:
  - $python3 YoloOpenPose.py 'input/whiteCarDroneLar.txt'
- *Output* na linha de comandos:
  - Time: 0:00:04.033333
  - car Location: [880.2271118164062, 473.6768798828125]

  - Color: gray/red/gray
  - Time: 0:00:04.200000
  - Total time: 135.45285511016846
  - Time spent with YOLO: 4.080198049545288 em 379 deteções
  - Time spent with Statistics: 0.01789093017578125 em 28 iterações
  - Time spent getting the dominant color: 117.30103731155396 em 184 iterações

- *Output* ficheiro *txt*:

  - ID: 281
  - Frame 2401
  - carLocation: [742.925537109375, 454.54168701171875]
  - Color: white/gray/black
  - Time: 0:01:20.033333
  - ID: 279
  - Frame 2411
  - carLocation: [152.77374267578125, 55.993228912353516]
  - Color: gray/teal/white
  - Time: 0:01:20.366667
  - Total number of different ocurrences: 282

- *Frame* do ficheiro de *output* em vídeo:

Figura 4.6: Valor do *train loss* obtido do treino do *dataset* de géneros

4.5.1 *Performance*

Principalmente devido à *performance* da função para adquirir a cor dominante, secção 4.2, optou-se por realizar deteções a cada 5 *frames*, poupando-se muito recursos mas acentuando o problema dos movimentos repentinos da câmara ou objeto detetado para o cálculo da deteção única.

Vídeo	Tempo *YOLO*	Nº *YOLO*	Tempo estatísticas	Nº estatísticas	Tempo cor dominante	Nº cor dominante	Tempo total
1280x720 30fps 1m20s	17,4s	3012	0,18s	485	10m55s	1112	16m35s
960×540 30fps 0m04s	4,1s	379	0,02s	28	1m50s	184	2m15s

Tabela 4.4: Número de vezes que cada funcionalidade corre num determinado vídeo e o tempo gasto nelas (o tempo total será o programa num todo, incluindo o processamento de imagens e vídeo, bem como de ficheiros).

Interpretando os resultados obtidos e exposto na tabela 4.4, conseguimos analisar o local do programa que impede o funcionamento em tempo real do mesmo, sendo então um ponto a melhorar para o futuro.

Capítulo

5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

Tendo sido este projeto o primeiro contacto tido com o mundo do *deep learning*, houve um grande progresso nos conhecimentos teóricos, derivado de todo o trabalho de pesquisa elaborado no capítulo 2. A componente prático demonstrou ser muito desafiadora enquanto que, ao mesmo tempo, se tornava satisfatória à medida que se completavam objetivos, um desafio extra e muito desanimador foi a necessidade de um constante refazer de trabalho e constantes reinstalações de software devido ao problemático computar que foi impeditivo de avançar nesta componente e, desse modo, melhorar conhecimentos e capacidades.

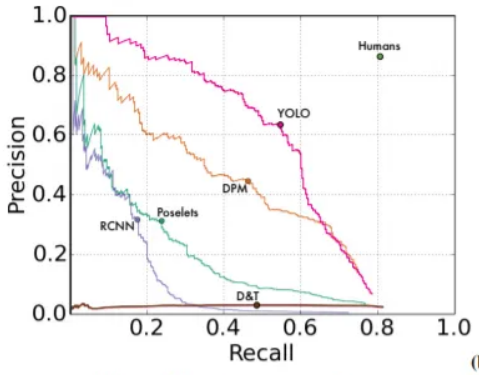
5.2 Trabalho Futuro

Visto que o trabalho efetuado na extração de características nas deteções de pessoas está muito incompleto, fica a faltar essa parte do trabalho, bem como otimizar a parte do programa referente à obtenção da cor dominante de uma imagem, já que esta é das maiores consumidoras de recursos durante a execução. Quanto ao *dataset* de deteção de géneros2.5.2, o treino provou-se um insucesso e é então necessário fazê-lo e até aumentar a variedade e quantidade de imagens nele, este problema. Estes últimos problemas foram, em parte, provocados pelas falhas no computador que nunca permitiram um fluxo contínuo de trabalho no que toca a um constante implementar e testar de funcionalidades. Como funcionalidades extra, seria interessante implementar a capacidade de procurar objetos ou pessoas utilizando como input fotografias dos mesmos ou de semelhantes.

Apêndice

A

Performances *das diferentes versões do* YOLO



(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figura A.1: Comparação da *performance* do *YOLOv1* com outros modelos[3]

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Figura A.2: Comparação da *performance* do *YOLOv2* com outros modelos[4]

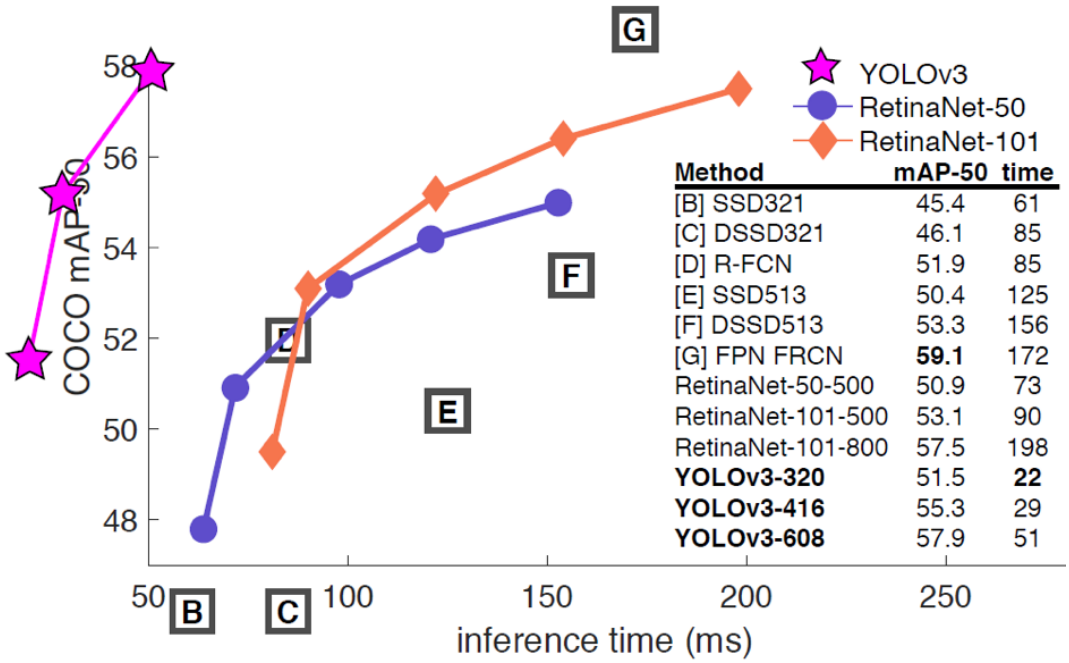


Figura A.3: Comparação da *performance* do *YOLOv3* com outros modelos[5]

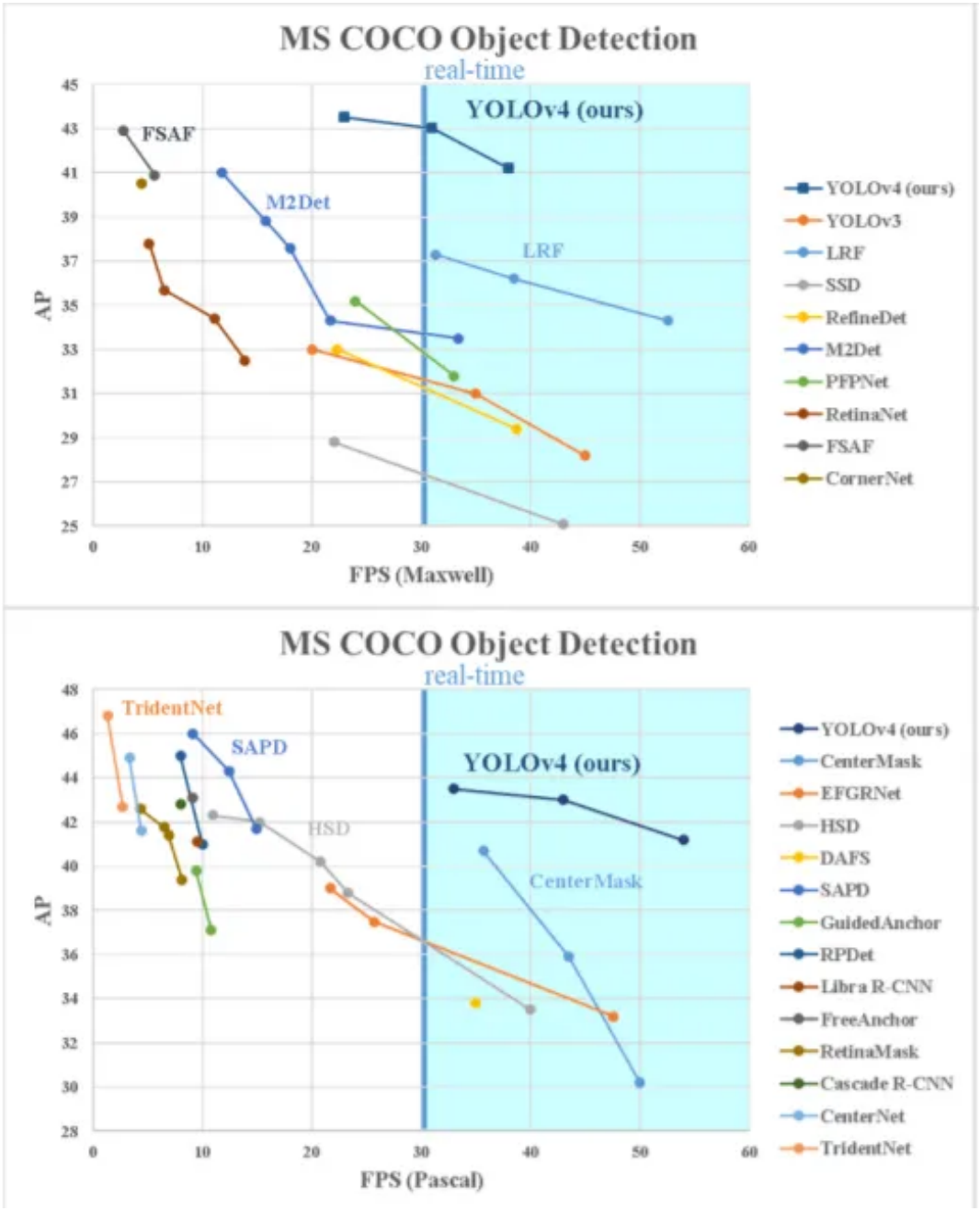


Figura A.4: Comparação da *performance* do *YOLOv4* com outros modelos[6]

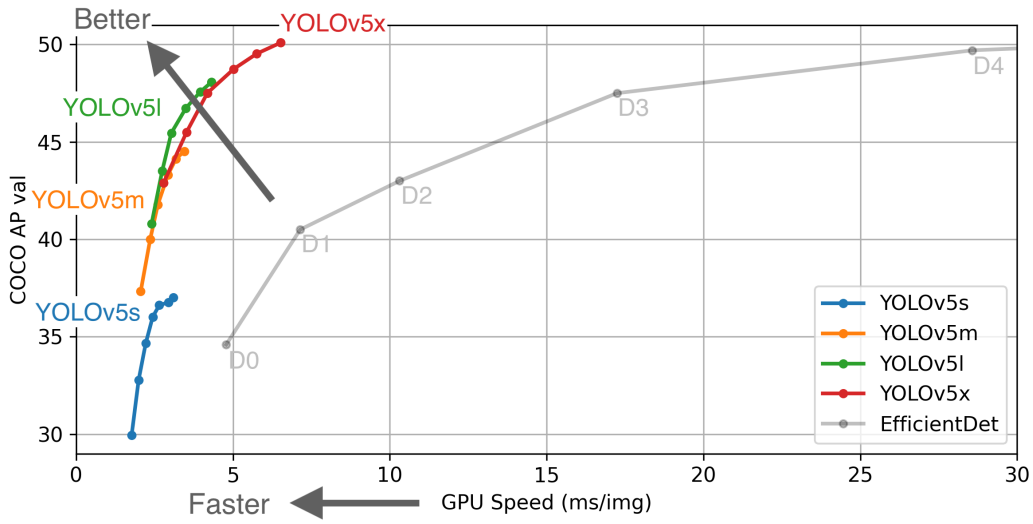


Figura A.5: Comparação da *performance* do *YOLOv5* com outros modelos[7]

Bibliografia

[1] Milan Kresović and Thong Duy Nguyen. Bottom-up approaches for multi-person pose estimation and it's applications: A brief review, 2021.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[4] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[5] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[7] Mateusz Choiński, Mateusz Rogowski, Piotr Tynecki, D.P.J. Kuijper, Marcin Churski, and Jakub Bubnicki. *A First Step Towards Automated Species Recognition from Camera Trap Images of Mammals Using AI in a European Temperate Forest*, pages 299–310. 09 2021.

[8] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. Convolutional neural network (cnn) for image detection and recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pages 278–282, 2018.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[10] Ross Girshick. Fast r-cnn, 2015.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[13] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields, 2018.

[14] Piotr Skalski. Make Sense. `https://github.com/SkalskiP/make-sense/`, 2019.

[15] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.