YoloTestes e VidereApp

Numa fase inicial do desenvolvimento do projeto, decidimos começar por experimentar, através de um pequeno programa, o algoritmo de Deep Learning do YOLOv3/Darknet, disponível no GitHub de pjreddie, com suporte ao YOLO normal e ao YOLOtiny, para dispositivos mais fracos, como telemóveis ou portáteis com gama U de processador. Este pequeno programa serviu para testar a capacidade do algoritmo em detetar objetos antes de começarmos com o projeto em si. Os resultados que obtivemos nestes testes foram, de forma geral, satisfatórios. Detetamos que o algoritmo deteta com bastante precisão, pessoas, ao ponto de só precisar de uma mão aparecer na câmara para o algoritmo dizer que é uma pessoa. Porém, nestes testes verificou-se que o algoritmo não é tão eficaz em detetar alguns objetos diferentes, ainda que os consegue reconhecer alguns. De seguida, apresenta-se algumas imagens (figuras 1 a 6) retiradas pelos elementos do grupo de forma a experimentar as capacidades do algoritmo e os seus resultados. Os resultados das figuras mostramos que o YOLO normal nas nossas máquinas locais apresenta bons resultados de precisão, porém a framerate é bastante baixa, enquanto que o YOLOtiny tem uma framerate superior, a eficácia do algoritmo em detetar coisas pode falhar ou a eficácia cair para metade.



*Figura 1- Rosto de Ricardo Cardoso a ser reconhecido pelo algoritmo com uma eficácia de 52% usando o YOLOTiny.*

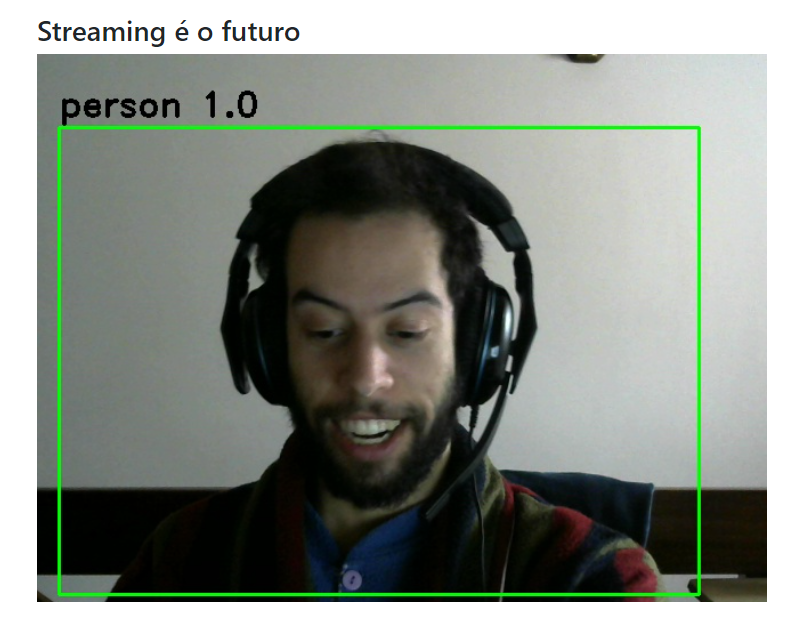


Figura 2- Rosto de Diogo Mendes reconhecimento pelo algoritmo com uma eficácia de 100% usando o YOLO.



Figura 3- Rosto de Reinaldo Coelho reconhecido pelo algoritmo usando o YoloTiny. Aqui verifica-se que o algoritmo ainda não consegue reconhecer uma caixa de um videojogo.

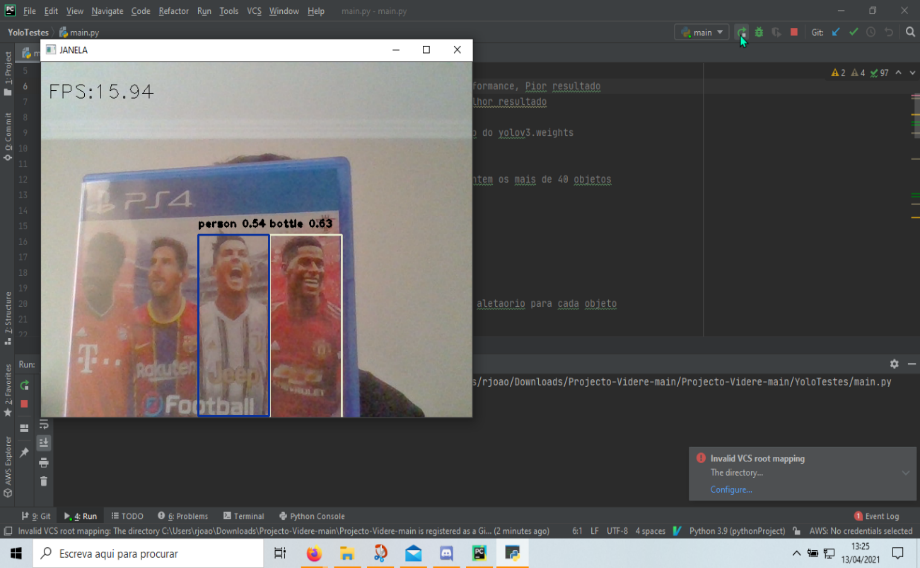


Figura 4 – Uma nova experiência de Reinaldo Coelho com outra caixa de videojogo mostra a alguma dificuldade do algoritmo em detetar corretamente pessoas em uma caixa. Pode-se ver que o algoritmo consegue reconhecer como uma pessoa o jogador Cristiano Ronaldo, já o jogador Marcus Rashford, o algoritmo reconhece-o como uma garrafa.

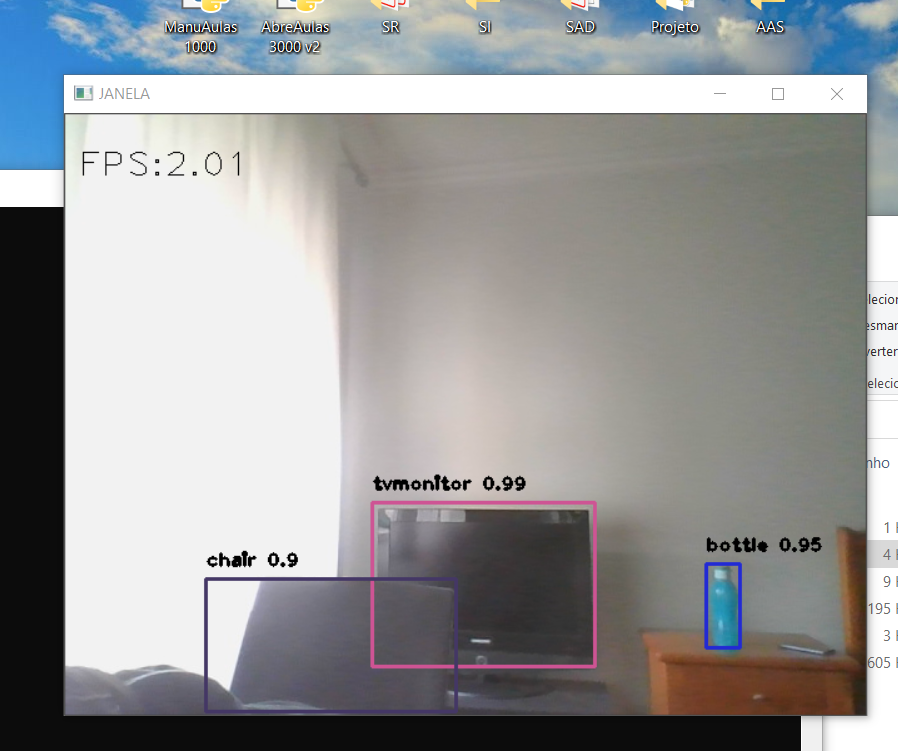


Figura 5- Quarto de Bruno Silva, onde o algoritmo reconhece uma televisão, cadeira e uma garrafa com uma eficácia acima de 90%, mostrando que o algoritmo consegue identificar mais objetos além de pessoas utilizando o YOLO.



Figura 6- Um teste ao algoritmo para reconhecer objetos através de um vídeo, como podemos ver, o algoritmo é capaz de reconhecer a bola de futebol e os dois jogadores que estão na disputa da bola com uma eficácia de 100% através do YOLO.

Começando agora a falar do programa em si do projeto, a VidereApp, ela contém vários ficheiros no seu código para o seu funcionamento. Entre os arquivos Python temos a app.py, camara.py, carregarFicheiros.py, dataset.py, config.py, galeria.py, novaCamara.py, run.py, utilizador.py e videredb.py. O config.py, é apenas o arquivo para correr o programa, a VidereApp, com a porta e a flag de debug configurada no código. Já o config.py, serve para configurar o programa para fazer as suas ligações, onde a porta da aplicação é 5000 e a flag de debug é True, os parâmetros do YOLO que vão ser utilizados, sendo a escolha pré-definida é o YOLO normal para o Path e para o PathWeights, caso seja usada uma máquina de menor potência, estes parâmetros tem de ser alterados para o Tiny YOLO que passa a ser yolo/yolov3-tiny.cfg e yolo/yolov3-tiny.weights invés de yolo/yolov3.cfg e yolo/yolov3.weights, já o outro parâmetro, o yoloDataset, está selecionado o coco.names, disponível no GitHub de pjreddie, que é um dataset do YOLO que contém 80 objetos, como pessoas, bicicletas ou sofás. Também neste ficheiro está a configuração da base de dados utilizada, no PostgreSQL com psycopg2, um adaptador do PostgreSQL para a linguagem Python e a sessão da chave.

O dataset.py está ligado ao dataset do YOLO, o coco.names, o classes e classes\_cores são duas arrays, uma que mostra as classes de nomes dos objetos com o ID e o nome e a outra que vai mostra a cor do quadrado da deteção do objeto através do YOLOv3 por OpenCV. Já as cores estão escolhidas num array de nome “cores” sob a forma de RGB. Segue-se uma pequena função quando o programa abre o dataset que irá mudar a cor do quadrado de deteção do objeto por cada 10 objetos diferentes identificados, havendo 80 objetos neste dataset, haverá 8 cores disponíveis.

No admin.py, foram feitos alguns imports diretamente do Flask como request, Response, Blueprint, render\_template e url\_for, e o redirect através de um import do werkzeug.utils, além de imports que estão no código do app.py e do videredb.py. O Blueprint é um conceito que o Flask usa para a realização de componentes de aplicação e suportar padrões comuns dentro de uma aplicação ou entre aplicações [1]. O Blueprint aqui feito no admin.py vai mostrar a página web do administrador, seguindo-se no código, processos de routing para mapear os objetos de uma tabela e os métodos de Post. Get. Put e Delete como ações que o administrador pode ter na base de dados. Na função janelaAdmin se o user\_id estiver na app.session, poderá haver requisito de método de Post, Put ou Delete. Caso ocorra alguma destas situações, vai ocorrer um pass, que é basicamente um null, onde nada vai ocorrer. Se não ocorrer nenhuma destas situações, o newTabela vai chamar o selectTabela do videredb.py dando depois um return ao template renderizado com o ficheiro HTML do admin, as tabelas classificadas e a tabela selecionada para alteração de valores, segue-se, por fim, um último return que redireciona o URL para o login. (Pode estar tudo incorreto, até porque ainda não meti o PostgreSQL em funcionamento)

[1] <https://flask.palletsprojects.com/en/2.0.x/blueprints/> , Modular Applications with Blueprints, consultado a 9 de julho de 2021.