

|  |
| --- |
| Universidade Fernando Pessoa |
| Deteção de Eletrodomésticos de Cozinha |
|  |
| Carolina Araújo, 33348  João Coelho, 33787  Visão Computacional 2020-2021 |

Índice

[Introdução](#_heading=h.gjdgxs) **2**

[Motivação](#_heading=h.30j0zll) 2

[Objetivos](#_heading=h.1fob9te) 2

[Estado de Arte](#_heading=h.3znysh7) **2**

[Descrição do trabalho](#_heading=h.2et92p0) **4**

[**Análise e Resultados**](#_heading=h.49hhwphskeo7) **8**

[Conclusões e perspetivas de desenvolvimento](#_heading=h.3dy6vkm) **9**

[Referências](#_heading=h.1t3h5sf) **9**

# Introdução

## Motivação

Neste projeto pretende-se que o sistema de visão computacional seja um auxiliar de utilização de equipamentos eletrodomésticos existentes nas nossas casas, em particular, nas cozinhas.

## Objetivos

O objetivo consiste na deteção e reconhecimento de equipamento que pretendemos utilizar, apontando a câmera do vídeo para esse equipamento, isto é, o sistema deve ser capaz de reconhecer o eletrodoméstico e a respetiva marca.

Através de algoritmos de Deep Learning com associação ao OpenCV, pretendemos desenvolver este tipo de deteção.

Com base no estado da arte, recorremos ao YOLO, como um detetor de objetos com base em algoritmos de aprendizagem em Deep Learning. Consiste numa combinação de localização de objetos e reconhecimento de objetos.

# Estado de Arte

Quando falamos em deteção de objetos, por base o Deep Learning, temos de considerar três tipos de detetores de objetos primários como R-CNN e as suas variantes (R-CNN, Fast R-CNN e Faster R-CNN), single shot detetor, SSDs e YOLO.

Um dos primeiros detetores, com base em algoritmos de Deep Learning, e exemplo de um detetor de “two-stage detector” é R-CNN.

Os autores Girschick et al., em 2013, propuseram um detetor de objetos, que por uso de algoritmos de pesquisa seletiva (*Selective search),* utilizava caixas delimitadoras (*bounding boxes)* candidatas a conter objetos.

Um problema com o R-CNN consistia no seu desempenho lento e no facto de não consistir num detetor de objetos de ponto a ponto (*end-to-end).*

Os mesmos autores Girschick et al, em 2013, propuseram um segundo detetor de objetos denominado por Fast R-CNN, que visava o melhoramento do seu antecessor. Aumentava exponencialmente a precisão e reduzia o tempo necessário para o “*forward-pass*”. No entanto, o modelo ainda dependia de algoritmos de proposta de região externa.

R-CNN tornou-se num verdadeiro detetor de objetos de aprendizagem profunda de ponto a ponto com o desenvolvimento do detetor, Faster R-CNN, proposto em 2015 pelos mesmos autores já referidos anteriormente.

Este detetor, em vez de utilizar a pesquisa seletiva, confia na rede da proposta de região (*Region Proposal Network, RPN)*, que consiste numa rede totalmente convolucional e consegue prever caixas delimitadoras de objetos e pontuações de “objetividade”. Os outputs são posteriormente “passados” para a componente R-CNN para a classificação final e a modelação. Define-se pontuações de “objetividade” como a quantificação da probabilidade de uma região de uma imagem conter uma imagem.

Apesar dos melhoramentos de precisão, R-CNN ainda possui desempenhos lentos, obtendo apenas 5 FPS numa GPU.

Para ajudar a aumentar a velocidade dos detetores, surgem os detetores de disparo único como os SSDs e YOLO. Tanto o SSDs e YOLO usam estratégias de detetor de um estágio (*one-stage detector strategy).*

Estes algoritmos tratam a deteção de objetos como um problema de regressão, com uma determinada imagem como entrada, aprendem simultaneamente as coordenadas da caixa delimitadora e as probabilidades de rótulo de classe correspondentes.

Em geral, os detetores de estágio único tendem a ser menos precisos que os detetores de dois estágios, no entanto, são significativamente mais rápidos.

“*You Only Look Once”*, abreviado para YOLO, é um exemplo de detetor de objetos de um único estágio.

Apresentado por Redmon et al [1], em 2015, consiste num detector de objetos capaz de super- deteção de objetos em tempo real, obtendo 45 FPS numa GPU.

É capaz de alcançar grandes números de deteção de objetos, com a execução do conjunto treino de deteção e classificação de objetos. Com o conjunto de treino, os autores treinam YOLO9000 simultaneamente no conjunto de dados de classificação ImageNet e a deteção de COCO.

O resultado é um modelo YOLO que pode prever deteções para o objeto classe que não tem dados de deteção rotulados.

Redmon e Farhadi publicaram recentemente um novo artigo YOLO, YOLOv3 que apresenta melhorias incrementais, sendo o que irá ser usado na nossa metodologia.

# Descrição do trabalho

Recorremos ao dataset do COCO, tendo em conta que é um dos datasets mais usados para a pesquisa de deteção de objetos [3].

COCO é uma abreviatura para objetos comum num contexto, onde, as imagens armazenadas neste dataset são extraídas de situações diárias. O “contexto” é adicionado aos objetos capturados nessas situações ou cenas.

Este dataset fornece a modelação, *labelling*, e a segmentação dos objetos nas imagens, conseguindo criar melhores deteções.

O algoritmo YOLO (“*You Only Look Once”)* treina sobre este dataset, possuindo assim um modelo treinado onde o peso das bounding boxes são calculadas pela probabilidade prevista do objeto na região detetada.

YOLO aplica apenas uma rede neural a uma imagem completa, dividindo consequentemente a imagem em várias regiões e prevê bounding boxes e as probabilidades para cada região.

Numa fase inicial, denominada pela primeira fase, desenvolveu-se este algoritmo com um modelo pré-treinado, com o dataset COCO.

Os pesos já se encontravam pré-treinados (o modelo já estava treinado) apenas fizemos o download do ficheiro associado para aplicar no dataset.

Modificamos o nível de confiança do algoritmo para 0.7 permitindo que o algoritmo consiga fazer uma filtragem de deteções fracas e apresentar um objeto com uma maior confiança de que o objeto é mesmo o que se pretende detetar.

Utilizamos o Deep Neural Network Module, em opencv, com o uso do readNet que consiste na leitura de redes de Deep Learning que é capaz de ler formatos como pesos (\*.weights).

É de referir que a função *getUnconnectedOutLayers*() fornece a posição e o nome das camadas com outputs desconectados, o output é um array de tamanho (1, …).

Determina-se os nomes das camadas de output, com a função *getLayer Names()* e constrói-se um blob da imagem com *cv2.dnn.blob FromImage.*

Um conceito mais informal de blob consiste numa (potencial coleção) de imagens com as mesmas dimensões espaciais (p.e, altura e largura) e a mesma profundidade (número de canais) que necessitam de ser pré-processadas da mesma forma. A função cv2.dnn.fromImage irá retornar um blob, sendo a imagem de input depois de serem feitas subtrações, normalizações e troca de canais.

Este método é usado para preparar imagens de input para classificação por modelos de Deep Learning pré-treinados.

É de referir que tanto a função cv2.dnn.blob FromImage e cv2.dnn.blob From Images são iguais, distinguindo-se apenas no facto da segunda aceitar imagens múltiplas, conseguindo processos grandes conjuntos de imagens.

Neste contexto, como apenas estamos a processar poucas frames, frames de vídeo à vez, não existe a necessidade de utilizar a segunda função.

Sendo assim, o algoritmo fornece o blob à rede obtendo predições, estas predições serão as predições com um maior nível de confiança possível.

Posteriormente, cria-se e inicializa-se as bounding boxes (*boxes)*, as taxas de confiança (*confidences).* Extraímos a classID e filtramos as predições fracas, depois escalamos as coordenadas e as dimensões da bounding boxes para uma visualização correta do vídeo.

Na linha 57, aplicamos explicitamente o “Non-Maxima Supression”, com o uso de cv2.dnn.NMSBoxes uma vez que o algoritmo não o aplica. Usamos para suprimir qualquer bounding boxes fracas ou sobrepostas, obtendo apenas as que o algoritmo tem a certeza de que são as ideais.

Apesar do dataset numa fase inicial fornecer uma grande lista de objetos, existem circunstâncias onde o objeto que se pretende identificar não se encontra incluído, especialmente, quando queremos desenvolver um modelo com um domínio ou cenário mais específico, como é o nosso caso, na deteção de eletrodomésticos, na cozinha.

Desta forma, existem duas abordagens mais comuns:

* **Modelação manual** onde os objetos sofrem uma modelação, utilizando bounding boxes ou a técnica de segmentação e uma rede neural é aplicada para o novo modelo.
* **Transferência de aprendizagem** onde um modelo pré-treinado é adaptado quando é desempenhado um objeto de reconhecimento num novo domínio. Esta técnica consiste em aproveitar as camadas escondidas neste novo domínio e consiste numa técnica mais avançada no campo da Visão Computacional.

Numa segunda fase, criamos um dataset personalizado para os eletrodomésticos presentes nas nossas cozinhas. O nosso dataset é constituído por volta de 98 imagens.

Tendo apenas o dataset constituído por imagens não é o suficiente, pelo que é necessário utilizar um software para efetuar a modelização, ou seja, o labeling. Isto é feito com a ferramenta de software LabelImg [5].

Após esta etapa estar concluída, é necessário treinar o dataset. Esta etapa fundamental foi feita no servidor gratuito da Google, Google Colab. É de referir que o número de classes que corremos no dataset é igual a 10 classes.

No ficheiro de configuração do modelo, modificamos o número de filtros utilizando a seguinte fórmula filters = (classes + 5) \* 3. O modelo de treino irá mudar o número de classes do dataset original (portanto 80 classes) para número de classes que especificamos.



***Figura 1*** *- Output consequente das alterações feitas aos filtros.*

As classes são classificadas em 10 categorias (portanto, de 0 a 9), onde:

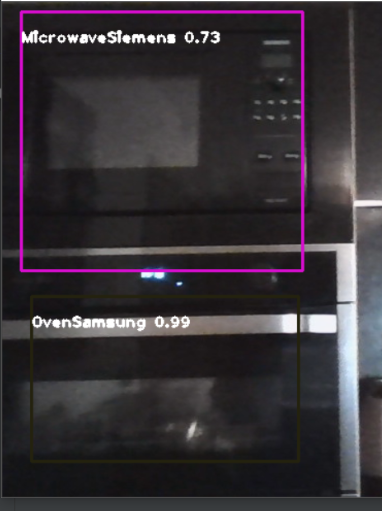
* 0 corresponde ao dishwasher da marca Balay.
* 1 corresponde ao frigorífico da marca Samsung.
* 2 corresponde ao forno de indução da marca Beken.
* 3 corresponde ao micro-ondas da marca Samsung.
* 4 corresponde ao forno da marca Teka.
* 5 corresponde ao robot Bimby.
* 6 corresponde à máquina de café da marca Delonghi.
* 7 corresponde ao micro-ondas da marca Siemens.
* 8 corresponde ao mini frigorífico da Fagor.
* 9 corresponde ao forno da marca Samsung.

Ele vai aprendendo, sabendo que dentro um quadrado formado por 4 pontos (dimensões espaciais) criados nos ficheiros de texto associados a cada imagem, existe a label. A partir daqui, sabe que tem algum objeto daquela classe. Por exemplo, se for “ 1 0.3 0.5 0.6 0.8”, ele sabe que vai existir um “fridgeSamsung”, correspondendo à categoria 1, dentro do quadrado resultante dos quatro pontos referidos no txt.

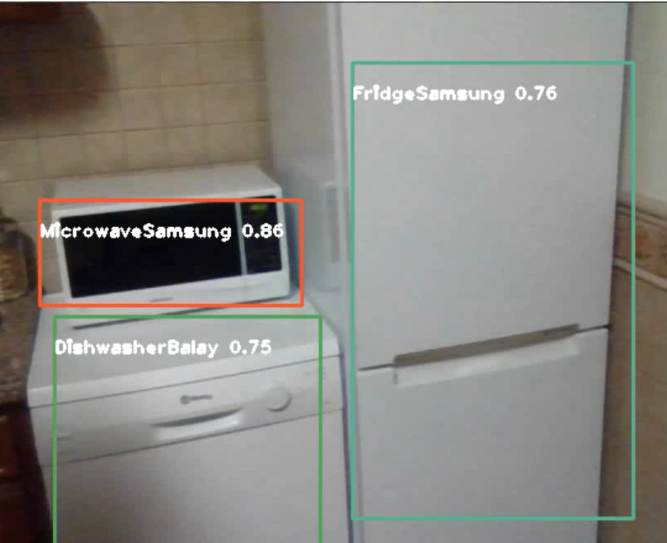
A fase posterior consiste em testar o modelo que foi criado, utilizando o modelo YOLO, modificando os ficheiros a utilizar.

# Análise e Resultados

Na fase inicial do modelo pré treinado, YOLO, com o uso do dataset relativo ao COCO, é possível obter precisões muito perto de 1, uma vez que o dataset possui inúmeras instâncias trabalhadas.

Na segunda fase do trabalho, com o nosso algoritmo personalizado, também conseguimos ter boas precisões, no entanto, para melhorar a sua deteção, será necessário instanciar mais imagens.







# Conclusões e perspetivas de desenvolvimento

O algoritmo YOLO consiste num algoritmo de rápido processamento comparado com outros algoritmos, diferenciando-se um pouco na precisão em comparação a muitos outros algoritmos de deteção de objetos.

De acordo com o estado de arte, é um algoritmo mais utilizado em termos de deteção de objetos e o que até agora revelou ter melhores desempenhos.

Os objetivos foram alcançados neste projeto, tendo apenas a limitação do dataset relativo ao número de imagens e o facto de não possuirmos “super-máquinas” que ajudem a processar um maior número de imagens.

Sendo também uma das melhorias a implementar, isto é, aumentar o nosso dataset de forma a proporcionar melhores precisões na deteção dos nossos eletrodomésticos.

# Referências

[1] <https://arxiv.org/abs/1506.02640> [consultado a 8 de Novembro de 2020]

[2] <https://tech.amikelive.com/node-718/what-object-categories-labels-are-in-coco-dataset/> [consultado a 30 de Dezembro de 2020]

[3] <https://www.perceptolab.com/data-the-fuel-of-machine-learning.html>

[Consultado a 30 de Dezembro de 2020]

[4] https://www.youtube.com/watch?v=\_FNfRtXEbr4&list=PL7XHUVRp-xkOwasaytRRgHSw2LhbW3rVw&index=4

[5] <https://tzutalin.github.io/labelImg/>