Reconhecimento e Classificação de Objetos Recorrendo a Deep Learning.

José Rafael Rijo Trêpo Bate
Faculdade de Ciências e Tecnologias da
Universidade Nova de Lisboa
Almada, Portugal
j.bate@campus.fct.unl.pt

***Resumo* — Este trabalho tem como objetivo fazer uma breve apresentação ao reconhecimento e classificação de objetos recorrendo a deep learning. De tal modo, ir-se-á realizar uma breve introdução a redes neuronais artificiais, seguida de uma apresentação de três arquiteturas de estado da arte de reconhecimento e classificação de objetos, nomeadamente You Only Look Once (YOLO), Region-based Convolutional Neural Network (R-CNN) e Single Shot Detector (SSD). Para terminar será demonstrado e explicado passo a passo como implementar a arquitetura YOLO e treinar um modelo personalizado.**

Palavras Chave — Deep learning; reconhecimento e classificação; YOLO; R-CNN; SSD.

I. INTRODUÇÃO

O reconhecimento e classificação de objetos dentro de uma determinada imagem é um assunto que tem ganho atenção a cada ano que passa e, à volta dele, tem existido uma grande investigação. O avanço de áreas como machine learning, convolutional neural networks (CNN) e o aumento do poder de processamento paralelo por parte de unidades de processamento gráfico (GPU), tem levado a que se tenham a desenvolver arquiteturas cada vez mais eficazes e eficientes no reconhecimento e classificação de objetos.

O reconhecimento de objetos em uma imagem tem por base, dada uma imagem como input, determinar a localização de determinados objetos dentro dessa imagem. Já a classificação corresponde ao processo de atribuir uma label aos objetos identificados. Neste trabalho vão ser estudadas três arquiteturas, baseadas em deep learning, capazes de realizar o trabalho de reconhecimento e classificação de objetos, nomeadamente You Only Look Once (YOLO), Region-based Convolutional Neural Network (R-CNN) e Single Shot Detector (SSD). Foram escolhidas estas três arquiteturas pois são, todas elas, de estado da arte e cada uma apresenta a sua vantagem distinta, não podendo apontar para uma como sendo a melhor. As três diferem em termos de qualidade/velocidade.

Neste trabalho vai-se, também, apresentar o que é uma rede neuronal, como ela funciona e como esta pode ser treinada de modo a reconhecer um determinado objeto numa imagem. As redes neuronais são a base de todas as arquiteturas de deep learning e é a parte mais importante das mesmas. Todas estas arquiteturas são baseadas numa classe específica de redes neuronais, nomeadamente convolutional neural networks (CNN's), que é especificamente destinada a processamento de imagem. A principal diferença entre uma rede neuronal "normal" e uma CNN é que o input de entrada de uma CNN está preparado para ser uma imagem, otimizando assim todo o processo de reconhecimento e classificação de objetos na mesma.

Por fim, vai-se realizar uma demonstração de aplicação da arquitetura YOLO. Nessa secção, vai-se mostrar, passo a passo, como obter um dataset pronto para ser utilizado pela arquitetura, como preparar tudo o que é necessário para que a arquitetura funcione, como treinar um modelo personalizado e, por fim, como reconhecer e classificar objetos em imagem ou vídeo. Toda esta parte do trabalho será realizada em google colab, um serviço cloud gratuito disponibilizado pela google, o que permite que qualquer pessoa em qualquer dispositivo seja capaz de construir o seu próprio classificador.

II. INTRODUÇÃO TEÓRICA

A. Rede neuronal artificial

1) O que é

Uma rede neuronal é uma "tentativa" de modelação de uma rede neuronal do cérebro humano. Esta rede neuronal artificial pode ser treinada através de um algoritmo e de datasets de informação, que permite ao computador aprender por si próprio, tal como os humanos aprendem quando são expostos a informação [1].

No lado humano, tem-se que o que constitui uma rede neuronal são neurónios, já no lado computacional, o que constitui a mesma é um bloco denominado de perceptron. Um perceptron (Figura 1), tem como input um conjunto de valores x1, x2 … xn, cada um com um determinado peso (weight) e o seu core é constituído por uma expressão matemática que resulta num um único output (f(x)). São o conjunto destes perceptrons que formam uma rede neuronal (Figura 2), a qual fica, assim, preparada para fazer o que se chama de deep learning.

A unidade computacional que carrega consigo uma rede neuronal aprende a realizar uma determinada tarefa (por exemplo reconhecer um objeto numa imagem) quando esta é submetida a um processo de treino, no qual vai analisar exemplos de treino, os quais foram previamente rotulados.

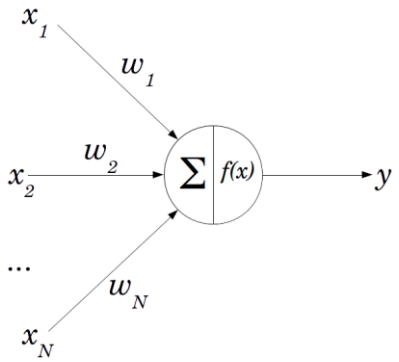


Figura 1. Representação de um perceptron. Tem como inputs o conjunto de valores x1, x2, … xn, cada com um determinado peso w1, w2, … wn e como output y consequente da expressão matemática f(x).

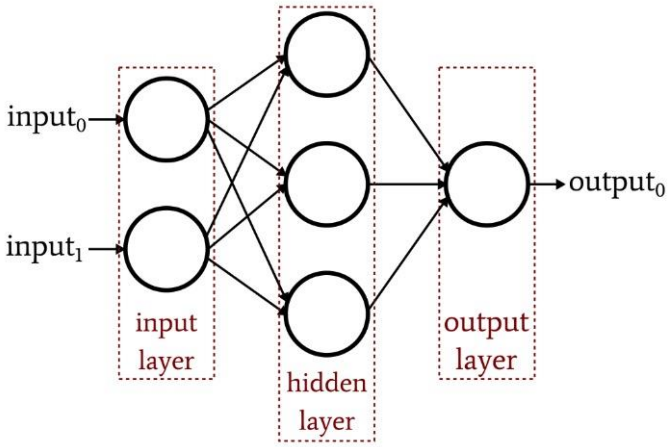


*Figura 2. Representação de uma rede neuronal. É composta por três layers: o de input, no qual se introduzem os dados relativos a um determinado dataset (por exemplo, numa imagem de 24x24 pixels, iram existir 24*24=576 inputs, cada um representando um pixel); hidden layer, no qual acontece todo o processo de decisão para que, assim, se obtenha um; output da rede neuronal. No que no cérebro humano conhecemos por neurónios, aqui estão representados como círculos e são denominados de perceptrons.*

2) Exemplo prático

Pegando no reconhecimento de objetos como um exemplo prático, imagine-se que se tem uma imagem na qual está presente um conjunto de carros, entre eles um Tesla. De modo a que o computador consiga aprender a diferenciar um Tesla de outros carros, é necessário que um humano diga ao computador que existe um Tesla na posição x, y naquela dada imagem. Este processo é denominado de rotulação ou labelling. Uma vez este processo (o qual é o único que necessita de intervenção humana) realizado para centenas ou milhares de imagens, obtém-se um conjunto de dados o qual se denomina de dataset. A unidade computacional é então alimentada com este dataset, a qual vai aprender por si própria a reconhecer o Tesla através de um processo de treino.

3) Processo de treino

O processo que permite a uma rede neuronal criar um caminho de decisão preciso desde o input até ao output é denominado de treino. Quanto mais treino a rede neuronal for submetida, melhor são os seus resultados. Neste processo, a rede neuronal é alimentada com o dataset, anteriormente mencionado, o qual contém os valores de input e os valores de output esperados (perguntas e soluções). O objetivo deste processo é modificar gradualmente as expressões matemáticas que fazem parte de cada perceptron de modo a que, idealmente, a rede seja capaz de calcular o output com até mesmo dados de input que nunca foram antes vistos. Basicamente está-se a dar ao computador uma imagem contém um Tesla e este vai tentar localizar o carro, no final desta tentativa, este vai olhar para o output que é suposto (a solução do problema) e vai comparar com o seu. Se este output for igual, então o computador conseguiu detetar o Tesla como era esperado, caso contrário este reajusta a sua rede neuronal e repete o mesmo processo até que esta fique tão perfeita quanto possível.

O "reajuste" da rede neuronal é essencialmente alterar o peso "weights" que cada perceptron representa. Ao longo do processo de treino, estes pesos vão sendo cada vez mais precisos, o que faz com que, idealmente, o computador seja capaz de reconhecer um objeto pretendido (e.g. Tesla) em qualquer imagem que seja.

4) Weights, o que são?

Como foi visto anteriormente na Figura 1, os weights fazem parte do input de perceptrons. Cada weight está associado a um input (w1 –> x1; w2 –> x2 …) e são usados para definir a importância do respetivo input x no perceptron. O output do perceptron é determinado dependendo se a soma pesada é menor ou maior que um valor de threshold. Tal como os weights, o threshold é um parâmetro do perceptron. Assim, o output é consequente da expressão matemática (1). São estes weights que, durante o processo de treino, são atualizados consoante o output do processo, de modo a que o output seja cada vez mais preciso.

$$output = \begin{cases} a\ se\ \sum_i w_j x_j \leq threshold \\ b\ se\ \sum_i w_j x_j > threshold \end{cases} \quad (1)$$

B. Arquiteturas de reconhecimento e classificação de objetos

Utilizar somente redes neuronais para o reconhecimento e classificação de imagem funciona, mas é um processo muito lento pois não tem qualquer tipo de otimização. Desta maneira, foram inventadas arquiteturas de reconhecimento e classificação de objetos, as quais usam redes neuronais como sua base, mas tentam otimizar todo o processo, desde a imagem de input até ao output, de modo a se ter o melhor ratio de precisão/velocidade de execução possível.

As arquiteturas que vão ser estudadas utilizam uma classe de redes neuronais denominada de redes neuronais convolucionais (conhecidas como CNN – Convolutional Neural Networks). O

seu princípio é fundamentalmente semelhante às redes neuronais normais, o que muda é que neste caso a arquitetura assume que os seus inputs são imagens, o que permite codificar certas propriedades da arquitetura, ficando, assim, a sua implementação mais eficiente, e a quantidade de parâmetros presentes na rede vastamente reduzidos. Pode-se fazer a analogia: se uma rede neuronal é inspirada no cérebro humano, então uma CNN é inspirada no sistema visual do córtex.

As arquiteturas que vão ser estudadas são: You Only Look Once (YOLO), Region-based Convolutional Neural Network (R-CNN) e Single Shot Detector (SSD). Estas arquiteturas têm como objetivo reduzir ao máximo o tempo necessário para a deteção e classificação de objetos assim como maximizar a sua precisão. Tal é conseguido tipicamente através de um pré processamento de imagens, o que vai reduzir a quantidade de informação que é passada para a CNN e, assim, reduzir o tempo de execução.

1) You Only Look Once (YOLO)

a) O que é

You Only Look Once, ou YOLO, é uma arquitetura de estado da arte de deteção de objetos. A arquitetura YOLO faz a deteção e classificação numa única passagem por uma CNN, dai o nome "You Only Look Once" [2]. Esta arquitetura é a que obtém resultados mais rapidamente, mas, em contrapartida, é a que apresenta um nível de precisão mais baixo dentro das três arquiteturas em estudo.

Resumidamente, pré processa-se uma imagem e submete-se a mesma numa CNN, e o output é um "feature vector" que contém retângulos que delimitam os objetos assim como a classificação (classe) de cada um, sendo apenas necessário juntar os outputs.

b) Como funciona

A arquitetura YOLO funciona em essencialmente em três passos: pré processamento de imagem, passagem pela CNN e junção dos resultados da CNN (Figura 3).

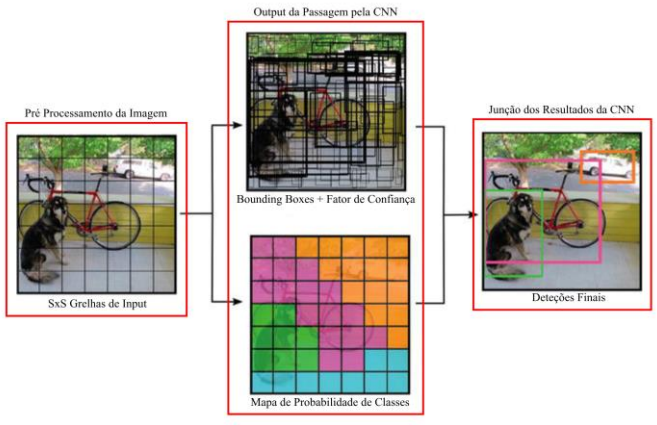


Figura 3. Flow diagram da arquitetura YOLO.

Na fase de Pré processamento, a imagem é dividida em SxS grelhas (adiante grid(s)). Dentro de cada grid, existem B bounding boxes, ficando assim com um total de SxSxB bounding boxes. Estas bounding boxes são o input da CNN, a qual produz bounding boxes com fatores de confiança associados e um mapa de probabilidades de classes. Finalmente, os resultados da CNN são combinados por forma a gerar a previsão final.

2) Region-based Convolutional Neural Network (R-CNN)

Na fase de Pré processamento, a imagem é dividida em SxS grelhas (adiante grid(s)). Dentro de cada grid, existem B bounding boxes, ficando assim com um total de SxSxB bounding boxes. Estas bounding boxes são o input da CNN, a qual produz bounding boxes com fatores de confiança associados e um mapa de probabilidades de classes. Finalmente, os resultados da CNN são combinados por forma a gerar a previsão final, na qual se obtém bounding boxes com classes associadas.

a) O que é

Region-based Convolutional Neural Network, ou R-CNN, foi das primeiras arquiteturas a reduzir significativamente o tempo de deteção e classificação de objetos em imagens.

Esta arquitetura veio revolucionar o mundo de reconhecimento e classificação de objetos através do seu método de "region proposals". Region proposals são essencialmente bounding boxes que são definidas através de "selective search" antes de submeter a imagem a uma CNN.

Existem várias variações da R-CNN, sendo uma delas a Faster R-CNN [3]. Esta variação é, apesar do nome, a mais lenta em comparação com as outras duas em estudo (YOLO e SSD) mas é, porém, a que consegue melhores resultados em termos de precisão.

b) Como funciona

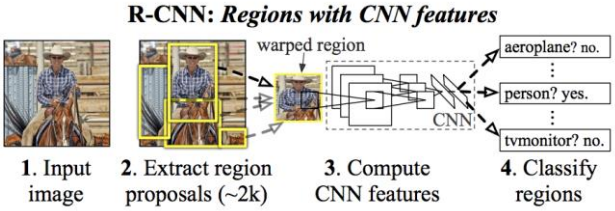


Figura 4. Flow diagram da arquitetura R-CNN.

Ora a R-CNN faz um pré processamento de imagem antes de submeter a mesma numa CNN. Este pré processamento tem como objetivo a definição de bounding boxes através de um método denominado de selective search [4]. Este método de selective search consiste em varrer a imagem através de um conjunto de janelas diferentes com tamanhos diferentes (estes tamanhos são completamente aleatórios) e, para cada janela, tenta juntar um grupo de pixels adjacentes por textura, cor ou intensidade para identificar objetos. Uma vez tendo a imagem definida com um conjunto de bounding boxes, esta é submetida

a uma CNN, a qual vai extrair características de cada bounding box previamente definida. O output da CNN é um vetor de características (feature vector), o qual vai servir como input a uma SVM (support vector machine), a qual é destinada a classificar um único objeto (uma SVM por objeto). De seguida, a R-CNN vai rotular a bounding box com a classe que teve melhores resultados na SVM. Uma vez definida a bounding box que contém um objeto de uma determinada classe, esta é refinada (o seu tamanho é reajustado) através de um regressor de bounding boxes, específico para cada classe.

3) Single Shot Detector (SSD)

a) O que é

A arquitetura Single Shot Detector é, entre a YOLO e a Faster R-CNN, a que apresenta resultados mais medianos. Ou seja, é a arquitetura que apresenta uma melhor relação de velocidade e precisão [5].

b) Como funciona

Tal como o nome indica, esta arquitetura produz um output em "apenas um shot". Isto é, uma dada imagem é submetida a uma CNN, a qual vai tratar de todos os passos até ao reconhecimento e classificação, e dai o nome "Single Shot Detector".

Pode-se dizer que esta CNN está dividida em conjuntos de layers, no qual o primeiro conjunto é denominado por VGG-16 [5]. Este "main" layer é o responsável por obter o feature vector, o qual é submetido a um segundo conjunto de layers, destinados à deteção dos objetos propriamente ditos. Estes últimos layers decrescem em tamanho progressivamente, de modo a que o detetor seja capaz de detetar objetos de várias escalas. As bounding boxes e os seus fatores de confiança são obtidas(os) através de não só um mas sim múltiplos feature maps de diferentes tamanhos que representam múltiplas escalas. A Figura 5 representa esta arquitetura.

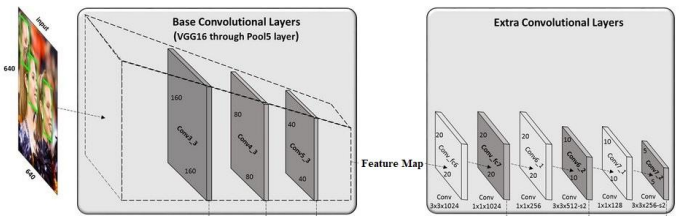


Figura 5. Flow diagram da arquitetura SSD.

4) YOLO vs R-CNN vs SSD

Uma vez apresentadas as três arquiteturas, surge a questão: qual delas a melhor? A resposta a esta pergunta é extremamente relativa, visto que cada uma é boa à sua maneira, cabendo ao utilizador saber qual deve de utilizar dependendo da situação.

Utilizando a Figura 6 como apoio, tem-se que, para uma melhor precisão em termos de reconhecimento e classificação, a Faster R-CNN é a que apresenta melhores resultados, porém a mesma apresenta uma pior velocidade de execução. Por outro lado, caso a velocidade seja o fator mais improtante na aplicação, então sem dúvida alguma que a arquiteturas a ser utilizada deve de ser a YOLO, uma vez que nenhuma se consegue comparar a esta em termos de velocidade. Por fim, existe a SSD, que se encontra no meio da Faster R-CNN e da YOLO. Esta apresenta uma relação qualidade/velocidade mediana entre a R-CNN e a YOLO.

Figura 6. Comparação das três arquiteturas em termos de performance e velocidade.

III. DO ZERO À DETEÇÃO E CLASSIFICAÇÃO

Neste capítulo será exemplificado como partir do zero até à deteção e classificação de objetos recorrendo a YOLO. Neste trabalho treinou-se uma CNN de modo a estar preparada a reconhecer atividade neuronal. Partiu-se de um dataset composto por imagens e labels representado atividade neuronal e o resultado foi detetar atividade neuronal em imagens que não foram utilizadas no processo de treino. Todo o processo revelou sucesso e de extrema facilidade de implementação.

A. Aquisição de um dataset

De modo a serem obtidos *ground-truth data* [5] (anteriormente referido como "solução" que a rede neuronal iria utilizar para ver se o seu output estava certo ou não), é necessário recorrer ao label de imagens de teste e, para tal, é necessário existirem tais imagens.

1) Aquisição de imagens

As imagens foram extraídas de um vídeo, recorrendo ao software "Free vídeo to JPG Converter". A Figura 7 ilustra o procedimento para obter todos os frames presentes nesse vídeo.

2) Labelling das Imagens

De modo ao detetor aprender a detetar neurónios numa imagem, este tem de ser alimentado com dados de treinamento rotulados ou labelled training data. No caso dos neurónios, significa fazer a rotulação manual dos mesmos, ao longo de centenas de imagens. Para obter resultados significativos, é aconselhável realizar o rotulamento de pelo menos 100 objetos (neurónios). Deste modo, foram rotulados cerca de 5 neurónios

por cada imagem num conjunto de 400 imagens, o que corresponde a 2000 objetos rotulados.

O labelling dos ditos objetos foi realizado recorrendo a um script em python que pode ser obtido em [6].

O procedimento é simples, começa-se por fazer o download do repositório [6]. De seguida, copia-se todas as imagens de treino para o folder "images". De modo a definirem-se os objetos que se pretende que o detetor final detete, é necessário definir os mesmos no ficheiro "class_list.txt", da forma: objeto1, seguido de enter, objeto2, e assim sucessivamente até se terem todos os objetos pretendidos. Neste caso, o único objeto pretendido são os neurónios, e para tal, foi apenas considerada uma classe denominada de neuron_active. Finalmente, corre-se o script de python presente nesse mesmo repositório, "run.py".

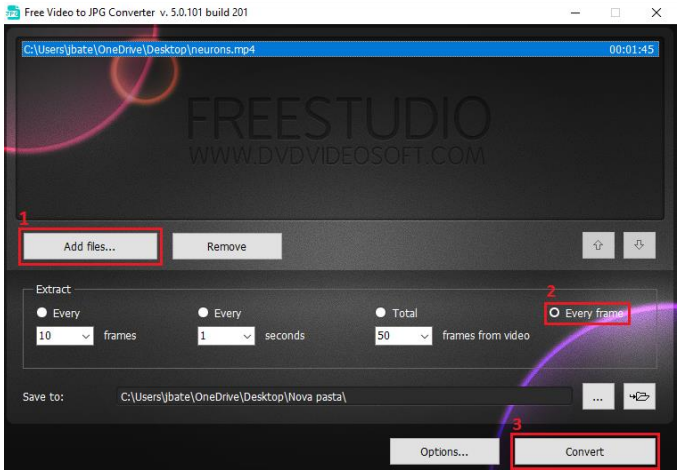


Figura 7. Aquisição de imagens. Utilização do software "Free Video to JPG Converter" para obter todos os frames presentes num vídeo.

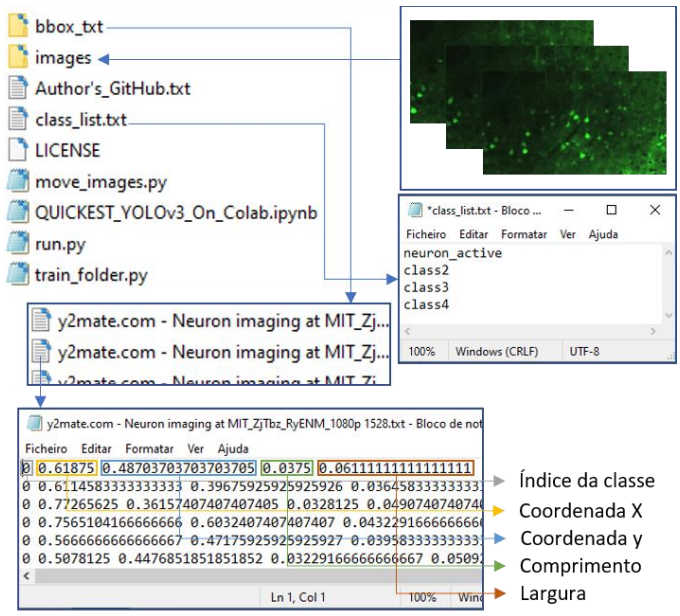


Figura 8. Processo para obter as labels dos objetos nas várias imagens de teste. a) introduzir imagens de teste na pasta "images"; b) introduzir os nomes dos diferentes objetos que se pretendem identificar, denominados como "classes"; c) correr o script "run.py" e identificar os objetos, de modo a obter d) ficheiros de texto, no qual um ficheiro corresponde a uma imagem; e) nos diferentes ficheiros de texto dentro da pasta "bbox_txt" existem informações relativas aos objetos identificados. Cada linha representa informações na forma de vetor sobre cada objeto, onde a sua estrutura é: índice da classe, coordenada x e y do objeto e comprimento e largura do mesmo.

Ao correr o script, é apresentada uma janela que disponibiliza as seguintes opções: dois sliders, um para escolher entre as várias imagens e outro para escolher entre as várias classes; uma interface gráfica, bastando fazer retângulos com o rato à volta dos objetos que são pretendidos o detetor final detetar. Finalmente, após terem sido varridas todas as imagens e terem sido rotulados todos os objetos, basta fechar a janela e, na pasta "bbox_txt" estarão ficheiros txt, em que cada um corresponde a uma imagem. É nestes ficheiros txt que se encontram os dados relativos a cada objeto identificado, nomeadamente a sua posição e o tamanho da caixa que o contém. A ordem desta informação no ficheiro vem como "índex da classe, coordenada x, coordenada y, comprimento, largura". São estes dados, juntamente com as imagens, que vão ser utilizados durante o treino do detetor, como se verá adiante.

A Figura 8 resume todo o procedimento necessário para se obter o dataset, que será utilizado durante o treino do detetor. Quanto melhor este dataset, melhor os resultados do treino e, consequentemente, melhor o detetor e melhores os resultados da deteção de objetos na imagem. Tem-se assim que esta parte do processo é fundamental para um bom resultado na deteção dos objetos.

O script para identificar as imagens é representado na Figura 9:

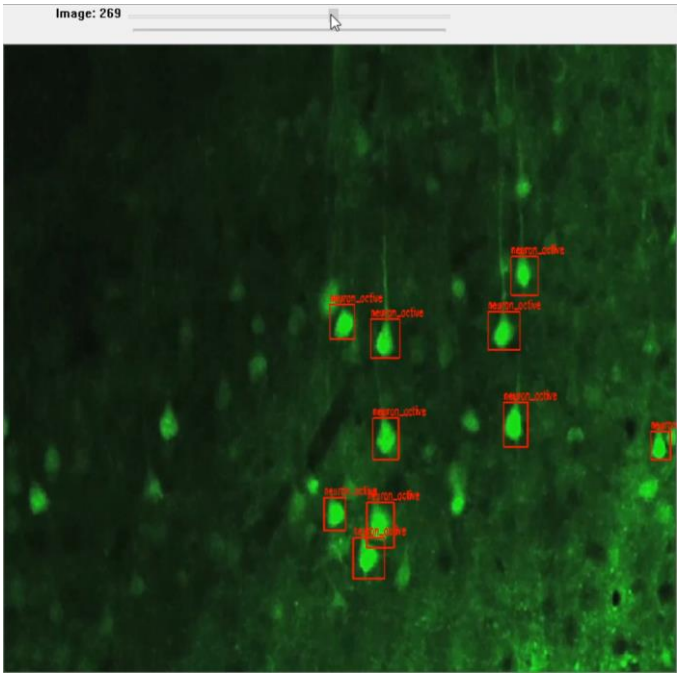


Figura 9. Script de rotulação de objetos. Na parte superior da imagem existem dois sliders, um para escolher entre as imagens e outro para escolher entre as classes. O slider das classes está bloqueado uma vez que, para o caso deste artigo, existia apenas uma classe denominada

"neuron_active". O labelling dos objetos é realizado inserindo caixas à volta dos objetos que se pretendem identificar. Neste caso pretendeu-se identificar neurónios que tivessem acabado de ser excitados, ou seja, neurónios muito claros.

B. Reconhecimento e classificação recorrendo a YOLO

Nesta parte do trabalho, vai-se explicar passo a passo como começar do zero e obter um detetor e classificador de objetos, recorrendo à arquitetura YOLO. Este procedimento é independente do computador que se usa, até se pode realizar num telemóvel, uma vez que todo o processamento será realizado na cloud, em google colab.

1) Setup

Este processo pode ser realizado num CPU ou numa GPU. Dependendo do CPU ou GPU, pode demorar entre horas a dias ou semanas para obter resultados relativamente satisfatórios. Neste trabalho foi utilizado uma ferramenta 100% gratuita chamada google colab. Google colab é um serviço de cloud disponibilizado pela google. É baseado em jupyter notebook e não requer qualquer tipo de setup prévio. Para este caso, o melhor que este serviço tem para oferecer é a sua GPU, que é uma NVIDIA Tesla P100 16GB. Assim, o treino vai ser realizado recorrendo à GPU e à sua API CUDA.

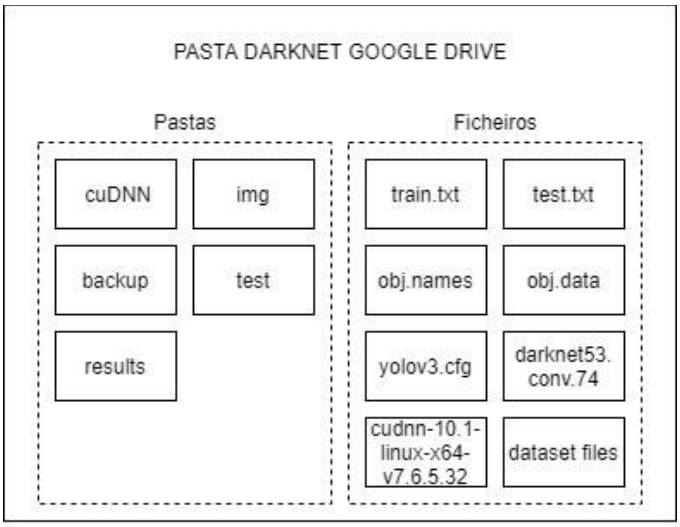
O único contra de utilizar o google colab é que este leva um factory reset a cada 12 horas, o que significa que, de 12 em 12 horas, tudo o que estiver no ambiente de desenvolvimento (código, ficheiros, etc.) é apagado. De modo a contornar este problema, utilizou-se a integração de google drive, o que permite não só realizar backups ao longo do treino para uma pasta na drive, mas também começar a treinar um modelo em apenas um click. A Figura 10 representa um diagrama com o setup necessário para treinar o modelo, assim como detetar os objetos.

Figura 10. Configuração para treino e deteção de objetos.

Como é ilustrado na figura, a google drive e a google colab comunicam entre si, trocando ficheiros necessários à realização do treino/deteção de objetos. Primariamente, tem-se de configurar a google drive com as pastas e ficheiros que o google colab vai necessitar para treinar/detetar os neurónios.

a) Configuração Google Drive

Vai-se então configurar a google drive para que esteja preparada a fornecer os ficheiros necessários à google colab. A Figura 11 representa um diagrama resumo das pastas e ficheiros necessários para a configuração da google drive.



darknet -> train.txt; test.txt; obj.names; obj.data; yolov3.cfg; darknet53.conv

darknet/cuDNN -> cudnn-10.1-linux-x64-v7.6.5.32

darknet/img -> dataset files

darknet/test -> desired test images or videos

Figura 11. Organização das pastas e dos ficheiros dentro da pasta "darknet" localizada na google drive.

De acordo com o diagrama, tem-se que é necessário criar uma pasta principal dentro da google drive chamada darknet. Dentro desta pasta, iram existir as outras restantes presentes no diagrama (cuDNN, img, backup, test, results). Os ficheiros necessários estão também presentes no diagrama, assim como onde os mesmos devem de ser colocados.

A pasta cuDNN contém o ficheiro de instalação da cuDNN. Este ficheiro deve de ser obtido na página oficial da Nvidia. A versão deste ficheiro depende da versão da CUDA instalada no computador do google colab que, de momento, é a versão 10 (caso seja pretendido verificar a versão da CUDA que está instalada no google colab, deve-se de utilizar o comando "!/usr/local/cuda/bin/nvcc –version" na mesma).

A pasta img contém as imagens e os respetivos ficheiros de rotulação previamente mencionados. Todos estes ficheiros ("dataset files") estão dentro desta pasta.

A pasta backup é utilizada durante o treino do detetor, na qual vai ser guardada os vários weight files. Durante o treino, é gerado um primeiro weight file depois de 100 iterações, os restantes weight files são gerados de 1000 em 1000 iterações e, quando o programa é terminado, é gerado um último weight file que corresponde à última iteração realizada.

É na pasta test que devem de ser colocadas as imagens e/ou vídeos que serão sujeitos ao detetor. A pasta results irá guardar

as imagens/vídeos com os objetos identificados resultantes do detetor.

O ficheiro train.txt contém os paths para cada imagem na google drive. Obter todos os paths manualmente tornar-se-ia uma tarefa monótona e exaustiva. Deste modo, é aconselhado a utilização de um script que o faça automaticamente. Este script pode ser encontrado no repositório [6] com o nome "move_images.py".

O ficheiro obj.data contém informações relativas ao número de classes existentes e a paths que serão utilizados durante o treino e durante a deteção. O ficheiro obj.names contém o nome das classes existentes.

Por fim, falta apenas alterar o ficheiro yolov3.cfg. Dentro deste ficheiro, vão ter de ser alteradas duas variáveis, nomeadamente a variável "filters" e "classes". A variável "filters" é definida pela expressão filters = (classes + 5) * 3. A variável "classes" deve de ter o valor do número de classes que estão a ser utilizadas (número de tipos de objetos que se pretende detetar). Neste caso, existe apenas uma classe (neuron_active) e, portanto, para a variável "filters", localizada nas linhas 603, 689, 776, será atribuído o valor 18 e para a variável "classes", localizada nas linhas 610, 696, 783, será atribuído o valor 1.

A Figura 12 é uma imagem resumo de como é suposto a google drive estar configurada depois dos passos em cima mencionados.

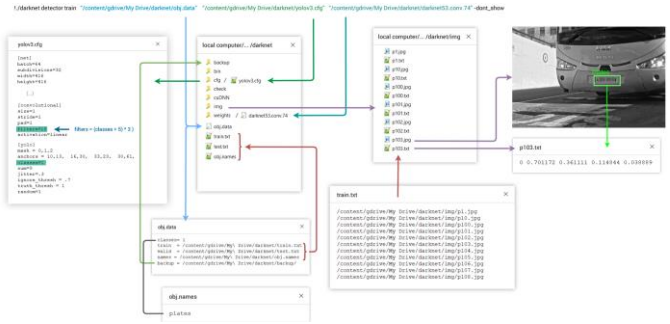


Figura 12. Imagem resumo da configuração da google drive.

b) Configuração Google Colab Notebook

Uma vez realizada a configuração da google drive, passa-se para a configuração do ambiente de desenvolvimento da google, google colab. Como foi mencionado anteriormente, google colab é um serviço web disponibilizado pela google, cuja interface é baseada em jupyter notebook. Visto ser baseada em jupyter notebook, existe neste ambiente de desenvolvimento, células onde se pode inserir códgo, onde cada célula pode ter um pedaço de código que pode ser executado em separado das restantes células. Segue então a explicação desta configuração.

Este notebook está dividido em três células, uma destinada a configurações, outra a treino do modelo e outra para a deteção dos objetos em imagem ou vídeo.

A primeira célula, de configuração, trata de estabelecer a conexão com a google drive previamente configurada, instalar packages, instalar a cuDNN, clonar o repositório da darknet assim como realizar o build do mesmo e definir um procedimento que trata de mostrar uma imagem dado um determinado path. Este procedimento será utilizado para mostrar a imagem resultante da deteção de objetos.

```
from google.colab import drive
drive.mount('/content/gdrive')

!sudo apt-get install tree
!tree /content/gdrive/My\ Drive/darknet/

!tar -xzvf gdrive/My\ Drive/darknet/cuDNN/cudnn-10.1-linux-x64-v7.6.5.32.tgz -C /usr/local/
!chmod a+r /usr/local/cuda/include/cudnn.h
!cat /usr/local/cuda/include/cudnn.h | grep CUDNN_MAJOR -A 2

!git clone https://github.com/kriyeng/darknet/
%cd darknet
!ls
!git checkout feature/google-colab
!make
!cp ./darknet /content/gdrive/My\ Drive/darknet/bin/darknet

def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline
  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)
  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()
```

Figura 13. Primeira célula do notebook. Corresponde às configurações do mesmo.

A segunda célula, de treino, tem dois comandos disponíveis. O primeiro é para ser utilizado na primeira vez que se realiza o treino, o qual vai gerar um ficheiro "yolov_last.weights". O segundo é para ser utilizado caso se pretenda continuar o treino. Como se pode reparar, no segundo comando, é utilizado o ficheiro weights gerado pelo último treino realizado, o qual vai ser utilizado para continuar o treino. Durante o processo, apenas um destes comandos tem de ser utilizado, deixando o outro comentado.

```
!./darknet detector train "/content/gdrive/My Drive/darknet/obj.data"
                          "/content/gdrive/My Drive/darknet/yolov3.cfg"
                          "/content/gdrive/My Drive/darknet/darknet53.conv.74"
                          -dont_show

!./darknet detector train "/content/gdrive/My Drive/darknet/obj.data"
                          "/content/gdrive/My Drive/darknet/yolov3.cfg"
                          "/content/gdrive/My Drive/darknet/weights/yolov_last.weights"
                          -dont_show
```

Figura 14. Segunda célula. Corresponde ao treino do modelo.

A terceira célula, a qual é destinada à deteção dos objetos, começa por mover os ficheiros, que são necessários para a deteção, da pasta criada na google drive para a pasta da darknet que foi previamente clonada. Seguidamente pode ser utilizado um de dois módulos, dependendo se os objetos a identificar estão em formato imagem ou em formato vídeo. Ambos os módulos têm um princípio de funcionamento idêntico, nos quais começa-se por mover o ficheiro onde estão os objetos a serem identificados da pasta da google drive para a pasta darknet e, seguidamente, realizar a deteção propriamente dita. Por fim copia-se o resultado para a pasta "results" da google drive. Caso

o resultado da deteção seja uma imagem, está é mostrada no ecrã utilizando o procedimento imgShow.

```
# Move necessary detector files to darknet/data folder.
!cp "/content/gdrive/My Drive/darknet/backup/yolov3_last.weights" data/
!cp "/content/gdrive/My Drive/darknet/obj.data" data/
!cp "/content/gdrive/My Drive/darknet/yolov3.cfg" data/

# Test model (IMAGE)
!cp "/content/gdrive/My Drive/darknet/test/testimage4.jpg" data/
!./darknet detector test data/obj.data data/yolov3.cfg data/yolov3_last.weights
                        data/testimage4.jpg -dont_show
!cp predictions.jpg "/content/gdrive/My Drive/darknet/results/"
imShow('predictions.jpg')

# Test model (VIDEO)
!cp "/content/gdrive/My Drive/darknet/test/testvideo.mp4" data/
!./darknet detector demo data/obj.data data/yolov3.cfg data/yolov3_last.weights
                        -dont_show data/testvideo.mp4 -i 0 -out_filename result.avi
!cp result.avi "/content/gdrive/My Drive/darknet/results/"
```

Figura 15. Terceira célula do notebook. Corresponde à deteção dos objetos.

2) *Treino da CNN*

Uma vez tendo o setup todo realizado, as restantes etapas são triviais, e a de treino não é exceção. Basta correr a célula presente na Figura 14. Caso seja a primeira vez a treinar a rede neuronal, usa-se o primeiro pedaço de código, o qual usa o ficheiro standard "darknet53.conv.74", deixando o segundo pedaço comentado. Caso já se tenha um ficheiro de weights, então corre-se o segundo pedaço, o qual usa o ficheiro de weights já obtido de um treinamento prévio, deixando o primeiro pedaço de código comentado.

O output previsto para quando se corre a fase de treino está representado na Figura 16. O primeiro número de cada frase (xxx:..) representa o número de iterações realizadas. Segundo a documentação, são consideradas suficientes 2000 iterações por cada classe (objeto) existente no dataset, mas não menos do que o número de imagens presentes no dataset nem menos do que 6000 iterações no total. Mais informação sobre quando parar o processo de treino pode ser encontrada na documentação [7].

```
 117: 232.224716, 342.050476 avg loss, 0.000000 rate, 5.999764 seconds, 7488 images
Loaded: 0.000050 seconds

 118: 224.574158, 330.302856 avg loss, 0.000000 rate, 6.028947 seconds, 7552 images
Loaded: 0.000044 seconds

 119: 211.388046, 318.411377 avg loss, 0.000000 rate, 5.925342 seconds, 7616 images
Loaded: 0.000042 seconds

 120: 204.816269, 307.051880 avg loss, 0.000000 rate, 5.945309 seconds, 7680 images
Resizing
480 x 480
 try to allocate additional workspace_size = 52.43 MB
 CUDA allocate done!
Loaded: 0.000040 seconds
```

Figura 16. Processo de treino da CNN.

3) *Resultados*

Uma vez realizado o processo de treino, basta correr o pedaço de código representado na Figura 15. Esta execução é extremamente rápida, visto que a arquitetura YOLO é a arquitetura mais rápida que existe de momento. A Figura 17 representa o resultado da deteção de neurónios utilizando um ficheiro de weights obtido através de um processo de treino.

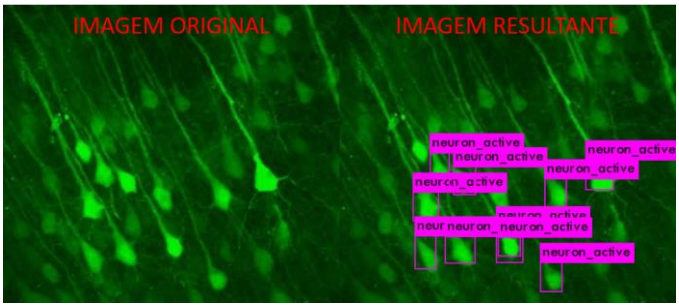


Figura 17. Resultado da deteção e classificação de neurónios recorrendo à arquitetura YOLO.

A imagem utilizada na Figura 17 não foi utilizada durante o processo de treino. É a primeira vez que a rede neuronal a viu e apresentou resultados bastante favoráveis. Tem-se assim que, para além da sua rapidez, a arquitetura YOLO é bastante precisa nas suas deteções e classificações.

IV. CONCLUSÃO

O reconhecimento e classificação de objetos em imagens é uma tarefa de elevado interesse nas mais diversas áreas e torna-se assim de elevada importância compreender como o fazer. Neste trabalho foram estudadas três arquiteturas de estado da arte, nomeadamente YOLO, R-CNN e SSD. Estas três arquiteturas são baseadas em deep learning e convolutional neural networks, o que possibilita que exista um processo de treino por parte do classificador e, consequentemente, que sejam obtidos resultados extremamente satisfatórios tanto em qualidade como em velocidade.

Durante o trabalho, foram analisadas as três técnicas, de onde se concluiu que as três divergem entre si em termos de qualidade de reconhecimento/classificação e de velocidade. Foi visto que, em termos de qualidade, a mais promotora é a R-CNN e, em contrapartida, é também a mais lenta no reconhecimento e classificação. A que apresenta resultados em termos de velocidade mais satisfatórios é a arquitetura YOLO que, em contrapartida, apresenta uma pior qualidade de reconhecimento/classificação de objetos. Por fim, a arquitetura SSD pode ser denominada de arquitetura mediana, sendo a que apresenta os resultados mais balançados entre qualidade de reconhecimento/classificação e velocidade.

No fim do trabalho, foi demonstrado como aplicar uma destas arquiteturas (YOLO) em google colab. Esta demonstração partiu desde a recolha de um dataset até à obtenção (com sucesso) de um classificador treinado para reconhecer atividade neuronal.

Pretende-se, assim, que, com a conclusão deste trabalho, esteja bem compreendido os conceitos acerca de redes neuronais, arquiteturas de reconhecimento e classificação de estado de arte, assim como se esteja preparado para implementar uma delas (YOLO).

Referências

[1] M. Nielsen, Neural Networks and Deep Learning.

[2] S. D. R. G. A. F. Joseph Redmon, “You Only Look Once: Unified, Real-Time Object Detection,” Maio 2016.

[3] K. H. R. G. J. S. Shaoqing Ren, “Raster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” Janeiro 2016.

[4] K. v. d. S. T. G. A. S. J.R.R. Uijlings, “Selective Search for Object Recognition,” 2012.

[5] D. A. D. E. C. S. S. R. C.-Y. F. A. C. B. Wei Liu, “SSD: Single Shot MultiBox Detector,” Dezembro 2016.

[6] A. Z. Karen Simonyan, “Very Deep Convolutional Networks For Large-Scale Image Recognition,” Abril 2015.

[7] A. Rosebrock, “Intersection Over Union for object detection,” Novembro 2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[8] I. Grov, “Open Labelling,” [Online]. Available: https://github.com/ivangrov/YOLOv3-Series/tree/master/%5Bpart%204%5DOpenLabelling.

[9] AlexeyAB, “darknet,” [Online]. Available: https://github.com/AlexeyAB/darknet#how-to-train-tiny-yolo-to-detect-your-custom-objects.