

Porto's Landmarks

Miguel Lucas
ei11140@fe.up.pt
2019-01-08

Faculdade de Engenharia
Universidade do Porto
Porto, Portugal

Abstract

Este projeto teve como principal objetivo o desenvolvimento de um sistema de reconhecimento de um conjunto de imagens previamente selecionadas, as quais apresentam diferentes perspetivas e foram capturadas perante ângulos e condições de iluminação distintos. As imagens, correspondentes a cinco pontos de atração turística da cidade do Porto (a ponte da Arrábida, a Casa da Música, a Casa de Serralves, a Câmara Municipal do Porto e a Torre dos Clérigos), ao serem processadas pelo programa desenvolvido deverão ser classificadas de acordo com o monumento a que dizem respeito, bem como, o mesmo deverá ser capaz de excluir uma imagem que não corresponda a nenhuma das estruturas em causa.

Para o efeito foi aplicada uma CNN [1], como meio para a classificação do conjunto de imagens, e o qual permite estabelecer, através de um algoritmo complexo, determinar quais os pontos de identificação para cada monumento considerando as diferenças entre as imagens de cada edifício.

1 Introdução

A análise de imagem é uma ferramenta poderosa no desenvolvimento de sistemas inteligentes, permitindo extrair e relacionar informação de forma automatizada, a qual é já utilizada de forma estratégica em vários setores.

Existem diferentes métodos conhecidos para esta análise, os quais podem apresentar vantagens e desvantagens entre si, mas neste caso o projeto foi desenvolvido através de CNNs, mais concretamente de uma implementação do algoritmo Keras, uma *Faster R-CNN*[2] denominada RetinaNet. O processo de desenvolvimento do projeto, usando este método de classificação é detalhado, permitindo descrever a sua implementação e identificar as principais limitações encontradas.

2 Convolution Neural Network

Uma Convolution Neural Network é uma rede neuronal convolucional que consiste numa rede neuronal artificial cuja principal aplicação é na classificação de imagens, permitindo, por exemplo, agrupá-las por parentezas e fazer o reconhecimento de objetos dentro das imagens em causa. Os algoritmos para este tipo de redes permitem por exemplo identificar caras, indivíduos, sinais de trânsito, tumores, entre outras aplicações.

A eficácia destas redes convolucionais (CNNs) no reconhecimento de imagens é uma das principais razões do reconhecimento da utilidade do *deep learning*, que tem contribuído para o avanços por exemplo no desenvolvimento da visão computacional com aplicações em diversas áreas como a robótica, segurança e diagnósticos médicos

3 Aplicação do método de classificação

Para ser possível comparar os resultados obtidos com métodos mais tradicionais, implementou-se a framework **RetinaNet** com o modelo ResNet50, descrita em Lin et al. [3], usando a API **Keras**. A arquitetura ResNet é uma arquitetura extremamente profunda mas com uma complexidade inferior às redes que a precederam. Com a ResNet introduziu-se uma nova framework, à base de residual learning, que facilita o treino destas redes tão profundas, que passaram a ser conhecidas por residual networks. Para o backend do Keras, foi usado o **TensorFlow**, uma biblioteca de software open-source com um suporte para *deep learning* e *machine learning*.

Numa primeira fase, e tendo como base o data-set de imagens adquirido pelos estudantes da Unidade Curricular Visão por computador do MIEIC da FEUP, este foi alvo de uma reformatação, sendo que 10 das imagens de cada categoria foi movida para um diretório de teste. Estas serão as imagens que irão testar o modelo, após este ser treinado. Com as restantes imagens em consideração, e a partir dos ficheiros *.xml* com as anotações

de cada imagem, foi efetuado um *parsing* destas, transformando-as num único ficheiro de anotações do tipo *.csv*, com o formato *path/to/image.jpg, x1, y1, x2, y2, class_name*, em que *path/to/image.jpg* corresponde ao diretório onde a imagem se encontra inserida no projeto, *x1, y1, x2, y2* corresponde aos limites da bounding box e *class_name* o nome da classe a que pertence esta imagem. Este formato é necessário para que as imagens sejam reconhecidas na biblioteca *RetinaNet*. Para a identificação das classes, foi criado um outro ficheiro *.csv* com a listagem das 5 categorias de imagens existentes, correspondendo aos 5 monumentos a identificar.

Nesta fase, verificaram-se que algumas imagens se encontravam incorretamente listadas, por vários motivos (formatos de imagem diferentes, valores de bounding box decimais em vez de inteiros, etc.), pelo que estas imagens com anotações incorretas foram descartadas.

Tendo o data-set pronto para ser analisado, procedeu-se ao treino de um modelo para identificação das imagens com base no *ResNet50*. Para efetuar este treino, foi usado o *Google Colab*, uma extensão da *Google Drive*, que permite de forma gratuita aceder a uma plataforma de edição online de ficheiros *Python* e que disponibiliza uma GPU para projetos de *deep learning*. Através do uso desta GPU, foi possível acelerar de forma significativa o tempo de treino do modelo. No entanto, devido a restrições computacionais e temporais, este modelo foi treinado com 10 Epochs, cada uma com 700 passos. Estes valores estão longe de um patamar aceitável, que deveriam ser na ordem das 50 Epochs com 10.000 passos cada uma. No entanto, tendo em conta que o treino nestas condições demorou cerca de 2,5 horas, e após testar este modelo com o data-set de testes, verificou-se já o potencial de identificação das imagens pretendidas.

No entanto, este modelo de treino não estaria pronto ainda para ser usado. Visto ser um modelo de treino, apenas possui as camadas necessárias para treino (regressão e valores de classificação). Para efetuar a deteção de objetos nas imagens, foi necessário converter este modelo de treino num modelo de inferência, usando novamente a biblioteca *RetinaNet* para este processo.

Após obter este modelo de inferência, tornou-se possível classificar uma imagem do data-set de teste. Verificou-se um nível aceitável de reconhecimento de imagens, tendo obtido resultados com sucesso em todas as categorias. Considerou-se um resultado com sucesso todos aqueles que teriam um score de reconhecimento acima de 50%, sendo rejeitados todos os que estariam abaixo deste valor.

4 Análise dos resultados obtidos

Os resultados obtidos para a classificação das imagens foram bastante bons tendo em conta o nível de treino reduzido do modelo. Nas figuras 1 e 2 foram identificados, respetivamente, a ponte da Arrábida e a Casa de Serralves, com os scores de aceitação de 90,8% e 71,4%. Como mencionado anteriormente, apenas imagens com um score superior a 50% foram aceites, sendo as restantes rejeitadas pelo programa, como na figura 4 onde não foi identificado nenhum dos edifícios. Verificou-se que os scores inseriam-se num intervalo entre os 50% e os 92%.



Figure 1: Identificação da ponte da Arrábida



Figure 2: Identificação da casa de Serralves

Verificaram-se também alguns casos onde o programa detetava que ocorriam sobreposição de imagens, obviamente de forma incorreta, como na figura 3. Existiram ainda casos onde as imagens foram categorizadas corretamente, mas a bounding box não ficou posicionada corretamente. No entanto, apenas se verificou isto no caso da Torre dos Clérigos (Figura 5).

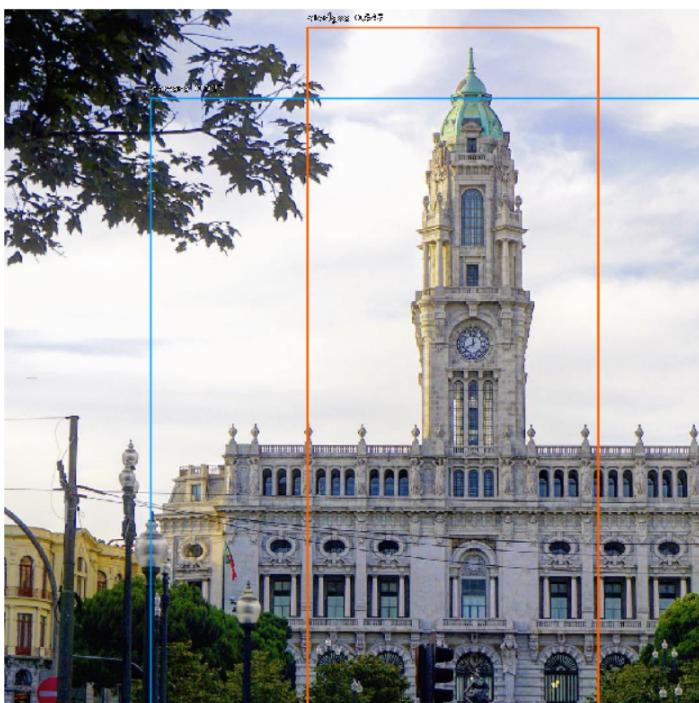


Figure 3: Sobreposição de imagens

De uma forma geral, os resultados foram positivos, com a exceção da Torre dos Clérigos, onde apenas uma das imagens de teste foi corretamente categorizada. Este resultado é particularmente intrigante, dado que do data-set usado, a Torre dos Clérigos era uma das categorias que possuía mais imagens para treino. No entanto, caso o modelo tivesse mais tempo de treino, estes erros seriam corrigidos.

Na tabela seguinte mostra-se os níveis de aceitação para cada categoria.

Ponto de referência	Identificação correta	Identificação incorreta
Arrábida	40%	10%
Câmara	80%	20%
Clérigos	10%	0%
Música	50%	0%
Serralves	60%	40%

Table 1: Níveis de aceitação para cada categoria

Relativamente ao treino do modelo, os seguintes valores de perda foram obtidos em cada *Epoch*:

# Epoch	Loss	Regression Loss	Classification Loss
1	3.2190	2.0752	1.1438
2	2.9263	1.7523	1.1740
3	2.5357	1.4716	1.0641
4	2.1186	1.3530	0.7657
5	1.7382	1.1585	0.5796
6	1.5823	1.0990	0.4833
7	1.4623	0.9871	0.4751
8	1.3341	0.9566	0.3775
9	1.3206	0.9142	0.4064
10	1.1985	0.8475	0.3510

Table 2: Níveis de perda para cada *Epoch*

Verifica-se um valor inicial relativamente alto, mas que vem diminuindo a cada *epoch*, indicador do bom funcionamento da rede. No entanto, um número mais elevado de *epochs*, iria ajudar a reduzir ainda mais este valor e a obter uma maior exatidão na classificação.

5 Grau de conclusão e trabalho futuro

De acordo com o enunciado da Unidade Curricular, foram alcançadas as etapas 1 e 2 com sucesso. A classificação de imagens é obtida com um grau de aceitação satisfatório, comparando com o nível de treino do modelo, e o objeto é identificado em cada imagem corretamente com muito poucas exceções. Ambos estes casos poderiam ter resultados superiores caso o modelo fosse treinado com outros parâmetros, o que levaria mais tempo para conclusão.

6 Conclusão

A implementação do projeto de classificação de imagens, com ajuda da implementação RetinaNet do Keras foi concluída e permitiu um erro aceitável tendo em conta a variedade das imagens pertencentes ao mesmo monumento. A melhoria dos resultados obtidos poderia ser também conseguida recorrendo a um treino mais intensivo do modelo o que permitiria aumentar a exatidão do resultado, no entanto, o mesmo implicaria um esforço computacional bastante superior. Por último, teria sido útil comparar os resultados obtidos com outras frameworks ou backends (Theano, YOLO) e verificar as diferenças nos resultados obtidos em outros métodos de classificação.

References

- [1] https://en.wikipedia.org/wiki/Convolutional_neural_network.
- [2] Ross Girshick Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. <https://arxiv.org/abs/1506.01497> Shaoqing Ren, Kaiming He.
- [3] R. Girshick K. He T. Y. Lin, P. Goyal and Oct 2017. doi: 10.1109/ICCV.2017.324. P. Dollár. Focal loss for dense object

detection. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 2999–3007.

A Anexos

A.1 Figuras



Figure 4: Falha na Identificação da Torre dos Clérigos



Figure 5: Falha na bounding box da Torre dos Clérigos



Figure 6: Identificação correta da Câmara Municipal do Porto

A.2 Código

```
1 from google.colab import drive
2 drive.mount('/content/gdrive', force_remount=True)
3 !ls '/content/gdrive/My Drive'
4
5 !rm -r keras-retinanet
6 !git clone https://github.com/fizyr/keras-retinanet.git
7
8 # %cd keras-retinanet
9 !pip install . --user
0 !python setup.py build_ext --inplace
1
2 # %cd /content
3 !ls -l
4
5 !python keras-retinanet/keras_retinanet/bin/train.py --epochs
       =10 --steps=700 csv 'gdrive/My Drive/VCOM/annotations.csv'
       , 'gdrive/My Drive/VCOM/classes.csv'
6
7 # %cd /content
8 !python keras-retinanet/keras_retinanet/bin/convert_model.py ,
       gdrive/My Drive/VCOM/resnet50_csv_10.h5' 'gdrive/My Drive/VCOM/inference_model_10.h5'
9
10 !mv 'snapshots/resnet50_csv_10.h5' 'gdrive/My Drive/VCOM'
```

Listing 1: train.ipynb

```
1 from keras_retinanet.models import load_model
2 from keras_retinanet.utils.image import read_image_bgr ,
       preprocess_image , resize_image
```

```
from keras_retinanet.utils.visualization import draw_box, draw_caption
from keras_retinanet.utils.colors import label_color
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os

def main():
    model = load_model('../models/inference_model_10.h5',
        backbone_name='resnet50')

    image = read_image_bgr('../testset/serralves/serralves-0174.jpg')
    imagecopy = image.copy()
    imagecopy = cv2.cvtColor(imagecopy, cv2.COLOR_BGR2RGB)

    image = preprocess_image(image)
    image, scale = resize_image(image)
    boxes, scores, labels = model.predict_on_batch(np.expand_dims(image, axis=0))

    boxes /= scale
    visualize_detections(imagecopy, boxes, scores, labels)

def visualize_detections(draw, boxes, scores, labels):
    # load label to names mapping for visualization purposes
    labels_to_names = {0: 'arrabida', 1: 'camara', 2: 'clerigos', 3: 'musica', 4: 'serralves'}

    # visualize detections
    for box, score, label in zip(boxes[0], scores[0], labels[0]):
        if score < 0.5:
            break

        color = label_color(label)
        b = box.astype(int)
        draw_box(draw, b, color=color)
        caption = "{} {:.3f}".format(labels_to_names[label], score)
        draw_caption(draw, b, caption)

    plt.figure(figsize=(15, 15))
    plt.axis('off')
    plt.imshow(draw)
    plt.show()

if __name__ == '__main__':
    main()
```

Listing 2: `cnn.py`

```
1 import xml.etree.ElementTree as ET
2 import csv
3 import os
4
5 ANNOTATIONS_RAW_DIR = '../dataset/annotations-raw/'
6 CSV_PATH = '../dataset/annotations/'
7 FOLDERS = ['arrabida', 'camara', 'clerigos', 'musica', 'serralves']
8
9
10 csv_data = open('../dataset/annotations/annotations.csv', 'w', newline=',')
11 csvwriter = csv.writer(csv_data)
12
13 for folder in FOLDERS:
14     path = ANNOTATIONS_RAW_DIR + folder
15     csv_path = CSV_PATH + folder
16
17     for filename in os.listdir(path):
18         tree = ET.parse(path + '/' + filename)
19         root = tree.getroot()
20
21         annotation = []
22
23         count = 0
24         filename = root.find('filename').text
25         filename_without_extension = filename.split('.')[0]
26         folder = filename.split('-')[0]
27         image_path = 'images/' + folder + '/' + filename
28
29         object = root.find('object')
30         bounding_box = object.find('bndbox')
31         xmin = bounding_box.find('xmin').text
32         xmax = bounding_box.find('xmax').text
33         ymin = bounding_box.find('ymin').text
34         ymax = bounding_box.find('ymax').text
35
36         if '.' in xmin or '.' in xmax or '.' in ymin or '.' in ymax:
37             continue
38
39         annotation.append(image_path)
40         annotation.append(xmin)
```

```
41 annotation.append(ymin)
42 annotation.append(xmax)
43 annotation.append(ymax)
44 annotation.append(folder)
45
46 #csv_data = open(csv_path + '/' + 
47 filename_without_extension + '.csv', 'w')
48 #csvwriter = csv.writer(csv_data)
49 csvwriter.writerow(annotation)
50 #csvwriter.writerow(annotation)
51 #csv_data.close()
52
53 csv_data.close()
```

Listing 3: xml_to_csv.py

```

1 #!/usr/bin/env python
2 from __future__ import division
3
4 import argparse
5 import os
6 from pprint import pprint
7
8 import keras
9 import keras.preprocessing.image
10
11 import tensorflow as tf
12
13 import keras_retinanet.losses
14 import keras_retinanet.layers
15 from keras_retinanet.models import load_model
16 from keras_retinanet.callbacks import RedirectModel
17 from keras_retinanet.preprocessing.csv_generator import
18 CSVGenerator
19 from keras_retinanet.models.resnet import resnet50_retinanet
20 from keras_retinanet.utils.keras_version import
21 check_keras_version
22
23 # parameters
24 batch_size = 1
25 steps_per_epoch = 700
26 epochs = 10
27 steps_per_training_epoch = 700
28 training_epochs = 10
29
30 # paths
31 repository_dir = os.getcwd()
32 repository_dir = os.path.dirname(repository_dir)
33 dataset_dir = os.path.join(repository_dir, 'dataset')
34 annotations_path = os.path.join(repository_dir, 'dataset\\'
35 annotations\\annotations.csv')
36 classes_path = os.path.join(repository_dir, 'dataset\\classes\\'
37 classes.csv')
38
39 def get_session():
40     config = tf.ConfigProto()
41     config.gpu_options.allow_growth = True
42     return tf.Session(config=config)
43
44 def create_models(num_classes):
45     # create "base" model (no NMS)
46     image = keras.layers.Input((None, None, 3))
47
48     # model = ResNet50RetinaNet(image, num_classes=num_classes,
49     # weights='imagenet', nms=False)
50     #model = load_model('../models/resnet50_coco_best_v2.1.0.h5',
51     # backbone_name='resnet50')
52
53     model = resnet50_retinanet(image, num_classes=5, weights=''
54     'imagenet', nms=False)
55     training_model = model
56
57     # append NMS for prediction only
58     #classification = model.outputs[1]
59     #detections = model.outputs[2]
60     #boxes = keras.layers.Lambda(lambda x: x[:, :, :4])(detections)
61     #detections = keras_retinanet.layers.
62     #NonMaximumSuppression(name='nms')([boxes, classification,
63     #detections])
64     #prediction_model = keras.models.Model(inputs=model.inputs,
65     #outputs=model.outputs[:2] + [detections])
66
67     # compile model
68     training_model.compile(
69         loss={
70             'regression' : keras_retinanet.losses.smooth_l1()
71         },
72             'classification': keras_retinanet.losses.focal(),
73         },
74         optimizer=keras.optimizers.adam(lr=1e-5, clipnorm
75 =0.001)
76     )
77
78     return model, training_model, prediction_model

```

```

69
70
71 def create_callbacks(model, training_model, prediction_model):
72     callbacks = []
73
74     # save the prediction model
75     checkpoint = keras.callbacks.ModelCheckpoint(
76         os.path.join(dataset_dir, 'resnet50_{epoch:02d}.h5'),
77         verbose=1
78     )
79     checkpoint = RedirectModel(checkpoint, prediction_model)
80     callbacks.append(checkpoint)
81
82     lr_scheduler = keras.callbacks.ReduceLROnPlateau(monitor='
83         loss', factor=0.1, patience=2, verbose=1,
84                         mode='
85             auto', epsilon=0.0001, cooldown=0, min_lr=0)
86     callbacks.append(lr_scheduler)
87
88     return callbacks
89
90
91 def create_generator():
92     # create image data generator objects
93     train_image_data_generator = keras.preprocessing.image.
94         ImageDataGenerator(
95             horizontal_flip=True,
96         )
97
98     train_generator = CSVGenerator(
99         annotations_path,
100        csv_class_file=classes_path,
101        base_dir=dataset_dir,
102        batch_size=batch_size
103    )
104
105     return train_generator
106
107 if __name__ == '__main__':
108     print(annotations_path)
109     print(classes_path)
110     # make sure keras is the minimum required version
111     check_keras_version()
112
113     keras.backend.tensorflow_backend.set_session(get_session())
114
115     # create the generators
116     train_generator = create_generator()
117
118     # create the model
119     print('Creating model, this may take a second... ')
120     model, training_model, prediction_model = create_models(
121         num_classes=5)
122
123     # print model summary
124     print(model.summary())
125
126     # create the callbacks
127     callbacks = create_callbacks(model, training_model,
128                                 prediction_model)
129
130     # start training
131     training_model.fit_generator(
132         generator=train_generator,
133         steps_per_epoch=steps_per_training_epoch,
134         epochs=training_epochs,
135         verbose=1,
136         callbacks=callbacks,
137     )

```

Listing 4: train.py