NATIONAL CHIAO TUNG UNIVERSITY

Institute of Computer Science and Engineering

Selected Topics in Visual Recognition using Deep Learning

Homework 3

Author: Alessandro Saviolo

Student Number: 0845086

26th November 2019

Summary

1. Introduction

The purpose of this document is to present and analyze the object detector built for Homework 3. The dataset used for training the model is The Street View House Numbers (SVHN) Dataset.

The model used is RetinaNET. Due to lack of computational power, the presented model is trained by first loading a pre-trained model and then training it for few epochs. Moreover, due to this problem, ResNet-50 model is used as feature extractor.

Since the evaluation metric is mAP, the hyper parameters of the model are tuned in order to obtain the highest mAP as possible.

In Chapter 2, the preprocessing technique, the model architecture and the fixed hyperparameters are discussed. Moreover, the speed benchmark obtained by the model is presented. In Chapter 3, the results obtained by using the trained model on the test set are illustrated. In Chapter 4, important findings are reported.

The code of the project has been uploaded to Github at the repository [3].

2. Methodology

2.1 Data Preprocessing

This project uses the The Street View House Numbers (SVHN) Dataset [9]. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting (Figure 2.1 displays some example images from CelebA dataset). It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

Figure 2.1: Images from the Street View House Numbers (SVHN) dataset.

The dataset provided for this homework is stored in a *.mat* file. At first, I used a *pandas* dataframe to extract and store the features of each image (i.e., height, image name, label, left, top, width, cut image, image, image height, image width). Then, I created an annotation file for each image using the Pascal Voc format [8]. This annotation format standardises image data sets for object class recognition and offers many advantages, such as it provides a common set of tools for accessing the data sets and annotations.

```
<annotation>
  <filename>1.png</filename>
  <size>
    <width>741</width>
    <height>350</height>
    <depth>3</depth>
  </size>
  <object>
    <name>1</name>
    <bndbox>
      <xmin>246</xmin>
      <ymin>77</ymin>
      <xmax>327</xmax>
      <ymax>296</ymax>
    </bndbox>
  </object>
  <object>
    <name>9</name>
    <bndbox>
      <xmin>323</xmin>
      <ymin>81</ymin>
      <xmax>419</xmax>
      <ymax>300</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 2.2: Pascal Voc annotation file for image "1.png" of the training set.

When using the model to make predictions, the images considered are first preprocessed. In particular, the ImageNet mean is subtracted from each image, by scaling the pixels between -1 and 1. Moreover, each image is resized such that the size is constrained to 416 and 448.

2.2 Tuning Hyperparameters

The model has been trained for 10 epochs, using Adam optimizer [1] with learning rate equal to 0.0001. The batch size is fixed to 64. The number of iterations per epoch is equal to the size of the training set divided by the number of epochs. By fixing these hyperparameters, the computational time required by the model to train is about 10 hours running the process on Google Colab [5]. The minimum score threshold for a box to be visualized has been variated in from 0.1 to 0.4. Further discussion about this and which value works best can be found in Chapter 4.

2.3 Model

Since this homework requires to build an accurate and fast image detector, I chose RetinaNET detector [2] based on ResNet-50 model as deep feature extractor. RetinaNET is a state-of-the-art real-time object detection system. Many of object detection systems need to go through the image more than one time in able to detect all the objects in the image. On the contrary, RetinaNET only needs to look once at the image to detect all the objects in it and this is the reason why RetinaNET is a very fast model.

There exists more deeper feature extractor models, such as ResNet-101 and ResNet-152. The choice of using ResNet-50 model as deep feature extractor is due to the need for more computational power for training the deeper models. Moreover, to deal with this problem I also decided to used a pretrained model (see Section A.1 for the reference), fix some parameters (as specified in Section 2.2) and continue the training. The results obtained by such method are presented in Chapter 3.

2.4 Speed Benchmark

```
[2]
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
[33]
from keras_retinanet import models
from keras_retinanet.utils.image import preprocess_image, resize_image
import cv2
import os
import numpy as np
import pandas as pd
import warnings
# pip install keras-retinanet
warnings.filterwarnings('ignore', category=DeprecationWarning)
warnings.filterwarnings('ignore', category=FutureWarning)
```

```
[34]
def load_inference_model(model_path=os.path.join('snapshots', 'resnet.h5')):
    model = models.load_model(model_path, backbone_name='resnet50')
    model = models.convert_model(model)
    return model
```

```
[35]
# load model
print('Loading model')
model = load_inference_model('/content/gdrive/My Drive/resnet.h5')

# load test images
print('Loading test image')
image = cv2.imread('/content/gdrive/My Drive/SVHNtest/1.png')

# preprocess image for network
print('Preprocessing test image')
draw = image.copy()
draw = cv2.cvtColor(draw, cv2.COLOR_BGR2RGB)
image = preprocess_image(image)
image, _ = resize_image(image, 416, 448)
```

```
Loading model
Loading test image
Preprocessing test image
```

```
[36]
def infer(image):
    # process image
    boxes, scores, labels = model.predict_on_batch(np.expand_dims(image, axis=0))
```

```
# testing inference time
print('Testing inference time')
%timeit infer(image)
```

```
Testing inference time
The slowest run took 145.99 times longer than the fastest. This could mean that an intermediate result is being cached.
1 loop, best of 3: 80.8 ms per loop
```

Figure 2.3: Model speed benchmark. The model inference speed for performing the object detection task on image "1.png" of the test set is 80.8 ms. Note that before performing the detection, the image goes through the preprocessing steps described in Section 2.1.

```
[40]  1 # load model
      2 print('Loading model')
      3 model = load_inference_model('/content/gdrive/My Drive/resnet.h5')
      4
      5 # load test images
      6 print('Loading test image')
      7 image = cv2.imread('/content/gdrive/My Drive/SVHNtest/2.png')
      8
      9 # preprocess image for network
     10 print('Preprocessing test image')
     11 draw = image.copy()
     12 draw = cv2.cvtColor(draw, cv2.COLOR_BGR2RGB)
     13 image = preprocess_image(image)
     14 image, _ = resize_image(image, 416, 448)

    Loading model
    Loading test image
    Preprocessing test image

[41]  1 def infer(image):
      2     # process image
      3     boxes, scores, labels = model.predict_on_batch(np.expand_dims(image, axis=0))

[42]  1 # testing inference time
      2 print('Testing inference time')
      3 %timeit infer(image)

    Testing inference time
    The slowest run took 283.42 times longer than the fastest. This could mean that an intermediate result is being cached.
    1 loop, best of 3: 52.7 ms per loop
```

Figure 2.4: Model speed benchmark. The model inference speed for performing the object detection task on image "2.png" of the test set is 52.7 ms.

```
[43]  1 # load model
      2 print('Loading model')
      3 model = load_inference_model('/content/gdrive/My Drive/resnet.h5')
      4
      5 # load test images
      6 print('Loading test image')
      7 image = cv2.imread('/content/gdrive/My Drive/SVHNtest/3.png')
      8
      9 # preprocess image for network
     10 print('Preprocessing test image')
     11 draw = image.copy()
     12 draw = cv2.cvtColor(draw, cv2.COLOR_BGR2RGB)
     13 image = preprocess_image(image)
     14 image, _ = resize_image(image, 416, 448)

    Loading model
    Loading test image
    Preprocessing test image

[44]  1 def infer(image):
      2     # process image
      3     boxes, scores, labels = model.predict_on_batch(np.expand_dims(image, axis=0))

[45]  1 # testing inference time
      2 print('Testing inference time')
      3 %timeit infer(image)

    Testing inference time
    The slowest run took 277.18 times longer than the fastest. This could mean that an intermediate result is being cached.
    1 loop, best of 3: 60.4 ms per loop
```

Figure 2.5: Model speed benchmark. The model inference speed for performing the object detection task on image "3.png" of the test set is 60.4 ms.

3. Results

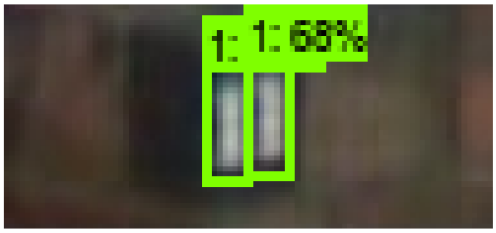
(a) Test image 1

(b) Test image 2

(c) Test image 3

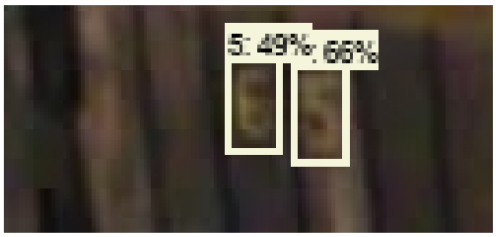
(d) Test image 4

(e) Test image 5

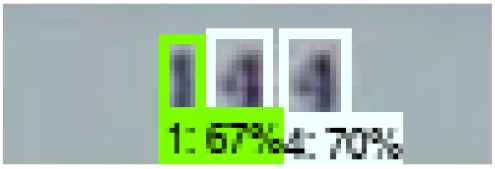


(f) Test image 6

(g) Test image 7



(h) Test image 8



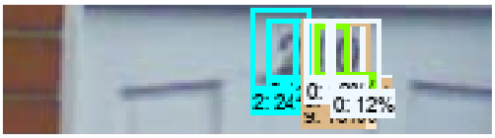
(i) Test image 9

(j) Test image 10

Figure 3.1: Predictions generated by the model after being trained for 10 epochs. The images are taken from the test set. The inference speed of each image is on average about 60 ms. The minimum score threshold for a box to be visualized is equal to 0.4.

(a) 1.png, threshold = 0.1

(b) 1.png, threshold = 0.4



(c) 2.png, threshold = 0.1

(d) 2.png, threshold = 0.4

Figure 3.2: Predictions generated by the model after being trained for 10 epochs. The images are taken from the test set. The inference speed of each image is on average about 60 ms. The minimum score threshold for a box to be visualized is equal to 0.4 in images (b) and (d), equal to 0.1 in images (a) and (c). From the images, it is clear that the lowest threshold makes the model to produce too many bounding boxes.

4. Findings

In this document, I have presented and described the RetinaNET model build and trained for Homework 3. Each choice made for building and training the mode has been discussed and motivated.

Due to the limited available computational power, I chose to use a pre-trained model and train it for only a few epochs. While adjusting the parameters and testing the model, I found out two important tricks that resulted fundamental for achieving mAP equal to 0.54501.

First, each time a box is predicted, I decided to approximate the generated coordinates to the smallest integral value greater than the predicted coordinate (by using function *math.ceil*). This choice has demonstrated to be really effective, since it made the mAP score increase from 0.40829 to 0.47896. I have also tried to round up the generated coordinates to the closest integral value (by using function *round*), but it did not improve the mAP score.

Second, I variated the minimum score threshold for a box to be visualized from 0.1 to 0.4. The most accurate predictions are obtained using 0.4 as threshold, obtaining mAP equal to 0.47896. But, the highest mAP is obtained using 0.1 as threshold, obtaining mAP equal to 0.54501 (see Figure 3.2). This may be due to the evaluation metric used for this challenge.

To conclude, the presented RetinaNET model is capable of detecting digits with impressive performances, achieving a mAP equal to 0.54501 in about 60 ms on average. For futher work, more computational power is needed so that the RetinaNET model can be trained from scratch and a deeper feature extractor model can be used, such as ResNet-152.

APPENDIX

A.1 Credits

The following GitHub Repositories have helped the development of this project:

- Pavitrakumar78 [6], the python file "construct_datasets.py" has been used to parse the .mat file and to create the annotations
- Fizyr [4], the entire repository has been imported and widely used to create and train the RetinaNET model
- Penny4860 [7], the pre-trained model has been taken from this repository

A.2 GitHub Repository

The code of the project is available on GitHub, in the repository [3].

Bibliography

[1] Kingma D., and Ba J. *Adam: A Method for Stochastic Optimization.* Conference paper at ICLR, 2015.

[2] Lin T., Goyal P., Girshick R., He K. and Dollar P. *Focal Loss for Dense Object Detection.* Facebook AI Research (FAIR), 2018.

Sitography

[3] AlessandroSaviolo Github Repository (2019), last access 25th Nov. 2019, [Online].
https://github.com/AlessandroSaviolo/CS_IOC5008_0845086_HW3

[4] Fizyr Github Repository, last access 16th Nov. 2019, [Online].
https://github.com/fizyr/keras-retinanet

[5] Google Colab (2015), last access 4th Nov. 2019, [Online].
https://colab.research.google.com

[6] Pavitrakumar78 Github Repository, last access 16th Nov. 2019, [Online].
https://github.com/pavitrakumar78/Street-View-House-Numbers-SVHN-Detection-and-Classification-using-CNN

[7] Penny4860 Github Repository, last access 16th Nov. 2019, [Online].
https://github.com/penny4860/retinanet-digit-detector

[8] The PASCAL VOC project (2005), last access 25th Nov. 2019, [Online].
http://host.robots.ox.ac.uk/pascal/VOC/

[9] The Street View House Numbers (SVHN) Dataset (2011), last access 17 Nov. 2019, [Online].
http://ufldl.stanford.edu/housenumbers/