Text Detection and Recognition

Max Nihlén Ramström
920429-3055
maxra@kth.se
Supervisor: Professor Yoshiaki Yasumura

Abstract

In this project, a text detection and recognition system was developed using a depthwise convolutional neural network together with the Tesseract software package. Transfer learning was used to train one of Google's pre-trained models to detect bounding boxes around text in images. The detected areas were then fed into the Tesseract text recognition engine. The detection model was also ported to an android application. The system provides good estimates of the bounding boxes while it was found that Tesseract is heavily dependent on image quality.

1 Introduction

With an ever growing amount of robots, self driving cars and other intelligent systems in the world, the need for applications which can help these programs perceive the world around them increases. For a translation system working with real life images, it is not simply enough to know how to translate words between different languages, the system must first determine where in the picture the text is located and then recognize each letter and connect them into words.

When detecting different objects in images one has to take into account that the computer perceives the image in a very different way than humans do. To us a slight difference in position of a text, a different color or light intensity might not make much of a difference in our ability to detect and read the text while to a computer such small perturbations might make the image look like something completely different. One of the most powerful approaches to detect objects in images is deep convolutional neural networks.

Convolutional neural networks, from here on referred to as CNNs, excel at localizing objects and have the benefit of being translation invariant[1].CNNs extracts useful features from the image and combines these features throughout its layers to in the end determine if an object has been detected or not. By changing the network structure slightly it is possible to not only detect the presence of an object but to also infer a so called bounding box which encapsulates the detected object. To recognize the text in the image one can for example look for certain characteristics of different letters as has been the most common approach in the past. One system utilizing this scheme is the so called Tesseract Optical Character Recognition engine[2].

In this project an attempt to train a CNN to perform the regression task of finding the bounding boxes around text was made using TensorFlow. To restrict the scope of the project it was assumed that only one text object would be present in each image. This constraint made it necessary to construct a custom dataset since existing datasets for text detection, such as COCO[3], contains several objects in each image meaning that more complex network structures such as YOLO[5] need to be used. The custom dataset was created using the IIIT 5K-word[6] dataset together with personal vacation pictures. It was found that the model performed poorly and a change was made to instead use transfer learning to fine tune one of Tensorflow's pre-trained CNN models. There were plans to try the system on a mobile device from the beginning and the choice of model thus fell on the SSDMobileNet which is a combination of SSD[7] and MobileNets[8], especially made to run quickly on smartphones. After the model had been trained it was combined with the Tesseract engine and also exported into an Android application.

1.1 Contribution

The work presented in this report offers a way of performing Text Detection and Recognition using open source software in an easy to implement fashion.

2 Theory

2.1 SSDMobileNet

The theory behind the SSDMobilenet will be briefly covered by explaining each part of the architecture, namely, the SSD and MobileNets.

2.2 SSD

SSD is an abbreviation for Single Shot MultiBox Detector and the algorithm was first proposed in the paper by W. Liu et al.[7].

The SSD is a feed-forward convolutional network which creates multiple feature maps out of the input data in different scales as the data is propagated through the network. In each of the feature map's cells a small set of default boxes of different aspect ratios is evaluated and a score for the presence of objects within each box is determined. In the end a non-maximum suppression step is applied to extract the most likely bounding box. The architecture can be seen in figure 1.

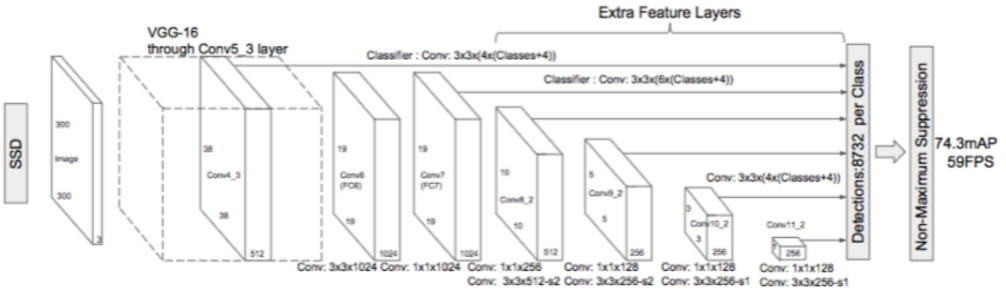
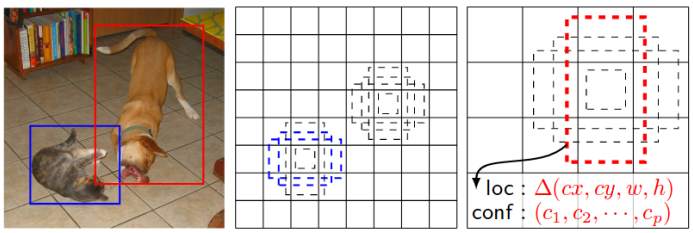


Figure 1: SSD Architecture[7]

In figure 2 we can see an example of two feature maps in different scales, the default boxes of two cells in a feature map as well as the confidence for the object categories.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 2: SSD Multibox[7]

2.3 MobileNets

MobileNets was presented by A.G. Howard et al.[8] at Google and is a convolutional network which makes use of so called Depthwise Separable Convolution. It is especially designed to be fast which enables it to be run on mobile devices, hence its name.

Depthwise Separable Convolution is a form of factorized convolutions meaning that it factorizes a standard convolution into a depthwise convolution together with a pointwise convolution which is a 1x1 convolution.
The depthwise convolution differs from regular convolution in that it applies a single filter to each input channel. The outputs from these filters are then combined through the pointwise convolution. In the paper[8] they show that by doing this the amount of computation needed is reduced by 8 to 9 times and the amount of parameters which has to be trained is reduced by the same amount. The procedure can be seen in figure 3.

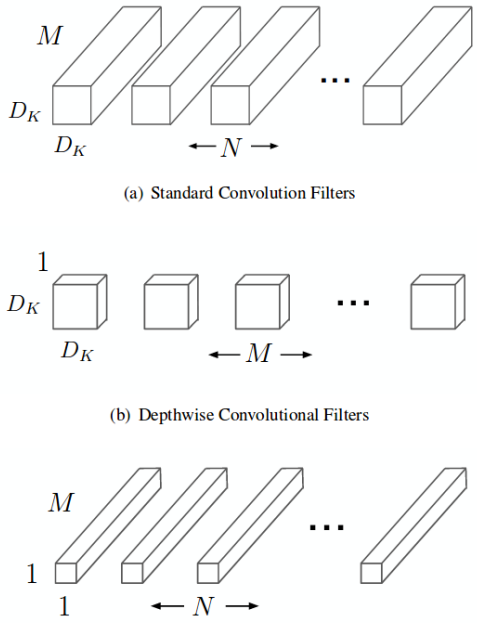


Figure 3: Depthwise Separable Convolution[7]

2.4 Tesseract

Tesseract is an OCR engine originally developed by HP. The following description closely follows the review of the system done by Ray Smith[2].

Tesseract follows a step-by-step pipeline to identify text in images. In the first step it performs a connected component analysis which stores the outlines of the components which are then gathered together into blobs. The blobs are then organized into text lines which are analyzed for fixed pitch or proportional text. Depending on the character spacing, the text lines are broken into words where fixed pitch text is chopped by character cells while proportional text is broken into words using definite spaces and fuzzy spaces. Now an attempt is made to recognize each word in turn. The satisfactory words are sent to an adaptive classifier as training data and the adaptive classifier then tries to more accurately recognize text further on. This recognition phase is run twice to give the classifier another chance to recognize words after having being trained on the entire text. Finally, the algorithm tries to resolve fuzzy spaces and check alternative hypotheses to locate small-cap text.

3 The Dataset

The dataset was created by combining the IIIT 5K-word dataset[6] with a set of vacation images displaying various backgrounds and scenarios. The IIIT 5K-word dataset provides images of text and corresponding labels indicating positions of individual letters as well as the full text in the image. Two example images can be seen below:

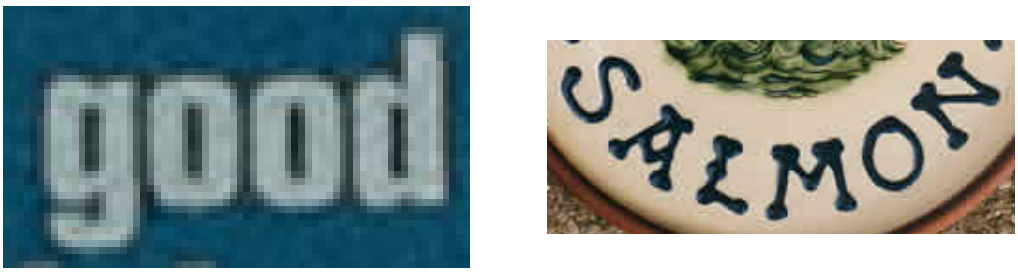


Figure 4

The vacation images were all taken by a mobile camera and had various sizes, in order to speed up training they were all re-sized to 128x128x3 pixels. To create a larger dataset new images were created by horizontally flipping the original images and then vertically flipping both the new and old images creating a dataset four times as big.

The images from the IIIT 5K-word dataset were then pasted upon the vacation images, one in each image and at random positions but held horizontally aligned. The pasted image was also slightly blurred to smooth the transition between the original image and the text box. The position where the text image was pasted was saved as the ground truth for the bounding box in the form

$$[y_{min}, x_{min}, y_{max}, x_{max}] \tag{1}$$

Two examples of the produced images can be seen in figure 5:

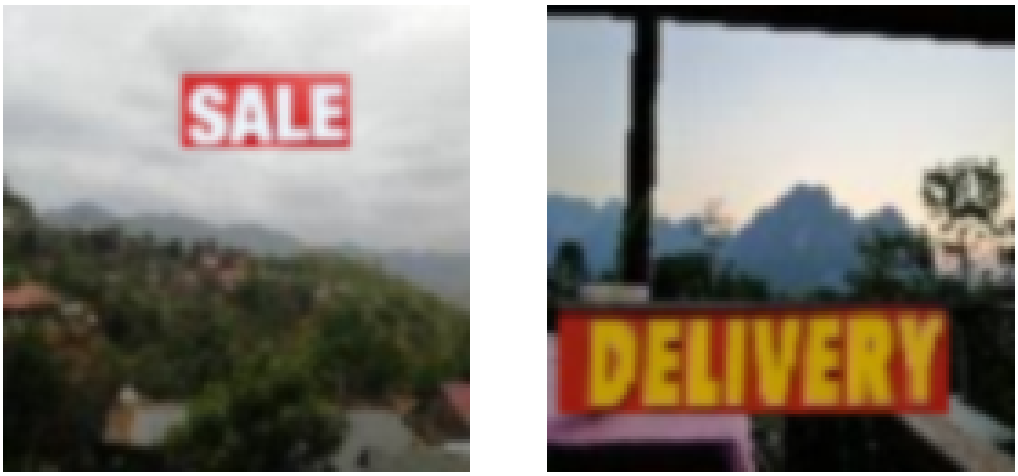


Figure 5

Finally the images and the labels were converted into TFRecord format which is a requirement for doing transfer learning using Tensorflow's Object Detection API.

4 Methodology

4.1 CNN - from scratch

When implementing the CNN from scratch in Tensorflow a network with five convolutional layers and two densely connected layers all with ReLU activations was used with a l_2 loss as described in the paper by J. Yu et al.[9].

4.2 Transfer Learning

Transfer learning means that something that has been learned in one setting(say distribution P_1) is utilized to improve generalization in another setting (say distribution P_2)[1]. The TensorFlow Object Detection API[4] offers the previously explained SSDMobilenet which has been trained on the COCO dataset. This pre-trained model can then be fine-tuned to be able to detect text objects in images and calculate the corresponding bounding boxes. To do this one has to change the number of available classes to also incorporate the new classes to be learned.

The model is then trained on the new dataset for 3000 steps using the RMSProp optimizer, batch normalization, ReLu activations, l_2 regularization for the box predictions, exponential decay on the learning rate, a batch size of 48(largest which could fit in memory) and focal loss for the classification task which focuses on hard, misclassified examples[10].

After the model has been trained it is saved as a frozen graph which can be used for inference on new data.

4.3 Text Detection and Recognition Pipeline

The frozen inference graph is used to infer a bounding box for the text found in an image. The area of the image inside the bounding box is then sliced out of the image and fed into the Tesseract software which produces the final output in the form of the text found in the image. The outline for the pipeline can be seen in the pseudo-code below where $Text\_Detector$ is a class containing the frozen inference graph with a method which can detect and classify objects in the sent in image(provided by the Tensorflow API):

Algorithm 1 Detects and Recognizes text in an image

```
1: procedure GetText(Img)
2:     img ← open(Img)
3:     detector = Text_Detector()
4:     box = detector.get_classification(img)
5:     y_min = box[0]
6:     x_min = box[1]
7:     y_max = box[2]
8:     x_max = box[3]
9:     get_text = img.crop((x_min, y_min, x_max, y_max))
10:    text = pytesseract.image_to_string(get_text, lang =' eng')
11: end procedure
```

4.4 Android

The Tensorflow Object Detection API contains an Android project which can be used to port the trained model unto an Android mobile device. Detailed steps can be found on the Tensorflow Object Detection API website[4]. The application was run on a Huawei Honor 9.

5 Results

5.1 CNN - from scratch

The loss function displayed a peculiar periodic behaviour no matter for how many epochs the model was trained. A plot of the loss function over one such training session can be seen below:

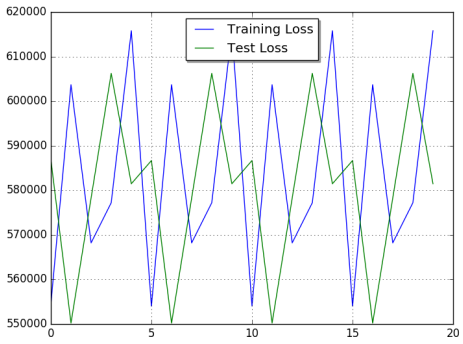


Figure 6: Final Loss

5.2 Transfer Learning

These are the plots of the classification loss, the localization loss and the total loss after 3000 steps.

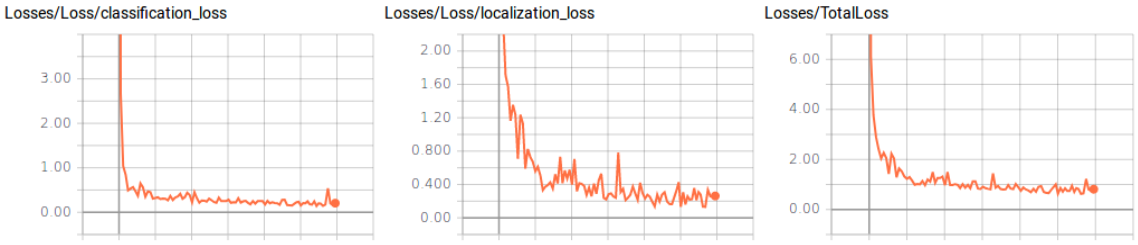


Figure 7: Final Loss

An example of the detection and recognition of text in a random picture taken from the Internet:

Figure 8: Text Detection

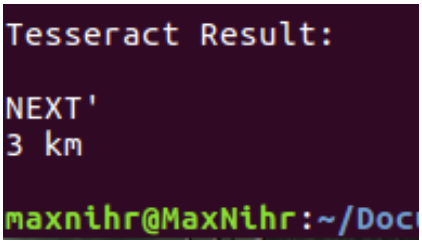


Figure 9: Text Recognition

And another example with more text:

Figure 10: Text Detection

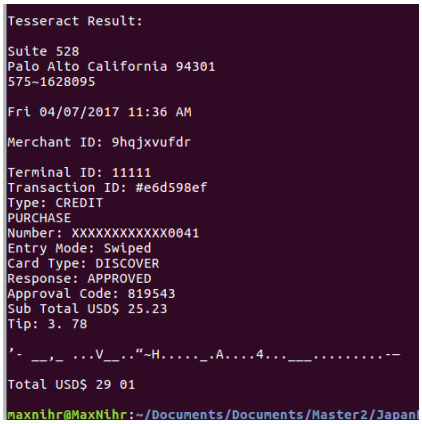


Figure 11: Text Recognition

5.3 Android Results

The following images shows screenshots of the text detection model running on a Huawei Honor 9. The application detects text objects in real time using the smartphone's video camera:

Figure 12

Figure 13

6 Summary and Conclusions

A text detection and recognition system has been implemented using TensorFlow and the Tesseract engine in order to detect and recognize text in real life images. The text detection model was ported to an Android application which detects text objects in real time from video.

It was found that the model performed well when finding the regions containing text in an image but it was much better at doing so when the text was contained in a surrounding frame. This is most likely due to the fact that the training dataset was constructed in such a fashion where all text was inside a box often of different color as compared to the background. To increase the generalization ability of the network one could thus include training samples containing free text pasted onto other images. The default boxes used for prediction in the SSD model are all horizontally aligned and their predicted boxes are thus not as good if the text is tilted. There are also no multi-line texts in the training set while real life text often comes in such a format, including such data in the training dataset should thus be considered in future work.

The reason for using the TensorFlow model with focal loss was to try out a new loss function but it is possible that it was ill suited for this particular task. The focal loss focuses on hard, misclassified examples and it turns out that there are quite a few samples in the training data which are really hard to classify, even for a human. Some of the images in the IIIT 5K-word dataset consist of a single letter and often end up becoming very small when pasted unto the other down-sized image. Instead of neglecting the poor, non-representative samples the loss function instead forced the network to do the exact opposite which might have led to a worse generalization ability.

As for Tesseract, it is known for producing very poor quality of its output if the images fed to it does not look in a specific way, meaning that a lot of pre-processing often is necessary. The project was initially meant to be about building an end to end text detector and recognizer and there was thus some reluctance to tweak the images too much just to boost the performance of an external package(Tesseract). Good results were obtained when the text was located on flat surfaces with a clear distinction from the background, such as in business cards or a regular page in a book. Tesseract would however often fail if the image had text in different sizes and fonts.

An explanation for the periodic behaviour of the loss function in the first

attempt to train the CNN was never found which was the reason for the abandonment of that approach in the first case. Several other loss functions such as l_1 and the IoU[9] loss was also tested with similar results.

In future work it would be interesting to investigate how the performance would change if Tesseract was replaced by a recurrent neural network, such as a LSTM, which could be connected directly to the CNN so that the CNN could extract a context from the image which could be fed as the initial hidden state into the LSTM.

References

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville *Deep Learning* MIT Press, http://www.deeplearningbook.org, 2016

[2] Ray Smith *An Overview of the Tesseract OCR Engine* Google Inc.

[3] `http://cocodataset.org/`

[4] `https://github.com/tensorflow/models/tree/master/research/object_detection`

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi *You Only Look Once: Unified, Real-Time Object Detection* University of Washington, Allen Institute for AI, Facebook AI Research

[6] Mishra, A. and Alahari, K. and Jawahar, C. V. *Scene Text Recognition using Higher Order Language Priors* BMVC, 2012

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg *SSD: Single Shot MultiBox Detector* UNC Chapel Hill, Zoox Inc., Google Inc., University of Michigan, Ann-Arbor

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* Google Inc.

[9] Jiahui Yu, Yuning Jiang, Zhangyang Wang, Zhimin Cao, Thomas Huang *UnitBox: An Advanced Object Detection Network* University of Illinois at Urbana Champaign, Megvii Inc

[10] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollar *Focal Loss for Dense Object Detection* Facebook AI Research (FAIR)