Character Recognition in Natural Scene Images

Luke Morson

December 2018

Abstract

This project aims to classify single characters in natural scenes using a deep learning approach. Classification of single character images in natural scenes is often difficult due to the range of backgrounds, fonts, textures and resolutions. Convolutional Neural Networks are particularly proficient at these sort of image classification tasks and will be used in this project to characterise images from the Chars74K dataset.

1 Introduction

1.1 Project Overview

Optical character recognition (OCR) deals with the conversion of images into machine encoded text. There has been a resurgence in interest in OCR in recent years, mainly for digitizing documents to increase the re-usability of data.

There are many applications of OCR and under certain conditions, such as with cleanly scanned book images, OCR is close to being a solved problem. OCR engines like Tesseract[1], which is now open source, have been in development since 1985. There are however a number of interesting challenges where there is still active research such as in character recognition in natural images [2] and handwriting detection.

1.2 Problem Statement

Character recognition in scene images is generally a harder task when compared to a more classical OCR problem such as recognising text from controlled book scans. This is due to the many possible variations in background, texture, font and resolution. Text in natural scenes contains a lot of valuable information, such as street signs or shop names, and an algorithm that is able to do text detection and recognition has a lot of use cases.

A complete text recognition system includes two general tasks: text localisation and word recognition. The text localisation detects the text locations from the image while the word recognition identifies the characters and then the

[1] https://github.com/tesseract-ocr/tesseract
[2] https://bit.ly/2rwi60T

words. This project will focus on the recognition of single characters in scene images.

1.3 Metrics

As there is a class imbalance in the dataset the predictions will be evaluated using the F1 score:

$$\text{F1 Score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

The aim of the project is to correctly classify labelled characters therefore the accuracy will reflect how well the algorithm is able to correctly classify the characters. The accuracy on the training and testing will be plotted over different training iterations steps to evaluate the performance of the model.

2 Analysis

2.1 Data Exploration

The Chars74K dataset contains symbols that can be used for character recognition in natural images. It contains symbols from English with 62 classes with the possible characters of 'A-Z', 'a-z', and '0-9'.[3]

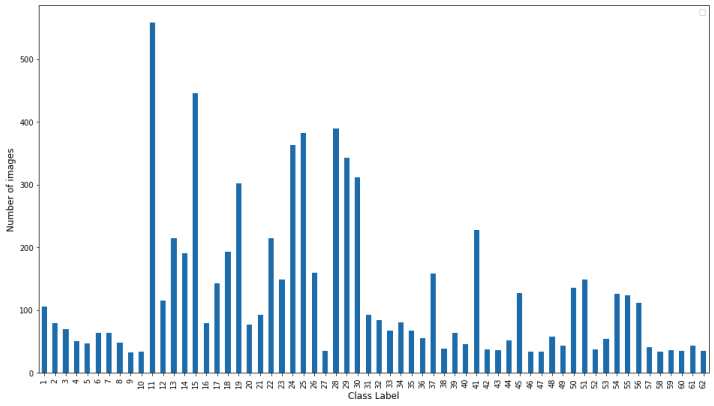
Figure 1: Example of the images from the dataset

The nature of the images can be seen above. The similarity between different images highlights the difficult of this sort of classification problem.

As can be seen below there is not an even distribution between the different classes which means that there should be some attempt to stratify the the input data.

[3] http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/download

Figure 2: Shows the class distribution across the 62 classes in the 7705 characters obtained from natural images in the Chars74K dataset



2.2 Algorithms and Techniques

2.3 Benchmark

This model achieved an accuracy of around 86% using a CNN approach on the Chars74K dataset. The conclusion of this model was that there was little room for improving the accuracy without the risk of overfitting. The main source of errors such as a difference between 0 and O would be difficult for a human to determine without more context, which is not available in this single character dataset. The main area for improvement is identified as improving the classification of rotated characters.

3 Methodology

3.1 Data Preprocessing

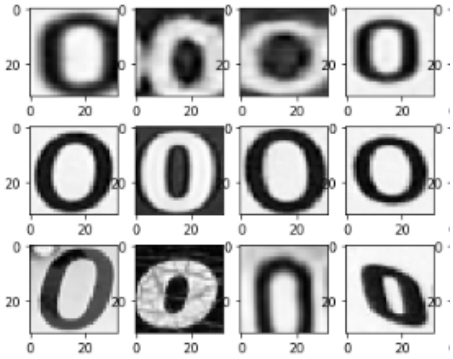
The original images are colour images of varying sizes. The steps taken to preprocess the images are shown below.

1. Images are split into training and test sets in an 90/10 split.[4]
2. Images are read from the array of image paths and flattened to grayscale. It is computationally less expensive to work with a single-channel image. The grayscale image is essentially the average of the RGB image.[5]
3. The images are re-sized to 32*32 pixels. This size should allow for faster training than using images twice the size without too much of a loss in accuracy.

[4] https://cs230-stanford.github.io/train-dev-test-split.html
[5] https://bit.ly/2Ei4Mop

4. The images are saved in new folder to allow them to be viewed.
5. The images are re-scaled by dividing every pixel in the image by 255.
6. The images are saved as a numpy file for faster loading later.

Figure 3: Example of images post processing steps



3.2 Implementation

3.2.1 Data Augmentation

We can apply some augmentation to increase the size of the training set with "augmented" images. These augmentations are random transformations applied to the initial data to product a modified version of it. Data augmentation lets us artificially increase the number of training examples by applying transformations and adding noise.[6]

We use the ImageDataGenerator class from Keras to generate batches of image data with real-time data augmentation. The ImageDataGenerator take a number of paramaters which define different augmentations that can be used. These random transformations prevent overfitting and should help the model generalize better.[7]

3.2.2 CNN Model

CNN are used for problems where we have images as the input because it is often the case that regular fully connected neural networks don't work well. This is because if each pixel is an input this leads an unmanageable number of layers.

In a neural network a convolution operation is performed on the input image matrix to reduce its shape. Convolution introduces some challenges including

[6] https://bit.ly/2KpVTgM
[7] https://bit.ly/2i4ZraH

shrinking output and data loss from the image corners. Padding is used to try and manage these issues.

Another layer is in a CNN is the pooling layer. This condenses the spatial size of the representation to reduce the number of parameters and computation in the network. Max pooling layers take a stack of feature maps as input.

There is another layer, the Dropout layer, which is often used to prevent overfitting by removing some of the nodes. Dropout is a technique where randomly selected neurons are ignored during training.[8]

The CNN architecutre investigated here follows the VGG style convolutional network. The final best network for VGGNet contains 16 CONV/FC layers and an architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. [9].

Figure 4: Outline of the chosen CNN architecture

- Conv2D: 3x3 kernels | 128 maps | init='he_normal' | relu | input shape = (1,32,32)
- Conv2D: 3x3 kernels | 128 maps | init='he_normal' | relu
- MaxPooling2D | 2x2 pool size
- Conv2D: 3x3 kernels | 256 maps | init='he_normal' | relu
- Conv2D: 3x3 kernels | 256 maps | init='he_normal' | relu
- MaxPooling2D | 2x2 pool size |
- Conv2D: 3x3 kernels | 512 maps | init='he_normal' | relu
- Conv2D: 3x3 kernels | 512 maps | init='he_normal' | relu
- Conv2D: 3x3 kernels | 512 maps | init='he_normal' | relu
- MaxPooling2D | 2x2 pool size |
- Flatten layer
- Fully Connected | 2048 units | relu
- Dropout (0.5)
- Fully Connected | 2048 units | relu
- Dropout (0.5)
- Fully Connected | 62 units | softmax

3.2.3 Algorithm and Techniques

As the more complex CNN architecture would not run locally on my CPU I used an Amazon EC2 instance AMI (Amazon Machine Image) to train the model. The configuration and setup of the instance is detailed here.

1. The optimizer Adadelta is used. This has an adaptive learning rate method which means that learning rate decay isn't implemented here.

2. The initialization method He[10] is used to initialize the network as the

[8] https://bit.ly/2PvjQ45
[9] http://florianmuellerklein.github.io/cnn_streetview/
[10] https://arxiv.org/abs/1502.01852

optimization is said to not find the optimum without the correct initialization.

3. Categorical crossentropy is used as the loss function
4. A batch size of 128 is used along with 100 epochs which is shown to be optimum for the combination of parameters used here.
5. The initial data was split 1/5 for validation and 4/5 for learning. Stratification was also used.

3.3 Refinement

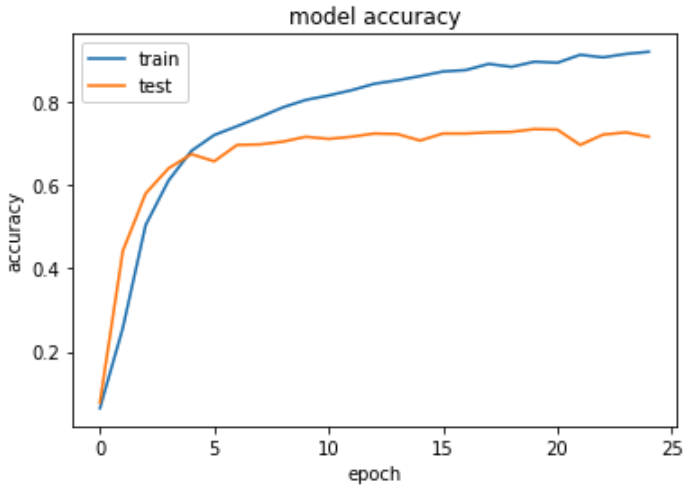
4 Results

4.1 Model Evaluation and Validation

An initial baseline model performance was explored with a simple CNN architecture that could be run on the CPU. With no data augmentation this model achieved an F1 score of around 61% after 15 epochs.

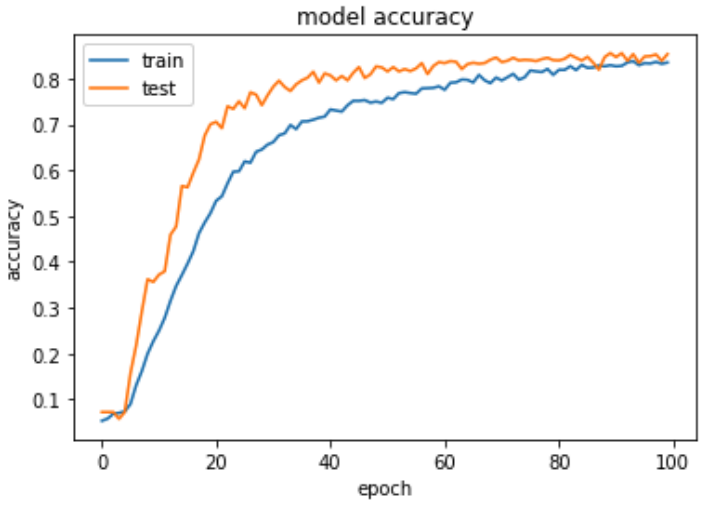
A number of mis-classified images are actually the lowercase/uppercase equivalent such as 'S/s, N/n, c/C'. The model performs well on distinctive characters like 'E' which is no surprise as 'E' has a distinctive pattern that would be well represented by a CNN. The model is shown to be underfitting and it's clear the CNN architecture could be improved in advance of other improvements.

Figure 5: Shows the initial selected model performance which is underfitting



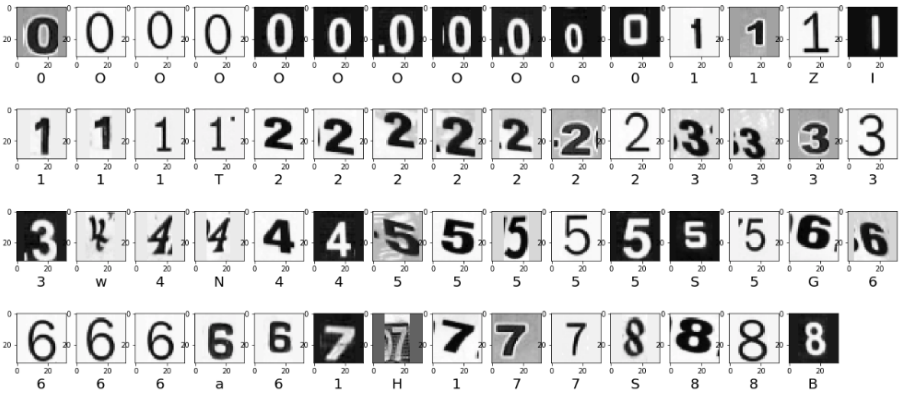
The next model used the CNN architecture above along with data augmentation. This model achieved an F1 score of 72% and an accuracy of 75%. This is an improvement on the previous model.

Figure 6: Shows the performance of the final model



The final model shows better performance but still performs poorly on letters of the same case such as "s/S","O,o" and also on pairs like "I,1". In this case it's likely the model performs worse than a human would at identifying these.

Figure 7: Shows a sample of images with the model's output



4.2 Justification

The model, with an F1 score of 72%, under-performs compared to the benchmark model which achieved a reported 86%. There are obviously improvements that could be made to the implementation shown here. For use in a real-world application or an OCR pipeline this level of accuracy would not be acceptable

to extract enough context from the image. If the performance was evaluated as case-insensitive then the accuracy would be much higher. It does show that a number of iterative improvements can be made to increase the performance and it's likely with more advanced approach and an improved dataset this problem could be solved enough for real-world applications.

5 Conclusion

5.1 Final Remarks

This proved a good project to get a baseline understanding of how to implement a image classification project using CNN. There was a marked improvement on the performance of the model with an improved CNN architecture and the use of data augmentation. It also valuable to learn how a cloud EC2 instance could be configured for deep learning. There are further concepts and deeper understandings that could be explored from this point to further improve on the performance of the model.

5.2 Improvements

I could have built a better framework that allowed for better testing of the models. This would have allowed me to run multiple models with different hyperparamaters in a less manual way to explore how this improved the performance of the model. There are undoubtedly gains in exploring variations in the data augmentation however I ran out of time to explore this. The same applied to using different optimizers and modifying the learning rate. This could have been explored further given more time.

Model performance has also been shown to improve with model averaging or stacking. This could have been explored here.