Machine Learning Engineer Nanodegree

Capstone Project

Kenny Chow

November 14, 2019

## ## I. Definition

## ### Project Overview

Optical Character Recognition (OCR) has been around since early 1900’s. It is a use of computer software to scan graphical objects and recognize them as characters.

OCR has been widely used in postal services in which mailing address is scanned and translated into computer-encoded text such that with minimal human aid, mail / parcels can be sorted by scanning systems. This improves delivery efficiency.

This project uses the vehicle plate template (as defined in the 1994 specs) in Ontario, Canada as a basis. I will create a model, which will be trained on 15245 sample plates with three English characters and three numbers on them. The goal is to train our model such that it is able to find patterns of characters/numbers and be able to recognize them in any given dataset.

### ### Problem Statement

Vehicle license plate recognition is useful for law enforcement purposes. Being able to use machines to read plates is essential for officers to pull vehicle-related information such as overdue tolls and offence history. The question is how to make our model learn to read correctly. In this capstone project, I will be implementing a machine learning program that attempts to recognize vehicle license plates containing numbers and alphabets.

Some examples of license plate look like the following:





With these plates, my solution will give the following output:

Predict: AAA 257

Actual(True): AAA 257

Predict: OMP 011

Actual(True): OMP 011

Predict: ZJW 655

Actual(True): ZJW 655

The ‘Predict’ value is a predicted output from the model; while the ‘Actual’ is the actual label of a given plate. If the ‘Predict’ output is different than the ‘Actual’ value, it is counted as an incorrect prediction.

### ### Metrics

Accuracy is measured using the following formula

## ## II. Analysis

### ### Data Exploration

There is a web tool developed by [ACME License Maker](https://www.acme.com/licensemaker/licensemaker.cgi?state=Ontario&text=&plate=1994) which is used to generate the dataset used in this project. A script is written to request a license plate and save it in a local storage. This process is automated to poll the data every four(4) seconds.

The script is written in JavaScript and can be run in NodeJS. You can find this script in my Github.

The dataset contains 15245 files for training purpose, 8 files for validation and 1981 files for testing.

With numbers and english alphabets combined, there are 36 unique characters in total. Each character (from A-Z) and number (from 0-9) in the plates is randomly generated.

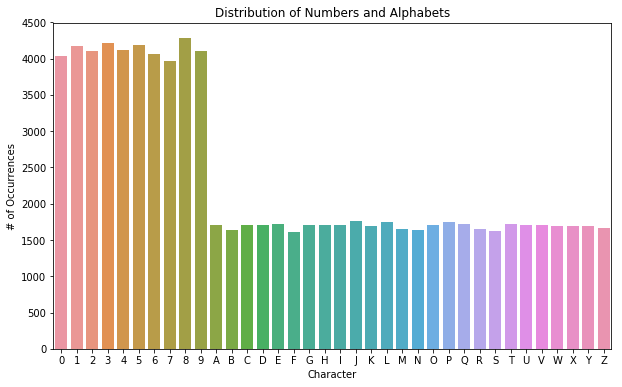
As illustrated below, the distribution of numbers (0-9) is reasonably equal with a minimum of occurrences 3972 and a maximum being 4282 which gives a variance of 310.

For alphabets, the minimum is 1613 and the maximum is 1767, this gives a variance of 154.

| Index | **Character** | **count** |
| --- | --- | --- |
| **1** | 0 | 4032 |
| **2** | 1 | 4169 |
| **3** | 2 | 4110 |
| **4** | 3 | 4209 |
| **5** | 4 | 4124 |
| **6** | 5 | 4191 |
| **7** | 6 | 4060 |
| **8** | 7 | 3972 |
| **9** | 8 | 4282 |
| **10** | 9 | 4101 |
| **11** | A | 1701 |
| **12** | B | 1633 |
| **13** | C | 1705 |
| **14** | D | 1705 |
| **15** | E | 1722 |
| **16** | F | 1613 |
| **17** | G | 1706 |
| **18** | H | 1709 |
| **19** | I | 1709 |
| **20** | J | 1767 |
| **21** | K | 1696 |
| **22** | L | 1748 |
| **23** | M | 1649 |
| **24** | N | 1635 |
| **25** | O | 1702 |
| **26** | P | 1747 |
| **27** | Q | 1727 |
| **28** | R | 1657 |
| **29** | S | 1627 |
| **30** | T | 1721 |
| **31** | U | 1702 |
| **32** | V | 1709 |
| **33** | W | 1698 |
| **34** | X | 1694 |
| **35** | Y | 1693 |
| **36** | Z | 1662 |

### ### Exploratory Visualization

Following the above table, it is clear to visualize that the distribution of the numbers and alphabets combined is reasonably equal. It is important to note that the number of alphabets is 26, twice the number of number (0-9), resulting the bars twice as high in the chart.

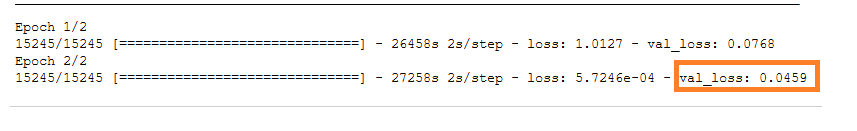


### ### Algorithms and Techniques

|  |  |
| --- | --- |
|  | In order to recognize characters from the images, two Convolutional Neural Networks (CNN) combined with max pooling layers are used to extract features. The Reshape and Dense layer are to reduce dimensions thus reduce computational workload.  The bi-directional Gated Recurrent Unit (two GRU combined with Add/Concatenate) are for processing sequential data, capable of predicting current state from previous time steps or from later time steps. Last is the Dense/Softmax combination that is responsible for decoding and classifying each character. |

### ### Benchmark

### A benchmark for this problem is a classifying model that takes an input of three characters and three numbers in order. Using Stochastic Gradient Descent(SGD) and Softmax, the goal is to decode and classify the characters from the plate. The validation loss from the result of second epoch, is 0.0459 which I would set as the benchmark for this project.



### ## III. Methodology

First, I split all the images into three categories: training, validation and testing. Then I put them into three folders, train, validation and test respectively. I implement a function ImageDataGenerator as an image generator to pull the files from the train folder, one at a time for our model to learn about the pattern of each character. It uses the files in validation to computes prediction accuracy against the current model.

#### #Training

The training process consists of utilization of two Convolutional Neural Networks (CNN), each followed by a Max Pooling layer. Then the output is passed onto four Gated Recurrent Units (GRU) before reaching to the Softmax layer. The design of this architecture follows the example at Keras\*.

#### #Loss function

The loss function used here is the Connectionist Temporal Classification function which is able to recognize text found in the input image. This loss function looks for characters in the image according to number of characters specified by the user and decodes the output into a string of characters.

### ### Data Preprocessing

Each image in this project has three English characters, followed by three numbers. This information is saved in its file name. Given this, the dataset is complete with about fifteen thousand images (training and testing combined) and is ready for us.

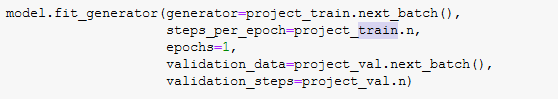
### Implementation

My implementation is built upon the ANPR OCR code base by Supervisely (Vehicle Plate) which is derived from the example of the Kera’s OCR demo\*. I modified the ImageDataGenerator to read the label from its image file name.



The ImageDataGenerator is initialized to read the ‘train’ folder specified by the user. The folder here is the training folder in which plates are stored for training. This generator takes five parameters, file path, width, height, number of characters and downsample\_factor. Each image is 200px wide and 100px high with 7 characters (including a space ‘ ‘) and 4 as the downsample\_factor which later is defined in the train method

The build\_data method executes on the folder specified and builds a list of images along with its labels in a separate array.



Now that we are ready to train our model by feeding the images one by one. It can take up to 7 hours without GPU. The ‘train’ method sets up the parameters required and executes the ‘fit\_generator’ method. This is required because the memory on the machine cannot hold up all training data. Instead, it is done one by one. This method is applicable for training and testing purposes as either process holds around a few thousands of images, which exceeds the memory capacity.

### ### Refinement

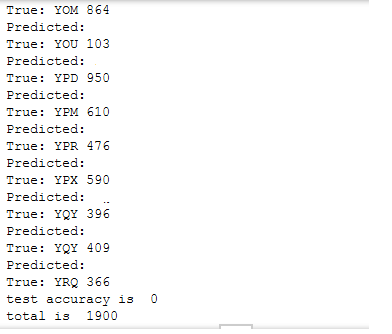
At this stage, I tried taking out the ‘clipnorm’ option and increase learning rate (lr) from 0.02 to 0.03 to see if training would take more or less time.



Before it was,



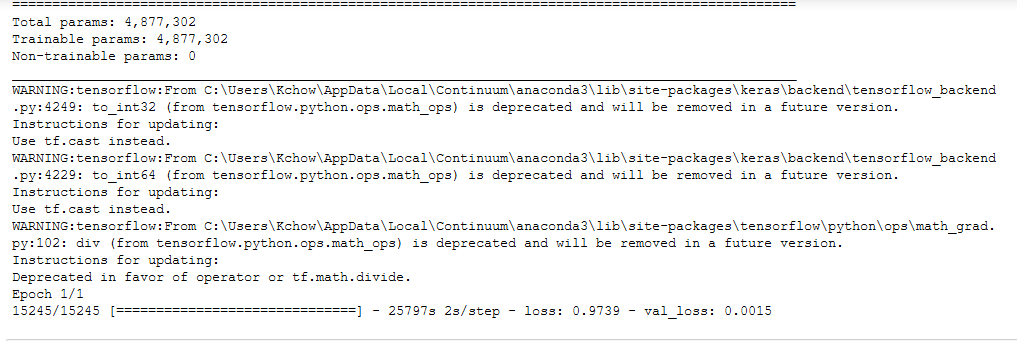
However, it turns out that it cannot predict any image which I cannot figure out why.



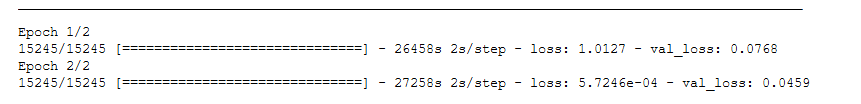
Having ‘clipnorm’ set to ‘5’ makes prediction work again. At this point, keeping the ‘clipnorm’ parameter is important to avoid Exploding Gradients. At this point, I cannot think of any improvement to be made to the algorithm.

#### ## IV. Results

The training took 7 hours to complete for one epoch. The loss value is 0.9739 and the validation loss is 0.0015. This is reasonably a good start with this statistics as it will definitely improve with more epochs.

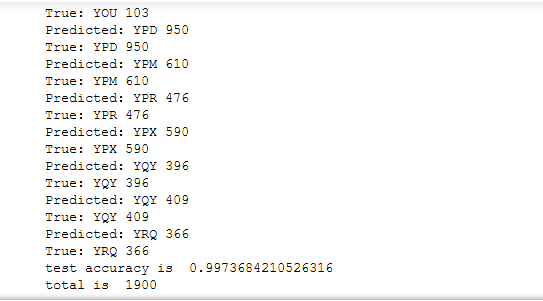


Running it the second time for two epochs, the result looks more satisfying as the loss value drops to 0.00057246. This is a significant drop since we have characters which is less complex than real life images for recognition. However, some may argue the model is overfitting since it is very well trained with the images given. What about the images in the test folder? I am curious too. Let’s find out.

### Model Evaluation and Validation

Testing on the ‘test’ folder, I found that the model is doing a 97.7% accuracy out of 1900 images.

|  |  |
| --- | --- |
|  |  |
|  |  |



The architecture design of the model was discussed in the Algorithms and Techniques section. This model utilized CNN layers with ‘Relu’ as the activation function to allow non-linear classification. This is commonly found in image classification problems. The optimizer used here is Stochastic Gradient Descent with a learning rate(lr) at 0.02, decay at 1e-6, momentum at 0.9, nesterov=True, clipnorm=5.

An important note on the ‘clipnorm’ is that it controls gradient clipping. What it does is that it will clip all parameter gradients to a maximum of 5.

### Justification

Given the accuracy discussed above, it is suffice to confirm the model is not overfitting and is well trained to recognize characters/numbers in the form of three characters, a space, and three numbers. E.g.

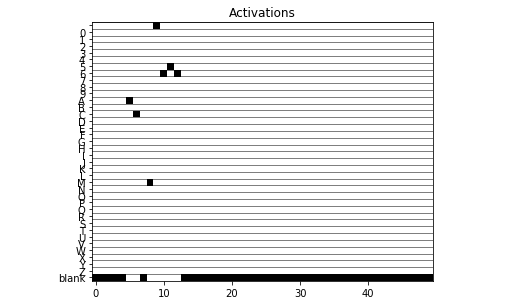
ABC 321

When comparing the resulting accuracy (0.997) with the benchmark (validation loss) 0.0459, that is equivalent to 0.9541 accuracy. My model is slightly higher than the benchmark model.

## V. Conclusion

### Free-Form Visualization





### Reflection

This project uses GRUs and requires the use of the CTC loss function, which are beyond the topics of this nanodegree program. The most challenging part was to find a dataset that is large enough for training, which [Supervisely](https://github.com/DeepSystems/supervisely-tutorials/tree/master/anpr_ocr) was a good starting point. However, the skills and techniques required had to be found elsewhere.

### Improvement

The decode method used here is ‘Best Path’ for simplicity. However, Beam Search with a dictionary are more common in real world applications. At this point, the algorithm implemented is good for recognizing the Ontario vehicle plates on 1994 specs. Plates with different designs are likely to disturb the prediction and transfer training will be required. I intend to go further by extending the project to use Single Shot Detection(SSD) with video streaming as input in a mobile application.

#References:

Ontario license plates: https://www.ontario.ca/page/choose-licence-plate-graphic

Acme - <https://www.acme.com/licensemaker/licensemaker.cgi?plate=1994&state=Ontario&text>=

Keras documentation: Image OCR - <https://keras.io/examples/image_ocr/>

Supervisely: ANPR OCR dataset: <https://app.supervise.ly/projects/36053/datasets>

Supervisely ANPR OCR tutorial: https://github.com/DeepSystems/supervisely-tutorials/tree/master/anpr\_ocr

How to train a keras model to recognize variable length text: <https://www.dlology.com/blog/how-to-train-a-keras-model-to-recognize-variable-length-text/>

Understanding GRU (Gated Recurrent Unit) network: <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>

Connectionist Temporal Classification (CTC) Loss function: <https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c>