# CS771 : Introduction to Machine Learning

## Assignment 3

23rd November 2019

Group 19

Siddharth Jayashankar (170699) Keshav Bansal (170335)

Harsh Agarwal (170287) Snehal Raj (170705)

Durgesh Rajendra Agrawal (170262)

## 1 Problem Statement

Describe the method you used to solve both the problems i.e. counting how many chars are there in the image and then finding out what those characters are in what order. Give all details such as algorithm used including hyperparameter search procedures, validation procedures. You have to give a detailed explanation even if you used an algorithm/implementation from the internet – make sure to give proper credit to the person/source from where you took code/algorithm. There is no penalty for using someone else's code/algorithm but there would be heavy penalty for doing so without giving proper credit to that person/source.

## 2 Pre-processing:

For an input image, we apply a strong Gaussian Blur filter. Since the lines are much thinner than the alphabet, the filter blurs most of the lines with the background but retains approximate positions of the letters.

We compare the blurred image with the original image and wherever the letters show up in the blurred image, we retain the original image. All other pixels are coloured black.

The image formed above is inverted and converted to a binary image.

(a) Original Image

(b) After Blurring

(c) Applying The Mask

(d) Converting to Binary

## 3 Counting the Number of Letters:

The binary image is processed column wise from left to right. The regions in which the number of black pixels cross a threshold (12/150 pixels) are identified as different letters. To ensure that a small spot of black pixels is not mistaken for a letter, we ignore the region if its thickness is less than certain threshold (12 pixels). The count of letters identified in this stage is reported as the count of letters in the image. Since the image is scanned from left to right, the letters identified are identified in sequence.

(a) Letter 1

(b) Letter 2

(c) Letter 3

## 4 Predicting the Code:

After splitting the image into the letters, each image is resized to a $28 \times 28$ image, after padding each image to make it square. The prediction model is invoked on each image and it's output is reported as the interpretation of the letter.

## 5 Charater Recognition with Deep CNN

We followed the tutorial : Yassine Ghouzam, Introduction to CNN Keras, kaggle.com (Link) and made suitable modifications to the hyperparametres while training. (1)

We used Keras sequential API to make our neural network. The layer-wise architecture of our neural network is as follows:

1. Input Layer
2. 2D convolution with ReLu activation function
3. MaxPool 2D Layer
4. Dropout layer
5. 2D convolution with ReLu activation function
6. MaxPool 2D Layer
7. Dropout layer
8. Flatten
9. Dropout
10. Dense
11. Output Layer

Both of the 2D convolution filters use 32 filters which transforms part of the image according to the kernel. The ReLu activation function is used add non-linearity to the network.

The MaxPool layer acts as a downsampling layer. The maxpooling layer picks the maximum valued pixel among 4 neighbouring pixels with a stride of $2 \times 2$

By combining these layers, the CNN is able to learn newer features and use this in the classification.

We used dropout as a regularisation method to prevent overfitting and improve generalisation.

The flatten layer converts the final feature maps into a single 1D vector.

The final dense layer uses the softmax activation function to predict the probability of each class.

## 6 Loss Function, Optimiser and Annealer

We use a specific form for categorical classifications (>2 classes) called the "categorical_crossentropy". It is the error rate between the observed labels and the predicted ones.
We chose RMSprop with L.R. = 0.001, $\rho = 0.9$ $\epsilon = 10^{-8}$ and D.K. = 0.0 as our optimizer function. The RMSProp update adjusts the Adagrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate.

We used the metric function accuracy ( percentage of correct predictions) to evaluate the performance of our model. These results were used only for validation and not for training.

In order to make the optimizer converge faster and closest to the global minimum of the loss function, we used an annealing method of the learning rate (LR).
For fast computation we started with a high LR and decrased it dynamically every X steps (epochs) depending on necessity (when accuracy is not improved).
With the `ReduceLROnPlateau` function from `Keras.callbacks`, We choose to reduce the LR by half if the accuracy has not improved after 3 epochs.

## 7 Augmenting the Data Set:

To avoid over-fitting, we used our existing training to generate some more training data by randomly rotating some images by $10°$, randomly zooming some images by $10\%$, randomly translating some images left or right by $10\%$ and randomly translating some images up or down by $10\%$ of the height.

## 8 Training

We split the original dataset into two parts via a 80-20 split and used held-out validation to perform the training.

### 8.1 Hyperparameter Tuning

We started with the model architecture provided in (1), and pruned the layers to get a model specific to our usecase. We set the number of nodes in the layers as hyperparameters and ran a simple grid search over the probable values. Finally we chose the set of hyperparameters which maximized our validation accuracy while having a small enough model size. The hyperparameters in relation to the model were

- The number of filters in the first convolution layer
- Pool size of the first MaxPool layer
- The number of filters in the second convolution layer
- Pool size of the second MaxPool layer
- Stride length of the second MaxPool layer
- Number of nodes in the fully connected layer

The exact parameters in the original model and the parameters after tuning are shown below

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_9 (Conv2D) | (None, 28, 28, 32) | 832 |
| conv2d_10 (Conv2D) | (None, 28, 28, 32) | 25632 |
| max_pooling2d_9 (MaxPooling2 | (None, 14, 14, 32) | 0 |
| dropout_13 (Dropout) | (None, 14, 14, 32) | 0 |
| conv2d_11 (Conv2D) | (None, 14, 14, 64) | 18496 |
| conv2d_12 (Conv2D) | (None, 14, 14, 64) | 36928 |
| max_pooling2d_10 (MaxPooling | (None, 7, 7, 64) | 0 |
| dropout_14 (Dropout) | (None, 7, 7, 64) | 0 |
| flatten_5 (Flatten) | (None, 3136) | 0 |
| dense_9 (Dense) | (None, 256) | 803072 |
| dropout_15 (Dropout) | (None, 256) | 0 |
| dense_10 (Dense) | (None, 26) | 6682 |

Total params: 891,642
Trainable params: 891,642
Non-trainable params: 0

(a) Old model

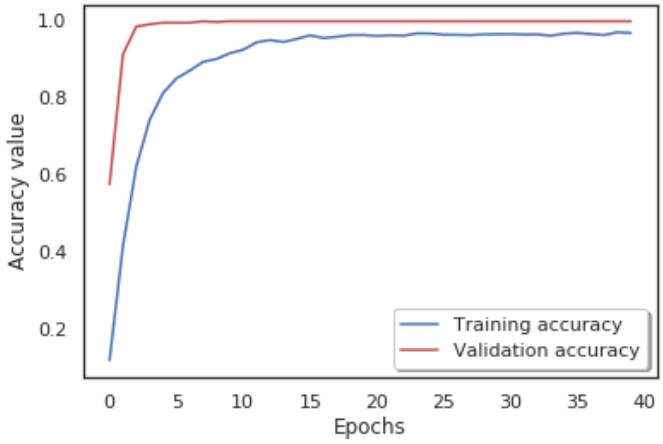
| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_7 (Conv2D) | (None, 28, 28, 32) | 832 |
| max_pooling2d_7 (MaxPooling2 | (None, 14, 14, 32) | 0 |
| dropout_10 (Dropout) | (None, 14, 14, 32) | 0 |
| conv2d_8 (Conv2D) | (None, 14, 14, 32) | 9248 |
| max_pooling2d_8 (MaxPooling2 | (None, 7, 7, 32) | 0 |
| dropout_11 (Dropout) | (None, 7, 7, 32) | 0 |
| flatten_4 (Flatten) | (None, 1568) | 0 |
| dense_7 (Dense) | (None, 128) | 200832 |
| dropout_12 (Dropout) | (None, 128) | 0 |
| dense_8 (Dense) | (None, 26) | 3354 |

Total params: 214,266
Trainable params: 214,266
Non-trainable params: 0

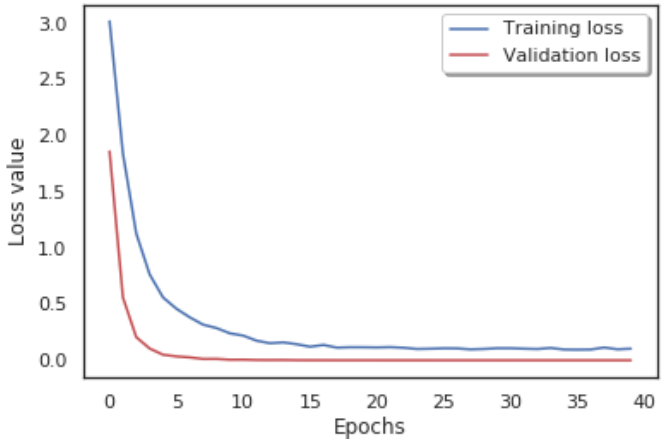
(b) Model after tuning

## 9 Evaluation

We have plotted the training accuracy with the validation accuracy and training loss with the validation loss for the first 40 epochs (with batch size 20) of training.




(a) Accuracy




(b) Loss Function

## 10 Confusion Matrix

We received 100% accuracy on the dataset provided and hence every entry on the matrix was present on the true diagonal of the matrix.

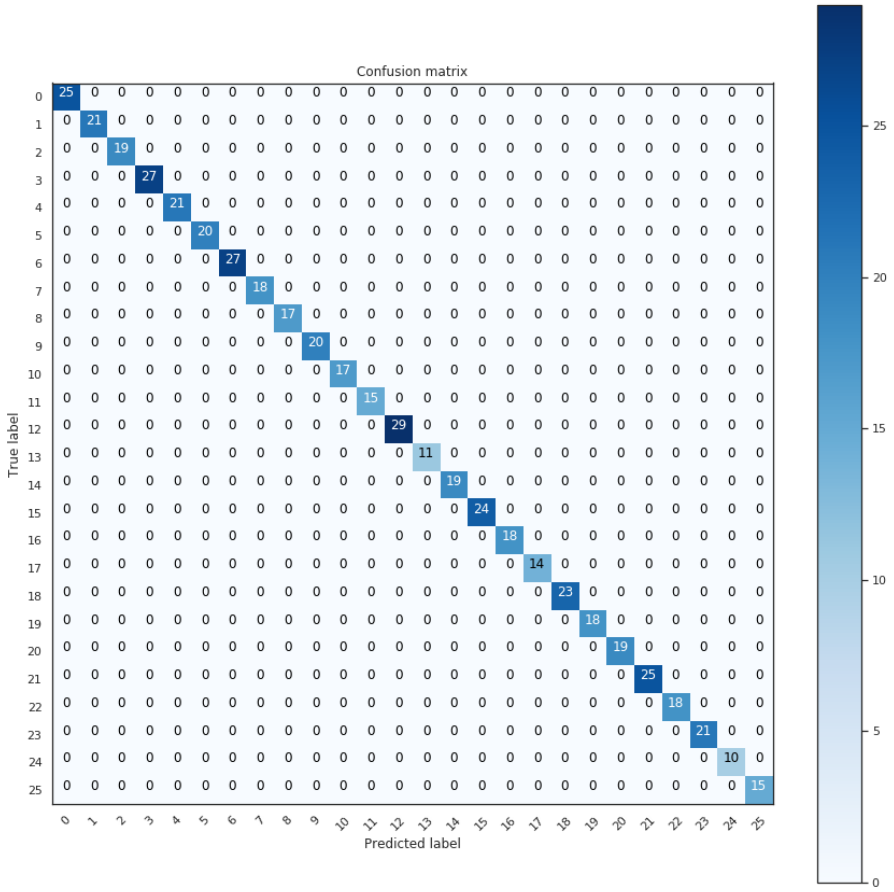



Figure 5: Confusion Matrix

## References

[1] Ghouzam, Yassine. "Introduction to CNN Keras - Acc 0.997 (top 8%)" *Kaggle*, 18 July 2017, https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6.