cs512 Assignment 3: Report

Amit Nikam
Department of Computer Science
Illinois Institute of Technology

November 9, 2020.

Abstract

In this programming assignment 3, I have implemented an application that takes an input image of a handwritten digit and tries to guess if the number is even or odd. Wide range of concepts related to computer vision including state of the art topics like deep learning and some core image transformation concepts were put to test. To achieve the outcome Keras, OpenCV and NumPy were used. In this report I cover my application's complete implementation, testing and the outcomes.

1 Problem Statement - CNN

The goal of this application was to perform a prediction on an input image and predict whether the number is odd or even. This required to build a convolutional neural network as specied in the problem statement provided. The particularly challenging part about this was to get the right architecture, parameters and hyper-parameters such that the model predicts well what the digit is. The application had to be made capable of using with the help of some opencv functionalities.

2 Solution

To understand the solution better, let's breakdown the problem first.

The purpose of this assignment was to predict if a digit is even or odd, which is binary, we were expecting a categorical binary outcome. Thus, the loss function used was 'binary_crossentropy' in the last layer, along with a single unit/node layer with a sigmoid activation. The optimizer used was ADAM but for a while RMSprop was used with a learning rate of 0.001.

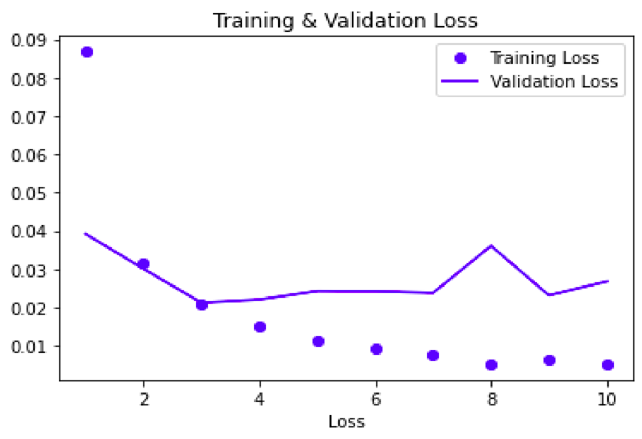
The entire Mnist dataset of 70000 records was merged, preprocessed and split into 55000, 10000, 5000 chunks for training, validation and testing respectively.

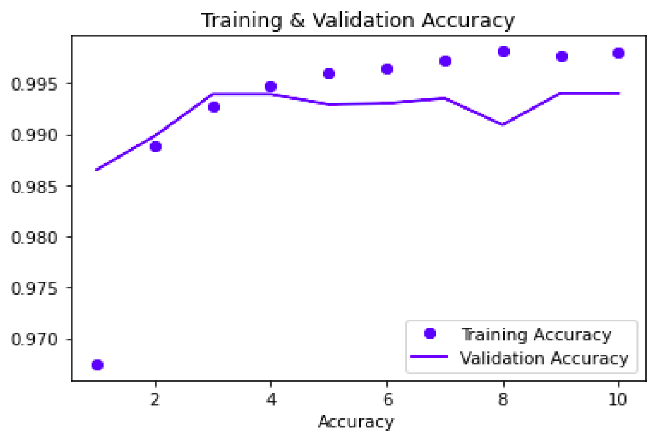
For the neural network itself, the following architecture was used:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 26, 26, 64)        640

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 64)       0

_________________________________________________________________
conv2d_1 (Conv2D)           (None, 11, 11, 128)       73856

_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 128)        0

_________________________________________________________________
flatten (Flatten)           (None, 3200)              0

_________________________________________________________________
dense (Dense)               (None, 128)               409728

_________________________________________________________________
dropout (Dropout)           (None, 128)               0

_________________________________________________________________
dense_1 (Dense)             (None, 1)                 129
=================================================================
Total params: 484,353
Trainable params: 484,353
Non-trainable params: 0
```

Total of 2 convolution layers with pooling, a dropout layer and 2 dense layer were used.

Loss and Accuracy metrics for this model were as follows.





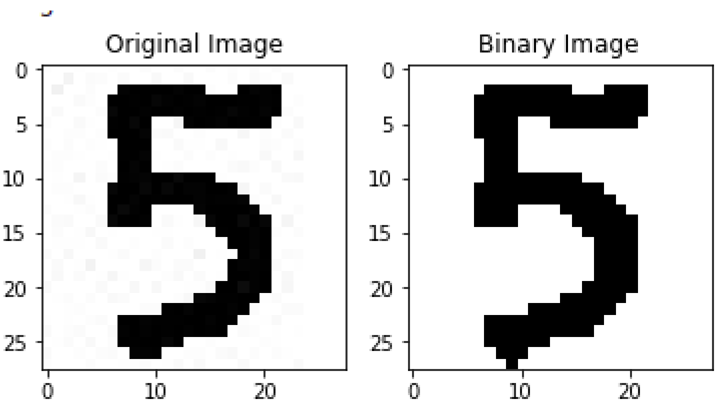
```
Loss at Final Training Step is: 0.005201996769756079
Accuracy at Final Training Step is: 0.9980000257492065
```

After understanding the accuracy and lose as a function of epoches, let’s look at the test loss and test accuracy.

```
157/157 [==============================] - 3s 19ms/step - loss: 0.0127 - accuracy: 0.9974
Test Loss for the model is 0.012691144831478596
Test Accuracy for the model is 0.9973999857902527
```

A total test accuracy of 0.9973 was achieved. The convolution were as deep as 128 in depth. A dropout layer was used in between the FC layers to get better randomization of the variables.

After importing and testing an image preprocessed in opencv, I was able to correctly predict the even and odd numbers.



```
Prediction: odd number with a confidence of 0.9999874033019296
```

Evaluation of the 2nd question of hyper-parameters tuning is shown after the implementation.

3 Implementation Details

The following files are included in src folder:

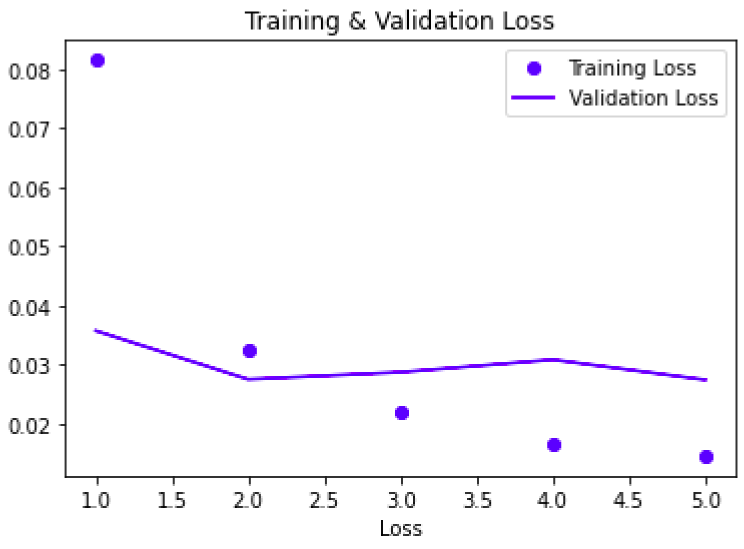
```
- AS3.ipynb
- mnist_even-odd (folder)
      |_ saved_model.pb
      |_ variables (folder)
      |_ assets (folder)
```

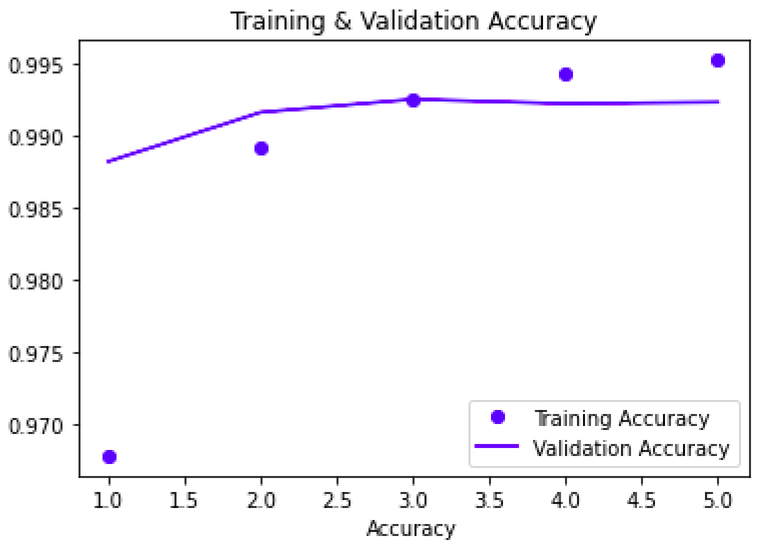
AS3.ipynb is the main program. “mnist_even-odd” happens to be the saved model to load from.

The jupyter notebook imports the model in the second half, do no execute the first half that trains the model to try this.

4 Hyper-parameter Tuning & Results

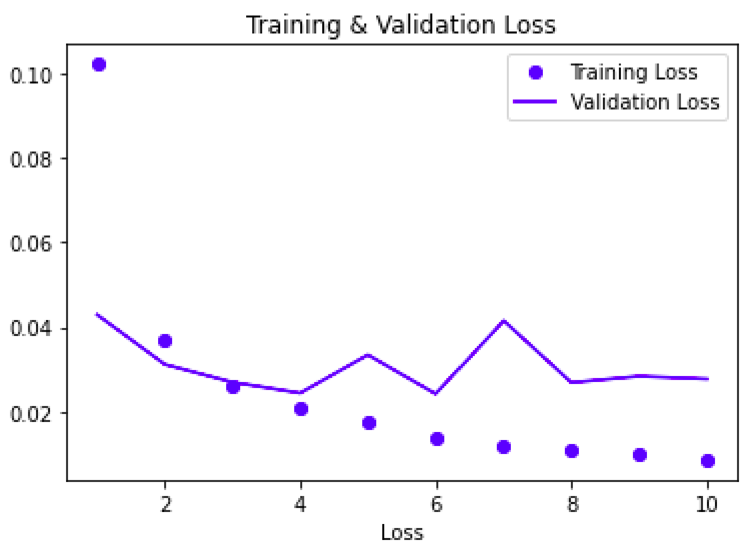
I had to experiment a multiple times until I could finally make a good model. Initially I tried increasing the number of layers or shuffling the layers among themselves where ever possible and got a accuracy of 0.9952. So I tried and experimented with the receiptive filed and stride, and this did not increase the accuracy. This could be due to the already small size of the image.

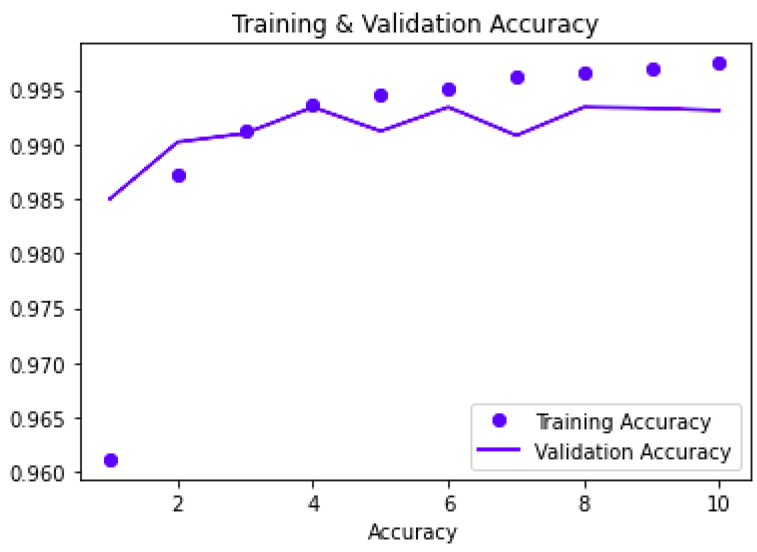




```
Loss at Final Training Step is: 0.014553512446582317
Accuracy at Final Training Step is: 0.9952181577682495
```

Then next I tweaked around with optimizer, I was originally using RMSprop with a learning rate of 0.001. But after switching to ADAM optimizer, I was able to get better results. Furthermore this eliminated the need to set learning rate manually. Although I updated to dropout rate to 0.15 from 0.05 to check what impact it has on the accuracy. And I also went from a 5 epoch approach to 10 epoches and thus the following statistic is for 10 epoches. But it is clear that accuracy went from 0.9952 to a 0.9974 which is why I decided to keep these changes.

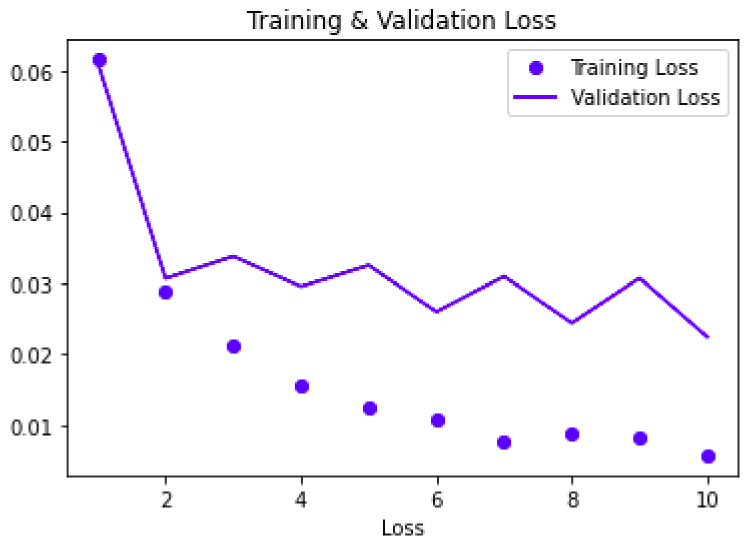


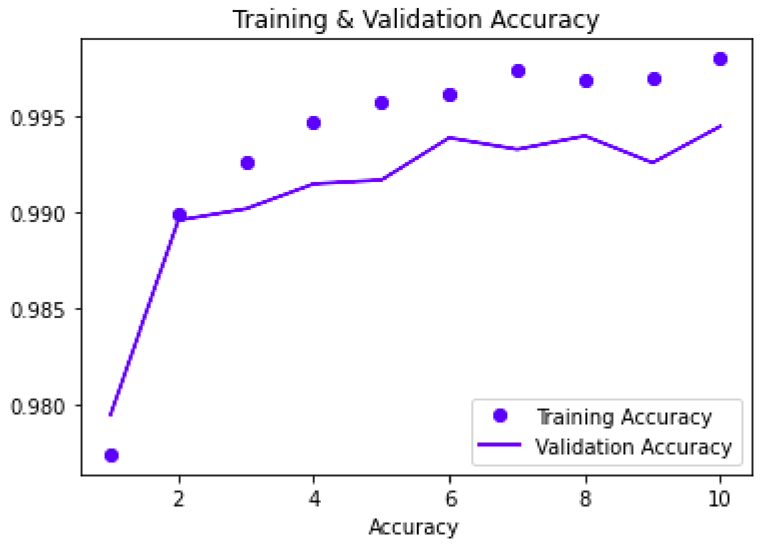


```
Loss at Final Training Step is: 0.008545895107090473
Accuracy at Final Training Step is: 0.9974363446235657
```

Lastly I tried initializing the weights, but this had a negative impact on the accuracy and so I decided to experiment with batch normalization instead. Not surprisingly, normalizing the fully connected top layers further improved the performance of the network significantly. The accuracy went from now 0.9974 to a total high of 0.9980.

This was the peak performance that was achieved although this normalization layer was later removed since the part 1 mention to only make use of convolution layers with pooling, dropout layer and 2 fully connected layers.





```
Loss at Final Training Step is: 0.005725102499127388
Accuracy at Final Training Step is: 0.9980363845825195
```

Conclusion: In the end the application to predict the even or odd nature of the digit from an image was completed. It was a good learning experince and I tried using neural networks for the first time to acheive this. This assignment definitely put my knowleadge of the deep learning to use. Other than neural network, basics of image processing were also put to use. I look forward to more such assignments.