Convolutional Neural Network for MNIST Dataset

Neel Patel Sayee Shruthi Zhikai Lin

February 26, 2021

Abstract

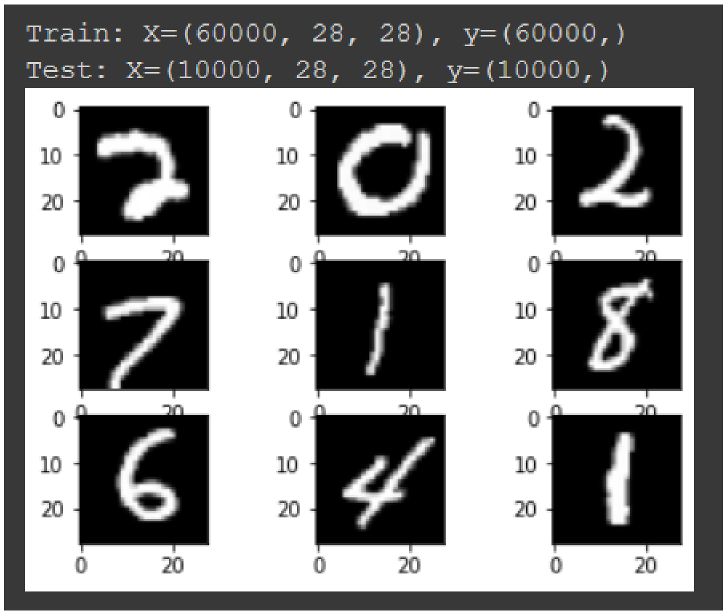
In recent times where technology has taken a big leap, Artificial Intelligence has brought changes into many fields. Deep Learning is remarkably being used into many domains. Convolutional Neural Networks(CNN) are being widely used in Image processing and Classification Tasks. They are very similar to Artificial Neural Network(ANN). The goal of this project is to develop a CNN that can classify handwritten digits based on the image input provided from MNIST Database[1] using Keras which is a high level API of Tensorflow.

1 Introduction

It is very easy task for humans to identify items or numbers from images. With recent advances in the field of Deep Learning, we can use CNN for identifying patterns from Images. For this, we need to train the CNN model using a dataset. Here, we are using CNN model on MNIST dataset that contains images of handwritten digits.

2 Loading the data

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.[1]. Each Image has 28x28 pixel grayscale images of handwritten digits from 0 to 9. This exercise classifies an input image into one of the 10 classes between 0 to 9.



```
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```

Figure 1: Loading MNIST data

3 Pre-processing the data

Firstly, we start by reshaping our 2D data array into single channel that would help model know that images taken as input are in grayscale. Following this, as pixel value for any image in dataset lies in range 0 to 255, we perform a normalization over the values to effectively convert them into 0 to 1. We perform normalization by converting datatype into float from integer and then divide the pixel value by 255. We know that there are 10 classes for the output which is digit 0 to 9. Hence, we perform one hot encoding on the output to obtain a binary array of length 10 which contains 1 in for it's class and 0 otherwise. We perform this using *.to_categorical()* function from keras util library. Moreover, We split further training data into train and validation data-set for tuning hyper-parameters.

4 Developing CNN Model

A typical CNN design is divided into 2 parts. It begins with feature extraction that is made from convolutional and pooling layers and finishes with classification layers.

While developing a baseline model, we used only one convolutional layer with 16 filters and kernel of size (3, 3). This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs[2]. Moreover, activation functions is applied to each layer according to parameters given. Each convolutional layer is followed by a Max Pooling 2D Layer. Max Pooling Layer downsamples the input by taking the maximum value over the window defined by pool_size for each dimension along the features axis[3]. Then we shift the window by value of stride provided. After this process, the filter maps needs to be flattened in order to provide it as input to classification layer. As this is a multi-class classification problem, the number of nodes in output layer has to be 10. Also, we need probability distribution for each class in order to predict the class. This can be done by using softmax activation function for the last layer of the network. We used Keras *Categorical Crossentropy* as loss function as it works best when labels are one-hot encoded and we need to classify into multiple classes.

5 Improving Performance

We start by choosing a experimenting different type of optimizer for the minimal base model which had only one *Conv2D* Layer and output layer. From the experiments, we could see that *Adam* had the highest accuracy and it learned fastest among other optimizers. This can be seen in Figure 2. Other optimizers like Adadelta performed worse and could not achieve accuracy of 70%. We are setting *Early Stopping* callback for model. The

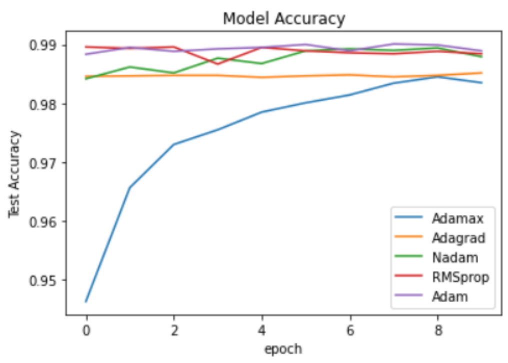


Figure 2: Accuracy of different Optimizers

model will stop training if we the accuracy keeps on decreasing for number of provided epochs. As a result, model will only train upto necessary number of epochs.[4]

For improving performance of the model, we can combine multiple convolutional layer with different number of filters. We can also add multiple classification layers accordingly. Firstly, we tried experimenting with different number of filters in a single Convolutional 2D Layer. We generated a single keras layer conv2D model with 8, 16 and 32 filters. After testing, we found that both models with 32 & 16 filters had higher accuracy as compared to model that had 8 filters as we can see in Figure 3. Also, we used *ReLU* activation function for all the convolutional layers as part of good practice.

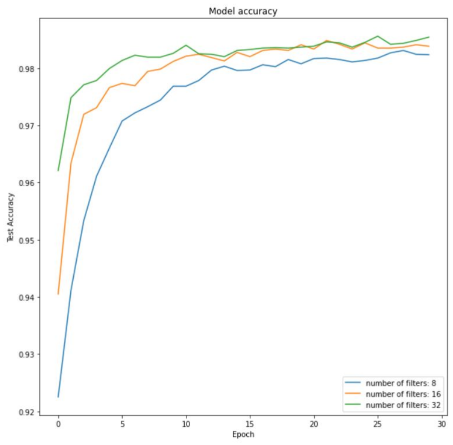


Figure 3: Accuracy for No of Filters in a Single Layer Conv2D model

After that, we generated a three models with each model containing conv2D layers of 2, 3 and 4 size respectively & predicted model accuracy and model loss. Each layer was followed by a Max Pooling 2D Layer. Figure 4 (on next page) indicates that model that contained 3 Conv2D layers had highest test accuracy. It had least loss which was between the range 0.11 to 0.12.

The result from the above experiment specifies that model is better with 3 layers. We continued the experiment by increasing the number of filters in each layer. The layers combinations we used were (8,16,32), (16,32,64), (32,64, 128), (64,128,256). Figure 5 (on next page) shows the test accuracy for these different models. The test outcomes reveals that layers with filter combination as (32,64,128) had higher accuracy as compared to other models.

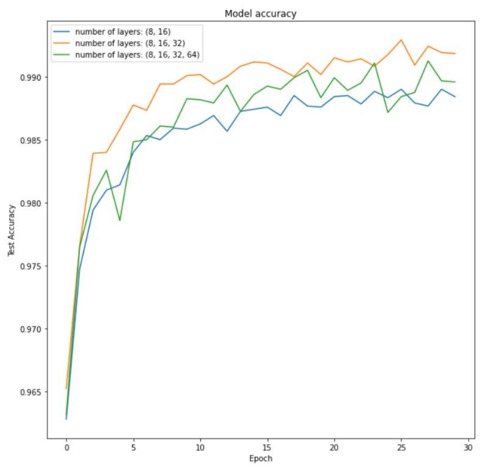


Figure 4: Accuracy for No of Layers in Conv2D model

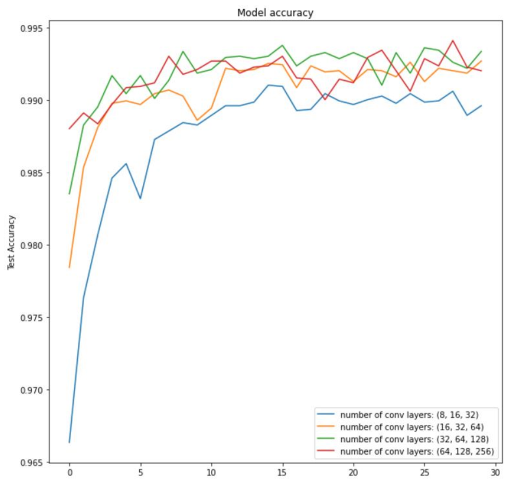


Figure 5: Accuracy for No of Filters in each Layer

At this point, we had the best possible combination of convolutional layers for the feature extraction front model. After this, we started experimenting with in similar way for dense layers which were required for classifier back-end.

So, we started by increasing the number of units of single dense layer and we found that all of them performed well. So, we decided to move forward with dense layer containing 256 units. Adding to this, we tried changing number of dense layers. Figure 6 shows that accuracy is decreasing as number of dense layer(s) increases. Following this layer, the last layer has to be output layer with 10 nodes as this is classification problem.

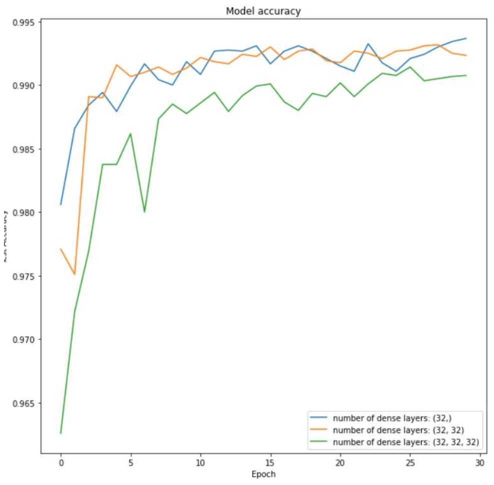


Figure 6: Accuracy for Dense Layers

Moving on, we tried different activation functions which are ReLu, leakyRelu and ELU with our final model. Figure 7 shows that comparison and ReLU performs with better accuracy.

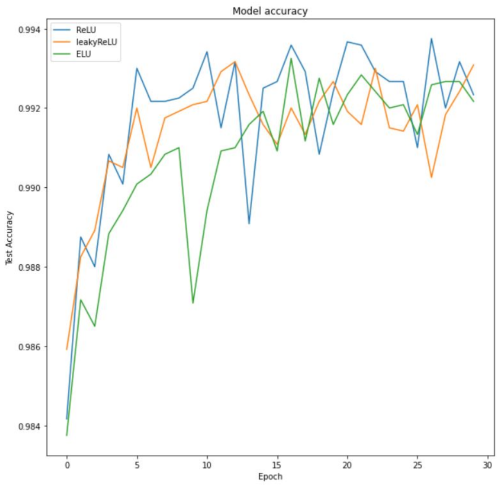
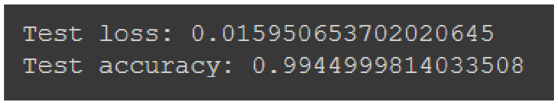


Figure 7: Activation Function Performance

6 Conclusion

We started by experimenting for the first part of the network which is convolutional layer and eventually followed to the second part classification. Finally, after all the experiments done with hyper parameters and models, we designed a 3 layer Conv2D model with (32,64,128) filter combination followed by flattening layer, dense layer with 256 units and output layer of 10 units. We trained this model with *Adam* as optimizer, *ReLU* activation function for convolutional layers and *Softmax* activation function for Output Layer to obtain **99.45** percent test accuracy and **0.015** model loss. Even though accuracy of this model is very high, it can be still be increased to approximately 99.75% when we provide input of approximately 25 million images generated by rotating, scaling, and shifting Original Dataset's images.[5]



```
Test loss: 0.015950653702020645
Test accuracy: 0.9944999814033508
```

Figure 8: Output Performance

Link to our code is here.

References

[1] The MNIST Database of Handwritten Digits.
`http://yann.lecun.com/exdb/mnist`

[2] Conv2D Layer in Keras Library.
`https://keras.io/api/layers/convolution_layers/convolution2d`

[3] Max Pooling 2D Layer in Keras Library.
`https://keras.io/api/layers/pooling_layers/max_pooling2d`

[4] EarlyStopping in Keras Library.
`https://keras.io/api/callbacks/early_stopping/`

[5] 25 Million Images! [0.99757] MNIST.
`https://www.kaggle.com/cdeotte/25-million-images-0-99757-mnist`