Neural Network

Kannada MNIST Image Classification

EXTRA CREDIT REPORT

KASHYAP MEHTA (ID: 014489891)

(PROJECT PARTNER: MUKUND VYAS)

Table of Contents

Figure of Contents

1. Problem Statement

The goal of this competition is to classify hand written image of kannada language digit recognition. The goal also includes to joining the kaggle competition and submitting the code.

2. Introduction

Kannada is a language spoken predominantly by people of Karnataka state in India. The language has nearly 45 million native speakers and written using the Kannada script.

Each image is of 28 pixels in height and 28 pixels in width, and 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

In addition to the main dataset, it also disseminated an additional real world handwritten dataset (with 5k images), termed as the 'Dig-MNIST dataset' that can serve as an out-of-domain test dataset. It was created with the help of volunteers that were non-native users of the language, authored on a smaller sheet and scanned with different scanner settings compared to the main dataset. This 'dig-MNIST' dataset serves as a more difficult test-set and achieving >98% accuracy on this test dataset would be rather commendable.

Features:

Image with pixel dimensions: 28 x 28 pixels
Training Data: 6000
Test Data: 5000

Labels:

The output is assigned to digits from 0 to 9 in kannada language.

3. Import Data

Kannada MNIST dataset requires downloading the dataset from kaggle server or directly using data if using kaggle notebook. In this project, we are using kaggle notebook.

4. Data Visualization

The feature 28x28 pixel if represented in graph than it is represented as below:

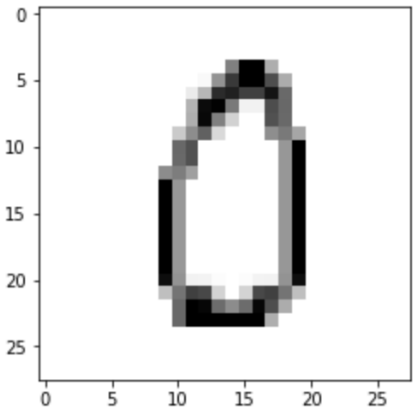


Figure 1: Data Visualization

The total number of training datasets are 60000 while test datasets are 5000. The input feature is reshaped to 28x28 = 784. In this project, training data is converted into train data & validation data as the test data does not have labels for verification.

The distribution of output for training data is represented as below. It could be depicted from the graph that the fashion MNIST dataset is equally distributed among all the output.

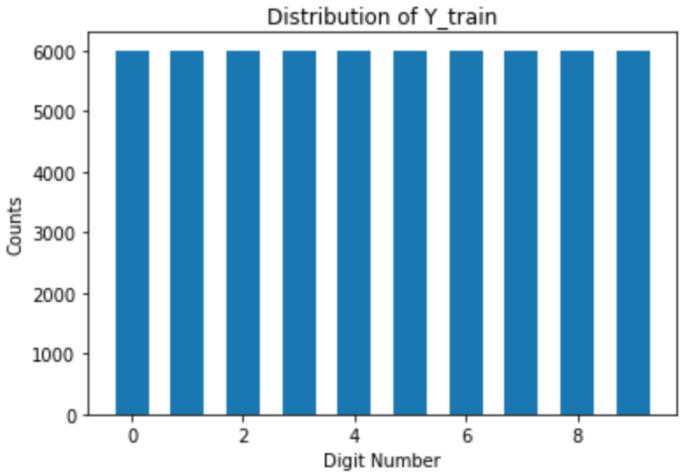


Figure 2: Training Data Distribution

5. Data Preprocessing - Feature Scaling

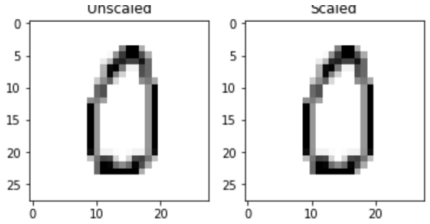
It is a step of Data Pre-Processing which is applied to independent variables or features of data. It basically helps to normalize the data within a particular range. It also helps in speeding up the calculations in an algorithm as if the scaling is not done than it might require large amount of iterations to converge to calculate the weights.

Scaling can be done in two ways:

i. **Min Max Scaling**: It converts the given features into the scale of 0 to 1. The standard formula to convert min max scaling is given below:

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

If given features are applied for min-max scaling than we cannot identify the scaling in images as the imshow function scales the input from 0-1 to 0-255 for representation of image. Hence we use Standardized Scaling method to scale.



ii. **Standardized Scaling:** It is a very effective technique which re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1. The standard equation to scale the input is as given below:

$$X_{new} = \frac{X_i - X_{mean}}{\text{Standard Deviation}}$$

The image shown below represents the unscaled input and scaled input for each image classified.

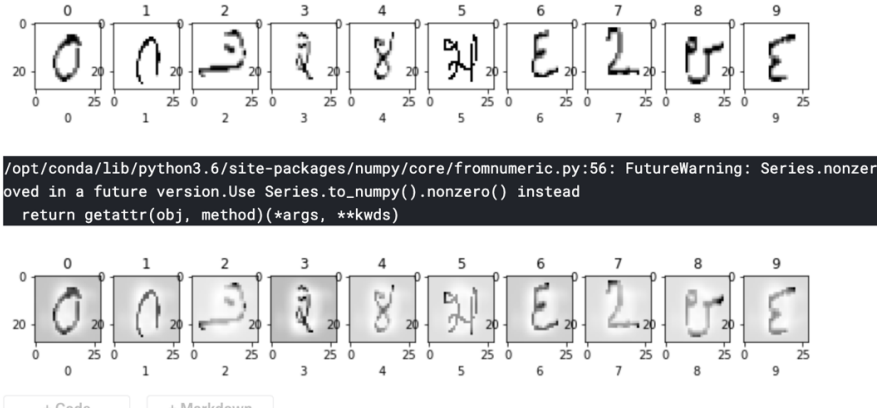


Figure 3: Scaled vs Unscaled Image Classification

6. Data Preprocessing - Split the dataset

The training data is divided into train & validation data using scikit learn library. The train data is divided into 48000 training data & 12000 validation data.

7. Building Model

In this project, convolution neural network is used for image classification. Following steps are performed for developing convolution neural network:

- Convolution
- Polling
- Flattening

To develop a model following are the design parameters for creating a fully connected feedforward convolution neural network:

- Number of kernels
- Dimensions of kernels
- Stride & padding
- Pooling
- Number of convolution layer
- Image Augmentation
- Number of Neurons in hidden layer
- Number of Hidden Layers
- Batch Size
- Number of Epochs
- Activation Function of Hidden Layer
- Activation Function of Output Layer
- Compiler

To get the best test & train accuracy following are the parameters which should be considered to design a model.

1. Complexity
    a. Change in Number of kernel size
    b. Change in Number of Convolution Layers
    c. Change in No. of Neurons
    d. Change in No. of Hidden Layers
    e. Change in batch size & epochs

2. Activation Function
    a. Hidden Layer
    b. Output Layer

3. Compiler
    a. Optimizer
    b. Metrics

I. Change in Number of kernel layers:

Variable:

- Kernel Dimensions:
    a. 32
    b. 64
    c. 128

Constants:

- Convolution Layer: 1
- Hidden Layer Activation Function: RELU
- Output Layer Activation Function: Sigmoid
- No. of Neurons: 128
- Hidden Layer: 1
- Batch Size: 16
- Epochs: 5

Observations:

- Figure 4 represents training accuracy with increase in epoch for different number of neurons in hidden layer.
- With increase in number of neurons in the hidden layer, the complexity of the model increases and hence the training accuracy increases.
- Figure 5 represents the accuracy of training & test data for different number of kernels in single convolution layer.
- For 64 number of kernels in convolution layer the test accuracy is achieved to be maximum and hence the design parameter for single convolution layer is chosen to be 64.
- It could be depicted that with increase in number of kernels the accuracy increases but after particular number of kernels in convolution layer, the accuracy does not increase with increase in number of kernels.

Output:

- 64 number of kernels is selected in convolution layer.

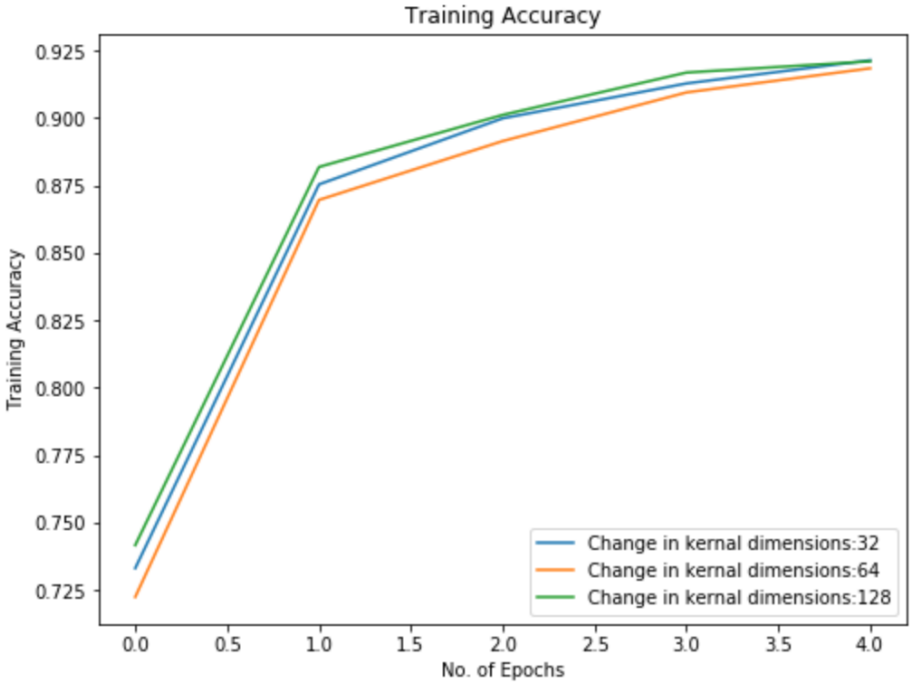


Figure 4: Training accuracy vs Epoch with change in kernel dimensions

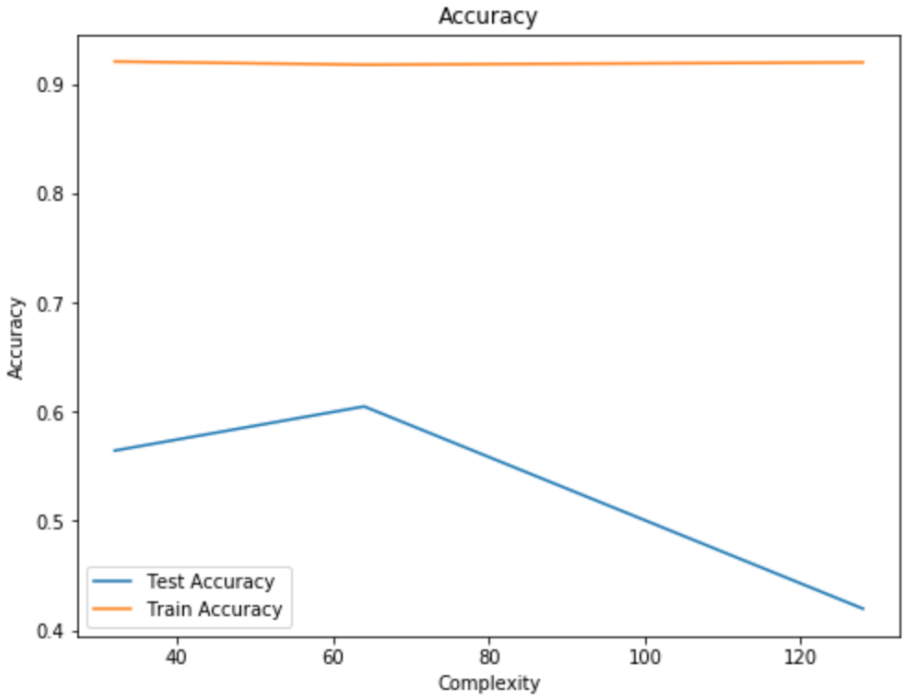


Figure 5: Training Accuracy with change in kernel dimensions

II. Change in No. of Convolution Layers:

Variable:

- No. of Convolution layers:
    a. 1 Layer with no. of kernel layers 64
    b. 2 Layers with no. of kernel layers 64
    c. 3 Layers with no. of kernel layers 64

Constants:

- Number of Neurons: 128
- Hidden Layer Activation Function: RELU
- Output Layer Activation Function: Sigmoid
- Batch Size: 16
- Epochs: 5

Observations:

- Figure 6 represents training accuracy with increase in epoch for different number of convolution layers.
- The training accuracy is almost similar with change in number of convolution layer.
- Figure 7 represents the training & test accuracies with change in number of convolution layers. The graph represents the accuracy for 1, 2 & 4 convolution layers.
- As shown in figure, the test accuracy is least for 2 convolution layer and it increases afterwards as the model is fitted with the complexity.

Output:

- 1 Convolution Layer is selected as the test accuracy is maximum

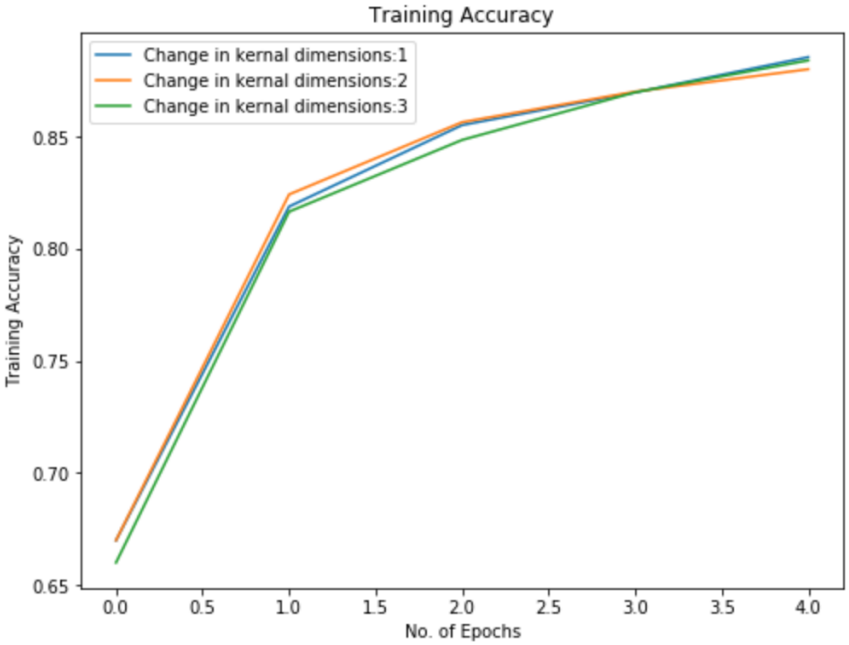


Figure 6: Training Accuracy v/s Epoch with change in number of convolution layer

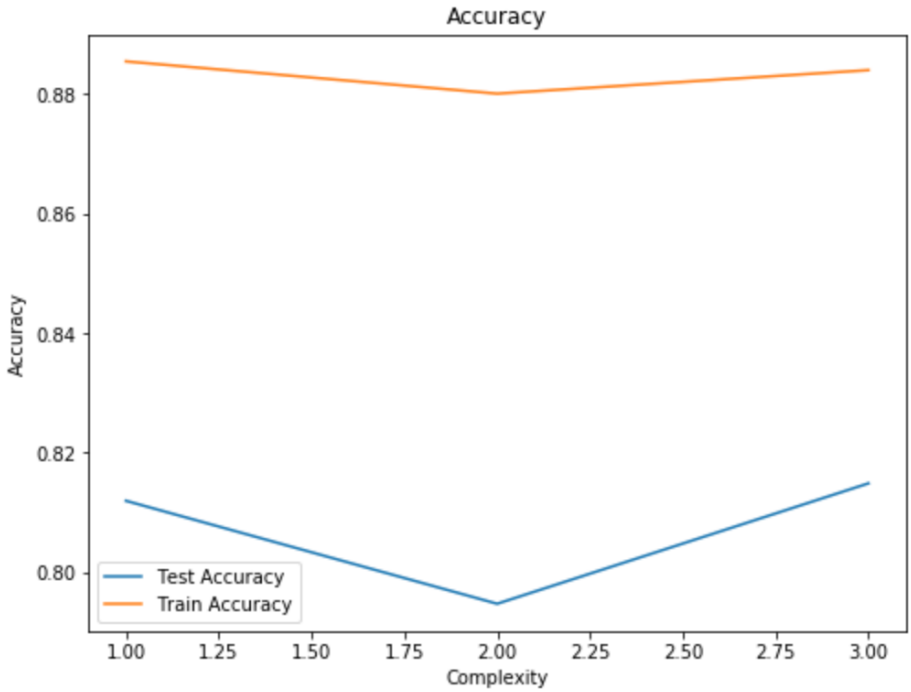


Figure 7: Test accuracy with change in number of convolution layers

III. Change in Batch Size

Variable:

- No. of Batch Size & No. of Epochs:
    a. 64 Batch Size for 5 Epochs
    b. 128 Batch Size for 8 Epochs
    c. 1024 Batch Size for 10 Epochs
    d. 4096 Batch Size for 12 Epochs

Constants:

- Number of Neurons: 128
- Hidden Layer Activation Function: RELU
- Output Layer Activation Function: Sigmoid
- Hidden Layer: 1
- Epoch: 5

Observations:

- Figure 8 represents training accuracy with increase in batch size for different epochs used to get the output.
- For large batch size the training accuracy is less for very first iteration. As the batch size is large, it requires less time to converge and hence we have to increase the number of epochs.
- With increase in batch size the training accuracy decreases as well as it requires a greater number of epochs to converge. The training accuracy is highest for 128 Batch size for 8 Epochs.
- Figure 9 represents training & test accuracy with respect to Batch Size. The accuracy is increases gradually for Batch size 64 to 128 than it decreases with increase in Batch.
- Hence, we could conclude that the best Batch size could be 128 with Epochs 8.

Output:

- Batch Size & No. of Epochs is selected as 128 & 8 respectively for the model.

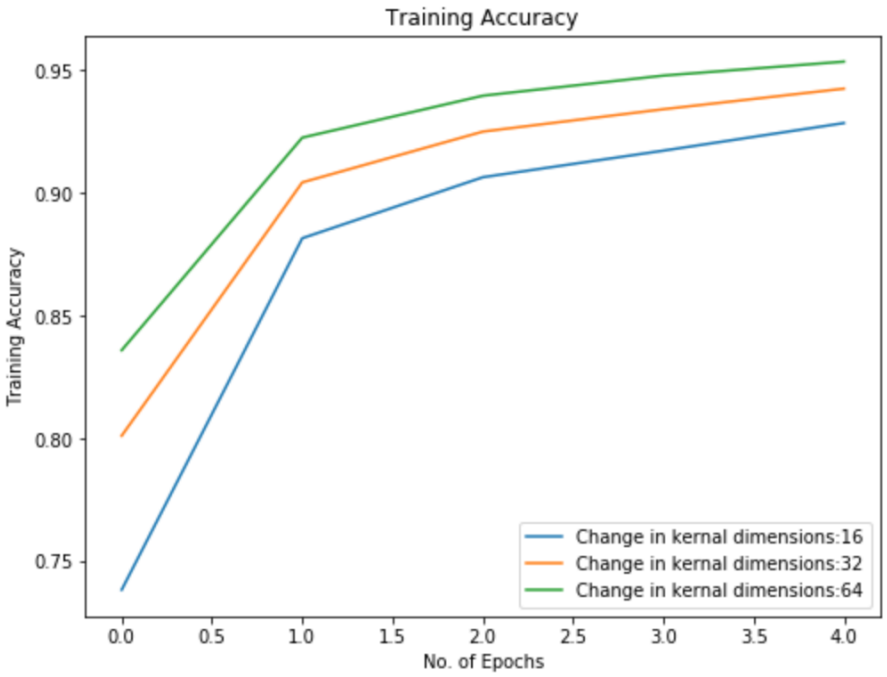


Figure 8: Training Accuracy v/s Epochs with change in batch size

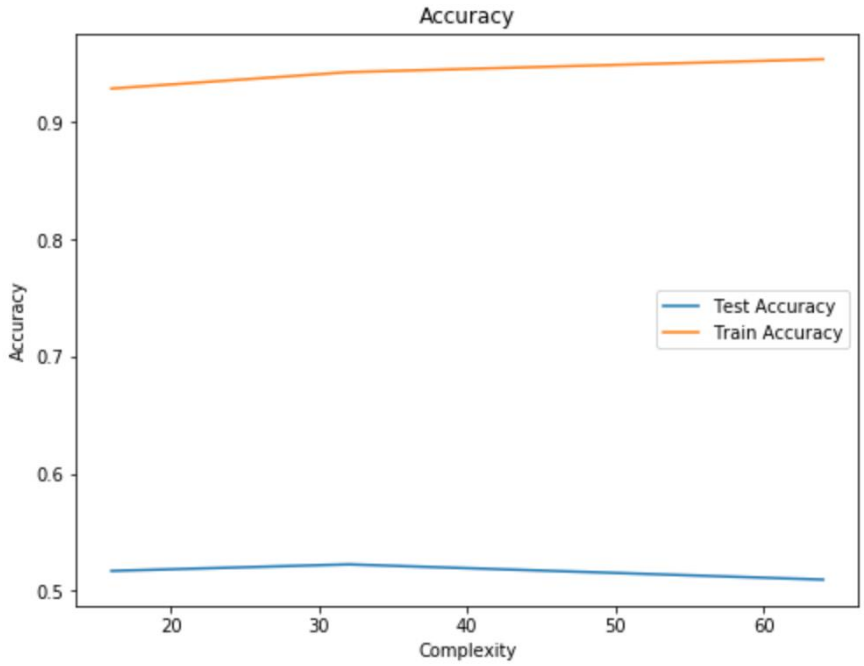


Figure 9: Test Accuracy with change in batch size

IV. Change in Number of Hidden Layers

Variable:

- Number of Hidden Layers:
    a. 1
    b. 2
    c. 5

Constants:

- Number of Neurons: 30 in each hidden layer
- Convolution Layer: 1
- No. of Kernels: 64
- Batch Size: 64
- Epochs: 5
- Output Layer Activation Function: Sigmoid

Observations:

- Training accuracy & test accuracy is represented in figure as below
- Figure 10 represents the accuracy of training & test data with respect to change in number of hidden layer.
- Figure 11 represents the test accuracy with respect to change in number of hidden layers. It could be observed that the test accuracy is maximum achieved for two hidden layers.

Output:

- Two hidden layer is selected for the model

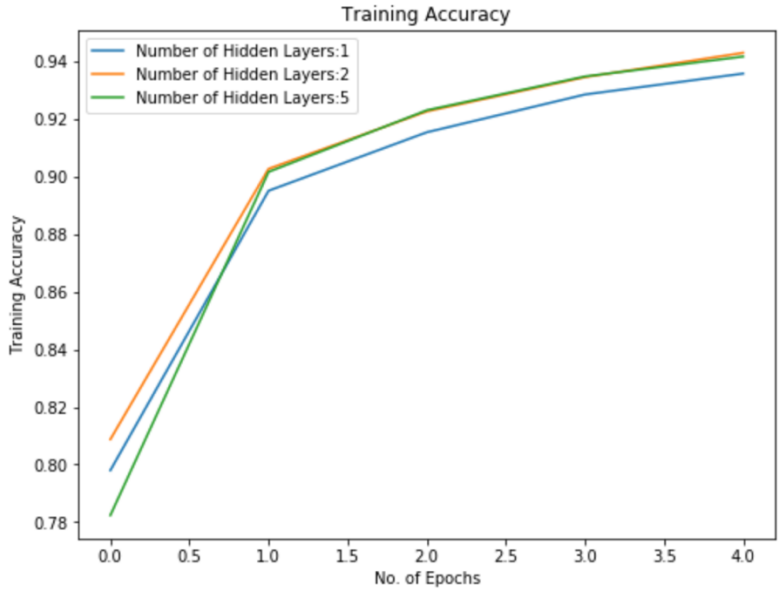


Figure 10: Training Accuracy v/s Epochs with change in number of hidden layers

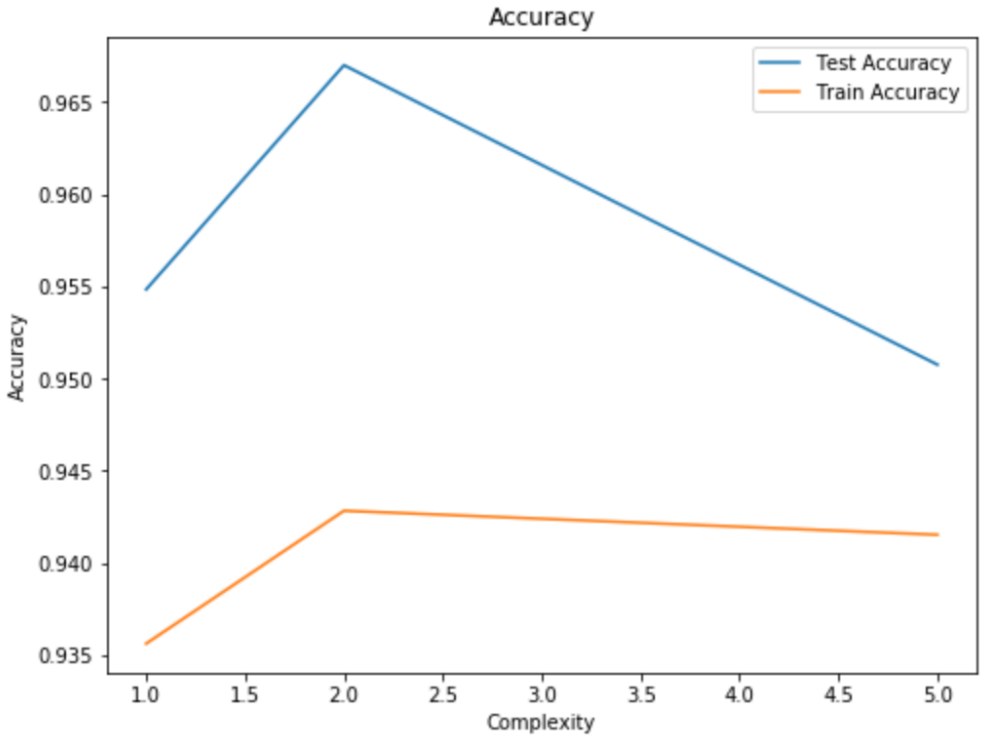


Figure 11: Test accuracy with change in hidden layers

8. Model Training

Parameters:

- Convolution Layer: 2
- Kernels: 64 for first convolution layer; 128 for second convolution layer
- -Number of Neurons: 384
- Number of Hidden Layer: 2
- Batch Size: 64
- Epochs: 10
- Activation Function of Hidden Layer: RELU
- Activation Function of Output Layer: Sigmoid
- Optimizer: Adam
- Loss: Sparse Cross Entropy
- Metrics: Accuracy

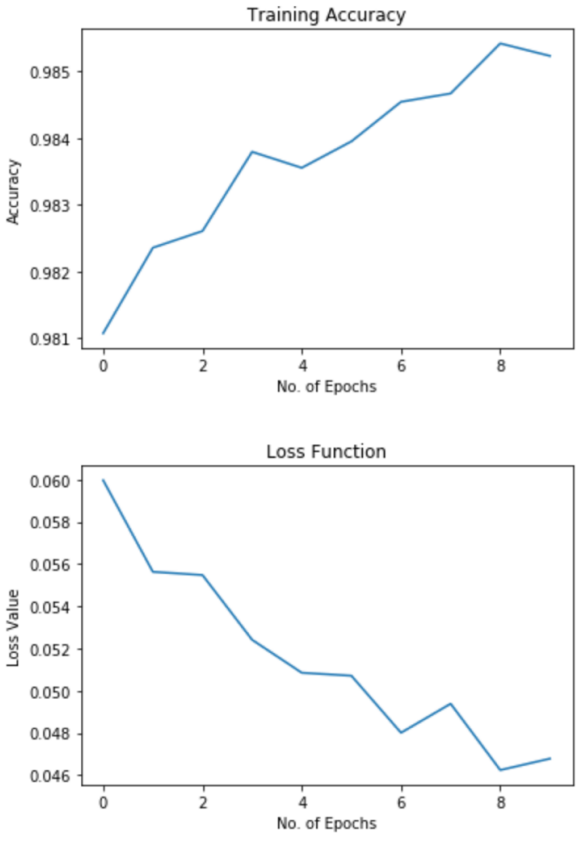


Figure 12: Accuracy & Loss function of the trained model

After selecting design parameters for the model and iterating the model for multiple times the above parameters are fixed. The graph below represents the training accuracy and loss function with increase in number of epochs. As the number of epochs increases the training accuracy will increase as the model will try to over fit with reduction in loss function.

In this project, training data is divided into test data for validation. The training data size is selected as 48000 datasets while the test data size is selected as 12000 datasets. With the convolution neural network, the training accuracy is achieved to 98.12% and the validation accuracy reached to 98.5%.

We have submitted the code on kaggle and received the Public Score as 0.9626.

9. Challenges of the project

The biggest challenge of the project was to submit the code online to Kaggle, as the code was not generalized for different test data and hence the Kaggle website was not able to identify the submitted file.

Another challenge of the project was to plot the graphs to build the model. For plotting a single graph of training the model, it requires to run the model multiple times, save the parameters in array, and then plot the respective graph. To train a model in loop multiple times with changing parameters was hectic and very difficult in computation for GPU server.

10. Conclusion

The Kannada data MNIST image classification project consist of hand written image classification of kannada language, using convolution neural network algorithms. Using two convolution layers the model is trained and the training accuracy is achieved to 98.12% while the test accuracy reached to 98.5% and public score of 0.9626.

In future, the model could be trained using random forest algorithm to reduce the complexity and time to train the model as the random forest is used for multi-class feature detection.