Contents:

Chapter-1

1.1 Abstract:

High performing neural networks are under development so that they can be used to identify and understand multiple languages, one such opportunity is given by kannda MNIST dataset, which allows me to create a neural network which can identify digits of kannada language which is language originated from ancient state of Karnataka located in India.

1.2 Goal of the project:

Goal of this project is to develop a CNN based classification model which can classify the images from kannda MNIST dataset.

1.3 Introduction:

Kannada MNIST dataset is the collection of various images of digits from ancient kannada language, the given dataset has 65,000 images each with a 28x28 pixel size. This project report is split into 2 dominant parts, where first part elaborates about the type of dataset being used, majorly detailing about the features for better understanding of dataset. Second part of this report details about creating and implementing CNN based classifying model for the given dataset.

1.4 Design Methodology

In this project I will be developing initial understanding about the given dataset and then I will create a CNN based model which can classify the images from given dataset. I have followed the design methodology given below:

A. Understanding and visualizing the dataset.
B. Creating a baseline model and discussing about all CNN parameters.
C. Modifying, testing and comparing different models.
D. Submitting the final model to kaggle.
E. Concluding the project, with final rank and my understanding.

Chapter-2: Dataset Description.

2.1 Loading the dataset:

Raw data set can be downloaded used from kaggle as mentioned by YonminMa [1], the training dataset has 60,000 images whereas test dataset has 5,000 images, where each image has 28x28 pixel size.

2.2 Description of the Kannada MNIST fashion dataset:

The dataset is split between 2 files denoted by train.csv and test.csv as mentioned above, train.csv has 60,000 images and test.csv has 5000 images, consider figure-2.2.0 which shows a part of the dataset, in this section I have used resources provided by Shahules786 [2].

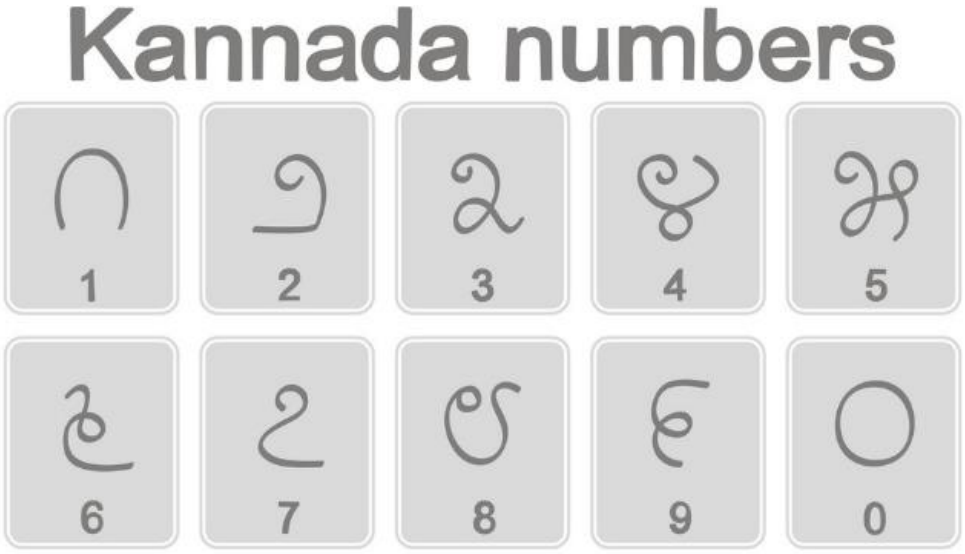

Figure-2.2.0: Glimpse of the Kannada MNIST dataset [1].

Each image is an array of 28x28 pixel values which is flattened down to 785 pixels, therefore converting training into array of 60,000 rows with 785 columns and test data into array 5,000 rows with 785 columns. The dataset has 10 distinct labels ranging from 0 to 9, where each label denotes one distinct kannada digit.

2.3 Describing the dataset through Visualizing:

Consider figure-2.3.1 which shows numerical values of flattened training data, as mentioned above each pixel represents a column and each image represents a row. Similarly figure-2.3.2 shows flattened values of test dataset.

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel7
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
59995	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
59996	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
59997	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
59998	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
59999	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

60000 rows × 785 columns

Figure-2.3.1: Flattened values of training dataset.

	id	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel78
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4995	4995	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4996	4996	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4997	4997	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4998	4998	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4999	4999	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5000 rows × 785 columns

Figure-2.3.2: Flattened values of test dataset.

Consider figure-2.3.3 which shows total counts for each label, it should be denoted that entire dataset has 10 distinct label where each corresponds to one distinct kannada digit.

```
9    6000
8    6000
7    6000
6    6000
5    6000
4    6000
3    6000
2    6000
1    6000
0    6000
Name: label, dtype: int64
```

Figure-2.3.3: Total number of count for all 10 distinct labels.

It is evident from the image above that none of the data values are missing and data is evenly balanced, since each distinct label has count of 6000, whereas entire dataset contains a total of 60,000 images refer to figure-2.3.4 shows randomly generated images from the training dataset.

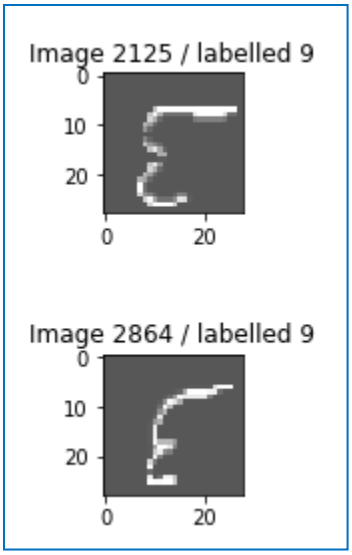

Figure-2.3.4: Randomly generated images from training dataset.

Since the maximum and minimum values of each pixel are between 0 to 255, therefore entire dataset can be **normalized between 0 to 1** by dividing it by 255, this will ease out the computation.

Chapter-3: Creating Baseline model & Methodology.

3.0 Reason behind using CNN for this project.

Convolution neural network (CNN) is exclusively designed to train for image classification based problems since it follows weight sharing mechanism thus it can operate on large input data while easily extracting features using various filters and therefore it consumes small memory.

3.1 Design Methodology before setting up the CNN.

Since this is an image classification problem, therefore I added **data augmentation methodology** in my model, which help in regularization and train more data for better results, the parameters for applying this method are shown in table-3.1.1.

Table-3.1.1: Implementation of data augmentation.

Parameter Name	Parameter Value
Rotation range	10
Width Shift Range	0.2
Height Shift Range	0.2
Shear Range	10
Horizontal Flip	False
Zoom Range	0.15

Data augmentation helps in creating excessive data by translating, rotating, scaling, changing the height-width and zooming various pixels in the given dataset which are used while training my model, therefore it makes my network robust and more accurate. Before building the model it is essential to reshape my image data into 2D array, so that 2D convolution can be deployed on this data, refer to figure-3.1.1 which shows new dimensions of training data.

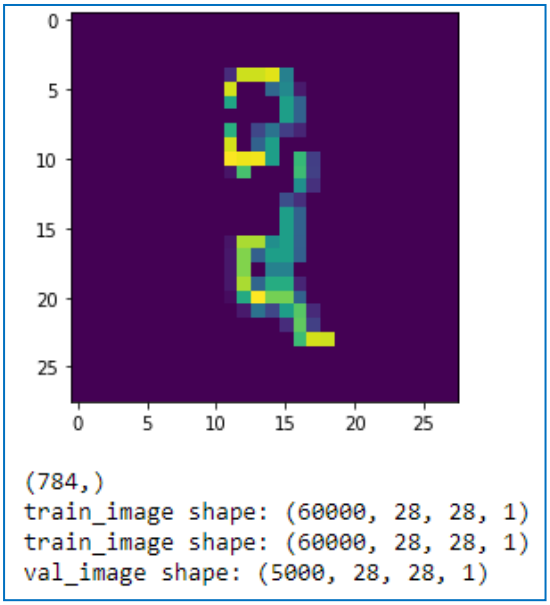

Figure-3.1.1: Training and validation data reshaped to 2D array for convolution.

There are 3 major parts in CNN namely convolution layer, pooling layer and fully connected layer, I will share a quick overview of these layers in following section, refer to fgire-3.1.2 for viewing the overall architecture of the cnn model.

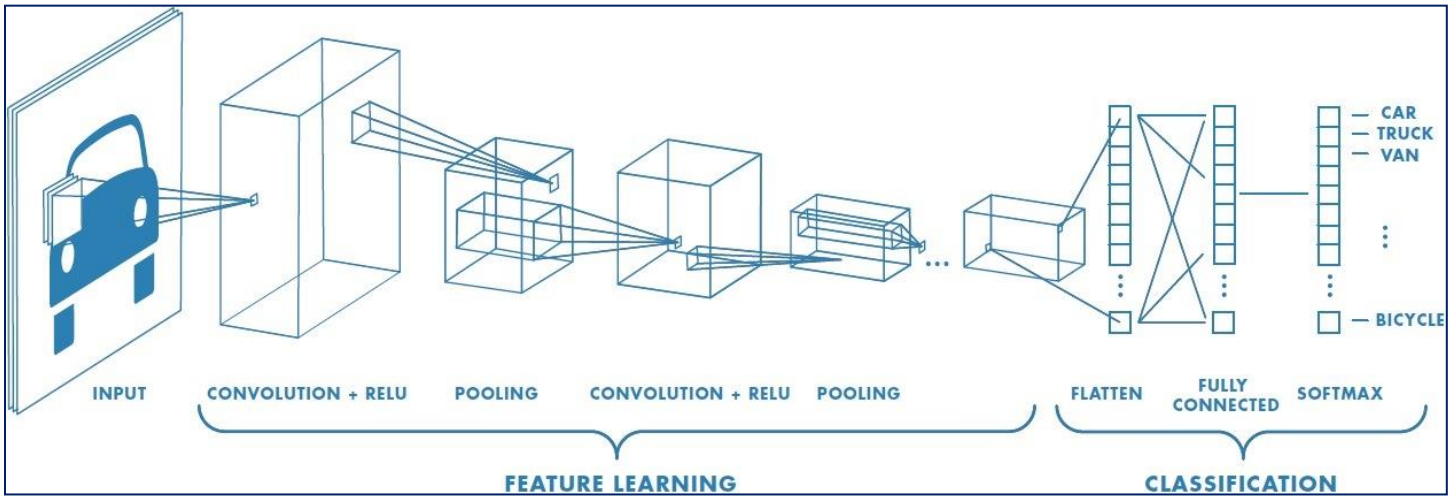

Figure-3.1.2: Overall architecture of a cnn based model.

Convolution layer applies convolution operation on the input layer, while using a dot product operation between the input data and weights, the result is passed on to following layer, this operation changes the dimension of the resultant data since convolution filter matrix might vary.

Pooling layer performs down sampling operation along the width of the input data in order to reduce spatial dimensions. In this project I have used max pooling since it takes the largest element from the given window.

A fully connected layer is also known as a dense layer which is typically a fully connected layer of neurons which receive inputs from previous layers, have their own weights and bias. The output of this layer is given by equation shown below.

output = activation(dot(input, kernel) + bias)……..eq1

Output of a fully connected layer depends upon the output of activation function used in that layer therefore activation refers to activation function, kernel refers to weights associated with that neuron, refer to figure-3.1.2 which shows overall architecture of my cnn model.

3.2 Creating CNN model.

Consider table 3.2.1 which shows parameters used and configured for creating a cnn model for this project. Parameter names are given on the left column whereas their configured values are in the right column.

Table-3.2.1: Representing configuration of CNN model.

Parameter Name	Layer-1	Layer-2	Layer-3	Layer-4	Layer-5	Layer-6	Layer-7
Number of Neurons	64	64	64	128	128	128	256
Activation Function	Relu	Relu	Relu	Relu	Relu	Relu	Relu
Batch Normalization	0.9	0.9	0.9	0.9	0.9	0.9	0.9
Padding	Same	Same	Same	Same	Same	Same	Same
Leaky relu	Alpha=0.1	Alpha=0.1	Alpha=0.1	Alpha=0.1	Alpha=0.1	Alpha=0.1	Alpha=0.1
Kernel Size	3	3	5	5	5	5	5

It is evident from the table above that there are 7 convolution layers used in this model, each with relu as a activation function, the major difference between **leaky relu** and relu is that, leaky rely can obtain negative values while latter can not attain it.

Relu activation function returns element-wise max(x, 0) for all x>0, else it is 0, this provides very simple and fast computation abilities in comparison to sigmoid or tanh functions. Output is always linear but non-negative, therefore it generalizes input data by setting negative inputs to 0 during final output unlike sigmoid function.

Batch normalization helps in reducing the number of epochs required to train the model because it stabilizes the learning process.

Padding is kept same so that bits are padded evenly from both left and right side and uneven padding can be avoided.

Whereas **Max Pooling** size is fixed as 2x2, the pooling layer helps in reducing the dimension of the incoming data values by selecting the maximum values from the 2x2 pool.

Dropout Regularization is set to 0.3, this helps in dropping out random values while training, this implies that contribution of activation functions by dropped out neurons is ignored in the forward pass, while their weights are not updated during the backward pass.

The final layer is a **flattened dense layer** as described above which has 256 fully connected feed forward neurons and uses softmax as activation function in the output layer because it gives probabilities as output for classifying given inputs, therefore sum of this function for a given input is always one and output can be easily determined by selecting the one having maximum probability. Using these parameters CNN is compiled successfully and in upcoming section I will discuss the results in detail.

Kernel size is set to 3 and later changed to 5, this means that convolution window size is set to 3x3 for layer-1 and layer-2 whereas it is changed to 5x5 thereafter.

Number of epochs is set to 1 due to time constraints, though setting higher number of epochs will provide better accuracy since weights will be trained more number of times, whereas **batch size** is set to 64.

Activation function as Softmax, in the output layer activation functions are decision maker mathematical functions which give an output to a set of inputs, various activation functions like tanh, sigmoid, softmax etc. exist which can be used while creating a network. It is worth mentioning that softmax activation function is used in most of the output layers since it is a multi class classification problem, whereas activation functions and other parameters are varied in hidden layers. Yet another advantage of softmax activation function in output layer is that it gives probabilities as output for classifying given inputs, therefore sum of this function for a given input is always one and output can be easily determined by selecting the one having maximum probability.

Chapter-4: Results of the Baseline model.

4.0: Simulating the baseline model

I have used parameters described above in my base model, which compiled successfully, to produce the results shown in table-4.0.1, additionally figure-4.0.1 represents the same values in a line chart.

Table-4.0.1: Training, validation accuracy and loss.

Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
0.90	0.986	0.96	0.13

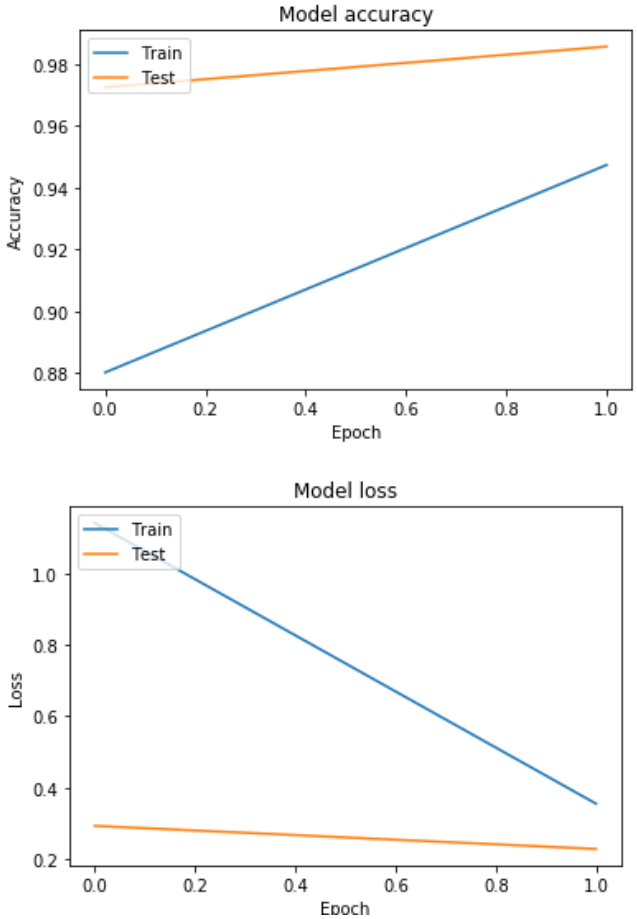

Figure-4.0.1: Training, validation accuracy and loss line graph.

With increasing number of epochs validation accuracy increases, which is represented by orange line, whereas validation loss reduces significantly. The validation accuracy is 98.6% which is expected in a cnn based model after using data augmentation on the given dataset, since it trains the network to be more robust, refer to figure-4.0.2 which shows the confusion matrix of the dataset.

	0	1	2	3	4	5	6	7	8	9
0	5945	55	0	0	0	0	0	0	0	0
1	2	5998	0	0	0	0	0	0	0	0
2	9	0	5991	0	0	0	0	0	0	0
3	2	0	1	5975	2	0	0	20	0	0
4	1	0	0	0	5995	0	0	0	0	4
5	0	0	0	2	0	5998	0	0	0	0
6	0	0	0	1	0	0	5946	41	0	12
7	0	0	0	2	0	0	1	5997	0	0
8	0	0	0	0	0	0	0	0	6000	0
9	1	0	0	0	0	0	15	0	1	5983

Figure-4.0.2: Confusion matrix of the kannada mnist dataset.

Confusion matrix is shown in figure-4.0.2, that represents a multiclass matrix with true positives and false positives, it should be noted that for this problem accuracy matrix is not of much use, though recall and precision matrix matter the most for evenly balanced multiclass dataset, refer to figure-4.0.3 which shows a heat of the same values.

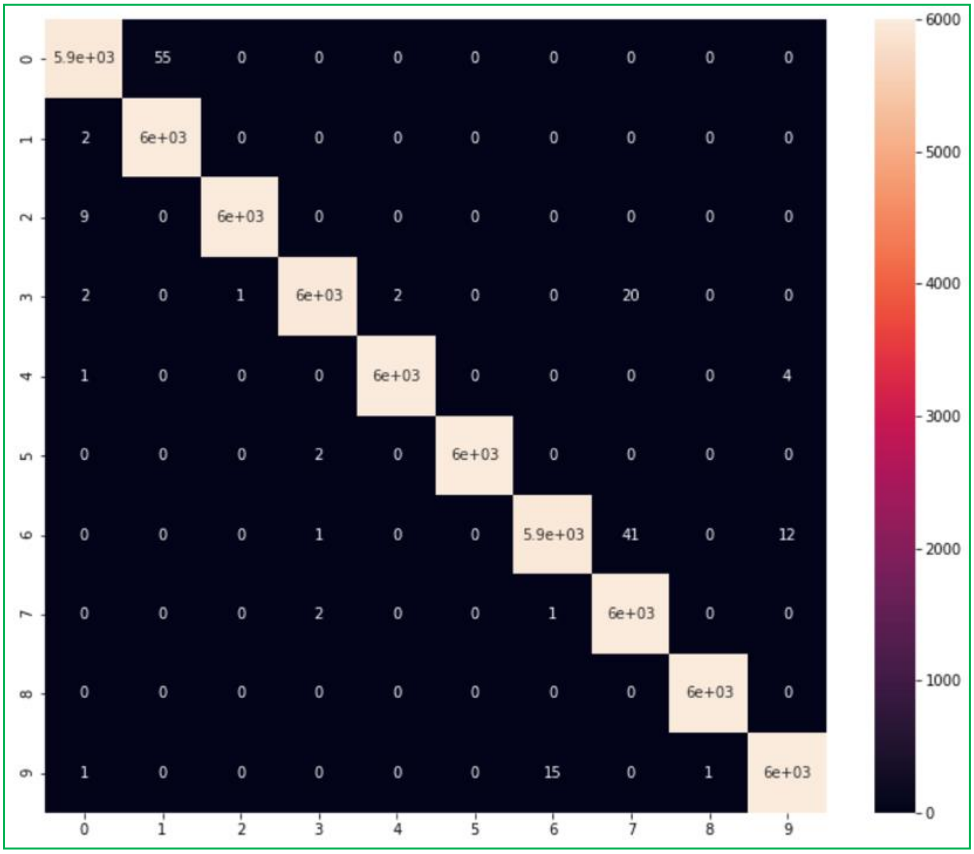

Figure-4.0.3: Representing heat map of confusion matrix displayed above.

4.1: Changing the parameters of baseline model.

Changing Optimizer:

SGD or stochastic gradient descent is a variant of traditionally used gradient descent with a distinguish feature of selecting data randomly for optimizing the cost function. Unlike traditional gradient descent algorithm which slowly samples through the data computing partial derivatives, SGD randomly selects data which makes it

faster than gradient descent. But in this case Adam optimizer has better training, validation and test accuracy since ADAM optimizer adapts typically computing adaptive learning rate by updating from first & second moments of the gradients, refer to table-4.1.0 and figure-4.1.0 for the optimized values.

Table-4.1.0: ADAM as optimizer.

ADAM as Optimizer			
Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.1733	0.97	0.1014	0.99

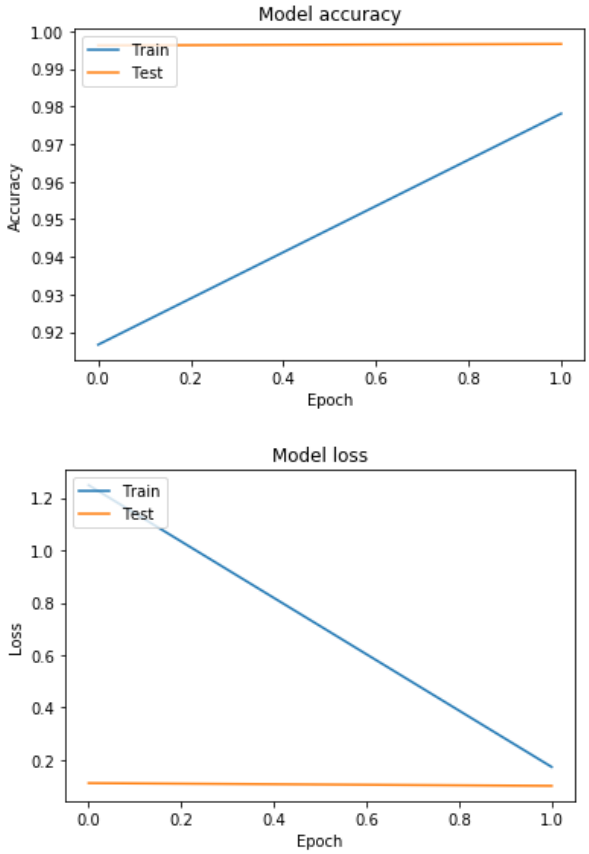

Figure-4.1.0: ADAM as optimizer representing test and validation accuracy and loss.

Undoubtedly, RMSprop performs better than previously used SGD optimizer because, RMSprop uses technique of applying weighted average method while computing the gradient of second moment. Thereby finding more weights than SGD, but ADAM optimizer outperforms RMSprop since it uses techniques of RMSprop along with bias correction for computing adaptive learning rate by updating from first & second moments of the gradients.

Changing Batch Size:

I have changed batch size to 64 and 1024, in order to compare the results please refer to table-4.1.1, shows the comparison between two batch sizes.

Table-4.1.1: Comparing accuracies of different batch sizes.

Batch Size	Training Accuracy	Training Loss
64	90%	0.96%
1024	63.9%	8.71%

It is evident from table-4.1.1 that larger batch size does not perform well because global minima is always not a solution for a non-convex function. Therefore smaller batch size focuses on local minima or settles around it and gives better accuracy results.

Changing Learning rate:

Learning rates are a very tricky parameter but this model clearly concludes that very high or very low learning rates will not provide optimal weights, therefore a mid value serves as the best learning rate, refer to table-4.1.2 for reading training, validation accuracy and error values for new learning rate of 0.01.

Table-4.1.2: Training, validation accuracy and error values for new learning rate.

Training, validation accuracy and error values for new learning rate = 0.01			
Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
3.591	0.92	7.13	0.93

High learning rate will converge quickly but it misses the minima, on the other hand a mid point value of leaning rate converges slowly but settles around the minima.

Changing number of neurons and number of layers:

Complicating the network by increasing more layers and number of neurons causes over fitting in this case, since more neurons and hidden layers will over fit the weights around local minima of the available cost function in the backward pass of back propagation algorithm, refer to table-4.1.3 for reading training, validation accuracy and error values for increased complexity.

Table-4.1.3: Training, validation accuracy and error values for increased complexity.

Training, validation accuracy and error values for new learning rate = 0.01 New number f neurons in each layer 512-512-1024-10 respectively.			
Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.30	0.96	0.33	0.92

Therefore number of layers and number of neurons given the baseline model serve as the best degree of polynomial (complexity) for this dataset, since more number of neuron in each layer highly decrease the validation accuracy and on the other hand training accuracy is increased due to over fitting as shown in the figure-4.1.3.

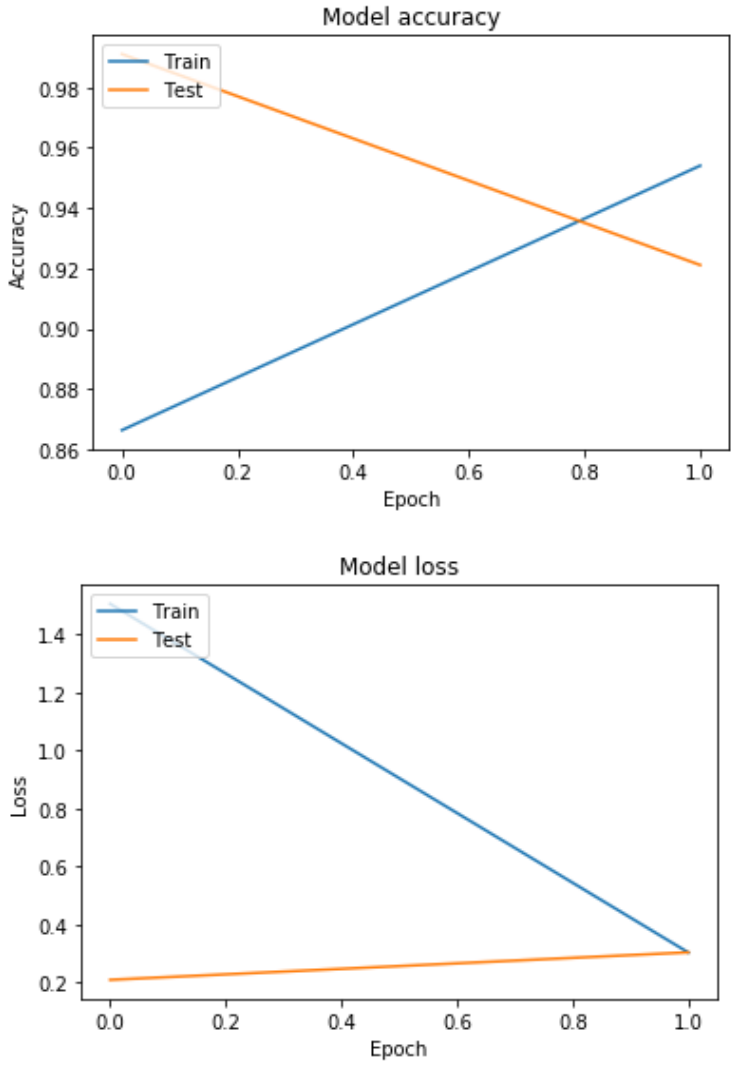

Figure-4.1.3: Line chart for training, validation accuracy and error values for increased complexity.

Chapter-7: Conclusion and Kaggle Submission.

In this project I used a cnn based model to classify kannada mnist dataset with a very high accuracy of 98.6%, these results were submitted in kaggle and my team rank is shown in following section. I changed various parameters of cnn and realized that my baseline model was the best performing model with small batch size and softmax activation function the output layer. Complicating the network by increasing more layers and number of neurons will only cause over fitting after a certain point. Therefore my base model has optimal parameters for this project. Data augmentation is a powerful method of generating extra data that helps in making my model robust and more accurate.

Kaggle Submission:

Our baseline model with validation accuracy of 98.6% was submitted on Kaggle which provided us a very good rank of 476, which places us in top 41% among all the participants, please refer to figure-7.0.1.

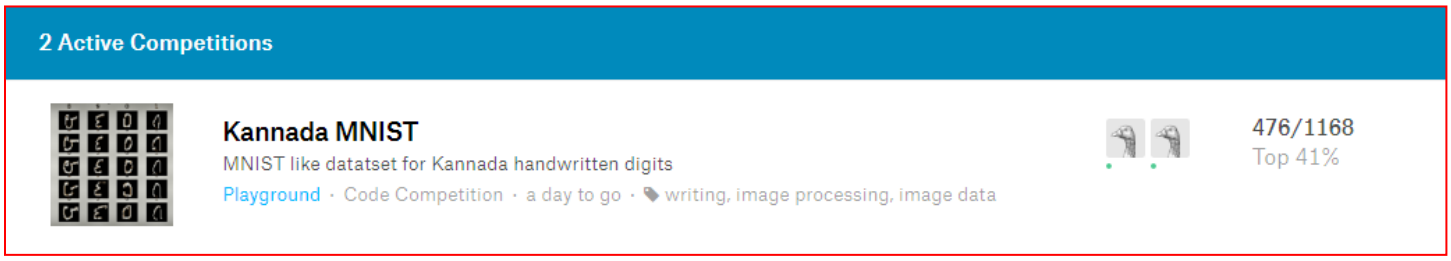

Figure-7.0.1: Kaggle submission and team rank is 476 out of 1168.

Challenges faced:

The biggest issue with this project is time constraint that forces me to limit my work, on top of that dependency on kaggle, training n their platform is a very big overhead.

Training takes 2 hours and every small change made takes a long amount of time, this is very time consuming.

Since we depend on kaggle for simulating the results, instead of simulating them on my laptop, I could only change hyper parameters once, I was unable to simulate my model for a bigger range of different hyper parameters.

Future work:

This project should be executed on Nvidia's gpu board for real time synthesis.

Image size should not be fixed, algorithm should automatically find the image size and then perform convolution.

Computer vision based real time image capturing algorithm should be combined with this model, so that real time image synthesis can be performed on kannada language.

Special Mention:

I will like to mention this project was possible because of the online available material given by YonminMa [1] and Shahules786 [2], their code and data analysis with appropriate pre processing provided a very strong guidance to our team.

Reference.

[1] https://www.kaggle.com/yonminma/keras-easy-with-0-9892-score

[2] https://www.kaggle.com/shahules/indian-way-to-learn-cnn