**Introduction –**

The MNIST (**Modified National Institute of Standards and Technology**) database is a large database of handwritten numbers or digits that are used for training various image processing systems. The dataset also widely used for training and testing in the field of **machine learning**. Just the way our brain and eyes work together to recognize any numbered image, we can use Convolutional Neural Networks to extract features like shape, etc in order to determine which number has been written.

**Dataset** –

The Training Dataset consists of 60,000 labelled images and the Test set contains 10,000 labelled images, each containing pictures of handwritten digits. The MNIST dataset is a multilevel dataset consisting of 10 classes in which we can classify numbers from 0 to 9. A single data point comes in the form of an image of 28x28 pixels, in grayscale. This means that a single image from the MNIST database has a total of 784 pixels that must be analysed.

Graphical user interface

Description automatically generated

**Background –**

One must think why there is a need to use a separate kind of neural networks when there is a possibility to directly give the images in the form of a multi-dimensional array to an artificial neural network. But one needs to realise the cost complexity of such a task. An average full HD RGB image has 1920x1080p pixels and 3 different values of red, green and blue colours between 0-255. So if we do the maths, it turns out to a staggering 6 million neurons in the first layer, and this is just for one image. If we have to train a network to recognize such images, we will need multiple layers and thousands of such images to train the network and the cost complexity increases exponentially.

Secondly, slight lateral or horizontal movements of the object in the image, or zooming, or tilting the image, or any kind of cropping will immensely change the values of the individual pixels. To the computer, it will seem like a completely different picture. Lastly, if the object in the image is in a different location, it will still be the same object in the same image but at different pixels and hence the computer will treat it differently.

Table

Description automatically generated with medium confidence

In order to combat such problems, Convolutional Neural Networks were designed. They use the concept of filters, which can recognise various characteristics of an image, such as horizontal contours, vertical contours, diagonals, spheres, corners, etc. Multiple filters are applied to the same image and these features/characteristics are determined for each image. The location of these filters w.r.t each other is what is actually used to recognise patters and then ultimately classify the objects.

The best part about these feature detectors is that we do not have to supply the exact features to the model. We can specify the filter size and the quantity(possible). These filters are in the form of matrices, which are used to make convolutions with the image which itself is in the form of a numeric matrix. In order to introduce non-linearity, we introduce Relu activation to get rid of all the negative values in the resultant matrices, which helps speed up training by increasing the compute speed.

Table

Description automatically generated

This part of object detection is known as Feature extraction, and consists of convolutional layers, and maxpooling layers (used to reduce the model complexity/trainable parameters and model overfitting by reducing the dimensions of the images). Multiple such blocks of Convolution layers and Maxpooling layers, along with normalisation of the weights are stacked on top of each other to detect these features of the image, and finally we flatten it’s output, in order to obtain a single dimension array. So the feature information from the input images are converted to single dimension arrays, and these can be fed to an artificial neural networks with Dense layers and some sort of regularisation, to reduce the model overfitting. This forms the basis of Convolutional Neural Networks.

A picture containing chart

Description automatically generated

**My Methodology** –

1. First import the dataset from Keras.
2. Prepare the images in the train and test by normalising them from 0-255 to between 0-1.
3. The test set is not to be used during K-Fold cross validation. It is to be kept aside for testing our **final model’s** accuracy at the end.
4. We will be using the 60,000 training images to divide them into **5 folds** each of 12,000 images.
5. We use 4 folds (48,000 images) for training and see the validation loss on the other remaining fold of 12000 images. This process is done 5 times since 5 different folds can become the validation set.
6. After each such Fold of this testing on 4 folds and validating on the remaining 1 fold, we measure the validation and training loss at each epoch.
7. We then plot these 5 validation and training loss plots on the Y-Axis, against Number of epochs on the X axis (Max-Epochs = 15).
8. The average Validation loss of all 5 models for each epoch is also calculated for all 5 models and plotted against the epochs.
9. After this K-Fold Cross validation, we have obtained 2 plots for each model, and we get a general idea how this model structure is performing on different training and validation data folds. We are now ready to use these hyperparameter values, and now train on the entire Training Data.
10. For the final model, out of the entire training data of 60,000 images, we set aside 10% of the training data(6000 images) for monitoring the training and validation loss for each epoch of this particular model and use the remaining 90% of training data (54,000 images) to train our final model.
11. We plot the training and validation loss for each epoch of this final model, and monitor the downwards trend of training loss, and find the best weights by monitoring the 10% data we used for validation.
12. We then make our predictions on the test set of 10,000 images using the final model, after carefully monitoring the validation loss and training loss of this final model, for each epoch.
13. These values of Training loss, training accuracy, validation loss, and validation accuracy were monitored and printed for each epoch of each model.
14. The accuracy of the final trained model is calculated on this 10,000 images test set and printed for all the 4 models.

**My Remark** – In steps 6 and 7 above, since Keras is able to print and plot the validation loss function values for all different 5 models, and I am also plotting them, one thing that can be observed is that these 5 different models take a different number of epochs for optimal training. Since all these 5 models have slightly different training and validation data, some models are able to converge early, and some take more epochs to train fully. Hence, when we take the mean of these 5 validation losses at each epoch and plot them, they values might not be commensurate with the actual training of each individual model, and therefore, might be incapable of reflecting the actual validation loss trend of each model separately, again because we have taken average Validation losses of all 5 models, which might take more or lesser epochs to train fully. Nevertheless, the individual training and validation losses are also plotted, in order to better visualize the downward trend of training loss and validation losses, along with average validation loss.

**Results -**

**Model 1** – This is a model with a structure as shown below.

A picture containing timeline

Description automatically generated

We are using **Adam Optimizer** with **Learning Rate = 0.001**, and **batch size = 32** for training the models.

**Adam Optimizer** is being used because it combines the best features from adaptive gradient algorithm, and root mean square propagation, and this algorithm calculates an exponential moving average of the gradient and the squared gradient.

**Learning rate** was experimented in the range of 0.01 to 0.001 and it was found that the best fitting was observed at 0.001.

**Batch size** – batch sizes of 16,32 and 64 were tried, and the best results were observed at 32.

**PLOT 1A – Training Loss in Blue and Validation Loss in Orange for 5 models –**

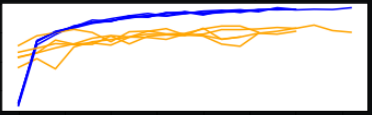
**Chart, line chart

Description automatically generated**

Remark – training loss is reducing over time, and validation loss also.

**Plot 1B – Training(Blur) and Validation Accuracy(Orange) over epochs for each of the 5 models –**

Sir I can very well recall that we have absolutely no need to view/analyse this graph since we already have the validation loss graph, but since this information is also measured and stored at each epoch, I am plotting this just in case.



Remark – training accuracy is increasing over time, and validation accuracies also.

**PLOT 2 -Average of these 5 different validation losses of 5 different models of K-Fold –**

**Line chart

Description automatically generated with medium confidence**

Remark - As expected, this graph is all over the place since all models are unique and have different validation loss trends at different epochs.

**PLOT 3A – Training the final model with 90% of training data and 10% used for monitoring the validation loss(Blue for training set loss and Orange for Validation set loss) –**

**Chart, line chart

Description automatically generated**

**PLOT 3B – Training(Blue) and Validation Accuracy(Orange) over epochs of this Final Model –**

**Icon

Description automatically generated**

Epoch 1/30

1688/1688 - 4s - loss: 0.1086 - accuracy: 0.9667 - val\_loss: 0.0549 - val\_accuracy: 0.9835

Epoch 2/30

1688/1688 - 4s - loss: 0.0435 - accuracy: 0.9871 - val\_loss: 0.0420 - val\_accuracy: 0.9888

Epoch 3/30

1688/1688 - 4s - loss: 0.0330 - accuracy: 0.9896 - val\_loss: 0.0484 - val\_accuracy: 0.9873

Epoch 4/30

1688/1688 - 4s - loss: 0.0261 - accuracy: 0.9919 - val\_loss: 0.0446 - val\_accuracy: 0.9902

Epoch 5/30

1688/1688 - 4s - loss: 0.0212 - accuracy: 0.9936 - val\_loss: 0.0514 - val\_accuracy: 0.9872

Epoch 6/30

1688/1688 - 4s - loss: 0.0179 - accuracy: 0.9941 - val\_loss: 0.0435 - val\_accuracy: 0.9885

Epoch 7/30

1688/1688 - 4s - loss: 0.0130 - accuracy: 0.9958 - val\_loss: 0.0386 - val\_accuracy: 0.9915

Epoch 8/30

1688/1688 - 4s - loss: 0.0140 - accuracy: 0.9959 - val\_loss: 0.0345 - val\_accuracy: 0.9933

Epoch 9/30

1688/1688 - 4s - loss: 0.0099 - accuracy: 0.9971 - val\_loss: 0.0567 - val\_accuracy: 0.9893

Epoch 10/30

1688/1688 - 4s - loss: 0.0080 - accuracy: 0.9975 - val\_loss: 0.0501 - val\_accuracy: 0.9913

Epoch 11/30

1688/1688 - 4s - loss: 0.0099 - accuracy: 0.9969 - val\_loss: 0.0373 - val\_accuracy: 0.9933

Epoch 12/30

1688/1688 - 4s - loss: 0.0082 - accuracy: 0.9977 - val\_loss: 0.0556 - val\_accuracy: 0.9897

Epoch 13/30

1688/1688 - 4s - loss: 0.0088 - accuracy: 0.9973 - val\_loss: 0.0306 - val\_accuracy: 0.9937

Epoch 14/30

1688/1688 - 4s - loss: 0.0061 - accuracy: 0.9981 - val\_loss: 0.0387 - val\_accuracy: 0.9932

Epoch 15/30

1688/1688 - 4s - loss: 0.0051 - accuracy: 0.9983 - val\_loss: 0.0502 - val\_accuracy: 0.9917

Epoch 16/30

1688/1688 - 4s - loss: 0.0068 - accuracy: 0.9979 - val\_loss: 0.0476 - val\_accuracy: 0.9913

Epoch 17/30

1688/1688 - 4s - loss: 0.0052 - accuracy: 0.9982 - val\_loss: 0.0446 - val\_accuracy: 0.9933

Epoch 18/30

1688/1688 - 4s - loss: 0.0059 - accuracy: 0.9982 - val\_loss: 0.0517 - val\_accuracy: 0.9923

Epoch 19/30

1688/1688 - 4s - loss: 0.0053 - accuracy: 0.9984 - val\_loss: 0.0531 - val\_accuracy: 0.9925

Epoch 20/30

1688/1688 - 4s - loss: 0.0038 - accuracy: 0.9989 - val\_loss: 0.0630 - val\_accuracy: 0.9913

Epoch 21/30

1688/1688 - 4s - loss: 0.0068 - accuracy: 0.9983 - val\_loss: 0.0610 - val\_accuracy: 0.9922

Epoch 22/30

1688/1688 - 4s - loss: 0.0045 - accuracy: 0.9988 - val\_loss: 0.0434 - val\_accuracy: 0.9943

Epoch 23/30

Restoring model weights from the end of the best epoch.

1688/1688 - 4s - loss: 0.0044 - accuracy: 0.9989 - val\_loss: 0.0515 - val\_accuracy: 0.9930

Epoch 00023: early stopping

**Accuracy on the Test Set of 10,000 images – 99.240**

**Model 2 –** Everything same as model 2, Just experimenting by removing the Batch normalisation after each CONV2D layer.

Text

Description automatically generated with medium confidence

We are using **Adam Optimizer** with **Learning Rate = 0.001**, and **batch size = 32** for training the models.

**PLOT 1A – Training Loss in Blue and Validation Loss in Orange for 5 models –**

**Chart, line chart

Description automatically generated**

Remark – Similar trend in reducing training loss, validation loss decreases initially and then starts to increase again, indicating we must stop the training now.

**Plot 1B – Training(Blue) and Validation Accuracy(Orange) over epochs for each of the 5 models –**

Chart, line chart

Description automatically generated

**PLOT 2 -Average of these 5 different validation losses of 5 different models of K-Fold –**

**Line chart

Description automatically generated**

**PLOT 3A – Training the final model with 90% of training data and 10% used for monitoring the validation loss(Blue for training set loss and Orange for Validation set loss) –**

**Chart, line chart

Description automatically generated with medium confidence**

Remark – Training loss reduces, and validation loss first reduces, then starts to increase.

**PLOT 3B – Training(Blue) and Validation Accuracy(Orange) over epochs of this Final Model –**

**Chart, line chart

Description automatically generated**

Remark – Accuracies increasing with epochs

Epoch 1/30

1688/1688 - 3s - loss: 0.1140 - accuracy: 0.9639 - val\_loss: 0.0445 - val\_accuracy: 0.9862

Epoch 2/30

1688/1688 - 3s - loss: 0.0413 - accuracy: 0.9870 - val\_loss: 0.0340 - val\_accuracy: 0.9897

Epoch 3/30

1688/1688 - 3s - loss: 0.0279 - accuracy: 0.9914 - val\_loss: 0.0308 - val\_accuracy: 0.9915

Epoch 4/30

1688/1688 - 3s - loss: 0.0206 - accuracy: 0.9934 - val\_loss: 0.0381 - val\_accuracy: 0.9890

Epoch 5/30

1688/1688 - 3s - loss: 0.0172 - accuracy: 0.9942 - val\_loss: 0.0375 - val\_accuracy: 0.9920

Epoch 6/30

1688/1688 - 3s - loss: 0.0135 - accuracy: 0.9951 - val\_loss: 0.0385 - val\_accuracy: 0.9908

Epoch 7/30

1688/1688 - 4s - loss: 0.0103 - accuracy: 0.9964 - val\_loss: 0.0429 - val\_accuracy: 0.9912

Epoch 8/30

1688/1688 - 3s - loss: 0.0096 - accuracy: 0.9968 - val\_loss: 0.0435 - val\_accuracy: 0.9915

Epoch 9/30

1688/1688 - 3s - loss: 0.0085 - accuracy: 0.9970 - val\_loss: 0.0397 - val\_accuracy: 0.9920

Epoch 10/30

1688/1688 - 3s - loss: 0.0069 - accuracy: 0.9980 - val\_loss: 0.0376 - val\_accuracy: 0.9928

Epoch 11/30

1688/1688 - 3s - loss: 0.0083 - accuracy: 0.9974 - val\_loss: 0.0459 - val\_accuracy: 0.9918

Epoch 12/30

Restoring model weights from the end of the best epoch.

1688/1688 - 3s - loss: 0.0054 - accuracy: 0.9984 - val\_loss: 0.0497 - val\_accuracy: 0.9920

Epoch 00012: early stopping

**Accuracy on the Test Set of 10,000 images – 99.070**

Remark - The following 2 model structures include a structure slightly similar to the infamous VGG structure, meaning one more set of 2 CONV2D layers and then a maxpooling layer, but not as many layers as the full-fledged VGG-16 model. This model was trained and tested purely for my own experimentation purpose by trying different optimizers and also added some also the use of Dropout Regularisation in the dense layers.

**MODEL 3 –**

**Timeline

Description automatically generated with medium confidence**

Using **Stochastic Gradient Descent with Learning rate = 0.001 and Momentum = 0.9**

**Stochastic Gradient Descent** was used this time in order to experiment a much simpler gradient descend algorithm since this is a simple problem and might actually converge easier.

**Dropout Regularisation** was used to reduce the model overfitting and 10% of the weights were dropped for this structure, in the fully connected layers at the end.

**PLOT 1A – Training Loss in Blue and Validation Loss in Orange for 5 models –**

**Chart

Description automatically generated with medium confidence**

Remark - Both training and validation losses decrease

**Plot 1B – Training(Blue) and Validation Accuracy(orange) over epochs for each of the 5 models –**

Line chart

Description automatically generated with medium confidence

Remark – Both training and Validation Accuracy increases

**PLOT 2 -Average of these 5 different validation losses of 5 different models of K-Fold –**

**A picture containing chart

Description automatically generated**

Remark – we can notice that the average validation loss is decreasing with more epochs.

**PLOT 3A – Training the final model with 90% of training data and 10% used for monitoring the validation loss(Blue for training set loss and Orange for Validation set loss) –**

**Chart, line chart

Description automatically generated**

**PLOT 3B – Training(Blue) and Validation Accuracy(Orange) over epochs of this Final Model –**

**Chart, line chart

Description automatically generated**

Epoch 1/30

1688/1688 - 5s - loss: 0.2235 - accuracy: 0.9302 - val\_loss: 0.1070 - val\_accuracy: 0.9695

Epoch 2/30

1688/1688 - 4s - loss: 0.0737 - accuracy: 0.9772 - val\_loss: 0.0546 - val\_accuracy: 0.9842

Epoch 3/30

1688/1688 - 4s - loss: 0.0537 - accuracy: 0.9832 - val\_loss: 0.0467 - val\_accuracy: 0.9868

Epoch 4/30

1688/1688 - 4s - loss: 0.0408 - accuracy: 0.9869 - val\_loss: 0.0483 - val\_accuracy: 0.9857

Epoch 5/30

1688/1688 - 4s - loss: 0.0363 - accuracy: 0.9887 - val\_loss: 0.0462 - val\_accuracy: 0.9868

Epoch 6/30

1688/1688 - 5s - loss: 0.0285 - accuracy: 0.9909 - val\_loss: 0.0352 - val\_accuracy: 0.9913

Epoch 7/30

1688/1688 - 5s - loss: 0.0250 - accuracy: 0.9914 - val\_loss: 0.0310 - val\_accuracy: 0.9918

Epoch 8/30

1688/1688 - 5s - loss: 0.0215 - accuracy: 0.9932 - val\_loss: 0.0350 - val\_accuracy: 0.9902

Epoch 9/30

1688/1688 - 4s - loss: 0.0183 - accuracy: 0.9936 - val\_loss: 0.0375 - val\_accuracy: 0.9902

Epoch 10/30

1688/1688 - 4s - loss: 0.0169 - accuracy: 0.9944 - val\_loss: 0.0367 - val\_accuracy: 0.9912

Epoch 11/30

1688/1688 - 4s - loss: 0.0138 - accuracy: 0.9955 - val\_loss: 0.0378 - val\_accuracy: 0.9898

Epoch 12/30

1688/1688 - 4s - loss: 0.0122 - accuracy: 0.9959 - val\_loss: 0.0327 - val\_accuracy: 0.9928

Epoch 13/30

1688/1688 - 4s - loss: 0.0102 - accuracy: 0.9965 - val\_loss: 0.0384 - val\_accuracy: 0.9915

Epoch 14/30

1688/1688 - 4s - loss: 0.0084 - accuracy: 0.9973 - val\_loss: 0.0347 - val\_accuracy: 0.9927

Epoch 15/30

1688/1688 - 4s - loss: 0.0083 - accuracy: 0.9972 - val\_loss: 0.0355 - val\_accuracy: 0.9918

Epoch 16/30

1688/1688 - 4s - loss: 0.0077 - accuracy: 0.9973 - val\_loss: 0.0406 - val\_accuracy: 0.9920

Epoch 17/30

Restoring model weights from the end of the best epoch.

1688/1688 - 4s - loss: 0.0062 - accuracy: 0.9979 - val\_loss: 0.0408 - val\_accuracy: 0.9905

Epoch 00017: early stopping

**Accuracy on the Test Set of 10,000 images – 98.92**

**MODEL 4 –** Same as model 3, except we will be using 64 filters instead of 32 filters in the final 2 CONV2D layers.

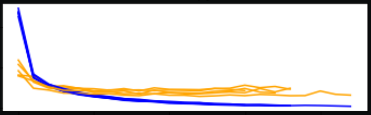
**A picture containing timeline

Description automatically generated**

Using **Stochastic Gradient Descent with Learning rate = 0.001 and Momentum = 0.9**

Remark – This was done in order to try to complicate the model, and this one change reduced the training time for each epoch from 4 seconds in model 3 to 50 seconds in model 4.

**PLOT 1A – Training Loss in Blue and Validation Loss in Orange for 5 models –**

****

**Plot 1B – Training(Blue) and Validation Accuracy(Orange) over epochs for each of the 5 models –**

A picture containing shape

Description automatically generated

**PLOT 2 -Average of these 5 different validation losses of 5 different models of K-Fold –**

**A picture containing chart

Description automatically generated**

**Remark –** The average validation loss in all the 5 Folds is observed to be reducing.

**PLOT 3A – Training the final model with 90% of training data and 10% used for monitoring the validation loss(Blue for training set loss and Orange for Validation set loss) –**

**Chart

Description automatically generated with medium confidence**

**PLOT 3B – Training(Blue) and Validation Accuracy(Orange) over epochs of this Final Model –**

**A picture containing line chart

Description automatically generated**

**Accuracy on the Test Set of 10,000 images – 99.090**

Epoch 1/30

1688/1688 - 46s - loss: 0.2107 - accuracy: 0.9334 - val\_loss: 0.0641 - val\_accuracy: 0.9792

Epoch 2/30

1688/1688 - 46s - loss: 0.0718 - accuracy: 0.9775 - val\_loss: 0.0600 - val\_accuracy: 0.9838

Epoch 3/30

1688/1688 - 46s - loss: 0.0508 - accuracy: 0.9837 - val\_loss: 0.0398 - val\_accuracy: 0.9885

Epoch 4/30

1688/1688 - 46s - loss: 0.0392 - accuracy: 0.9878 - val\_loss: 0.0458 - val\_accuracy: 0.9868

Epoch 5/30

1688/1688 - 47s - loss: 0.0321 - accuracy: 0.9894 - val\_loss: 0.0403 - val\_accuracy: 0.9885

Epoch 6/30

1688/1688 - 45s - loss: 0.0258 - accuracy: 0.9920 - val\_loss: 0.0415 - val\_accuracy: 0.9892

Epoch 7/30

1688/1688 - 45s - loss: 0.0205 - accuracy: 0.9933 - val\_loss: 0.0381 - val\_accuracy: 0.9902

Epoch 8/30

1688/1688 - 45s - loss: 0.0171 - accuracy: 0.9946 - val\_loss: 0.0389 - val\_accuracy: 0.9900

Epoch 9/30

1688/1688 - 46s - loss: 0.0151 - accuracy: 0.9950 - val\_loss: 0.0364 - val\_accuracy: 0.9912

Epoch 10/30

1688/1688 - 46s - loss: 0.0127 - accuracy: 0.9961 - val\_loss: 0.0388 - val\_accuracy: 0.9898

Epoch 11/30

1688/1688 - 51s - loss: 0.0102 - accuracy: 0.9966 - val\_loss: 0.0467 - val\_accuracy: 0.9883

Epoch 12/30

1688/1688 - 52s - loss: 0.0106 - accuracy: 0.9963 - val\_loss: 0.0410 - val\_accuracy: 0.9900

Epoch 13/30

1688/1688 - 52s - loss: 0.0083 - accuracy: 0.9972 - val\_loss: 0.0397 - val\_accuracy: 0.9905

Epoch 14/30

1688/1688 - 49s - loss: 0.0065 - accuracy: 0.9980 - val\_loss: 0.0439 - val\_accuracy: 0.9902

Epoch 15/30

1688/1688 - 49s - loss: 0.0056 - accuracy: 0.9981 - val\_loss: 0.0392 - val\_accuracy: 0.9915

Epoch 16/30

1688/1688 - 50s - loss: 0.0053 - accuracy: 0.9984 - val\_loss: 0.0497 - val\_accuracy: 0.9903

Epoch 17/30

1688/1688 - 50s - loss: 0.0052 - accuracy: 0.9981 - val\_loss: 0.0464 - val\_accuracy: 0.9913

Epoch 18/30

1688/1688 - 50s - loss: 0.0042 - accuracy: 0.9987 - val\_loss: 0.0463 - val\_accuracy: 0.9907

Epoch 19/30

Restoring model weights from the end of the best epoch.

1688/1688 - 51s - loss: 0.0030 - accuracy: 0.9990 - val\_loss: 0.0392 - val\_accuracy: 0.9923

Epoch 00019: early stopping

**REMARK –** Compared to models 1 and 2, these deeper models 3 & 4 have slightly lower accuracy, but the visualized trends of training and validation losses are much cleaner, since SGD is being used compared to Adam Optimizer in the earlier models.