The MNIST Database of Handwritten Digit Images

I. INTRODUCTION

Handwritten digit recognition is the ability of a computer to recognize the human handwritten digits from different sources like images, papers, touch screens, etc, and classify them into 10 predefined classes (0-9). This has been a topic of boundless-research in the field of deep learning. Digit recognition has many applications like number plate recognition, postal mail sorting, bank check processing, etc. In Handwritten digit recognition, we face many challenges because of different styles of writing of different peoples as it is not an Optical character recognition. Handwritten has been used as a test case for theories of pattern recognition and machine learning algorithms for many years. Historically, to promote machine learning and pattern recognition research, several standard databases have emerged in which the handwritten digits are pre-processed, including segmentation and normalization, so that researchers can compare recognition results of their techniques on a common basis. The freely available MNIST database of handwritten digits has become a standard for fast-testing machine learning algorithms for this purpose.

II. DATA

General site for the MNIST database: http: //yann.lecun.com/exdb/mnist

The MNIST database was constructed out of the original NIST database; hence, modified NIST or MNIST. There are 60,000 training images (some of these training images can also be used for cross-validation purposes) and 10,000 test images, both drawn from the same distribution. All these black and white digits are size normalized, and centred in a fixed size image where the centre of gravity of the intensity lies at the centre of the image with (28 * 28) pixels. Thus, the dimensionality of each image sample vector is 28 * 28 = 784, where each element is binary. This is a relatively simple database for people who want to try machine learning techniques and pattern recognition methods on real-world data while spending minimal efforts on pre-processing and formatting. Using the references provided on the Web site, students and educators of machine learning can also benefit from a rather comprehensive set of machine learning literature with performance comparison readily available.

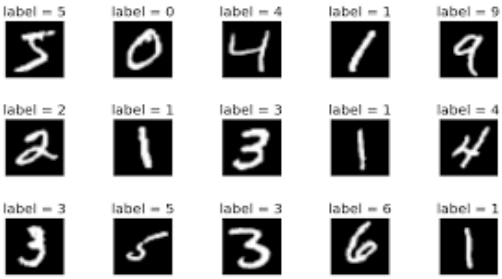


Fig. 1. Sample images of MNIST handwritten digit dataset

III. INTRODUCING CNN

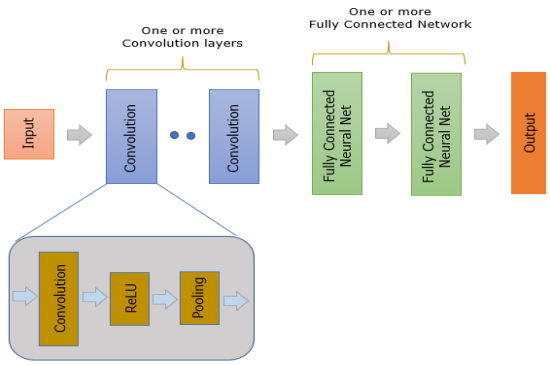


Figure 2. This figure shows the architectural design of CNN layers in the form of a Flow chart.

With time the numbers of fields are increasing in which deep learning can be applied. In deep learning, Convolutional Neural Networking (CNN) is being used for visual imagery analysing. Object detection, face recognition, robotics, video analysis, segmentation, pattern recognition, natural language processing, spam detection, topic categorization, regression analysis, speech recognition, image classifications are some of the examples that can be done using Convolutional Neural Networking. The accuracies in these fields including handwritten digits recognition using Deep

Convolutional Neural Networks (CNNs) have reached human level perfection. Mammalian visual systems' biological model is the one by which the architecture of the CNN is inspired. Cells in the cat's visual cortex are sensitized to a tiny area of the visual field identified which is recognized as the receptive field. It was found by D. H. Hubel et al. in 1062. The neocognitron, the pattern recognition model inspired by the work of D. H. Hubel et al was the first computer vision. It was introduced by Fukushima in 1980. In 1998, the frame work of CNNs is designed by LeCun et al which had seven layers of convolutional neural networks. It was adept in handwritten digits classification direct from pixel values of images. Gradient descent and back propagation algorithm is used for training the model. In handwritten recognition digits, characters are given as input. The model can be recognized by the system. A simple artificial neural network (ANN) has an input layer, an output layer and some hidden layers between the input and output layer. CNN has a very similar architecture as ANN. There are several neurons in each layer in ANN. The weighted sum of all the neurons of a layer becomes the input of a neuron of the next layer adding a biased value. In CNN the layer has three dimensions. Here all the neurons are not fully connected. Instead, every neuron in the layer is connected to the local receptive field. A cost function generates in order to train the network. It compares the output of the network with the desired output. The signal propagates back to the system, again and again, to update the shared weights and biases in all the receptive fields to minimize the value of cost function which increases the network's performance. The goal of this article is to observe the influence of hidden layers of a CNN for handwritten digits. We have applied a different type of Convolutional Neural Network algorithm on Modified National Institute of Standards and Technology (MNIST) dataset using TensorFlow, a Neural Network library written in python. The main purpose of this paper is to analyse the variation of outcome results for using a different combination of hidden layers of Convolutional Neural Network. Stochastic gradient and backpropagation algorithm IS used for training the network and the forward algorithm is used for testing.

IV. MODELING OF CONVOLUTIONAL NEURAL NETWORK TO CLASSIFY HANDWRITTEN DIGITS

The input layer consists of 28 by 28pixel images which mean that the network contains 784 neurons as input data. The input pixels are grayscale with a value 0 for a white pixel and 1 for a black pixel. Here, this model of CNN has five hidden layers. The first hidden layer is the convolution layer 1 which is responsible for feature extraction from an input data. This layer performs convolution operation to small localized areas by convolving a filter with the previous layer. In addition, it consists of multiple feature maps with learnable kernels and rectified linear units (ReLU). The kernel size determines the locality of the filters. ReLU is used as an activation function at the end of each convolution layer as well as a fully connected layer to enhance the performance of the model. The next hidden layer is the pooling layer 1. It reduces the output information from the convolution layer and reduces the number of parameters and computational complexity of the model. The different types of pooling are max pooling, min pooling, average pooling. Here, max pooling is used to subsample the dimension of each feature map. Convolution layer 2 and pooling layer 2 which has the same function as convolution layer 1 and pooling layer 1 and operates in the same way except for their feature maps and kernel size varies. A Flatten layer is used after the pooling layer which converts the 2D featured map matrix to a 1D feature vector and allows the output to get handled by the fully connected layers. A fully connected layer is another hidden layer also known as the dense layer. It is similar to the hidden layer of Artificial Neural Networks (ANNs) but here it is fully connected and connects every neuron from the previous layer to the next layer. In order to reduce overfitting, dropout regularization method is used at fully connected layer1. It randomly switches off some neurons during training to improve the performance of the network by making it more robust. This causes the network to become capable of better generalization and less compelling to overfit the training data. The output layer of the network consists of ten neurons and determines the digits numbered from 0 to 9. Since the output layer uses an activation function such as softmax, which is used to enhance the performance of the model, classifies the output digit from 0 through 9 which has the highest activation value. The MNIST handwritten digits database is used for the experiment. Out of 70,000 scanned images of handwritten digits from the MNIST database, 60,000 scanned images of digits are used for training the network and 10,000 scanned images of digits are used to test the network. The images that are used for training and testing the network all are the grayscale image with a size of 28×28 pixels. Character x is used to represent a training input where x is a 784-dimensional vector as the input of x is regarded as 28×28 pixels. The equivalent desired output is expressed by y(x), where y is a 10-

dimensional vector. The network aims is to find the convenient weights and biases so that the output of the network approximates y(x) for all training inputs x as it completely depends on weight values and bias values. To compute the network performances, a cost function is defined. To reduce the cost function to a smaller degree as a function of weight and biases, the training algorithm has to find a set of weight and biases which cause the cost to become as small as possible. This is done using an algorithm known as gradient descent. In other words, gradient descent is an optimization algorithm that twists its parameters iteratively to minimize a cost function to its local minimum. And to attain the global minimum of the cost function shown below:

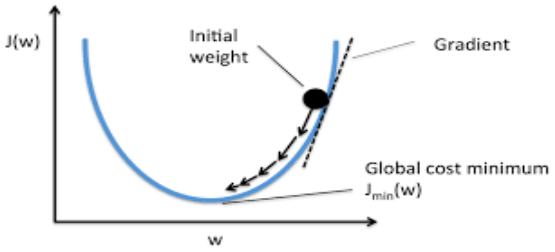
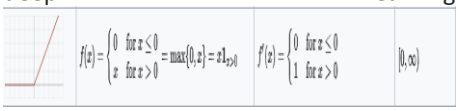


Fig. 3. Graphical Representation of Cost vs. Weight

V. THE ACTIVATION FUNCTIONS

1. **ReLU (Rectified Linear Unit) or Rectified Linear Activation function:**

   This activation function was first introduced to a dynamical network by Hahnloser et al. in 2000 with strong biological motivations and mathematical justifications. It was demonstrated for the first time in 2011 to enable better training of deeper networks, compared to the widely used activation functions prior to 2011, e.g., the logistic sigmoid (which is inspired by probability theory and logistic regression) and its more practical counterpart, the hyperbolic tangent. The rectifier is, as of 2017, the most , the most popular activation function for deep neural networks. A unit employing the rectifier is also called a **rectified linear unit** (**ReLU**). The main reason ReLU, despite being one of the best activation functions was not frequently used before recently. The reason for this was because it was not differentiable at the point 0. Researchers tended to use differentiable functions like sigmoid and tanh. However, it is now found that ReLU is the best activation function for deep learning.



The ReLU activation function is differentiable at all points except at 0. For values greater than 0, we just consider the max of the function. This can be written as:

$$f(x) = \max\{0, z\}$$

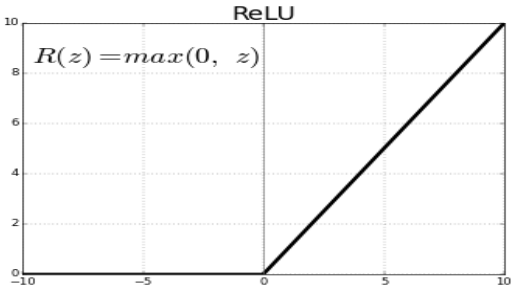


Fig. 4. Graphical Representation of ReLU Activation function

All the negative values default to 0, and the maximum for the positive number is taken into consideration.

For the computation of the backpropagation of neural networks, the differentiation for the ReLU is relatively easy. The only assumption we will make is the derivative at the point 0, which will also be considered as 0. This is usually not such a big concern, and it works well for the most part. The derivative of the function is the value of the slope. The slope for negative values is 0.0, and the slope for positive values is 1.0.

The main advantages of the ReLU activation function are:

1. Convolutional layers and deep learning: They are the most popular activation functions for training convolutional layers and deep learning models.

2. Computational Simplicity: The rectifier function is trivial to implement, requiring only a max() function.

3. Representational Sparsity: An important benefit of the rectifier function is that it is capable of outputting a true zero value.

4. Linear Behaviour: A neural network is easier to optimize when its behaviour is linear or close to linear. However, the main issue with the Rectified Linear Unit is that all the negative values become zero immediately, which decreases the ability of the model to fit or train from the data properly, that means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turn affects the resulting graph by not mapping the negative values

appropriately. This can however be easily fixed by using the different variants of the ReLU activation function like the Leaky ReLU.

2. SOFTMAX Activation Function:

When working on machine learning problems, specifically, deep learning tasks, Softmax activation function is a popular name. It is usually placed as the last layer in the deep learning model.It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.
Mathematically, Softmax is defined as,

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

where,

y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e{-}24$ (A very small value close to 0) **Note**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

What is the advantage of softmax?

The main advantage of using Softmax is the **output probabilities range**. The range will 0 to 1, and the sum of all the probabilities will be equal to one. If the softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.

VI. IMPLEMENTATION

We have discussed in detail about the implementation with Convolutional Neural Network Classifier explicitly below to create a flow of this analysis to create a fluent and accurate model.

I. PRE-PROCESSING

Pre-processing is an initial step in the machine and deep learning which focuses on improving the input data by reducing unwanted impurities and redundancy. To simplify and break down the input data we reshaped all the images present in the dataset in 2-dimensional images i.e (28,28,1). Each pixel value of the images lies between 0 to 255 so, we Normalized these pixel values by converting the dataset into 'float32' and then dividing by 255.0 so that the input features will range between 0.0 to 1.0. Next, we performed one-hot encoding to convert the label values into zeros and ones, making each number categorical, for example, an output value 4 will be converted into an array of zero and one i.e [0,0,0,0,1,0,0,0,0,0].

II. CONVOLUTIONAL NEURAL NETWORK

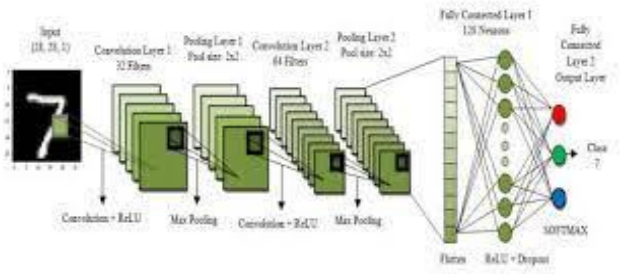


Fig. 5. Representation of CNN Algorithm

In the Model 1 case shown in figure 6, the first hidden layer is the convolutional layer 1 which is used for the feature extraction. It consists of 32 filters with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. The next hidden layer is the convolutional layer 2 consists of 64 filters with a kernel size of 3×3 pixels and ReLU. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. A regularization layer dropout is used next to the pooling layer 1 where it randomly eliminates 25% of the neurons in the layer to reduce overfitting. A flatten layer is used after the dropout which converts the 2D filter matrix into 1D feature vector before entering into the fully connected layers. The next hidden layer used after the flatten layer is the fully connected layer 1 consists of 128 neurons and ReLU. A dropout with a probability of 50% is used after the fully connected layer 1. Finally, the output layer which is used here as fully connected layer 2 contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A softmax activation function is incorporated with the output layer to output digit from 0 through 9. The CNN is fit over 15 epochs with a batch size of 100. The overall validation accuracy in the performance is found at 98.97%. At epoch 1 the minimum training

accuracy of 97.91% and at epoch 5, the maximum training accuracy is found 98.79%.The total test loss for this case is found approximately 0.0313917.

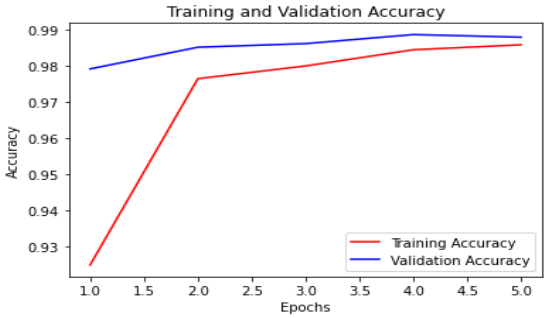


Fig. 6. Representation of Train and Test Accuracy graph for Model 1 (3 Hidden Layers with 2 Dropouts)

In the Model 2 case shown in figure 7, the first hidden layer is the convolutional layer 1 which is used for the feature extraction. It consists of 64 filters with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. The next hidden layer is the convolutional layer 2 consists of 64 filters with a kernel size of 3×3 pixels and ReLU. Next, again a pooling layer 2 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. The next hidden layer is the convolutional layer 3 consists of 128 filters with a kernel size of 3×3 pixels and ReLU. And lastly a pooling layer 3 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. A regularization layer dropout is used next to the pooling layer 1 where it randomly eliminates 25% of the neurons in the layer to reduce overfitting. A flatten layer is used after the dropout which converts the 2D filter matrix into 1D feature vector before entering into the fully connected layers. The next hidden layer used after the flatten layer is the fully connected layer 1 consists of 64 neurons and ReLU. A dropout with a probability of 50% is used after the fully connected layer 1. Finally, the output layer which is used here as fully connected layer 2 contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A softmax activation function is incorporated with the output layer to output digit from 0 through 9. The CNN is fit over 20 epochs with a batch size of 120. The overall validation accuracy in the performance is found at 98.42%. At epoch 1 the minimum training accuracy of 98.64% and at epoch 5, the maximum training accuracy is found 98.76%.The total test loss for this case is found approximately 0.05305422.

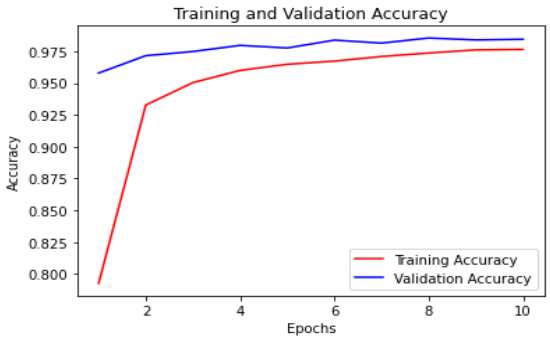


Fig. 7. Representation of Train and Test Accuracy graph for Model 2 (6 Hidden Layers with 2 Dropouts)

In the Model 3 case shown in figure 8, the first hidden layer is the convolutional layer 1 which is used for the feature extraction. It consists of 32 filters with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. The next hidden layer is the convolutional layer 2 consists of 64 filters with a kernel size of 3×3 pixels and ReLU. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. A flatten layer is used after the dropout which converts the 2D filter matrix into 1D feature vector before entering into the fully connected layers. The next hidden layer used after the flatten layer is the fully connected layer 1 consists of 128 neurons and ReLU. Finally, the output layer which is used here as fully connected layer 2 contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A softmax activation function is incorporated with the output layer to output digit from 0 through 9. The CNN is fit over 15 epochs with a batch size of 120. The overall validation accuracy in the performance is found at 98.65%. At epoch 1 the minimum training accuracy of 97.53% and at epoch 5, the maximum training accuracy is found 98.08%.The total test loss for this case is found approximately 0.0407725.

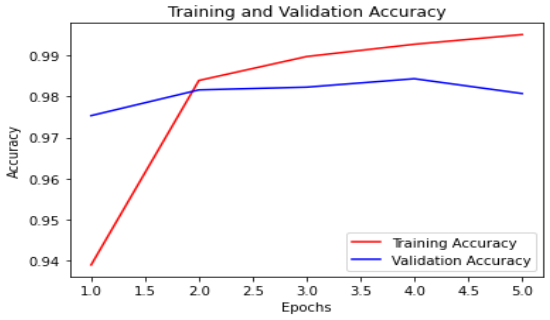


Fig. 8. Representation of Train and Test Accuracy graph for Model 3 (3 Hidden Layers without Dropouts)

In the Model 4 case shown in figure 9, the first hidden layer is the convolutional layer 1 which is used for the feature extraction. It consists of 64 filters with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. The next hidden layer is the convolutional layer 2 consists of 64 filters with a kernel size of 3×3 pixels and ReLU. Next, again a pooling layer 2 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. The next hidden layer is the convolutional layer 3 consists of 128 filters with a kernel size of 3×3 pixels and ReLU. And lastly a pooling layer 3 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. A flatten layer is used after the dropout which converts the 2D filter matrix into 1D feature vector before entering into the fully connected layers. The next hidden layer used after the flatten layer is the fully connected layer 1 consists of 64 neurons and ReLU. Finally, the output layer which is used here as fully connected layer 2 contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A softmax activation function is incorporated with the output layer to output digit from 0 through 9. The CNN is fit over 20 epochs with a batch size of 120. The overall validation accuracy in the performance is found at 98.35%. At epoch 1 the minimum training accuracy of 96.51% and at epoch 7, the maximum training accuracy is found 98.09%.The total test loss for this case is found approximately 0.05888309.

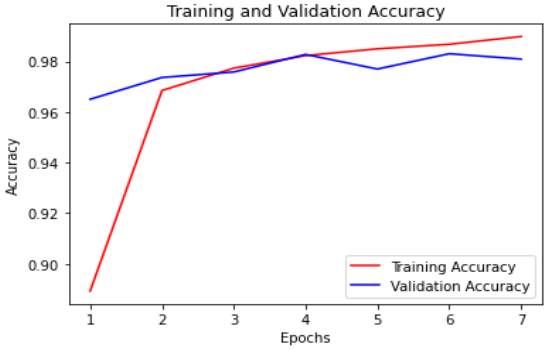


Fig. 9. Representation of Train and Test Accuracy graph for Model 4 (6 Hidden Layers without Dropouts)

In the Model 5 case shown in figure 10, the first hidden layer is the convolutional layer 1 which is used for the feature extraction. It consists of 32 filters with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. The next hidden layer is the convolutional layer 2 consists of 64 filters with a kernel size of 3×3 pixels and ReLU. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution. A flatten layer is used after the dropout which converts the 2D filter matrix into 1D feature vector before entering into the fully connected layers. The next hidden layer used after the flatten layer is the fully connected layer 1 consists of 128 neurons and ReLU. A dropout with a probability of 25% is used after the fully connected layer 1. Finally, the output layer which is used here as fully connected layer 2 contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A softmax activation function is incorporated with the output layer to output digit from 0 through 9. The CNN is fit over 15 epochs with a batch size of 120. The overall validation accuracy in the performance is found at 98.96%. At epoch 1 the minimum training accuracy of 97.58% and at epoch 5, the maximum training accuracy is found 98.85%. The total test loss for this case is found approximately 0.0330130904.

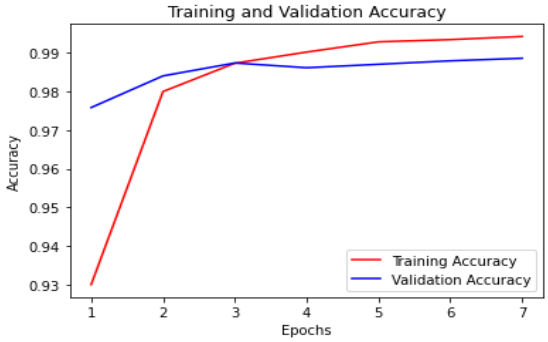


Fig. 10. Representation of Train and Test Accuracy graph for Model5 (3 Hidden Layers with 1 Dropouts)

In the Model 6 case shown in figure 11, the first hidden layer is the convolutional layer 1 which is used for the feature extraction. It consists of 64 filters with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. The next hidden layer is the convolutional layer 2 consists of 64 filters with a kernel size of 3×3 pixels and ReLU. Next, again a pooling layer 2 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. The next hidden layer is the convolutional layer 3 consists of 128 filters with a kernel size of 3×3 pixels and ReLU. And lastly a pooling layer 3 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial

size of the output of a convolution layer. A flatten layer is used after the dropout which converts the 2D filter matrix into 1D feature vector before entering into the fully connected layers. The next hidden layer used after the flatten layer is the fully connected layer 1 consists of 64 neurons and ReLU. A dropout with a probability of 25% is used after the fully connected layer 1. Finally, the output layer which is used here as fully connected layer 2 contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A softmax activation function is incorporated with the output layer to output digit from 0 through 9. The CNN is fit over 15 epochs with a batch size of 120. The overall validation accuracy in the performance is found at 98.69%. At epoch 1 the minimum training accuracy of 96.02% and at epoch 9, the maximum training accuracy is found 98.23%.The total test loss for this case is found approximately 0.0496253.

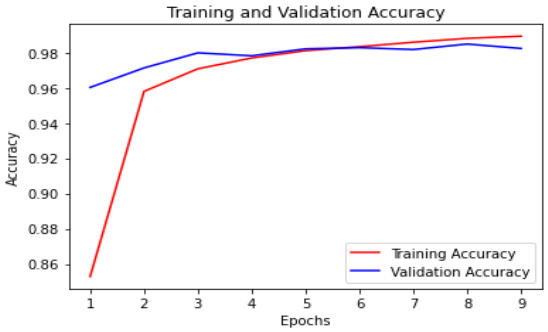


Fig. 11. Representation of Train and Test Accuracy graph for Model4 (6 Hidden Layers without Dropouts)

- **DATA AUGMENTATION**

In the Data Augmentation case shown in figure 12, the first three hidden layer is the convolutional layer 1,2,3 which is used for the feature extraction. The three convolutional layer consists of 64,64,128 filters respectively with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. Next, again two convolutional layer 4,5 which is used for the feature extraction and two convolutional layer consists of 128,256 filters respectively with the kernel size of 3×3 pixels and the rectified linear units (ReLU) is used as an activation function to enhance the performance. Next, a pooling layer 1 is defined where max pooling is used with a pool size of 2×2 pixels to minimize the spatial size of the output of a convolution layer. Again a convolutional layer of 256 filter is used with kernel size 3x3 pixel and the ReLU activation function. Next a Pooling layer is used where max pooling is used with a pool size of 2×2 pixels. A flatten layer is used after the dropout which converts the 2D filter matrix into 1D feature vector before entering into the fully connected layers. The next hidden layers used after the flatten layer are the four fully connected layer 1,2,3 consists of 512,128,64 neurons respectively and ReLU. . Finally, the output layer which is used here as fully connected layer 2 contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A softmax activation function is incorporated with the output layer to output digit from 0 through 9.

Next, we did Data Augmentation with Image Data Generator. We randomly rotate some training images by 10 degrees, randomly Zoom by 10% some training image, randomly shift images horizontally by 10% of the width and randomly shift images vertically by 10% of the height We did not apply a vertical flip nor horizontal flip since it can lead to misclassify symmetrical numbers such as 6 and 9. In order to make the optimizer converge faster and closest to the global minimum of the loss function, we used an annealing method of the learning rate (LR). The LR is the step by which the optimizer walks through the 'loss landscape'. The higher LR, the bigger are the steps and the quicker is the convergence. However, the sampling is very poor with an high LR and the optimizer could probably fall into a local minima. It's better to have a decreasing learning rate during the training to reach efficiently the global minimum of the loss function. To keep the advantage of the fast computation time with a high LR, i decreased the LR dynamically every steps (epochs) depending if it is necessary (when accuracy is not improved). With the ReduceLROnPlateau function from Keras.callbacks, we choose to reduce the LR. The Data Augmentation is fit over 10 epochs with a batch size of 200. The overall validation accuracy in the performance is found at 99.44%. At epoch 1 the minimum training accuracy of 98.65% and at epoch 10, the maximum training accuracy is found 99.44%.The total test loss for this case is found approximately 0.01903.

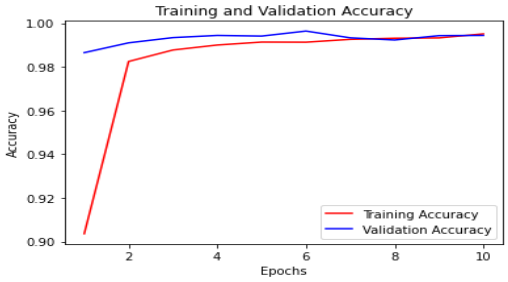


Fig. 12. Representation of Train and Test Accuracy graph for Data Augmentation (6 Hidden Layers without Dropouts)

VII. RESULTS

Table 1 shows the different Accuracy and Loss of Different Models:

Model	Accuracy	Loss
Model 1 (3 Hidden Layers with 2 Dropouts)	98.97%	0.03139
Model 2 (6 Hidden Layers with 2 Dropouts)	98.42%	0.05305
Model 3 (3 Hidden Layers without Dropouts)	98.65%	0.04077
Model 4 (6 Hidden Layers without Dropouts)	98.35%	0.05888
Model 5 (3 Hidden Layers with 1 Dropout)	98.96%	0.03301
Model 6 (6 Hidden Layers with 1 Dropout)	98.69%	0.04962
Data Augmentation	99.44%	0.01903

Table.1. Accuracy and Loss of Models

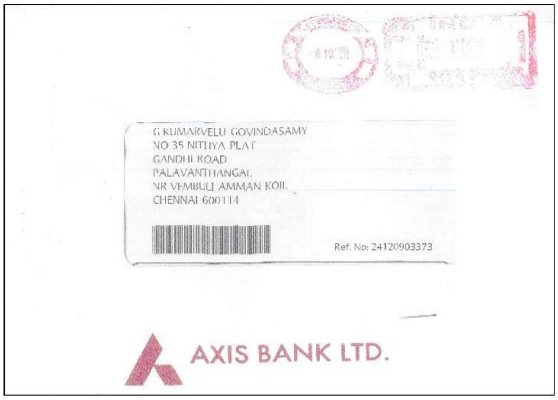
From the above Table we can conclude that Model 1, 1.e 3 hidden layers and 2 dropouts has good accuracy of 98.97%, but after Data Augmentation the Accuracy is 99.44% and loss is 0.01903. Thus, after Regularization, i.e. Data Augmentation the Accuracy has improved and we can perform well with this model as shown in figure 13.

Fig. 13. Prediction Result

VIII. CONCONCLUSION

In this, we have implemented seven models for handwritten digit recognition using MNIST datasets, based on deep learning algorithm (CNN). We compared them based on their characteristics to appraise the most accurate model among them. We have found that Data Augmentation gave the most accurate results for handwritten digit recognition. So, this also makes us conclude that CNN is best suitable for any type of prediction problem including image data as an input. Next, by comparing execution we have concluded that increasing the number of epochs without changing the configuration is useless because of the limitation of a certain model.

IX. APPLICATIONS



(a) National ID number recognition system
(b) Postal office automation with code number recognition on Envelope
(c) Automatic license plate recognition
(d) Bank automation.
(e) Data entry

----xxx----