Spring 2019

Nitrogenase Iron Protein Detection using Neural Network

Ishan Shinde
San Jose State University

Nitrogenase Iron Protein Detection using Neural Network

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by Ishan Shinde

May 2019

The Designated Project Committee Approves the Project Titled

Nitrogenase Iron Protein Detection using Neural Network

by

Ishan Shinde

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2019

Dr. Philip Heller	Department of Computer Science
Dr. Sami Khuri	Department of Computer Science
Dr. Nada Attar	Department of Computer Science

ABSTRACT

Nitrogenase Iron Protein Detection using Neural Network

By Ishan Shinde

Nitrogenase Iron Protein (*nifH*) is the enzyme responsible for nitrogen fixation. Microbes with *nifH* gene are responsible for injecting reduced nitrogen into the biosphere, which is essential for all living things. Obtaining sequences from GenBank database is problematic due to annotation errors, nomenclature variation and paralogues. One possible solution could be to retrieve sequences from the GenBank database and use a sequence classifier to label the sequences. In this research, we convert sequences to images and build a *nifH* sequence classifier using image processing and convolutional neural network. We built a *nifH* classification model which can classify sequences with an accuracy of around 99%.

ACKNOWLEDGMENTS

I would like to thank my family and friends for motivating me and always keeping me boosted throughout my journey. I would also like to thank Dr. Philip Heller for providing me with a great problem statement. Dr. Heller not only provided me a clean dataset but helped me explore options to integrate image classification problem with sequence classification problem.

I would also like to thank my parents, Dr. Mohan Shinde and Dr. Anita Shinde, who stood with me through the entire journey without whom the project wouldn't have been possible. They constantly pushed me and made me stay positive in my ups and downs. I would also like to thank my sister Mrs. Prachi Jain for always being there for me and motivate me.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1. Introduction

Dinitrogen (N_2) is the most abundant gas in the atmosphere. This nitrogen gas is accessible just to microorganisms with the ability of biological nitrogen fixation, the decrease of atmospheric N_2 to ammonia [1]. So, for its consumption, N_2 needs to be converted to ammonia by the process of Nitrogen fixation[2].

Nitrogen fixation is the process of converting atmospheric nitrogen to nitrogen compounds useful for other chemical processes such as ammonia, nitrate, nitrogen dioxide as shown in Figure 1 and therefore this process is vital to sustaining life on Earth. Nitrogen fixation makes up for the deficiency of nitrogen relative to phosphorus in many lakes, contributing to phosphorus-limited status of these systems [3] . Nitrogen fixation is also a major input of nitrogen to individual aquatic ecosystems allowing primary production to continue when fixed nitrogen supplies are depleted.

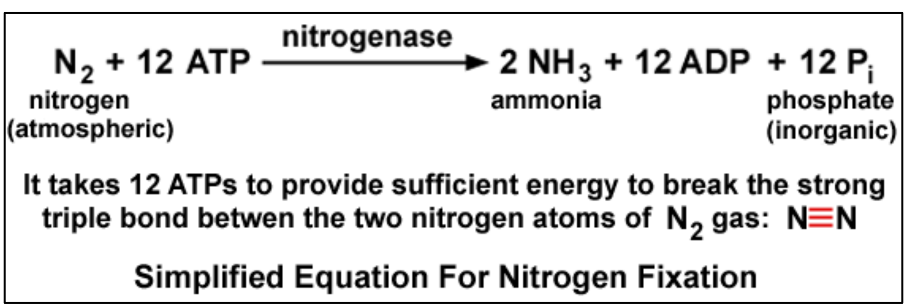

Figure 1: Equation of nitrogen fixation by nitrogenase

Nitrogenase is the enzyme responsible for nitrogen fixation [1] . The nitrogenase iron protein gene *nifH* is considered as one of the oldest existing and functional genes in the history of gene evolution[4] . Nitrogenase is found to be present in diverse lineages of prokaryotes and is considered ancient[5] . Previous studies [6] have shown that the nitrogen-fixing populations and diverse habitats supporting nitrogen fixation are far more variable than previously documented. Table 1 shows the classification of major types of *nifH* sequences collected [7].

There were a number of advances including degenerate PCR primers [8], where degenerate oligonucleotides were used to amplify *nifH* gene from the marine *cyanobacterium Trichodesmium thiebautii*. In another study, huge number of rice researchers in China have led to the explosion of *nifH* sequences giving rise to a large dataset [9]. All known nitrogenases comprises of two components: component I (dinitrogenase or Fe-Mo protein), an alpha2beta2 tetramer encoded by *nifD* and *nifK* genes, and component II (dinitrogenase reductase or Fe protein) a homodimer encoded by *nifH* gene.

Table 1: Classification of major types of nifH sequences obtained from marine picoplankton and zooplankton samples [7]

Phylogenetic affiliation	GenBank accession no.	Sample type	Location/date	Depth (m)	Sequence(s)[a]
Heterocystous cyanobacteria	AF059624	Plankton	BATS station/August 8, 1996	1	BT1101
	AF059625	Plankton	BATS station/February 15, 1996	200	BT1118
Unicellular cyanobacteria	AF016616	Picoplankton	Atlantic Ocean (19.1°N 58.2°W)/May 29, 1994	160	AO11
	AF059626	Picoplankton	HOT station (22°45′N 158°W)/May 22, 1996	175	HT1103
	AF059627	Picoplankton	HOT station (22°45′N 158°W)/May 6, 1997	25	HT1150
Filamentous nonheterocystous cyanobacteria	AF059628	Plankton	HOT station (22°45′N 158°W)/May 6, 1997	25	HT1169
α proteobacteria	AF016612	Picoplankton	BATS station/February 15, 1996	10	BT13
	AF016610, AF016611	Picoplankton	BATS station/February 15, 1996	75	BT11, BT12
	AF016615	Picoplankton	Atlantic Ocean (23.1°N 69.4°W)/May 25, 1994	125	AO12
	AF059644	Diatom collections	Pacific Ocean/September 1, 1992	Surface	PO3120
	AF059645	Diatom (*Hemiaulus*) collections	Pacific Ocean/September 22, 1995		PO3133
β proteobacteria	AF016618	Picoplankton	Atlantic Ocean (17.3°N 53.2°W)/May 31, 1994	40	AO14
	AF059643	Picoplankton	Near Bahama Islands/September 4, 1991	Surface	BH1132
	AF016602	*Acartia tonsa*	Gulf of Mexico (29°N 84°W)	2	GM26
	AF059647	Diatom collections	Pacific Ocean/September 4, 1992		PO3137
	AF059646	Diatom collections	Pacific Ocean/September 4, 1992		PO3135
Distantly related to γ proteobacteria	AF016592	*Labidocera aestiva*	Gulf of Mexico (29°N 84°W)	2	GM23
	AF016601, AF016600	*A. tonsa*	Gulf of Mexico (29°N 84°W)	2	GM24, GM21
γ proteobacteria	AF016603, AF016609	*A. tonsa*	Gulf of Mexico (29°N 84°W)	2	GM25, GM22
	AF016614	Picoplankton	Atlantic Ocean (23°N 69.2°W)/May 25, 1994	160	AO13
	AF016617	Picoplankton	Atlantic Ocean (19.1°N 58.23°W)/May 29, 1994	70	AO16
	AF059622	Picoplankton	Atlantic Ocean (18–23°N 43–62°W)/October 21, 1996	20	AO1104
	AF059623	Picoplankton	Atlantic Ocean (18–23°N 43–62°W)/October 21, 1996	40	AO1113
	AF016613	Picoplankton	Atlantic Ocean (23°N 69.2°W)/May 25, 1994	125	AO15
	AF059629	Picoplankton	HOT station (22°45′N 158°W)/May 6, 1997	50	HT1177
	AF059621	Picoplankton	Atlantic Ocean (18–23°N 43–62°W)/October 21, 1996	0	AO1102
Unidentified relatives of clostridia and sulfate reducers	AF016595, AF016598, AF016597, AF016599, AF016596	*A. tonsa*	Gulf of Mexico (29°N 84°W)	2	GM215, GM216, GM27, GM29, GM210
	AF016594, AF016593	*L. aestiva*	Gulf of Mexico (29°N 84°W)	2	GM212, GM214

[a] Sequence prefixes: AO, Atlantic Ocean; GM, Gulf of Mexico; BT, BATS station; PO, Pacific Ocean; HT, HOT station; BH, near Bahama Islands.

Due to the fact that genome forms the blue-print of the cell, protein sequences and nucleic acid are of immense interest to molecular biologists [10] . Currently, the most direct approach to get labelled *nifH* sequences include a database search [11] . GenBank is one such sequence database. But the problem with the database is that the size of the GenBank database containing *nifH* sequences is growing at an alarming rate, and not all the methods are reliable due to annotation errors, nomenclature variation and paralogues [2] . Another issue is the structure and tools provided by GenBank are not efficient enough to search by function.

A lot of methods have been devised to detect *nifH* sequences. In 2009, Gaby and Buckley [12] published a database of *nifH* sequences creating a database of around 17000 sequences [2] initially which continued to grow. To search for a particular sequence over the database, BLAST [13] which searches for sequence similarity rather than function was used. But approaches based on BLAST are likely to be over-sensitive. Another database called the fugene database, was created using hidden Markov model which classifies according to similarity to a composite profile model [2] . Finally, ARBitrator (a software pipeline) was developed, which required a little human intervention and retrieves up-to-date *nifH* sequences within a few hours [2] . ARBitrator uses similarity to representative *nifH* sequences and to *nifH* conserved domain in order to classify.

Since each of the previously discussed techniques to detect *nifH* sequences have long execution time, this provided motivation to explore more efficient techniques. In a previous study [10] , the neural network uses 3-layered, feed-forward, back propagation configuration to detect *nifH* sequences. In this study, the input sequences are encoded into input vectors and fed to neural network. Sensitivity close to 90% is achieved, suggesting a huge scope of improvement in this area.

In this paper, an approach to detect *nifH* sequences has been discussed in which given sequences (both positive and negative *nifH* sequences) are converted into black and white images, and Convolutional Neural Network model is implemented using LeNet-5 CNN architecture. The model is being trained by scaling the images to different scales over fixed number of 25 epochs (phase-1 testing). After phase-1 testing, the best performing scale 7X500 is being used over varying number of epochs and the best results are discussed

2. Methods

Dataset Description

The size of the dataset is described as below:

Positive *nifH* sequences: 40,754

Negative *nifH* sequences: 2,013

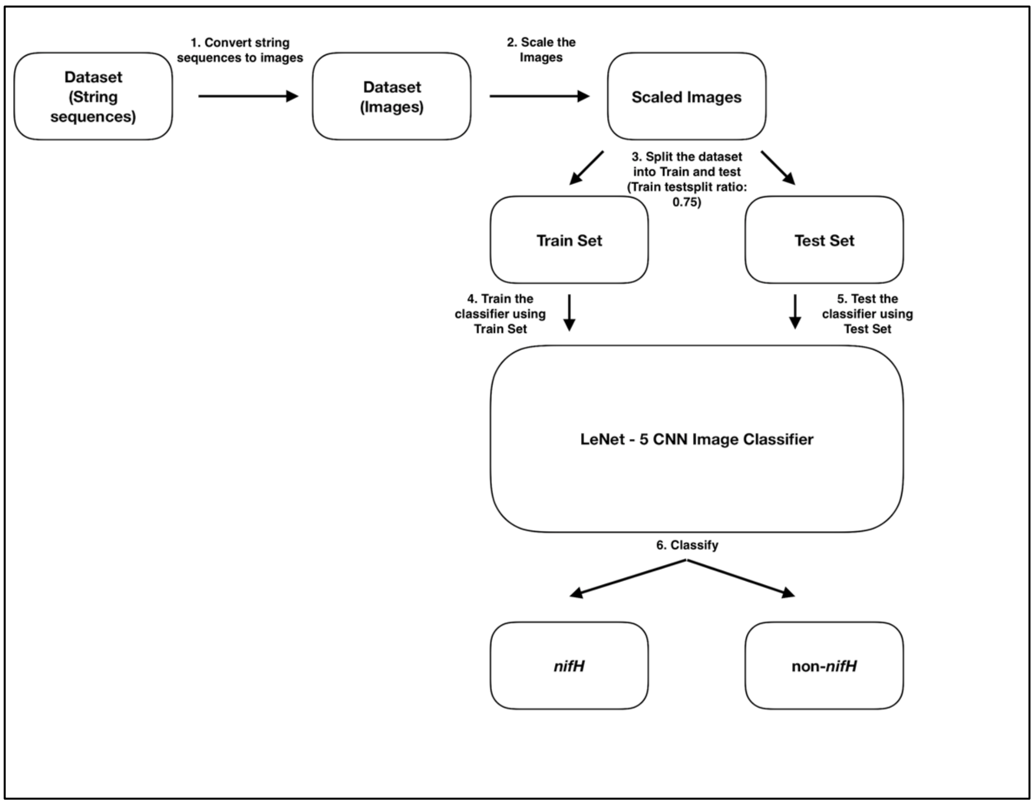

Figure 2: Experiment workflow

The workflow is shown in Figure 2. First, the dataset is cleaned. After the cleaning of dataset, sequences are converted to images. Then the black and white images are used to train the Convolutional Neural Network model.

2.1 Clean the dataset

The dataset used was in the csv format containing 2 columns. Column 1 containing unique sequence identifier and Column 2 containing the sequence string as shown below.

Table 2: Sample Data

Sequence Name	String Sequence
AAA67137.1	STTSCNISVALAKRGKKVLQIGCDPKHDSTFTLTGFLIPTIIDTL QEKDFHYEDIWPEDVIYKGYGGVDCVEAGGPPAGAGCGGYV VGETVKLLKELNAFDEYDVILFDVLG

On careful analysis of the strings, the following issues were observed due to which the dataset needed to be cleaned:

- String contained lower case characters
- String contained characters apart from (A-Z, a-z)
- Some strings were empty
- Duplicate strings

Cleaning/Processing:

- All the strings were checked for characters having ASCII in the range 65-90 (A-Z) and 97-122 (a-z) and all other ASCII values were discarded

- Lowercase characters if found in the string were all converted to Uppercase
- Empty strings/strings containing no alphabet characters were discarded
- All the duplicate strings after doing the above operations were discarded

2.2 Convert Protein Sequences to Images

Following algorithm was used to convert all the positive and negative *nifH* sequences to images:

Algorithm:

1. Given a string sequence containing only Uppercase alphabets (A-Z), convert all the characters to ASCII
2. A 2-dimensional array is created and each character's ASCII value (65-90) is being converted to binary (7 Bites) and for every character's binary equivalent, a row is added in the matrix.

   Depending upon the length of the string sequence, a binary matrix will be formed containing 7 columns and n rows where n is length of string
3. In the binary matrix, replace all 1's with 255 which corresponds to white pixel
4. Create an image using the matrix where 0 corresponds to a black pixel and 255 corresponds to a white pixel

Example:

Consider a string: ABCDDCBAD

1. Convert to ASCII values

A: 1000001

B: 1000010

C: 1000011

D: 1000100

D: 1000100

C: 1000011

B: 1000010

A: 1000001

D: 1000100

2. Form a 2D matrix with dimensions: 9 X 7

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

3. Replace all 1's with 255's

$$\begin{bmatrix} 255 & 0 & 0 & 0 & 0 & 0 & 255 \\ 255 & 0 & 0 & 0 & 0 & 255 & 0 \\ 255 & 0 & 0 & 0 & 0 & 255 & 255 \\ 255 & 0 & 0 & 0 & 255 & 0 & 0 \\ 255 & 0 & 0 & 0 & 255 & 0 & 0 \\ 255 & 0 & 0 & 0 & 0 & 255 & 255 \\ 255 & 0 & 0 & 0 & 0 & 255 & 0 \\ 255 & 0 & 0 & 0 & 0 & 0 & 255 \\ 255 & 0 & 0 & 0 & 255 & 0 & 0 \end{bmatrix}$$

4. Create image

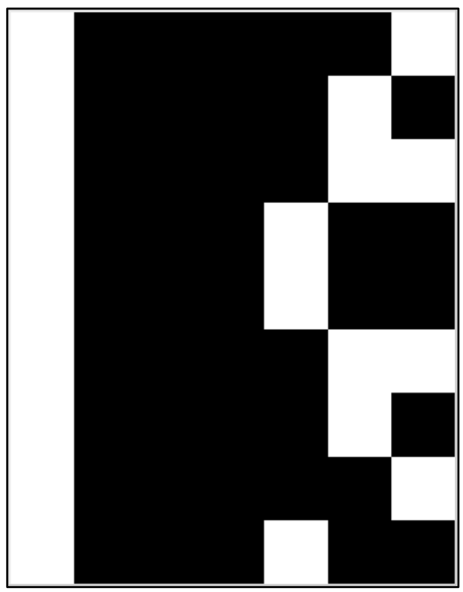

Figure 3: 9 X 7 Image

5. Same image scaled to 200 X 200

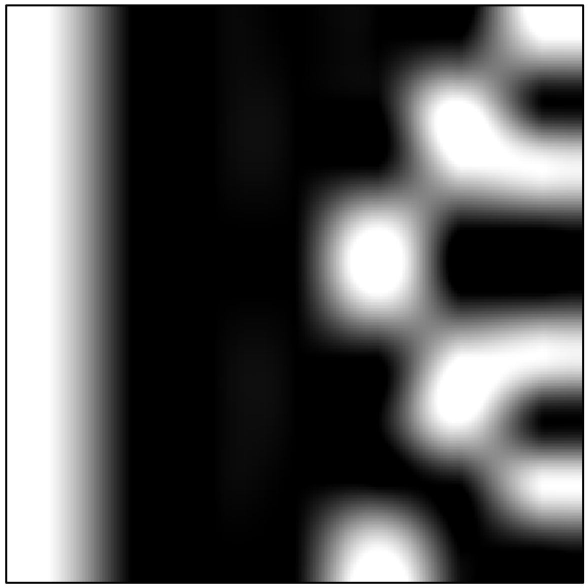

Figure 4: 7 X 9 Image scaled to 200 X 200

2.3 Building CNN model

2.3.1 Artificial Neural Network Introduction

Artificial Neural Networks are computational processing systems influenced by the mechanism of biological nervous system[14] . They basically consist of densely interconnected neurons, collecting input and delivering optimized output.

Basic structure of ANN is shown in Figure 5. Input is loaded in form of a multidimensional vector which gets distributed to the hidden layers. The hidden layers take input from the input layer, learning about the image features and finally delivering the output/classification.

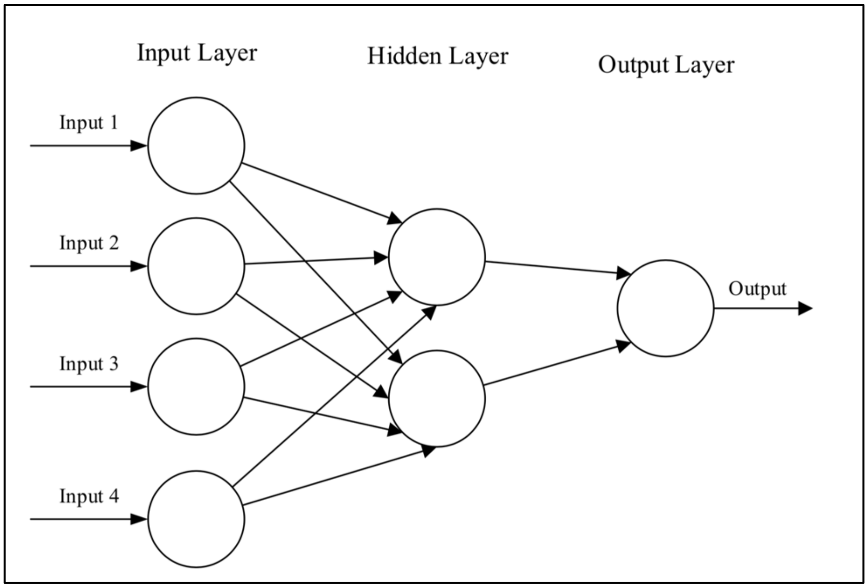

Figure 5: A Simple 3 layered feedforward Neural Network [15]

Inputs are being provided to the perceptron which mimics the behavior of a neuron. Each perceptron in one layer is connected to perceptron in another layer through a weighted link. When a perceptron transmits a signal to a forward perceptron, the output goes to the transfer function, generating aggregated output which goes through the activation function as shown in Figure 6.

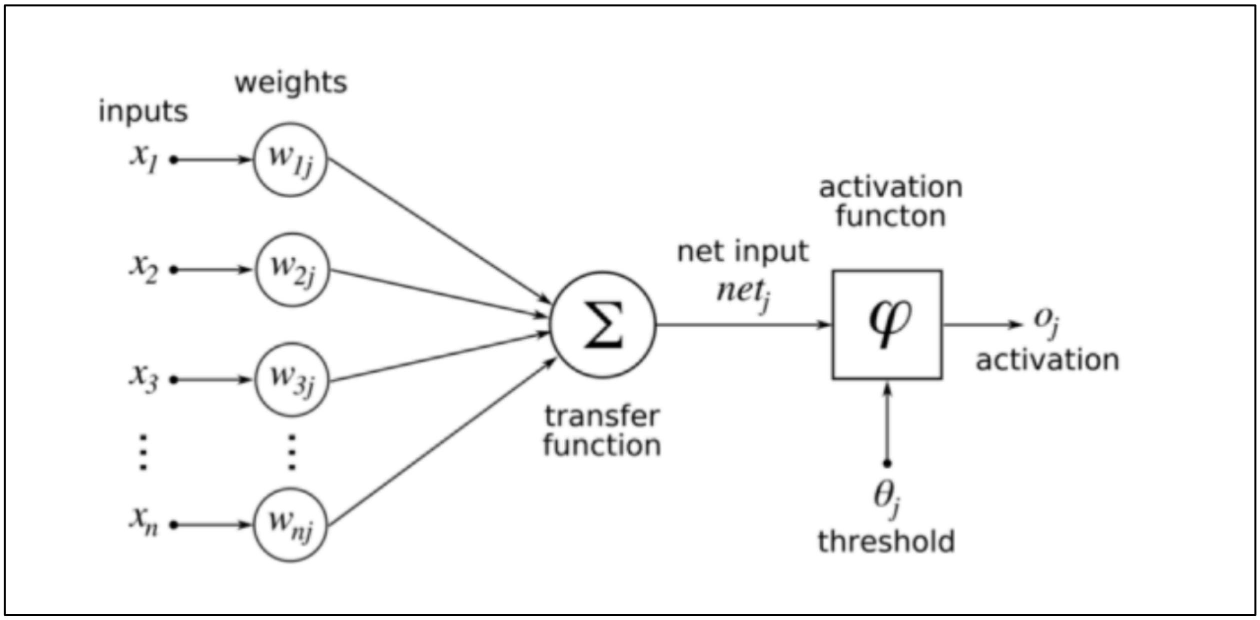

Figure 6: Scoring in a Neural Network [16]

2.3.2 Overfitting

In order to deal with high dimensional images, one possible solution seems to increase number of neurons and layers. But it is not a practical idea since this will result in increased time and computational complexity.

Another reason is to avoid overfitting is that it restricts the learning of a model and occurs when a function too closely fit to a limited set of data points. This constitutes the main reason for reduced complexity of ANN.

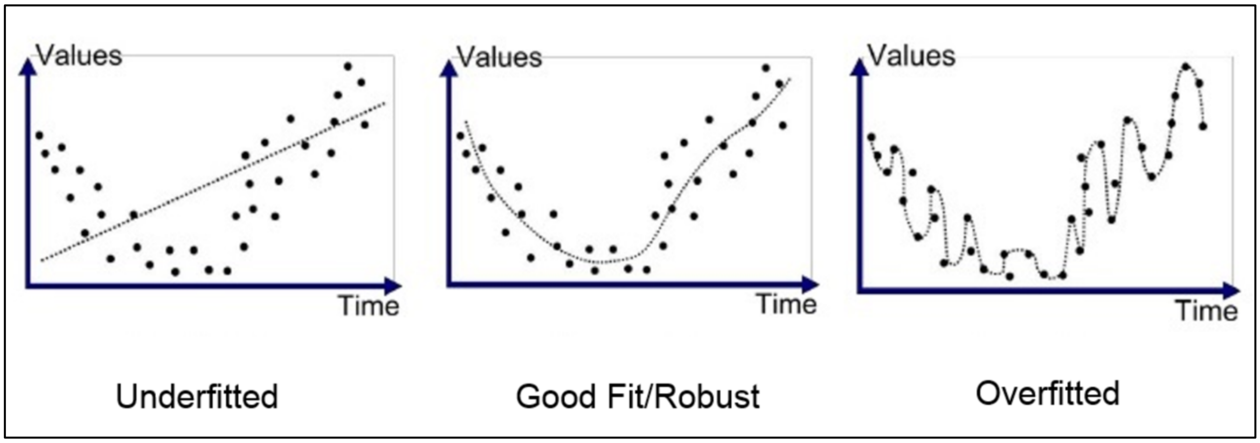

Figure 7:Comparision of Underfitted, good fit and overfitted model [15]

2.3.3 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are almost equivalent to conventional ANNs since they contain neurons that self-optimize through learning[14] . Every neuron will get inputs and perform some operation. The last layer contains the loss function, and all the traditional ANN rules are applied.

CNNs are specifically used for pattern recognition and win over traditional ANN which tend to struggle with computational complexity required to compute image data. In case of images of the order of 28X28, ANNs can be used but as the dimension of the images increase to say 64x64, the number of weights on a single increase to 12,288.

2.3.3.1 CNN Architecture

The key difference which sets apart CNN is the neurons in the different layers of CNN are organized into 3 dimensions: height, width and depth of the image. The neurons within a layer will connect to a partial region of preceding layer.

Architecture

CNNs are composed of 3 variety of layers:

- Convolutional layers
- Pooling layers
- Fully-connected layers

These layers when clubbed together, give rise to a CNN model. For e.g., a simple CNN configuration for MNIST classification is shown in Figure 8.

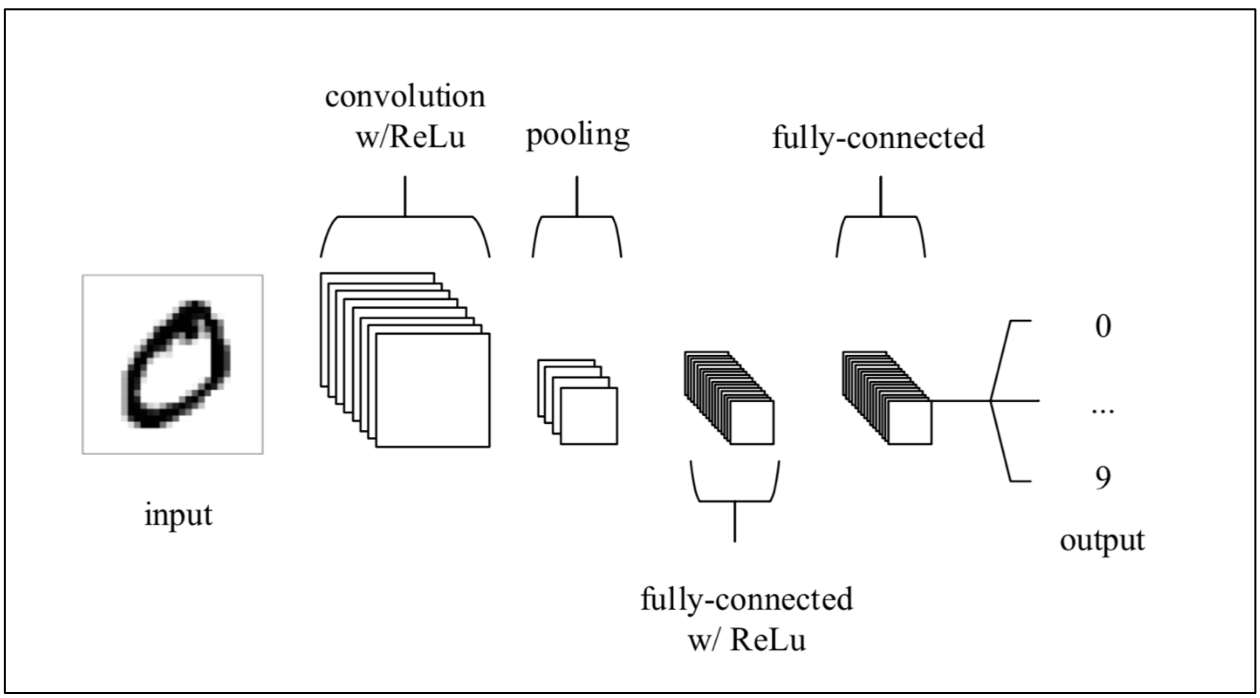

Figure 8: CNN Architecture for MNIST classification [15]

The functionality of CNN is:

1. **Input layer** holds the pixel values of the image
2. **Convolutional layer** determines the output of neurons linked to input layer through links having weights. There is an element wise activation function which decides whether to propagate the signal to the next layer
3. **Pooling layer** performs down sampling along the spatial dimensionality of input reducing number of parameters within the activation
4. **Fully-connected layer** generate class scores, used for classification

Convolutional Layer

Convolutional layer comprises of learnable **kernels**. Kernels are small in size but are spread through the entirety of the input. At the point when the information hits a convolutional layer, the layer convolves each channel over the spatial dimensionality of input to deliver a 2D activation map. The filter convolves around the input, calculating scalar product in the grid to generate a pooled vector as shown in Figure 9. Each kernel has its corresponding activation map forming full output volume from the convolutional layer.

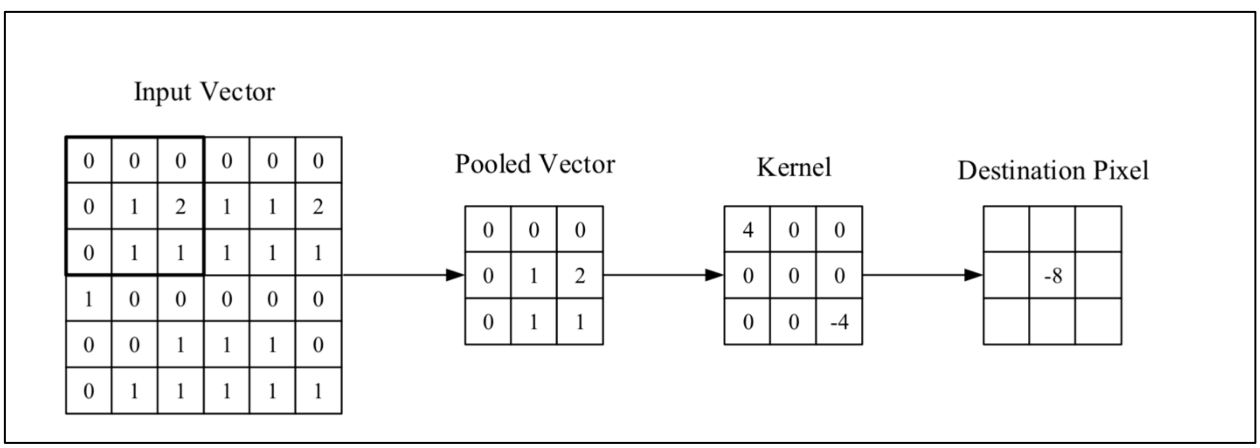

Figure 9: Visual representation of convolutional layer [15]

Main role of this layer is to lower the complexity of the model by optimizing the output. Convolutional layer uses three hyperparameters to control the complexity:

- Depth
- Stride
- Zero-padding

Depth

Output depth is controlled by number of neurons within the layer. In traditional ANN's, each neuron in a layer is connected to neuron in preceding layer. Reducing this hyperparameter not only means reducing the complexity of the model by bringing down the count of neurons but will also result in reduced pattern recognition capabilities of the model.

Stride

Stride is the depth around spatial dimensionality of input for the receptive field. Lower stride results in heavily overlapped receptive field generating large activations. On the other hand, high stride reduces overlapping and generates an output of lower spatial dimensions.

Zero-padding

Zero-padding involves padding of the input resulting in better controlling of dimensionality of output volumes.

The parameters are correlated with each other using the formula:

$$\frac{(V - R) + 2Z}{S + 1}$$

Where:

V: Input Volume size (height x width x depth)

R: Receptive field size

Z: Amount of zero padding

S: Stride size

Pooling payer

Pooling layer plays a vital role in reducing dimensionality of representation, reducing the number of parameters and computational complexity of the model.

The pooling layer works over every initiation map in the information, and scales its dimensionality utilizing the "Maximum" work. In many CNNs, these come as max-pooling layers with parts of a dimensionality of 2 × 2 connected with a stride of 2 along the spatial elements of the input. This scales the enactment map down to 25% of the first size - while keeping up the depth volume to its standard size.

Fully-connected layer

Fully-connected layer contains neurons connected to all neurons on the adjacent layer, skipping connections to neurons in the same layer.

2.3.3.2 LeNet

LeNet is a classical image classification deep learning CNN. Following are the network architecture[17]:

1. Baseline Linear Classifier
2. One-Hidden-Layer fully connected multilayer NN
3. Two-Hidden-Layer fully connected multilayer NN
4. LeNet-1
5. LeNet-4
6. Boosted LeNet-4
7. LeNet-5

Baseline Linear Classifier

Baseline linear classifier is a linear classifier. Each input pixel contributes to a weighted sum for each output unit. For classification, the output with the highest value is used. In the Figure 10, for a 20 x 20 pixel image (400 pixels), the image is converted to a 1-D array of 400 length connected to a 10-output vector.

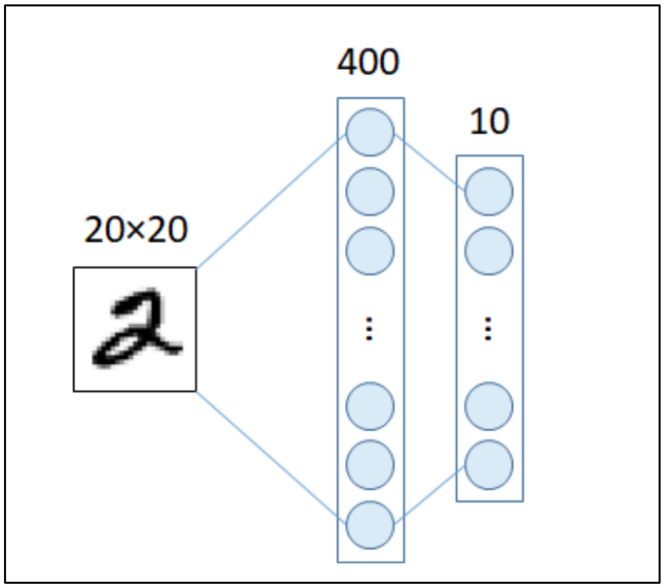

Figure 10: Baseline Linear Classifier [17]

One-Hidden-Layer fully connected multilayer NN

One-Hidden-Layer fully connected multilayer NN is a Baseline Linear Classifier with one hidden layer sandwiched between the input and the output layer as shown in Figure 11. The hidden layer in this case contains between 300-1000 neurons.

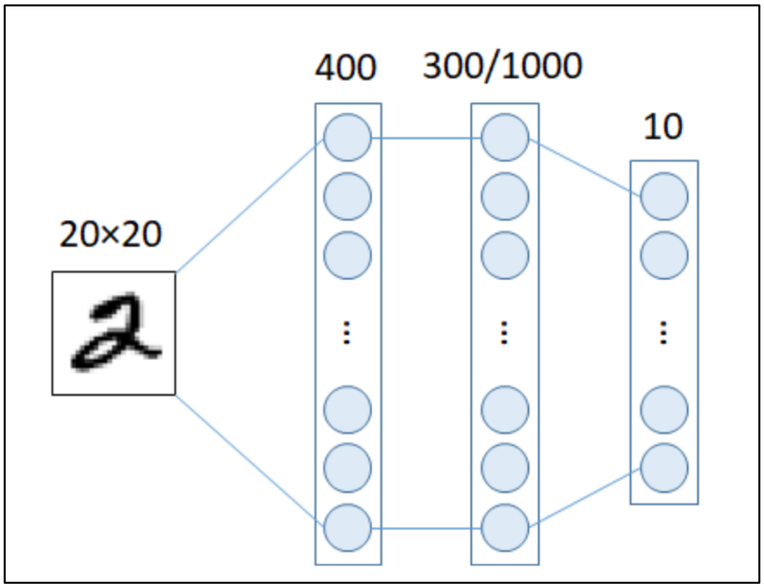

Figure 11: One-Hidden-Layer fully connected multilayer NN [17]

Two-Hidden-Layer fully connected multilayer NN

Two-Hidden-Layer fully connected multilayer NN contains two hidden layers in between the input and the output layer. Hidden layer 1 can contain between 300-1000 neurons and hidden layer 2 can contain between 100-150 neurons as shown in Figure 12.

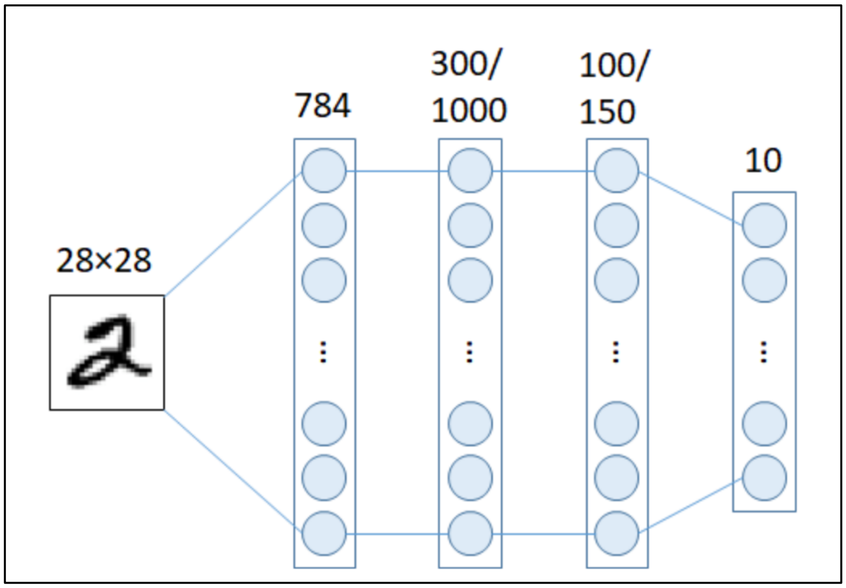

Figure 12: Two-Hidden-Layer fully connected multilayer NN [17]

LeNet-1

In the LeNet-1 architecture, average pooling layers were used outputting the average values of 2 X 2 feature maps.

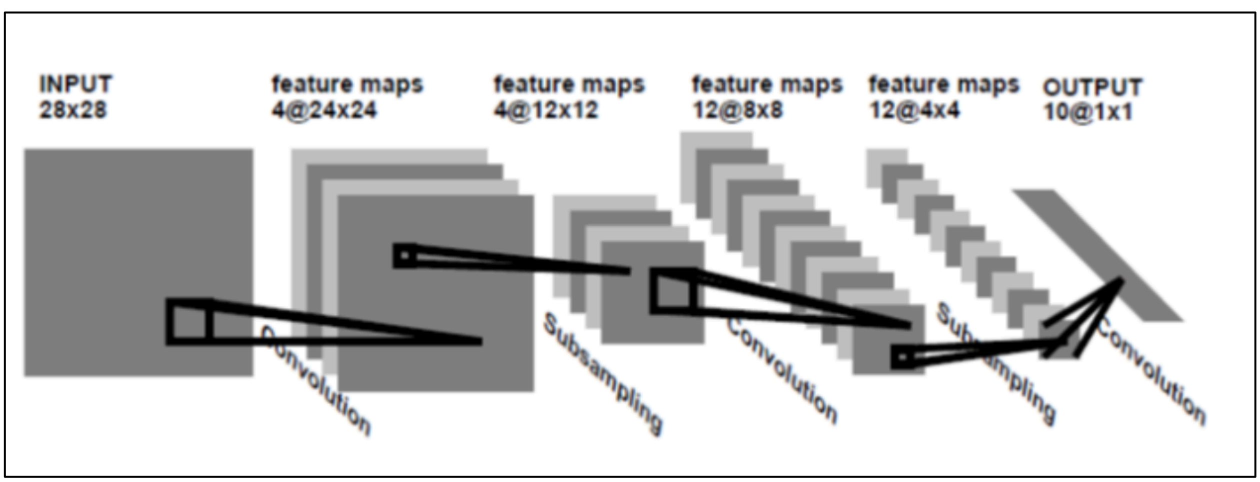

Figure 13: LeNet-1 Architecture [17]

As shown in Figure 13, the model consists of the following configuration:

Input Image: 28 X 28

Convolutional Layer 1: 4 (24 x 24) feature maps convolutional layer (size = 5 X 5)

Pooling layer 1: 2 X 2 size

Convolutional Layer 2: 12 (8 X 8) feature maps convolutional layer (size = 5 X 5)

Pooling layer 2: 2 X 2 size

Output layer

LeNet-4

LeNet-4 contains two fully connected layers as compared to one in LeNet-1 as shown in Figure 14.

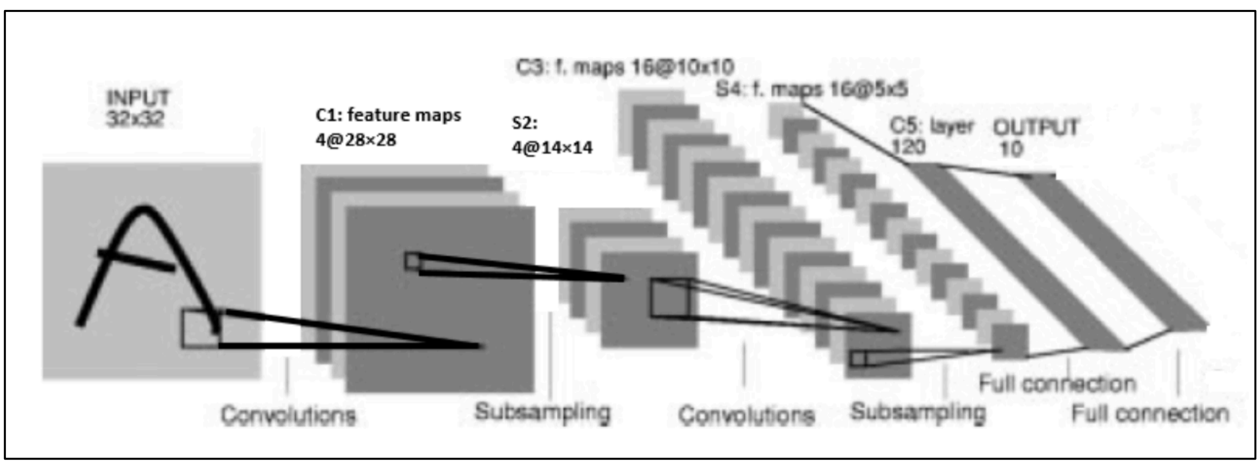

Figure 14: LeNet-4 Architecture [17]

Configuration:

Input image: 32 X 32

Convolutional Layer 1: 4 (28 x 28) feature maps convolutional layer (size = 5 X 5)

Pooling layer 1: 2 X 2 size

Convolutional Layer 2: 16 (10 X 10) feature maps convolutional layer (size = 5 X 5)

Pooling layer 2: 2 X 2 size

Output layer 1: Fully connected to 120 neurons

Output layer 2: Fully connected to 10 neurons

Boosted LeNet-4

Technique of boosting is used to improve the performance of combined weak classifiers to get accurate results. In case of boosted LeNet-4, performance of 3 LeNet-4 is combined, and the value of the maximum one is used for classification. The architecture is shown in Figure 15.

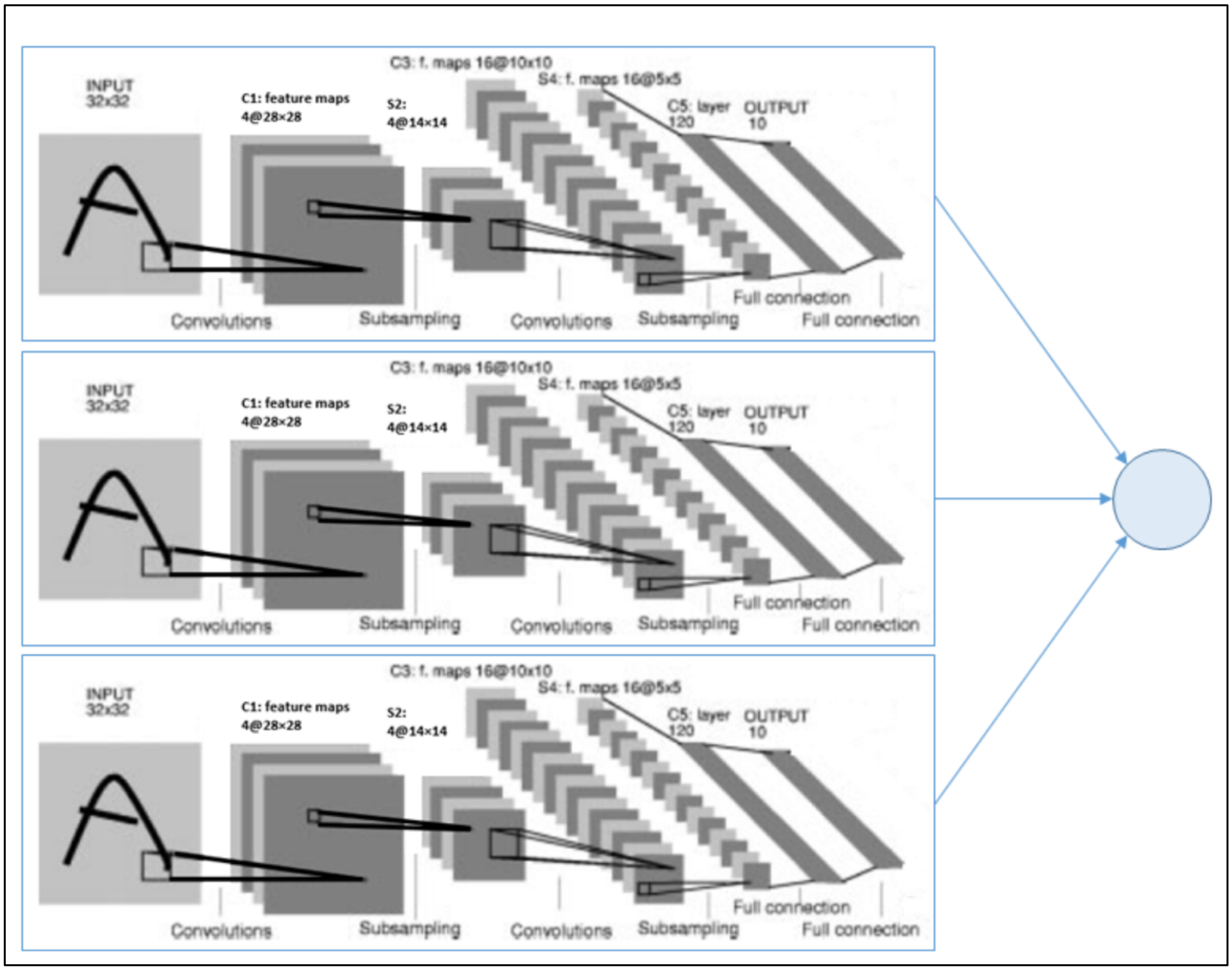

Figure 15: Boosted LeNet-4 Architecture [17]

Note: In order to develop an efficient model given the dataset and problem statement might require a lot of experimentation, tweaking different parameters for optimized results.

LeNet-5

LeNet-5 consists of two sets of convolutional and pooling layers followed by a flattening convolutional layer, then two fully connected layers and finally a soft-max classifier. The architecture is shown in Figure 16.

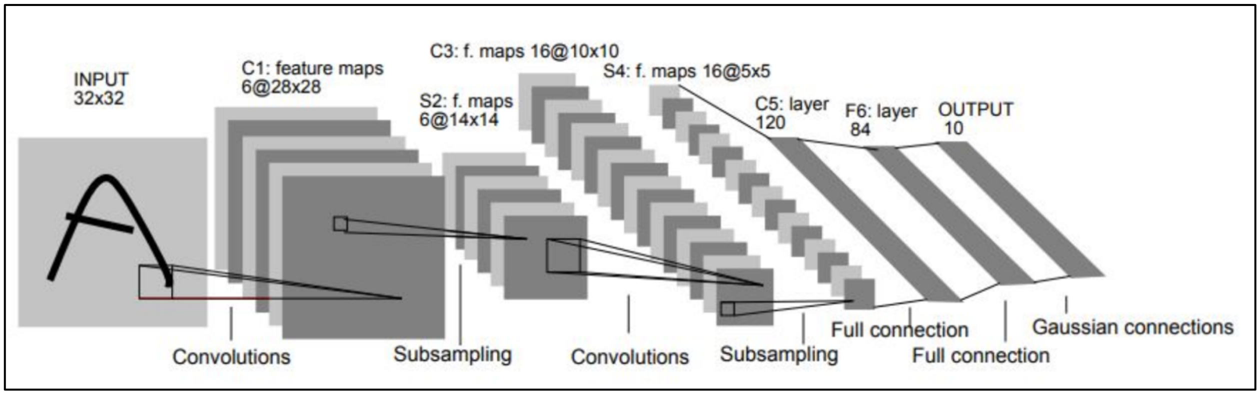

Figure 16: LeNet-5 Architecture [17]

Configuration:

Input image: 32 X 32

Convolutional Layer 1: 6 (28 x 28) feature maps convolutional layer (size = 5 X 5)

Pooling layer 1: 2 X 2 size

Convolutional Layer 2: 16 (10 X 10) feature maps convolutional layer (size = 5 X 5)

Pooling layer 2: 2 X 2 size

Output layer 1: Fully connected to 120 neurons

Output layer 2: Fully connected to 80 neurons

Output layer 3: Fully connected to 10 neurons

First Layer

The input image 32 X 32 X 1 is passed through the first convolutional layer having 6 feature maps, having size 5X5 and a stride value of 1. After the processing of image through this layer, image dimension changes from 32 X 32 X 1 to 28 X 28 X 6 as shown in Figure 17.

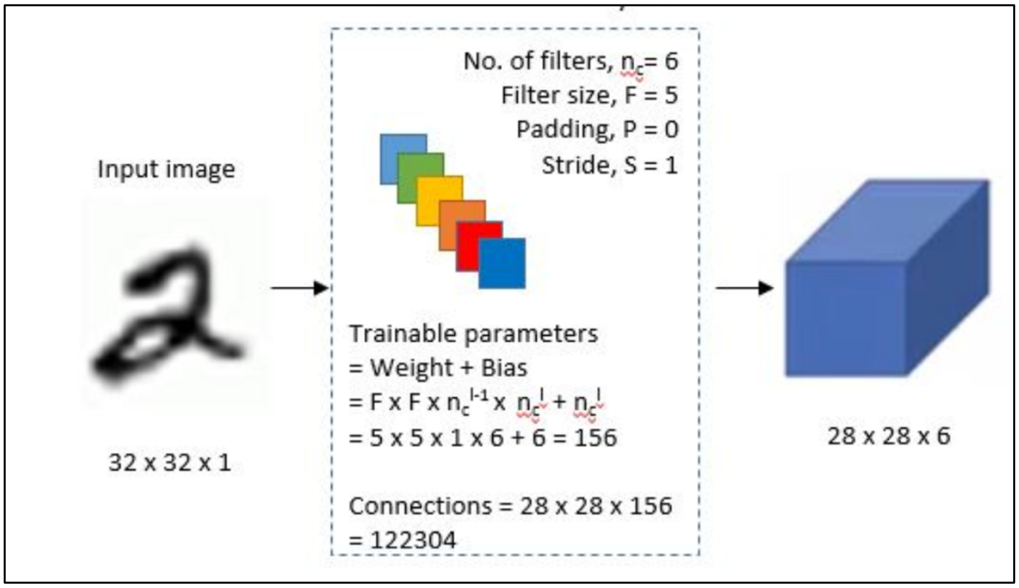

Figure 17: C1: Convolutional Layer [18]

Second Layer

In the second layer, pooling/sub-sampling occurs with a filter size of 2 X 2 and a stride of 2. The output of this layer is an image of dimension 14 X 14 X 6 as shown in Figure 18.

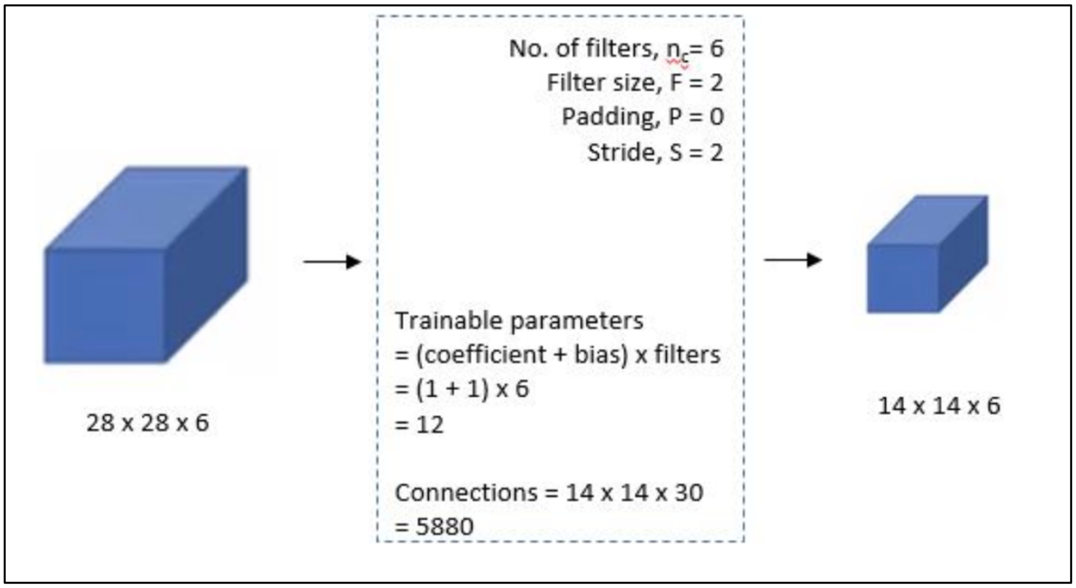

Figure 18: S2 Average pooling layer [18]

Third Layer

The third layer is a convolutional layer consisting of 16 feature maps each of size 5 X 5 and a stride value of 1. Here, 10 feature maps are connected to 6 of the previous layer as shown in Figure 19.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

Figure 19: Each column indicate which feature map in S2 are combined by the units in a particular feature map of C3[19]

The input to this layer is 14 X 14 X 6 and the output is 10 X 10 X 16 as shown in Figure 20.

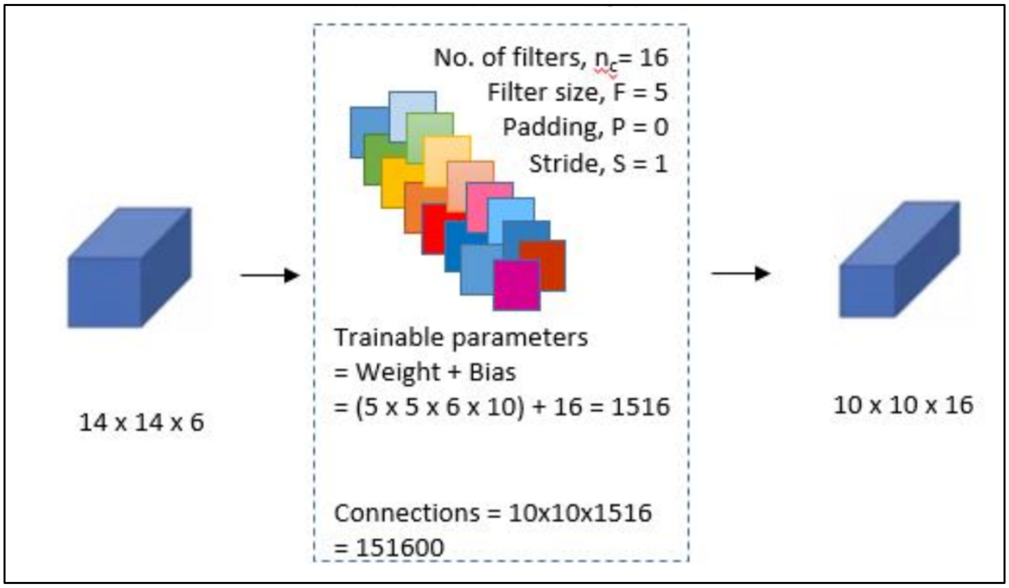

Figure 20: C3: Convolutional Layer [18]

Fourth Layer

In the fourth layer again, pooling happens using the filter size of 2 X 2 and a stride value of 2. It resembles second layer, the only difference being it has 16 feature maps. The output of this layer is an image of dimension 5 X 5 X 16 as shown in Figure 21.

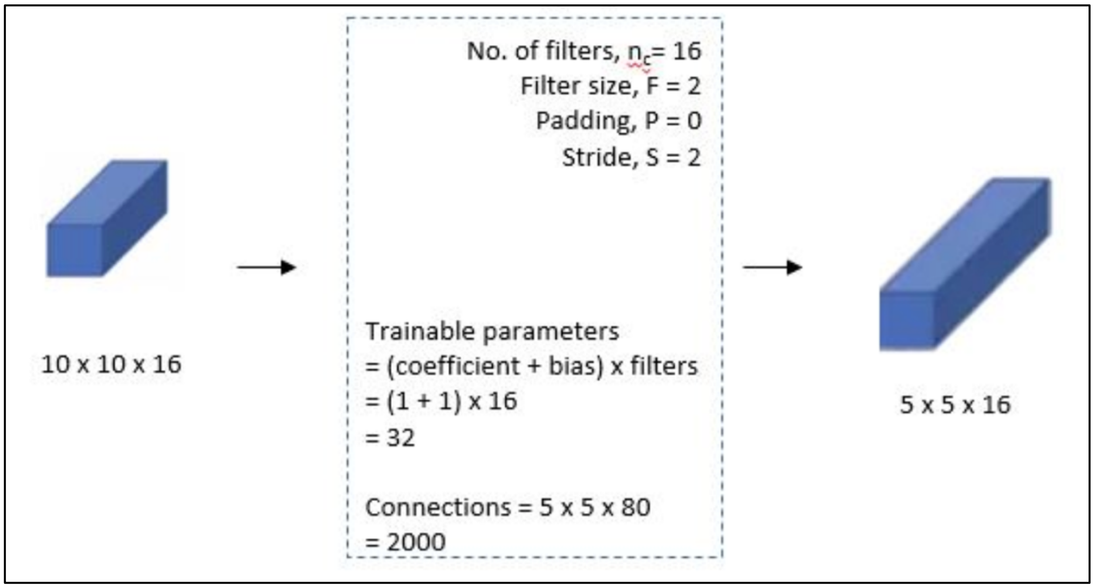

Figure 21: S4: Average pooling layer [18]

Fifth Layer

The fifth layer is basically a fully connected layer having 120 feature maps of size 1 X 1 each. Each unit in C5 (120 in total) is connected to all 400 (5 X 5 X 16) nodes from the fourth layer as shown in Figure 22.

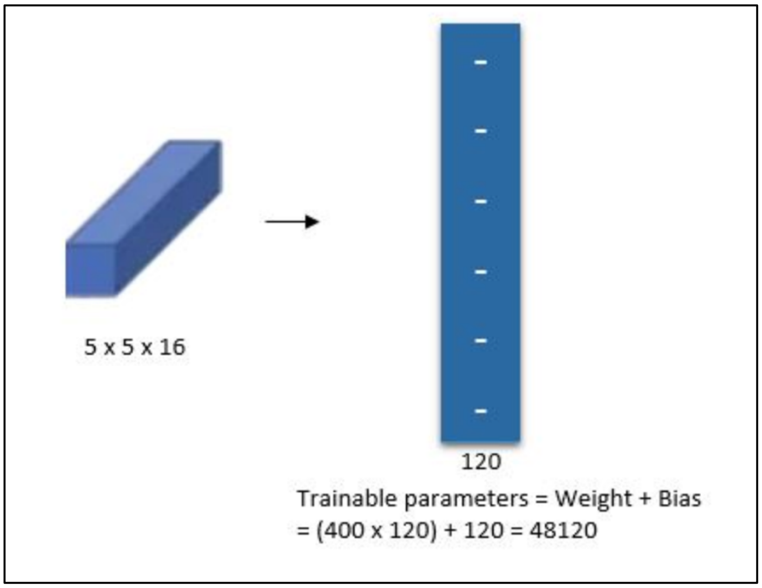

Figure 22: C5: Fully connected layer [18]

Sixth Layer

Sixth layer is again, a fully connected layer having 84 units as shown in Figure 23. 120 neurons are fully connected to 84 in this sixth layer.

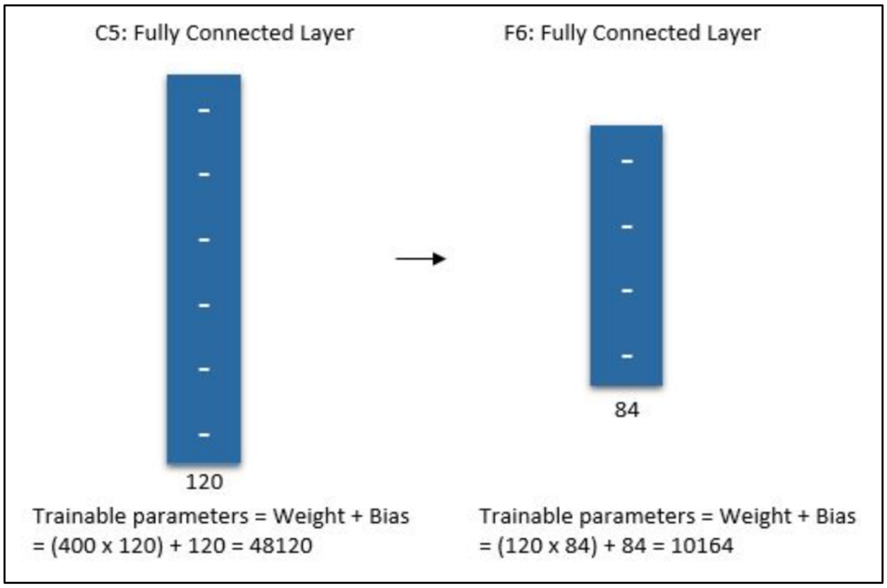

Figure 23: F6: Fully connected layer [18]

Output Layer

This is the final layer which is a fully connected soft-max layer with the final classification as shown in Figure 23.

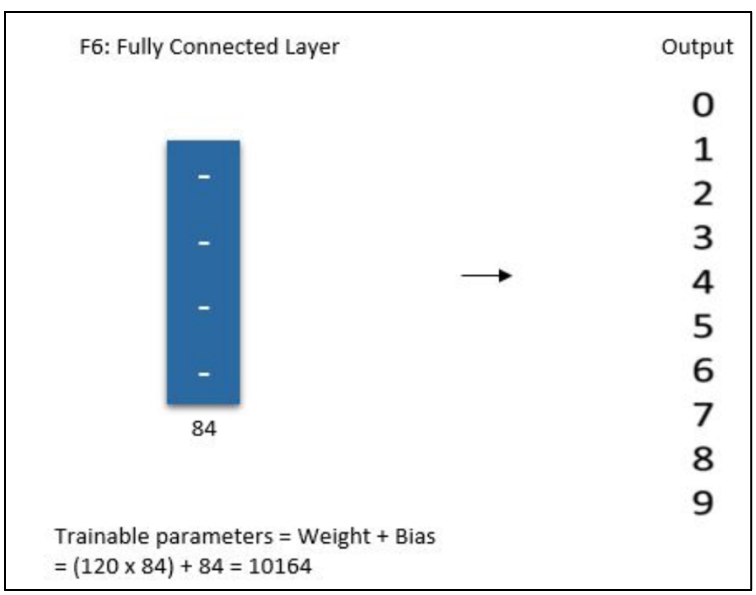

Figure 24: Output layer [18]

The layers with their configuration have been summarized in the Figure 25.

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Figure 25: LeNet 5 Architecture configuration summary [18]

2.3.3.2 Project configuration

As discussed in the previous section, LeNet-5 CNN architecture has been implemented with the following configuration:

1. CONV layer with 20 convolution filters, each filter being 5X5
2. ReLu activation function
3. 2X2 max pooling in both X and Y direction with a stride of 2
4. CONV layer with 50 convolution filters, each filter being 5X5
5. ReLu activation function
6. 2X2 max pooling in both X and Y direction with a stride of 2
7. Fully connected layer with 500 nodes
8. ReLu activation function

9. SoftMax classifier with 2 nodes and SoftMax activation function

Parameters:

Epochs: Varying starting from 1 to a maximum value of 50

Initial learning rate: 1 x e^{-3}

Batch size: 32

Algorithm for training the model:

1. Import the python library packages
2. Load images from disk
3. Preprocess the images as per the specification
4. Instantiate Convolutional Neural Network
5. Initialize the parameters
6. Grab image paths and shuffle them
7. Loop over the input images
    a. Load the image, process it and store in data list
    b. Extract class label from image path and update the label list
8. Scale the raw pixel intensities to [0,1]
9. Partition data into training and testing dataset (0.75 is the train test split ratio)
10. Convert labels from integers to vectors
11. Construct image generator for data augmentation

12. Train the Network
13. Save model to the disk

For all the different configurations, model files are created to be used in future to detect positive *nifH* sequences.

3. Results

The experiment has been conducted in two phases. In phase 1, different scales with 25 epochs have been tried. In phase II, the scale with the best result is tried against different epoch values.

3.1 Phase I:

In this phase, number of epochs equal to 25 are fixed for each case and variations on the scale are done.

3.1.1 Scale 28 X 28:

Image Scale: 28X28

No of epochs: 25

Table 3: Loss, Accuracy, Val Loss Val Accuracy VS 25 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1758	0.9521	0.1284	0.9554
2	0.1429	0.9567	0.1056	0.9640
3	0.1273	0.9604	0.1010	0.9660
4	0.1190	0.9613	0.0905	0.9680
5	0.1122	0.9631	0.0927	0.9684
6	0.1087	0.9631	0.0852	0.9678
7	0.1028	0.9651	0.0820	0.9701
8	0.1008	0.9655	0.1016	0.9644
9	0.0978	0.9664	0.0877	0.9693
10	0.0946	0.9678	0.0746	0.9745
11	0.0932	0.9659	0.0711	0.9741
12	0.0901	0.9678	0.0758	0.9727
13	0.0903	0.9680	0.0742	0.9749
14	0.0876	0.9682	0.0652	0.9761
15	0.0859	0.9691	0.0619	0.9777
16	0.0876	0.9699	0.0675	0.9757
17	0.0832	0.9693	0.0634	0.9775
18	0.0823	0.9707	0.0635	0.9773
19	0.0803	0.9708	0.0639	0.9774
20	0.0830	0.9705	0.0623	0.9781

21	0.0770	0.9721	0.0692	0.9762
22	0.0780	0.9722	0.0600	0.9787
23	0.0756	0.9726	0.0612	0.9790
24	0.0780	0.9721	0.0583	0.9800
25	0.0734	0.9728	0.0705	0.9781

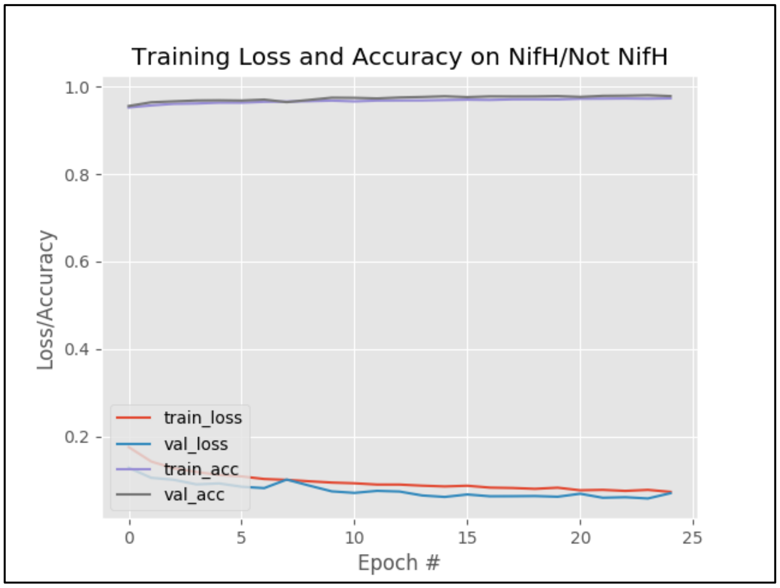

Figure 26: Loss/Accuracy for 28X28 VS Epochs (25)

3.1.2 Scale 7 X 28:

Image Scale: 7X28

No of epochs: 25

Table 4: Loss, Accuracy, Val Loss Val Accuracy VS 25 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1892	0.9525	0.1523	0.9541
2	0.1668	0.9530	0.1592	0.9598
3	0.1557	0.9540	0.1310	0.9644
4	0.1484	0.9562	0.1222	0.9658
5	0.1421	0.9578	0.2382	0.9575
6	0.1358	0.9584	0.1048	0.9676
7	0.1328	0.9593	0.1168	0.9685
8	0.1278	0.9610	0.1076	0.9603
9	0.1288	0.9603	0.1271	0.9676
10	0.1264	0.9603	0.1157	0.9669
11	0.1246	0.9610	0.1038	0.9670
12	0.1234	0.9618	0.0998	0.9683
13	0.1213	0.9618	0.0892	0.9694
14	0.1195	0.9622	0.0860	0.9703
15	0.1199	0.9622	0.0961	0.9675
16	0.1186	0.9621	0.0955	0.9675
17	0.1148	0.9636	0.0927	0.9691
18	0.1155	0.9623	0.1383	0.9661
19	0.1139	0.9635	0.1029	0.9668
20	0.1140	0.9629	0.0894	0.9672

21	0.1144	0.9630	0.0966	0.9661
22	0.1119	0.9633	0.1469	0.9652
23	0.1127	0.9635	0.0992	0.9677
24	0.1109	0.9638	0.1033	0.9673
25	0.1110	0.9635	0.0883	0.9669

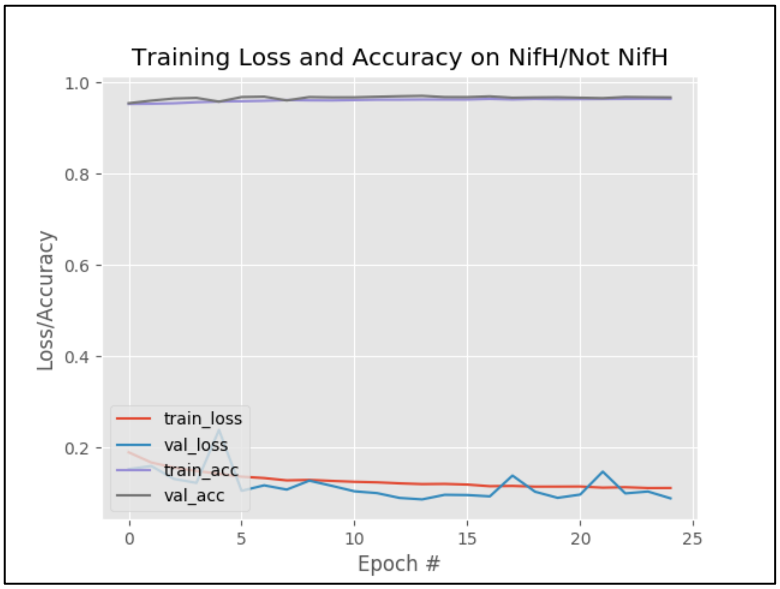

Figure 27: Loss/Accuracy 7X28 VS Epochs (25)

3.1.3 Scale 7 X 50:

Image Scale: 7X50

No of epochs: 25

Table 5: Loss, Accuracy, Val Loss Val Accuracy VS 25 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1794	0.9526	0.2770	0.9541
2	0.1471	0.9546	0.1308	0.9543
3	0.1361	0.9582	0.1114	0.9634
4	0.1282	0.9587	0.1060	0.9622
5	0.1234	0.9610	0.1052	0.9609
6	0.1178	0.9615	0.1519	0.9413
7	0.1159	0.9622	0.1826	0.9387
8	0.1117	0.9634	0.1154	0.9648
9	0.1088	0.9639	0.0838	0.9696
10	0.1128	0.9627	0.3508	0.8793
11	0.1071	0.9637	0.2175	0.9241
12	0.1069	0.9633	0.1250	0.9542
13	0.1020	0.9660	0.2885	0.8917
14	0.1029	0.9659	0.1795	0.9298
15	0.1003	0.9653	0.0809	0.9697
16	0.1020	0.9653	0.1106	0.9593
17	0.0954	0.9672	0.0864	0.9673
18	0.0986	0.9662	0.0773	0.9713
19	0.0980	0.9665	0.1434	0.9426
20	0.0952	0.9680	0.1044	0.9619

21	0.0924	0.9675	0.1757	0.9266
22	0.0940	0.9678	0.0853	0.9684
23	0.0926	0.9677	0.1711	0.9277
24	0.0907	0.9678	0.1525	0.9338
25	0.0903	0.9689	0.0611	0.9788

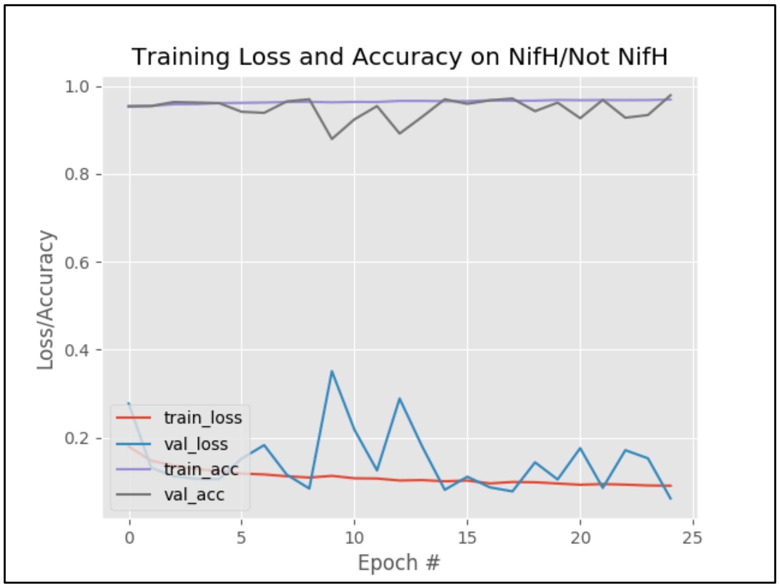

Figure 28: Loss/Accuracy 7X50 VS Epochs (25)

3.1.4 Scale 7 X 100:

Image Scale: 7X100

No of epochs: 25

Table 6: Loss, Accuracy, Val Loss Val Accuracy VS 25 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1706	0.9518	0.1091	0.9541
2	0.1232	0.9584	0.0778	0.9736
3	0.1080	0.9630	0.0876	0.9807
4	0.0972	0.9654	0.3893	0.8212
5	0.0968	0.9655	0.0591	0.9774
6	0.0909	0.9673	0.1132	0.9491
7	0.0862	0.9679	0.1984	0.9198
8	0.0846	0.9690	0.1272	0.9478
9	0.0815	0.9699	0.1585	0.9404
10	0.0812	0.9705	0.0954	0.9592
11	0.0790	0.9703	0.0689	0.9730
12	0.0749	0.9726	0.0472	0.9803
13	0.0736	0.9727	0.1487	0.9412
14	0.0717	0.9725	0.3884	0.8764
15	0.0699	0.9730	0.0526	0.9790
16	0.0696	0.9737	0.0725	0.9718
17	0.0692	0.9741	0.0697	0.9759
18	0.0685	0.9733	0.1336	0.9487
19	0.0666	0.9746	0.0967	0.9646
20	0.0644	0.9753	0.0539	0.9781

21	0.0653	0.9755	0.0358	0.9868
22	0.0634	0.9754	0.0718	0.9725
23	0.0653	0.9751	0.0336	0.9866
24	0.0616	0.9766	0.1001	0.9630
25	0.0622	0.9762	0.1545	0.9474

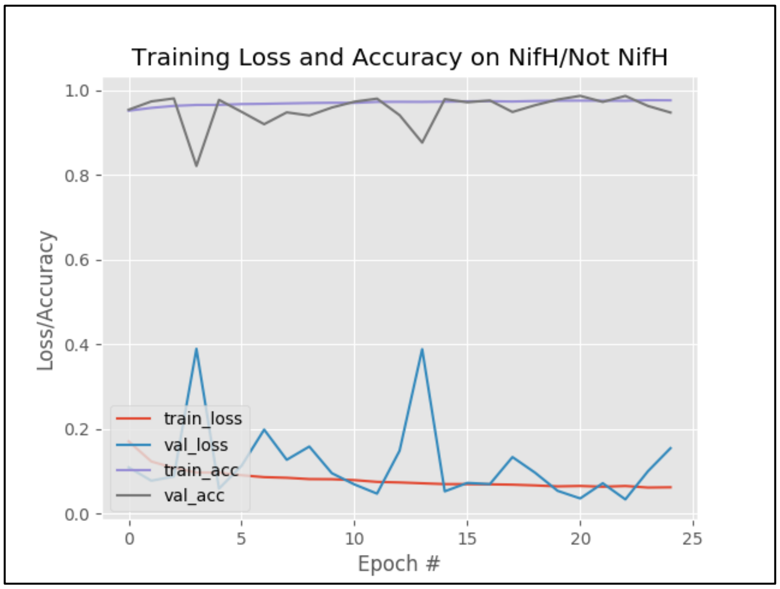

Figure 29: Loss/Accuracy 7X100 VS Epochs (25)

3.1.5 Scale 7 X 500:

Image Scale: 7X500

No of epochs: 25

Table 7: Loss, Accuracy, Val Loss Val Accuracy VS 25 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1323	0.9545	0.2695	0.9552
2	0.0932	0.9656	0.1523	0.9665
3	0.0787	0.9718	0.1208	0.9757
4	0.0715	0.9751	0.0812	0.9556
5	0.0675	0.9767	0.0946	0.9559
6	0.0606	0.9786	0.0889	0.9682
7	0.0568	0.9804	0.2325	0.9750
8	0.0562	0.9806	0.1063	0.9772
9	0.0515	0.9819	0.0924	0.9596
10	0.0504	0.9822	0.1262	0.9563
11	0.0481	0.9829	0.0976	0.9733
12	0.0492	0.9827	0.0933	0.9573
13	0.0464	0.9836	0.0816	0.9841
14	0.0446	0.9843	0.0689	0.9758
15	0.0420	0.9854	0.0591	0.9721
16	0.0440	0.9842	0.1206	0.9515
17	0.0404	0.9866	0.3788	0.8410
18	0.0401	0.9863	0.0378	0.9886
19	0.0374	0.9869	0.0423	0.9856
20	0.0364	0.9877	0.0499	0.9877

21	0.0370	0.9877	0.0847	0.9763
22	0.0351	0.9878	0.0372	0.9892
23	0.0368	0.9877	0.0814	0.9663
24	0.0345	0.9885	0.1023	0.9478
25	0.0347	0.9881	0.0464	0.9865

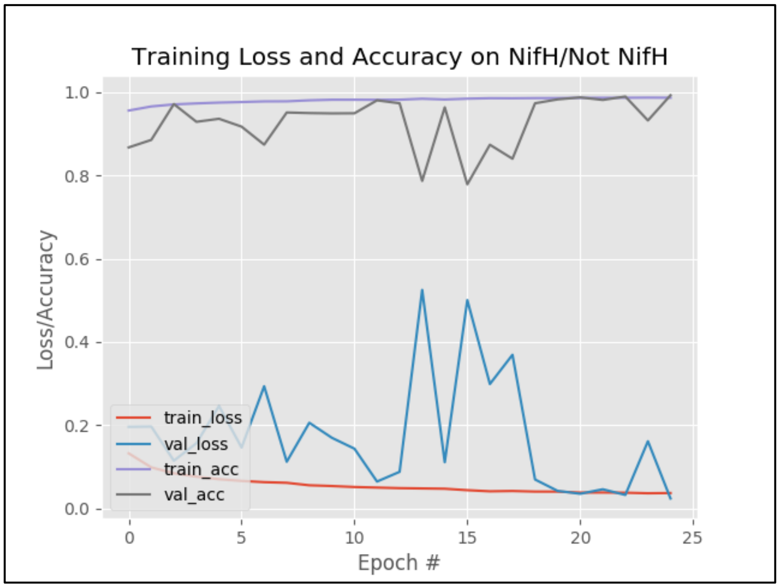

Figure 30: Loss/Accuracy 7X500 VS Epochs (25)

3.1.6 Scale 7 X 1000:

Image Scale: 7X1000

No of epochs: 25

Table 8: Loss, Accuracy, Val Loss Val Accuracy VS 25 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.7608	0.9516	0.7361	0.9541
2	0.7619	0.9525	0.7361	0.9541
3	0.7619	0.9525	0.7361	0.9541
4	0.7579	0.9527	0.7361	0.9541
5	0.7604	0.9526	0.7361	0.9541
6	0.7614	0.9525	0.7361	0.9541
7	0.7624	0.9524	0.7361	0.9541
8	0.7594	0.9526	0.7361	0.9541
9	0.7588	0.9527	0.7361	0.9541
10	0.7584	0.9527	0.7361	0.9541
11	0.7673	0.9521	0.7361	0.9541
12	0.7584	0.9527	0.7361	0.9541
13	0.7604	0.9526	0.7361	0.9541
14	0.7569	0.9528	0.7361	0.9541
15	0.7664	0.9522	0.7361	0.9541
16	0.7614	0.9525	0.7361	0.9541
17	0.7599	0.9526	0.7361	0.9541
18	0.7604	0.9526	0.7361	0.9541
19	0.7614	0.9525	0.7361	0.9541
20	0.7559	0.9528	0.7361	0.9541

21	0.7654	0.9523	0.7361	0.9541
22	0.7644	0.9523	0.7361	0.9541
23	0.7589	0.9527	0.7361	0.9541
24	0.7579	0.9527	0.7361	0.9541
25	0.7629	0.9524	0.7361	0.9541

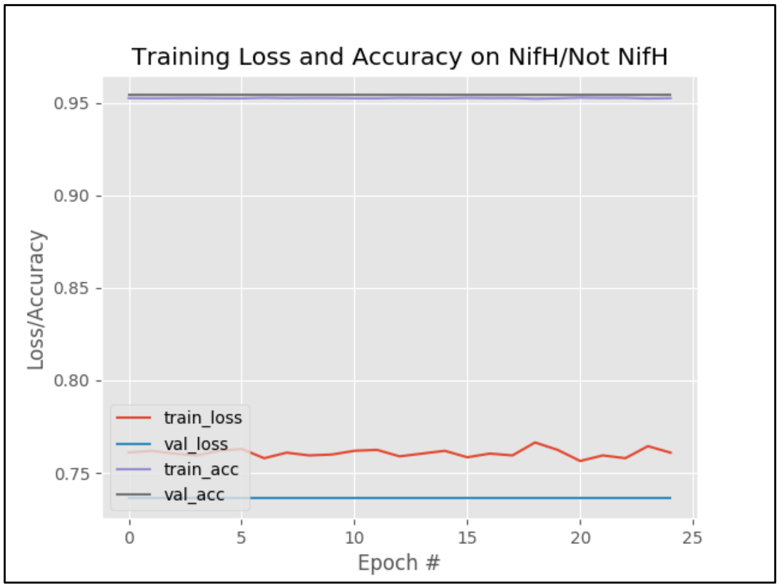

Figure 31: Loss/Accuracy 7X1000 VS Epochs (25)

Result Analysis of phase I:

Overall accuracy kept on increasing starting from 28X28. At 7X500, maximum accuracy of around 98.5 was achieved, further compelling for 7X1000 experiment. The accuracy was 95.26% in case of 7X1000 which is not good. This fact compelled to experiment with scaling as 7X500 and varying the epochs.

3.2 Phase II:

As mentioned above, since scaling of 7X500 yielded optimal results, this scaling was fixed and number of epochs were varied for producing test results in phase II. Also, in each case, confusion matrix is mentioned as well for getting a better idea of classification.

3.2.1 Scale 7X500, Epochs 1:

Image Scale: 7X500

No of epochs: 1

Table 9: Loss, Accuracy, Val Loss Val Accuracy VS 1 Epoch

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.7605	0.9526	0.7361	0.9541

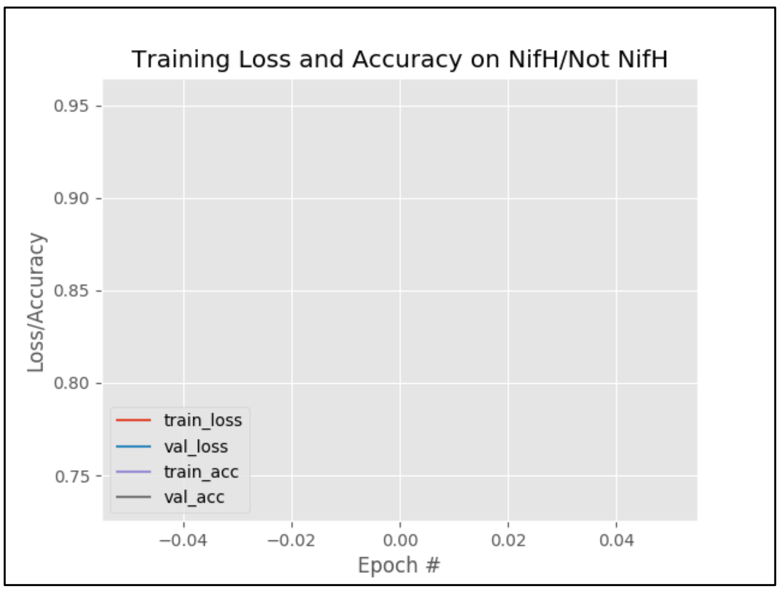

Figure 32: Loss/Accuracy 7X500 VS Epochs (1)

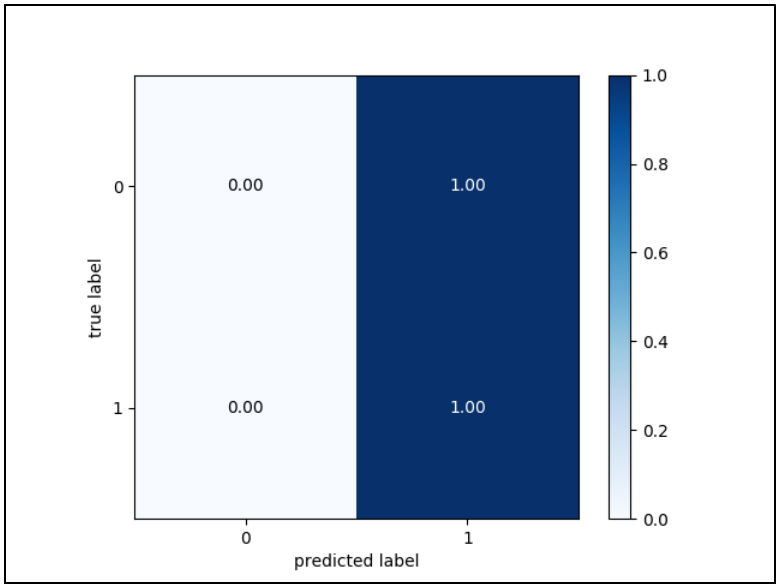

Figure 33: Confusion Matrix for 7X500(1 Epoch)

3.2.2 Scale 7X500, Epochs 5:

Image Scale: 7X500

No of epochs: 5

Table 10: Loss, Accuracy, Val Loss Val Accuracy VS 5 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1331	0.9553	0.1006	0.9550
2	0.0967	0.9646	0.0825	0.9545
3	0.0905	0.9678	0.1130	0.9541
4	0.0832	0.9702	0.1550	0.9743
5	0.0783	0.9772	0.1015	0.9608

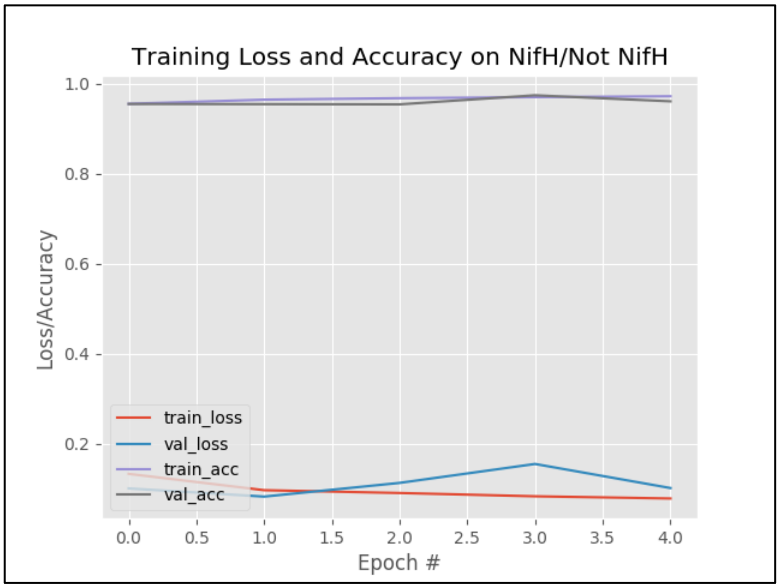

Figure 34: Loss/Accuracy 7x500 VS Epochs (5)

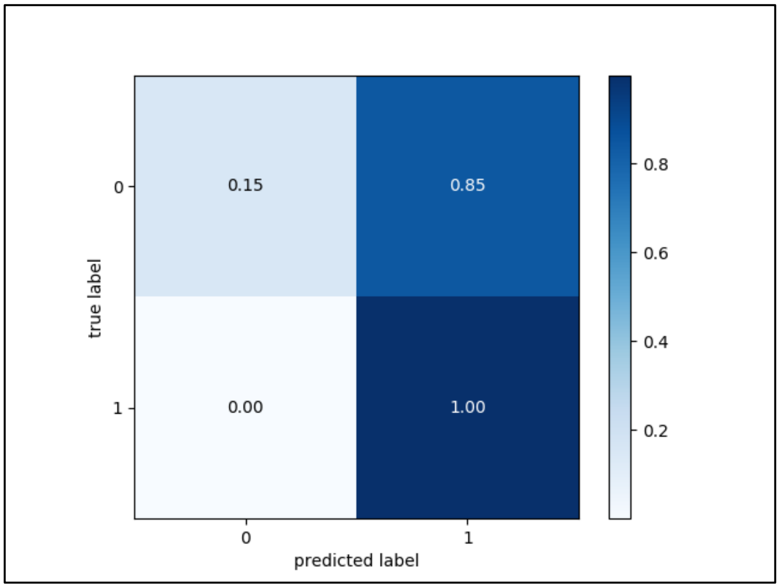

Figure 35: Confusion Matrix for 7X500(5 Epochs)

3.2.3 Scale 7X500, Epochs 10:

Image Scale: 7X500

No of epochs: 10

Table 11: Loss, Accuracy, Val Loss Val Accuracy VS 10 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1250	0.9555	0.4796	0.7879
2	0.0941	0.9664	0.1058	0.9839
3	0.0791	0.9723	0.1569	0.9610
4	0.0734	0.9739	0.0740	0.9733
5	0.0655	0.9766	0.1117	0.9738
6	0.0636	0.9779	0.0777	0.9877
7	0.0571	0.9801	0.0844	0.9818
8	0.0542	0.9816	0.0854	0.9838
9	0.0515	0.9814	0.1014	0.9882
10	0.0491	0.9833	0.0621	0.9873

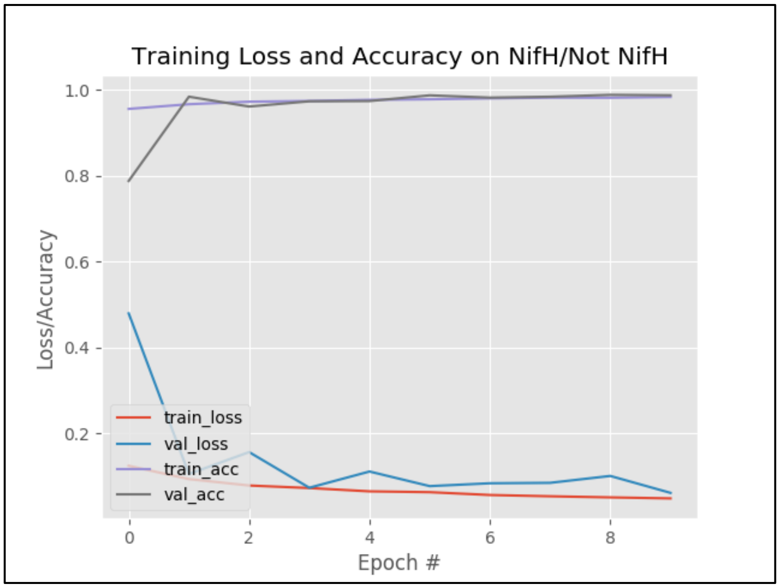

Figure 36: Loss/Accuracy 7X500 VS Epochs (10)

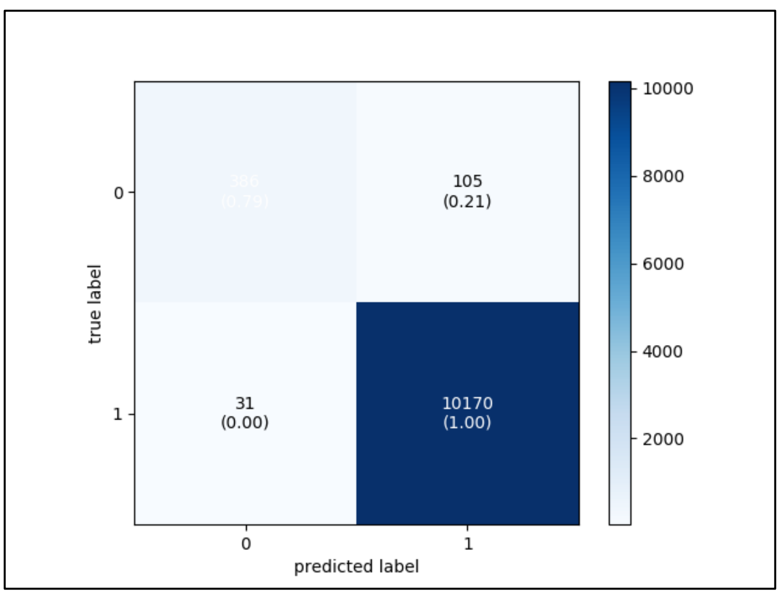

Figure 37: Confusion Matrix for 7X500(10 Epochs)

3.2.4 Scale 7X500, Epochs 15:

Image Scale: 7X500

No of epochs: 15

Table 12: Loss, Accuracy, Val Loss Val Accuracy VS 15 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1428	0.9537	0.1545	0.9469
2	0.1003	0.9630	0.6951	0.6714
3	0.0880	0.9686	0.5846	0.8218
4	0.0793	0.9719	0.1432	0.9579
5	0.0715	0.9747	0.1435	0.9236
6	0.0686	0.9756	0.2038	0.9497
7	0.0627	0.9774	0.4235	0.8680
8	0.0610	0.9788	0.1509	0.9772
9	0.0602	0.9800	0.5899	0.7135
10	0.0563	0.9807	0.0909	0.9786
11	0.0539	0.9808	0.3160	0.8685
12	0.0511	0.9823	0.1820	0.9706
13	0.0491	0.9825	0.0764	0.9788
14	0.0487	0.9822	0.0851	0.9805
15	0.0451	0.9829	0.1614	0.9731

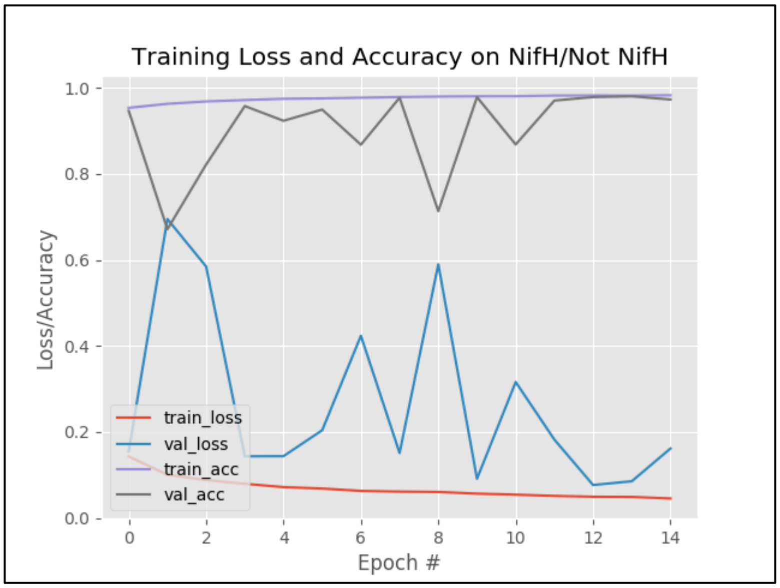

Figure 38: Loss/Accuracy 7X500 VS Epochs (15)

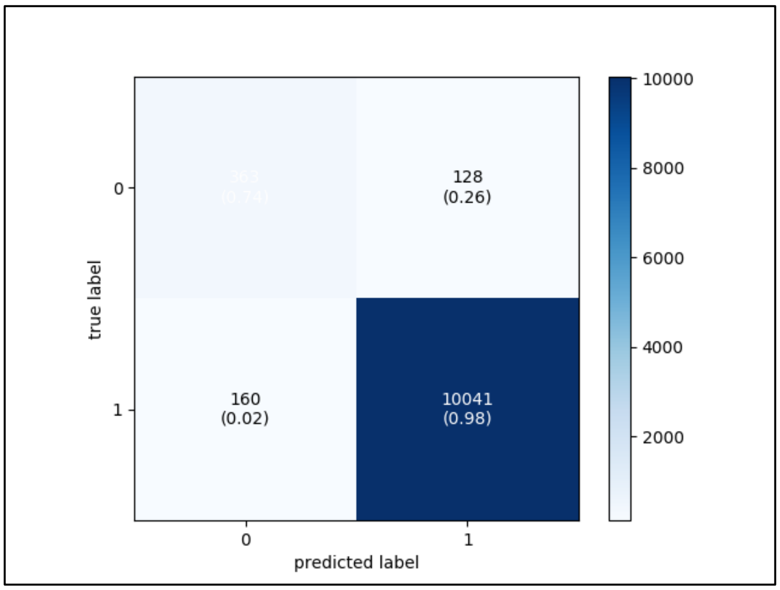

Figure 39: Confusion Matrix for 7X500(15 Epochs)

3.2.5 Scale 7X500, Epochs 20:

Image Scale: 7X500

No of epochs: 20

Table 13: Loss, Accuracy, Val Loss Val Accuracy VS 20 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1347	0.9549	0.2502	0.9541
2	0.0990	0.9632	0.0936	0.9617
3	0.0898	0.9679	0.0823	0.9598
4	0.0836	0.9702	0.1103	0.9722
5	0.0797	0.9720	0.3404	0.8395
6	0.0739	0.9732	1.1383	0.2282
7	0.0721	0.9757	0.3867	0.8511
8	0.0674	0.9769	0.1529	0.9428
9	0.0618	0.9775	1.6989	0.1661
10	0.0608	0.9785	0.1515	0.9237
11	0.0592	0.9794	0.2814	0.8691
12	0.0576	0.9801	0.2542	0.8789
13	0.0551	0.9810	0.1320	0.9627
14	0.0498	0.9819	0.1898	0.9365
15	0.0511	0.9829	0.4443	0.8142
16	0.0490	0.9831	0.2525	0.8995
17	0.0470	0.9836	0.0491	0.9850
18	0.0461	0.9831	0.0733	0.9674
19	0.0451	0.9844	0.2055	0.9136

20	0.0437	0.9846	0.4300	0.8328

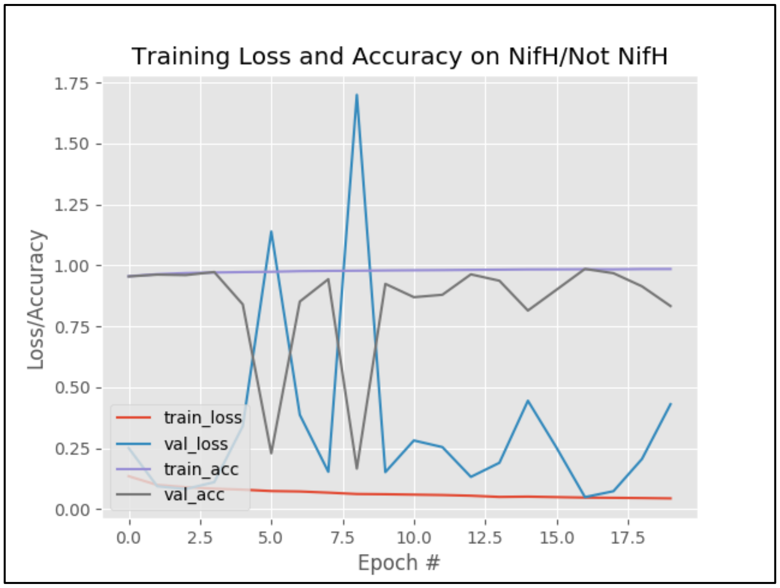

Figure 40: Loss/Accuracy 7X500 VS Epochs (20)

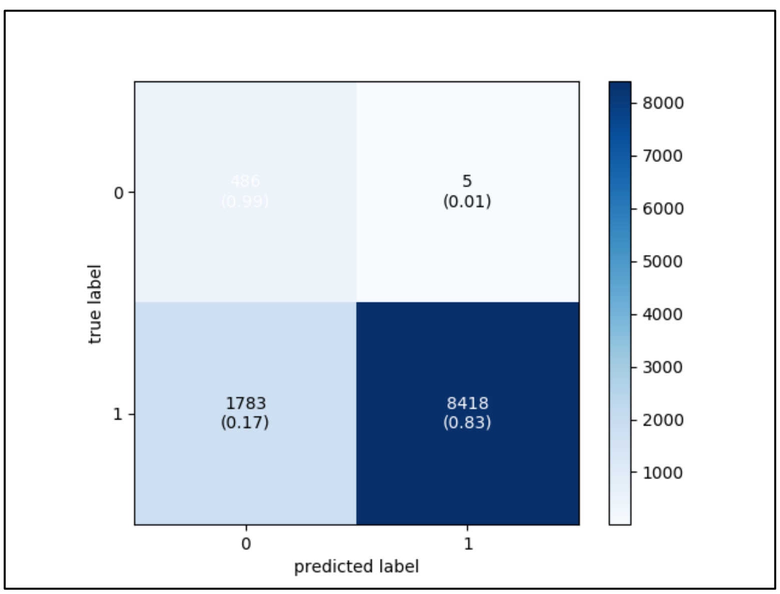

Figure 41: Confusion Matrix for 7X500(20 Epochs)

3.2.6 Scale 7X500, Epochs 25:

Image Scale: 7X500

No of epochs: 25

Table 14: Loss, Accuracy, Val Loss Val Accuracy VS 25 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1323	0.9545	0.2695	0.9552
2	0.0932	0.9656	0.1523	0.9665
3	0.0787	0.9718	0.1208	0.9757
4	0.0715	0.9751	0.0812	0.9556
5	0.0675	0.9767	0.0946	0.9559
6	0.0606	0.9786	0.0889	0.9682
7	0.0568	0.9804	0.2325	0.9750
8	0.0562	0.9806	0.1063	0.9772
9	0.0515	0.9819	0.0924	0.9596
10	0.0504	0.9822	0.1262	0.9563
11	0.0481	0.9829	0.0976	0.9733
12	0.0492	0.9827	0.0933	0.9573
13	0.0464	0.9836	0.0816	0.9841
14	0.0446	0.9843	0.0689	0.9758
15	0.0420	0.9854	0.0591	0.9721
16	0.0440	0.9842	0.1206	0.9515
17	0.0404	0.9866	0.3788	0.8410
18	0.0401	0.9863	0.0378	0.9886
19	0.0374	0.9869	0.0423	0.9856

20	0.0364	0.9877	0.0499	0.9877
21	0.0370	0.9877	0.0847	0.9763
22	0.0351	0.9878	0.0372	0.9892
23	0.0368	0.9877	0.0814	0.9663
24	0.0345	0.9885	0.1023	0.9478
25	0.0347	0.9881	0.0464	0.9865

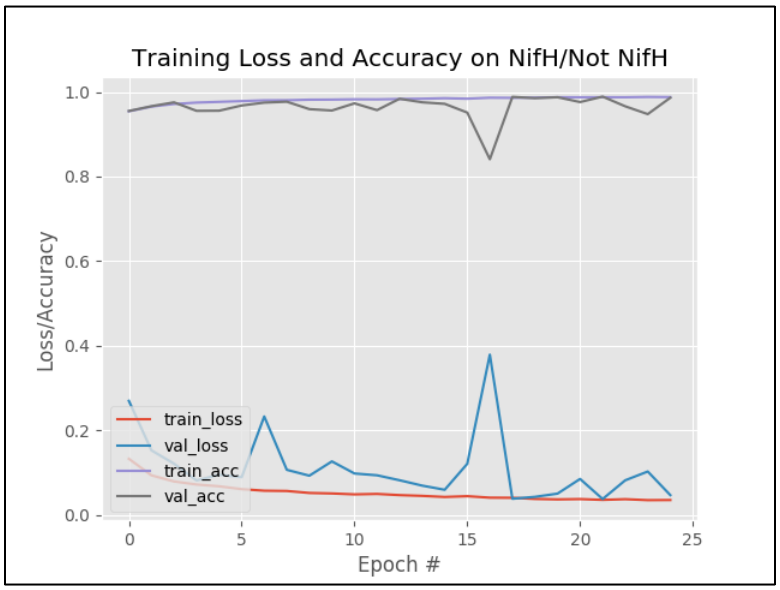

Figure 42: Loss/Accuracy 7X500 VS Epochs (25)

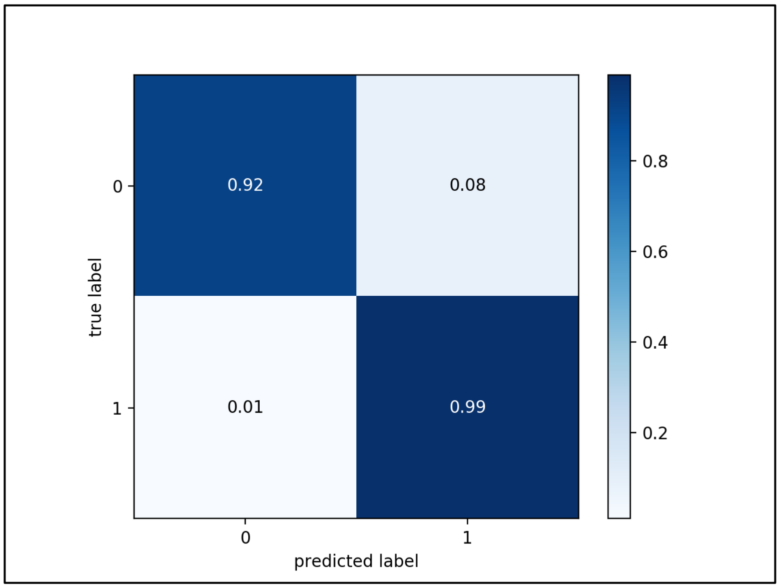

Figure 43: Confusion Matrix for 7X500(25 Epochs)

3.2.7 Scale 7X500, Epochs 30:

Image Scale: 7X500

Epochs: 30

Table 15: Loss, Accuracy, Val Loss Val Accuracy VS 30 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1320	0.9538	0.4955	0.8562
2	0.1028	0.9626	0.1690	0.9534
3	0.0828	0.9706	0.1768	0.9511
4	0.0735	0.9740	0.2346	0.9808
5	0.0667	0.9760	0.1002	0.9546
6	0.0643	0.9780	0.3632	0.9559
7	0.0610	0.9782	0.1335	0.9743
8	0.0577	0.9791	0.1592	0.9702
9	0.0519	0.9819	0.0801	0.9722
10	0.0521	0.9822	0.1320	0.9640
11	0.0521	0.9816	0.1197	0.9734
12	0.0490	0.9833	0.0614	0.9713
13	0.0475	0.9844	0.0866	0.9695
14	0.0461	0.9840	0.0678	0.9748
15	0.0444	0.9851	0.0617	0.9752
16	0.0431	0.9846	0.0518	0.9848
17	0.0414	0.9862	0.0436	0.9888
18	0.0410	0.9852	0.0741	0.9847
19	0.0386	0.9866	0.1051	0.9752

20	0.0386	0.9868	0.0531	0.9805
21	0.0382	0.9869	0.0521	0.9757
22	0.0370	0.9868	0.0493	0.9847
23	0.0371	0.9875	0.0787	0.9873
24	0.0361	0.9879	0.0788	0.9706
25	0.0348	0.9881	0.0806	0.9644
26	0.0328	0.9890	0.0644	0.9773
27	0.0316	0.9892	0.0529	0.9831
28	0.0323	0.9888	0.0310	0.9884
29	0.0299	0.9898	0.0695	0.9834
30	0.0310	0.9892	0.1046	0.9832

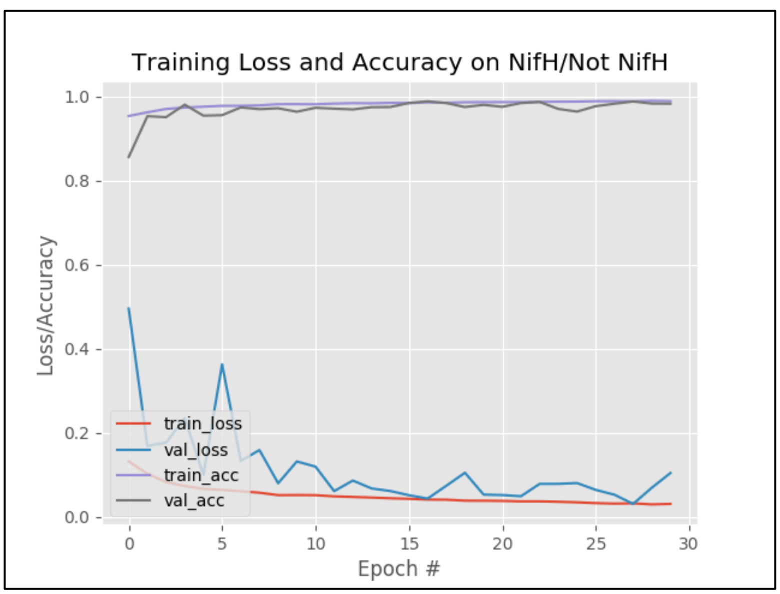

Figure 44: Loss/Accuracy 7X500 VS Epochs (30)

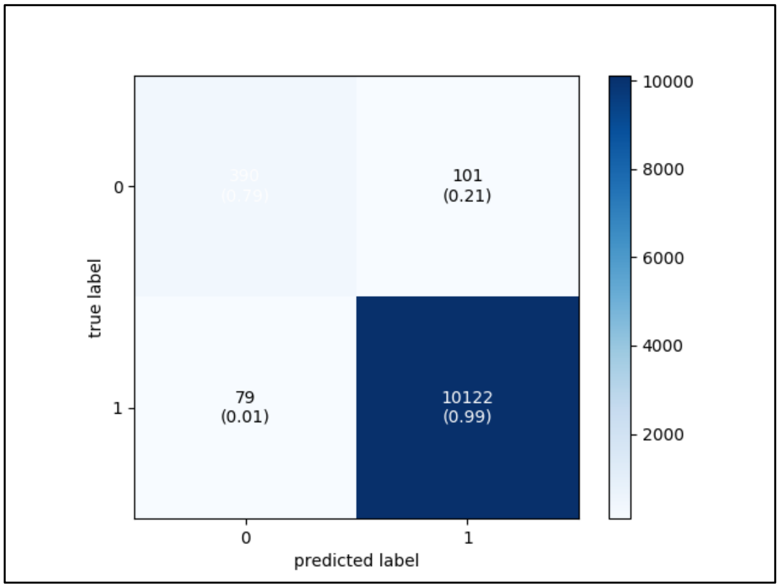

Figure 45: Confusion Matrix for 7X500(30 Epochs)

3.2.8 Scale 7X500, Epochs 35:

Image Scale: 7X500

Epochs: 35

Table 16: Loss, Accuracy, Val Loss Val Accuracy VS 35 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.7600	0.9525	0.7361	0.9541
2	0.7604	0.9526	0.7361	0.9541
3	0.7604	0.9526	0.7361	0.9541
4	0.7614	0.9525	0.7361	0.9541
5	0.7599	0.9526	0.7361	0.9541
6	0.7559	0.9528	0.7361	0.9541
7	0.7658	0.9522	0.7361	0.9541
8	0.7584	0.9527	0.7361	0.9541
9	0.7624	0.9524	0.7361	0.9541
10	0.7564	0.9528	0.7361	0.9541
11	0.7654	0.9523	0.7361	0.9541
12	0.7644	0.9523	0.7361	0.9541
13	0.7564	0.9528	0.7361	0.9541
14	0.7643	0.9523	0.7361	0.9541
15	0.7604	0.9526	0.7361	0.9541
16	0.7599	0.9526	0.7361	0.9541
17	0.7614	0.9525	0.7361	0.9541
18	0.7594	0.9526	0.7361	0.9541
19	0.7614	0.9525	0.7361	0.9541

20	0.7639	0.9523	0.7361	0.9541
21	0.7599	0.9526	0.7361	0.9541
22	0.7569	0.9528	0.7361	0.9541
23	0.7534	0.9530	0.7361	0.9541
24	0.7713	0.9519	0.7361	0.9541
25	0.7614	0.9525	0.7361	0.9541
26	0.7579	0.9527	0.7361	0.9541
27	0.7649	0.9523	0.7361	0.9541
28	0.7579	0.9527	0.7361	0.9541
29	0.7579	0.9527	0.7361	0.9541
30	0.7604	0.9526	0.7361	0.9541
31	0.7624	0.9524	0.7361	0.9541
32	0.7619	0.9525	0.7361	0.9541
33	0.7643	0.9523	0.7361	0.9541
34	0.7361	0.9541	0.7361	0.9541
35	0.7619	0.9525	0.7361	0.9541

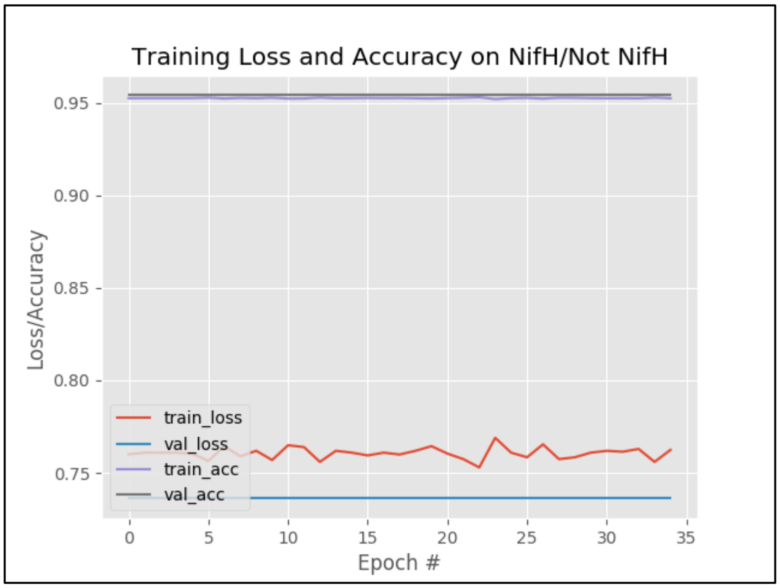

Figure 46: Loss/Accuracy 7X500 VS Epochs (35)

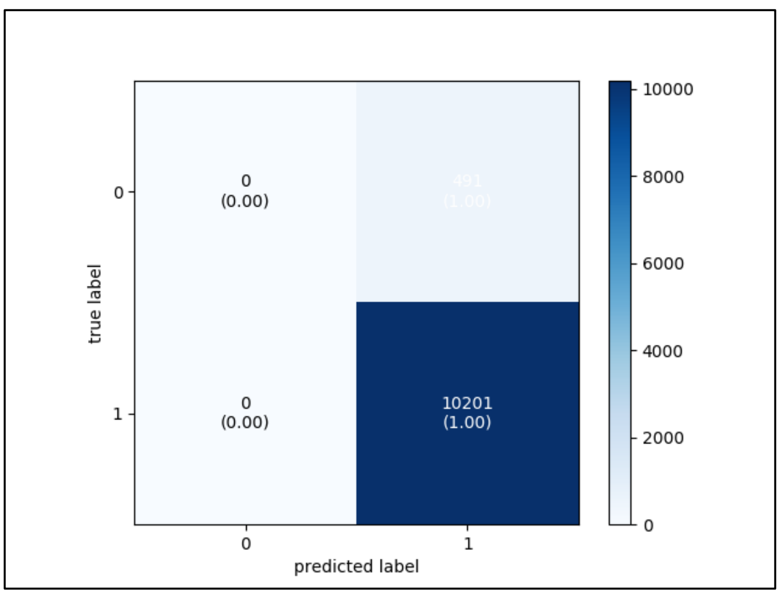

Figure 47: Confusion Matrix for 7X500(35 Epochs)

3.2.9 Scale 7X500, Epochs 40:

Image Scale: 7X500

Epochs: 40

Table 17: Loss, Accuracy, Val Loss Val Accuracy VS 40 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1252	0.9580	0.5044	0.7902
2	0.0948	0.9659	0.1394	0.9451
3	0.0885	0.9687	0.1585	0.9077
4	0.0771	0.9729	0.1120	0.9790
5	0.0720	0.9746	0.1409	0.9364
6	0.0670	0.9765	0.1399	0.9718
7	0.0644	0.9772	0.3628	0.8322
8	0.0611	0.9788	0.1495	0.9810
9	0.0561	0.9811	0.1585	0.9424
10	0.0559	0.9794	0.4699	0.8435
11	0.0536	0.9809	0.1449	0.9271
12	0.0510	0.9813	0.1631	0.9216
13	0.0496	0.9828	0.2372	0.9164
14	0.0481	0.9827	0.0960	0.9755
15	0.0452	0.9839	0.1129	0.9555
16	0.0459	0.9842	0.0618	0.9838
17	0.0428	0.9845	0.0763	0.9819
18	0.0443	0.9848	0.0447	0.9839
19	0.0411	0.9857	0.0431	0.9873

20	0.0412	0.9854	0.1336	0.9411
21	0.0399	0.9858	0.0533	0.9788
22	0.0359	0.9876	0.0584	0.9831
23	0.0382	0.9863	0.1023	0.9594
24	0.0386	0.9862	0.1400	0.9689
25	0.0365	0.9874	0.0581	0.9819
26	0.0351	0.9876	0.1973	0.9224
27	0.0365	0.9872	0.0946	0.9660
28	0.0333	0.9887	0.0910	0.9623
29	0.0337	0.9881	0.0906	0.9615
30	0.0341	0.9880	0.1118	0.9564
31	0.0321	0.9883	0.0636	0.9806
32	0.0328	0.9888	0.0558	0.9779
33	0.0301	0.9895	0.0577	0.9865
34	0.0307	0.9893	0.0788	0.9693
35	0.0309	0.9891	0.1777	0.9487
36	0.0297	0.9894	0.2219	0.9307
37	0.0300	0.9891	0.1978	0.9358
38	0.0290	0.9903	0.0968	0.9663
39	0.0285	0.9900	0.1399	0.9822
40	0.0290	0.9902	0.0938	0.9783

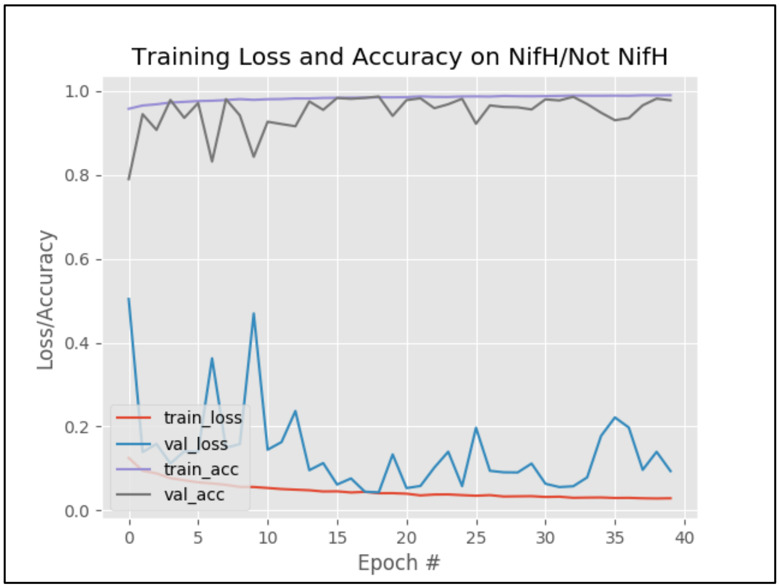

Figure 48: Loss/Accuracy 7X500 VS Epochs (40)

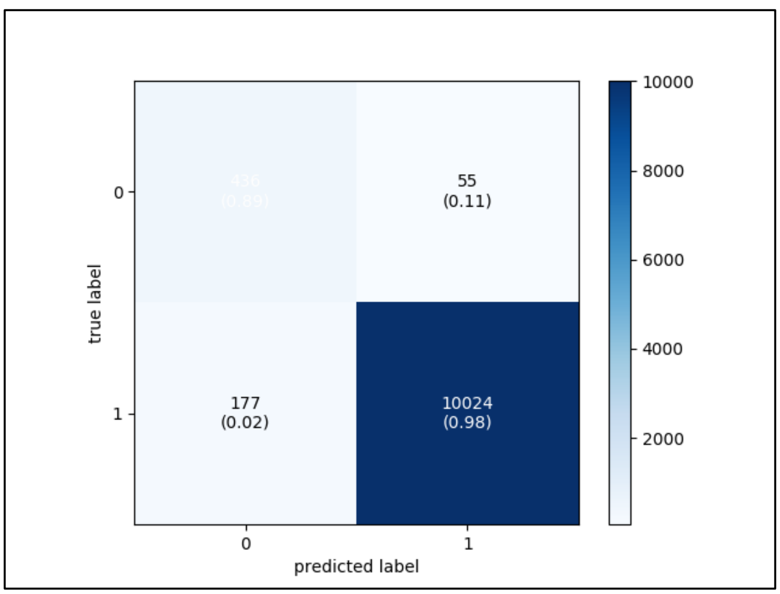

Figure 49: Confusion Matrix for 7X500(40 Epochs)

3.2.10 Scale 7X500, Epochs 45

Image Scale: 7X500

Epochs: 45

Table 18: Loss, Accuracy, Val Loss Val Accuracy VS 45 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1337	0.9556	0.1369	0.9378
2	0.1043	0.9635	0.3923	0.8642
3	0.0956	0.9679	0.2230	0.9328
4	0.0834	0.9710	0.1410	0.9339
5	0.0788	0.9722	0.3316	0.9752
6	0.0714	0.9736	0.1220	0.9750
7	0.0667	0.9764	0.0984	0.9766
8	0.0636	0.9775	0.1757	0.9805
9	0.0580	0.9790	0.2883	0.9413
10	0.0573	0.9804	0.1742	0.9751
11	0.0542	0.9812	0.1466	0.9559
12	0.0514	0.9814	0.3619	0.9710
13	0.0505	0.9813	0.3621	0.8596
14	0.0495	0.9826	0.0967	0.9831
15	0.0472	0.9832	0.3266	0.9713
16	0.0467	0.9836	0.0852	0.9798
17	0.0428	0.9859	0.2102	0.9762
18	0.0441	0.9849	0.3634	0.9504
19	0.0423	0.9851	0.1207	0.9650

20	0.0416	0.9857	0.2310	0.9582
21	0.0389	0.9857	0.2174	0.9563
22	0.0420	0.9854	0.1689	0.9706
23	0.0377	0.9872	0.3265	0.8374
24	0.0385	0.9866	0.1474	0.9386
25	0.0370	0.9872	0.6355	0.7239
26	0.0358	0.9874	0.2612	0.8811
27	0.0371	0.9880	0.7773	0.6739
28	0.0348	0.9881	0.2785	0.9391
29	0.0366	0.9866	0.2759	0.8591
30	0.0328	0.9885	0.1372	0.9601
31	0.0340	0.9884	0.1246	0.9551
32	0.0326	0.9894	0.0829	0.9784
33	0.0324	0.9887	0.1156	0.9552
34	0.0323	0.9891	0.0757	0.9819
35	0.0342	0.9890	0.2554	0.9350
36	0.0299	0.9897	0.2005	0.9372
37	0.0324	0.9893	0.2042	0.9484
38	0.0299	0.9896	0.4855	0.7953
39	0.0319	0.9894	0.1329	0.9429
40	0.0307	0.9895	0.1260	0.9522
41	0.0279	0.9904	0.2741	0.9054
42	0.0291	0.9898	0.2053	0.9526
43	0.0271	0.9909	0.0798	0.9747
44	0.0262	0.9906	0.2262	0.9274
45	0.0300	0.9903	0.1301	0.9575

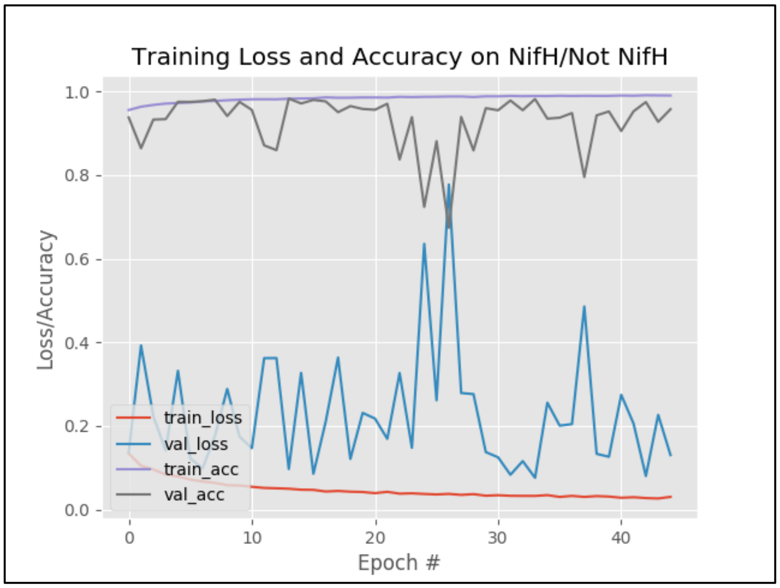

Figure 50: Loss/Accuracy VS Epochs (45)

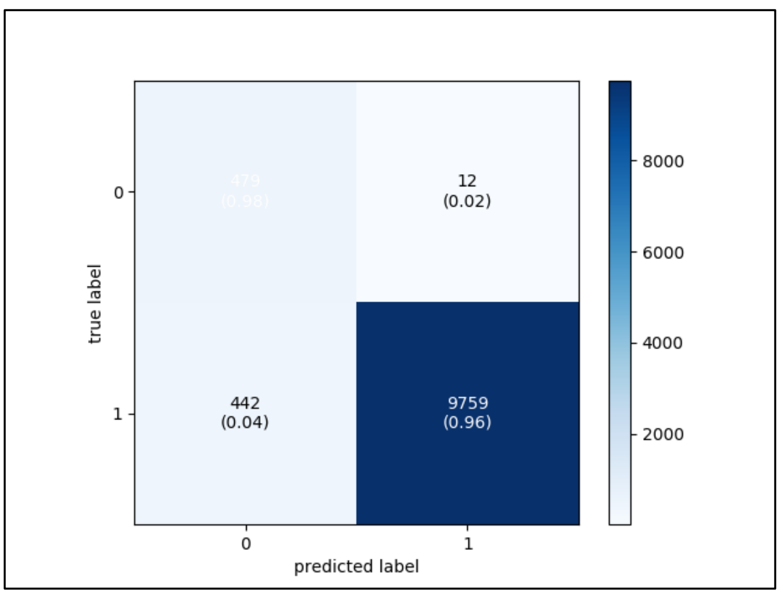

Figure 51: Confusion Matrix for 7X500(45 Epochs)

3.2.11 Scale 7X500, Epochs 50

Image Scale:	7X500
Epochs:	50

Table 19: Loss, Accuracy, Val Loss Val Accuracy VS 50 Epochs

Epoch	Loss	Accuracy	Val_loss	Val_accuracy
1	0.1479	0.9545	0.3502	0.9541
2	0.1045	0.9630	0.1501	0.9541
3	0.0870	0.9688	0.1194	0.9836
4	0.0802	0.9723	0.1200	0.9864
5	0.0741	0.9741	0.0774	0.9811
6	0.0682	0.9765	0.0641	0.9788
7	0.0655	0.9766	0.0541	0.9874
8	0.0625	0.9780	0.0561	0.9846
9	0.0593	0.9795	0.1241	0.9633
10	0.0579	0.9795	0.0474	0.9866
11	0.0544	0.9804	0.1646	0.9406
12	0.0510	0.9819	0.0624	0.9843
13	0.0497	0.9825	0.0890	0.9722
14	0.0470	0.9836	0.6084	0.8058
15	0.0467	0.9833	0.4919	0.8462
16	0.0460	0.9836	0.0406	0.9904
17	0.0427	0.9851	0.0352	0.9882
18	0.0430	0.9844	0.2190	0.9263
19	0.0429	0.9848	0.4219	0.8813
20	0.0424	0.9846	0.1131	0.9675
21	0.0393	0.9856	0.0830	0.9735

22	0.0387	0.9859	0.7629	0.8206
23	0.0373	0.9866	0.2629	0.9265
24	0.0377	0.9864	0.0542	0.9862
25	0.0350	0.9879	0.2006	0.9436
26	0.0371	0.9877	0.3334	0.9287
27	0.0354	0.9877	0.6911	0.8332
28	0.0335	0.9889	0.3536	0.8908
29	0.0329	0.9883	0.3467	0.9133
30	0.0319	0.9892	0.1715	0.9565
31	0.0350	0.9881	0.6587	0.8469
32	0.0322	0.9889	0.9930	0.7942
33	0.0302	0.9895	0.5804	0.8712
34	0.0309	0.9898	0.6971	0.8086
35	0.0301	0.9896	0.3296	0.8931
36	0.0294	0.9902	0.2373	0.9552
37	0.0310	0.9893	0.3355	0.9246
38	0.0301	0.9886	0.3763	0.8799
39	0.0298	0.9905	0.6072	0.8326
40	0.0268	0.9909	0.2927	0.9430
41	0.0293	0.9901	0.2086	0.9525
42	0.0289	0.9900	0.6052	0.8277
43	0.0286	0.9906	0.2764	0.9534
44	0.0276	0.9907	0.0415	0.8915
45	0.0299	0.9899	0.4993	0.9276
46	0.0276	0.9903	0.5157	0.8572
47	0.0275	0.9903	0.9538	0.8360
48	0.0262	0.9907	0.3084	0.9415
49	0.0285	0.9907	0.1525	0.9588
50	0.0274	0.9905	0.3544	0.8987

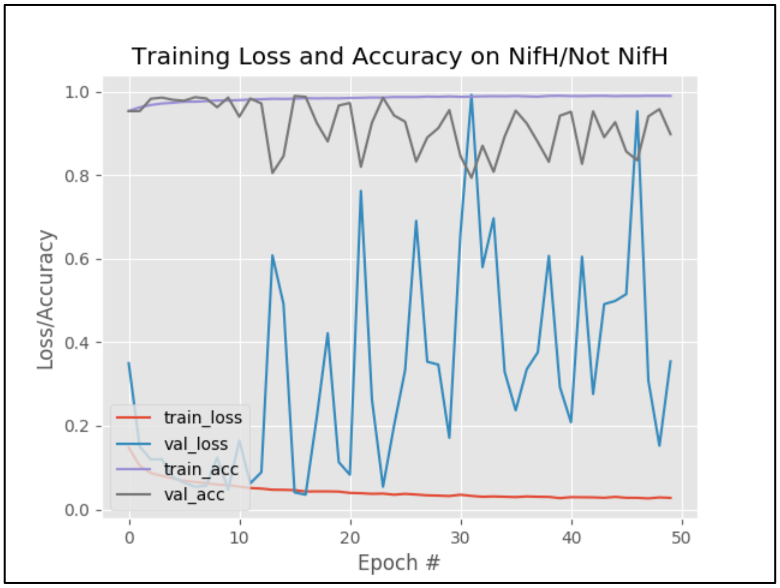

Figure 52: Loss/Accuracy 7X500 VS Epochs (50)

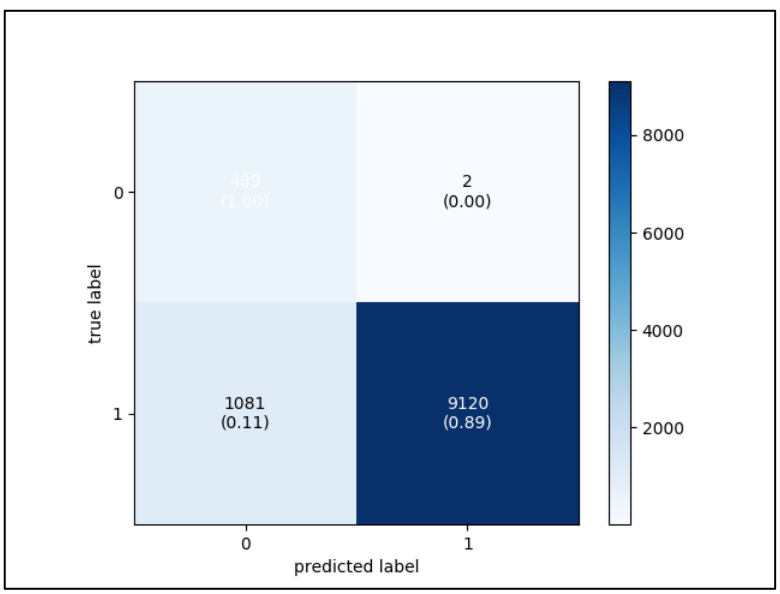

Figure 53: Confusion Matrix for 7X500(50 Epochs)

4. Result discussion

4.1 Performance Comparison

The table below summarize the results achieved in 7X500. As shown in the table below, the best accuracies were achieved in case of 7 and 9 with 30 and 40 epochs respectively.

Table 20: Result Summary

S. No	Epochs	Loss	Accuracy	Val_Loss	Val_Accuracy	TP	FP	TN	FN
1	1	0.7605	0.9526	0.7361	0.9541	10201	491	0	0
2	5	0.0783	0.9722	0.1015	0.9608	10197	415	76	4
3	10	0.0491	0.9833	0.0621	0.9873	10170	105	386	31
4	15	0.0451	0.9829	0.1614	0.9731	10041	128	363	160
5	20	0.0437	0.9846	0.4300	0.8328	8418	5	486	1783
6	25	0.0338	0.9886	0.1497	0.9697	9954	78	413	247
7	**30**	**0.0310**	**0.9892**	**0.1046**	**0.9832**	**10122**	**101**	**390**	**79**
8	35	0.7619	0.9525	0.7361	0.9541	10201	491	0	0
9	**40**	**0.0290**	**0.9902**	**0.0938**	**0.9783**	**10024**	**55**	**436**	**177**
10	45	0.0300	0.9903	0.1301	0.9575	9759	12	479	442
11	50	0.0274	.9905	0.3544	0.8987	9120	2	489	1081

For case 7,

Sensitivity

$$= \frac{number\ of\ true\ positives}{number\ of\ true\ positives + number\ of\ false\ negatives}$$

$$= \frac{10122}{10122+79}$$

= 0.9922

Specificity

$$= \frac{number\ of\ true\ negatives}{number\ of\ true\ negatives + number\ of\ false\ positives}$$

$$= \frac{390}{390+101}$$

= 0.7943

For case 9,

Sensitivity

$$= \frac{number\ of\ true\ positives}{number\ of\ true\ positives + number\ of\ false\ negatives}$$

$$= \frac{10024}{10024+177}$$

= 0.9826

Specificity

$$= \frac{number\ of\ true\ negatives}{number\ of\ true\ negatives + number\ of\ false\ positives}$$

$$= \frac{436}{436+55}$$

= 0.8879

4.2 Discussion

In summary, labelled string sequences were converted to images. Then using the obtained images, we conducted experiments in two phases I and II. In phase 1, best results were obtained when images were scaled to 7X500. This laid the foundation for testing in phase II where scale was fixed as 7X500 and number of epochs were varied. The best results in phase II testing were found in case of number of epochs as 30 and 40 were accuracy and validation accuracy were found to be:

Epochs	Accuracy	Validation Accuracy
30	98.92%	98.32%
40	99.02%	97.83%

High accuracies suggest this approach can be used as a starting point to build complex and heavier *nifH* classifiers.

4.3 Future Work

The results above have established a good starting point to address the problem statement. There is a lot of future work possible from here:

- This approach of converting sequences to images yielded promising results. Thus, this algorithm can be used to generate image dataset from sequences.
- Other sequence to image conversion algorithms is worth experimenting with.
- Boosted LeNet-4 has outperformed conventional LeNet-4. Thus, it will be interesting to observe boosting LeNet-5 results.

- Other bi-image classification algorithms such as Support Vector Machines integrated with feature reduction algorithm such as Principal Component Analysis (PCA) could produce high accuracy with a reduction in computation complexity.
- Lastly, experimentation with parameters used in convolutional neural network such as learning rate, batch size, using other loss functions could yield better results.

References

1. Fani R, Gallo R, Lio P. Molecular evolution of nitrogen fixation: The evolutionary history of the nifD, nifK, nifE, and nifN genes. *J Mol Evol*. 2000;51(1):1-11.

2. Heller P, Tripp HJ, Turk-Kubo K, Zehr JP. ARBitrator: A software pipeline for on-demand retrieval of auto-curated nifH sequences from GenBank. *Bioinformatics*. 2014;30(20):2883-2890.

3. Howarth RW, Marino R, Lane J, Cole JJ. Nitrogen fixation in freshwater, estuarine, and marine ecosystems. 1. rates and importance 1. *Limnol Oceanogr*. 1988;33(4part2):669-687.

4. Ueda T, Suga Y, Yahiro N, Matsuguchi T. Remarkable N2-fixing bacterial diversity detected in rice roots by molecular evolutionary analysis of nifH gene sequences. *J Bacteriol*. 1995;177(5):1414-1417.

5. Young J. Phylogenetic classification of nitrogen-fixing organisms. *Biological nitrogen fixation*. 1992;1544:43-86.

6. Zehr JP, Capone DG. Problems and promises of assaying the genetic potential for nitrogen fixation in the marine environment. *Microb Ecol*. 1996;32(3):263-281.

7. Zehr JP, Mellon MT, Zani S. New nitrogen-fixing microorganisms detected in oligotrophic oceans by amplification of nitrogenase (*nifH*) genes. *Appl Environ Microbiol*. 1998;64(9):3444. http://aem.asm.org/content/64/9/3444.abstract.

8. Zehr JP, McREYNOLDS LA. Use of degenerate oligonucleotides for amplification of the nifH gene from the marine cyanobacterium trichodesmium thiebautii. *Appl.Environ.Microbiol.* 1989;55(10):2522-2526.

9. Ueda T, Suga Y, Yahiro N, Matsuguchi T. Remarkable N2-fixing bacterial diversity detected in rice roots by molecular evolutionary analysis of nifH gene sequences. *J Bacteriol*. 1995;177(5):1414-1417.

10. Wu C, Berry M, Shivakumar S, McLarty J. Neural networks for full-scale protein sequence classification: Sequence encoding with singular value decomposition. *Mach Learning*. 1995;21(1-2):177-193.

11. Segrest JP, De Loof H, Dohlman JG, Brouillette CG, Anantharamaiah GM. Amphipathic helix motif: Classes and properties. *Proteins: Structure, Function, and Bioinformatics*. 1990;8(2):103-117.

12. Gaby JC, Buckley DH. A global census of nitrogenase diversity. *Environ Microbiol*. 2011;13(7):1790-1799.

13. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. *J Mol Biol*. 1990;215(3):403-410.

14. O'Shea K, Nash R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*. 2015.

15. Haykin S, Network N. A comprehensive foundation. *Neural Networks*. 2004;2(2004):41.

16. Deng L. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process Mag*. 2012;29(6):141-142.

17. Bottou L, Cortes C, Denker JS, et al. Comparison of classifier methods: A case study in handwritten digit recognition. . 1994:77.

18. Gu J, Wang Z, Wang G, et al. Recent advances in convolutional neural networks. *Pattern Recognition*. 2018;77:354-377. https://www.sciencedirect.com/science/article/pii/S0031320317304120. doi: 10.1016/j.patcog.2017.10.013.

19. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. *Proc IEEE*. 1998;86(11):2278-2324.