Digit classifier using three-layer Back Propagation Neural Network & four and six-layer Convolutional Neural Network using MNIST dataset on Keras framework

Yashad Samant
Master's of Electrical & Computer Engineering
Colorado State University

Abstract

This paper references to the implementation of couple of classifiers on the well-known handwritten digit dataset MNIST. We have implemented two algorithms, Back propagation Neural Network (Three-layer network) and Convolutional Neural Network (four and six-layer network) and compared the algorithms on the basis of accuracy/error (confusion matrix), computation time, computational complexity. We have also mentioned the previous implementations of algorithms on MNIST and how our implementations performed with respect to those. Thus, the basic objective of this paper is to give us an insight on the two algorithms and discuss their results on MNIST dataset.

I. Introduction

Text recognition has a rich history in pattern recognition and natural language processing has become one of the booming machine learning fields lately. We can trace back its developments in 1980s when three-layer neural network were used to recognize the text only to achieve accuracy of 40%. After the development of CNN in the late 90s, CNN produced an error rate of 0.7% which was a significant improvement on the neural net. As the years passed by, CNN developed further and in 2012, AlexNet got a breakthrough accuracy of 92% on the MNIST dataset getting a huge improvement on all the previous models.
While we have seen various implementations of neural networks like CNN, K-NN, Maximum Posterior Probability, Back propagation Neural Network, Self-organizing map etc. to evaluate results on MNIST dataset, achieving higher accuracies as good as human eyes is still a myth.
In this paper, we reference the same problem and try to achieve higher accuracy by modelling a three-layer back propagation neural network and compare its results to four and six-layer convolutional neural network. The data used is MNIST dataset which has total of 60,000 training samples and 10,000 testing samples. More than 15% of the data was used for the practical implementation of CNN and BPNN. The algorithms were implemented and tested in Keras framework with tensorflow used as backend in python programming language.

II. Data

DATASET: The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a result of an anti-aliasing technique used by a normalization algorithm. These images were centers using the center mass of the pixels and later translated to form a 28x28 image. Sample image is as follows:

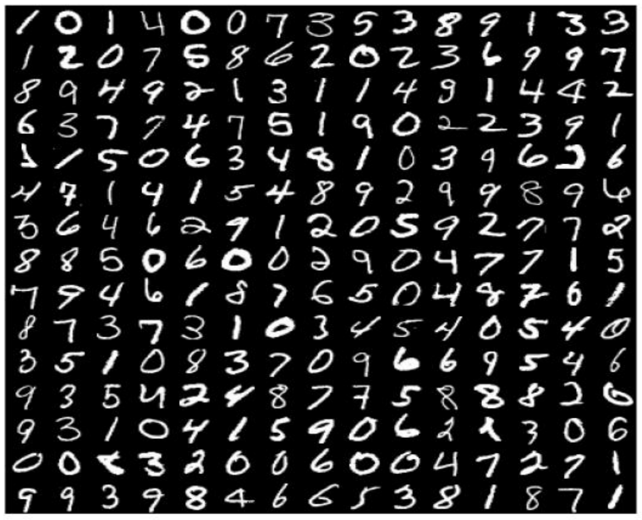


Fig 1: Sample data

To obtain relevant data for this assignment, we used the keras framework to read the whole data. Out of 60,000 samples, first 10,000 samples were selected as the training data and out of 10,000 samples from testing set, first 1000 were selected for testing and the next 1000 were selected for validation set. The number of samples were reduced to take into consideration the time constraint as training a network with 60000 samples will take hours of computations on a normal computer.
Standardization: The resultant data obtained from the preprocessing were sent through a standardization process. We implemented standardization by subtracting each value by the mean of the column and dividing by the standard deviation. Thus, the resultant data after standardization has zero mean and standard deviation equal to 1.
Standardization of data is very important as there's a possibility where the samples which are different range won't contribute equally to the analysis.
Finally, before going to the implementation of our algorithms let's go through the previous implementations and results on the MNIST dataset. The table is as follows:

TABLE I

PREVIOUS RESULTS ON MNIST DATABASE

Classifier	Preprocessing	Error rate (%)
Pairwise linear classifier	Deskewing	7.6 [12]
Boosted Stumps	Haar features	0.87 [14]
SVM	Deskewing	0.56 [15]
kNN	Shiftable edges	0.52 [13]
Neural Network	None	0.35 [16]
CNN	Width normalizations	0.23 [2]

III. ALGORITHMS

Convolutional Neural Network:

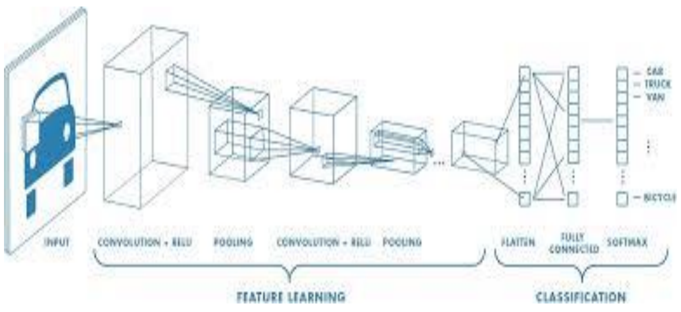


Fig 2: Structure of CNN

CNNs got their name from the convolution operation that takes place in the architecture. CNN is inspired by how neurons in the brain identify features. CNN consists of three main parts which are different from the normal neural nets.

1. Convolution layer: It is the first layer in the CNN and as the name suggests, its responsible for convolution operation between the weight matrix and the input matrix. Thus, the weight matrix learns the features of the image using this operation making it an integral part of the network.
2. Activation function: In tradition CNNs, we use RELU as the activation function which is basically a thresholding at 0.
3. Pooling layer: It is responsible for down-sampling the weight matrix to avoid the computational complexity.

We can connect a series of aforementioned networks making the CNN bigger making it easier to understand. As you increase the size of the network, number of computations increases.
The last layer of every CNN is a fully connected layer which accumulates the feature matrix together and it’s same as the ordinary neural networks. The fully-connected layer means that each neuron is connected to every other neuron in the previous layer.
The step-by-step graphical representation of the nine-layer CNN is given as follows:

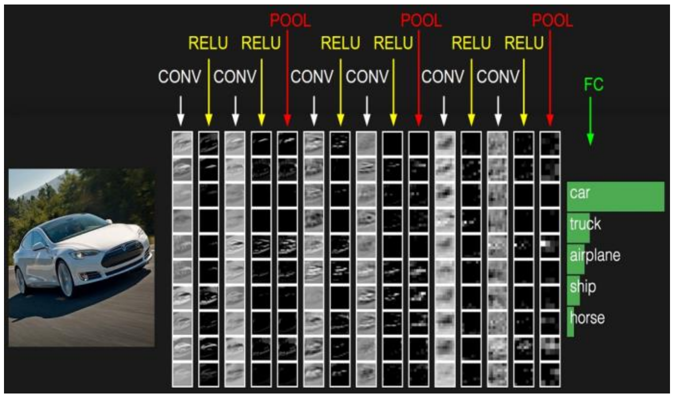


Fig 3: ConvNet

Back-propagation neural network:

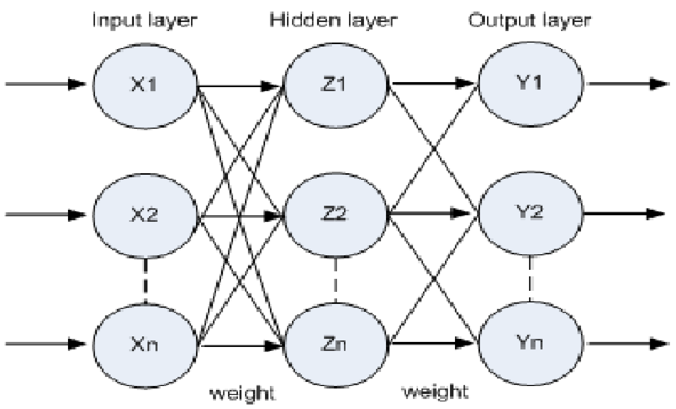


Fig 4: Three-layer BPNN network

Back propagation algorithm is the most commonly used algorithm in any kind of neural net. It is used to calculate the gradient which is needed in the updating of the weights after the error is produced.
Back propagation comes into the picture after the feed forward network evaluates the result and the error is calculated using between the true output and the predicted output. The error is propagated back to the first layer and weights are updated accordingly which makes the neural net model robust.

IV. IMPLEMENTATION

Convolutional Neural Network: Two types of CNN were implemented, one was a four-layer network and other was a six-layer network. Both the networks were designed with keras framework with tensorflow at the backend. As CNN is known to take data as feature map, input dimensions are given as [28,28,1]. The nets were designed as follows:

Four-Layer Network:

- Firstly, we initialized it as a sequential model as the input will pass sequentially through the network one layer after the other
- First layer was the convolution layer with the kernel size as 5x5, thus the 5x5 kernel will be sliding through the input vector which is now 32x32 because of the zero padding so that the corner features are not missed. The

result of the convolution is passed through a RELU activation function.

- The second layer is a pooling layer which finds out the max value within a 2x2 window and down-samples the feature matrix.
- The resultant down-sampled matrix then goes through another convolution where the kernel size is again 5x5 with a RELU at the end.
- The fourth layer is again max pooling layer with again a window of 2x2.
- We also added a dropout layer in between to drop all the sticky weights to increase the accuracy of the model.
- The result is then given to a fully connected network with 1000 neurons which are fully connected which are later connected to output layer with softmax activation, consisting of 10 neurons representing 10 labels or classes.

Six-Layer Network:
This network was designed similar to four-layer network with the addition of two more layers. The sequence is as follows:

- Firstly, we initialized it as a sequential model as the input will pass sequentially through the network one layer after the other
- First layer was the convolution layer with the kernel size as 5x5, thus the 5x5 kernel will be sliding through the input vector which is now 32x32 because of the zero padding so that the corner features are not missed. The result of the convolution is passed through a RELU activation function.
- The second layer is a pooling layer which finds out the max value within a 2x2 window and down-samples the feature matrix.
- The resultant down-sampled matrix then goes through another convolution where the kernel size is again 5x5 with a RELU at the end.
- The fourth layer is again max pooling layer with again a window of 2x2.
- Fifth layer is again a convolution layer, now with a sliding window of 4x4 with RELU at the end.
- It is followed by a max pooling layer with window 2x2.
- Again, a dropout layer was added to drop all the sticky weights to increase the accuracy of the model.
- The resultant is then given to a fully connected network with 1000 neurons which are fully connected which are later connected to output layer with softmax activation, consisting of 10 neurons representing 10 labels or classes.

Back-propagation neural network:
A simple back propagation network was implemented with three layers: input layer, hidden layer and output layer. Input layer consisted of 1000 neurons while hidden layer consisted of 500 neurons and output layer was just ten neurons representing 10 labels. The input dimensions were 784 as the image was represented as vector and passed through the network. The learning rate was taken to be 0.01 and momentum factor was also initialized as 0.01 to accelerate the convergence. The BPNN algorithm uses delta rule to update weights and consequently train the network. K-folds method was used with k=10 to counter the problem of local minima. BPNN is known for local minima problem and hence it's advisable to use k-folds so that we can start training the model with different initial weights and chose the best model for training. The model was implemented in keras framework with tensorflow in the background in python programing language.

V. Experimental Results

Both the algorithms were successfully implemented in python. We achieved appreciable results even after using a reduced dataset. The standard for variable factors for comparison were set as follows:
Batch size: 100
Learning rate: 0.01
Epoch: 20
Momentum factor: 0.01
Data: Training – 10,000, Validation – 1000, Test – 1000
Activation function: RELU, SOFTMAX
Error: MSE
Changing these factors affect the algorithms in different ways as discussed below.

Convolutional Neural Network: Convolution neural network was the best algorithm among the two giving very high test accuracy which was pretty much expected from the state-of-the-art.
CNN has various variable factors such as the learning rate, batch size, addition of dropout layer, activation functions, dependence on data etc. Each factor and its effects are mentioned below:

1. Batch size: Lesser the batch size, better is the accuracy. It affects both the computation time and accuracy. It's basically a trade-off where as we increase the batch size the accuracy decreases while the time increases and vice-versa.
2. Learning rate: It also creates a trade-off between time and accuracy, lower the learning rate higher the accuracy but consequently larger the time and vice-versa.
3. Dropout Layer: Accuracy increases after addition of dropout layer. It's obvious, as the model is trained better when the insignificant weights are dropped.
4. Activation: In this implementation, we tried using various activations function and it affects the model differently. Mostly, it's dependent on data and for this data RELU works better than others.
5. Error: In this implementation, MSE was used to calculate error but we can use any error function and try to minimize it.

Four-layer network:
The four-layer CNN worked very well giving us a training accuracy of 95.16% while the validation set obtained an accuracy of 93.21%.

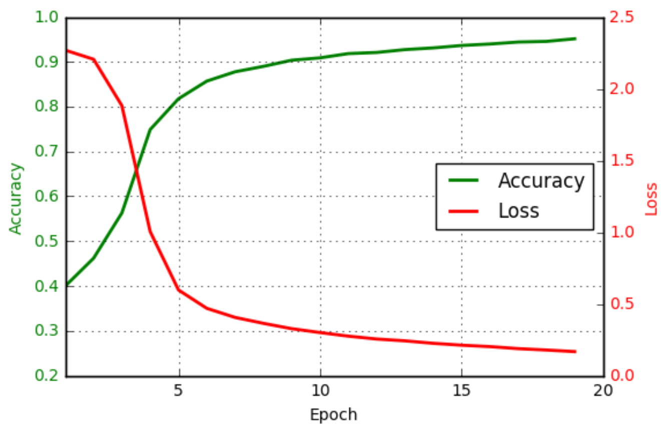


Fig 5: Four-layer computation results

The above plot gives us the evidence of how well the model is trained. We have number of epochs on the X-axis and accuracy and error forms the y-label where green line is represented by accuracy and red line is represented by error. We can very well see that that model has trained successfully with the increasing epochs.
Model was trained for 20 epochs with it being trained properly within 10 epochs giving an accuracy above 90%. I believe there was more scope of training the model with the increasing epochs.
We could also observe the error decreasing and in our case it decreased to mere 0.18%.
The addition of dropout layer in the CNN increased the accuracy by another 0.92% which is a significant deal and thus we can see how important dropout can be.
The four-layer network running for 20 epochs gave a test accuracy of 94.1% while the validation dataset gave an accuracy of 93.21%. The training set was trained with 95.16% accuracy.
Confusion Matrix: Confusion matrix is used to represent most of the classification. In the matrix, rows and columns represent the classes and ideally, we should get high numbers on the diagonal and rest of the matrix should be 0.

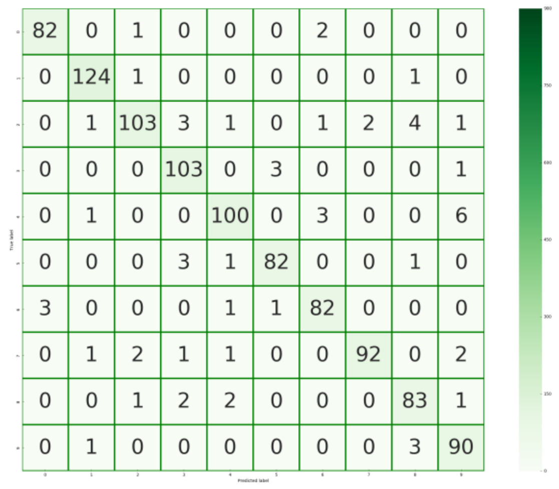


Fig 6: Four-layer confusion matrix

In this case, we have achieved good results where ‘0’ was represented correctly 82 times while it was represented as ‘6’ thrice. Similarly, we have got high accuracy for other labels where model has correctly classified the digit.
0: Correct-82, Wrong-3
1: Correct-124, Wrong-2
2: Correct-103, Wrong-7
3: Correct-103, Wrong-4
4: Correct-100, Wrong-10
5: Correct-82, Wrong5
6: Correct-82, Wrong-2
7: Correct-92, Wrong-5
8: Correct-83, Wrong-5
9: Correct-90, Wrong-4

Six-layer network:

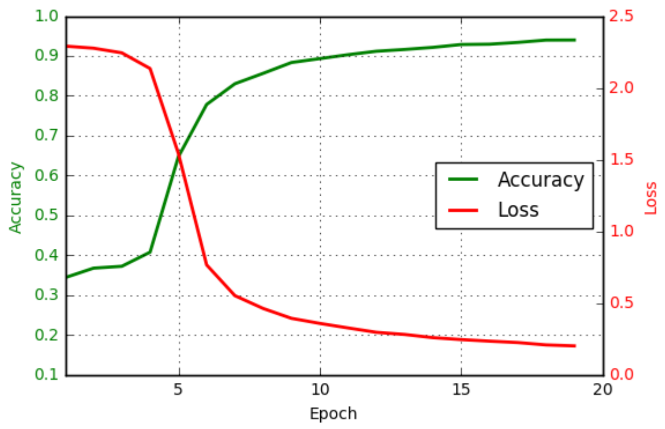
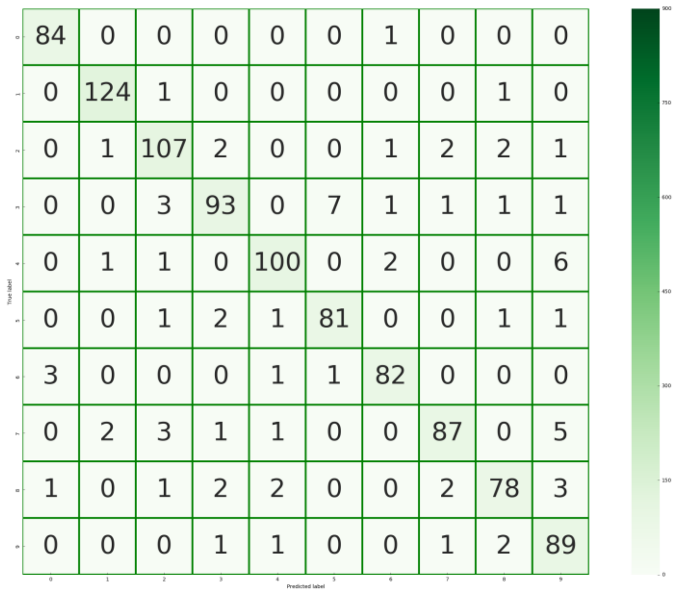


Fig 7: Six-layer computation results

We can very well see that that model has trained successfully with the increasing epochs.
Model was trained for 20 epochs with it being trained properly within 10 epochs giving an accuracy above 90%. I believe there was more scope of training the model with the increasing epochs as the slope of the line doesn’t seem to be stable. Thus, if we increase the number of epochs, we can increase the accuracy of the model better.
We could also observe the error decreasing and in our case it decreased to mere 0.20%.
The addition of dropout layer in the CNN increased the accuracy by another 0.81% which is a significant deal and thus we can see how important dropout can be.
The six-layer network running for 20 epochs gave a test accuracy of 92.5% while the validation dataset gave an accuracy of 91.17%. The training set was trained with 93.99% accuracy.



84	0	0	0	0	0	1	0	0	0
0	124	1	0	0	0	0	0	1	0
0	1	107	2	0	0	1	2	2	1
0	0	3	93	0	7	1	1	1	1
0	1	1	0	100	0	2	0	0	6
0	0	1	2	1	81	0	0	1	1
3	0	0	0	1	1	82	0	0	0
0	2	3	1	1	0	0	87	0	5
1	0	1	2	2	0	0	2	78	3
0	0	0	1	1	0	0	1	2	89

Fig 8: Six-layer confusion matrix

We can compare the result as done in four-layer network.

Four-layer vs Six-layer:

Usually, we would assume that the six-layer network would work better than a four-layer network as it has larger number of neurons, thus able to learn more features but it didn't happen in this case.
This could be few reasons for the performance constraint:
Data: While 10,000 images were enough to train a four-layer network, we might need more images to train a six-layer network. This is because they have higher number of weights than a four-layer network.
Batch size: Decreasing the batch size, increases the performance of the model.
Learning rate: Lowering the learning rate will increases the accuracy of the model.

Back-propagation neural network:
BPNN didn't work as great and consistently as CNN giving variable accuracies. This might be because of being stuck in different local minima. Due to that, we weren't getting a consistent validation accuracy as well. It's test accuracy has ranged from 53% to 86% with the base factors that we have used. The other factors which we can use to improve the accuracy of the model are:

1. Batch size: Lesser the batch size, better is the accuracy. It affects both the computation time and accuracy. It's basically a trade-off where as we increase the batch size the accuracy decreases while the time increases and vice-versa.
2. Learning rate: It also creates a trade-off between time and accuracy, lower the learning rate higher the accuracy but consequently larger the time and vice-versa.
3. Dropout Layer: Accuracy increases after addition of dropout layer. It's obvious, as the model is trained better when the insignificant weights are dropped.
4. Activation: In this implementation, we tried using various activations function and it affects the model differently. Mostly, it's dependent on data and for this data RELU works better than others.
5. Error: In this implementation, MSE was used to calculate error but we can use any error function and try to minimize it.
6. Momentum factor: It is used to converge the network faster and it ranges from 0 to 1. 0 represents normal BPNN. Higher the value faster the network will converge but there's a possibility of unwanted oscillations while implementing gradient descent. Moreover, it might decrease the accuracy.

From the computation results we can see that model is trained successfully. BPNN was trained for 20 epochs and we can see the error decreasing over the period of time.
We achieved a training accuracy of more than 80% while the error is reduced to 0.5%.
We could also assume that the accuracy will be better with the increase in epochs as the line is not stable.
We got a training accuracy of exactly 85.62% while the validation accuracy was 82%.
The test accuracy was appreciably less as compared to CNN models. We got a accuracy of 75%.

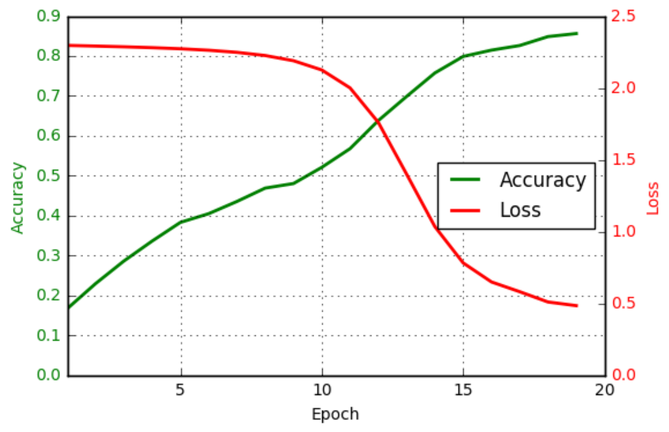
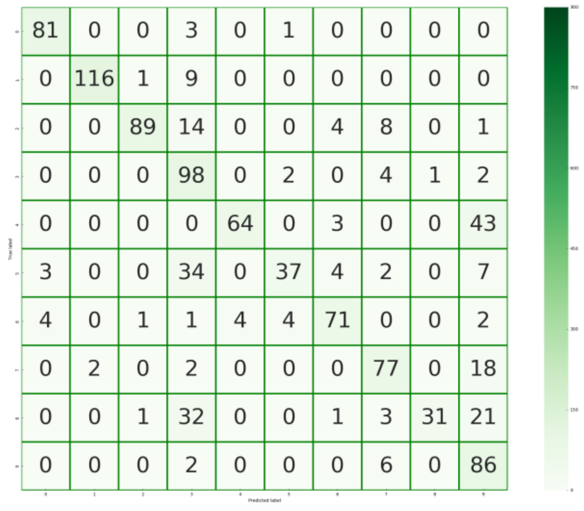


Fig 8: BPNN computation results



81	0	0	3	0	1	0	0	0	0
0	116	1	9	0	0	0	0	0	0
0	0	89	14	0	0	4	8	0	1
0	0	0	98	0	2	0	4	1	2
0	0	0	0	64	0	3	0	0	43
3	0	0	34	0	37	4	2	0	7
4	0	1	1	4	4	71	0	0	2
0	2	0	2	0	0	0	77	0	18
0	0	1	32	0	0	1	3	31	21
0	0	0	2	0	0	0	6	0	86

Fig 9: BPNN confusion matrix

We can observe in the confusion matrix that the results are not as great as the CNN model. We can see that many classes are incorrectly predicted.

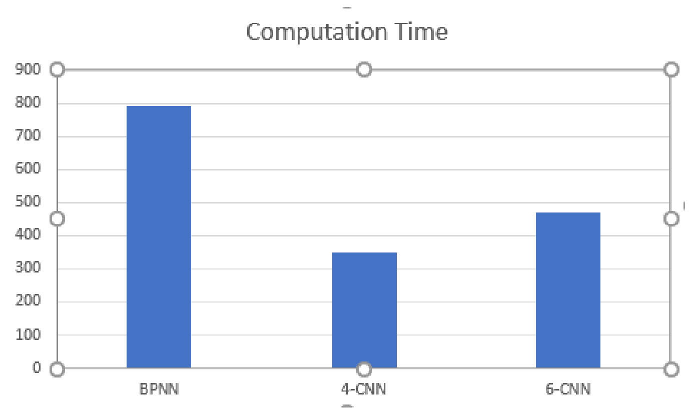


Fig 10: Computation time

In the above plot the computation time of each algorithm is plotted with standards as mentioned in the implementation. The y-axis is the time in seconds. As expected, BPNN takes enormous amount of time to converge as it consists of three fully-connected layers. It takes more than double the time of 4-layer CNN. We expect higher computation speed from CNN as it is not fully-connected and we reduce the size of operations by using pooling layer. Six-layer CNN takes more time than four-layer CNN because of more number of layers.

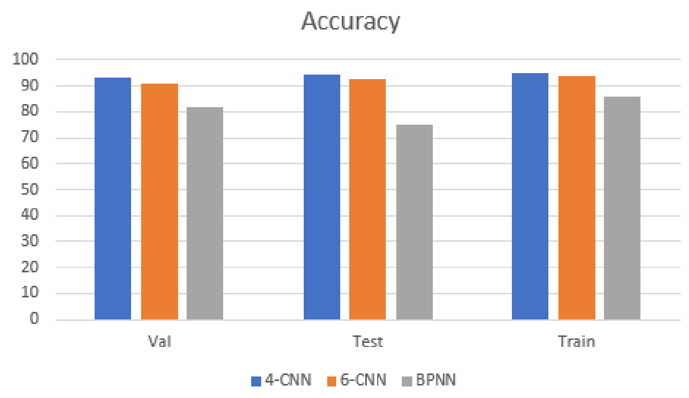


Fig 11: Computation performance

In the above plot, we can see the validation, test and train accuracy of each algorithm is plotted. While we can see that 4-CNN has outperformed others but the difference is not that much.
But we can also say that BPNN not as computational complex as the CNNs. CNN include the use of convolutional layers which increases the complexity of the algorithm.

VI. Conclusion

Thus, we can conclude that 4-layer network works better than the other network with an accuracy as high as 97% with best variable factors. Six-layer CNN performance curve is almost as good as four-layer network but it needs either more data or more epochs to train perfectly. BPNN works well with best variable factors but is inconsistent because of it's local minima problem. Even after using k-folds it shows signs of inconsistency. Thus, in the day-to-day applications such as facial recognition, CNN is most consistent, accurate and less time-consuming application but is more computational complex. But with the development of multi-core processors and parallel programming languages like CUDA, CNN computations can be handled very efficiently, thus making it the best in the game.

VII. References

1. http://yann.lecun.com/exdb/mnist/
2. http://cs231n.github.io/convolutional-networks/
3. https://en.wikipedia.org/wiki/Backpropagation
4. https://www.ripublication.com/ijaer17/ijaerv12n17_70.pdf
5. https://www.tensorflow.org/api_guides/python/nn
6. Y. LeCun. (2013) Lenet-5, convolutional neural networks. [Online]. Available: http://yann.lecun.com/exdb/lenet/