EE 569 HOMEWORK 4

GUNJAN JHAWAR
gunjanjh@usc.edu

Contents

1.Problem 1

1.1Abstract and Motivation

Convolutional Neural Network are inspired by visual cortex of human brain. They are made up of neurons that have learnable weights and biases. Each neuron gets an input passes through a series of hidden layer and provides an output. 012 was the first year that neural nets grew to prominence as Alex Krizhevsky used to win ImageNet dropping the classification error record from 26% to 15%, an astounding improvement at the time. [1] The fundamental understanding of how it works is still lagging though. This poses a challenge to understand how the process inside the layer works and how the layers are designed. This knowledge can be utilized to improve the accuracy of Convolutional Neural Network.

In this problem, we look in detail the working and architecture design of Convolutional Neural Network. Explaining over fitting and why does it come up as an issue. Compare how CNNs work better than traditional methods in computer vision problems. Explaining the loss function and the classical back propagation optimization procedure to train a neural network.

In this problem, we will train Le-Net-5 in MNIST dataset. Examine how different network parameters affect. Compare accuracy using different parameters and plotting the performance curves

In this problem, we try to modify the baseline CNN. Trying different activations functions and optimization algorithms. Reporting the best accuracy achieved among the different network architecture and training parameter settings.

1.2Approach and Procedures

It explains CNN Training and its application to the MNIST Dataset. There are three parts in this section. The first part describes the architecture and operational mechanism of convolutional neural network. Answers various questions on the same described in A. Then Le-Net 5 is trained on MNIST Dataset and observation is done as described in B. The aim is to improve the Le-Net-5 for MNIST Dataset by trying different optimization algorithms and network parameters as described in C.

1.2.1CNN Training and Application to MNIST Dataset

A more detailed overview of what CNNs does it that it takes the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output. The output can be a single class or a probability of classes that best describes the image. But the important part is understanding what each of these layers do.

A. CNN Architecture and Training:

a)

Convolutional layer: The first layer in CNN is always a convolutional layer. The input to the layer is the input image say a 32 ×32×3 array of pixel values. Then we use a filter which slides over the entire input image say the filter is 5×5×3. The important thing to note is that the depth of filter (also referred as kernel) must be equivalent to that of the input image. As the filter slides over the input image, it multiplies the values in the filter with the original pixel values of the image. Then all the multiplications are summed up and the single value that is obtained is put to location corresponding to the input image.

Padding: Based on filter size, after convolution, the input image size id reduced. In such case filter is not applied to border elements. To apply filter to border images, the image must be zero padded at the border. This allows control over size of feature maps.

Stride: Now, the next step would be moving the filter right by the stride number say 1 unit for example. The same process is repeated for every location in the input volume. After sliding we obtain a 28 ×28×1 array which is referred as feature or activation map. The mathematics behind getting that is:

$$\frac{\mathrm{I/P} - \mathrm{F} + 2\mathrm{P}}{S} + 1 = O/P$$

I/P refers to size of the input image
F refers to the size of filter
P refers to padding
S refers to stride
O/P refers to the size of output obtained

For above example: I/P = 32 ×32, F = 5×5, P = 0, S = 1

$$\mathrm{O/P} = \frac{32 - 5 + 2*0}{1} + 1$$
$$= 28$$

O/P obtained is 28 ×28

The last thing to note is that if we use 2 filters, each of size 5×5×3 then the output volume would be 28×28×2.
The aim of using convolution layer is that each filter that is used in it is actually a feature identifier. The values assigned to a filter is such that it detects a necessary feature in an image. Feature could be edges, curves, etc.

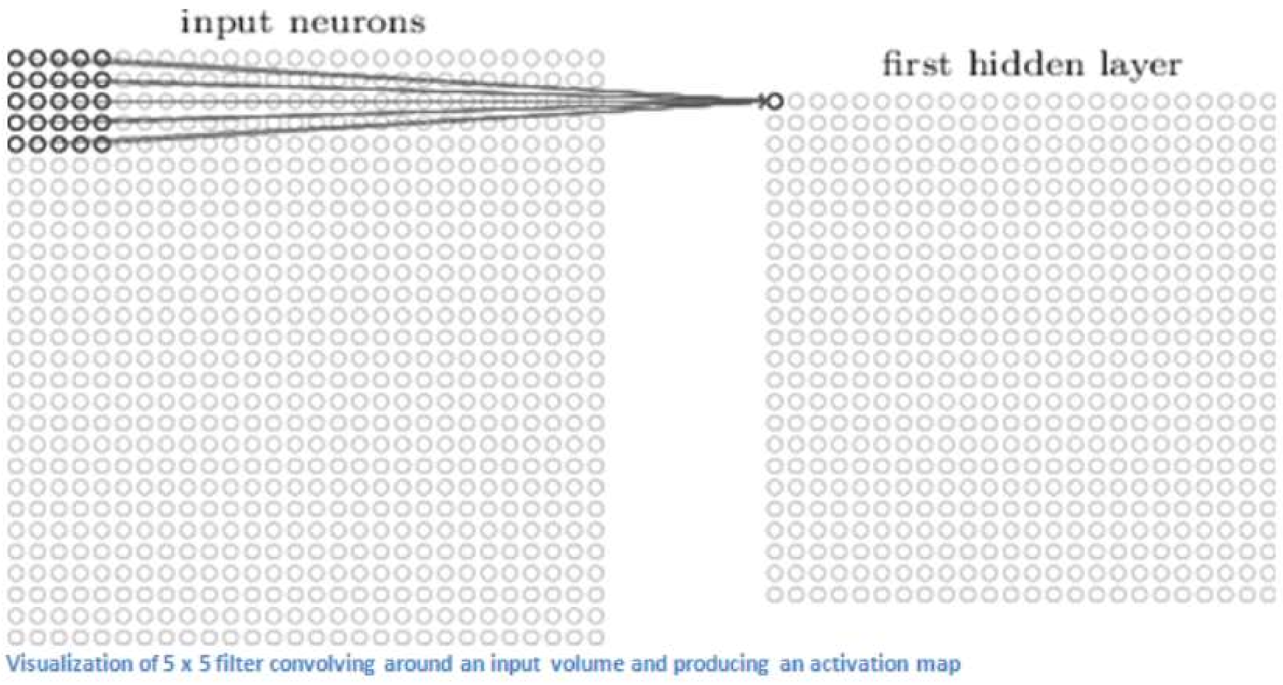

input neurons
first hidden layer
Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Figure 1Showing how convolution occurs using filter[1]

Activation function: After every convolution layer, a nonlinear layer is applied immediately after it. The purpose of using this layer is to introduce non- linearity to the system. The convolution performed before applying an activation is a linear function and to make the system more robust non linearity is introduced.

There are number of different nonlinear functions that can be applied like tanh,sigmoid, ReLU. Researchers found that ReLU layers work far better because the network is able to train a lot faster without affecting the accuracy much. It also helps in solving the problem of vanishing gradient. The ReLU function is:

$$f(x) = \max(0, x)$$

This is applied to all values in the input volume. As we can see that the function changes all negative values to 0.

Sigmoid function: It is nonlinear function and thus could be used as an activation function. It has a smooth gradient and is bounded to (0,1). It is popular activation function when performing classification.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Tanh function: It is a nonlinear function. It is bound to range of (-1,1) so activation would not blow up. Deciding between tanh and sigmoid will depend on the gradient strength requirement. Although tanh has a stronger gradient.

$$f(x) = \frac{2}{1 + e^{-2x}}$$

Max pooling layer: It can be said to a down sampling layer. There can be different kinds of pooling layer but max pooling is the most commonly used. It takes a filter and a stride. It then applies it to the input volume and outputs the max of that region. The filter moves around the whole image only preserving the maximum volume found in the filter.

The aim of using it is that it reduces the parameters thus lessening the computation cost. As we say that where ever the filter moves it takes the maximum volume out of the whole volume of the filter. Also, it controls overfitting. It eliminates the pixel value which might be contributing to overfitting.

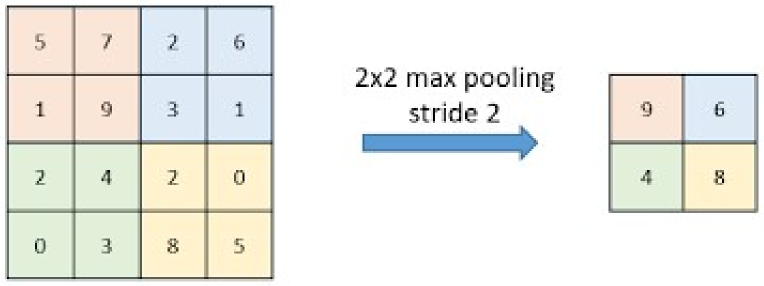


Figure 2Max Pooling [2]

Fully connected layer: After performing convolution, pooling layer and using activation function, we have features with us which represents the image. Now, our aim is to take out the features that most correlate to a class out of the high-level features that we have found from the previous layer. For example, if the program predicts a car, it will have high values in activation maps that represent the door of a car, steering etc. So, the FC layer looks at the high-level features that most correlate to a particular class and assign particular weights so that when we compute the product of weights and the feature we get the correct probabilities for different classes.

SoftMax function: The SoftMax function when applied the output layer gives the output of the layer in terms of a probability distribution. It scores and map each value between 0 and 1. As the neuron carries probability of the input feature to be in the class. So, the overall sum of probabilities of any layer should be 1. SoftMax function turns out to be handy in this case. Also, the network is commonly trained using log loss under which using SoftMax function at the output layer seems more beneficial as the exponential in the SoftMax function roughly cancels out the log in cross entropy loss.

b)

Overfitting occurs when the model considers too much information from the training data or the prior knowledge and make the model very precisely on the observation of the training data and fails to fit additional data or predict future observations. The best way to understand over fitting is suppose you give task to a couple of students to make a model to predict the growth of some kinds of plants. The collect the information of the plant throughout the year and make a model out of it. In the model, it will predict the exact characteristics for the data similar to the training data and produce very fine results. But if the plants suffer from seasonal change, the model will not be able to predict. Thus, overfitting the training data and not being able to generalize well to the test data which have variations in comparison to the training data.

While training the CNN, pooling layers helps to reduce overfitting. Pooling tries to do feature selection by reducing the dimension of the feature. Overfitting on the other hand can be thought of as fitting patterns that do not exist due to high number of features. So, by removing or selecting a subset of features it is less likely to find false patterns, which is what pooling does. The idea to select a subset of features is commonly known as dropout.

Other than max pooling, training with more data might help algorithms to detect the signal better. Early stopping can be used as the model is trained iteratively, at every iteration it can be seen how well the model performs and the point at which the model's ability to generalize seems to get weaken means it gets less accuracy on the test data, it should be stopped at that iteration. Methods like regularization, bagging and boosting would also help though they are not performed while training the network.

c)

CNN is a feed forward artificial neural network. The network is trained in such a way that it uses minimal amount of processing. They exploit spatial relationship rather than spectral relationship. CNN is observed to work better than traditional methods in computer vision. Though it is true but the traditional methods are not obsolete because the neural network still has a lot of drawbacks like it needs a lot of training data to perform well, it does not have mathematical foundation underlying its working it is still more of a hit and trial method to train its hyperparameters for example etc.

The regions where CNN outperform traditional methods are:

1) **Hierarchical learning Vs Human Driven:** CNN have an inherent advantage of learning hierarchical features, to find out which features are useful, how to compute them and how much to weigh them for the model to perform well. On the other hand, the traditional way figures out the features relevant to the problem, figure out to compute those features and then use those features to compute result. In case of image classification, filter defined in convolutional layer takes care of different features but as the network moves forward, it weighs and updates which features are more useful using back propagation.

2) **Dimensionality reduction** is done within the layer of the convolutional neural network. The layers are not fully connected, the perform procedure like max pooling which reduces the dimensionality as well as prevent over fitting. On the other hand, in case of traditional methods algorithms like PCA must be implemented to reduce the dimensionality.

3) **Easier to develop**: CNN are easier to develop in comparison to the traditional methods used in CV. In terms of image classification, we can use the different layers and try different architecture

like Conv-> Pool->Conv->FC or Conv->Pool->FC but the basic layers remain the same while in traditional methods extensive study is needed on which algorithm to use in the given case for example, to use SIFT/ SURF/BRIEF.

4) **Sensitive to transformation**: Traditional methods have drawbacks like certain image features like edges and corners are not scale invariant when SIFT is implemented. Whereas, CNN performs well as the training data is augmented to make the model robust to transformation.

d)

The loss function guides the training process of a neural network. For classification, mean squared error and cross entropy loss are widely used and for regression L1 loss is usually used. It basically determines the difference between the ground truth label and the prediction by the model.

Mean Squared Error is the most intuitive loss function. It is a straight lune between two points in Euclidean space in neural network. We apply back propagation algorithm to iteratively minimize the MSE so that the network can learn from the data. When it sees similar data to that of the training it gives similar output to the training one.

$$MSE = \frac{1}{n}\sum_{1}^{n} e^2$$

$$\textit{where n is the number of classes and e is the difference between the ground truth and model prediction}$$

Cross entropy loss is more advanced than mean squared error, the induction of cross entropy comes from maximum likelihood estimation in statistics. The cross entropy is:

$$H(y) = -\sum_{1}^{n} y' log(y)$$

$$\textit{where } y' \textit{is the ground truth label and } y \textit{ is the prediction of the classifier}$$

Back propagation is used to learn the parameters of CNN such as weight corresponding to every layer. To explain, how all of it work let's suppose $(x_{i,}z_{i,})$ are the desired input-output related where x represents the input data and z represents the output values.
l $(z, z_{i,})$ denotes the penalty of predicting $z_{i,}$ instead of z.
Thus, the loss function L can be expressed as:

$$L(w) = \frac{1}{n}\sum_{1}^{n} l(z, f(x, w))$$

Our aim is to minimize L and that is done using gradient descent most commonly. The formula of gradient descent is as follows [3]:

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta_t \frac{\partial f}{\partial \mathbf{w}}(\mathbf{w}^{t})$$

Now, lets look at the CNN where f is the composition of L layers f_l each with parameters w_l . The chain will look like

$$\begin{tikzcd}
\mathbf{x}_0 \arrow[r] & \boxed{f_1} \arrow[r] & \mathbf{x}_1 \arrow[r] & \boxed{f_2} \arrow[r] & \mathbf{x}_2 \arrow[r] & \cdots \arrow[r] & \mathbf{x}_{L-1} \arrow[r] & \boxed{f_L} \arrow[r] & \mathbf{x}_L \\
& \mathbf{w}_1 \arrow[u] & & \mathbf{w}_2 \arrow[u] & & & & \mathbf{w}_L \arrow[u] &
\end{tikzcd}$$

During learning, the last layer of the network corresponds to the loss function that needs to be minimized. Back propagation allows computing the output derivatives in a memory efficient way using tensors. It stacks all vector together and the computation for all neurons present in a layer is done all at once for all the training data instead of doing it one after the another.

The basic idea is to compute the derivatives $dx/_{dw}$ over every layer and update it using the formula of gradient descent and the L, loss function described above measures how well the network is performing and how much the parameters must be updated.

B. Train Le-Net-5 on MNIST Dataset

a) Initialization of network parameters

A proper initialization of weights in neural network is critical for convergence and final quality of the network. When we talk about parameters we mean parameters like learning rate, number of epochs, batch size, filter size, filter weight etc.

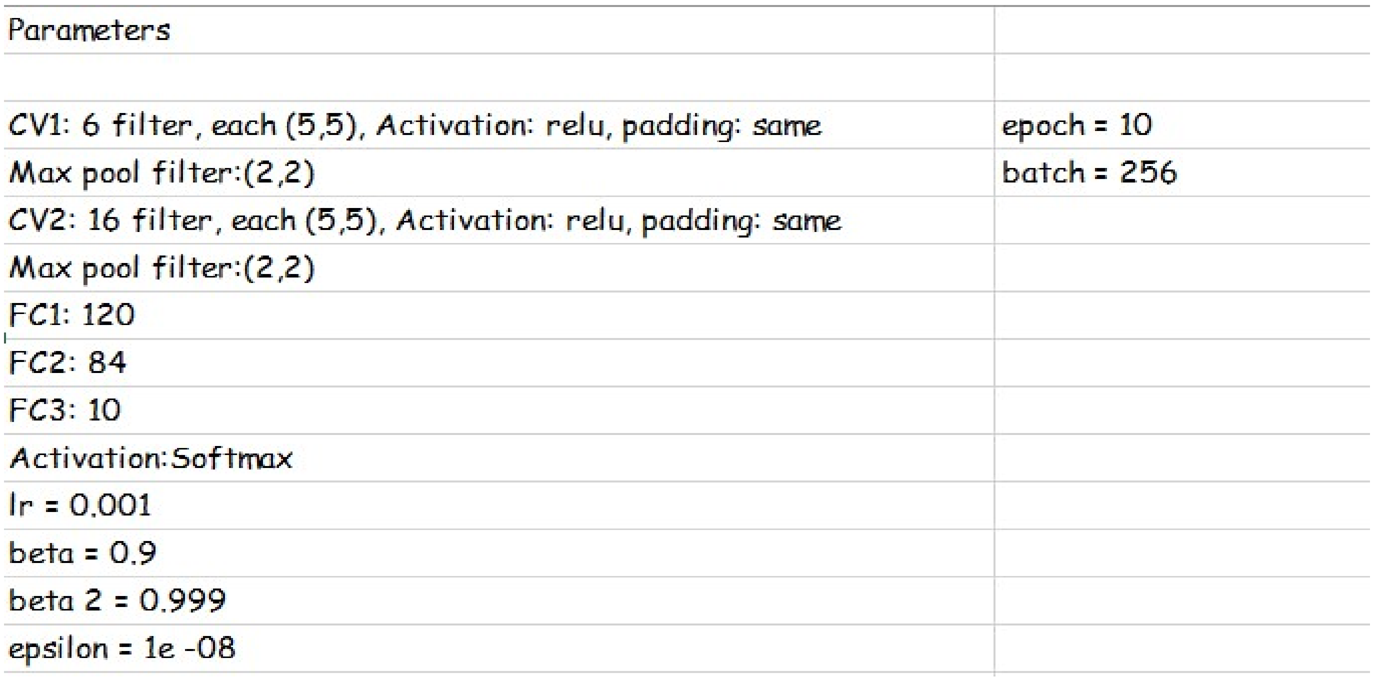
Effect of different parameters:

1. **Weight initialization**: If we initialize the weights to zero, the model does not learn anything. It turns out that the neuron in the network computes the same output and the same gradient during backpropagation and undergo the same parameter updates. Random initialization helps the network to perform distinct updates and integrate all that in one network. Glorot initialization work well for trivial and tanh activations but do not work well with ReLU activations. Normal initialization tries to spread the network weights in a gaussian way.
2. **Number of Epochs**: They highly depend on the ANN structure/complexity as well as the complexity of the problem. They are useful to see how the model is performing as more and more data is introduced to it. This can be done for both training and validation data to find out where the data is enough for the network to perform well.

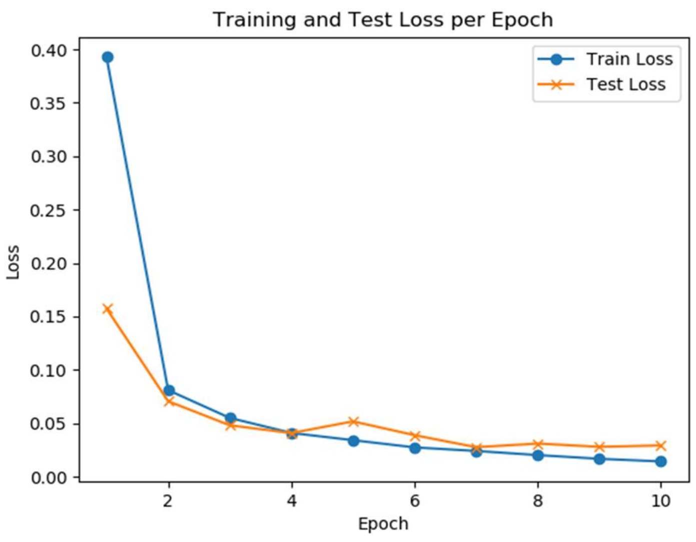
3. **Batch Size:** It determines the number of samples that will be propagated through the network. The advantage of using it is that it requires less memory. It plays a major role where dataset is so large unable to fit in memory. The network trains faster in mini batches. The gradient is not accurately measured when done in mini batches that is its limitation.
4. **Learning rate:** The step size determines how fast a local optimum is reached. It has to be decided carefully as we might get an output but that can be local optimal instead of global. The optimization algorithm does their own tuning to find the step size that searches for the parameter that minimizes the loss function. The larger learning rates appear to help the model locate regions of general, large-scale optima, while smaller rates help the model focus on one local optimum.
5. **Filter size:** The size of filter determines the kind of feature it extracts. For small size of filters, they extract local features and small details of the image while for high size filter extracts the more complex features combined by primitive shapes.

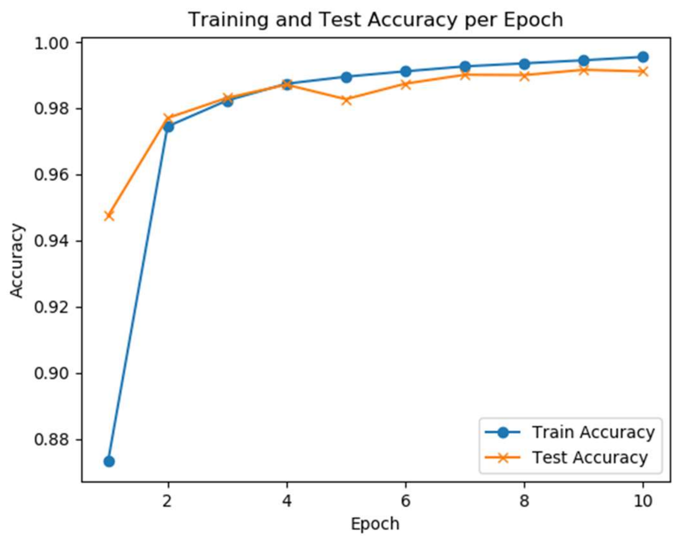
1.3Experimental Results for B

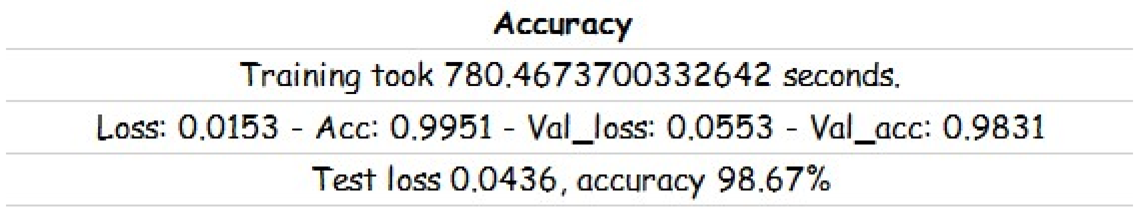
a. Base Model



Parameters	
CV1: 6 filter, each (5,5), Activation: relu, padding: same	epoch = 10
Max pool filter:(2,2)	batch = 256
CV2: 16 filter, each (5,5), Activation: relu, padding: same	
Max pool filter:(2,2)	
FC1: 120	
FC2: 84	
FC3: 10	
Activation:Softmax	
lr = 0.001	
beta = 0.9	
beta 2 = 0.999	
epsilon = 1e -08	

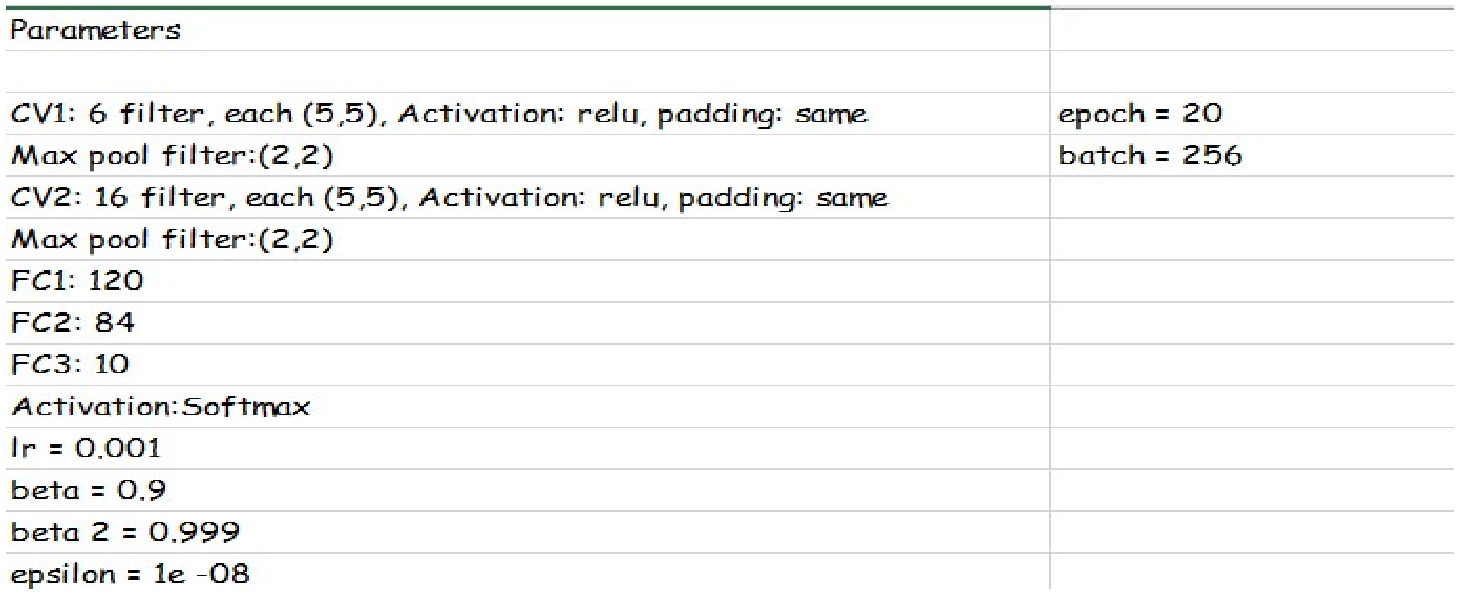




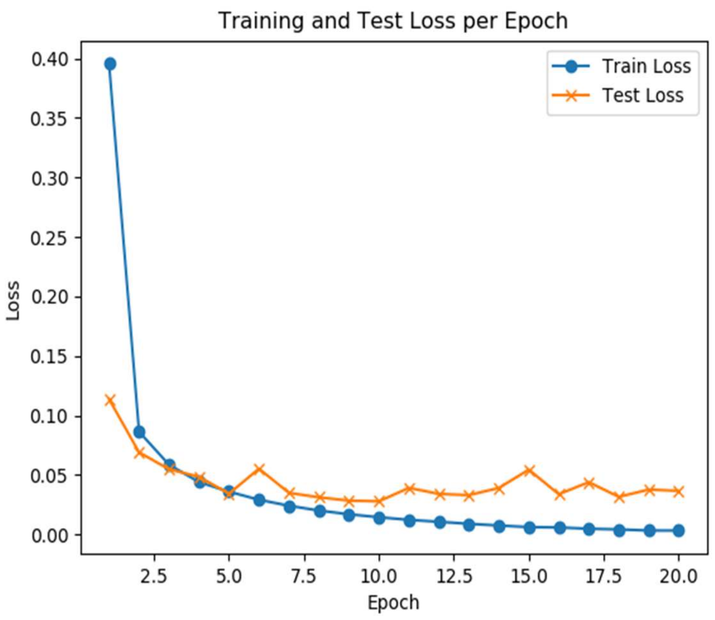


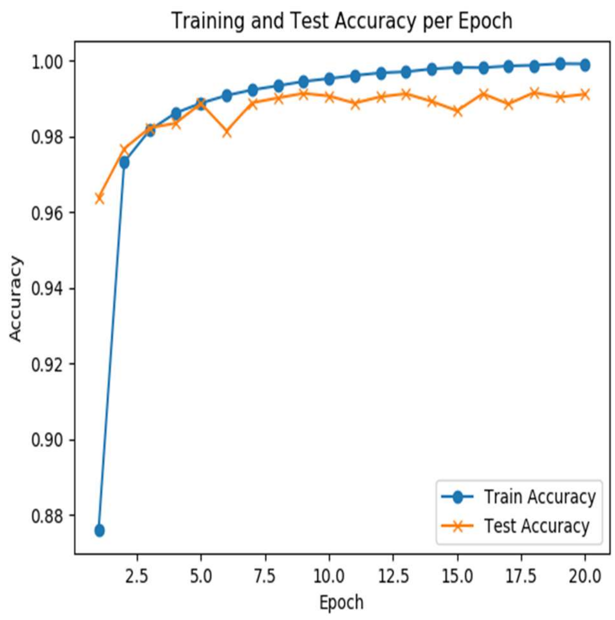
Accuracy
Training took 780.4673700332642 seconds.
Loss: 0.0153 - Acc: 0.9951 - Val_loss: 0.0553 - Val_acc: 0.9831
Test loss 0.0436, accuracy 98.67%

b. Epoch increased



Parameters	
CV1: 6 filter, each (5,5), Activation: relu, padding: same	epoch = 20
Max pool filter:(2,2)	batch = 256
CV2: 16 filter, each (5,5), Activation: relu, padding: same	
Max pool filter:(2,2)	
FC1: 120	
FC2: 84	
FC3: 10	
Activation:Softmax	
lr = 0.001	
beta = 0.9	
beta 2 = 0.999	
epsilon = 1e -08	

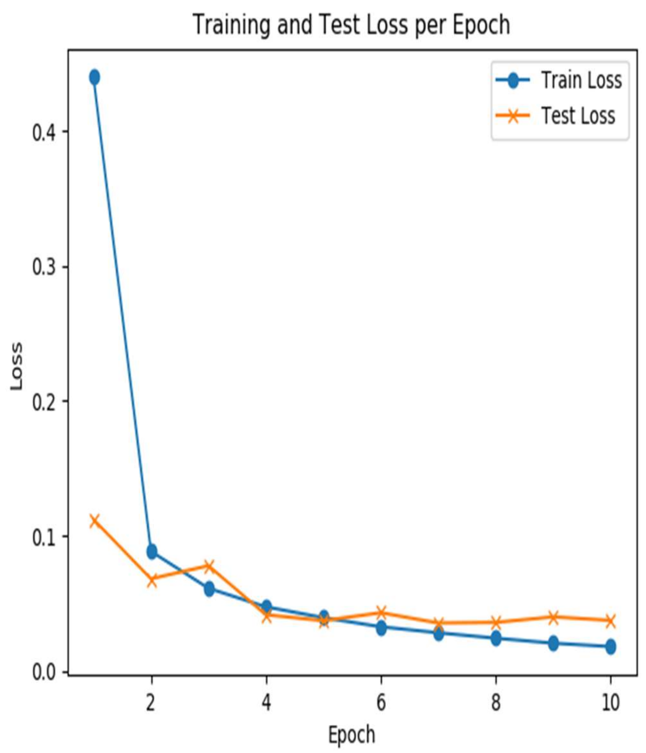


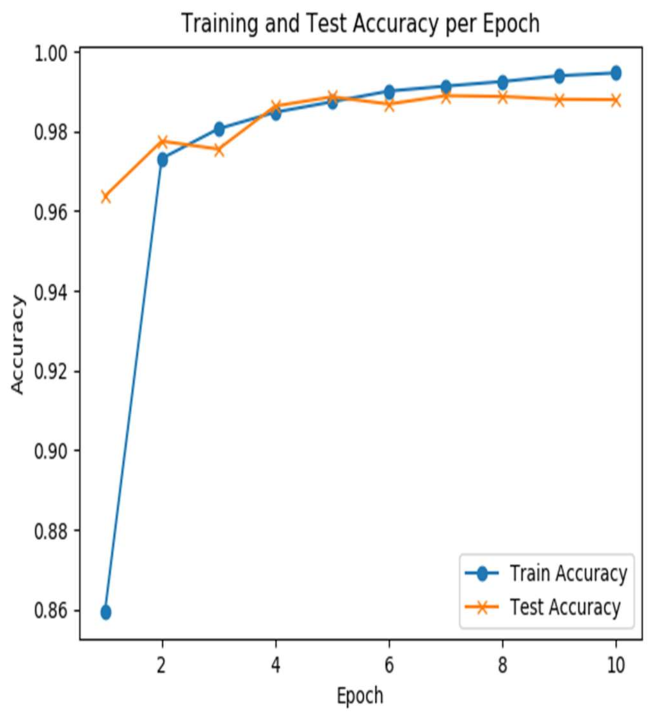


Accuracy
Training took 1389.2134075164795 seconds.
Loss: 0.0034 - Acc: 0.9991 - Val_loss: 0.0367 - Val_acc: 0.9911
Test loss 0.0367, accuracy 99.11%

c. Learning rate increased

Parameters	
CV1: 6 filter, each (5,5), Activation: relu, padding: same	epoch = 10
Max pool filter:(2,2)	batch = 256
CV2: 16 filter, each (5,5), Activation: relu, padding: same	
Max pool filter:(2,2)	
FC1: 120	
FC2: 84	
FC3: 10	
Activation:Softmax	
lr = 0.01	
beta = 0.9	
beta 2 = 0.999	
epsilon = 1e -08	

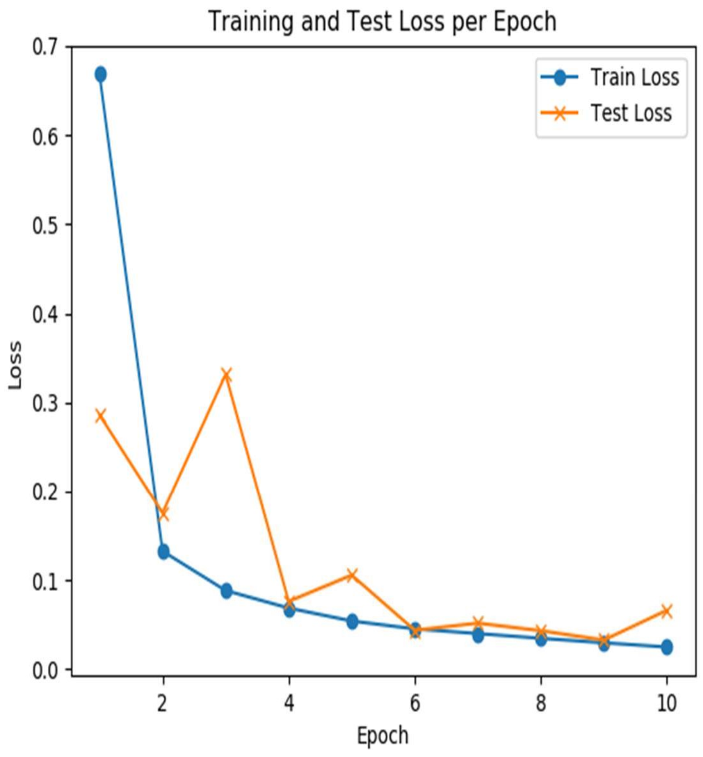


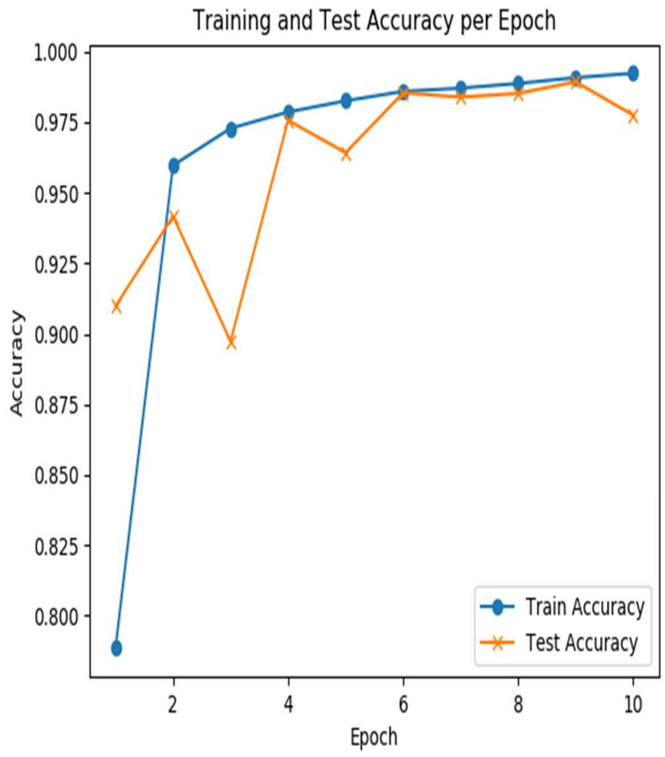


Accuracy
Training took 737.4942388534546 seconds.
Loss: 0.0179 - Acc: 0.9946 - Val_loss: 0.0372 - Val_acc: 0.9879
Test loss 0.0372, accuracy 98.79%

d. Batch Size increased

Parameters	
CV1: 6 filter, each (5,5), Activation: relu, padding: same	epoch = 10
Max pool filter:(2,2)	batch = 512
CV2: 16 filter, each (5,5), Activation: relu, padding: same	
Max pool filter:(2,2)	
FC1: 120	
FC2: 84	
FC3: 10	
Activation:Softmax	
lr = 0.01	
beta = 0.9	
beta 2 = 0.999	
epsilon = 1e -08	

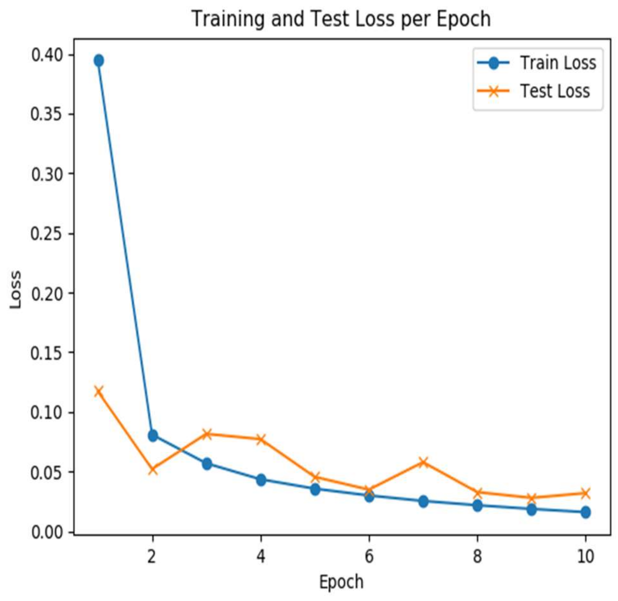


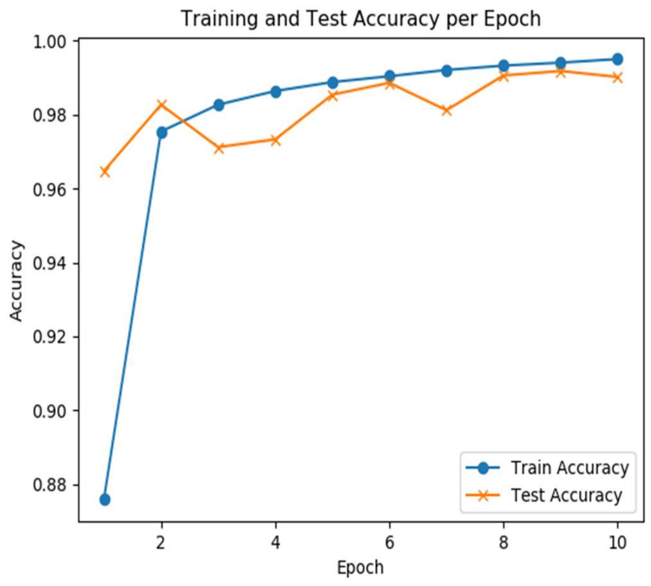


Accuracy
Training took 662.1190936565399 seconds.
Loss: 0.0250 - Acc: 0.9923 - Val_loss: 0.0663 - Val_acc: 0.9775
Test loss 0.0663, accuracy 97.75%

e. Filter initialization to 'normal'

Parameters	
CV1: 6 filter, each (5,5), Activation: relu, padding: same, Filter_init = Normal	epoch = 10
Max pool filter:(2,2)	batch = 256
CV2: 16 filter, each (5,5), Activation: relu, padding: same, Filter_init = Normal	
Max pool filter:(2,2)	
FC1: 120	
FC2: 84	
FC3: 10	
Activation:Softmax	
lr = 0.01	
beta = 0.9	
beta 2 = 0.999	
epsilon = 1e -08	
Filter_init = Normal	

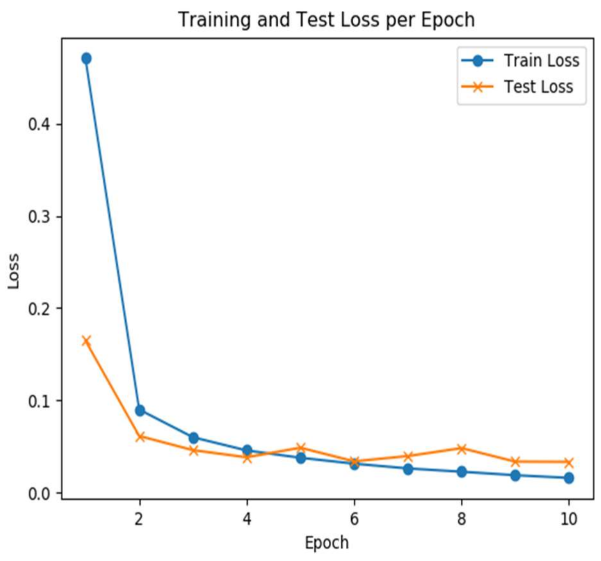


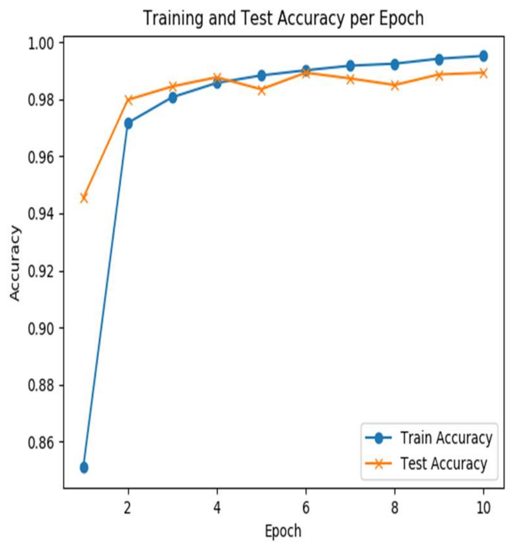


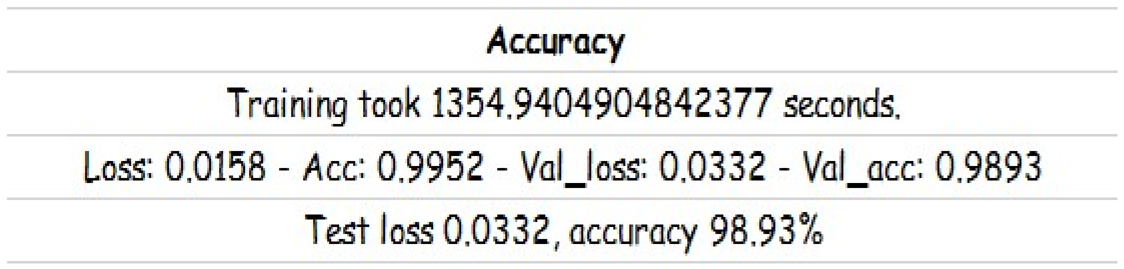
Accuracy
Training took 565.7049510478973 seconds.
Loss: 0.0160 - Acc: 0.9950 - Val_loss: 0.0319 - Val_acc: 0.9902
Test loss 0.0319, accuracy 99.02%

f. Filter Size decreased

Parameters	
CV1: 6 filter, each (3,3), Activation: relu, padding: same	epoch = 10
Max pool filter:(2,2)	batch = 256
CV2: 16 filter, each (3,3), Activation: relu, padding: same	
Max pool filter:(2,2)	
FC1: 120	
FC2: 84	
FC3: 10	
Activation:Softmax	
lr = 0.001	
beta = 0.9	
beta 2 = 0.999	
epsilon = 1e -08	







Accuracy
Training took 1354.9404904842377 seconds.
Loss: 0.0158 - Acc: 0.9952 - Val_loss: 0.0332 - Val_acc: 0.9893
Test loss 0.0332, accuracy 98.93%

1.4Discussion for b)

1. We observe that when epoch is increased when 10 to 20 both train and test accuracy increased but also time increased. This is intuitively correct as when we are using more epochs we are observing our data for a longer cycle and see how it is performing. And over here the CNN architecture works better with 20 epochs.

2. We see that when we increase the learning rate by 10 times. The training accuracy rises but there isn't significant increase in test accuracy. Also, the reduction in time is not much. This shows that the faster our network was taking step we were able to find better parameters that optimized the loss function.

3. When we increased the batch size from 256 to 512, we observe that the training accuracy increases but test accuracy decreases and time for training also decreases. This happens because at a point of time in an iteration we are considering but overall time to go through the whole dataset decreases.

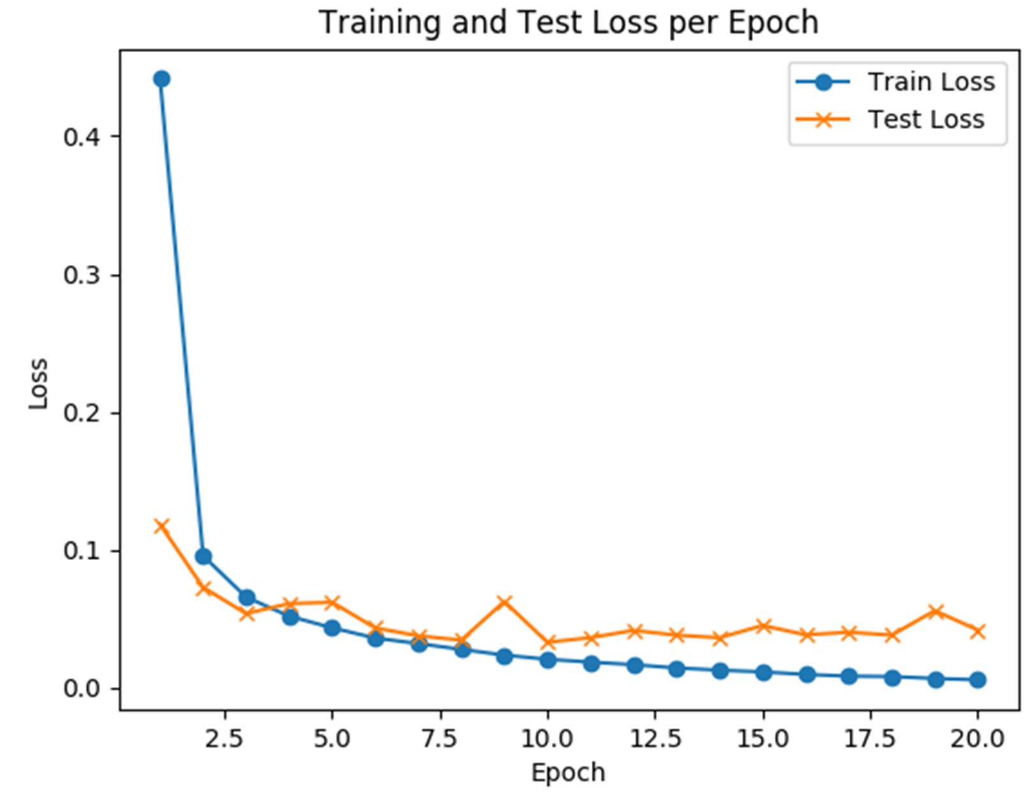
4. When we change the filter initialization to normal distribution, we get better training and test accuracy and also the time decreases. Thus, the normal distribution gives a better initialization which is wide spread and the makes the network more robust to change. Thus, the accuracy at test set also decreases.

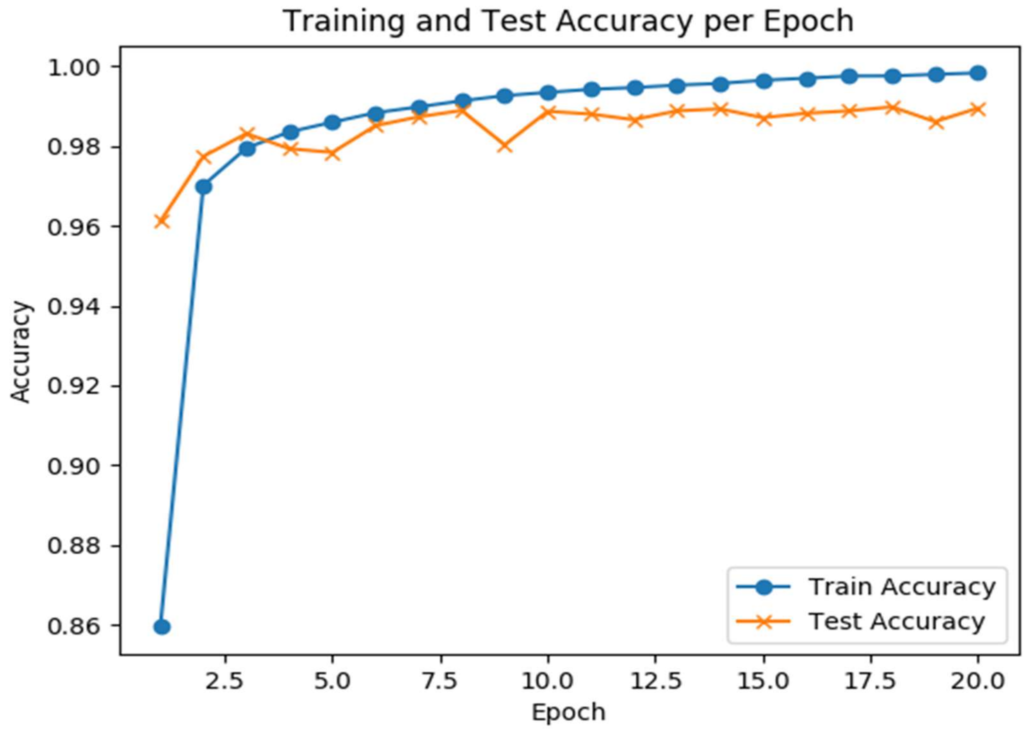
5. As we decrease the filter size from (5,5) in convolutional layer to (3,3), we observe that the time taken is more which is obvious because now we are considering every small detail as compared to earlier model. The training data accuracy increase but there isn't significant increase in test accuracy.

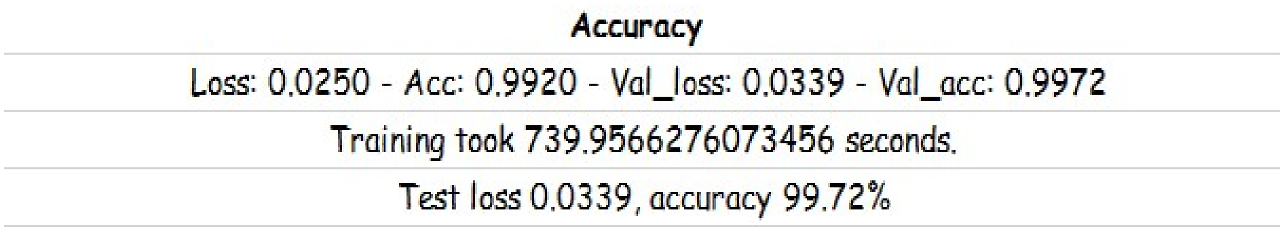
b) **Best network**

The best parameter which was able to achieve highest accuracy where by:

- CV1: 6 filter, each (3,3), Activation: ReLU, padding: same
- Max pool filter:(2,2)
- CV2: 16 filter, each (3,3), Activation: ReLU, padding: same
- Max pool filter:(2,2)
- FC1: 120
- FC2: 84
- FC3: 10
- Activation: SoftMax
- lr = 0.001
- beta = 0.9
- beta 2 = 0.999
- epsilon = 1e -08
- Increasing epoch to 20
- Increasing learning rate to 0.01
- Batch size to 256


Training and Test Loss per Epoch
Train Loss
Test Loss
Loss
Epoch
0.0
0.1
0.2
0.3
0.4
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Training and Test Accuracy per Epoch
Train Accuracy
Test Accuracy
Accuracy
Epoch
0.86
0.88
0.90
0.92
0.94
0.96
0.98
1.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Accuracy
Loss: 0.0250 - Acc: 0.9920 - Val_loss: 0.0339 - Val_acc: 0.9972
Training took 739.9566276073456 seconds.
Test loss 0.0339, accuracy 99.72%

C. Improve Le-Net5 for MNIST data

Couple of techniques were applied to understand the CNN architecture. The following section will discuss every single technique with the theory behind it, advantages and disadvantages of using it. Also, testing the same method and calculating the accuracy and loss.

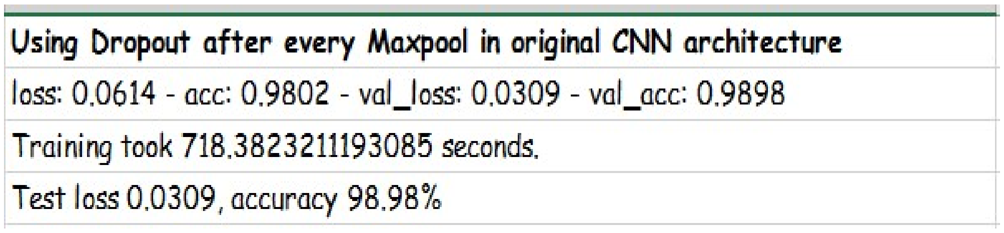
1) **Using Dropout**:

It refers to ignoring units/ neurons during the certain training phase and it is done randomly. It basically means to not include those units that training iteration. The technique of regularization is performed because it reduces over fitting by adding the penalty to the loss function. In the same way, dropout regularizer the neural network which helps in reducing interdependent learning amongst the neurons.

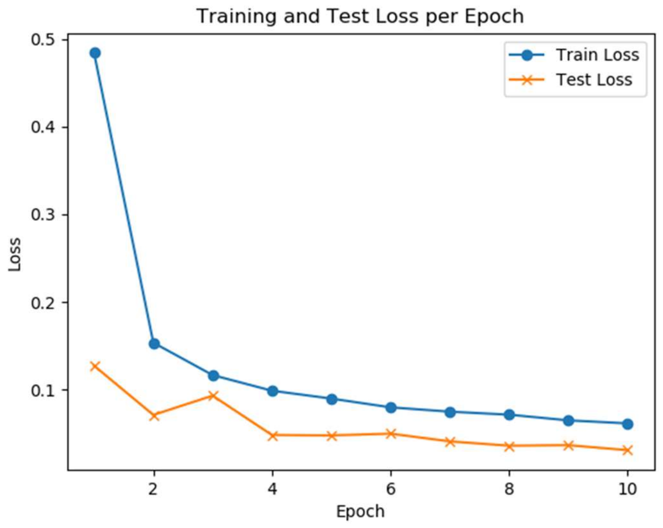
The advantages of using dropout is that forces the network to learn features with different random subsets of other neurons. It makes the model more to robust to change.

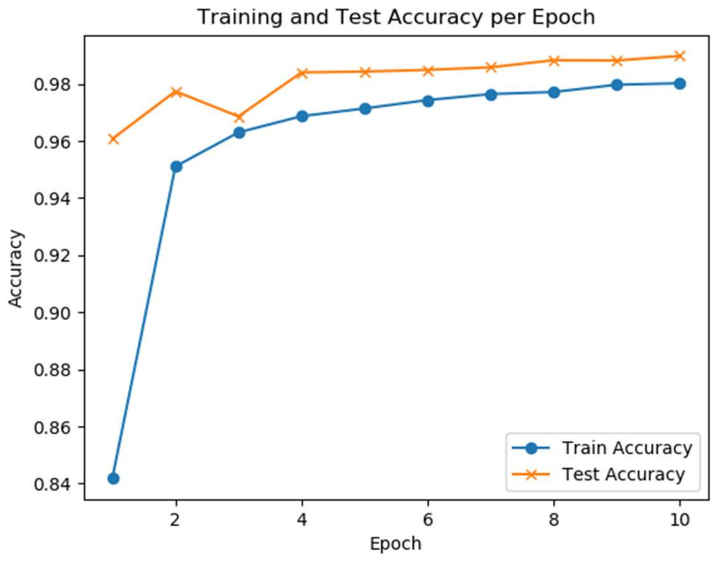
The disadvantages of using dropout is that it doubles the number of iterations required to converge.

The following research paper [4] proves that using dropout is a useful technique. A statement from the paper “We also empirically show that the effect of convolutional dropout is not trivial, despite the dramatically reduced possibility of over-fitting due to the convolutional architecture.”



Using Dropout after every Maxpool in original CNN architecture
loss: 0.0614 - acc: 0.9802 - val_loss: 0.0309 - val_acc: 0.9898
Training took 718.3823211193085 seconds.
Test loss 0.0309, accuracy 98.98%





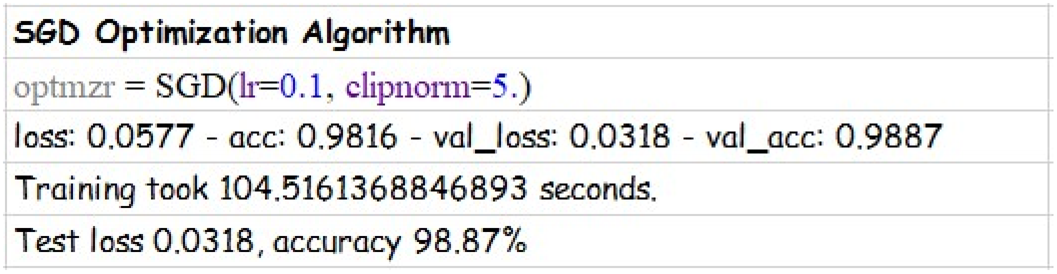
2. **Optimization Algorithm: Stochastic Gradient Descent**

We initially applied the Adam Algorithm, we now implement Stochastic Gradient Descent. It is the basic optimization algorithm. It divides the datasets into small batches of examples, choose samples of different classes then computes the gradient using a single batch making an update. Then moves to the next batch of examples. The key parameters which we are set in SGD are size of mini batch, learning rate and number of iteration per mini batch.

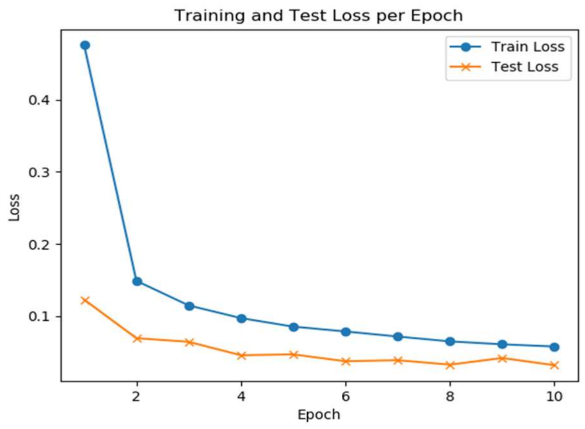
The advantage of Stochastic Gradient Descent is the speed that can also be seen in the result that we get.

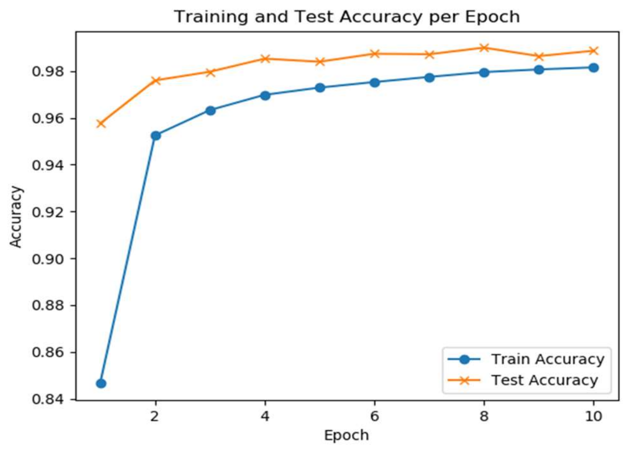
The disadvantage of using it usually detect the local optima and misses the global optimum. When the parameters are in different scales, a low learning rate will make the learning slow and large learning rate might lead to oscillations.

The following research paper[5] shows that using optimization algorithm like AdaDelta, AdaGrad and RMS Prop might give superior results than SGD.



SGD Optimization Algorithm
optmzr = SGD(lr=0.1, clipnorm=5.)
loss: 0.0577 - acc: 0.9816 - val_loss: 0.0318 - val_acc: 0.9887
Training took 104.5161368846893 seconds.
Test loss 0.0318, accuracy 98.87%



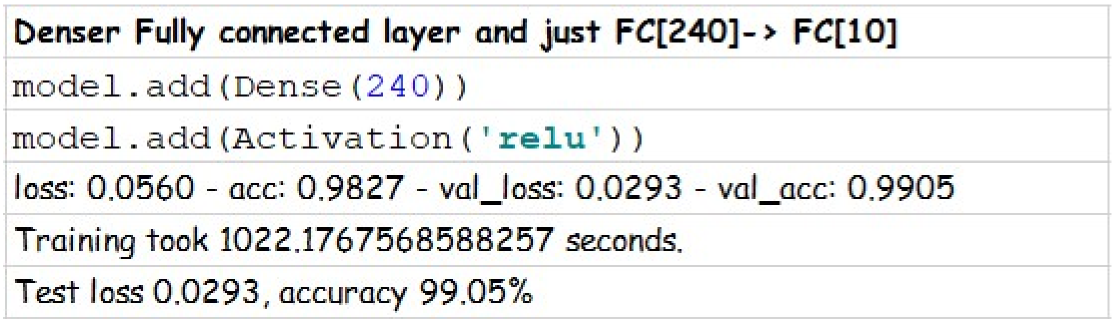


3. **Using denser fully connected layer**

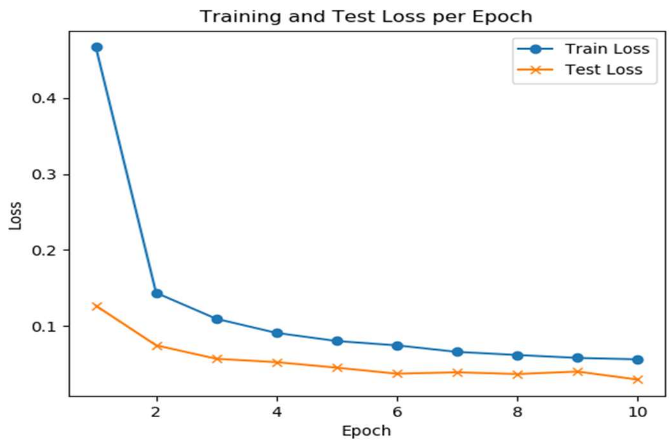
We saw the use of fully connected and how it works in the above layer. Our aim here is to study how the model works if we increase the depth of fully connected layer. The fully connected layer helps the model to understand combination of features rather than just the features it has seen before. In case like this, something more sophisticated than SoftMax layer is needed and fully connected layer comes into action.

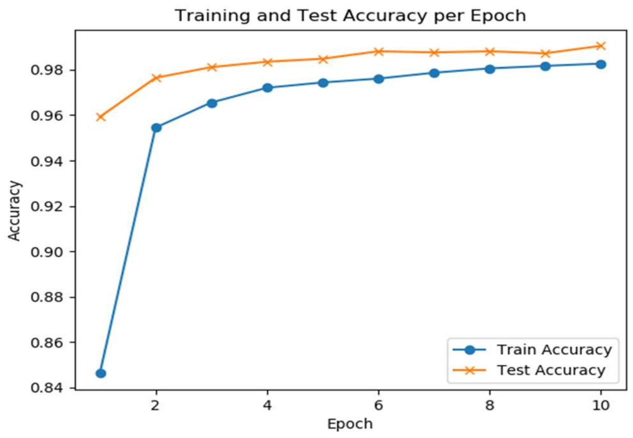
It was carried out to answer the Quora question in a better way, "What is the purpose of using more than 1 fully-connected layer in a convolutional neural network? [6]

We tested what if we use just one fully connected layer denser than the original architecture what results does it give. We were able to find that denser fully connected layer was able to perform better. But still not able to reach greater accuracy than if we used denser layer and couple of layers.



Denser Fully connected layer and just FC[240]-> FC[10]
model.add(Dense(240))
model.add(Activation('relu'))
loss: 0.0560 - acc: 0.9827 - val_loss: 0.0293 - val_acc: 0.9905
Training took 1022.1767568588257 seconds.
Test loss 0.0293, accuracy 99.05%



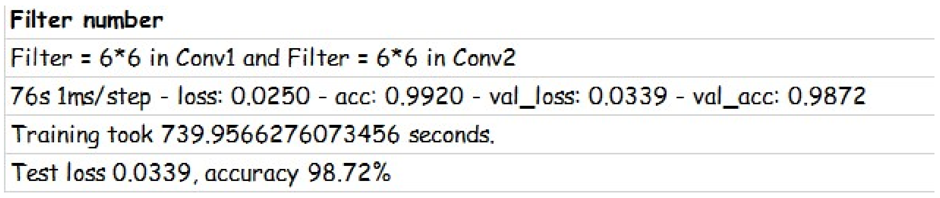


4. **Changing filter number**

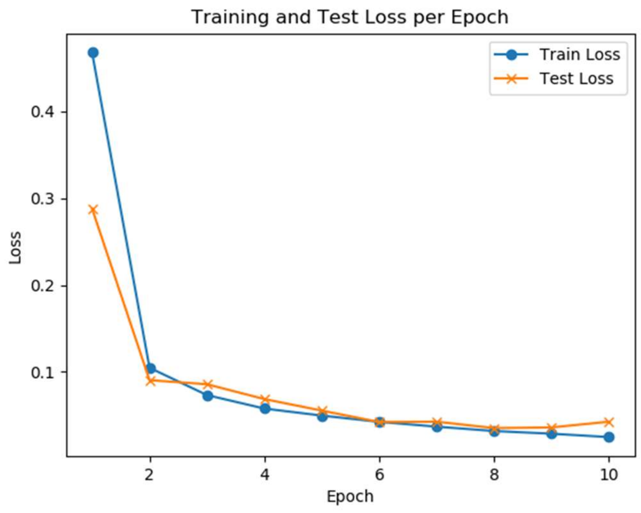
The filters signify the feature detectors. In terms of signal processing, filter acts to remove a frequency that could be something like high pass filter. It can be considered as a feature map. Feature maps allows to learn explanatory factors within the image so more the number of filter more that we learn. It also depends on the domain of the problem, task characteristics and data.

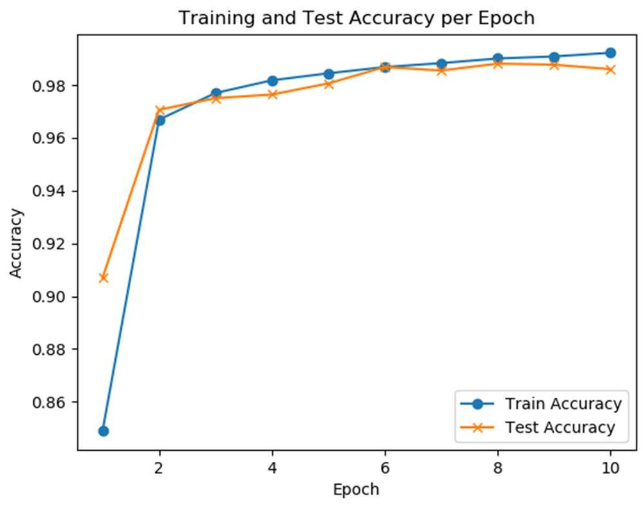
We tried using lesser filter that is 6 filters on both the convolutional layer rather than 6 and 16 filters respectively.

This experiment help to answer this question better How do we choose the filters for the convolutional layer of a Convolution Neural Network? [6]



Filter number
Filter = 6*6 in Conv1 and Filter = 6*6 in Conv2
76s 1ms/step - loss: 0.0250 - acc: 0.9920 - val_loss: 0.0339 - val_acc: 0.9872
Training took 739.9566276073456 seconds.
Test loss 0.0339, accuracy 98.72%





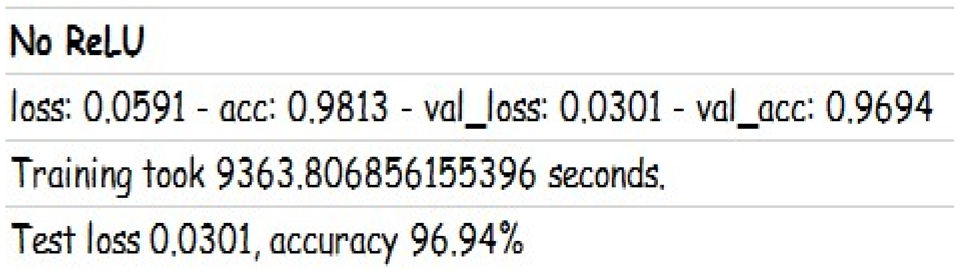
5.Model without ReLU

The rectifier function is an activation function *f(x) = Max(0, x)*.

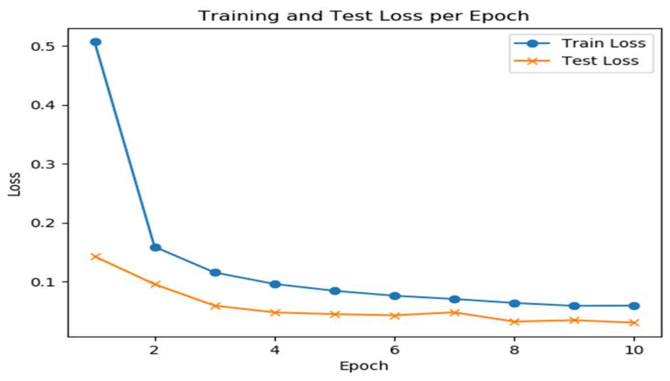
The main reason that it is used is because of how efficiently it can be computed compared to more conventional activation functions like the sigmoid and hyperbolic tangent, without making a significant difference to generalization accuracy.

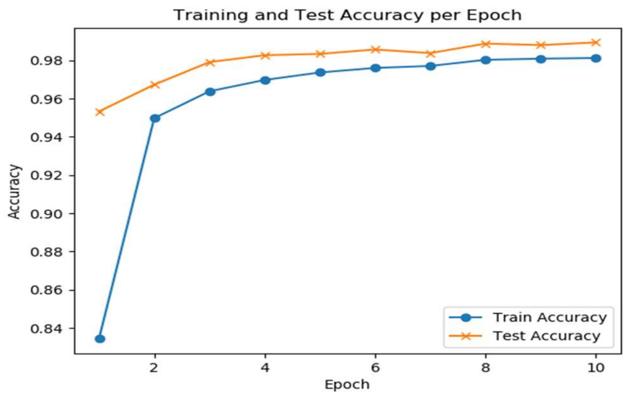
The rectifier activation function is used instead of a linear activation function to add non- linearity to the network, otherwise the network would only be able to compute a linear function. To check how the network performs and what all parameters get transformed after removing all ReLU layers from the model.

The research paper encompasses the advantages and disadvantages of using ReLU. Stating "ReLU is computational efficient by just outputing zero for negative inputs". Also mentioned that ReLU does simply restrains the negative value to hard-zero, which provides sparsity but results negative missing.



```
No ReLU
loss: 0.0591 - acc: 0.9813 - val_loss: 0.0301 - val_acc: 0.9694
Training took 9363.806856155396 seconds.
Test loss 0.0301, accuracy 96.94%
```





1.4Discussion for c)

1. As comparing to the LeNet 5 model, when dropout layers we added we say an increase from 98.67 -> 98.98 showing empirically that the effect of convolutional dropout is not trivial and it dramatically reduces possibility of over-fitting due to the convolutional architecture.
2. When SGD is used as the optimization algorithm. We observed that loss increased in case of training data and accuracy decreased. We can say that SGD works as a good optimization algorithm in case of Le-Net5 considering that it does not worsen the network. But incease test accuracy by a little.
3. Using densely connected layer, increased the accuracy on test data from 98.67 -> 99.05 which proves that using densely connected layer in Le-Net5 would help to increase the accuracy.
4. We observe that using less filter in case of Le-Net5 decreases the training accuracy signifying that the model was not able to compute a lesser feature making the model not a good fit. Thus, the larger the number of filters the better but not too big that the computation time increases manifold.
5. When model was trained without ReLU, it was observed that the testing and validation accuracy decreased. Thus, making the model not robust enough to be able to identify complex features. Also, the time for training increased signifying that convergence took more time than usual. Thus, adding non- linearity to the model is a must in order to make it robust.

2.Problem 2

2.1 Abstract and Motivation

Saak transform offers a new angle to representation problem. It provides powerful spatial spectral Saak features for pattern recognition. It is focused on the lossless Saak transform in the model that was studied. F-Test score to select Saak coefficients of higher discriminant power. The transform can be applied to extract discriminant feature from a huge number of Saak coefficients to high dimensional random vectors if they composed to lower dimensional sub-vectors hierarchically. They can be applied for image synthesis and generation.

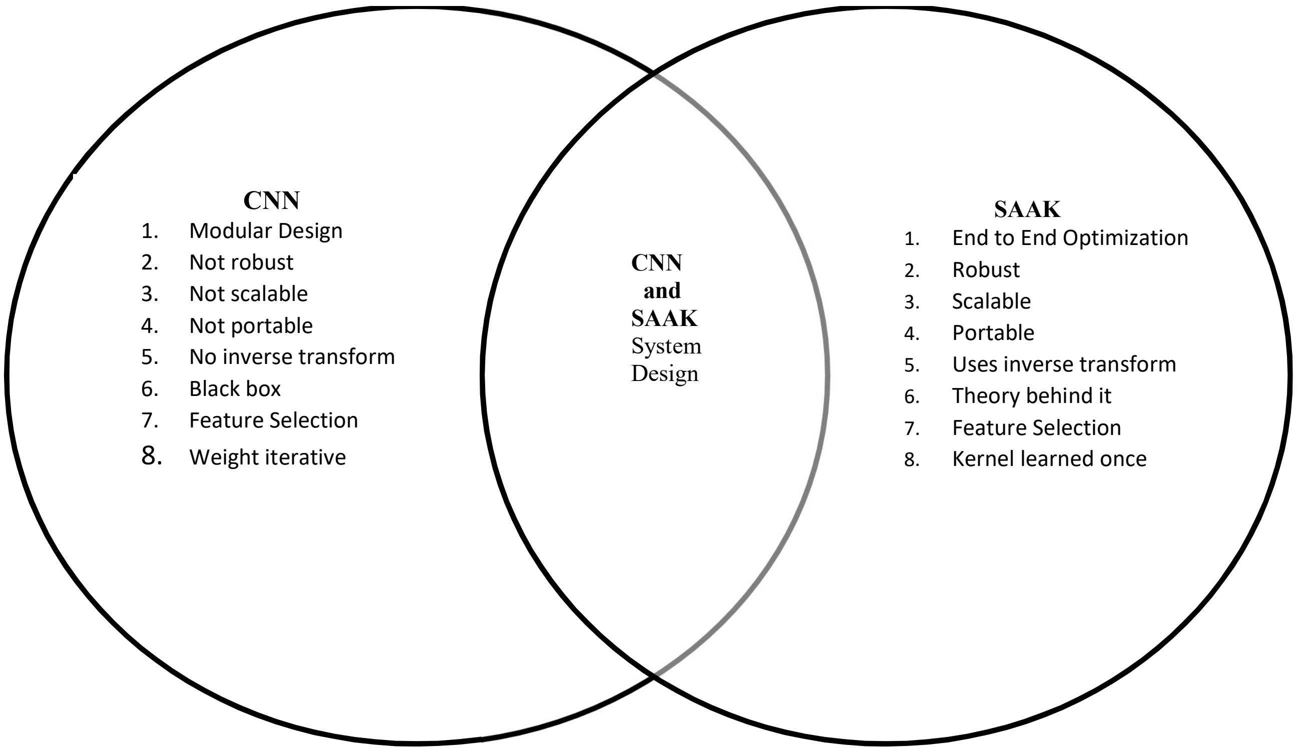
In this problem, we compare the difference between SAAK and Convolutional Neural Network and also the commonalities between both the approaches. We apply the SAAK transform to MNIST dataset, we mainly focus on Saak Coefficients Transform, Saak coefficients selection and dimension reduction and then utilize of the same class label to collect features of training samples and compare the performance of SVM and RF classifiers with three features dimensions. At the end, we compare classification errors cases arising from CNN and Saak transform.

2.2Approach and Procedures

There are three parts in this section. The comparison of CNN and SAAK along with similarities is done in A. Application of SAAK transform on MNIST dataset is done in B and Error Analysis is done in C.

2.2.1Comparison between SAAK Transform and CNN

A. Comparison between SAAK transform and CNN:



Differences:

1. End to end Optimization versus Modular Design

CNN: The structure is based on an architecture which is end to end. The recognition is performed in one architecture. It is not divided into sub parts. So, when an input image is passed the neural network is able to find the features defining an image and at the end recognizing it to be among ones from the train data. focus is to optimize the cost function iteratively. The filter weights are updated at every iteration and changed in a way that they learn from their mistakes in this iteration and change it to perform better. This is done using backpropagation.

SAAK: To perform recognition, the architecture is comprised of two separate tasks: 1. Feature extraction 2. Classification. It is one pass feedforward network. It is not iteratively done but the transform kernels are defined which are derived from second order statistics of input random vector. Then selection of saak coefficient of higher discriminant power is done to form a feature vector for pattern recognition.

2. Robust

CNN: They are not robust against perturbation. It has been seen that CNN are surprisingly robust to many internal perturbations in higher convolutional layers but the bottom convolutional layers re more fragile. Some deep CNN make incorrect predictions because of some perturbation. In that case some remedy algorithm must be implemented.
SAAK: They are expected to be more robust. It won't have affected much because the PCA is performed on the last stage coefficient and they only take into consideration the features with the highest variance so the overall system won't get affected to the perturbation caused.

3. Scalability against class number

CNN: If the object class number is changed the decision subnet is affected and therefore all the weights of the network must be re trained.
SAAK: They are less sensitive to the variation given that the covariance matrix does not change much. As the name suggest covariance matrix denotes the variation in a two-random vector so it won't be affected if class number is changed. If this happens then the same network can be used to generate different Saak features for new object classes as well.

4. Portability among different datasets

CNN: The architecture of CNN is developed in such a way that they are designed to perform a specific task and give best cost, flexibility and generalization under specific setting. Thus, they are not portable. The architecture learns the weights to recognize the test data accurately. The weights are iteratively learned in order to optimize cost function. One architecture and its weights can't be used for other kind of datasets.
SAAK: They are compatible to wide range of variety as they focus on each module of the recognition process separately. They are more algorithm focused than structure focused. Thus, they can be used to a wide range of datasets.

5. Inverse Transform

CNN: They only inverse type of transform is RECOS presented in [2]. While CNN does comprise of Generative and Discriminative Network and Autoencoders. They allow generative network to synthesize image of remarkable quality. The generator inverts a predefining embedding operator. Defining image generator as a solution to the inverse problem provides a mathematical framework which is closer to probabilistic model but not quite there.

SAAK: They consist of inverse transform which are easily defined and used. It is more focused on the traditional signal analysis and synthesis framework. The cascade of forward and inverse multi stage Saak transform result in identity operator showing that they are basically opposite of each other and can be used in applications like presenting a low-resolution image and transform it to high resolution image.

6. Theoretical Foundation

CNN: Despite their remarkable performance there is no underlying fundamentals to validate them. Unfortunately, as the network grows it becomes even harder to mathematically tract their behavior.
SAAK: It is built on linear algebra and statistics, thus presenting a more reliable framework to validate how and why any particular observation is noticed.

7. Filter Weight Determination

CNN: The weights are determined by the data used for training and their corresponding labels. As described in the 1) that filter weights are updated iteratively using backpropagation. The optimal weight is the one which maximizes the cost function. Usually the number of iterations is huge.
SAAK: It on the other hand uses fully automatic method to determine the transform kernel. It is built of second order statistics of the input data which selects orthonormal eigenvectors of the covariance matrix as transform kernels. The method is known as KLT. It is one pass feedforward process from leaf to root. In case of higher dimensional vector, it is decomposed into multiple lower dimensional sub vectors and KLT is performed recursively in hierarchical manner. It neither needs backpropagation nor labels so characterization of supervised and non-supervised learning not applicable to SAAK.

8. Feature Selection

CNN: The features in it can be learned automatically for the data itself instead of handcrafted feature. It goes from local to abstract level. But the feature selection is still unexplored area due to the black box nature.
SAAK: In order to obtain saak features we need data labels. The multi stage saak coefficient of samples from class computed to build feature vector of that class. We chose those features that have higher discriminant power among object class to reduce the feature dimension.

Similarities:

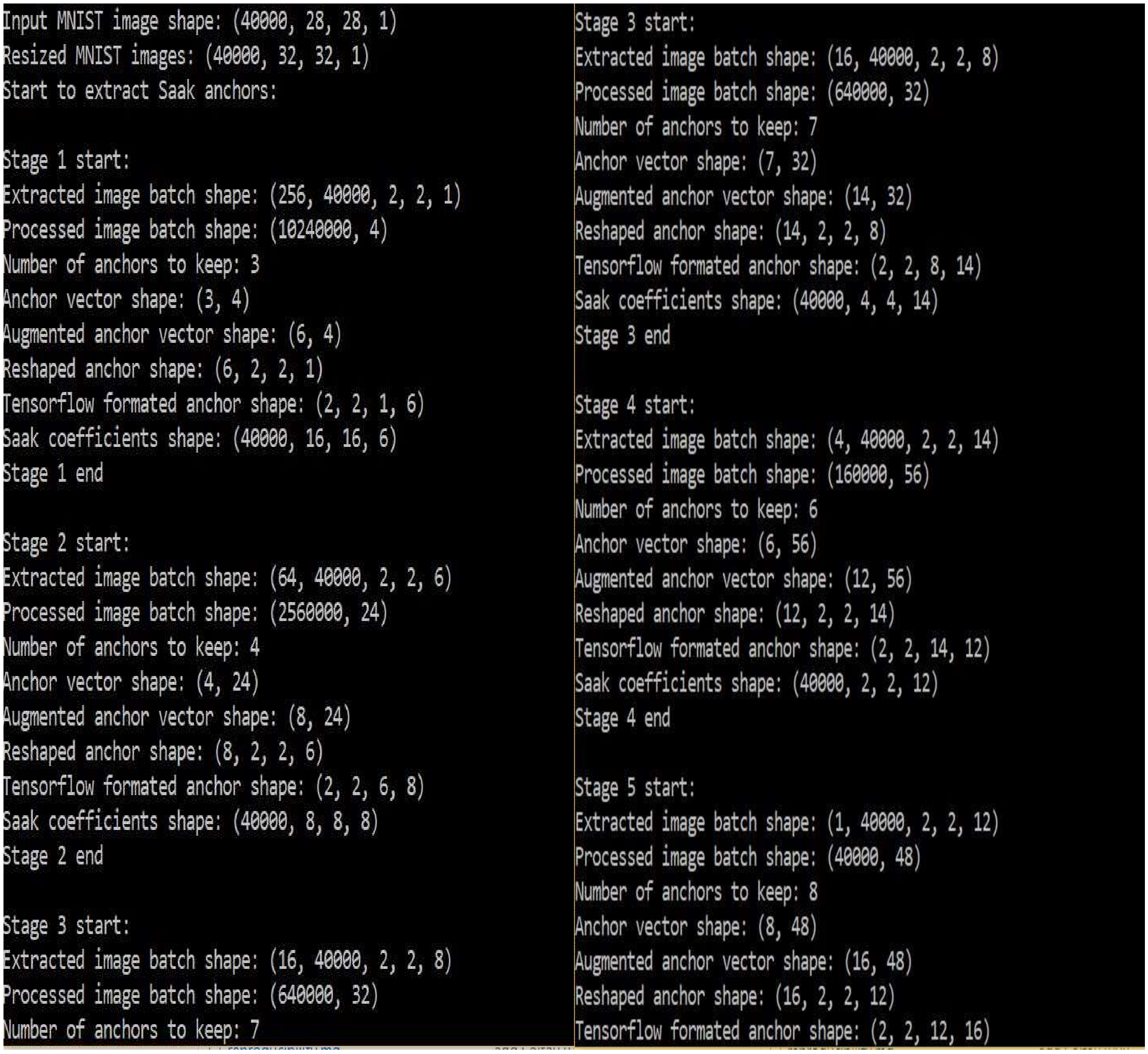
1. System Design

They both perform convolution or transform operations which is followed by RELU operation before it is passed onto to the next layer/stage to introduce non- linearity to the system in CNN. In SAAK if two or more transforms are cascaded directly a sign confusion problem occurs. To resolve it, RELU is used.

B. Application of SAAK to MNIST dataset

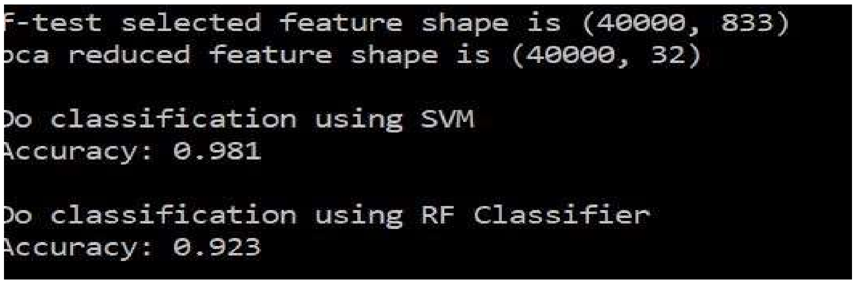
The following procedure was applied to apply SAAK transform on MNIST data is mentioned in homework question.

2.3Experimental Results for B



```
Input MNIST image shape: (40000, 28, 28, 1)
Resized MNIST images: (40000, 32, 32, 1)
Start to extract Saak anchors:

Stage 1 start:
Extracted image batch shape: (256, 40000, 2, 2, 1)
Processed image batch shape: (10240000, 4)
Number of anchors to keep: 3
Anchor vector shape: (3, 4)
Augmented anchor vector shape: (6, 4)
Reshaped anchor shape: (6, 2, 2, 1)
Tensorflow formated anchor shape: (2, 2, 1, 6)
Saak coefficients shape: (40000, 16, 16, 6)
Stage 1 end

Stage 2 start:
Extracted image batch shape: (64, 40000, 2, 2, 6)
Processed image batch shape: (2560000, 24)
Number of anchors to keep: 4
Anchor vector shape: (4, 24)
Augmented anchor vector shape: (8, 24)
Reshaped anchor shape: (8, 2, 2, 6)
Tensorflow formated anchor shape: (2, 2, 6, 8)
Saak coefficients shape: (40000, 8, 8, 8)
Stage 2 end

Stage 3 start:
Extracted image batch shape: (16, 40000, 2, 2, 8)
Processed image batch shape: (640000, 32)
Number of anchors to keep: 7
```

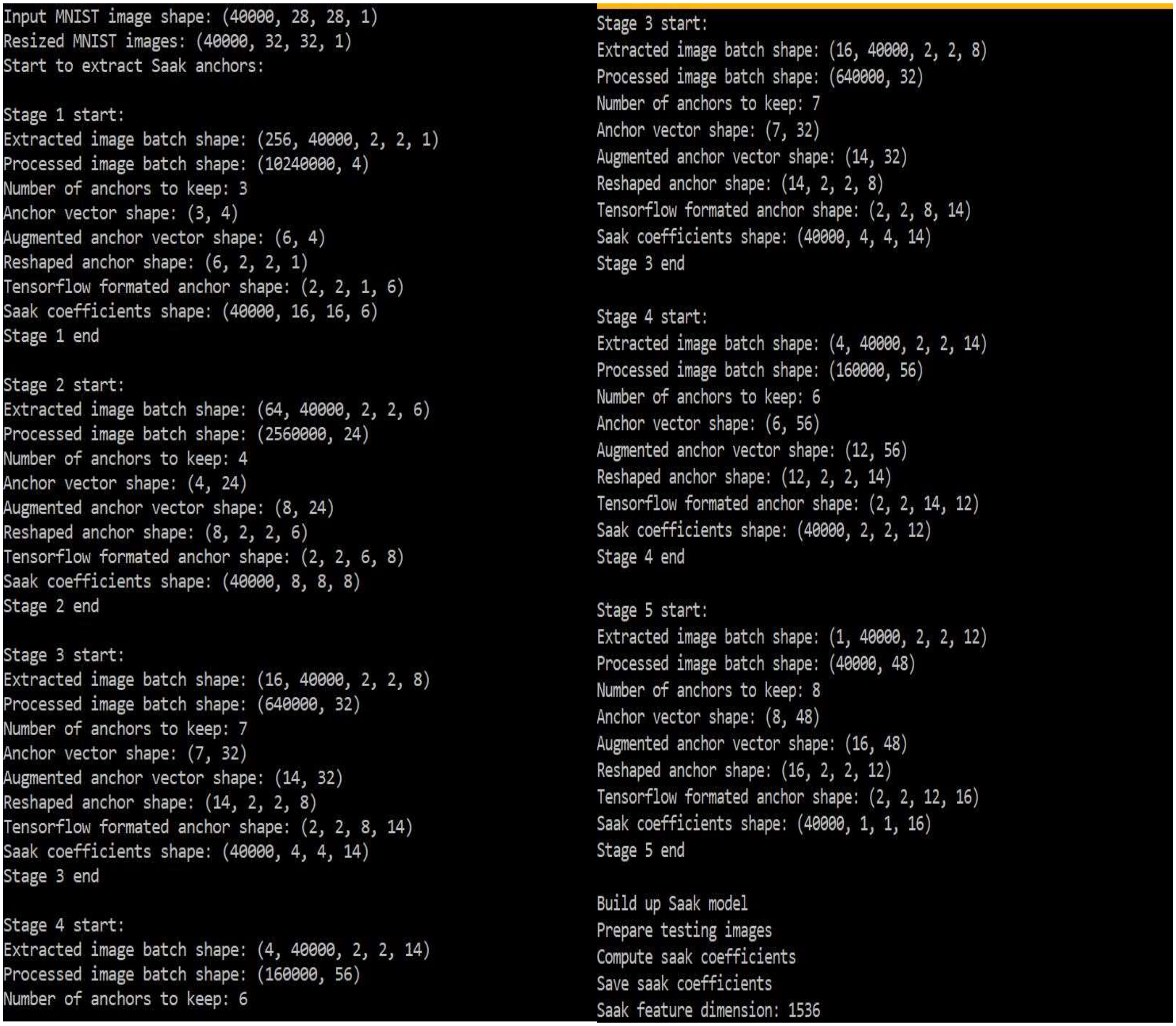
```
Stage 3 start:
Extracted image batch shape: (16, 40000, 2, 2, 8)
Processed image batch shape: (640000, 32)
Number of anchors to keep: 7
Anchor vector shape: (7, 32)
Augmented anchor vector shape: (14, 32)
Reshaped anchor shape: (14, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 14)
Saak coefficients shape: (40000, 4, 4, 14)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 40000, 2, 2, 14)
Processed image batch shape: (160000, 56)
Number of anchors to keep: 6
Anchor vector shape: (6, 56)
Augmented anchor vector shape: (12, 56)
Reshaped anchor shape: (12, 2, 2, 14)
Tensorflow formated anchor shape: (2, 2, 14, 12)
Saak coefficients shape: (40000, 2, 2, 12)
Stage 4 end

Stage 5 start:
Extracted image batch shape: (1, 40000, 2, 2, 12)
Processed image batch shape: (40000, 48)
Number of anchors to keep: 8
Anchor vector shape: (8, 48)
Augmented anchor vector shape: (16, 48)
Reshaped anchor shape: (16, 2, 2, 12)
Tensorflow formated anchor shape: (2, 2, 12, 16)
```



```
f-test selected feature shape is (40000, 833)
pca reduced feature shape is (40000, 32)

Do classification using SVM
Accuracy: 0.981

Do classification using RF Classifier
Accuracy: 0.923
```

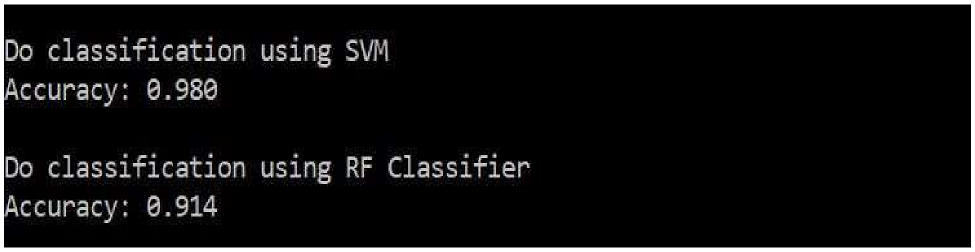
`

Figure 3 Classification when 32 feature dimensions when training data 40,000



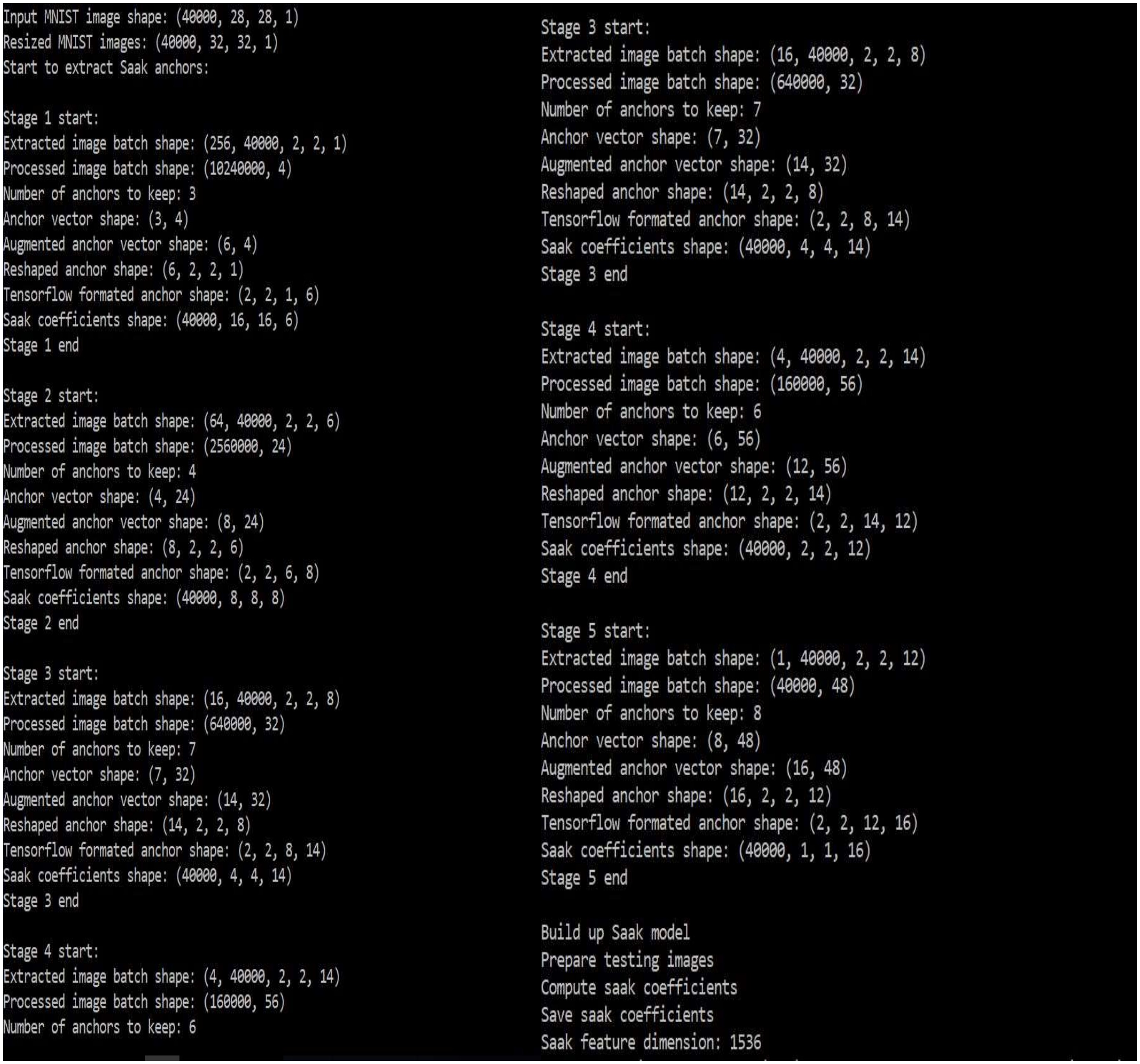
```
Input MNIST image shape: (40000, 28, 28, 1)
Resized MNIST images: (40000, 32, 32, 1)
Start to extract Saak anchors:

Stage 1 start:
Extracted image batch shape: (256, 40000, 2, 2, 1)
Processed image batch shape: (10240000, 4)
Number of anchors to keep: 3
Anchor vector shape: (3, 4)
Augmented anchor vector shape: (6, 4)
Reshaped anchor shape: (6, 2, 2, 1)
Tensorflow formated anchor shape: (2, 2, 1, 6)
Saak coefficients shape: (40000, 16, 16, 6)
Stage 1 end

Stage 2 start:
Extracted image batch shape: (64, 40000, 2, 2, 6)
Processed image batch shape: (2560000, 24)
Number of anchors to keep: 4
Anchor vector shape: (4, 24)
Augmented anchor vector shape: (8, 24)
Reshaped anchor shape: (8, 2, 2, 6)
Tensorflow formated anchor shape: (2, 2, 6, 8)
Saak coefficients shape: (40000, 8, 8, 8)
Stage 2 end

Stage 3 start:
Extracted image batch shape: (16, 40000, 2, 2, 8)
Processed image batch shape: (640000, 32)
Number of anchors to keep: 7
Anchor vector shape: (7, 32)
Augmented anchor vector shape: (14, 32)
Reshaped anchor shape: (14, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 14)
Saak coefficients shape: (40000, 4, 4, 14)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 40000, 2, 2, 14)
Processed image batch shape: (160000, 56)
Number of anchors to keep: 6
```

```
Stage 3 start:
Extracted image batch shape: (16, 40000, 2, 2, 8)
Processed image batch shape: (640000, 32)
Number of anchors to keep: 7
Anchor vector shape: (7, 32)
Augmented anchor vector shape: (14, 32)
Reshaped anchor shape: (14, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 14)
Saak coefficients shape: (40000, 4, 4, 14)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 40000, 2, 2, 14)
Processed image batch shape: (160000, 56)
Number of anchors to keep: 6
Anchor vector shape: (6, 56)
Augmented anchor vector shape: (12, 56)
Reshaped anchor shape: (12, 2, 2, 14)
Tensorflow formated anchor shape: (2, 2, 14, 12)
Saak coefficients shape: (40000, 2, 2, 12)
Stage 4 end

Stage 5 start:
Extracted image batch shape: (1, 40000, 2, 2, 12)
Processed image batch shape: (40000, 48)
Number of anchors to keep: 8
Anchor vector shape: (8, 48)
Augmented anchor vector shape: (16, 48)
Reshaped anchor shape: (16, 2, 2, 12)
Tensorflow formated anchor shape: (2, 2, 12, 16)
Saak coefficients shape: (40000, 1, 1, 16)
Stage 5 end

Build up Saak model
Prepare testing images
Compute saak coefficients
Save saak coefficients
Saak feature dimension: 1536
```



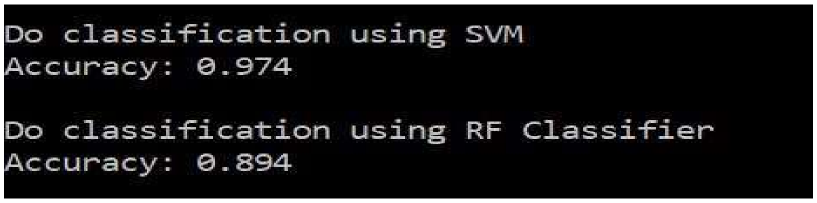
```
Do classification using SVM
Accuracy: 0.980

Do classification using RF Classifier
Accuracy: 0.914
```

Figure 4 Classification when 64 feature dimensions when training data 40,000



```
Input MNIST image shape: (40000, 28, 28, 1)
Resized MNIST images: (40000, 32, 32, 1)
Start to extract Saak anchors:

Stage 1 start:
Extracted image batch shape: (256, 40000, 2, 2, 1)
Processed image batch shape: (10240000, 4)
Number of anchors to keep: 3
Anchor vector shape: (3, 4)
Augmented anchor vector shape: (6, 4)
Reshaped anchor shape: (6, 2, 2, 1)
Tensorflow formated anchor shape: (2, 2, 1, 6)
Saak coefficients shape: (40000, 16, 16, 6)
Stage 1 end

Stage 2 start:
Extracted image batch shape: (64, 40000, 2, 2, 6)
Processed image batch shape: (2560000, 24)
Number of anchors to keep: 4
Anchor vector shape: (4, 24)
Augmented anchor vector shape: (8, 24)
Reshaped anchor shape: (8, 2, 2, 6)
Tensorflow formated anchor shape: (2, 2, 6, 8)
Saak coefficients shape: (40000, 8, 8, 8)
Stage 2 end

Stage 3 start:
Extracted image batch shape: (16, 40000, 2, 2, 8)
Processed image batch shape: (640000, 32)
Number of anchors to keep: 7
Anchor vector shape: (7, 32)
Augmented anchor vector shape: (14, 32)
Reshaped anchor shape: (14, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 14)
Saak coefficients shape: (40000, 4, 4, 14)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 40000, 2, 2, 14)
Processed image batch shape: (160000, 56)
Number of anchors to keep: 6
```

```
Stage 3 start:
Extracted image batch shape: (16, 40000, 2, 2, 8)
Processed image batch shape: (640000, 32)
Number of anchors to keep: 7
Anchor vector shape: (7, 32)
Augmented anchor vector shape: (14, 32)
Reshaped anchor shape: (14, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 14)
Saak coefficients shape: (40000, 4, 4, 14)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 40000, 2, 2, 14)
Processed image batch shape: (160000, 56)
Number of anchors to keep: 6
Anchor vector shape: (6, 56)
Augmented anchor vector shape: (12, 56)
Reshaped anchor shape: (12, 2, 2, 14)
Tensorflow formated anchor shape: (2, 2, 14, 12)
Saak coefficients shape: (40000, 2, 2, 12)
Stage 4 end

Stage 5 start:
Extracted image batch shape: (1, 40000, 2, 2, 12)
Processed image batch shape: (40000, 48)
Number of anchors to keep: 8
Anchor vector shape: (8, 48)
Augmented anchor vector shape: (16, 48)
Reshaped anchor shape: (16, 2, 2, 12)
Tensorflow formated anchor shape: (2, 2, 12, 16)
Saak coefficients shape: (40000, 1, 1, 16)
Stage 5 end

Build up Saak model
Prepare testing images
Compute saak coefficients
Save saak coefficients
Saak feature dimension: 1536
```


```
Do classification using SVM
Accuracy: 0.974

Do classification using RF Classifier
Accuracy: 0.894
```
Figure 5Classification when 128 feature dimensions when training data 40,000

Training data/ Dimension	SVM Accuracy	RF Classifier Accuracy
40,000 / 32	98.1	92.3
40,000 / 64	98.0	91.4
40,000 / 128	97.4	89.4
55,000/32	98.3	92.9
55,000/64	98.3	92.2
55,000/128	97.7	90.0

2.4Experimental Results for B

1. We observe that even when the training data is 2k less than the one used in training for CNN, we get accuracy comparable or higher in SAAK.
   For example, when we used 40,000 training data with reduced 32 dimensions, we achieve 98.1% accuracy. Similarly, when we used 60,000 training data with 512 batch sizes in Le-Net5 architecture, we achieve 97.75%.

2. A significant decrease in training time is observed in SAAK as compared to CNN.
   For example, when using 50,000 training data with reduced dimensions 64, it took 120 secs as comparable to when we use 6 filters in fully connected layer 1 and fully connected layer 2 other parameters same as Le-Net5 architecture which took 740 secs to train and both achieve comparable accuracy.

3. The optimal parameter determination is trial and error in case of CNN while that does not happen in case of SAAK. It is more stable system in that terms.
   For example, the procedure for using SAAK remains same while in case of CNN different parameters were tried and every parameter changes the frequency in different ways. Feature selection is a time-consuming process for CNN.

4.Small perturbation in CNN worsens the accuracy while that does not happen in case of SAAK. Thus, CNN is not robust. While, even features are increased or reduced, the accuracy does not drop that much.
For example, in case of CNN when dropout layer is removed the accuracy drops to a 96.94%
While in SAAK when features are reduced differently accuracy remains around 98%.

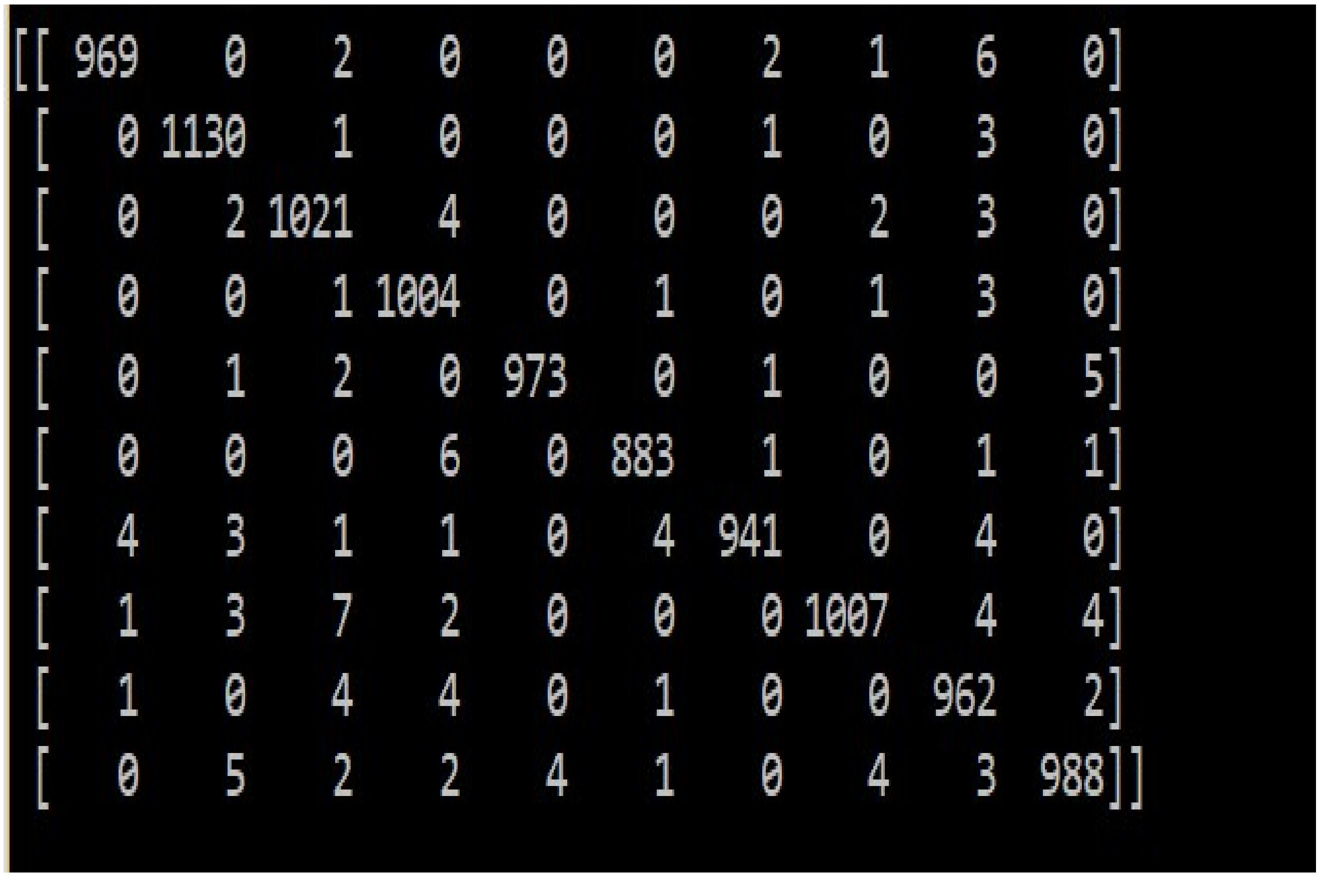
5.CNN can achieve higher accuracy with the most optimal features as compared to SAAK.

For example, CNN can achieve 99.72% as compared to 98.3%.

6. When random classifier is used, SAAK is not able to perform as well as with methods like SVM as compared to CNN.
   For example, when RF classifier is used, the accuracy achieved is 92.3% as compared to 97/98/99%.

C. Error Analysis

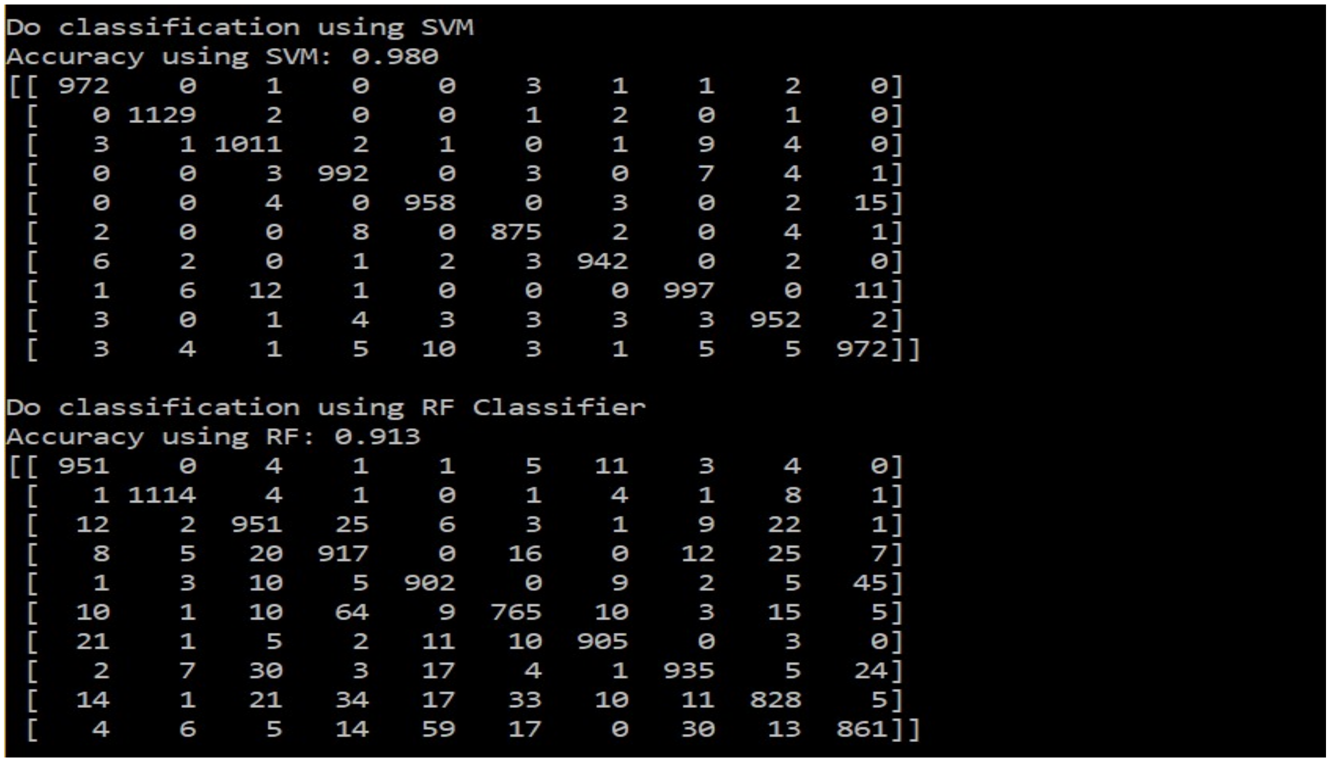
Model 1: Using CNN based Le-Net5 on MNIST data which gave 98.67% accuracy.



```
[[ 969    0    2    0    0    0    2    1    6    0]
 [    0 1130    1    0    0    0    1    0    3    0]
 [    0    2 1021    4    0    0    0    2    3    0]
 [    0    0    1 1004    0    1    0    1    3    0]
 [    0    1    2    0  973    0    1    0    0    5]
 [    0    0    0    6    0  883    1    0    1    1]
 [    4    3    1    1    0    4  941    0    4    0]
 [    1    3    7    2    0    0    0 1007    4    4]
 [    1    0    4    4    0    1    0    0  962    2]
 [    0    5    2    2    4    1    0    4    3  988]]
```

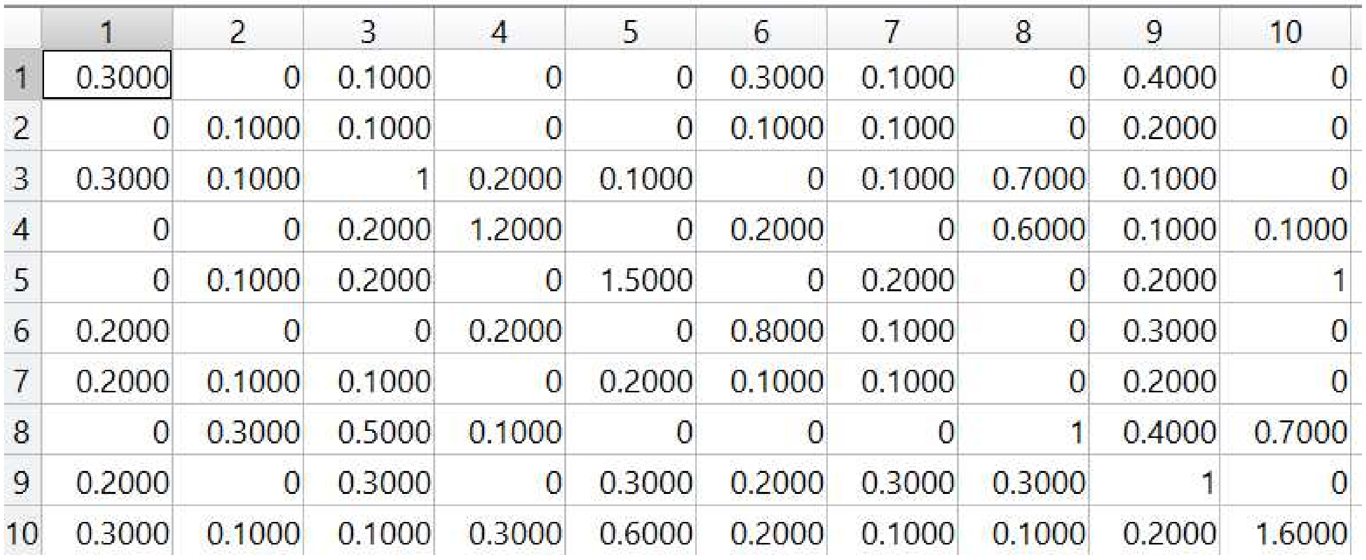
Figure 6 Confusion matrix of CNN

Model 2: Using SAAK on 40,000 training data with 32 reduced dimensions which gave 98.1% accuracy with SVM classification and 91.3% using RF classifier.



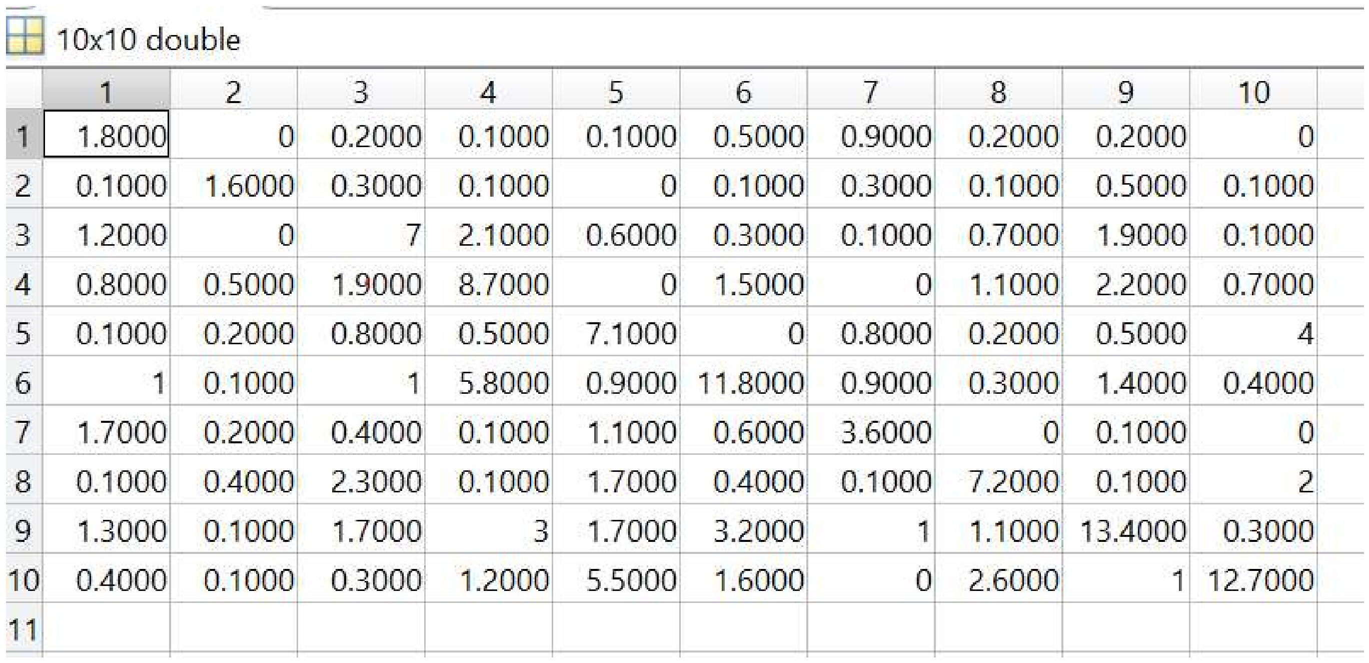
```
Do classification using SVM
Accuracy using SVM: 0.980
[[ 972    0    1    0    0    3    1    1    2    0]
 [   0 1129    2    0    0    1    2    0    1    0]
 [   3    1 1011    2    1    0    1    9    4    0]
 [   0    0    3  992    0    3    0    7    4    1]
 [   0    0    4    0  958    0    3    0    2   15]
 [   2    0    0    8    0  875    2    0    4    1]
 [   6    2    0    1    2    3  942    0    2    0]
 [   1    6   12    1    0    0    0  997    0   11]
 [   3    0    1    4    3    3    3    3  952    2]
 [   3    4    1    5   10    3    1    5    5  972]]

Do classification using RF Classifier
Accuracy using RF: 0.913
[[ 951    0    4    1    1    5   11    3    4    0]
 [   1 1114    4    1    0    1    4    1    8    1]
 [  12    2  951   25    6    3    1    9   22    1]
 [   8    5   20  917    0   16    0   12   25    7]
 [   1    3   10    5  902    0    9    2    5   45]
 [  10    1   10   64    9  765   10    3   15    5]
 [  21    1    5    2   11   10  905    0    3    0]
 [   2    7   30    3   17    4    1  935    5   24]
 [  14    1   21   34   17   33   10   11  828    5]
 [   4    6    5   14   59   17    0   30   13  861]]
```

Figure 7Confusion matrix for SVM and RF



	1	2	3	4	5	6	7	8	9	10
1	0.3000	0	0.1000	0	0	0.3000	0.1000	0	0.4000	0
2	0	0.1000	0.1000	0	0	0.1000	0.1000	0	0.2000	0
3	0.3000	0.1000	1	0.2000	0.1000	0	0.1000	0.7000	0.1000	0
4	0	0	0.2000	1.2000	0	0.2000	0	0.6000	0.1000	0.1000
5	0	0.1000	0.2000	0	1.5000	0	0.2000	0	0.2000	1
6	0.2000	0	0	0.2000	0	0.8000	0.1000	0	0.3000	0
7	0.2000	0.1000	0.1000	0	0.2000	0.1000	0.1000	0	0.2000	0
8	0	0.3000	0.5000	0.1000	0	0	0	1	0.4000	0.7000
9	0.2000	0	0.3000	0	0.3000	0.2000	0.3000	0.3000	1	0
10	0.3000	0.1000	0.1000	0.3000	0.6000	0.2000	0.1000	0.1000	0.2000	1.6000

Figure 8Percentage of difference between CNN and SVM results


10x10 double

	1	2	3	4	5	6	7	8	9	10
1	1.8000	0	0.2000	0.1000	0.1000	0.5000	0.9000	0.2000	0.2000	0
2	0.1000	1.6000	0.3000	0.1000	0	0.1000	0.3000	0.1000	0.5000	0.1000
3	1.2000	0	7	2.1000	0.6000	0.3000	0.1000	0.7000	1.9000	0.1000
4	0.8000	0.5000	1.9000	8.7000	0	1.5000	0	1.1000	2.2000	0.7000
5	0.1000	0.2000	0.8000	0.5000	7.1000	0	0.8000	0.2000	0.5000	4
6	1	0.1000	1	5.8000	0.9000	11.8000	0.9000	0.3000	1.4000	0.4000
7	1.7000	0.2000	0.4000	0.1000	1.1000	0.6000	3.6000	0	0.1000	0
8	0.1000	0.4000	2.3000	0.1000	1.7000	0.4000	0.1000	7.2000	0.1000	2
9	1.3000	0.1000	1.7000	3	1.7000	3.2000	1	1.1000	13.4000	0.3000
10	0.4000	0.1000	0.3000	1.2000	5.5000	1.6000	0	2.6000	1	12.7000
11										

Figure 9Percentage of difference between CNN and SVM results

In figure 6 and 7, we have a confusion matrix when we use CNN and when we use SVM and RF classifier respectively.

In figure 8 and 9, we computed the percentage of difference between the confusion matrix of CNN and SVM and CNN with RF classifier respectively. Thus, 0 signifies that those parameters are recognized similarly in both the models while the other number signifies by how much percentage they differ in identifying.

Observation: We observe that both models are able to identify quite similarly the percentage of error that is different is also very less and it reaches are maximum difference of errors between both the model to 13.4% . But usually the difference in error is near to 1% average in case of svm and 3% average in case of RF classifier.

Improvement: In case of CNN:
1. We saw that more layers which are deeper improves the CNN result so in case of Le-Net5 architecture it is recommended to use to use more deeper layers like increasing the depth of fully connected layer.
2. It is recommended to use some regularization like dropout to prevent overfitting and improve accuracy.
3.Using data augmentation might also help to make the model more robust.
4. We also observed that when we increased the number of filters we got better results, so more filters should be used to detect intricate feature present.

5. Cyclical learning rate can be used in which we can make a function to set the maximum and minimum of learning rate and the model trains with learning rate within the range and finds the most optimal one.

Improvement: In case of SAAK

1.A better classifier leads to better classification. This can be clearly seen when accuracy difference is seen in SVM vs RF classifier. SVM performs much better than RF. So, deciding classifier is an important task in SAAK.

2.PCA provides an important role in reducing the features. The feature dimension is an important parameter and a method should be included in SAAK to find the best feature dimension for a particular dataset.

3.We did not take into consideration the loss encountered in each stage but that might be a possible extension. To use the loss function and model the parameter accordingly at each stage.

4.As SAAK can compute inverse transform it can be used for image generation process.

4 References

[1] https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

[2] https://cntk.ai/pythondocs/CNTK_103D_MNIST_ConvolutionalNeuralNetwork.

[3] http://www.robots.ox.ac.uk/~vgg/practicals/cnn/#part-2-back-propagation-and-derivatives

html. -

[4] https://arxiv.org/ftp/arxiv/papers/1512/1512.00242.pdf

[5] https://arxiv.org/pdf/1703.03633.pdf

[6] https://www.quora.com/What-is-the-purpose-of-using-more-than-1-fully-connected-layer-in-a-convolutional-neural-network

[7]
https://www.researchgate.net/post/How_do_we_choose_the_filters_for_the_convolutional_layer_of_a_Convolution_Neural_Network_CNN1