EE569-HOMEWORK 5

RAKSHITHA PANDURANGA

rpandura@usc.edu

USC ID- 7890-1614-34

PROBLEM 1: CNN AND ITS APPLICATION TO THE MNIST DATASET

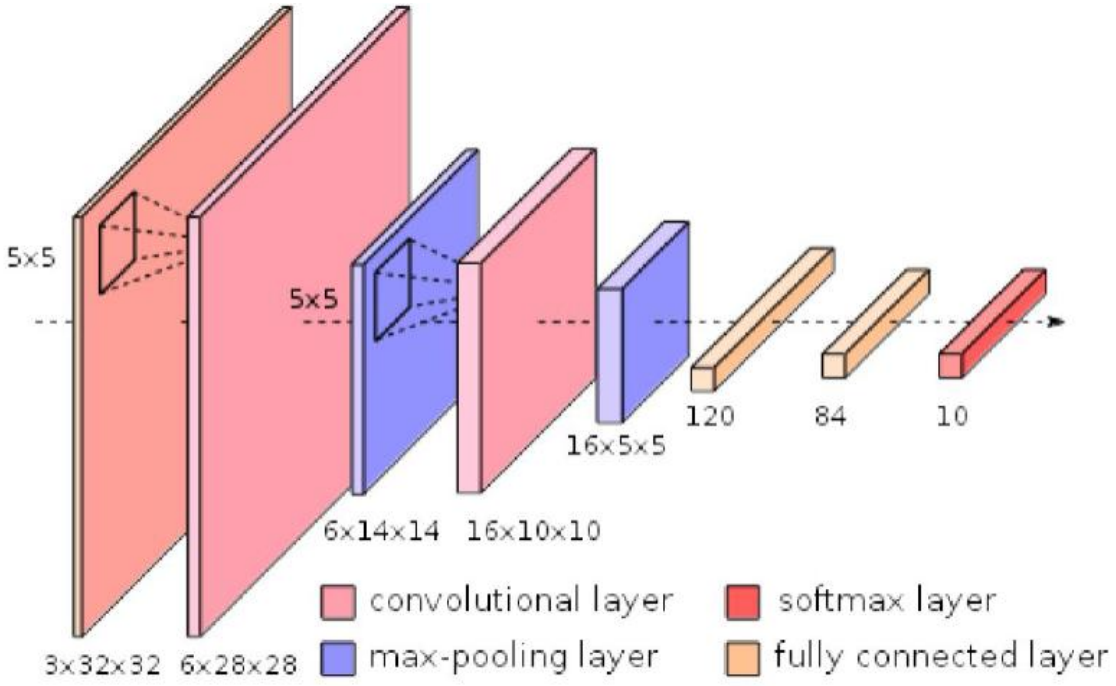
Problem 1.a) CNN Architecture and Training

ABSTRACT AND MOTIVATION

CNN are made up of a lot of neurons that depend on a lot of parameters like weights and biases. The inputs to these are images and can be encoded with properties to make up their architecture. This helps in making up the forward function and it increases its efficiency and it also vastly reduces the number of parameters that are used in a network. Every layer in a Convolutional Neural Network consists of a simple API which transforms the 3Dvolume Input to a 3D volume Output with a lot of differentiable function containing parameters or sometimes, not.

DISCUSSION OF THE QUESTIONS ASKED.

ANSWER 1:



The different CNN components are given below-

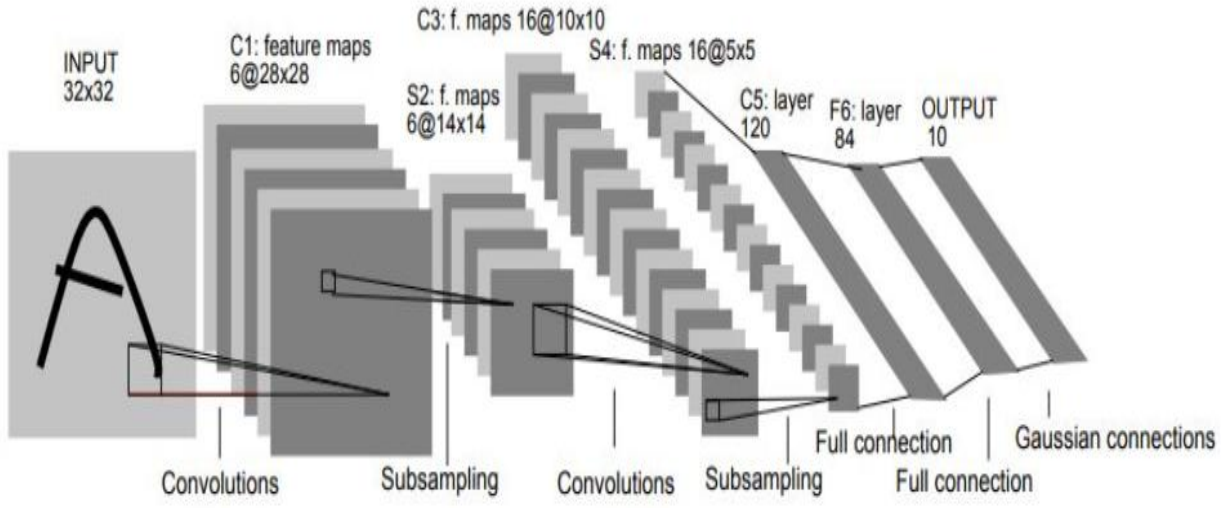
1. The fully Connected Layer –

- The outputs of the Convolutional and the max pooling layers, the neural network has all fully connected layers and these layers are used for the reasoning of the convolutional neural network.

- A fully connected layer will always have neurons connected to all the activations of the previous layers that it is connected to.
- Affine transformation along with the Matrix Multiplications along with a bias or many biases are used for the computation of these Fully Connected Neural Networks.
- The difference between a Fully Connected Layer and a Convolutional Layer are that the Convolutional Layer is just that the Convolutional Layer is connected only to a very local part of the input.
- But, a large number of the neurons in a Convolutional Layer and their volume share parameters which is also a huge difference between the Fully Connected Layer and the Convolutional Layer.
- The main similarity between these two layers is that all the neurons from all the layers are still used to compute the dot products and they function in a very identical way.
- Keeping these differences and similarities, a Fully Connected Layer and a Convolutional Layer can be used to convert it into one another.
- If there is a Convolutional Layer, a Fully Connected Layer will be present that will be used for the implementation of a Forward Function.
- But, it needs a huge weight matrix which has zeros everywhere because it has only a few blocks that are locally connected to each other.
- But, the weights need to be equal in most of the blocks because their parameters will be shared with each other which will lead to parameter sharing
- In contradiction, when the FC layer has to be converted to a Convolutional layer then the filter size has to be absolutely the same as the size of the input volume.
- For example, if a Fully Connected layer has a K = 1024, then the input volume is the same because of the size = 7*7*32. This is done in order to set the depth column fits across the volume of the input.
- The most useful and the most applicable of these two conversions is the Fully Connected Layer to the Convolutional layer.
- Let us take into picture a Convolutional Network Architecture, suppose it has an input size of 224*224*3 image, which will have a lot of Convolutional Layers and Pool Layers that will help in decreasing the image into the volume of the activations that is 7*7*512.
- Summarizing the steps that are required for converting the Fully Converted Layer to the Convolutional Layers are –

- 1. The first and foremost Fully Connected Layer which faces the 7*7*512 volume which uses a filter size = 7which will result in giving the volume of the output which is 1*1*4096
- 2. The next Fully Connected Layer with a Convolutional Layer that has a Filter Size which is 1 and will definitely give an output which is of the output 1*1*4096.
- 3. The last Fully Connected Layer which is similar to the previous layer with 1, will definitely give the output 1*1*1000
- These conversions are done by varying the weight matrices separately in each of these conversions.
- This makes the convolutional layer very much very efficient across many spatial positions.
- More than iterating it through the Convolutional Network over 36 points, Forwardation of the Converted Convolutional layer makes the model much more efficient.
- Therefore, it is best to Resize the image to a bigger one and use that one for the computation of the class scores at many different spatial locations and taking the mean of them.
- In order to use the Convolutional network efficiently and use a stride value which is much much lesser value, then we need to have the allowance for multiple forward passes.
- Example – For a stride of length 16 pixels, we can do it by taking the combination of the Convolutional Network two times rather than just taking it once. It is taken once over the Input or the Original Image and the other time over the input image with a 16 pixel length of spatial shift both row wise and column wise.
- The image shows a fully connected layer which is as shown below and it also shows how a fully connected layer has all the connections of its

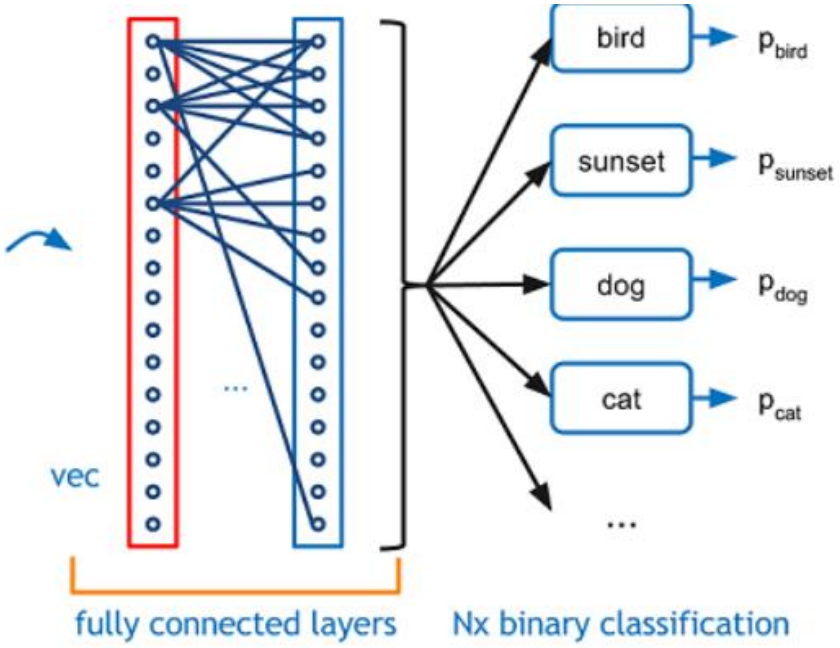
activations to the previous layer which it is connected to.



Original Image published in [LeCun et al., 1998]

Functionality –

- The Fully connected layer is used for having aa fully connected layer in the convolutional network.
- The fully connected layer will have full connections to the previous layers that it is connected to via its activations.
- The activations of the Fully connected network can be easily computed by taking a matrix multiplication and making it being followed by the bias offset.
- The output that comes out of a Convolutional Neural Network is a Fully Connected Layer and is very helpful while extracting the features to the maximum extent.
- The below image shows the functionality of the Fuly Connected layer and from that it can be seen that a Fully Connected Layer has all its activations connected to the previous layer that it is connected to.



2. The Convolutional Layer

- The Convolutional Layer is the “brain” of the Convolutional neural network. It is considered as the basic building block of the Convolutional Neural Network
- The Convolutional layer does the computational liftings that are for the most part of the Convolutional Neural Network.

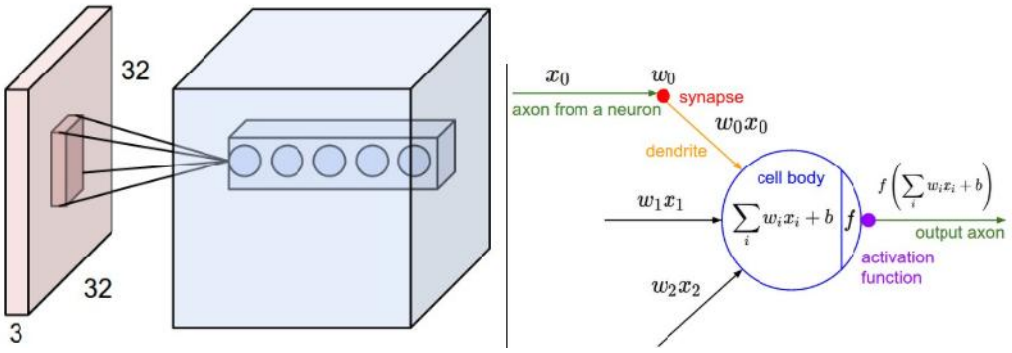
Bird eye view of the Convolutional Layer –

- The parameters of the Convolutional layer are comprised of a huge set of filters that are learnable.
- Even though each of the filters are very very small spatially, that is considering through the length and width of it, it has an extension
- A forward pass is carried out by sliding the filter throughout the image and the Dot Products in between the the different elements belonging to the filter and the input at any position.
- The output of the filter will be a 2D activation map indicating the varying response of the filter on each of these positions, spatially.
- This is how the network runs and learns about the different features that are present in the various spatial positions of the image and they create a particular visual feature that is needed for the representation of the edge of a particular orientation.

- Each of these convolutional layers will have their own set of filters(6 in our case) and they are used to form the 2D activation maps at the end of them.
- At the end of it, these activation maps are used to merge along the depth which will in return produce the volume of the output.

Analogy with the brain –

- Every output of the Convolutional layer can be used to compare it to the outputs of the neurons in a brain.
- These each entry in a 3D output volume is viewed as the output of a brain neuron.
- They have **Local Connectivity**.
- When the inputs are very very high dimensional, connecting each and every neuron to each and every neuron is almost impossible.
- Therefore, every neuron is connected to just its local region of just its input volume.
- The **Receptive Filter** is used as a hyperparameter and decides the degree of spatial connectivity between the neurons.
- The degree of connectivity or the amount of which it gets connected is very equal to the amount of deepness of the volume of the input which is given.
- Giving an example to explain this, if the volume of the input is of the size height = 32, width = 32, depth = 3, with a receptive filter having a size of 25(5*5), then a particular connection in a Convolutional Layer is going to have weights with 75. And for this, the degree of connectivity can only be 3 as the depth maximum can be of 3.



- The image shows how the neurons each of the convolutional layer has connections only with its local region and to the input volume spatially.
- They are **arranged spatially.**
- The three parameters that control the volume of the output and its size are –

- 1. Depth
- 2. Zero padding and
- 3. Stride
- **1. Depth** represents the number of filters that are needed to use and this represents the various sets of neurons.
- 2. **Zero padding** is needed to monitor the spatial size of the volumes of the output. It is used to successfully preserve the various spatial sizes of the volumes of the output.
- 3. **Strides** represents the way the filter move through the input image. It is used to get small volumes of output spatially.
- **Parameter sharing** defines the number of parameters that are needed. The number of parameters needed are to be very high such that the features that are maximum.
- Therefore, the inputs along with the parameter sharing and the three hyper parameters help in getting a **Locally-Connected layer.**

Functionality –

1. Takes in a input with volume height*width*depth.

2. 4 hyper-parametres are properly set –

- Number of filters – depth
- Strides
- Zero padding
- Parameter sharing

3. Gives an output with another volume of height*width*depth

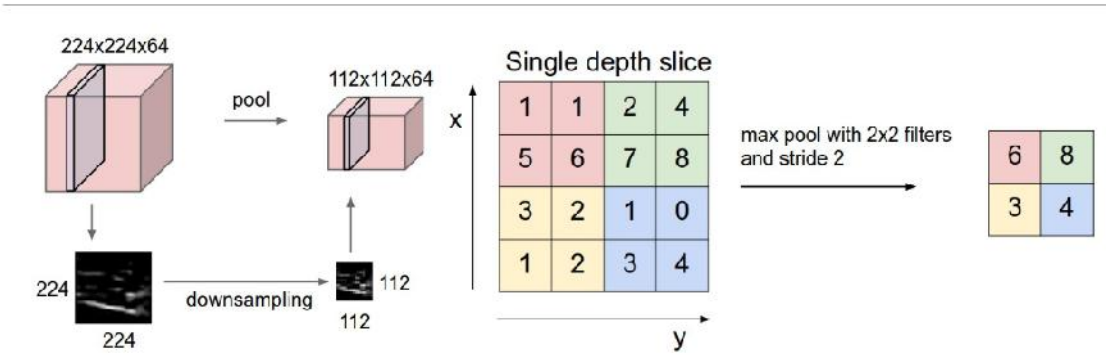
4. The parameter sharing introduces weights and the biases.

5. The best setting woud be to take number of filters to be 3, strides s 1 and parameter setting to be 1.

3. The Max Pooling Layer –

- The amax-pooling layer is usually added in between the Convolutional layers
- It is usually used to decrease the spatial size and also to reduce to amount of parameters.
- It also reduced the computation of the network.

- It also reduces the amount of Over-fitting.
- The max pooling Layer behaves independently and separately on each and every layer and it is then resized spatially. For this, the max pooling layer s used again.
- The widely used max-pool layer has a filter of size 2*2 and with a stride of 2 down-samples.
- Each operation of the max pooling wil take 4 numbers but the dimension of the depth will be the same.
- Therefore, the max pooling layer will do the work with the dimension of the depth will be the same.
- The other operations that the max pool layer are average pooling and L2 norm pooling.
- Back propogation is mainly the backward pass of a maximum function



Functionality –

- The max pooling layer is used to decrease the spatial size
- It is also used to reduce the computation of the network.
- Reduce the amount of parameters
- Reduces the over -fitting of the model.
- The pooling layer can also not be use because of several reasons.
- The first reason is that it can be done is by using the more number of strides and and
- Can also be done by using more number of convolutional layers than that is usually required
- Can also use GANS and VAEs to substitute for the max-pooling function of the convolutional network.

4. The Activation Function-

- An activation function is a basically a non-linear mapping between the input image and the output of that particular layer.
- They are known to bring out the non-linear properties of any network.
- The most fundamental job of an activation function is to convert an input signal to an output signal which is an activated neural network.
- The activation function is used in order to add more efficiency to the network and therefore, making the model learn in the best way possible.
- This is how the non linear mappings between the input and the output of that particular layer is obtained and activated
- The main requirement for an activation function is that it has to be differentiable and that is the most essential part in order to perform back propagation optimization strategy on the model.

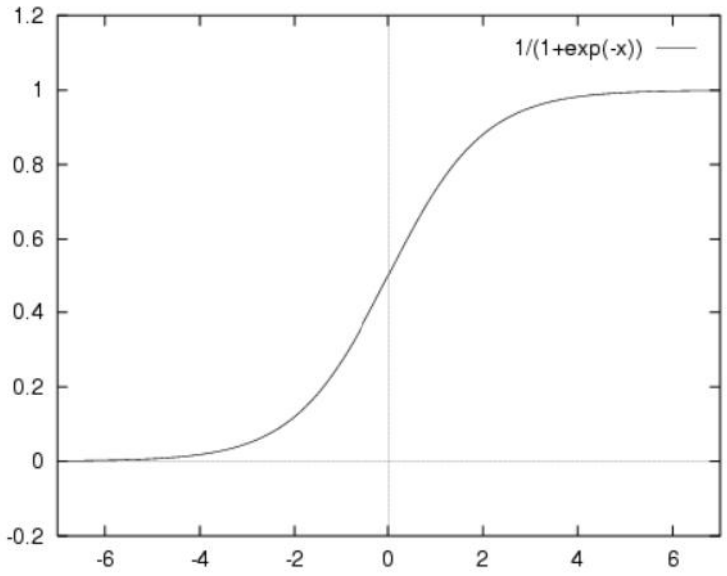
$$Y = \sum (weight * input) + bias$$

- If Y is considered to a neuron, there are no bounds as to till where the values can range from and not.
- Therefore, activation functions are added in order to conclude whether a connection should stay or has to be removed for the model to perform at its best.
- The activation function is activated such that the Y value reaches a particular threshold or not.
- If the threshold is reached, then the activation is activated.
- Or else, the activation function is not activated and stays redundant.

$$A = cx$$

- Derivative w.r.t c is the main function for activation.
- Actiavation function is basically used to find the output of a particular node given the different sets of inputs.
- The output is then behaved as an input to the next layerwhich is again required to be activated in the same way.
- The main properties of an activation function are –
- 1. They have to be non-linear. But, since the identity property will not necessarily satisy the condition of non-linearity, we need to not use it.
- 2.They have to be Monotonic. That is, they can be used to get a surface for a layer which is error free.
- 3. They should be differentiable – this is mainly used methods based on gradient descent .

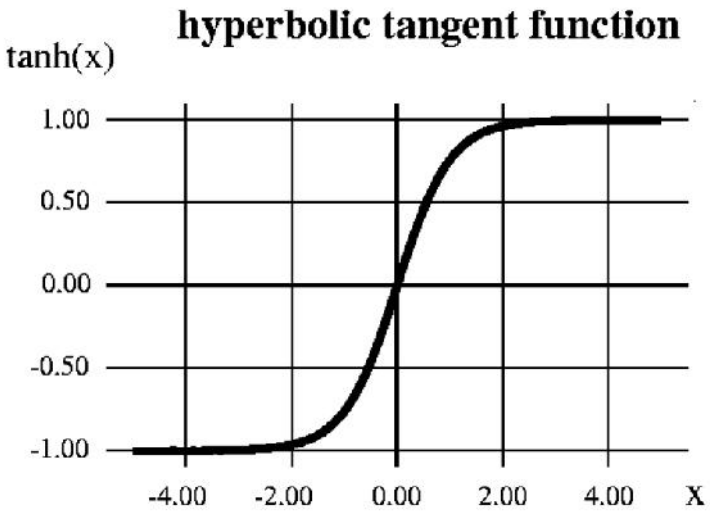
- 4. Range – The range of the activation function has to be finite and should not have any instability between the values because making the activation function otherwise will render the model to become more unstable making the weights very limited.
- 5. Should approximate the identity function near the origin – this means that it should be very similar to the identity function near the origin and should take special care while initialising the weights of the function.
- The different types of activation function that are very commonly used are –
- 1. Sigmoid function-



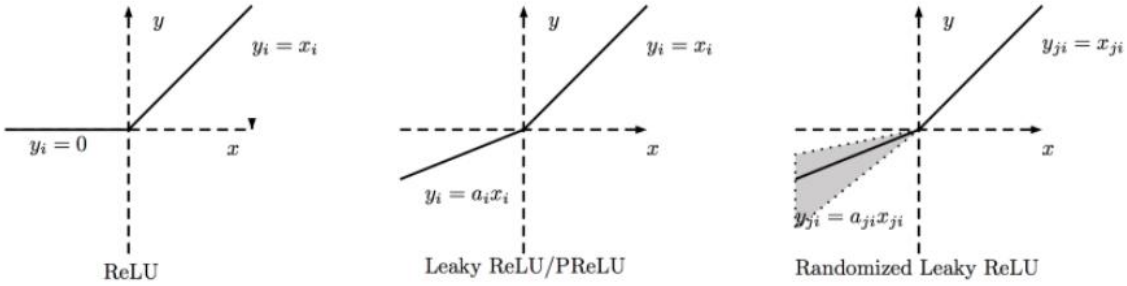
- The activation function of the sigmoid is very similar to the function -

$$f(x) = 1 / 1 + \exp(-x) .$$

- The range of this function lies between 0 and 1.
- It is very good for gradients with vanishing slopes problems.
- The main disadvantage is that they saturate and remove the gradients significantly.
- They also don't converge very easily.
- 2. Tanh function –



- f(x) = 1—
- This function has the mathematical equation as - exp(-2x) / 1 + exp(-2x).
- The range is between -1 and 1
- It performs much better than the sigmoid function.
- 3. Relu function



- The rectified Linear units function is the best of all three of them.
- It gets rid of the gradients which are vanishing and performs better than any other activation function possible.
- The main disadvantage of the relu function is that it can be used only within ny of the hidden layers.
- Relu could sometimes result in dead neurons.

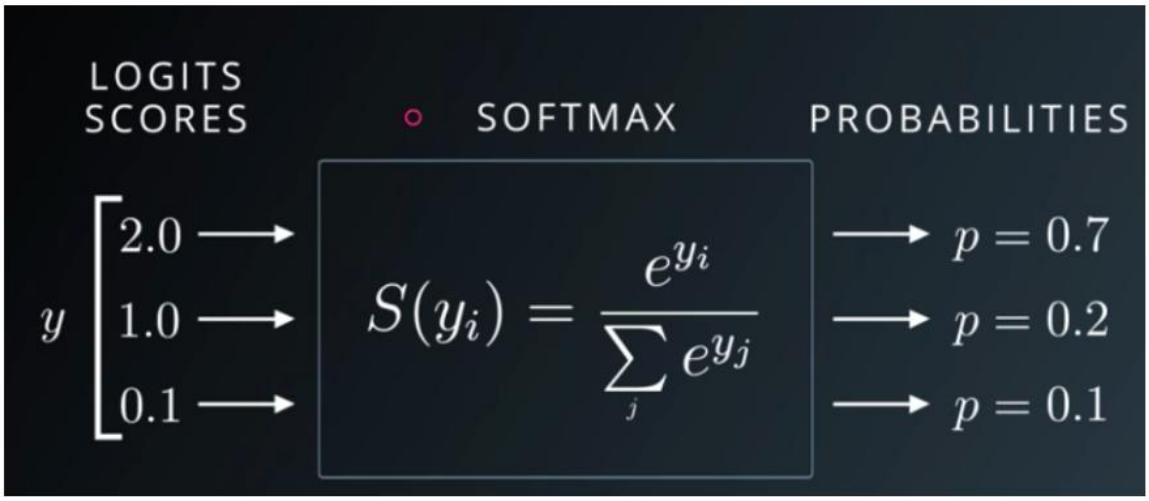
Functionality –

- An activation function is a basically a non-linear mapping between the input image and the output of that particular layer.
- They are known to bring out the non-linear properties of any network.

- The most fundamental job of an activation function is to convert an input signal to an output signal which is an activated neural network.

5. The Soft-Max Function

- A soft max function is one of the activation functions that are used to convert the numbers to probabilities whose addition will result in a 1.
- It is one of the most fundamental elements in deep learning classifcations.



-
- The numerical layer is also called at the logit layer and is used in classification of things.
- Logits are the class scores that are given my the last dense layer of the Convolutional neural network.
- These functions are added before activation.
- The softmax function consists of two main components –
- 1. Special Number e which is taken a power of and the divided by a sum.
- 2. Division by a sum
- A softmax function is basically used in order to smoothen the approximation of the arg max function.
- The softmax function is also called as the softargmax function.
- The logit score should be huge for the softmax output to be huge.
- Softmax function is usually exponential and increases the differences between 0 and 1.
- The output of the softmax function should sum up to a 1 and this can be taken into consideration for probability analysis.
- If the soft max function is chosen as the final function for the classification, then the Cross Enrtopy Loss function has to be used for finding.

- The behaviour of the softmax function can be told as the output numbers that are used for the representation of the probabilities between 0 and 1.
- All the output numbers of the softmax function can either be 0 or positive.
- The sum of the complete vector should be equal to 1.
- The way of choosing a soft max function can be very arbitrary.
- The output of a softmax function is always a probability distribution.
- Properties –
- 1. Used for mapping the vector space to a simplex function
- 2. It is invariant on the same value.
- 3. Softmax is done usually along the diagnols.

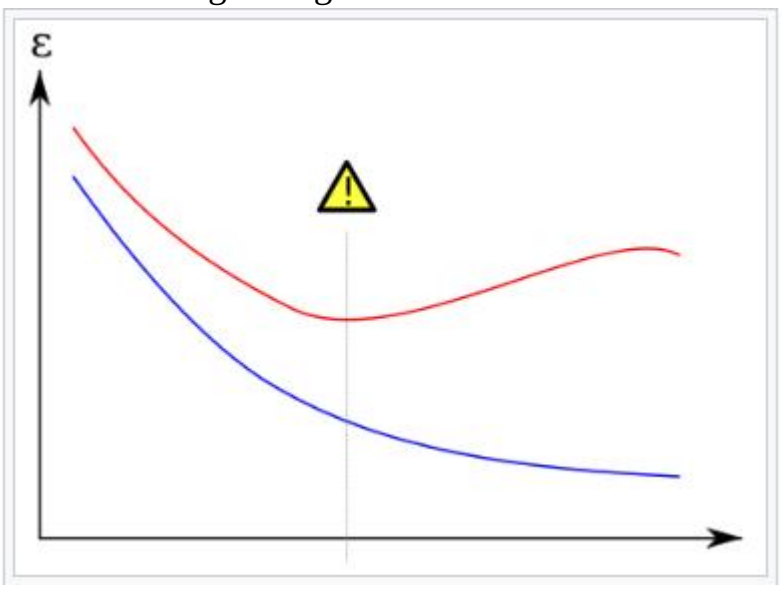
Functionality –

- Smoothening the approximation of the arg function.
- Multi class classification methods
- Linear discriminative analysis.
- The softmax function is also called as the softargmax function.
- The logit score should be huge for the softmax output to be huge.
- Softmax function is usually exponential and increases the differences between 0 and 1.
- The output of the softmax function should sum up to a 1 and this can be taken into consideration for probability analysis.

ANSWER 2-

- Overfitting is the process where the model fits too well to the training data.
- This is an issue while training the model because the model learns all the details and then it learns it so well that it affects the model in a very negative way.
- In the process of overfitting, what happens is that if there are any fluctuations or any noise that is present in the model is considered as a feature and it is definitely learned by the model.
- So when the testing data lacks these errors or noises which the model has considered as a feature and trained upon, it gives an error.
- Because of this, the model fits perfectly to the training data and gives a very good accuracy for it.
- Whereas the testing data performs very badly and the accuracy goes down.
- The testing accuracy is much more smaller than the training accuracy.

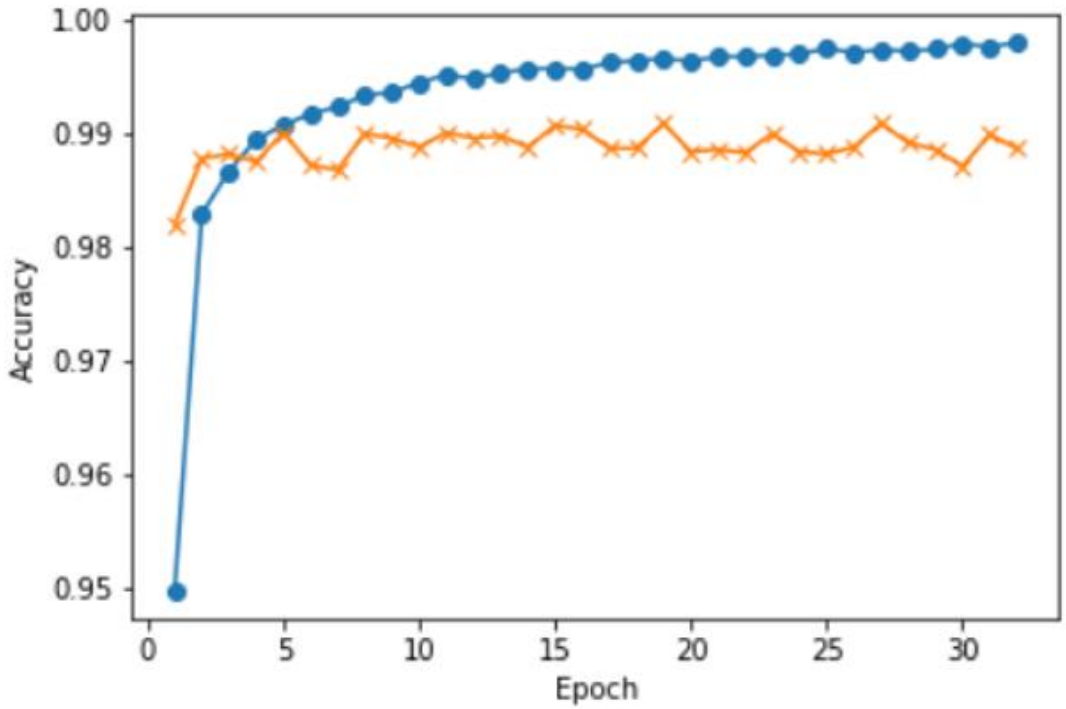
- The overfitting can occur due to various reasons and the main ones being
- 1. Not choosing the right criterion of parameters.
- 2. Not choosing the right structure of the data



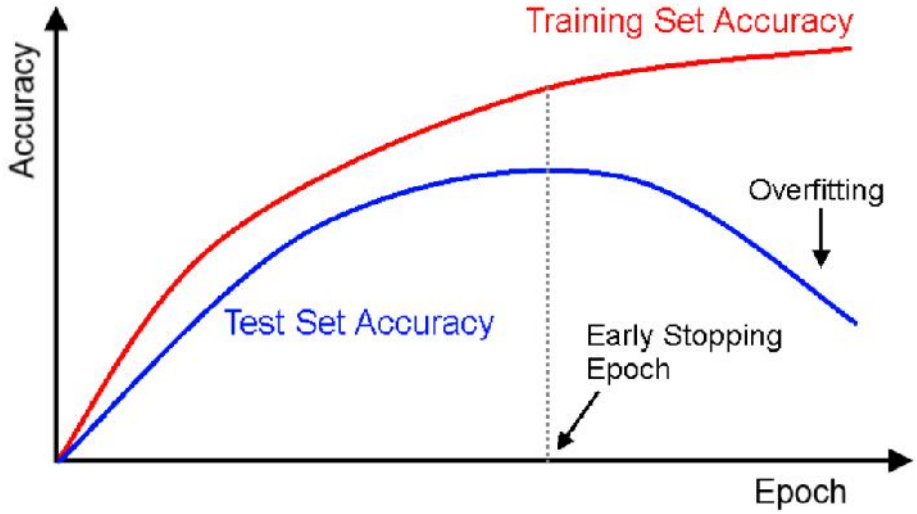
- If the fitted model does not have a lot of parameters then the fitted relationship between the training set and the model is very good such that when a new set comes along, the fitted model performs worse than the fitted model.
- The above image shows the problem of overfitting.
- Example of over fitting would be choosing the number of paramters to be more than that of the number of observations.
- **Detection of** overfitting –
- 1. There is possible way to know whether the model is over fitted or under fitted without the running the machine on a set of test data
- 2. When the data is tested on a particular set of test data, if the model performs really badly on the test data and if it performs very good, in fact, even excellent with the training data, then we can for sure say that the model is undergoing overfitting.
- 3. This is mainly because the noise and the fluctuations that are present within the training set are considered as different features and learnt by the machine
- 4. Therefore, when the model is tested using a test data which is unique to the model, the model performs very badly giving a significantly lower testing accuracy. But the accuracy of the training data set is much larger than the accuracy of the test data.
- Overfitting can be **reduced by** a lot of methods-

- Discussion about how overfitting while training the given model –

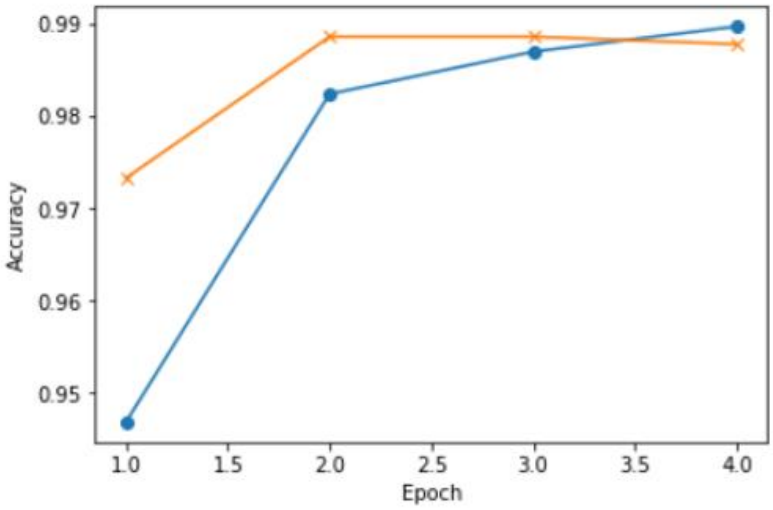
- In our training model, giving the epoch size increased the training accuracy to a very huge extent.
- But it went through over fitting and the testing accuracy is much lesser compared to the training accuracy.
- I ran the training for 32 Epochs and this was the output that I got the over fitting of the model
- As you can see that the testing accuracy(orange line) is significantly lesser and keeps getting worse.
- But the training accuracy, with the model almost reaching a 100% is very good.
- This is happening because the model has learnt and gotten really used to the training dataset that the noise and the fluctuations that are present are also taken down and learnt as features and are used for learning.
- Therefore, the testing data performed very badly as shown in the figure.



- Technique to reduce overfitting –
- EARLY STOPPING –



- Early stopping is one of those techniques that are used to reduce over fitting.
- It is an iterative method that uses gradient descent techniques to reduce over fitting of the model.
- Early stopping is a kind of trigger that uses a monitored performance metric to stop the training when the condition is met.
- That is the training is automatically stopped when the performance of the validation or the testing data set decreases significantly compared to the training epoch.
- The different types of triggers that can be used to start an early stopping are listed below –
- 1. If there are no changes or variations in the particular metric for a significantly good amount of epochs.
- 2. A significantly higher change in a particular metric
- 3. Reduction in the performance of the model over a huge number of epochs.
- 4. Mean of the changes in a particular given metric is over a huge number of epochs.
- In my training of the model, I have made use of Early stopping to wait until the particular " patience level" is met. That is, the output is significantly similarly decreasing over a few epochs.
- An example of which is shown below,



- Where I have given the patience level to be 2.

- This means that the data will stop training as soon as the patience level criteria is met as shown below.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 98s 2ms/step - loss: 0.1700 - acc: 0.9469 - val_loss: 0.0909 - val_acc: 0.9733
Epoch 2/10
60000/60000 [==============================] - 95s 2ms/step - loss: 0.0577 - acc: 0.9824 - val_loss: 0.0385 - val_acc: 0.9886
Epoch 3/10
60000/60000 [==============================] - 92s 2ms/step - loss: 0.0427 - acc: 0.9870 - val_loss: 0.0376 - val_acc: 0.9886
Epoch 4/10
60000/60000 [==============================] - 93s 2ms/step - loss: 0.0351 - acc: 0.9897 - val_loss: 0.0415 - val_acc: 0.9878
Epoch 00004: early stopping
```

As you can see that the training is stopped after the two consistent mean of training accuracies are met.

- The other techniques that can be used for reduction of overfitting are Cross Validation, Regularisation, Ensembling.
- Early stopping with cross validation is shown in problem 1 b. and how it is used to combat early fitting is shown in that problem.

ANSWER 3-

CNNs perform much better than the traditional methods in solving most of the CV problems because –

1. Images are very hard to compute. Computer Vision is considered to be very hard for the Artificial Intelligent machines because solving them is very different than the traditional classifications or algorithms that require just the conventional Image Processing models.

2. The traditional methods are not based on non-linear transformations because ofwhich, while classifying the images, it is not possible to extract all the features that are needed for the classification.

3. Most of the traditional methods use the typical Mathematical approach while dealing with the images also. They use this mapping in order to connect the pixel space to the label space. But, CNNs do consider the mathematical mappings but, they consider the adversarial properties of the inputs, in this case, the images.

4. CNNs perform better with differentiable functions, but most of the traditional vision methods are not capable of taking that into consideration.

5. CNNs have been able to learn hierarchical features which say which features are more relevant for classification and which are not and also gives various methods to compute them.

6. Regular Neural Networks when they receive an input They convert it into various layers of hidden layers. But, the Convolutional Neural Network allows us to import a few unique properties into the architecture.

7. Regular Neural Networks do not have the capability to scale well to complete images. That is if the images given for us are of size 32*32*3. A regular Neural Network wil convert it to a single, but fully connected layer. But, when the input dimensions are increased to a significantly larger amount, they will have problems because it is computationally very difficult for them to convert it to a single layer which is fully connected.

8. Whereas, the CNNs make use of the way these can easily arrange the outputs of each layer in the form of a 3D volumes of neurons.

They basically make use of the fact the inputs are images and they control the architecture after each layer depending on the input layer.

So, the CNNs have 3D to represent the outputs out of each layers –

1. Height

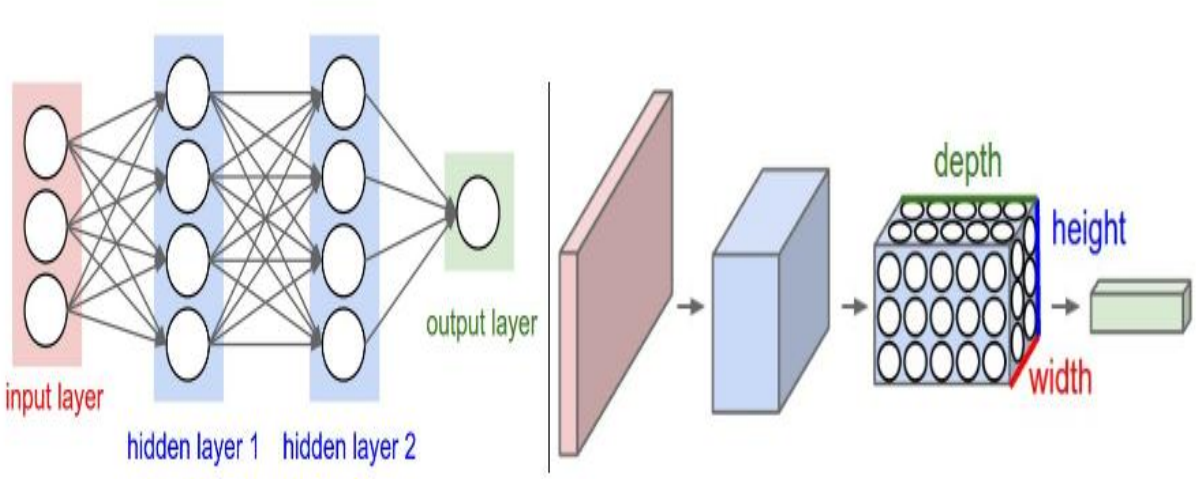
2. Width

3. Depth

For the input given to us, 32*32*3 are the dimensions of the volume given.

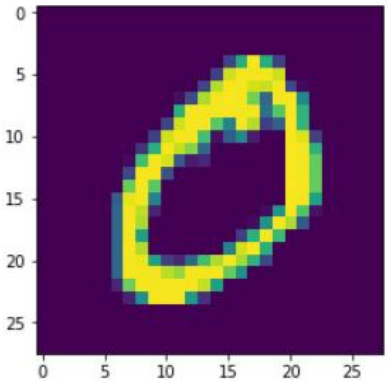
9. Also, the CNNs have local connectivity which means that each part of the CNN is just connected to its local region unlike the regular neural network where all the neurons are connected to every other neuron.

10. The CNNs thus reduce the entire image to a single connected layer of scores. Whereas the regular Neural networks donot have this capability and therefore, CNNs are preferred for images.



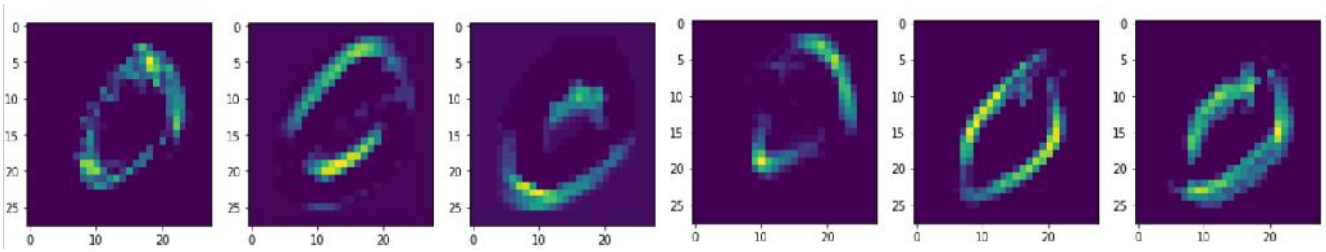
- The above image shows how the CNN(right side) converts each and every input received by each and every layer and converts it into a 3D space as shown above.
- The below diagram shows how the outputs of each layer of a CNN is an image and how this helps the CNNs to scale the image to a full scale.

Input –

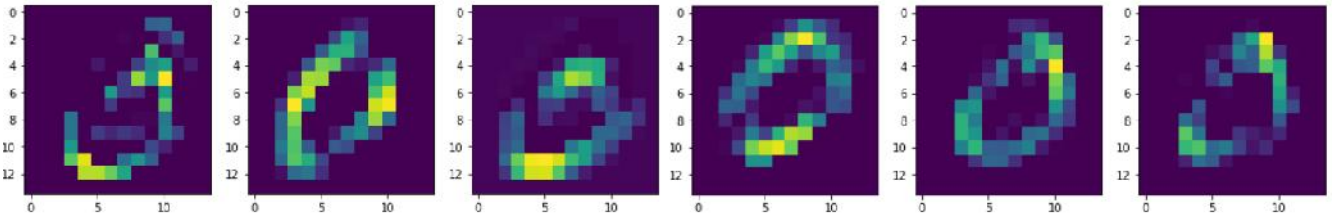


Comparison of the Output after each layer for six filters

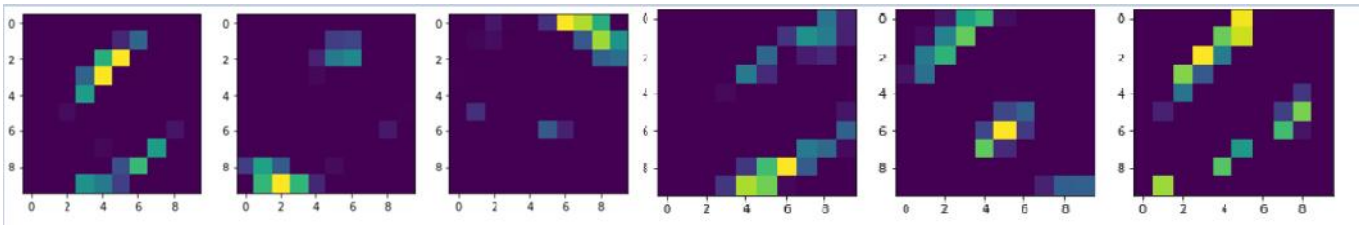
First layer -



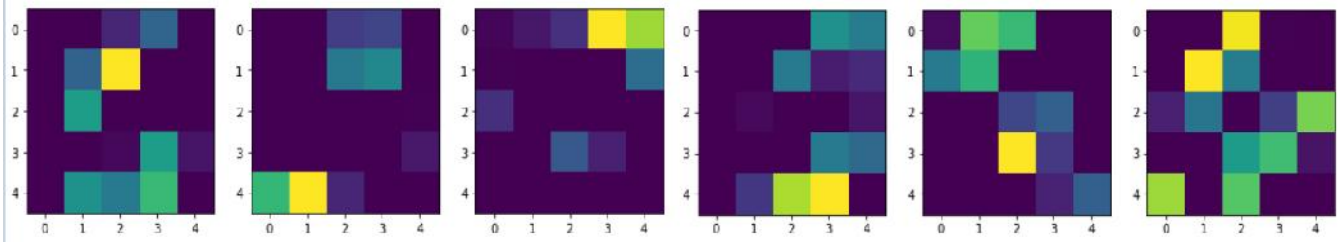
Second layer –



Third layer –



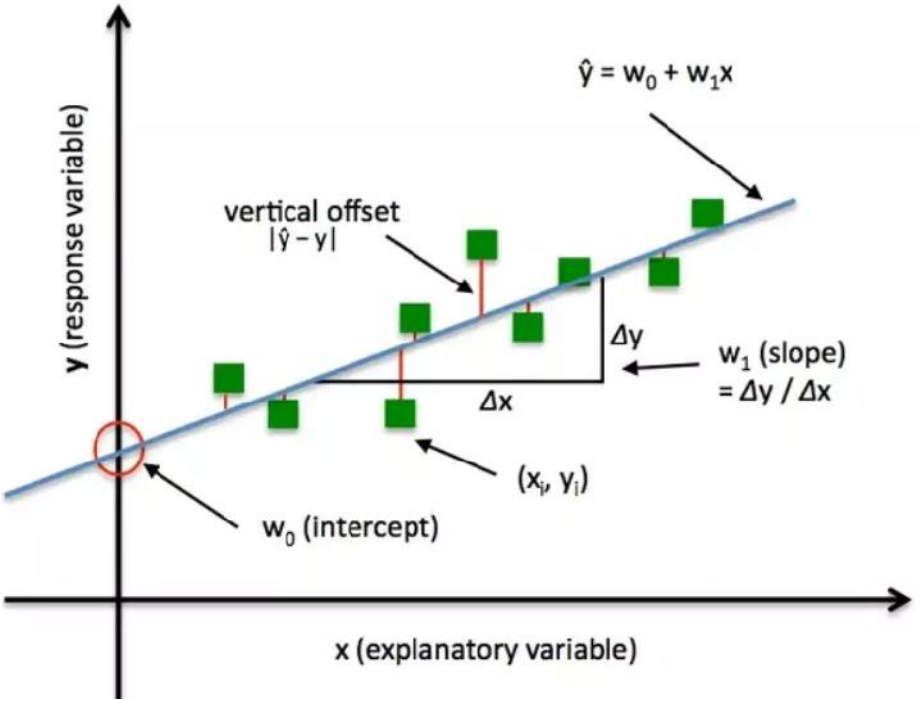
Fourth layer –



ANSWER 4-

Loss function –

- It is one of the best ways to determine whether a particular model is performing well on not the given dataset.
- That is, if the model predictions are mostly right, then the loss function for that particular model will be very less indicating the loss incurred due to such a model performance is very low and very insignificant.
- But, if the model predictions are very much wrong and the model is predicting all the data wrong, i.e., if the testing accuracy is very low then, the Loss Function will be significantly higher indicating that the model has incurred a lot of loss and the predictions are very very wrong.



- The figure shows how a Loss function is calculated.
- It is calculated by taking the slope and the intercepts of the performance curves to determine whether the model is performing well on the given dataset.
- Different factors come into picture while taking the loss function into consideration
- 1. The easiness with which the derivatives can be calculated and
- 2. Percentage of outliers that are present within the dataset.
- Loss Functions can easily be categorised into 2 –
- 1. Regression Losses
- 2. Classification Losses
- The different types of regression losses are as follows-

- 1. Mean squared Error Loss function

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

- This is a type of regression loss.
- It is used to find the average of squared difference between the predicted values and the actual values.
- 2. Mean Absolute Error

$$MAE = \frac{\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid}{n}$$

- This is also a type of regression loss.
- It is used by finding the average of the absolute differences between the predicted and the actual values.
- The different types of Classification Losses are-
- 1. Cross-Entropy Loss/ Negative Log Likelihood –
- This is the most fundamental setting that is used for classification

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

- The Cross entropy loss is given by the above mentioned formula.
- The main feature of this Cross entropy loss is that it restricts all the data that are confident but still wrong.
- We use this loss function in our training of the model in the next section.
- 2. Multi-class SVM loss –

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

- The score of the correct category is greater than the sum of the incorrect ones.
- It is a convex function.

Classical Back Propogation Strategy –

- Back propogation basically calculates a gradient that is required for the estimation of weights.
- It uses the delta rule to find the backward propogation of errors in a convolutional neural network.
- It is another method for finding the automatic differentiation of the convolutional neural network.
- The backward pass for a convolutional layer would also be adding another convolutional layer.
- But they will be spatially flipped upside down.
- **Algorithm -**
- Part 1 - 1*1 Convolution-
- Usually taking 1*1 convolution is very insignificant.
- But in Convolutional Neural Networks, it is not the case. In fact it is much more efficient to use back propogation.
- That is possible because the filters have the capability to extend through the depth of the volume of the input that has come to that layer.
- Part 2 – Dilated Convolutions –
- Dilation is one of the other paramters that is added in order to dilate the convolutional layer.
- For example, if we take a 1D filter which is having a size of 3 then we can easily find out the x as w0*x0+w1x1+w2x2
- But if there are 2 convolutional layers which are stacked one top of the other, then, a receptive field is required which is used for the dilation.
- The back propogation has basically 4 equations on which it is based on –
- 1. One equation to find the error in the output layer which is given by –

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).$$

-
- The first term of the RHS tells how fast the change is happening
- The second equation is given for the rate of change of the cost w.r.t the bias in the network.
- That is given by,

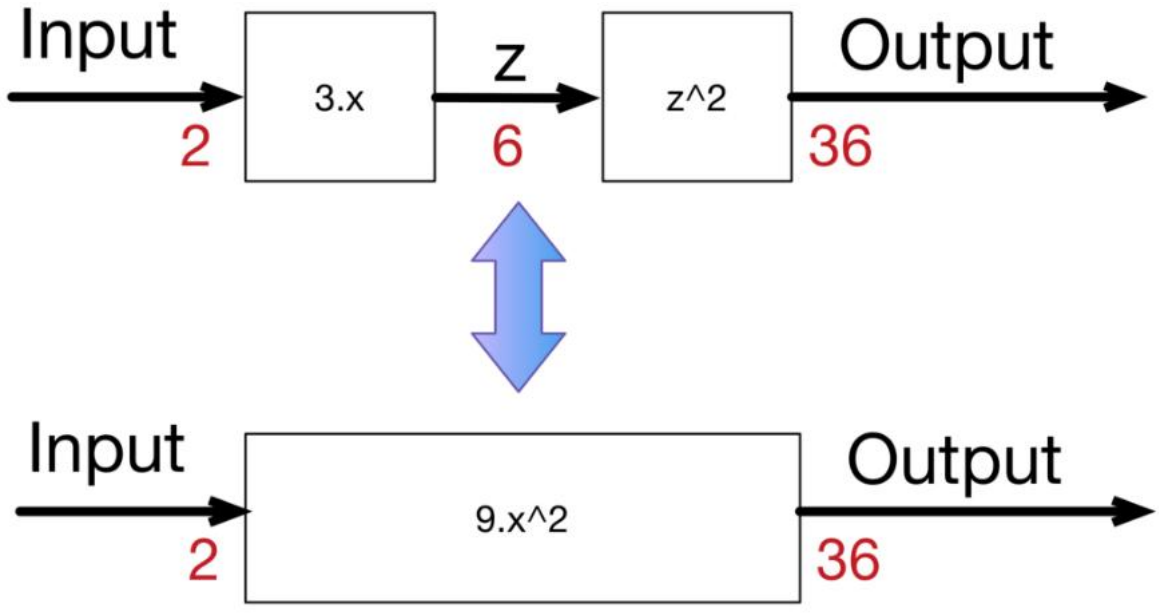
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

- Where the RHS is equal to the rate of change of the cost function.
- The third equation is for the rate of change of the cost w.r.t the weights.
- That is given by,

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l.$$

- The fourth and the final equation that is very important in terms of the back propogation algorithm is that it is for the error which is present in the current layer w.r.t to the errors that are prevalent in the next layer.

$$\delta^L = (a^L - y) \odot \sigma'(z^L).$$



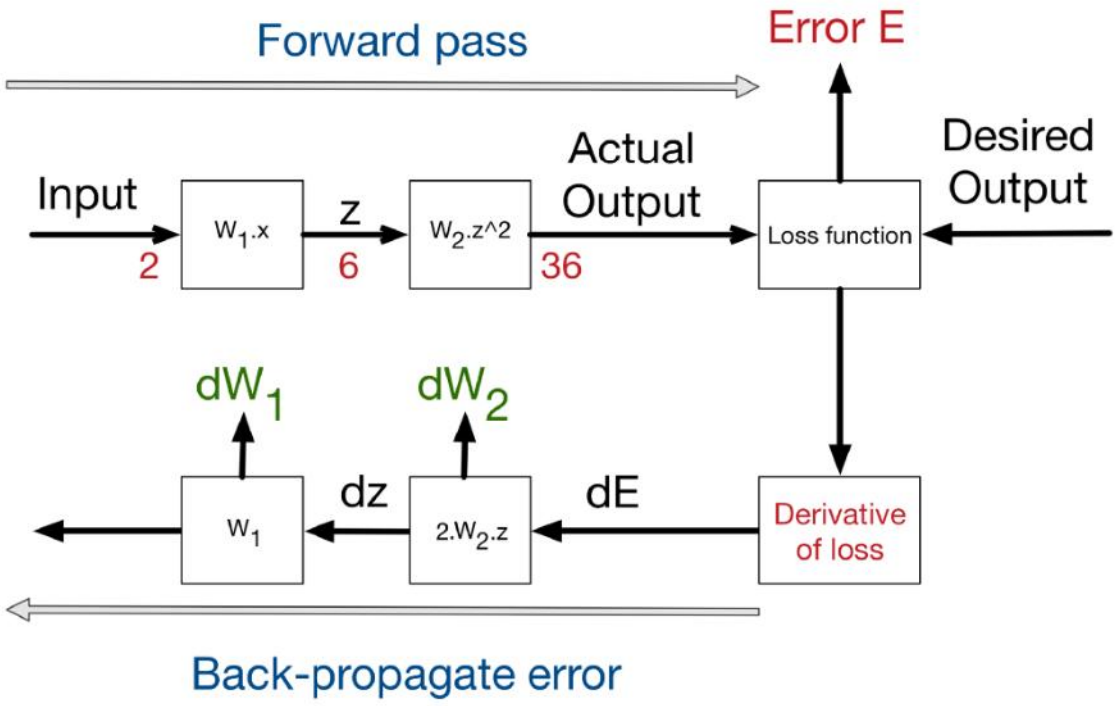


Diagram of Forward and Backward paths

- Disadvantages of using the Back Propogation algorithm –
  1. In gradient descent problems it cannot really find a global minimum. This is caused because of the non-convexity nature of the various error functions that are duely present in a Neural Netwrork.
  2. Normalisation of inputs is not a requirement but a significant increase in the performance.

Problem 1.b)

ABSTRACT AND MOTIVATION

CNN are made up of a lot of neurons that depend on a lot of parameters like weights and biases. The inputs to these are images and can be encoded with properties to make up their architecture. This helps in making up the forward function and it increases its efficiency and it also vastly reduces the number of parameters that are used in a network. Every layer in a Convolutional Neural Network consists of a simple API which transforms the 3Dvolume Input to a 3D volume Output with a lot of differentiable function containing parameters or sometimes, not.

APPROACH AND PROCEDURE -

- The different layers were added like convolutional, maxpooling, dense layers
- The model is trained on train data using the model.fit function
- The model is tested on test data by using model.evaluate.
- The various parameters were set as shown below and the various accuracies are shown below and the various confusion matrices, epoch-accuracy curves are plotted as shown in the figures below.

The 5 different settings that are used are –

Setting – 1 –no variations -default parameters given

1. Parameters used -

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	6	5	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	16	5	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	Types
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	Types/Values/Size
Batch	32
Epoch	10 (with Early stopping)

Testing Stage –

Parameters	Types/Values/Size

Batch	32

2. Accuracy report

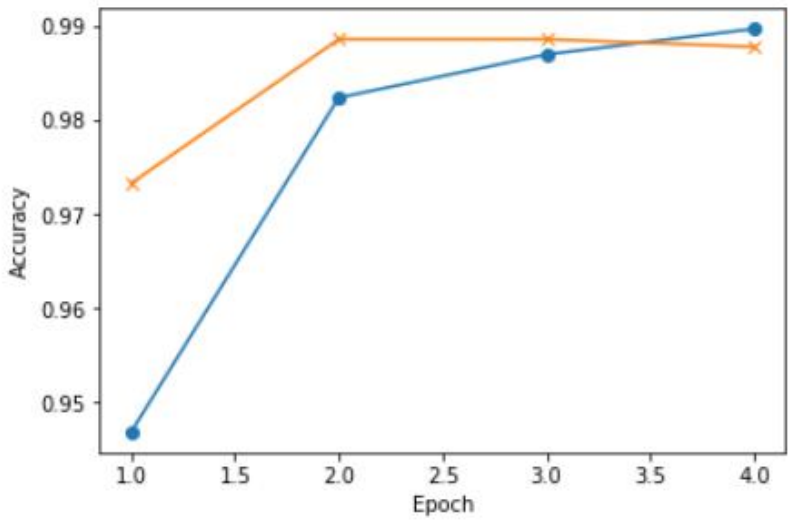
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 98s 2ms/step - loss: 0.1700 - acc: 0.9469 - val_loss: 0.0909 - val_acc: 0.9733
Epoch 2/10
60000/60000 [==============================] - 95s 2ms/step - loss: 0.0577 - acc: 0.9824 - val_loss: 0.0385 - val_acc: 0.9886
Epoch 3/10
60000/60000 [==============================] - 92s 2ms/step - loss: 0.0427 - acc: 0.9870 - val_loss: 0.0376 - val_acc: 0.9886
Epoch 4/10
60000/60000 [==============================] - 93s 2ms/step - loss: 0.0351 - acc: 0.9897 - val_loss: 0.0415 - val_acc: 0.9878
Epoch 00004: early stopping
```

```
10000/10000 [==============================] - 4s 374us/step

[0.04153129385408884, 0.9878]
```

Training Accuracy – 98.97%

Testing Accuracy – 98.78%

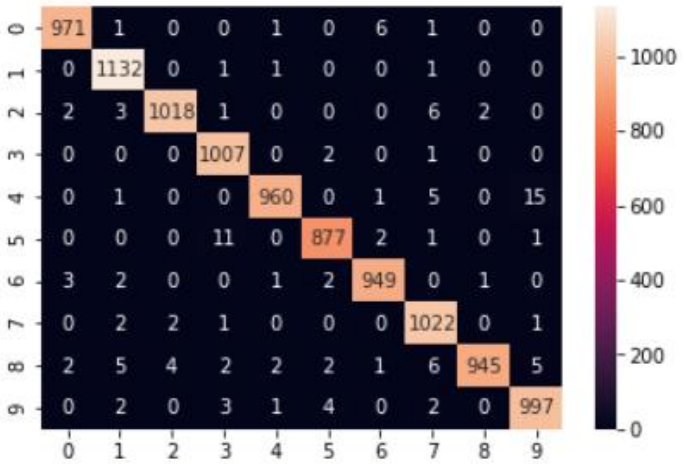
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 971,    1,    0,    0,    1,    0,    6,    1,    0,    0],
       [   0, 1132,    0,    1,    1,    0,    0,    1,    0,    0],
       [   2,    3, 1018,    1,    0,    0,    0,    6,    2,    0],
       [   0,    0,    0, 1007,    0,    2,    0,    1,    0,    0],
       [   0,    1,    0,    0,  960,    0,    1,    5,    0,   15],
       [   0,    0,    0,   11,    0,  877,    2,    1,    0,    1],
       [   3,    2,    0,    0,    1,    2,  949,    0,    1,    0],
       [   0,    2,    2,    1,    0,    0,    0, 1022,    0,    1],
       [   2,    5,    4,    2,    2,    2,    1,    6,  945,    5],
       [   0,    2,    0,    3,    1,    4,    0,    2,    0,  997]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report

	precision	recall	f1-score	support
class 0	0.99	0.99	0.99	980
class 1	0.99	1.00	0.99	1135
class 2	0.99	0.99	0.99	1032
class 3	0.98	1.00	0.99	1010
class 4	0.99	0.98	0.99	982
class 5	0.99	0.98	0.99	892
class 6	0.99	0.99	0.99	958
class 7	0.98	0.99	0.99	1028
class 8	1.00	0.97	0.98	974
class 9	0.98	0.99	0.98	1009
avg / total	0.99	0.99	0.99	10000

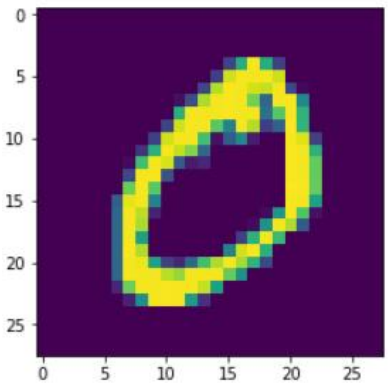
7. Model Summary-

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 6)	0
conv2d_2 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 120)	48120
dense_2 (Dense)	(None, 80)	9680
dense_3 (Dense)	(None, 10)	810

Total params: 61,182
Trainable params: 61,182
Non-trainable params: 0

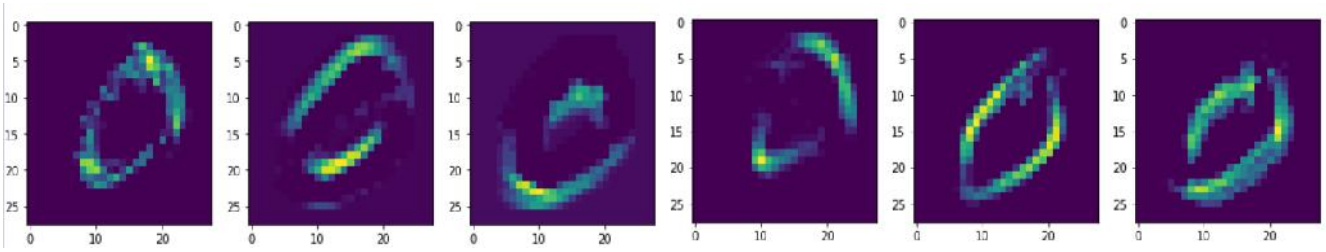
8. Outputs after each layer-

Input –

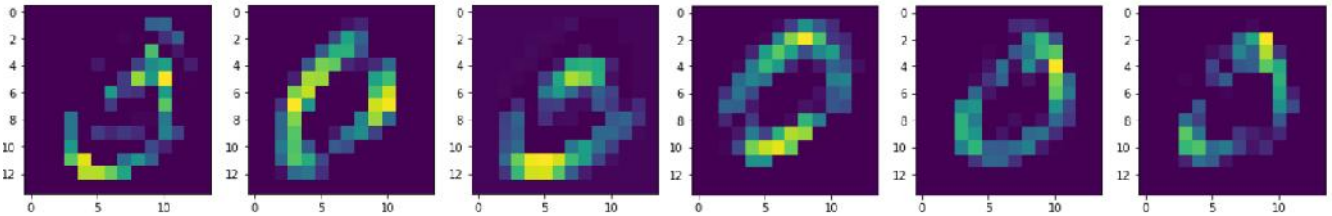


Comparison of the Output after each layer for six filters

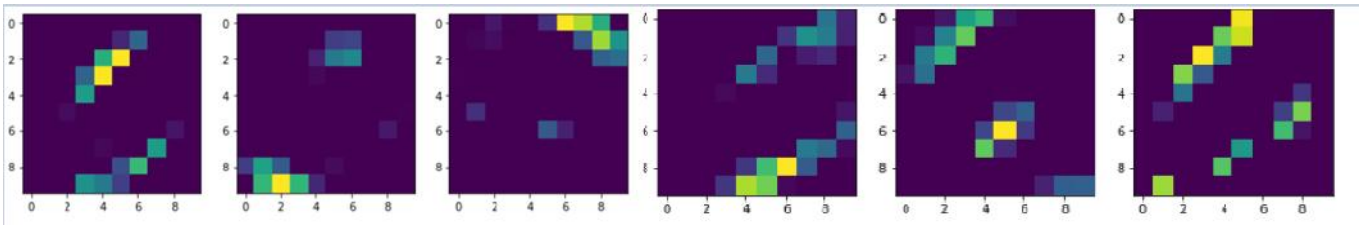
First layer -



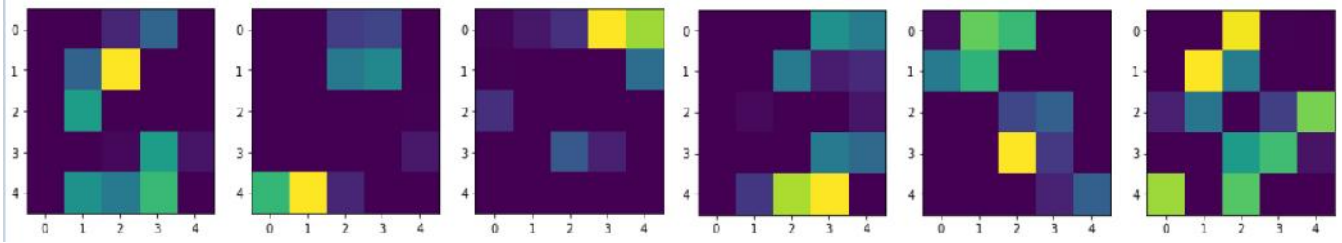
Second layer –



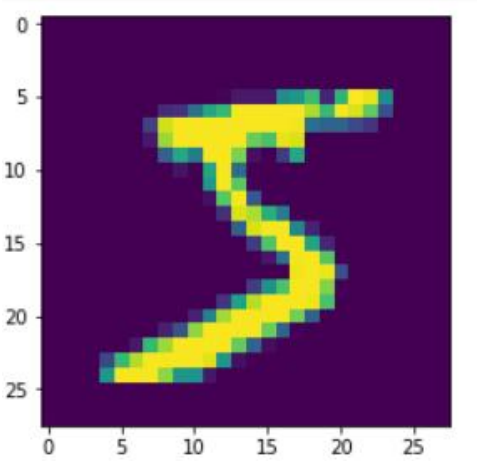
Third layer –



Fourth layer –

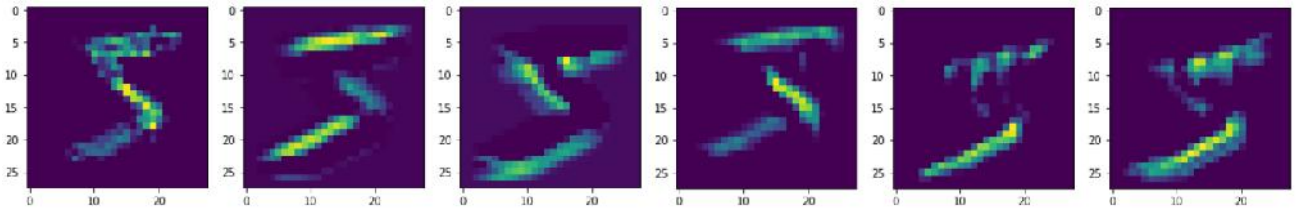


Input –

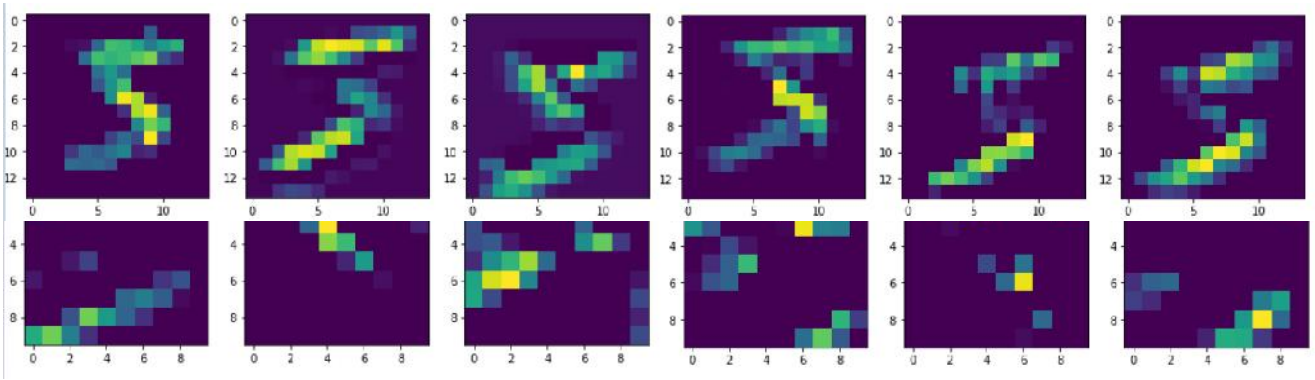


Output after each layer for six filters

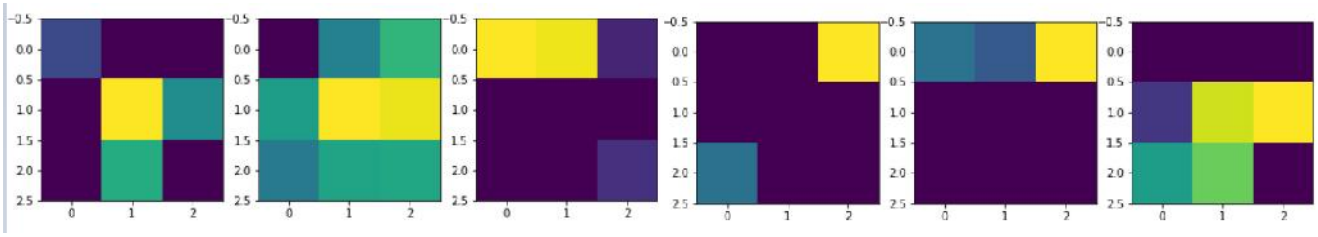
First layer –



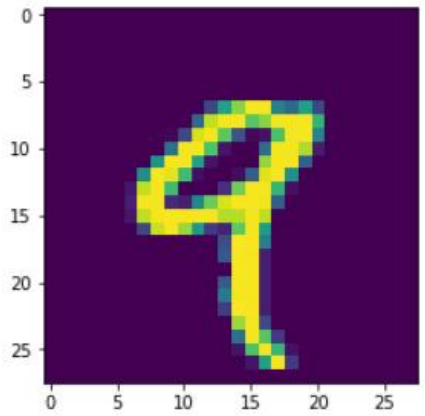
Second layer –



Fourth layer –

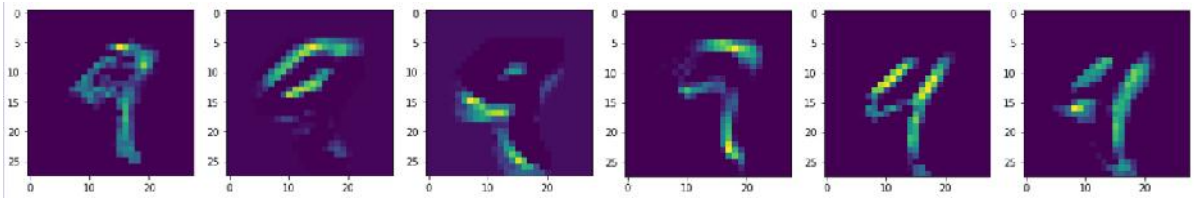


Input –

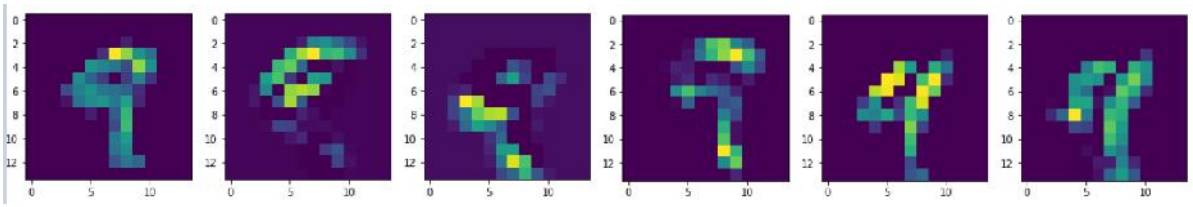


Output after each layer for six filters

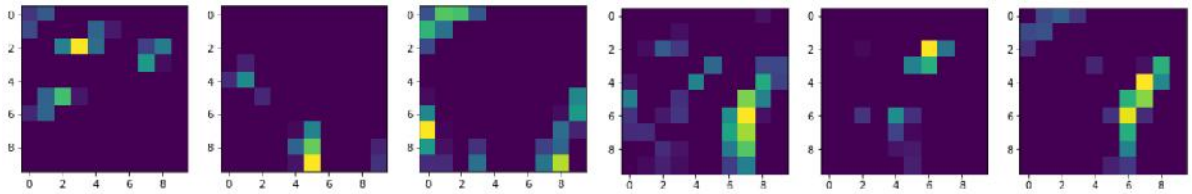
First layer –



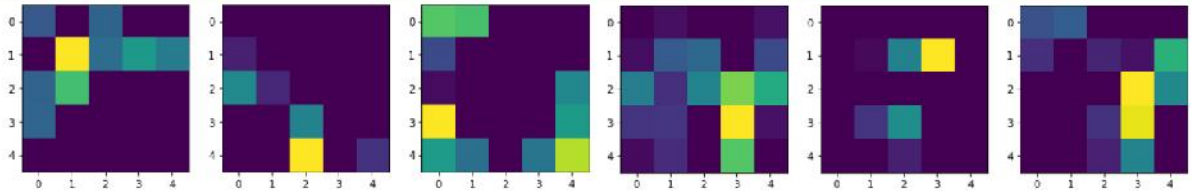
Second layer –



Third layer –



Fourth layer –



Discussion for the Setting-1-

- In this setting, I have used the normal parameters that are used and was recommended in the discussion
- The filter sizes are kept to be 6, 16, kernel sizes to be 5 and pool sizes of the Convolution, MaxPool layer and the Dense Layers are given.
- The input 32*32 grayscale images are passed through the first layer of convolution which extracts the 6 feature maps and the Comparison of the outputs after each layer shows the first layer outputs.
- The input of the maxpooling layer are the feature maps obtained by the convolutional layer.
- The maxpooling layer receives the 28*28 input filter and the output dimensions are decreased to 14*14*6.

- The number of strides given in this layer are 2. Strides are basically used to decide how the filter behaves or convolves around the input image sent.
- I have used Early Stopping here in order to stop the training to run through all the epochs. This is done to stop "OverFitting"
- The confusion matrix is used to represent the number of data that is correctly classified and which aren't.
- For a Confusion matrix if the diagnol elements are significantly larger than the other elments.
- The heat map is just a representation of the classification matrix as shown in the pictures.
- The model summary for that particular setting is as shown above.
- The comparison between the input layer outputs are shown above and you can see that with each layer the dimensions are reducing but the number of features extracted a lot.

Setting – 2 – variations in the filter sizes

1.Parameters used -

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	16	5	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	26	5	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	Types
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	Types/Values/Size
Batch	32
Epoch	10

Testing Stage –

Parameters	Types/Values/Size
Batch	32

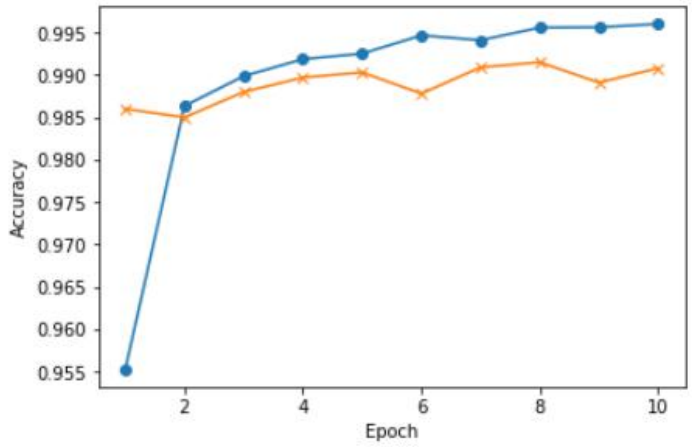
2. Accuracy report

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 74s 1ms/step - loss: 0.1406 - acc: 0.9553 - val_loss: 0.0440 - val_acc: 0.9860
Epoch 2/10
60000/60000 [==============================] - 70s 1ms/step - loss: 0.0448 - acc: 0.9863 - val_loss: 0.0510 - val_acc: 0.9850
Epoch 3/10
60000/60000 [==============================] - 67s 1ms/step - loss: 0.0342 - acc: 0.9899 - val_loss: 0.0438 - val_acc: 0.9880
Epoch 4/10
60000/60000 [==============================] - 73s 1ms/step - loss: 0.0283 - acc: 0.9919 - val_loss: 0.0353 - val_acc: 0.9897
Epoch 5/10
60000/60000 [==============================] - 66s 1ms/step - loss: 0.0254 - acc: 0.9925 - val_loss: 0.0360 - val_acc: 0.9903
Epoch 6/10
60000/60000 [==============================] - 66s 1ms/step - loss: 0.0209 - acc: 0.9947 - val_loss: 0.0475 - val_acc: 0.9878
Epoch 7/10
60000/60000 [==============================]  66s 1ms/step  loss: 0.0212  acc: 0.9941  val loss: 0.0387  val acc: 0.9909
Epoch 8/10
60000/60000 [==============================] - 66s 1ms/step - loss: 0.0176 - acc: 0.9956 - val_loss: 0.0475 - val_acc: 0.9915
Epoch 9/10
60000/60000 |==============================| - 66s 1ms/step - loss: 0.0176 - acc: 0.9956 - val loss: 0.0669 - val acc: 0.9891
Epoch 10/10
60000/60000 [==============================] - 65s 1ms/step - loss: 0.0168 - acc: 0.9960 - val_loss: 0.0677 - val_acc: 0.9908
Epoch 00010: early stopping

10000/10000 [==============================] - 5s 463us/step

[0.06772053957419066, 0.9908]
```

Accuracy on training data – 99.6%

Accuracy on testing data – 99.08%

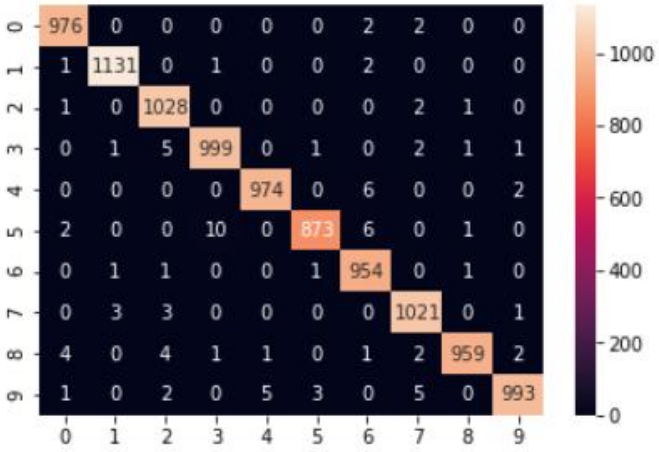
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 976,    0,    0,    0,    0,    0,    2,    2,    0,    0],
       [   1, 1131,    0,    1,    0,    0,    2,    0,    0,    0],
       [   1,    0, 1028,    0,    0,    0,    0,    2,    1,    0],
       [   0,    1,    5,  999,    0,    1,    0,    2,    1,    1],
       [   0,    0,    0,    0,  974,    0,    6,    0,    0,    2],
       [   2,    0,    0,   10,    0,  873,    6,    0,    1,    0],
       [   0,    1,    1,    0,    0,    1,  954,    0,    1,    0],
       [   0,    3,    3,    0,    0,    0,    0, 1021,    0,    1],
       [   4,    0,    4,    1,    1,    0,    1,    2,  959,    2],
       [   1,    0,    2,    0,    5,    3,    0,    5,    0,  993]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report

	precision	recall	f1-score	support
class 0	0.99	1.00	0.99	980
class 1	1.00	1.00	1.00	1135
class 2	0.99	1.00	0.99	1032
class 3	0.99	0.99	0.99	1010
class 4	0.99	0.99	0.99	982
class 5	0.99	0.98	0.99	892
class 6	0.98	1.00	0.99	958
class 7	0.99	0.99	0.99	1028
class 8	1.00	0.98	0.99	974
class 9	0.99	0.98	0.99	1009
avg / total	0.99	0.99	0.99	10000

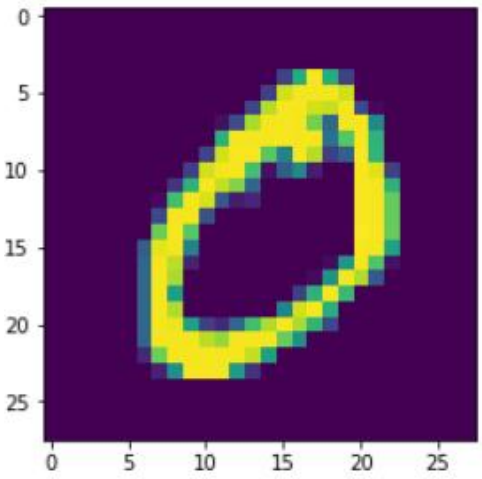
7. Model Summary-

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 28, 28, 16)	416
max_pooling2d_7 (MaxPooling2	(None, 14, 14, 16)	0
conv2d_8 (Conv2D)	(None, 10, 10, 26)	10426
max_pooling2d_8 (MaxPooling2	(None, 5, 5, 26)	0
flatten_4 (Flatten)	(None, 650)	0
dense_10 (Dense)	(None, 120)	78120
dense_11 (Dense)	(None, 80)	9680
dense_12 (Dense)	(None, 10)	810

Total params: 99,452
Trainable params: 99,452
Non-trainable params: 0

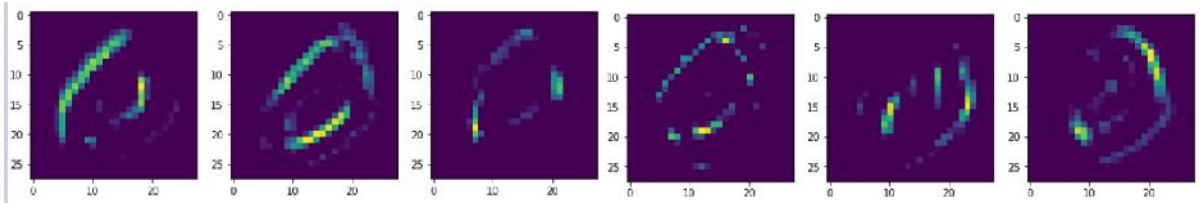
8. Outputs after each layer-

Input –

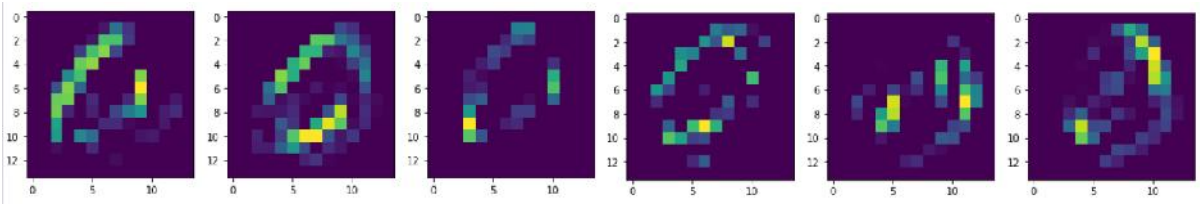


Output after each layer for six filters

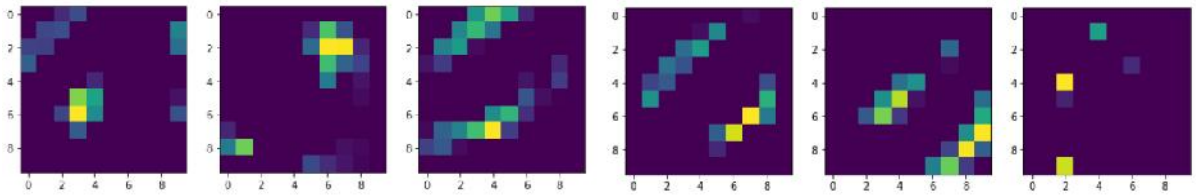
First layer –



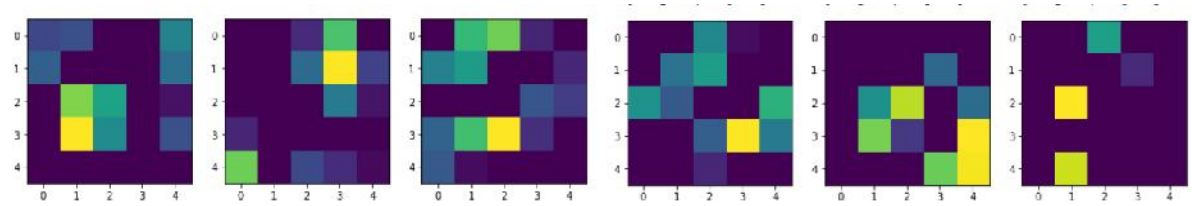
Second layer –



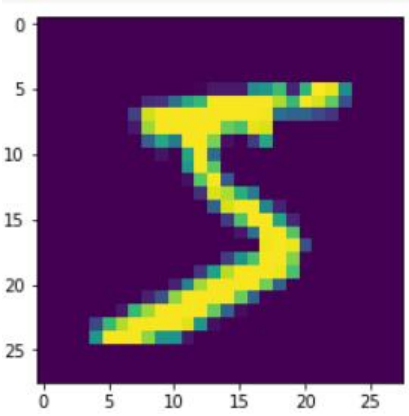
Third layer –



Fourth layer –

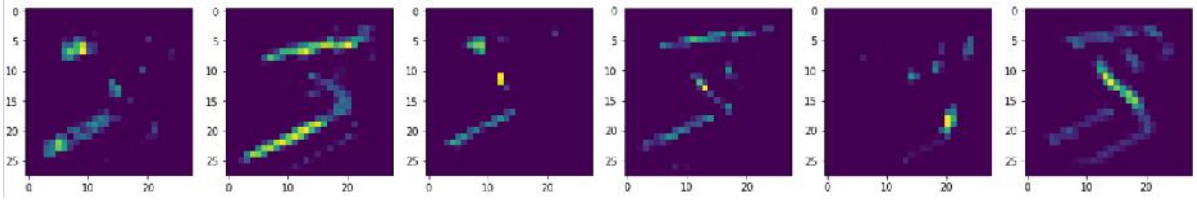


Input –

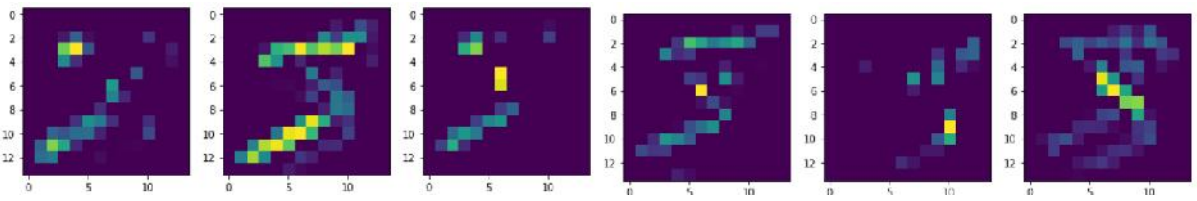


Output after each layer for six filters

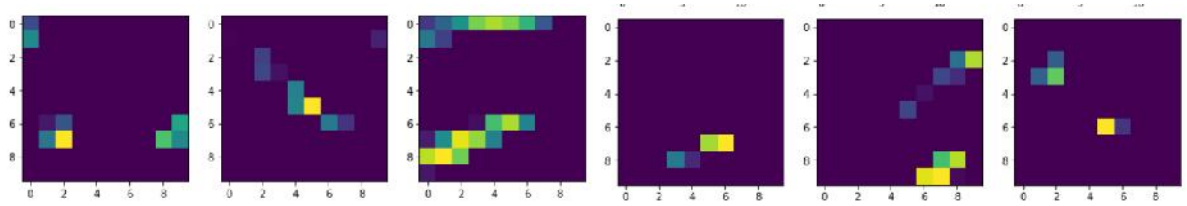
First layer –



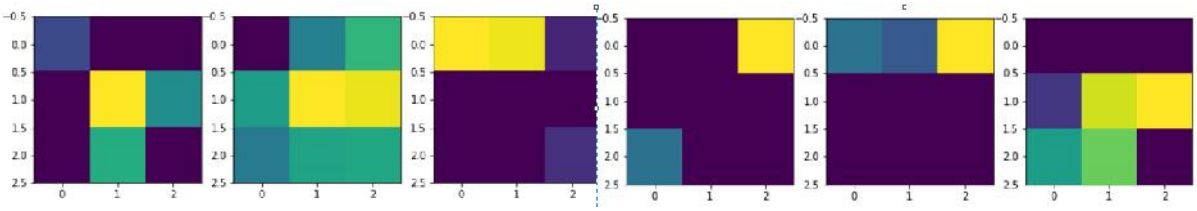
Second layer –



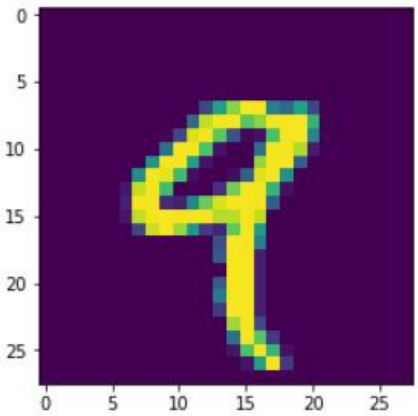
Third layer –



Fourth layer –

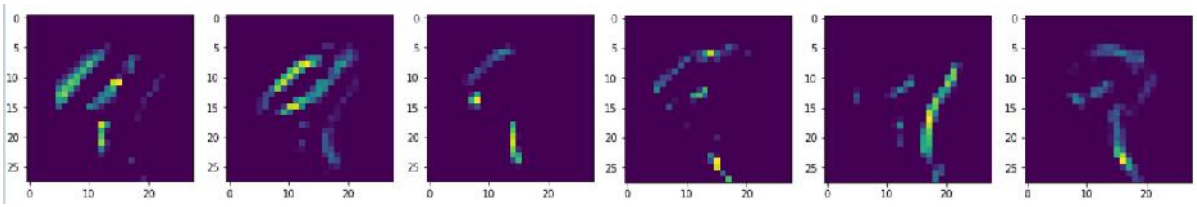


Input –

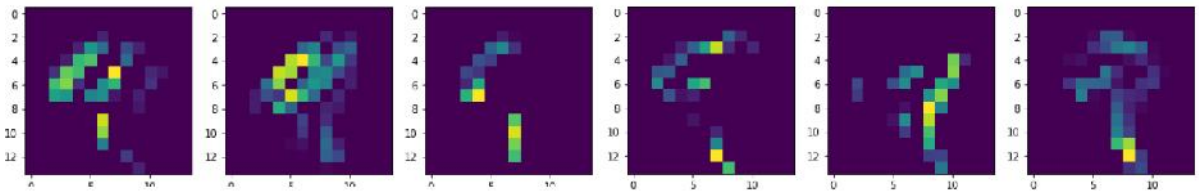


Output after each layer for six filters

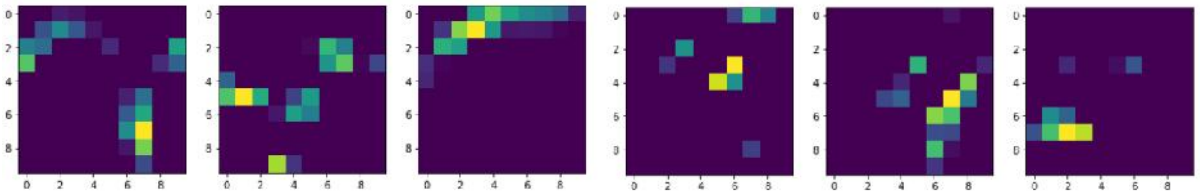
First layer –



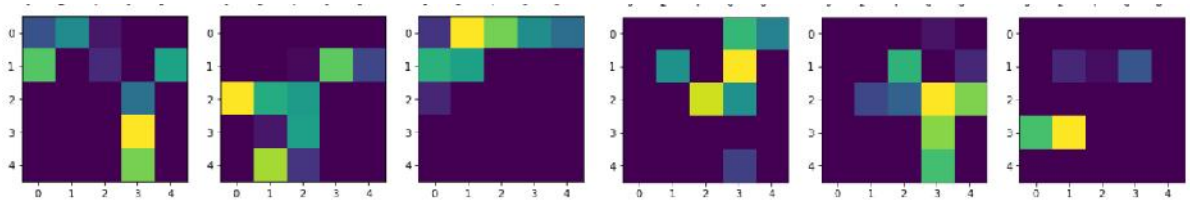
Second layer –



Third layer –



Fourth layer –



Discussion –

- In this setting, I have given a huge increase in the filter to extract extra features to increase the efficiency of the model developed.
- This can very effectively be seen in the accuracy report where there is a significant increase in the accuracies found in both the methods.
- **Discussion about variations in filter sizes –**
- First I tried to decrease the filter size, but the accuracy decreased significantly.
- That is because the filters are used to extract features from the input image.
- Therefore, I decided I have to give more filter size in order to extract extra features, more than the normal, default settings.
- This turned out to be true as there was a significant increase in the accuracy which is seen in the accuracy report from 98.78 to 99.08%

Discussion about Early Stopping-

- I have used Early Stopping here in order to stop the training to run through all the epochs. This is done to stop "OverFitting"
- Early stopping is a kind of trigger that uses a monitored performance metric to stop the training when the condition is met.
- That is the training is automatically stopped when the performance of the validation or the testing data set decreases significantly compared to the training epoch.
- In this case, I have given a patience level of 2. So the training is stopped if it meets with a consecutive 2 values which are significantly lesser than the training data.
- This helps in the model getting over-fitted and helps in the saving the computational time.

Discussion about the Computational Complexity –

- Running the same model without giving the Early stopping was very hard.
- It took a lot of time for the computation as the filter size was increased.
- But with early stopping, it did not run for all the epochs which helped in stopping the training early.
- Therefore, after early stopping the computational Complexity reduced significantly.

Setting – 3 – variations in kernel size

1.Parameters used -

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	6	9	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	16	9	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	Types
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	Types/Values/Size
Batch	32
Epoch	10

Testing Stage –

Parameters	Types/Values/Size
Batch	32

2. Accuracy report

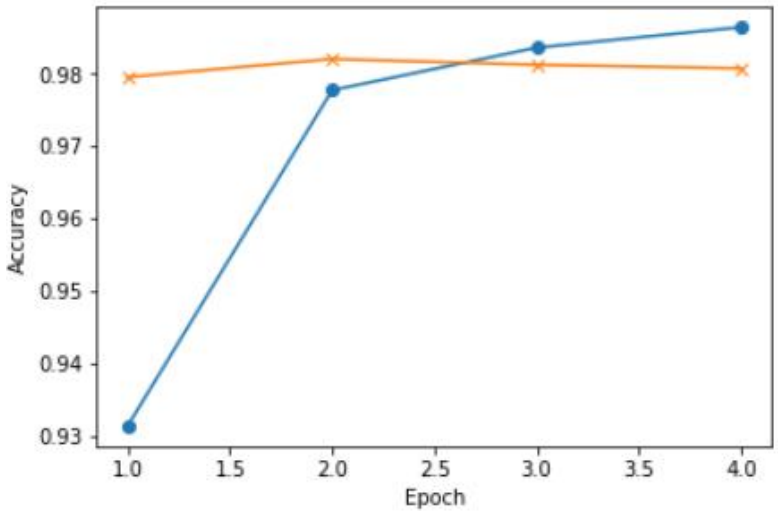
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================]   85s 1ms/step   loss: 0.2215   acc: 0.9313   val loss: 0.0672   val acc: 0.9795
Epoch 2/10
60000/60000 [------------------------------] - 85s 1ms/step - loss: 0.0712 - acc: 0.9777 - val_loss: 0.0558 - val_acc: 0.9820
Epoch 3/10
60000/60000 [==============================] - 84s 1ms/step - loss: 0.0543 - acc: 0.9836 - val_loss: 0.0657 - val_acc: 0.9812
Epoch 4/10
60000/60000 [------------------------------] - 90s 2ms/step - loss: 0.0467 - acc: 0.9864 - val_loss: 0.0717 - val_acc: 0.9807
Epoch 00004: early stopping
```

```
10000/10000 [==============================] - 6s 623us/step

[0.07166753204037049, 0.9807]
```

Accuracy on training data – 98.64%

Accuracy on testing data – 98.07%

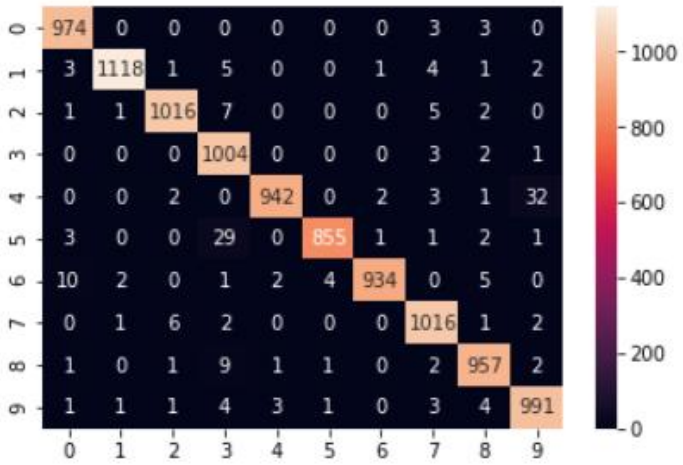
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 974,    0,    0,    0,    0,    0,    0,    3,    3,    0],
       [   3, 1118,    1,    5,    0,    0,    1,    4,    1,    2],
       [   1,    1, 1016,    7,    0,    0,    0,    5,    2,    0],
       [   0,    0,    0, 1004,    0,    0,    0,    3,    2,    1],
       [   0,    0,    2,    0,  942,    0,    2,    3,    1,   32],
       [   3,    0,    0,   29,    0,  855,    1,    1,    2,    1],
       [  10,    2,    0,    1,    2,    4,  934,    0,    5,    0],
       [   0,    1,    6,    2,    0,    0,    0, 1016,    1,    2],
       [   1,    0,    1,    9,    1,    1,    0,    2,  957,    2],
       [   1,    1,    1,    4,    3,    1,    0,    3,    4,  991]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report

	precision	recall	f1-score	support
class 0	0.98	0.99	0.99	980
class 1	1.00	0.99	0.99	1135
class 2	0.99	0.98	0.99	1032
class 3	0.95	0.99	0.97	1010
class 4	0.99	0.96	0.98	982
class 5	0.99	0.96	0.98	892
class 6	1.00	0.97	0.99	958
class 7	0.98	0.99	0.98	1028
class 8	0.98	0.98	0.98	974
class 9	0.96	0.98	0.97	1009
avg / total	0.98	0.98	0.98	10000

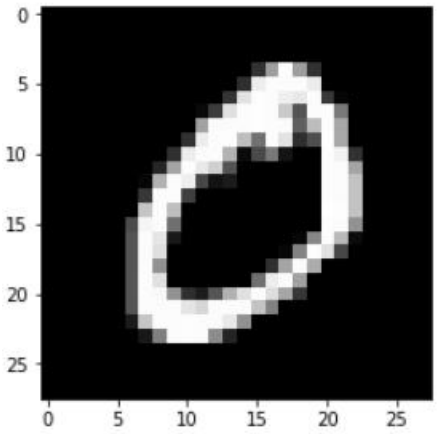
7. Model Summary-

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 28, 28, 6)	492
max_pooling2d_15 (MaxPooling	(None, 14, 14, 6)	0
conv2d_16 (Conv2D)	(None, 6, 6, 16)	7792
max_pooling2d_16 (MaxPooling	(None, 3, 3, 16)	0
flatten_8 (Flatten)	(None, 144)	0
dense_22 (Dense)	(None, 120)	17400
dense_23 (Dense)	(None, 80)	9680
dense_24 (Dense)	(None, 10)	810

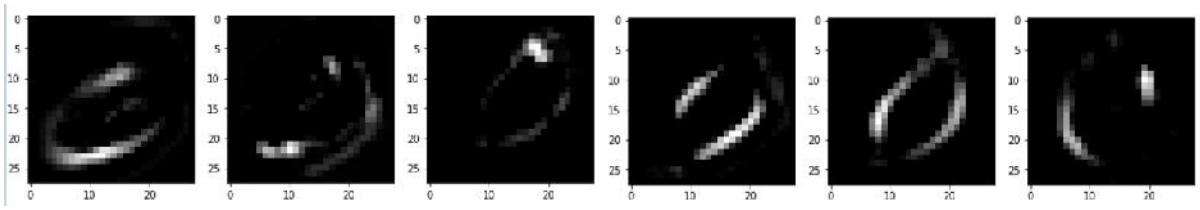
Total params: 36,174
Trainable params: 36,174
Non-trainable params: 0

8. Outputs after each layer-

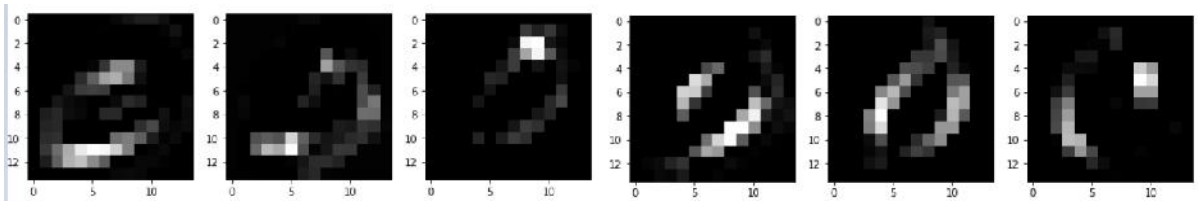
Input—



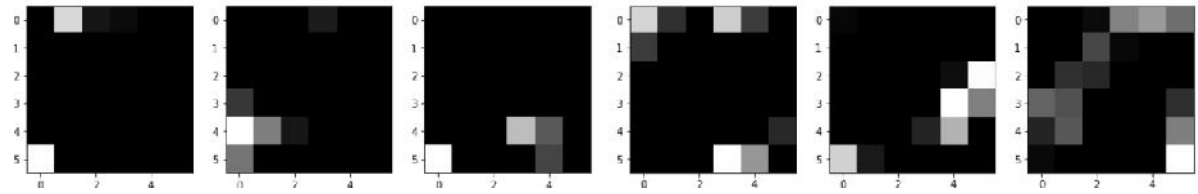
First layer –



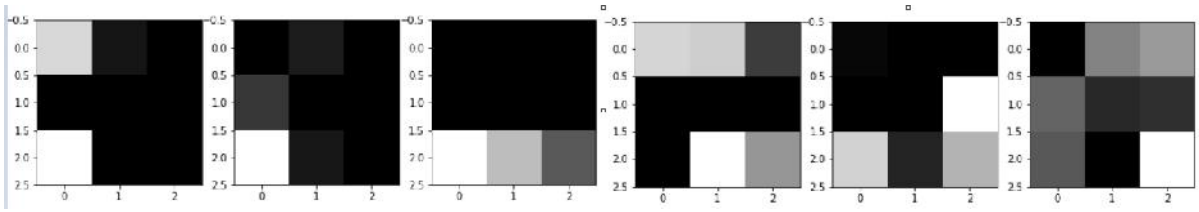
Second layer –



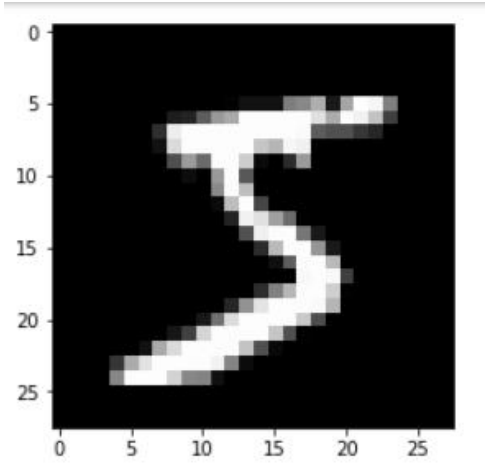
Third layer –



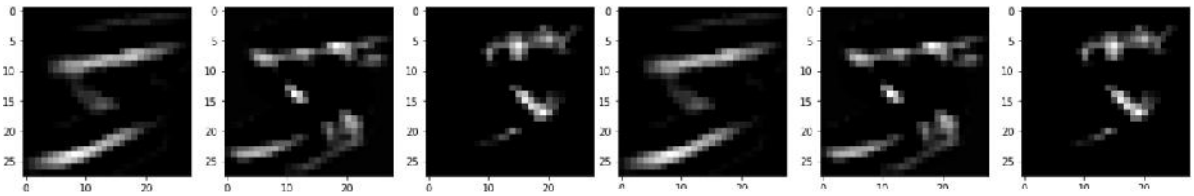
Fourth layer –



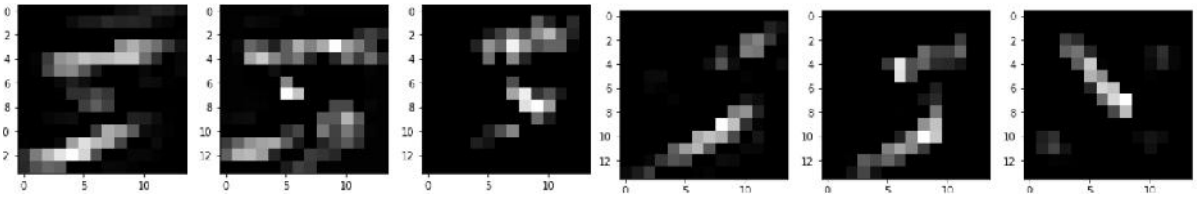
Input –



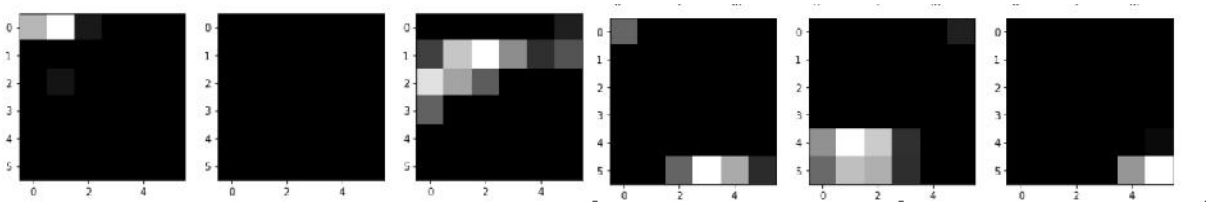
First layer –



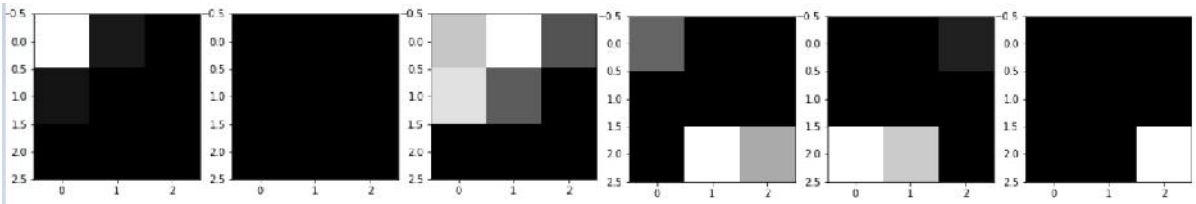
Second layer –



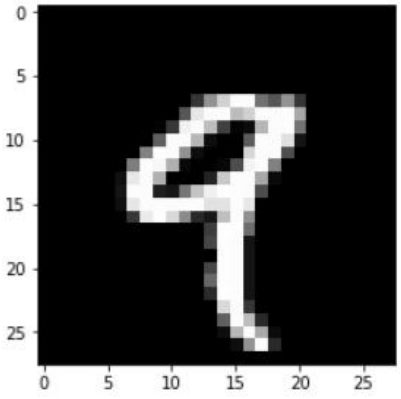
Third layer –



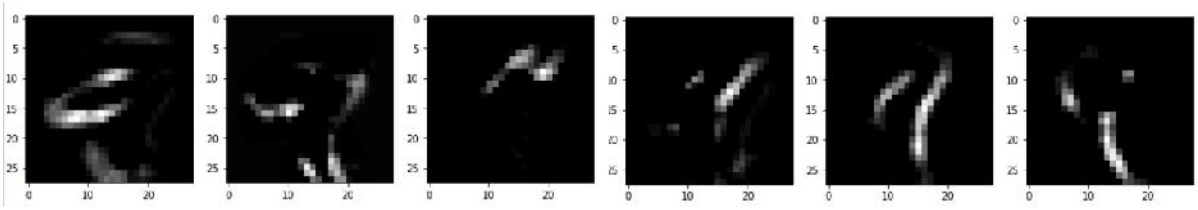
Fourth layer –



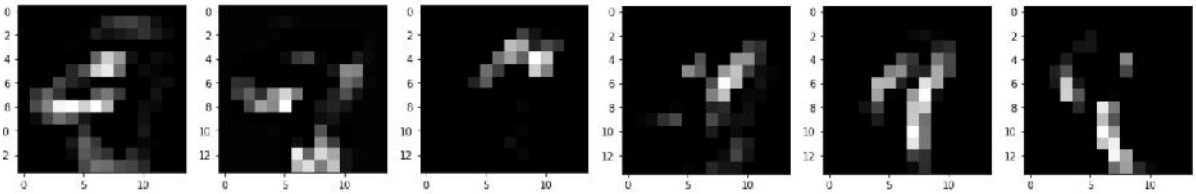
Input –



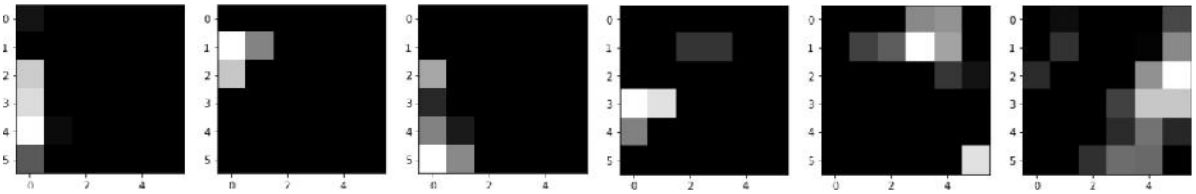
First layer –



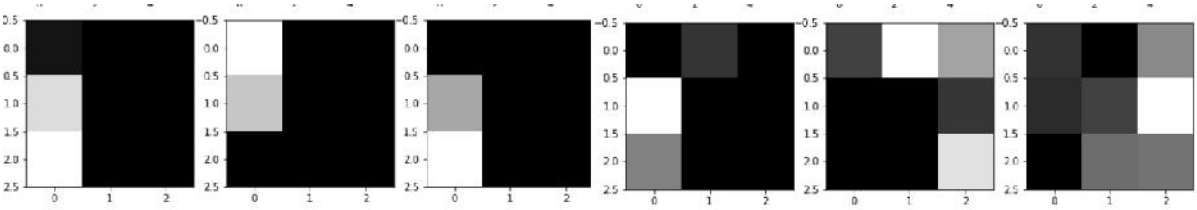
Second layer –



Third layer –



Fourth layer –



Discussion –

In this setting I have tried to vary the kernel sizes to determine the accuracies and below are my observations -

Discussion about the variations in the kernel size –

- First I tried to vary the kernel size by increasing it and I found that the accuracy increased but not very much.
- But the details in each of the layer was preserved as you can above in the comparison of the different layers.
- With the increase in the kernel size, the accuracy reached saturation and not much changes were observed.

Setting – 4 – both filter and kernel size increased

1.Parameters used -

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	16	9	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	26	9	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	Types
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	Types/Values/Size
Batch	32
Epoch	10

Testing Stage –

Parameters	Types/Values/Size
Batch	32

2. Accuracy Report –

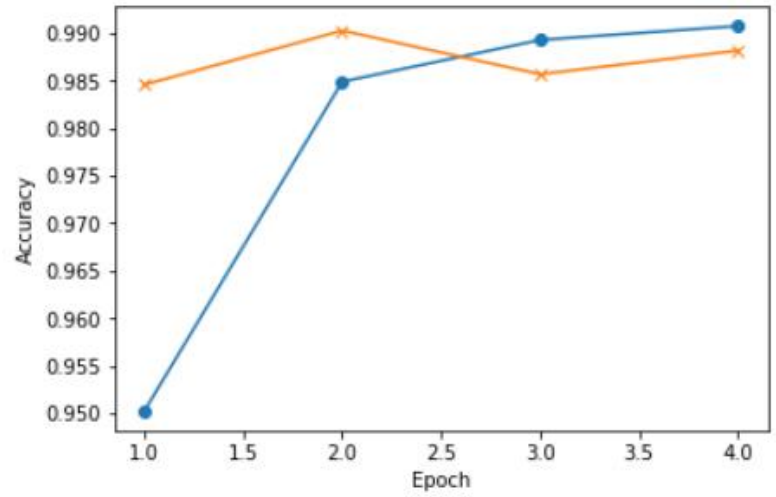
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 129s 2ms/step - loss: 0.1558 - acc: 0.9502 - val_loss: 0.0503 - val_acc: 0.9846
Epoch 2/10
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0530 - acc: 0.9849 - val_loss: 0.0305 - val_acc: 0.9903
Epoch 3/10
60000/60000 [==============================]  125s 2ms/step  loss: 0.0393  acc: 0.9893  val_loss: 0.0537  val_acc: 0.9857
Epoch 4/10
60000/60000 [==============================] - 124s 2ms/step - loss: 0.0344 - acc: 0.9908 - val_loss: 0.0448 - val_acc: 0.9882
Epoch 00004: early stopping
```

```
10000/10000 [==============================] - 8s 757us/step

[0.035658400966845055, 0.9909]
```

Accuracy on training data – 99.08%

Accuracy on testing data – 98.82%

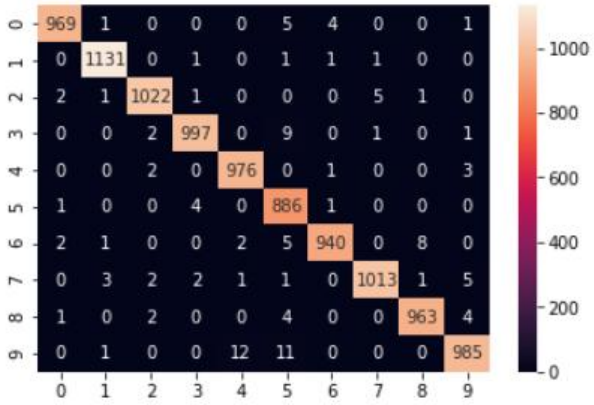
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 969,    1,    0,    0,    0,    5,    4,    0,    0,    1],
       [   0, 1131,    0,    1,    0,    1,    1,    1,    0,    0],
       [   2,    1, 1022,    1,    0,    0,    0,    5,    1,    0],
       [   0,    0,    2,  997,    0,    9,    0,    1,    0,    1],
       [   0,    0,    2,    0,  976,    0,    1,    0,    0,    3],
       [   1,    0,    0,    4,    0,  886,    1,    0,    0,    0],
       [   2,    1,    0,    0,    2,    5,  940,    0,    8,    0],
       [   0,    3,    2,    2,    1,    1,    0, 1013,    1,    5],
       [   1,    0,    2,    0,    0,    4,    0,    0,  963,    4],
       [   0,    1,    0,    0,   12,   11,    0,    0,    0,  985]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report

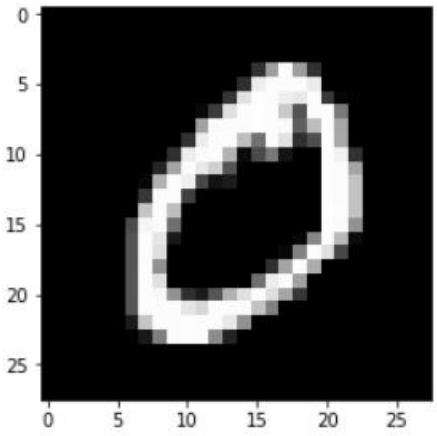
	precision	recall	f1-score	support
class 0	0.99	0.99	0.99	980
class 1	0.99	1.00	1.00	1135
class 2	0.99	0.99	0.99	1032
class 3	0.99	0.99	0.99	1010
class 4	0.98	0.99	0.99	982
class 5	0.96	0.99	0.98	892
class 6	0.99	0.98	0.99	958
class 7	0.99	0.99	0.99	1028
class 8	0.99	0.99	0.99	974
class 9	0.99	0.98	0.98	1009
avg / total	0.99	0.99	0.99	10000

7. Model Summary-

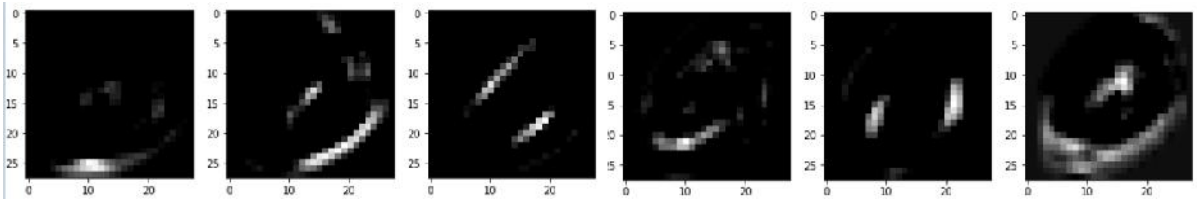
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 28, 28, 16)        1312
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 6, 6, 26)          33722
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 3, 3, 26)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 234)               0
_________________________________________________________________
dense_25 (Dense)             (None, 120)               28200
_________________________________________________________________
dense_26 (Dense)             (None, 80)                9680
_________________________________________________________________
dense_27 (Dense)             (None, 10)                810
=================================================================
Total params: 73,724
Trainable params: 73,724
Non-trainable params: 0
_________________________________________________________________
```

8. Outputs after each layer-

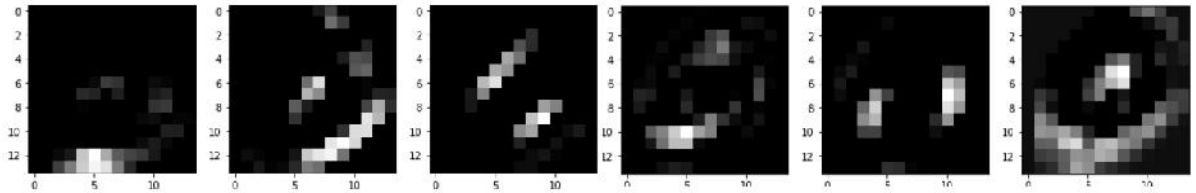
Input—



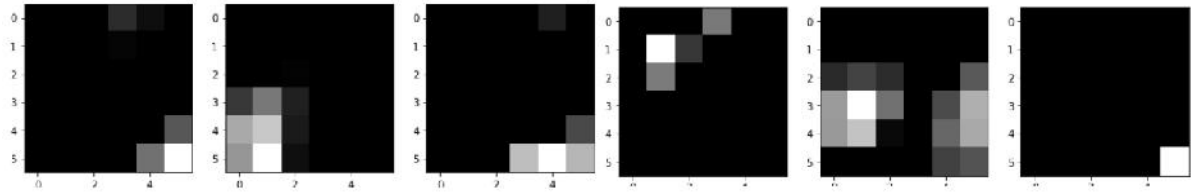
First layer –



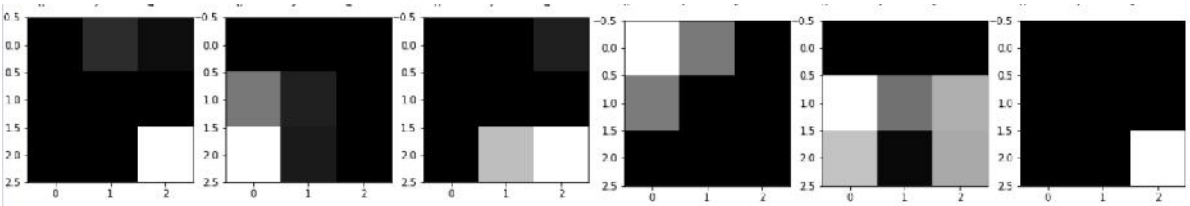
Second layer –



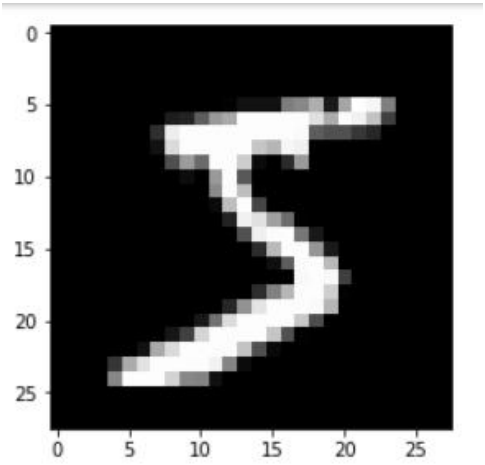
Third layer –



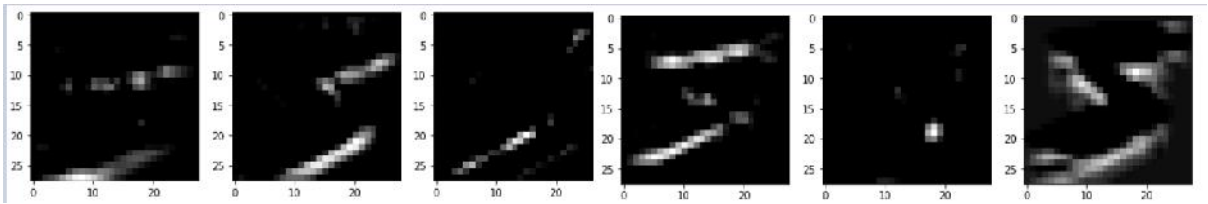
Fourth layer –



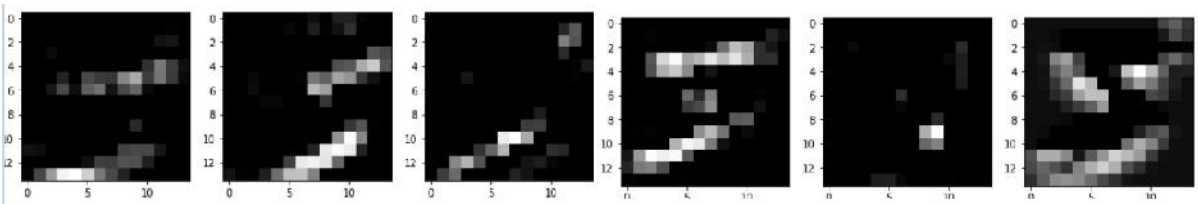
Input –



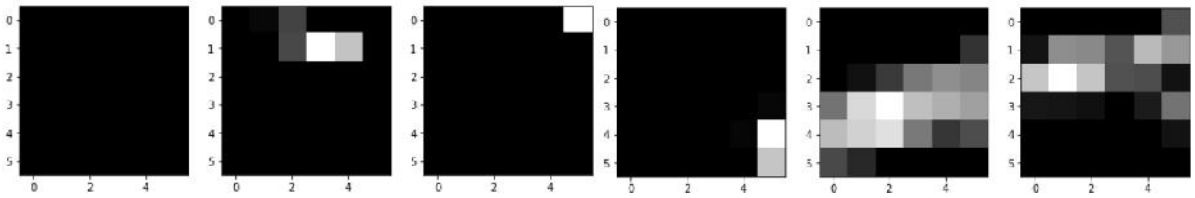
First layer –



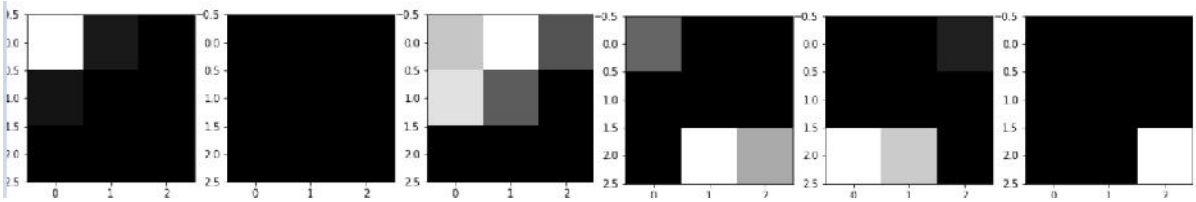
Second layer –



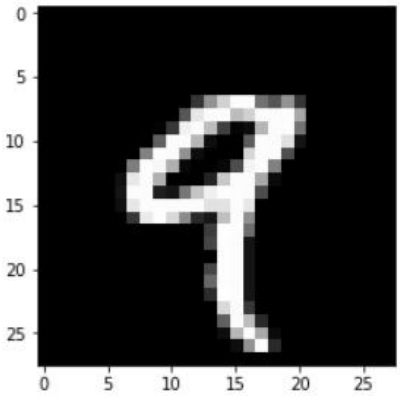
Third layer –



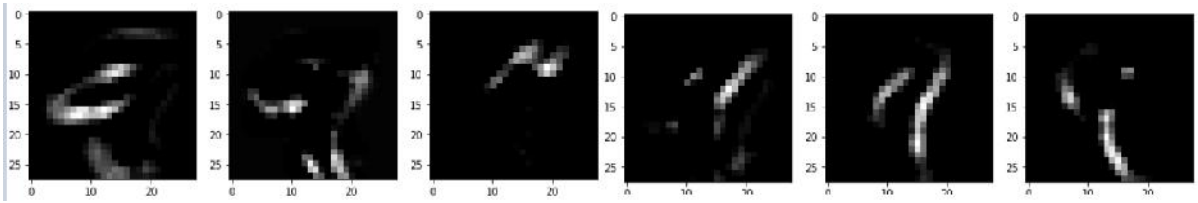
Fourth layer –



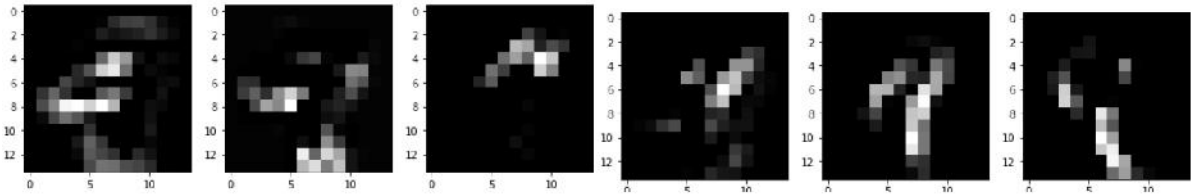
Input –



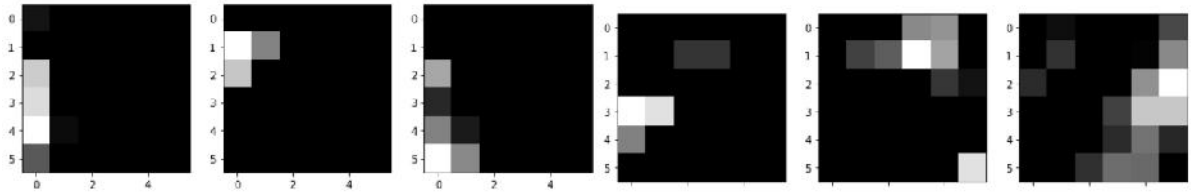
First layer –



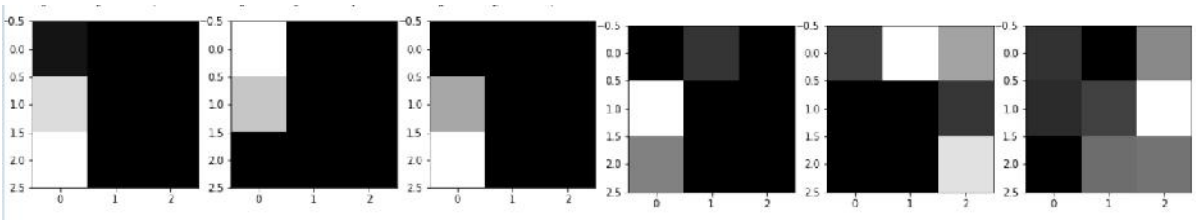
Second layer –



Third layer –



Fourth layer –



Discussion –

- **In this setting I have tried to increase both the filter size and the kernel size.**

- I observed that the accuracy was a little more than what was observed before.
- The increase in the filter size tries to increase the accuracy by a significant amount.
- But the increase in the kernel size, will make the accuracy go into saturation.
- Therefore, while the increase in the filter size will try to increase the accuracy of the model, the kernel size will try to stabilise it.
- Also, with the increase in both the kernel and the filter sizes, the finer details of the images are further lost.

Setting – 5 – variations the dense sizes

Adding Layers Stage -

Different Layers	**Filter Size**	**Kernel Size**	**Pool Size**	**Padding**	**Activation**	**No. of Connections/Neurons**
Convolutional2D	16	9	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	26	9	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	240
Dense 2	N/A	N/A	N/A	N/A	N/A	160
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	**Types**
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	**Types/Values/Size**
Batch	32
Epoch	10

Testing Stage –

Parameters	Types/Values/Size
Batch	32

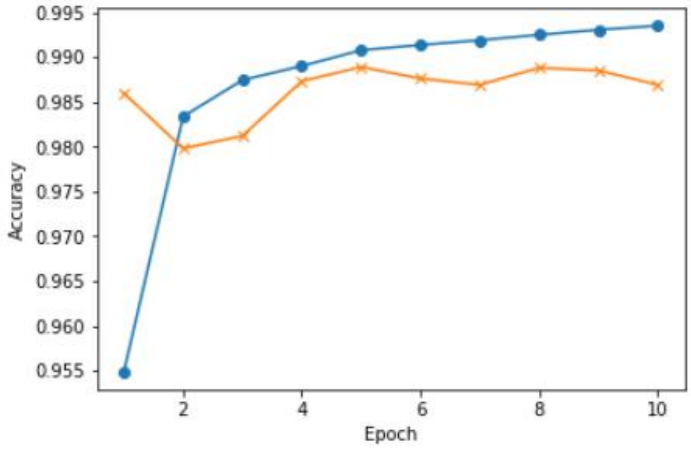
2. Accuracy report

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 49s 818us/step - loss: 0.1455 - acc: 0.9549 - val_loss: 0.0456 - val_acc: 0.9860
Epoch 2/10
60000/60000 [==============================] - 49s 814us/step - loss: 0.0559 - acc: 0.9834 - val_loss: 0.0613 - val_acc: 0.9798
Epoch 3/10
60000/60000 [==============================] - 49s 819us/step - loss: 0.0437 - acc: 0.9874 - val_loss: 0.0573 - val_acc: 0.9812
Epoch 4/10
60000/60000 [==============================] - 49s 824us/step - loss: 0.0383 - acc: 0.9890 - val_loss: 0.0450 - val_acc: 0.9873
Epoch 5/10
60000/60000 [==============================]  - 52s 873us/step - loss: 0.0338 - acc: 0.9908 - val_loss: 0.0434 - val_acc: 0.9889
Epoch 6/10
60000/60000 [==============================] - 52s 868us/step - loss: 0.0316 - acc: 0.9913 - val_loss: 0.0448 - val_acc: 0.9876
Epoch 7/10
60000/60000 [==============================] - 52s 862us/step - loss: 0.0302 - acc: 0.9919 - val_loss: 0.0482 - val_acc: 0.9869
Epoch 8/10
60000/60000 [==============================] - 51s 852us/step - loss: 0.0292 - acc: 0.9925 - val_loss: 0.0435 - val_acc: 0.9888
Epoch 9/10
60000/60000 [==============================] - 54s 898us/step - loss: 0.0272 - acc: 0.9930 - val_loss: 0.0508 - val_acc: 0.9885
Epoch 10/10
60000/60000 [==============================] - 48s 798us/step - loss: 0.0249 - acc: 0.9935 - val_loss: 0.0632 - val_acc: 0.9869

10000/10000 [==============================] - 3s 347us/step

[0.06320498333303276, 0.9869]
```

Accuracy on training data – 99.35%

Accuracy on testing data – 98.69%

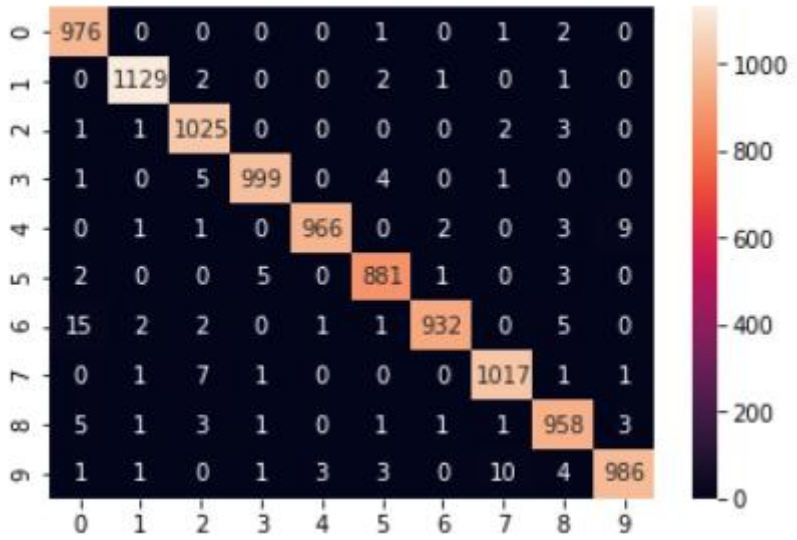
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 976,    0,    0,    0,    0,    1,    0,    1,    2,    0],
       [   0, 1129,    2,    0,    0,    2,    1,    0,    1,    0],
       [   1,    1, 1025,    0,    0,    0,    0,    2,    3,    0],
       [   1,    0,    5,  999,    0,    4,    0,    1,    0,    0],
       [   0,    1,    1,    0,  966,    0,    2,    0,    3,    9],
       [   2,    0,    0,    5,    0,  881,    1,    0,    3,    0],
       [  15,    2,    2,    0,    1,    1,  932,    0,    5,    0],
       [   0,    1,    7,    1,    0,    0,    0, 1017,    1,    1],
       [   5,    1,    3,    1,    0,    1,    1,    1,  958,    3],
       [   1,    1,    0,    1,    3,    3,    0,   10,    4,  986]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report

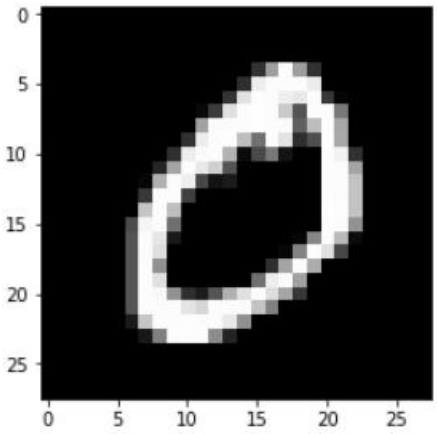
	precision	recall	f1-score	support
class 0	0.98	1.00	0.99	980
class 1	0.99	0.99	0.99	1135
class 2	0.98	0.99	0.99	1032
class 3	0.99	0.99	0.99	1010
class 4	1.00	0.98	0.99	982
class 5	0.99	0.99	0.99	892
class 6	0.99	0.97	0.98	958
class 7	0.99	0.99	0.99	1028
class 8	0.98	0.98	0.98	974
class 9	0.99	0.98	0.98	1009
avg / total	0.99	0.99	0.99	10000

7. Model Summary-

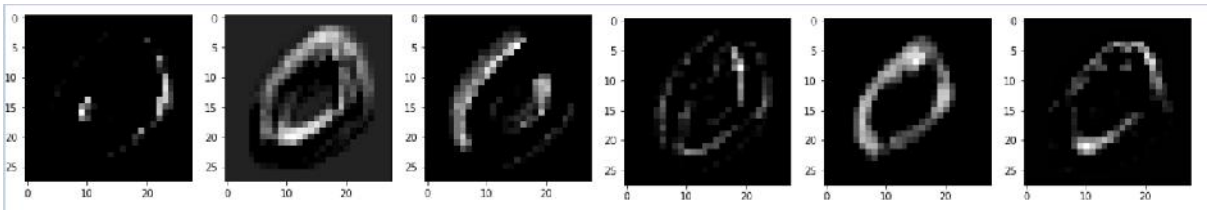
Layer (type)	Output Shape	Param #
conv2d_31 (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d_30 (MaxPooling	(None, 14, 14, 6)	0
conv2d_32 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_31 (MaxPooling	(None, 5, 5, 16)	0
flatten_14 (Flatten)	(None, 400)	0
dense_40 (Dense)	(None, 240)	96240
dense_41 (Dense)	(None, 160)	38560
dense_42 (Dense)	(None, 10)	1610

Total params: 138,982
Trainable params: 138,982
Non-trainable params: 0

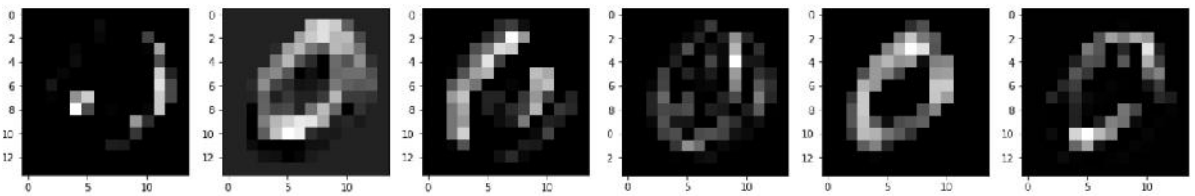
8. Outputs after each layer-Input--



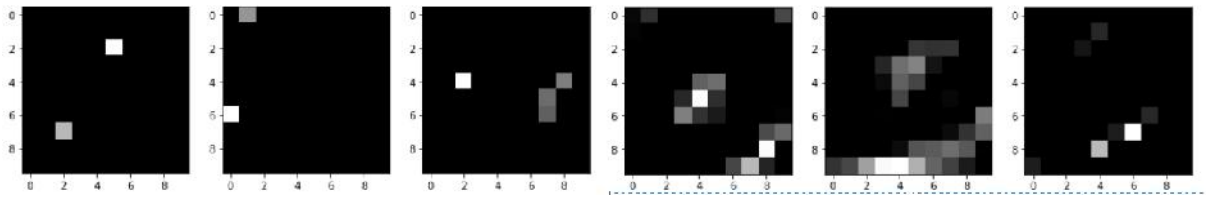
First layer –



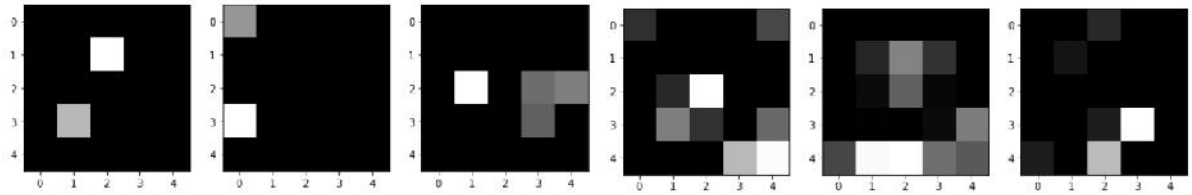
Second layer –



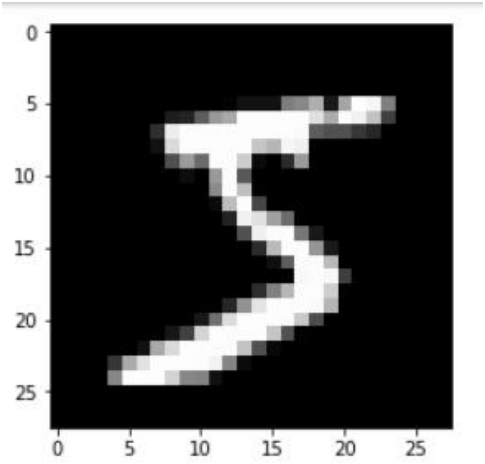
Third layer –



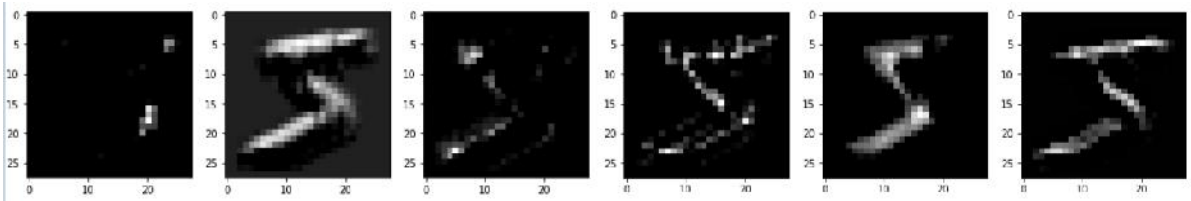
Fourth layer –



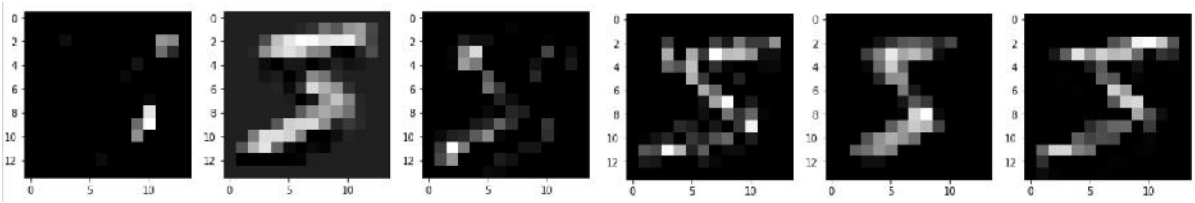
Input –



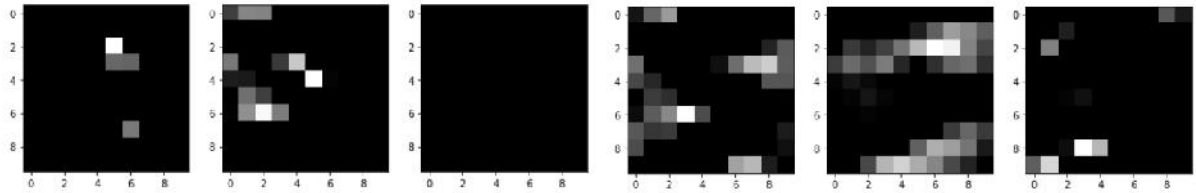
First layer –



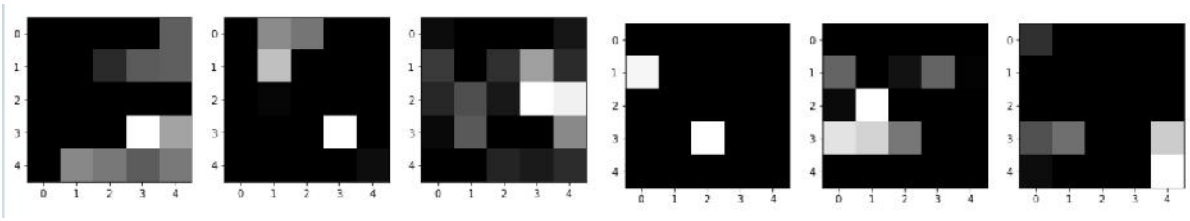
Second layer –



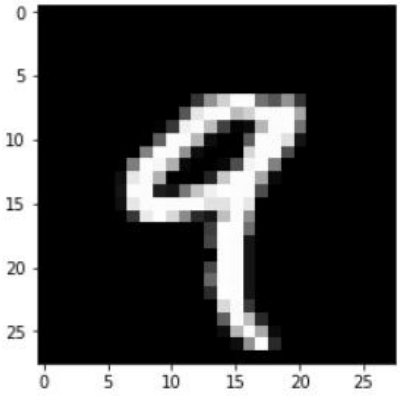
Third layer –



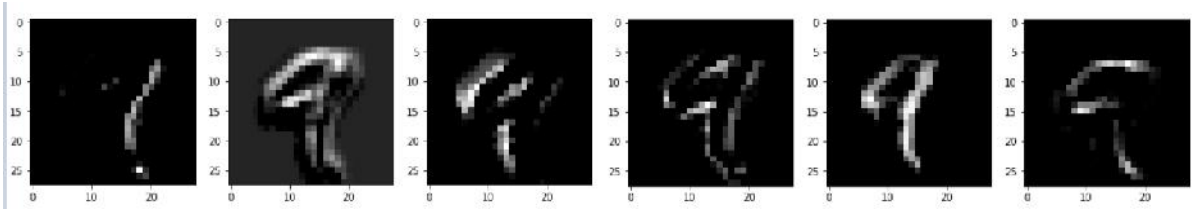
Fourth layer –



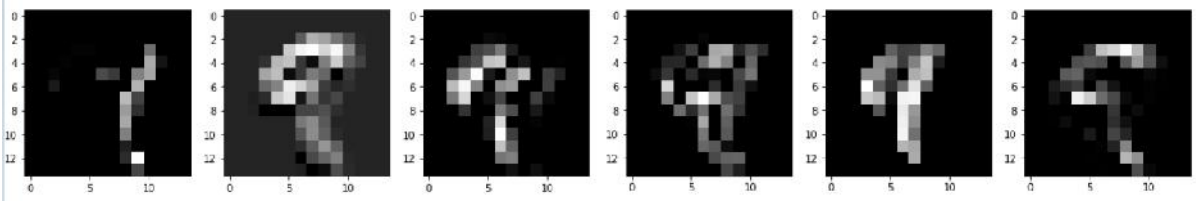
Input –



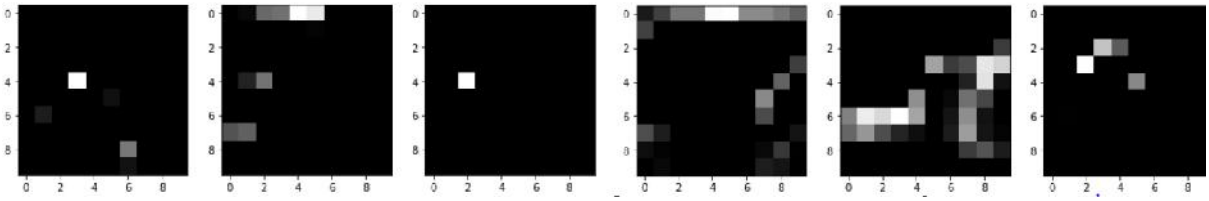
First layer –



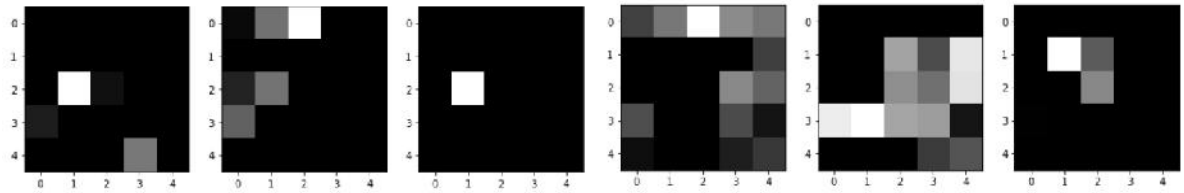
Second layer –



Third layer –



Fourth layer –



Discussion –

Discussion about variations in the dense sizes –

- Dense layer usually tells us the number of hidden layers that are responsible before the classificaton.
- Increasing, the dense layer size definitely put an increase to the accuracy as the number of hidden layers responsible were a lot and this helped in increasing the accuracy.

<u>Setting – 6 – increasing filter, kernel and dense</u>

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	16	9	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	26	9	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	240
Dense 2	N/A	N/A	N/A	N/A	N/A	160
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	Types
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	Types/Values/Size
Batch	32
Epoch	10

Testing Stage –

Parameters	Types/Values/Size
Batch	32

2. Accuracy report

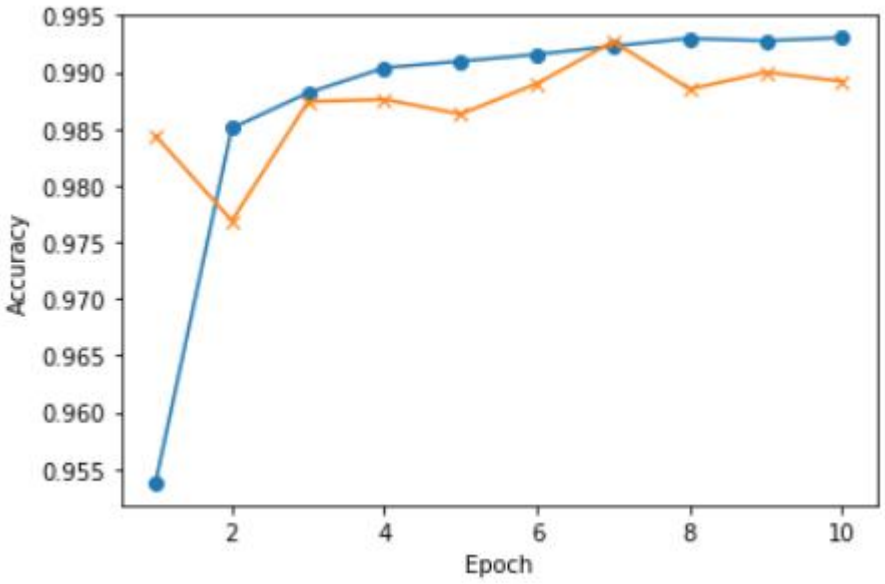
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 116s 2ms/step - loss: 0.1482 - acc: 0.9538 - val_loss: 0.0455 - val_acc: 0.9844
Epoch 2/10
60000/60000 [==============================] - 116s 2ms/step - loss: 0.0530 - acc: 0.9851 - val_loss: 0.0763 - val_acc: 0.9769
Epoch 3/10
60000/60000 [==============================] - 120s 2ms/step - loss: 0.0439 - acc: 0.9882 - val_loss: 0.0470 - val_acc: 0.9874
Epoch 4/10
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0379 - acc: 0.9904 - val_loss: 0.0514 - val_acc: 0.9876
Epoch 5/10
60000/60000 [==============================]  114s 2ms/step  loss: 0.0398  acc: 0.9910  val_loss: 0.0495  val_acc: 0.9863
Epoch 6/10
60000/60000 [==============================] - 115s 2ms/step - loss: 0.0392 - acc: 0.9916 - val_loss: 0.0572 - val_acc: 0.9890
Epoch 7/10
60000/60000 [==============================] - 115s 2ms/step - loss: 0.0367 - acc: 0.9923 - val_loss: 0.0427 - val_acc: 0.9927
Epoch 8/10
60000/60000 [==============================] - 116s 2ms/step - loss: 0.0390 - acc: 0.9930 - val_loss: 0.0512 - val_acc: 0.9885
Epoch 9/10
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0430 - acc: 0.9928 - val_loss: 0.0701 - val_acc: 0.9900
Epoch 10/10
60000/60000 [==============================] - 115s 2ms/step - loss: 0.0402 - acc: 0.9930 - val_loss: 0.0705 - val_acc: 0.9892
```

```
10000/10000 [==============================] - 7s 735us/step

[0.07046130691131161, 0.9892]
```

Accuracy on training data – 99.3%

Accuracy on testing data – 98.92%

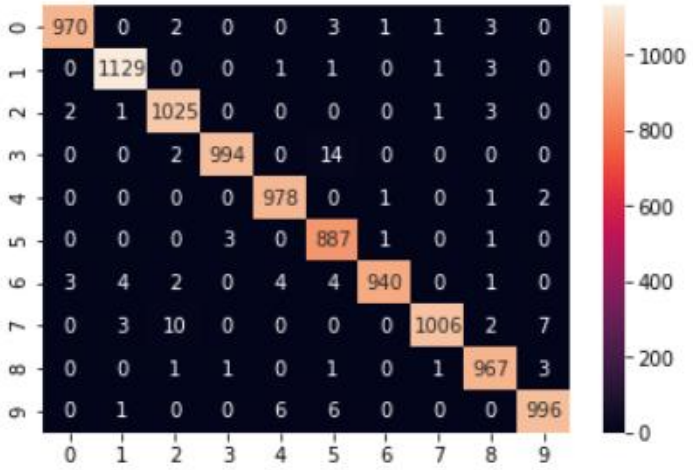
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 970,    0,    2,    0,    0,    3,    1,    1,    3,    0],
       [   0, 1129,    0,    0,    1,    1,    0,    1,    3,    0],
       [   2,    1, 1025,    0,    0,    0,    0,    1,    3,    0],
       [   0,    0,    2,  994,    0,   14,    0,    0,    0,    0],
       [   0,    0,    0,    0,  978,    0,    1,    0,    1,    2],
       [   0,    0,    0,    3,    0,  887,    1,    0,    1,    0],
       [   3,    4,    2,    0,    4,    4,  940,    0,    1,    0],
       [   0,    3,   10,    0,    0,    0,    0, 1006,    2,    7],
       [   0,    0,    1,    1,    0,    1,    0,    1,  967,    3],
       [   0,    1,    0,    0,    6,    6,    0,    0,    0,  996]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report-

	precision	recall	f1-score	support
class 0	0.99	0.99	0.99	980
class 1	0.99	0.99	0.99	1135
class 2	0.98	0.99	0.99	1032
class 3	1.00	0.98	0.99	1010
class 4	0.99	1.00	0.99	982
class 5	0.97	0.99	0.98	892
class 6	1.00	0.98	0.99	958
class 7	1.00	0.98	0.99	1028
class 8	0.99	0.99	0.99	974
class 9	0.99	0.99	0.99	1009
avg / total	0.99	0.99	0.99	10000

7. Model Summary-

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 28, 28, 16)        1312
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 6, 6, 26)          33722
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 3, 3, 26)          0
_________________________________________________________________
flatten_15 (Flatten)         (None, 234)               0
_________________________________________________________________
dense_43 (Dense)             (None, 240)               56400
_________________________________________________________________
dense_44 (Dense)             (None, 160)               38560
_________________________________________________________________
dense_45 (Dense)             (None, 10)                1610
=================================================================
Total params: 131,604
Trainable params: 131,604
Non-trainable params: 0
_________________________________________________________________
```

Discussion -

- Increasing all the three layers definitely put a significant increase to the accuracy.
- The filter and dense layers try to increase the number of features extracted and the number of hidden layers but the kernel size kind of stabilizes the accuracy and holds it down by over fitting
- Therefore, this is a very good model to consider for the training.

<u>Setting – 7 – increase in epoch size – example of over-fitting</u>

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	6	5	N/A	same	relu	N/A

Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	16	5	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	**Types**
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

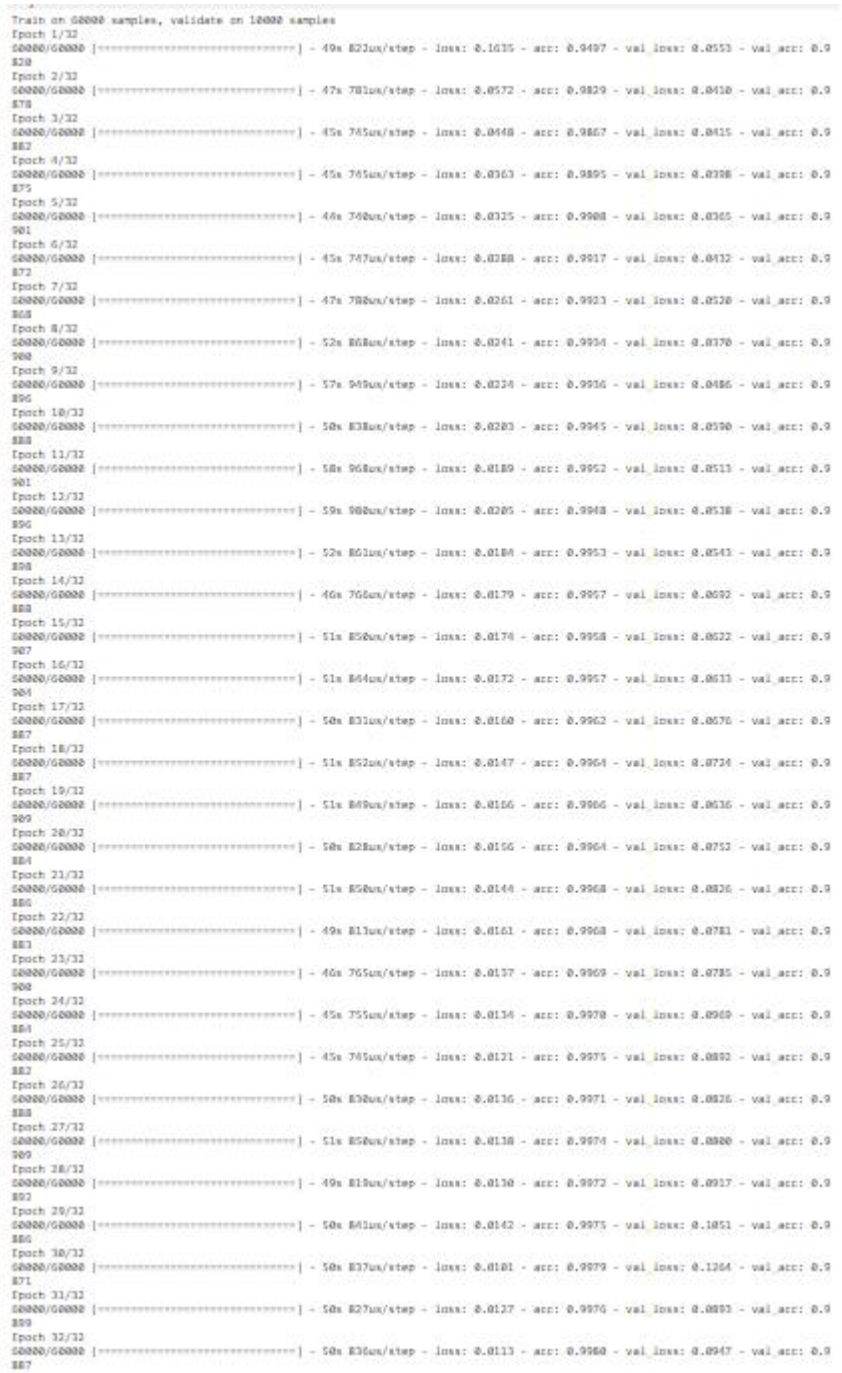
Training Stage –

Parameters	**Types/Values/Size**
Batch	32
Epoch	32

Testing Stage –

Parameters	**Types/Values/Size**
Batch	32

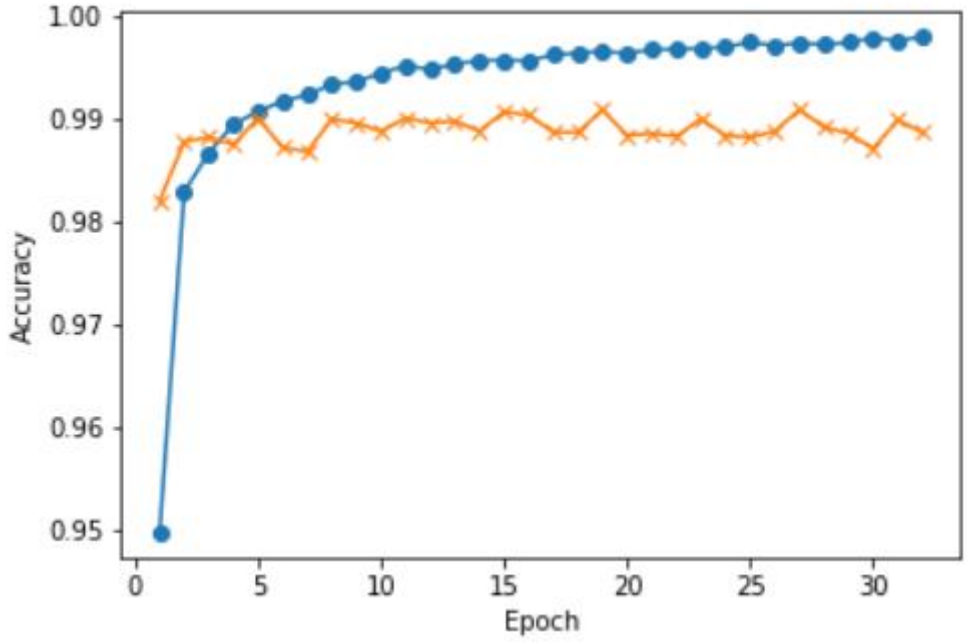
2. Accuracy report



Accuracy on training data – 99.8%

Accuracy on testing data – 98.87%

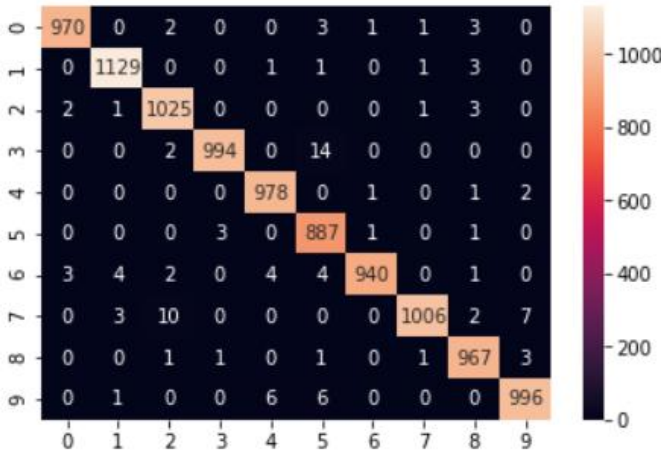
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 970,    0,    2,    0,    0,    3,    1,    1,    3,    0],
       [   0, 1129,    3,    0,    1,    1,    0,    1,    3,    0],
       [   2,    1, 1025,    0,    0,    0,    0,    1,    3,    0],
       [   0,    0,    2,  994,    0,   14,    0,    0,    0,    0],
       [   0,    0,    0,    0,  978,    0,    1,    0,    1,    2],
       [   0,    0,    0,    3,    0,  887,    1,    0,    1,    0],
       [   3,    4,    2,    0,    4,    4,  940,    0,    1,    0],
       [   0,    3,   10,    0,    0,    0,    0, 1006,    2,    7],
       [   0,    0,    1,    1,    0,    1,    0,    1,  967,    3],
       [   0,    1,    0,    0,    6,    6,    0,    0,    0,  996]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report-

	precision	recall	f1-score	support
class 0	0.99	0.99	0.99	980
class 1	0.99	0.99	0.99	1135
class 2	0.98	0.99	0.99	1032
class 3	1.00	0.98	0.99	1010
class 4	0.99	1.00	0.99	982
class 5	0.97	0.99	0.98	892
class 6	1.00	0.98	0.99	958
class 7	1.00	0.98	0.99	1028
class 8	0.99	0.99	0.99	974
class 9	0.99	0.99	0.99	1009
avg / total	0.99	0.99	0.99	10000

7. Model Summary-

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 28, 28, 16)        1312
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 6, 6, 26)          33722
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 3, 3, 26)          0
_________________________________________________________________
flatten_15 (Flatten)         (None, 234)               0
_________________________________________________________________
dense_43 (Dense)             (None, 240)               56400
_________________________________________________________________
dense_44 (Dense)             (None, 160)               38560
_________________________________________________________________
dense_45 (Dense)             (None, 10)                1610
=================================================================
Total params: 131,604
Trainable params: 131,604
Non-trainable params: 0
_________________________________________________________________
```

Discussion –

- **Discussion about the over fittting –**
- In our training model, giving the epoch size increased the training accuracy to a very huge extent.
- But it went through over fitting and the testing accuracy is much lesser compared to the training accuracy.
- I ran the training for 32 Epochs and this was the output that I got the over fitting of the model
- As you can see that the testing accuracy(orange line) is significantly lesser and keeps getting worse.
- But the training accuracy, with the model almost reaching a 100% is very good.

- This is happening because the model has learnt and gotten really used to the training dataset that the noise and the fluctuations that are present are also taken down and learnt as features and are used for learning.
- Therefore, the testing data performed very badly as shown in the figure.
- The epoch accuracy curve shows the best way to see how the testing data performed against the training data.

Discussion about Batch Size –

- Batch size determines how many inputs were considered before updation.
- With increase in the batch size, the model perfomed really well as the number of inputs considered before updation was a lot and that the accuracy significanty increased.
- But on the test data, since they were already so little, were not given much prominence before updation.

Setting – 8-- Adams

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	16	9	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	26	9	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	Types
Optimizer	Adam
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	Types/Values/Size
Batch	32
Epoch	10

Testing Stage –

Parameters	Types/Values/Size
Batch	32

2. Accuracy report

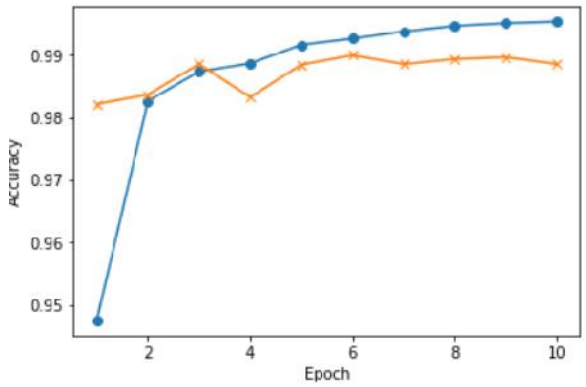
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 118s 2ms/step - loss: 0.1669 - acc: 0.9476 - val_loss: 0.0590 - val_acc: 0.9822
Epoch 2/10
60000/60000 [==============================] - 116s 2ms/step - loss: 0.0554 - acc: 0.9827 - val_loss: 0.0507 - val_acc: 0.9837
Epoch 3/10
60000/60000 [==============================] - 116s 2ms/step - loss: 0.0403 - acc: 0.9874 - val_loss: 0.0348 - val_acc: 0.9880
Epoch 4/10
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0347 - acc: 0.9887 - val_loss: 0.0504 - val_acc: 0.9833
Epoch 5/10
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0262 - acc: 0.9917 - val_loss: 0.0409 - val_acc: 0.9885
Epoch 6/10
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0240 - acc: 0.9927 - val_loss: 0.0329 - val_acc: 0.9900
Epoch 7/10
60000/60000 [==============================] - 125s 2ms/step - loss: 0.0203 - acc: 0.9938 - val_loss: 0.0383 - val_acc: 0.9880
Epoch 8/10
60000/60000 [==============================] - 123s 2ms/step - loss: 0.0174 - acc: 0.9947 - val_loss: 0.0386 - val_acc: 0.9894
Epoch 9/10
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0164 - acc: 0.9951 - val_loss: 0.0397 - val_acc: 0.9897
Epoch 10/10
60000/60000 [==============================] - 120s 2ms/step - loss: 0.0146 - acc: 0.9954 - val_loss: 0.0519 - val_acc: 0.9886
```

```
10000/10000 [==============================] - 8s 763us/step

[0.05192580051602745, 0.9886]
```

Accuracy on training data – 99.54%

Accuracy on testing data – 98.86%

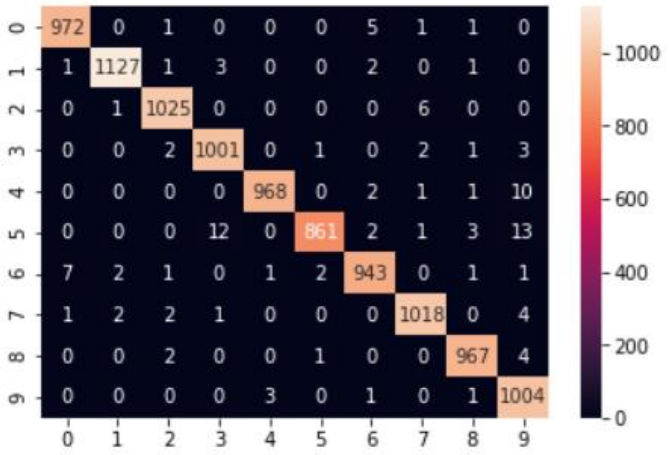
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 972,    0,    1,    0,    0,    0,    5,    1,    1,    0],
       [   1, 1127,    1,    3,    0,    0,    2,    0,    1,    0],
       [   0,    1, 1025,    0,    0,    0,    0,    6,    0,    0],
       [   0,    0,    2, 1001,    0,    1,    0,    2,    1,    3],
       [   0,    0,    0,    0,  968,    0,    2,    1,    1,   10],
       [   0,    0,    0,   12,    0,  861,    2,    1,    3,   13],
       [   7,    2,    1,    0,    1,    2,  943,    0,    1,    1],
       [   1,    2,    2,    1,    0,    0,    0, 1018,    0,    4],
       [   0,    0,    2,    0,    0,    1,    0,    0,  967,    4],
       [   0,    0,    0,    0,    3,    0,    1,    0,    1, 1004]],
      dtype=int64)
```

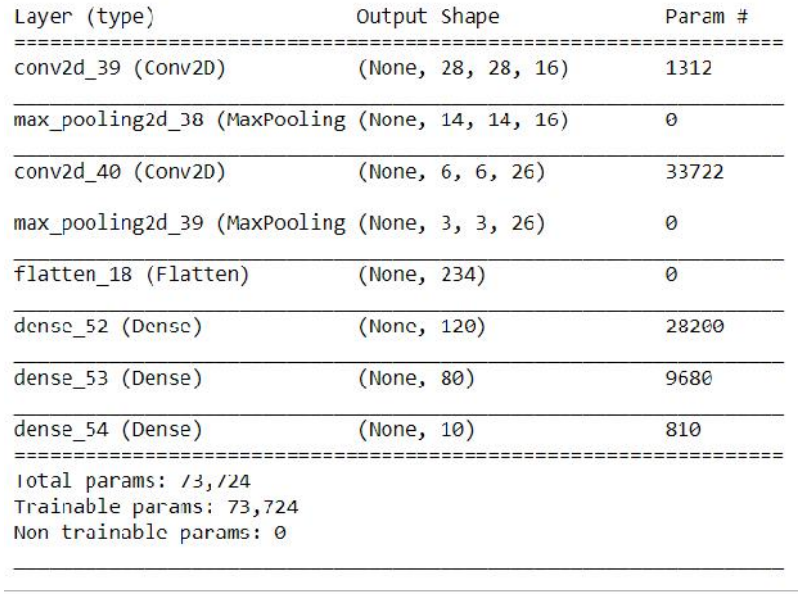
5. Heatmap indicating the confusion matrix-



6. Classification Report-

	precision	recall	f1-score	support
class 0	0.99	0.99	0.99	980
class 1	1.00	0.99	0.99	1135
class 2	0.99	0.99	0.99	1032
class 3	0.98	0.99	0.99	1010
class 4	1.00	0.99	0.99	982
class 5	1.00	0.97	0.98	892
class 6	0.99	0.98	0.99	958
class 7	0.99	0.99	0.99	1028
class 8	0.99	0.99	0.99	974
class 9	0.97	1.00	0.98	1009
avg / total	0.99	0.99	0.99	10000

7. Model Summary-



Discussion –

Discussion about Optimizers –

- Using Adams optimizer along with setting a learning rate to 0.0001 is really good and performs well.
- The accuracy has also increased significantly showing that the adam optimiser is better than the rmsprop
- This is true because in the rmsprop optimizer the updation is done for every parameter.
- Adam optimizer combines both the disadvantages of both the rmsprop and the Momentum optimizer.
- Because in Adam optimizer we consider the exponential average and the squre of the gradients of the parameters.
- The epoch- accuracy curve shows this and the way the accuracy has increased over it.

<u>Setting – 9 – Batch Normalisation and Drop out layers-- best</u>

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons
Convolutional2D	16	9	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	26	9	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A

Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	**Types**
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	**Types/Values/Size**
Batch	32
Epoch	10

Testing Stage –

Parameters	**Types/Values/Size**
Batch	32

2. Accuracy report

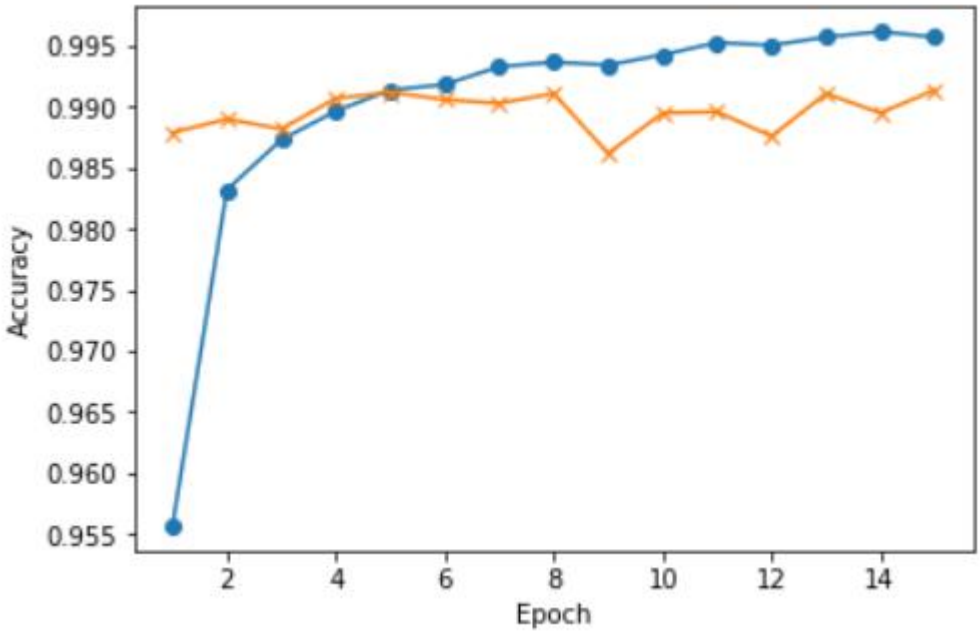
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 [==============================] - 361s 6ms/step - loss: 0.1440 - acc: 0.9557 - val_loss: 0.0438 - val_acc: 0.9879
Epoch 2/15
60000/60000 [==============================]  380s 6ms/step  loss: 0.0571  acc: 0.9832  val_loss: 0.0346  val_acc: 0.9890
Epoch 3/15
60000/60000 [==============================] - 383s 6ms/step - loss: 0.0453 - acc: 0.9873 - val_loss: 0.0349 - val_acc: 0.9882:
0.98 - ETA: 1s - loss: 0.0452 - a
Epoch 4/15
60000/60000 [==============================]  378s 6ms/step  loss: 0.0368  acc: 0.9897  val_loss: 0.0326  val_acc: 0.9907
Epoch 5/15
60000/60000 [==============================] - 376s 6ms/step - loss: 0.0322 - acc: 0.9914 - val_loss: 0.0315 - val_acc: 0.99129
1
Epoch 6/15
60000/60000 [==============================] - 378s 6ms/step - loss: 0.0288 - acc: 0.9918 - val_loss: 0.0360 - val_acc: 0.9906
Epoch 7/15
60000/60000 [==============================] - 376s 6ms/step - loss: 0.0236 - acc: 0.9933 - val_loss: 0.0419 - val_acc: 0.9903
Epoch 8/15
60000/60000 [==============================]  376s 6ms/step  loss: 0.0224  acc: 0.9937  val_loss: 0.0383  val_acc: 0.9911
Epoch 9/15
60000/60000 [==============================] - 379s 6ms/step - loss: 0.0243 - acc: 0.9934 - val_loss: 0.0628 - val_acc: 0.9862
Epoch 10/15
60000/60000 [==============================]  382s 6ms/step  loss: 0.0208  acc: 0.9943  val_loss: 0.0381  val_acc: 0.9895
Epoch 11/15
60000/60000 [==============================] - 382s 6ms/step - loss: 0.0177 - acc: 0.9953 - val_loss: 0.0482 - val_acc: 0.9896
Epoch 12/15
60000/60000 [==============================] - 345s 6ms/step - loss: 0.0175 - acc: 0.9950 - val_loss: 0.0526 - val_acc: 0.9876
Epoch 13/15
60000/60000 [==============================] - 223s 4ms/step - loss: 0.0159 - acc: 0.9957 - val_loss: 0.0398 - val_acc: 0.9911
Epoch 14/15
60000/60000 [==============================] - 229s 4ms/step - loss: 0.0142 - acc: 0.9962 - val_loss: 0.0638 - val_acc: 0.9895
Epoch 15/15
60000/60000 [==============================] - 226s 4ms/step - loss: 0.0171 - acc: 0.9957 - val_loss: 0.0424 - val_acc: 0.9914
```

```
10000/10000 [==============================] - 20s 2ms/step

[0.042399156506391776, 0.9914]
```

Accuracy on training data – 99.62%

Accuracy on testing data – 99.14%

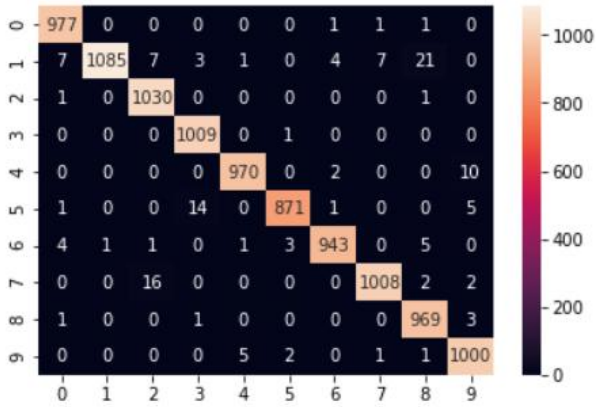
3. Epoch – Accuracy Curve



4. Confusion Matrix

```
array([[ 977,    0,    0,    0,    0,    0,    1,    1,    1,    0],
       [   7, 1085,    7,    3,    1,    0,    4,    7,   21,    0],
       [   1,    0, 1030,    0,    0,    0,    0,    0,    1,    0],
       [   0,    0,    0, 1009,    0,    1,    0,    0,    0,    0],
       [   0,    0,    0,    0,  970,    0,    2,    0,    0,   10],
       [   1,    0,    0,   14,    0,  871,    1,    0,    0,    5],
       [   4,    1,    1,    0,    1,    3,  943,    0,    5,    0],
       [   0,    0,   16,    0,    0,    0,    0, 1008,    2,    2],
       [   1,    0,    0,    1,    0,    0,    0,    0,  969,    3],
       [   0,    0,    0,    0,    5,    2,    0,    1,    1, 1000]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report

	precision	recall	f1-score	support
class 0	0.99	1.00	0.99	980
class 1	1.00	0.96	0.98	1135
class 2	0.98	1.00	0.99	1032
class 3	0.98	1.00	0.99	1010
class 4	0.99	0.99	0.99	982
class 5	0.99	0.98	0.98	892
class 6	0.99	0.98	0.99	958
class 7	0.99	0.98	0.99	1028
class 8	0.97	0.99	0.98	974
class 9	0.98	0.99	0.99	1009
avg / total	0.99	0.99	0.99	10000

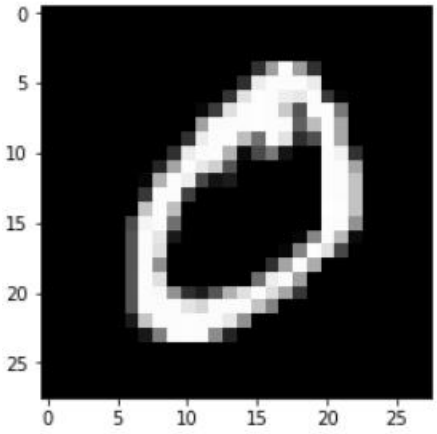
7. Model Summary-

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 16)	416
batch_normalization_1 (Batch	(None, 28, 28, 16)	64
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 16)	0
conv2d_2 (Conv2D)	(None, 10, 10, 26)	10426
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 26)	0
flatten_1 (Flatten)	(None, 650)	0
dense_1 (Dense)	(None, 240)	156240
dropout_1 (Dropout)	(None, 240)	0
dense_2 (Dense)	(None, 160)	38560
dropout_2 (Dropout)	(None, 160)	0
dense_3 (Dense)	(None, 10)	1610

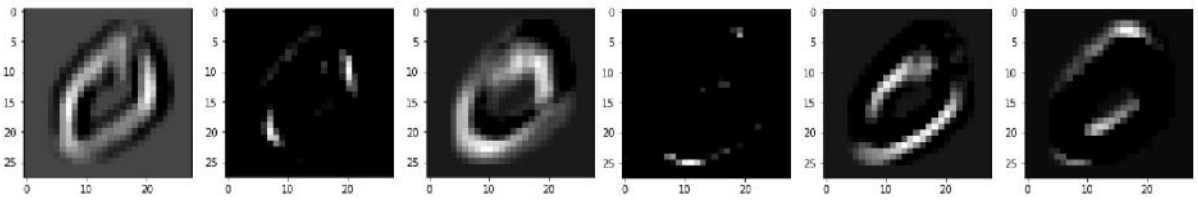
Total params: 207,316
Trainable params: 207,284
Non-trainable params: 32

8. Outputs after each layer-

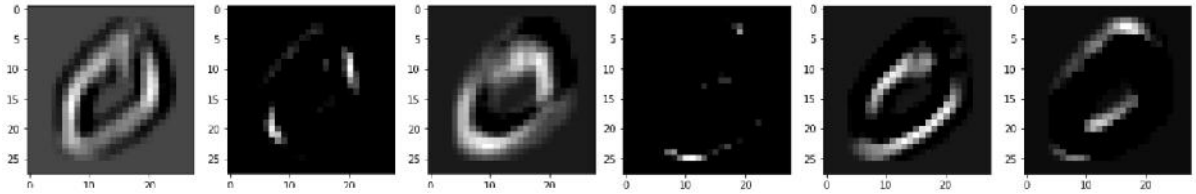
Input--



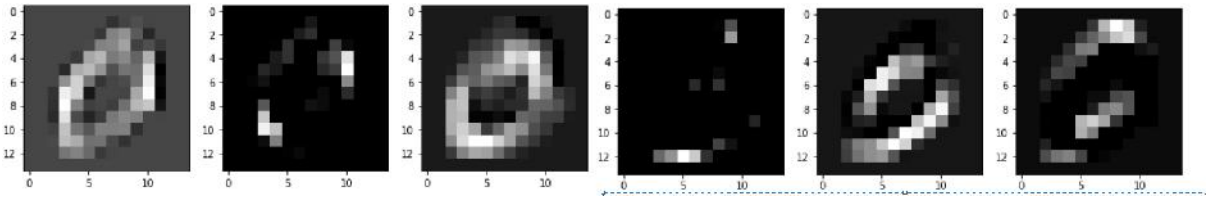
First layer –



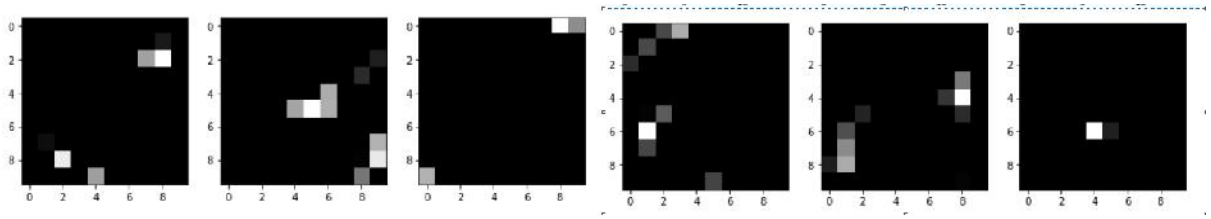
Second layer –



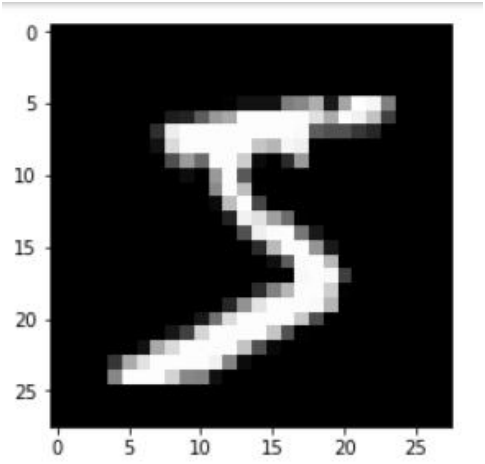
Third layer –



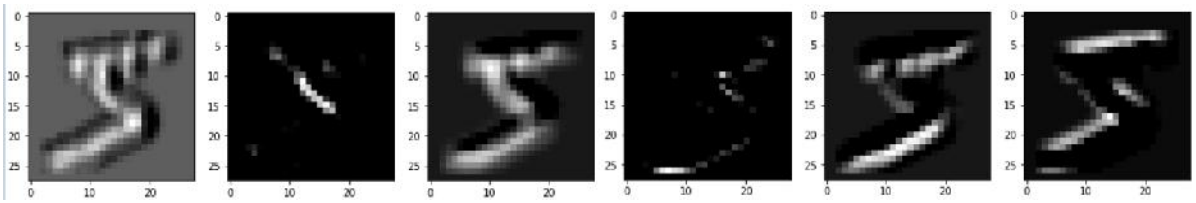
Fourth layer –



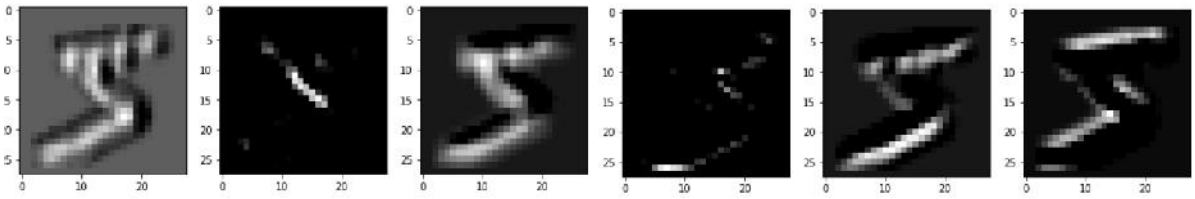
Input –



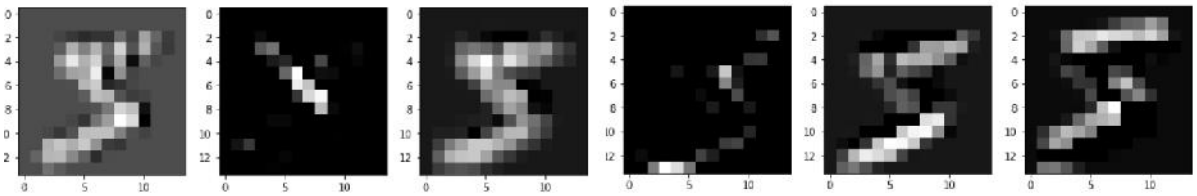
First layer –



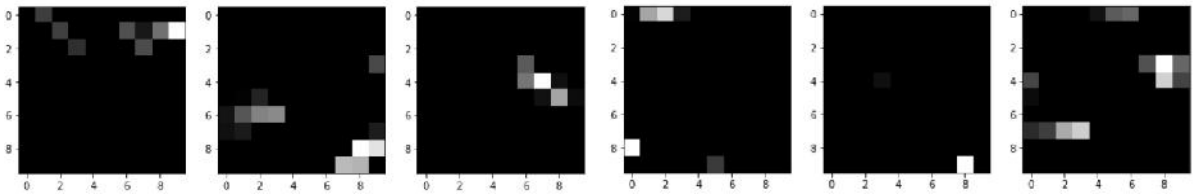
Second layer –



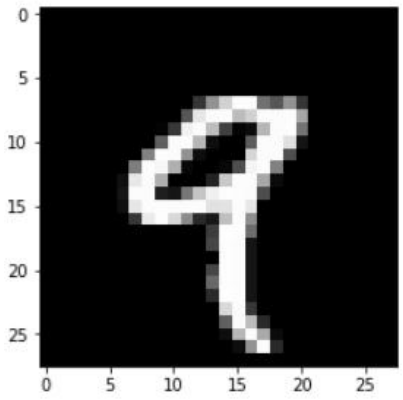
Third layer –



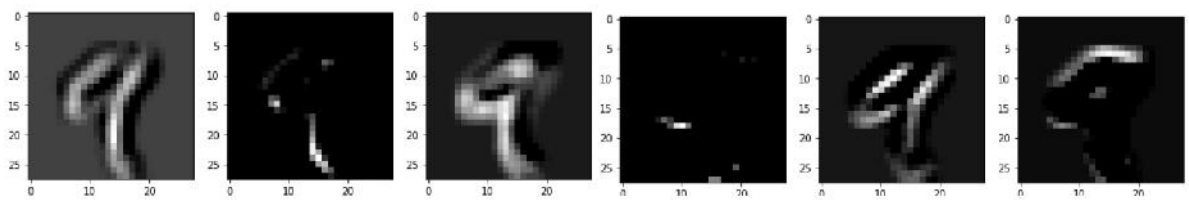
Fourth layer –



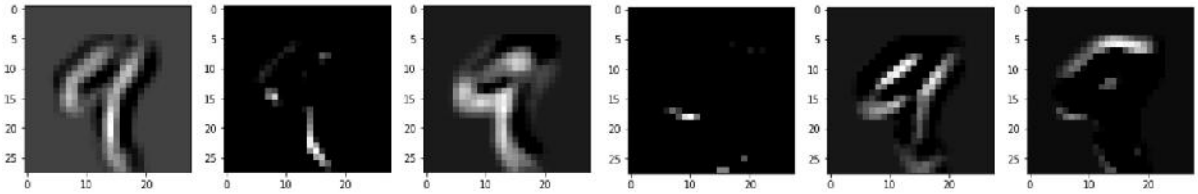
Input –



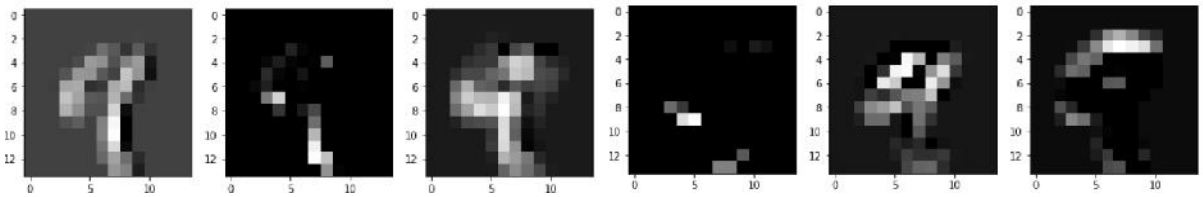
First layer –



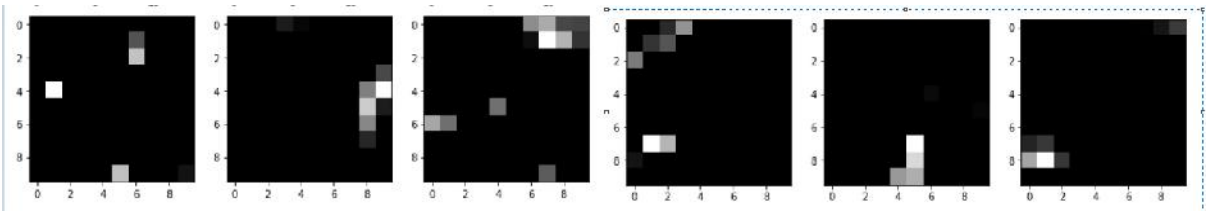
Second layer –



Third layer –



Fourth layer –



Discussion –

Discussion about Batch Normalisation –

- Batch normalisation is used for the reduction in the amount by the which the input ayer is hidden from the output layer.
- It represents the co variance shift
- It allows each and every layer to learn by itself independently and to a greater extent.
- In this I have experimented with the learning rates and found that the leanring rates can be made higher or not because batch normalisation makes sure that none of the activations of any of the layers are going significantly high or significantly low.
- It was also used to reduce the issue of “Over fitting” and we can easily see this in the epoch accuracy curve.
- The training and the testing data have performed well with the highest accuracies of 99.62 and 99.14%
- Batch normalisation normalises the output of the activation layer of the present input and this is done by adding its two parameters that are the mean and the standard deviation

Discussion about Pool Layers-

- The amax-pooling layer is usually added in between the Convolutional layers
- It is usually used to decrease the spatial size and also to reduce to amount of parameters.
- It also reduced the computation of the network.
- It also reduces the amount of Over-fitting.
- The max pooling Layer behaves independently and separately on each and every layer and it is then resized spatially. For this, the max pooling layer s used again.
- The widely used max-pool layer has a filter of size 2*2 and with a stride of 2 down-samples.
- Each operation of the max pooling wil take 4 numbers but the dimension of the depth will be the same.
- Therefore, the max pooling layer will do the work with the dimension of the depth will be the same.
- The other operations that the max pool layer are average pooling and L2 norm pooling.
- Back propogation is mainly the backward pass of a maximum function
- In our training a pooling layer is added after each of the convolutional layers in order to redice the over fitting

Discussion about the Drop out layers –

- I have added drop out layers after each and every dense layer with a parameter of 0,5
- Dropout layers are added further to prevent over fitting.
- It helpsin regularising the neural networks which helps in preventing the over fitting
- This is used to reduce the dependency of each layer on one another.
- The epoch accuracy curve shows how the training and the testing data are performing their max with least over fitting.

<u>Setting – 10 – Learning rate</u>

Adding Layers Stage -

Different Layers	Filter Size	Kernel Size	Pool Size	Padding	Activation	No. of Connections/Neurons

Convolutional2D	16	9	N/A	same	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Convolutional2D	26	9	N/A	-	relu	N/A
Max-Pooling	N/A	N/A	2	N/A	N/A	N/A
Flatten	N/A	N/A	N/A	N/A	N/A	N/A
Dense 1	N/A	N/A	N/A	N/A	N/A	120
Dense 2	N/A	N/A	N/A	N/A	N/A	80
Dense 3	N/A	N/A	N/A	N/A	N/A	10

Compiling Stage –

Parameters	**Types**
Optimizer	rmsprop
Loss	categorical_crossentropy
Metrics	accuracy

Training Stage –

Parameters	**Types/Values/Size**
Batch	32
Epoch	10

Testing Stage –

Parameters	**Types/Values/Size**
Batch	32

2. Accuracy report

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 212s 4ms/step - loss: 0.3971 - acc: 0.8911 - val_loss: 0.1310 - val_acc: 0.9614
Epoch 2/10
60000/60000 [==============================] - 213s 4ms/step - loss: 0.1151 - acc: 0.9639 - val_loss: 0.0802 - val_acc: 0.9753
Epoch 3/10
60000/60000 [==============================] - 216s 4ms/step - loss: 0.0783 - acc: 0.9757 - val_loss: 0.0592 - val_acc: 0.9821
Epoch 4/10
60000/60000 [==============================] - 215s 4ms/step - loss: 0.0613 - acc: 0.9806 - val_loss: 0.0570 - val_acc: 0.9828
Epoch 5/10
60000/60000 [==============================] - 204s 3ms/step - loss: 0.0518 - acc: 0.9841 - val_loss: 0.0491 - val_acc: 0.9825
Epoch 6/10
60000/60000 [==============================] - 209s 3ms/step - loss: 0.0444 - acc: 0.9858 - val_loss: 0.0508 - val_acc: 0.9800
Epoch 7/10
60000/60000 [==============================] - 205s 3ms/step - loss: 0.0394 - acc: 0.9879 - val_loss: 0.0369 - val_acc: 0.9877
Epoch 8/10
60000/60000 [==============================] - 204s 3ms/step - loss: 0.0352 - acc: 0.9892 - val_loss: 0.0374 - val_acc: 0.9884
Epoch 9/10
60000/60000 [==============================] - 201s 3ms/step - loss: 0.0311 - acc: 0.9900 - val_loss: 0.0383 - val_acc: 0.9874
Epoch 10/10
60000/60000 [==============================] - 200s 3ms/step - loss: 0.0280 - acc: 0.9910 - val_loss: 0.0329 - val_acc: 0.9889
```

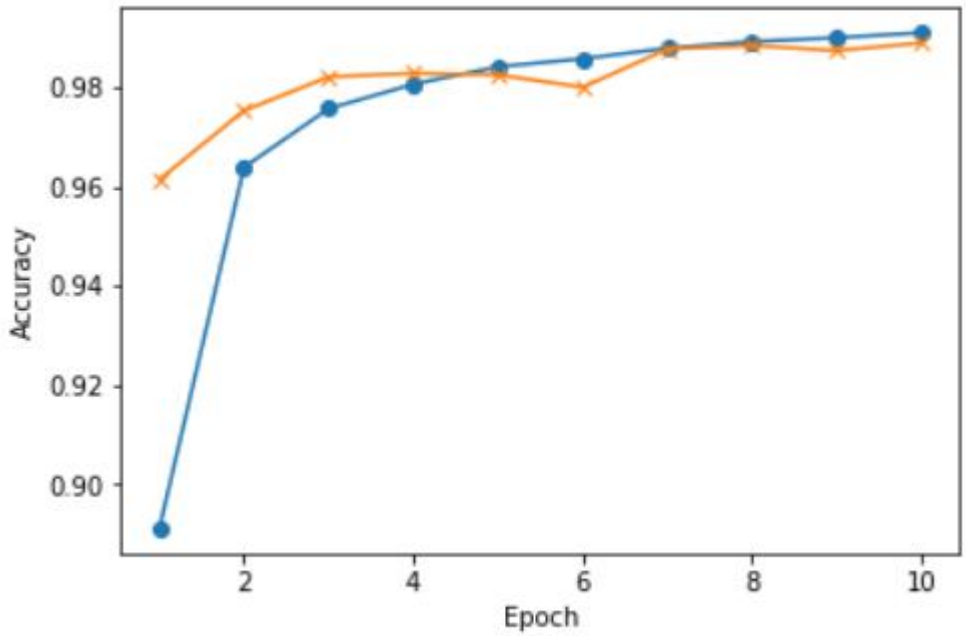
```
10000/10000 [==============================] - 12s 1ms/step
```

```
[0.032864195666089654, 0.9889]
```

Accuracy on training data – 99.10%

Accuracy on testing data – 98.89%

3. Epoch – Accuracy Curve

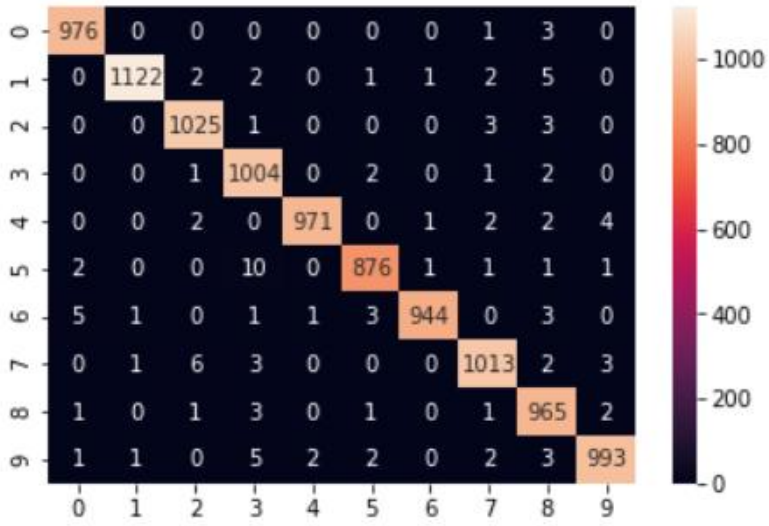


---

4. Confusion Matrix

```
array([[ 976,    0,    0,    0,    0,    0,    0,    1,    3,    0],
       [   0, 1122,    2,    2,    0,    1,    1,    2,    5,    0],
       [   0,    0, 1025,    1,    0,    0,    0,    3,    3,    0],
       [   0,    0,    1, 1004,    0,    2,    0,    1,    2,    0],
       [   0,    0,    2,    0,  971,    0,    1,    2,    2,    4],
       [   2,    0,    0,   10,    0,  876,    1,    1,    1,    1],
       [   5,    1,    0,    1,    1,    3,  944,    0,    3,    0],
       [   0,    1,    6,    3,    0,    0,    0, 1013,    2,    3],
       [   1,    0,    1,    3,    0,    1,    0,    1,  965,    2],
       [   1,    1,    0,    5,    2,    2,    0,    2,    3,  993]],
      dtype=int64)
```

5. Heatmap indicating the confusion matrix-



6. Classification Report

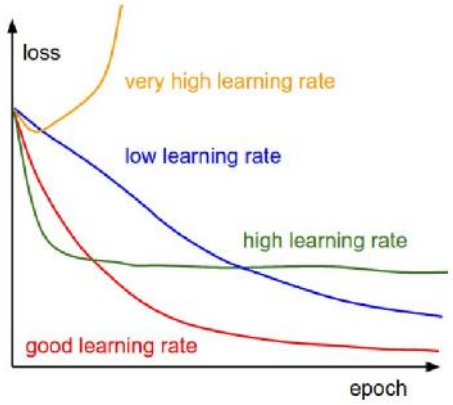
	precision	recall	f1-score	support
class 0	0.99	1.00	0.99	980
class 1	1.00	0.99	0.99	1135
class 2	0.99	0.99	0.99	1032
class 3	0.98	0.99	0.98	1010
class 4	1.00	0.99	0.99	982
class 5	0.99	0.98	0.99	892
class 6	1.00	0.99	0.99	958
class 7	0.99	0.99	0.99	1028
class 8	0.98	0.99	0.98	974
class 9	0.99	0.98	0.99	1009
avg / total	0.99	0.99	0.99	10000

7. Model Summary-

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 16)        1312
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 6, 6, 26)          33722
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 3, 3, 26)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 234)               0
_________________________________________________________________
dense_7 (Dense)              (None, 240)               56400
_________________________________________________________________
dense_8 (Dense)              (None, 160)               38560
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1610
=================================================================
Total params: 131,604
Trainable params: 131,604
Non-trainable params: 0
_________________________________________________________________
```

Discussion about learning rates –

- Learning rate determines the adjustment of our weights with respect to the gradient.
- The lower the value, the leser will be the slope pointing downwards and the local mimima is not reached.



Effect of various learning rates on convergence (Img Credit: cs231n)

- Just like in the diagram, lower the learning rate the gradient wil be less steeper.
- It determines at what point o how fast the global minima is reached.
- The best way for learning rate to affect the system is by letting it increase globally an slowly decreasing it with each update or iteration.

DISCUSSION –

The best model was found to be my 9th setting and the paramters chosen and the results are posted above. The training accuracy was – 99.62 and testing accuracy was 99.13%

The mean of the Training set over all the 10 settings was found to be = 99.14%

The SD was found to be = 2.62

The mean of the Testing set was found to be = 98.996%

The SD was found to be = 2.6

Discussion about variations in filter sizes –

- First I tried to decrease the filter size, but the accuracy decreased significantly.
- That is because the filters are used to extract features from the input image.
- Therefore, I decided I have to give more filter size in order to extract extra features, more than the normal, default settings.
- This turned out to be true as there was a significant increase in the accuracy which is seen in the accuracy report from 98.78 to 99.08%

Discussion about Early Stopping-

- I have used Early Stopping here in order to stop the training to run through all the epochs. This is done to stop "OverFitting"
- Early stopping is a kind of trigger that uses a monitored performance metric to stop the training when the condition is met.
- That is the training is automatically stopped when the performance of the validation or the testing data set decreases significantly compared to the training epoch.
- In this case, I have given a patience level of 2. So the training is stopped if it meets with a consecutive 2 values which are significantly lesser than the training data.
- This helps in the model getting over-fitted and helps in the saving the computational time.

Discussion about the Computational Complexity –

- Running the same model without giving the Early stopping was very hard.

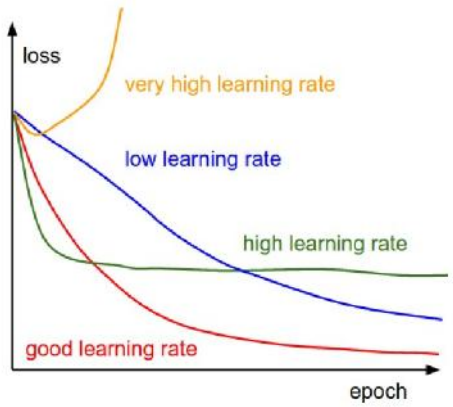
- It took a lot of time for the computation as the filter size was increased.
- But with early stopping, it did not run for all the epochs which helped in stopping the training early.
- Therefore, after early stopping the computational Complexity reduced significantly.

Discussion about Optimizers –

- Using Adams optimizer along with setting a learning rate to 0.0001 is really good and performs well.
- The accuracy has also increased significantly showing that the adam optimiser is better than the rmsprop
- This is true because in the rmsprop optimizer the updation is done for every parameter.
- Adam optimizer combines both the disadvantages of both the rmsprop and the Momentum optimizer.
- Because in Adam optimizer we consider the exponential average and the squre of the gradients of the parameters.
- The epoch- accuracy curve shows this and the way the accuracy has increased over it.

Discussion about learning rates –

- Learning rate determines the adjustment of our weights with respect to the gradient.
- The lower the value, the leser will be the slope pointing downwards and the local mimima is not reached.



Effect of various learning rates on convergence (Img Credit: cs231n)

- Just like in the diagram, lower the learning rate the gradient wil be less steeper.

- It determines at what point o how fast the global minima is reached.
- The best way for learning rate to affect the system is by letting it increase globally an slowly decreasing it with each update or iteration.

Discussion about Batch Normalisation –

- Batch normalisation is used for the reduction in the amount by the which the input ayer is hidden from the output layer.
- It represents the co variance shift
- It allows each and every layer to learn by itself independently and to a greater extent.
- In this I have experimented with the learning rates and found that the leanring rates can be made higher or not because batch normalisation makes sure that none of the activations of any of the layers are going significantly high or significantly low.
- It was also used to reduce the issue of "Over fitting" and we can easily see this in the epoch accuracy curve.
- The training and the testing data have performed well with the highest accuracies of 99.62 and 99.14%
- Batch normalisation normalises the output of the activation layer of the present input and this is done by adding its two parameters that are the mean and the standard deviation

Discussion about Pool Layers-

- The amax-pooling layer is usually added in between the Convolutional layers
- It is usually used to decrease the spatial size and also to reduce to amount of parameters.
- It also reduced the computation of the network.
- It also reduces the amount of Over-fitting.
- The max pooling Layer behaves independently and separately on each and every layer and it is then resized spatially. For this, the max pooling layer s used again.
- The widely used max-pool layer has a filter of size 2*2 and with a stride of 2 down-samples.
- Each operation of the max pooling wil take 4 numbers but the dimension of the depth will be the same.
- Therefore, the max pooling layer will do the work with the dimension of the depth will be the same.

- The other operations that the max pool layer are average pooling and L2 norm pooling.
- Back propogation is mainly the backward pass of a maximum function
- In our training a pooling layer is added after each of the convolutional layers in order to redice the over fitting

Discussion about the Drop out layers –

- I have added drop out layers after each and every dense layer with a parameter of 0,5
- Dropout layers are added further to prevent over fitting.
- It helpsin regularising the neural networks which helps in preventing the over fitting
- This is used to reduce the dependency of each layer on one another.
- The epoch accuracy curve shows how the training and the testing data are performing their max with least over fitting.

Discussion about the over fittting –

- In our training model, giving the epoch size increased the training accuracy to a very huge extent.
- But it went through over fitting and the testing accuracy is much lesser compared to the training accuracy.
- I ran the training for 32 Epochs and this was the output that I got the over fitting of the model
- As you can see that the testing accuracy(orange line) is significantly lesser and keeps getting worse.
- But the training accuracy, with the model almost reaching a 100% is very good.
- This is happening because the model has learnt and gotten really used to the training dataset that the noise and the fluctuations that are present are also taken down and learnt as features and are used for learning.
- Therefore, the testing data performed very badly as shown in the figure.
- The epoch accuracy curve shows the best way to see how the testing data performed against the training data.

Discussion about Batch Size –

- Batch size determines how many inputs were considered before updation.

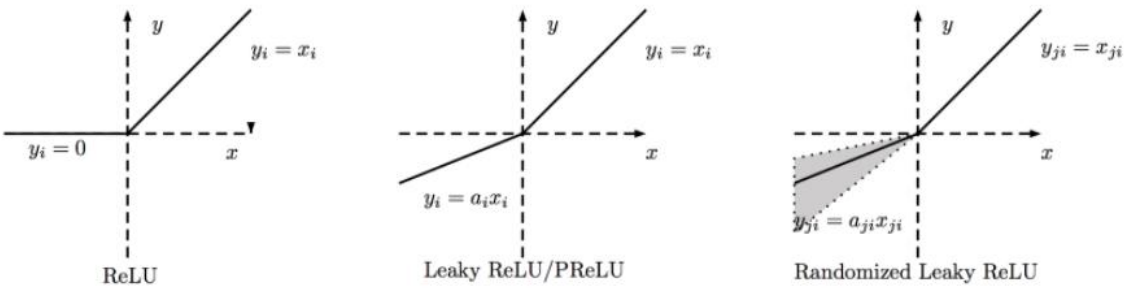
- With increase in the batch size, the model perfomed really well as the number of inputs considered before updation was a lot and that the accuracy significanty increased.
- But on the test data, since they were already so little, were not given much prominence before updation.

Discussion about the variations in the kernel size –

- First I tried to vary the kernel size by increasing it and I found that the accuracy increased but not very much.
- But the details in each of the layer was preserved as you can above in the comparison of the different layers.
- With the increase in the kernel size, the accuracy reached saturation and not much changes were observed.

Discussion about the choice of activation function –

- Actiavation function is basically used to find the output of a particular node given the different sets of inputs.
- The output is then behaved as an input to the next layerwhich is again required to be activated in the same way.
- The main properties of an activation function are –
- 1. They have to be non-linear. But, since the identity property will not necessarily satisy the condition of non-linearity, we need to not use it.
- 2.They have to be Monotonic. That is, they can be used to get a surface for a layer which is error free.
- 3. They should be differentiable – this is mainly used methods based on gradient descent
- Relu function
-



-
- The rectified Linear units function is the best of all three of them.

- It gets rid of the gradients which are vanishing and performs better than any other activation function possible.
- The main disadvantage of the relu function is that it can be used only within ny of the hidden layers.
- Relu could sometimes result in dead neurons.

Problem 1.c)

ABSTRACT AND MOTIVATION

CNN are made up of a lot of neurons that depend on a lot of parameters like weights and biases. The inputs to these are images and can be encoded with properties to make up their architecture. This helps in making up the forward function and it increases its efficiency and it also vastly reduces the number of parameters that are used in a network. Every layer in a Convolutional Neural Network consists of a simple API which transforms the 3Dvolume Input to a 3D volume Output with a lot of differentiable function containing parameters or sometimes, not.

APPROACH –

For negative test images –

- The model that was trained in the part b was used to test on the negative images which was found by taking 255-test images.
- The accuracy and the confusion metrices were plotted

Proposed model 1-

- A new model was created by taking the normal inputs first and then the negative of them are used to train.
- The train sets have 120000 images.
- Since the images are consistent over certain points in the data, the testing results were good but inconsistent.

Proposed model 2-

- Keeping the two drawbacks in mind, the new model was constructed by taking alternates of the normal test image and the negative test image.
- They were alternated between each other and then trained on these samples and the tested for better accuracies.

EXPERIMENTAL RESULTS ALONG WITH DISCUSSION

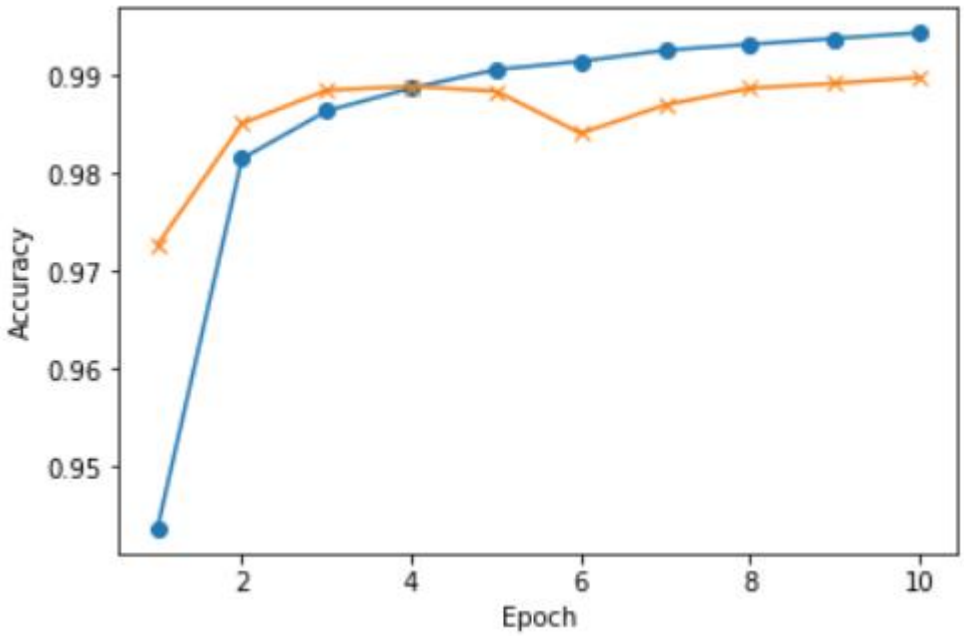
1. Accuracy report

For negative images -

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================]   85s 1ms/step   loss: 0.1788   acc: 0.9436   val_loss: 0.0659   val_acc: 0.9800
Epoch 2/10
60000/60000 [------------------------------] - 82s 1ms/step - loss: 0.0599 - acc: 0.9817 - val_loss: 0.0469 - val_acc: 0.9850
Epoch 3/10
60000/60000 [------------------------------] - 80s 1ms/step - loss: 0.0448 - acc: 0.9866 - val_loss: 0.0438 - val_acc: 0.9872
Epoch 4/10
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0373 - acc: 0.9889 - val_loss: 0.0394 - val_acc: 0.9893
Epoch 5/10
60000/60000 [==============================]   80s 1ms/step   loss: 0.0322   acc: 0.9907   val_loss: 0.0368   val_acc: 0.9894
Epoch 6/10
60000/60000 [------------------------------] - 80s 1ms/step - loss: 0.0283 - acc: 0.9917 - val_loss: 0.0484 - val_acc: 0.9887
Epoch 7/10
60000/60000 [------------------------------] - 81s 1ms/step - loss: 0.0261 - acc: 0.9929 - val_loss: 0.0403 - val_acc: 0.9898
Epoch 8/10
60000/60000 [==============================]   81s 1ms/step   loss: 0.0249   acc: 0.9932   val_loss: 0.0500   val_acc: 0.9892
Epoch 9/10
60000/60000 [------------------------------] - 80s 1ms/step - loss: 0.0223 - acc: 0.9943 - val_loss: 0.0613 - val_acc: 0.9873
Epoch 10/10
60000/60000 [------------------------------] - 81s 1ms/step - loss: 0.0234 - acc: 0.9939 - val_loss: 0.0588 - val_acc: 0.9875
```

```
10000/10000 [==============================] - 6s 632us/step

[8.287583767700195, 0.2994]
```

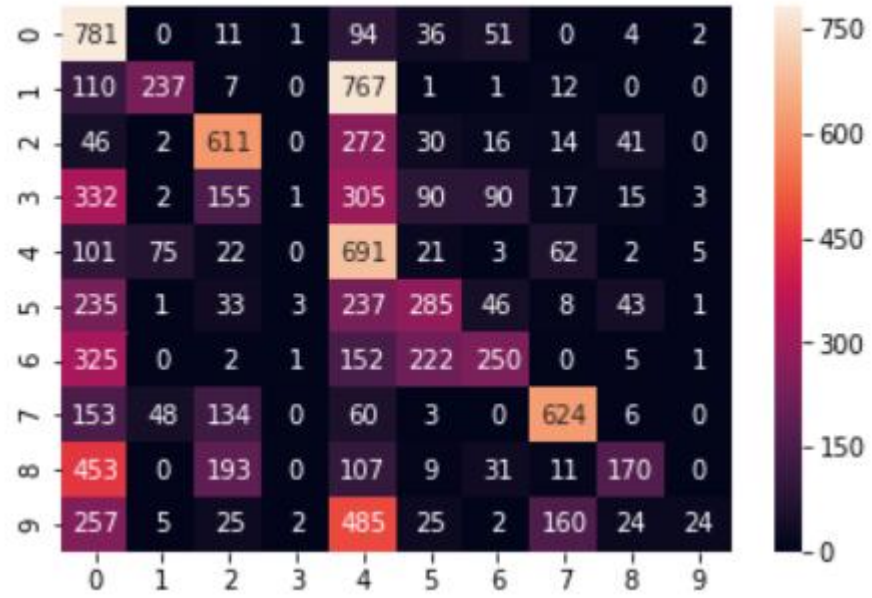
ANSWER 1.C.1) The accuracy of the negative test images was found to be equal to 29.94%



Epoch -accuracy curve

```
array([[781,   0,  11,   1,  94,  36,  51,   0,   4,   2],
       [110, 237,   7,   0, 767,   1,   1,  12,   0,   0],
       [ 46,   2, 611,   0, 272,  30,  16,  14,  41,   0],
       [332,   2, 155,   1, 305,  90,  90,  17,  15,   3],
       [101,  75,  22,   0, 691,  21,   3,  62,   2,   5],
       [235,   1,  33,   3, 237, 285,  46,   8,  43,   1],
       [325,   0,   2,   1, 152, 222, 250,   0,   5,   1],
       [153,  48, 134,   0,  60,   3,   0, 624,   6,   0],
       [453,   0, 193,   0, 107,   9,  31,  11, 170,   0],
       [257,   5,  25,   2, 485,  25,   2, 160,  24,  24]], dtype=int64)
```

Confusion matrix



Heat map

	precision	recall	f1-score	support
class 0	0.98	1.00	0.99	980
class 1	0.99	1.00	1.00	1135
class 2	0.99	1.00	0.99	1032
class 3	1.00	0.99	0.99	1010
class 4	0.99	1.00	0.99	982
class 5	0.99	0.99	0.99	892
class 6	1.00	0.98	0.99	958
class 7	0.99	0.98	0.98	1028
class 8	0.98	0.99	0.98	974
class 9	0.99	0.98	0.99	1009
avg / total	0.99	0.99	0.99	10000

Classification report

Layer (type)	Output Shape	Param #
conv2d_47 (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d_47 (MaxPooling	(None, 14, 14, 6)	0
conv2d_48 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_48 (MaxPooling	(None, 5, 5, 16)	0
flatten_24 (Flatten)	(None, 400)	0
dense_70 (Dense)	(None, 120)	48120
dense_71 (Dense)	(None, 80)	9680
dense_72 (Dense)	(None, 10)	810

Total params: 61,182
Trainable params: 61,182
Non-trainable params: 0

Model summary

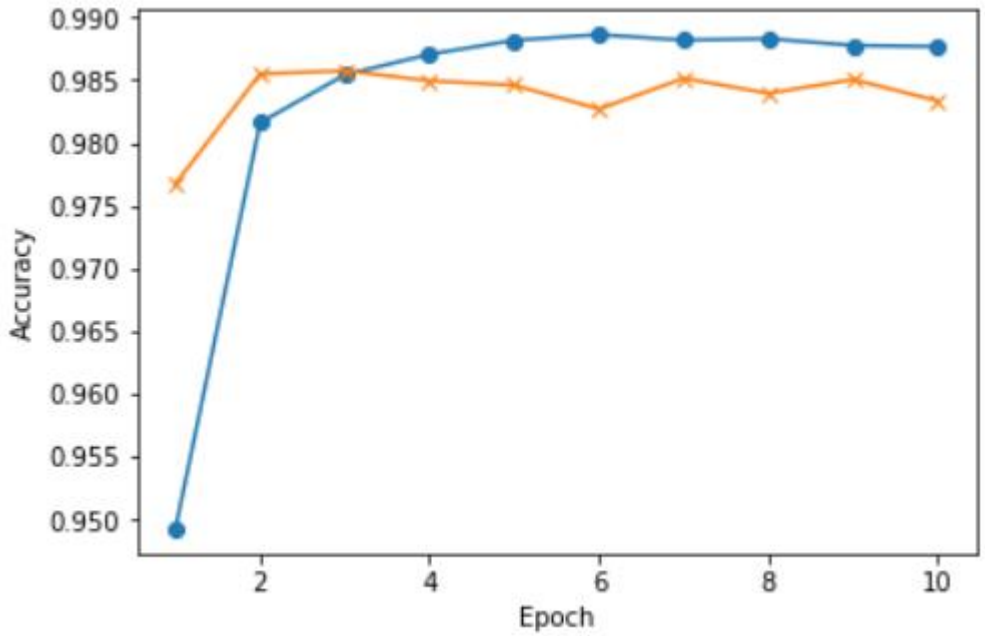
Proposed Network 1 –

Accuracy -

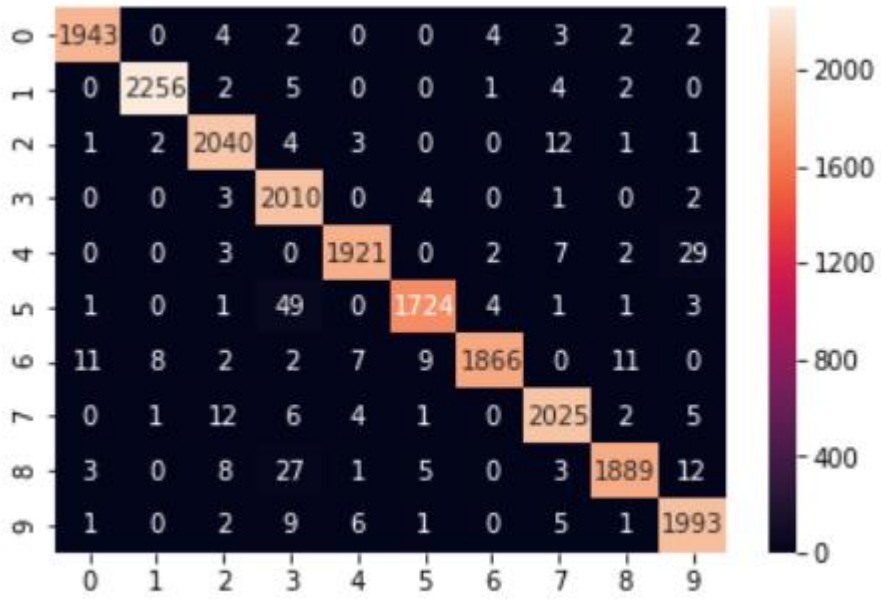
```
Train on 120000 samples, validate on 20000 samples
Epoch 1/10
120000/120000 [==============================] - 94s 787us/step - loss: 0.1743 - acc: 0.9446 - val_loss: 0.0768 - val_acc: 0.97
61
Epoch 2/10
120000/120000 |==============================|  96s 803us/step  loss: 0.0640  acc: 0.9811  val loss: 0.0444  val acc: 0.98
56
Epoch 3/10
120000/120000 [------------------------------] - 89s 742us/step - loss: 0.0514 - acc: 0.9849 - val_loss: 0.0491 - val_acc: 0.98
38
Epoch 4/10
120000/120000 [------------------------------] - 94s 779us/step - loss: 0.0452 - acc: 0.9871 - val_loss: 0.0451 - val_acc: 0.98
60
Epoch 5/10
120000/120000 [==============================] - 94s 781us/step - loss: 0.0412 - acc: 0.9885 - val_loss: 0.0442 - val_acc: 0.98
73
Epoch 6/10
120000/120000 |==============================|  101s 839us/step  loss: 0.0404  acc: 0.9887  val loss: 0.0752  val acc: 0.9
835
Epoch 7/10
120000/120000 [------------------------------] - 89s 744us/step - loss: 0.0410 - acc: 0.9893 - val_loss: 0.0494 - val_acc: 0.98
66
Epoch 8/10
120000/120000 [------------------------------] - 91s 759us/step - loss: 0.0408 - acc: 0.9893 - val_loss: 0.0579 - val_acc: 0.98
74
Epoch 9/10
120000/120000 [==============================] - 100s 829us/step - loss: 0.0406 - acc: 0.9899 - val_loss: 0.0635 - val_acc: 0.9
820
Epoch 10/10
120000/120000 |==============================|  102s 854us/step  loss: 0.0421  acc: 0.9896  val loss: 0.0602  val acc: 0.9
861
```

Accuracy of the training model = 98.96%

Accuracy of the testing model = 98%




```
array([[1943,    0,    4,    2,    0,    0,    4,    3,    2,    2],
       [   0, 2256,    2,    5,    0,    0,    1,    4,    2,    0],
       [   1,    2, 2040,    4,    3,    0,    0,   12,    1,    1],
       [   0,    0,    3, 2010,    0,    4,    0,    1,    0,    2],
       [   0,    0,    3,    0, 1921,    0,    2,    7,    2,   29],
       [   1,    0,    1,   49,    0, 1724,    4,    1,    1,    3],
       [  11,    8,    2,    2,    7,    9, 1866,    0,   11,    0],
       [   0,    1,   12,    6,    4,    1,    0, 2025,    2,    5],
       [   3,    0,    8,   27,    1,    5,    0,    3, 1889,   12],
       [   1,    0,    2,    9,    6,    1,    0,    5,    1, 1993]],
      dtype=int64)
```



	precision	recall	f1-score	support
class 0	0.99	0.99	0.99	1960
class 1	1.00	0.99	0.99	2270
class 2	0.98	0.99	0.99	2064
class 3	0.95	1.00	0.97	2020
class 4	0.99	0.98	0.98	1964
class 5	0.99	0.97	0.98	1784
class 6	0.99	0.97	0.98	1916
class 7	0.98	0.98	0.98	2056
class 8	0.99	0.97	0.98	1948
class 9	0.97	0.99	0.98	2018
avg / total	0.98	0.98	0.98	20000

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_45 (Conv2D)           (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
flatten_23 (Flatten)         (None, 400)               0
_________________________________________________________________
dense_67 (Dense)             (None, 120)               48120
_________________________________________________________________
dense_68 (Dense)             (None, 80)                9680
_________________________________________________________________
dense_69 (Dense)             (None, 10)                810
=================================================================
Total params: 61,182
Trainable params: 61,182
Non-trainable params: 0
_________________________________________________________________
```

Proposed Network 2–

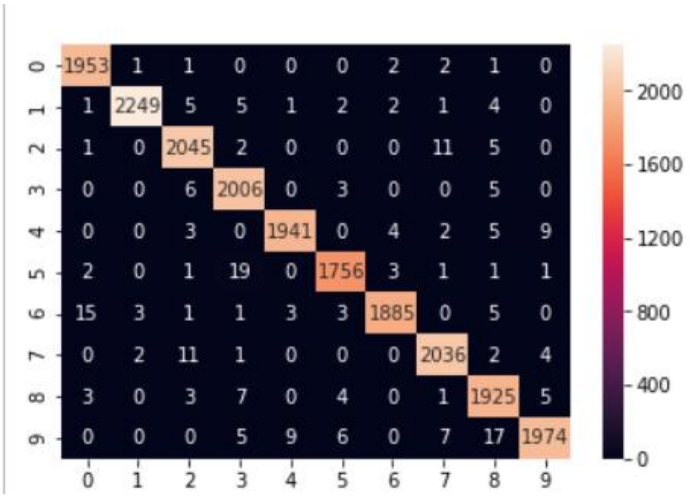
```
Train on 120000 samples, validate on 20000 samples
Epoch 1/10
120000/120000 [------------------------------] - 214s 2ms/step - loss: 0.1477 - acc: 0.9535 - val_loss: 0.0474 - val_acc: 0.984
4
Epoch 2/10
120000/120000 |==============================|   160s 1ms/step   loss: 0.0526   acc: 0.9836   val_loss: 0.0389   val_acc: 0.986
9
Epoch 3/10
120000/120000 [------------------------------] - 163s 1ms/step - loss: 0.0389 - acc: 0.9877 - val_loss: 0.0346 - val_acc: 0.988
4
Epoch 4/10
120000/120000 |==============================|   163s 1ms/step   loss: 0.0304   acc: 0.9902   val_loss: 0.0340   val_acc: 0.989
1
Epoch 5/10
120000/120000 [------------------------------] - 167s 1ms/step - loss: 0.0259 - acc: 0.9919 - val_loss: 0.0396 - val_acc: 0.987
9
Epoch 6/10
120000/120000 [==============================]   158s 1ms/step   loss: 0.0226   acc: 0.9926   val_loss: 0.0427   val_acc: 0.988
0
Epoch 7/10
120000/120000 [------------------------------] - 157s 1ms/step - loss: 0.0183 - acc: 0.9940 - val_loss: 0.0388 - val_acc: 0.989
1
Epoch 8/10
120000/120000 [==============================] - 156s 1ms/step - loss: 0.0169 - acc: 0.9943 - val_loss: 0.0414 - val_acc: 0.989
3
Epoch 9/10
120000/120000 [------------------------------] - 160s 1ms/step - loss: 0.0147 - acc: 0.9953 - val_loss: 0.0460 - val_acc: 0.987
9
Epoch 10/10
120000/120000 [==============================] - 160s 1ms/step - loss: 0.0141 - acc: 0.9954 - val_loss: 0.0501 - val_acc: 0.988
5
```

Accuracy of the training set = 99.54%

Accuracy of the testing set = 98.9%

```
array([[1953,    1,    1,    0,    0,    0,    2,    2,    1,    0],
       [   1, 2249,    5,    5,    1,    2,    2,    1,    4,    0],
       [   1,    0, 2045,    2,    0,    0,    0,   11,    5,    0],
       [   0,    0,    6, 2006,    0,    3,    0,    0,    5,    0],
       [   0,    0,    3,    0, 1941,    0,    4,    2,    5,    9],
       [   2,    0,    1,   19,    0, 1756,    3,    1,    1,    1],
       [  15,    3,    1,    1,    3,    3, 1885,    0,    5,    0],
       [   0,    2,   11,    1,    0,    0,    0, 2036,    2,    4],
       [   3,    0,    3,    7,    0,    4,    0,    1, 1925,    5],
       [   0,    0,    0,    5,    9,    6,    0,    7,   17, 1974]],
      dtype=int64)
```

Confusion matrix



Heat map

	precision	recall	f1-score	support
class 0	0.99	1.00	0.99	1960
class 1	1.00	0.99	0.99	2270
class 2	0.99	0.99	0.99	2064
class 3	0.98	0.99	0.99	2020
class 4	0.99	0.99	0.99	1964
class 5	0.99	0.98	0.99	1784
class 6	0.99	0.98	0.99	1916
class 7	0.99	0.99	0.99	2056
class 8	0.98	0.99	0.98	1948
class 9	0.99	0.98	0.98	2018
avg / total	0.99	0.99	0.99	20000

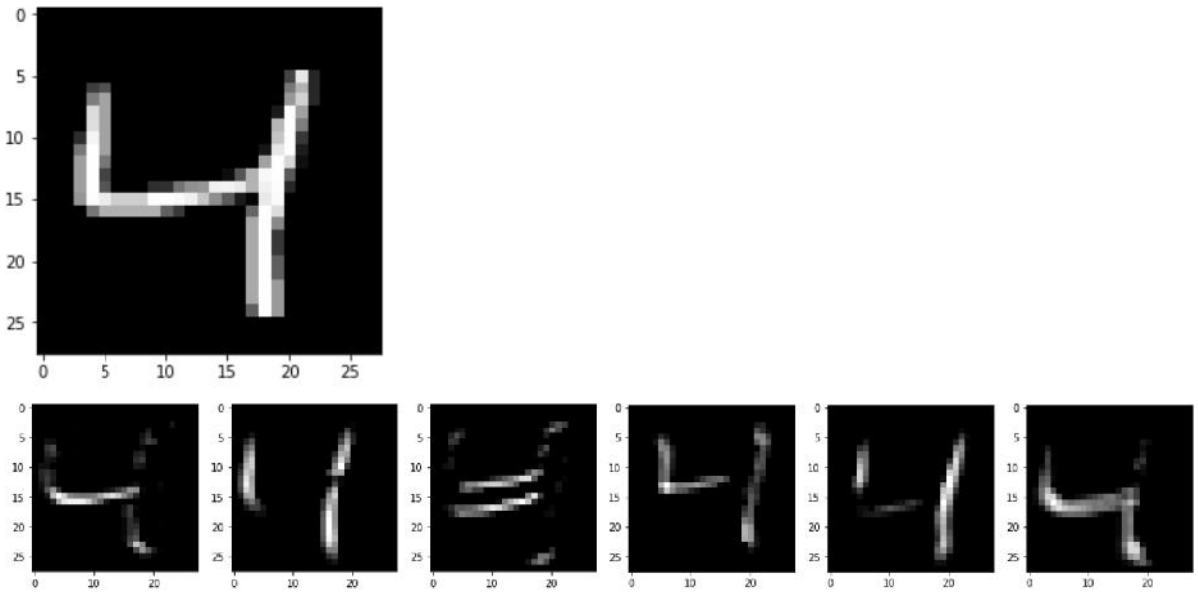
Classification report

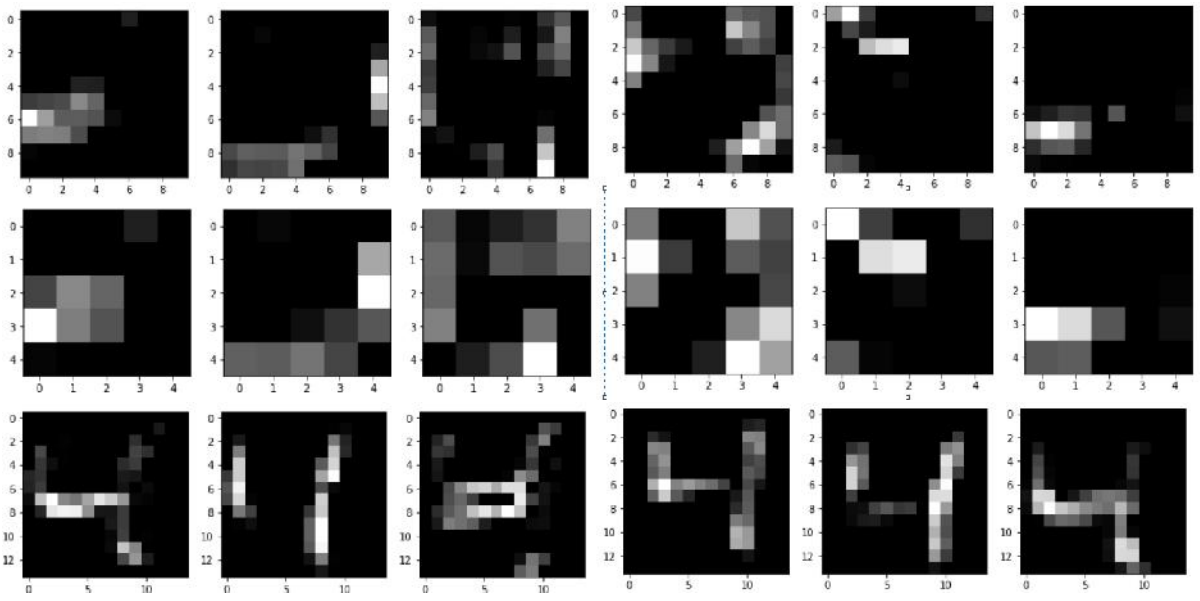
Layer (type)	Output Shape	Param #
conv2d_43 (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d_43 (MaxPooling	(None, 14, 14, 6)	0
conv2d_44 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_44 (MaxPooling	(None, 5, 5, 16)	0
flatten_22 (Flatten)	(None, 400)	0
dense_64 (Dense)	(None, 120)	48120
dense_65 (Dense)	(None, 80)	9680
dense_66 (Dense)	(None, 10)	810

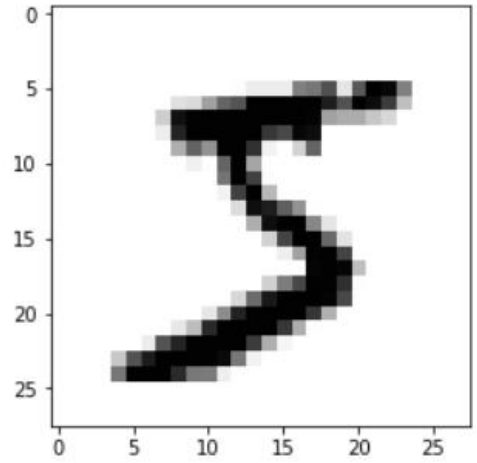
Total params: 61,182
Trainable params: 61,182
Non trainable params: 0

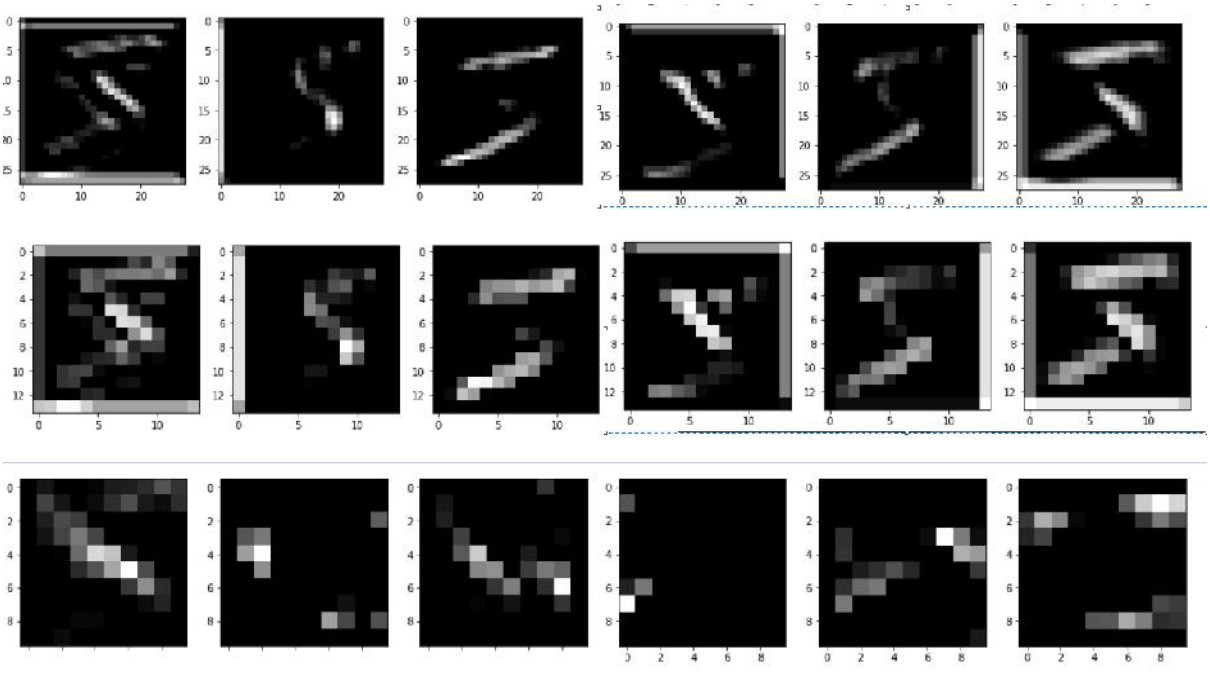
Model Summary

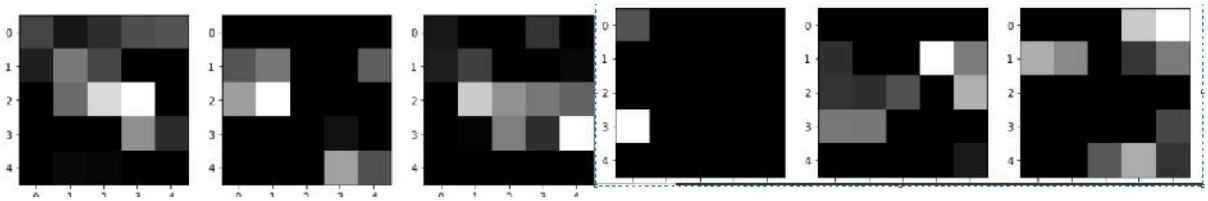
Comparison between each layers for one black ackground and one negative image is shown below.











DISCUSSION-

ANSWER 1.c.1)

- The Lenet 5 dataset doesn't understand handwritten digits as well as human beings.
- This can be proved by the feeding in the negative test images to the model that was trained in the part b.
- The accuracy when the negative test images were fed to the training model was as bad as 29%.
- This bad accuracy is because the Lenet has been trained only on the black background white object images.
- This is a major drawback of the CNNs.
- It cannot replicate the human eye because it fails in classifying the negative images as they are.
- The effective capacity of a CNN is limited to learning only those features or inputs on which it was trained on.
- When the negative test images are fed, the model fails to recognise and take the features that were responsible for the training of those images.
- This is can be gotten rid of by –
- 1. Either training the data on al kinds of inputs fed.
- 2. Using Semantical Generalisation which really gets rid of the differences in the inputs fed to the Convolutional Neural Network.
- The Confusion matrix is used to show how the negative test iages are wrongly classified and the accuracy for the testing negative images is very less.
- This is taken into consideration because of the badly performance of the negative test images of the model.
- Since, this is very bad for the performance, this is the main drawback of the CNN that it cannot predict on some data which it has not been trained on.

- I even tried to give more filter size and other paramters like in part b, but there was no significant change in the test accuracy.
- Therefore, it can be concluded that Lenet 5 is just another model and cannot replicate the human eye with its capabilities.

ANSWER 1c.2)

Proposed model 1 –

- Since the Lenet 5 performed very badly on the negative test images and didn't meet accuracies as expected, I tried to work on the main drawback.
- The main drawback was that it was not trained on such negative images which made it impossible for it to test the images which were negative.
- Therefore, I tried to train the model on the negative images.
- I first trained the model on the normal test data and later I trained it using the negative of these images.
- This was a good strategy because the model needs to be aware of the negative test images.
- By doing this, the accuracy went up significantly higher.
- It was around 98.96 for training data and 98% for the test data.
- So, this proves that the Lenet5 needs to learn about both the types of inputs that are present in the test data.
- Advantages –
- 1. The training and the testing data accuracy went up significantly higher.
- 2. The model was easily able to recognise the negative test images that were given in the test data
- 3. The model behaviour was almost similar to the human eye perception.
- 4. The classficaton report significantly improved as the model no longer new to the new set of test data.
- Disadvantages –
- 1. Although, the maximum accuracy of the test data was found to be 98%, the model had a lot of epochs with 90% accuracy which happened evenly spread throughout the model.
- 2. The accuracy was really good, but unstable.
- Therefore, in order to improve this inconsistency, I proposed another model which is discussed as below.

Proposed model 2 –

- Keeping the two drawbacks from the previous two models, I designed this model in order to get rid of both of the disadvantages.

- The first disadvantage was that a model was not able to recognise the negative test images that were given in order to test
- The second disadvantage is that the model becomes inconsistent with the test data if it was trained on the same consistent data given in the training data.
- Therefore, I designed a model which takes one input from the normal black background and white object and the other input is the negative image that was there.
- The whole training was done on this and then tested.
- The training accuracy went significantly high and the testing accuracy also went significantly high.
- The training accuracy was found to be 99.54% and the testing accuracy was found to be equal to 98.98% with no inconsistencies.
- The inconsistencies that were found in the previous model between the epochs were not found in this.
- Also, I tried out a few parameter settings as part b and found that the best result was obtained for the Setting 9.
- That yielded in better results with no inconsistencies and no over fitting of the model.
- I have also shown the different layers outputs within, and this shows how an image with a white background and black object is also recognised and the outputs between the layers are also very similar to the outputs obtained during the training with a black background and white image.

REFERENCES –

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

https://engmrk.com/lenet-5-a-classic-cnn-architecture/

https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

All links given in homework files, Discussions and reading material.

https://arxiv.org/pdf/1703.06857.pdf

https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/dropout_layer.html

https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

https://cs231n.github.io/convolutional-networks/#fc

https://en.wikipedia.org/wiki/Convolutional_neural_network#Fully_connected_layer

https://www.edureka.co/blog/backpropagation/

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

https://en.wikipedia.org/wiki/Loss_function

https://blog.algorithmia.com/introduction-to-loss-functions/

https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4

https://medium.com/intuitionmachine/why-deep-learning-is-radically-different-from-machine-learning-945a4a65da4d

https://arxiv.org/pdf/1811.00116.pdf

https://www.slideshare.net/ChaoHanchaohanvtedu/deep-cnn-vs-conventional-ml-81059441

https://en.wikipedia.org/wiki/Overfitting#Remedy

https://stats.stackexchange.com/questions/305452/understanding-early-stopping-in-neural-networks-and-its-implications-when-using

https://deeplearning4j.org/docs/latest/deeplearning4j-nn-early-stopping

https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

https://towardsdatascience.com/overfitting-vs-underfitting-a-conceptual-explanation-d94ee20ca7f9

https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f