EE 569 HOMEWORK V

Royston Marian Mascarenhas

Royston Marian Mascarenhas, Spring 19, USC

Convolutional Neural Networks

(a)

CNN Architecture and Training

1. **Abstract and Motivation:**

   Image classification is a daunting task especially when the number of target classes increase. Mathematical methods proved too computational and data driven methods proved no different. But the ease in computation in the early 2000s and the development of algorithms such as CNN have made image classification an automated task. Currently, up to 1000 output classes can be predicted. Therefore, CNNs form the very heart of computer vision. Here, we will explore CNN classification on the MNIST dataset with strong analysis.

2. **Explanations:**

Terminology:

1. Fully connected layer:

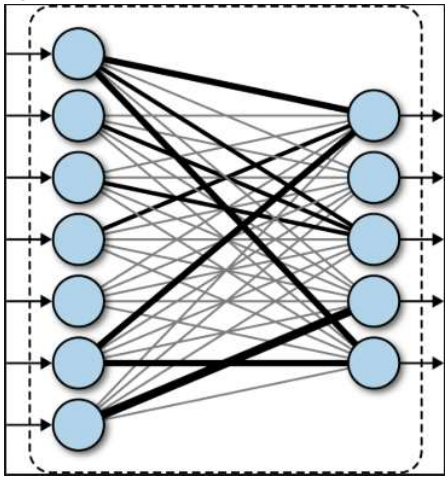
The fully connected layer has connections which are fully connected to the all the inputs of the previous layer. For this reason, the previous layer needs to be **flattened** since the previous CONV layers does not have compatible one dimensional nodes. Therefore, the previous CONV layer dimensions have to be unravelled in order to facilitate the functionality of a dense layer.

The output from the CONV layer represents high – level features in the data. Flattening the layer ensures that **the non linear combinations are learned the easy way**.

The convolutional layer is used for feature extraction. The dense layers are used to **classify** the input images based on the extracted features. Instead of using a SVM to classify the input based on the features, we use dense layers to make it end to end trainable.

The **number of trainable parameters** in a dense layer is input x output + bias. The bias is the number of nodes in that densely connected layer. The input is the connections from the previous layer and the output is the number of connections in that layer. A layer with 120 connections from the previous layer and 84 connections in the present layer will have 10164 trainable parameters (120 x 84 + 84). Thus, their activations can be learnt by matrix multiplication followed by an offset.

FC to Conv Layer: The FC layers can be converted to CONV layers by reshaping them into even equally sized arrays. Each of these conversions involve reshaping the weight matrix in each FC layer to CONV layer. This allows easier transitions of the convnet over the spatial positions in one forward pass.



FC layer	CONV layer
No parameter sharing	Parameter sharing among each slice of output volume depth
All nodes get all connections from the previous layer nodes	Each neuron is connected to a local region of the input

Summary of the purpose of the FC layer:

1. To predict output class labels.
2. To provide end to end connections with each node of the last stage to each node of the next stage.
3. Non linear combinations are learned the easier way.
4. Tuning weights during back propagation.
5.

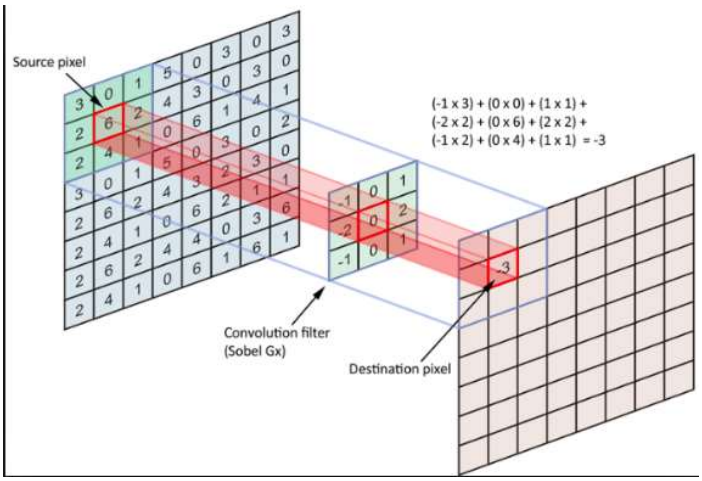
2. Convolutional Layer:

The convolutional layer is a layer consisting of various interpretations of the previous layer. It is designed in such a way so as to allow for the input to be an image instead of just one dimensional nodes stacked together.

It takes after the name 'convolution' because it involves the operation of sliding a nx n window over the image and multiplying each input with the corresponding filter weight. When all the n x n entries are obtained, they are added together to create the entry for the center pixel of that window.

Filter*: Every filter is a tool for interpreting the image in its own unique way.* A n x n filter consists of n x n **(kernel size)** learnable weights plus a bias of one unit. Every filter extends through the dimensions of the input volume. If the input was RGB, the filter would extend through 3 dimensions of the input. Each filter produces a 2 dimensional activation map which corresponds to a specific visual feature. These activation maps are stacked together to produce the output volume. Every entry in the output volume can be thought of a neuron gaining insight into a particular aspect of the input. To

makes sure it scales to big input sizes, we will connect each neuron only to a specific region of the input. This is known as the receptive field of the neuron. For LeNet, each neuron will have weights 5x5x1 + 1 = 26 in number.



The convolutional operation in a CONV layer

Spatial properties: Three hyperparameters determine the output size of the CONV layer. The **depth** of the output volume is the number of the filters employed in that CONV layer. The different neurons along the depth of the volume might activate based on different triggers such as edges, blobs, textures etc. The **stride** is used to slide the window of the filter by n pixels. By default, the stride is taken to be 1 so that the output volume is not reduced. The **extent of zero padding** is used to manage the spatial imbalance that occurs while choosing different parameters.

The correctness of the properties can be chosen by this formula:

$$Fit = (W - F + 2P) / (S + 1)$$

where W is the input volume size, F is the receptive field size of the conv layer neurons, S is the stride and P is the amount of zero padding. If fit is an integer, the parameters chosen are correct.

The CONV layer tends itself to **parameter sharing**. Each of the 28 x 28 inputs in the image use the slice of the depth (6 filters). This results in 6 x 5 x 5 + 6 = 156 filters instead of 28 x 28 = 784 connections.

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary of the CONV layer dimensions and parameters *ref* [1]

Suppose we want to stack one CONV with kernel size of 7 x7 instead of 3 conv layers with kernel size of 3 x 3. The first method is not advisable as three layers would involve more non linearities making the features more expressive and it would result in loss of information as well. Although in the latter case, we might need more memory to hold all the weight during backpropagation.

Summary of the purpose of the FC layer:

1. Feature extraction.
2. Parameter sharing.
3. Spatial independence due to parameter sharing.
4. Tuning weights during backpropagation.

3. Max Pooling Layer:

A max pooling layer considers a square window of the input pixels and takes the maximum pixel magnitude of the window to represent that window in the output frame.

Purpose:

1. A pooling layer's main purpose is to downsample the input to make computations easier and runtime faster. By extension, it also helps reduce overfitting since it allows only responses with the highest magnitude of energies.

2. The pooling layer works independently on each slice of the depth of the previous CONV layer to spatially downsize it. There are many **types** of pooling such as average pooling, median pooling but the most used is MAX pooling. With a window size of 2 x 2 and a stride of 2, it reduces the input activations by 75%.

A pooling layer usually has two settings: a 3x3 window with a stride of 2 or a 2x2 window with a stride of 2. Higher window sizes have a destructive effect and result in loss of information.

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

Summary of the max pool layer dimensions and parameters *ref* [1]

4. Activation Function:

The activation function taken a single input and produces an output after mathematical operation on it.

Their purpose is to

1. limit the input to a particular range
2. introduce non linearity
3. firing only the required neurons

Understanding the main purpose:

The main purpose of an activation function is to decide whether the neuron has fired or not. If the neuron has data which is not significant in deciding a particular outcome, then the activation function does not fire. That value is thus discarded.

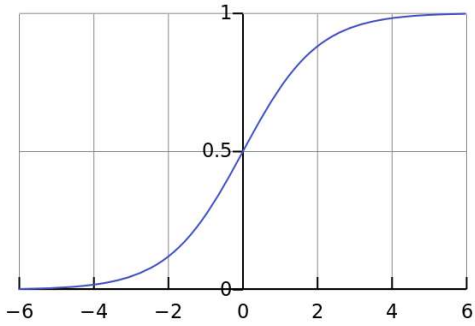
$$Y = \sum (weight * input) + bias$$

Consider above, the output of a neuron. Whether this fires or not is decided by the activation function.

There are various types of activation functions. Some of the most common are:

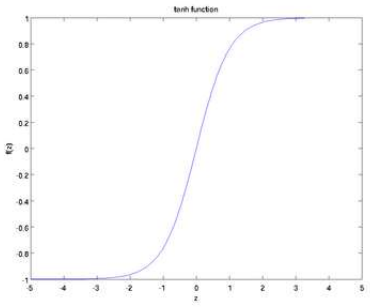
1. Sigmoid
   It limits the input ot the range (0,1). Largenegative numbers tend to 0 and large positive numbers tend to 1. However, sigmoids saturate the gradient and sigmoid outputs are not zero centered. Thus, they are no longer is use.

$$A = \frac{1}{1+e^{-x}}$$



2. Tanh

   A tanh is simply a sigmoid activation with zero center. The gradients saturate but the latter feedback is taken care of. Therefore, it is preferred over sigmoid.



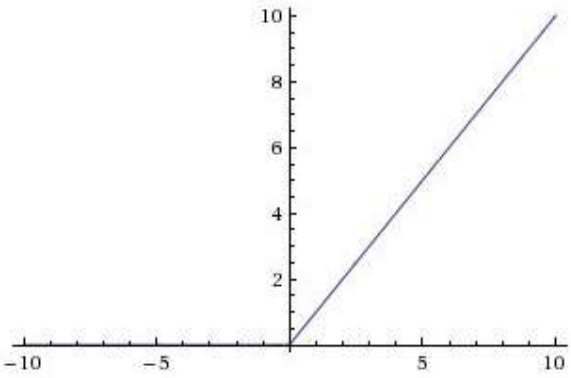
$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

3. Relu

   Relu activation function allows only positive values to pass. Negative values are clipped off.

   A = max(0,x) is the activation function of relu

   The reason why relu is used in a CNN is because with multitudes of neurons, sparse matrices make computations much easier. However, relu might make a network passive since it deletes all negative components which might result in loss of information. To solve this problem, we allow a small portion of the negative range to leak through. This is called leaky relu activation.



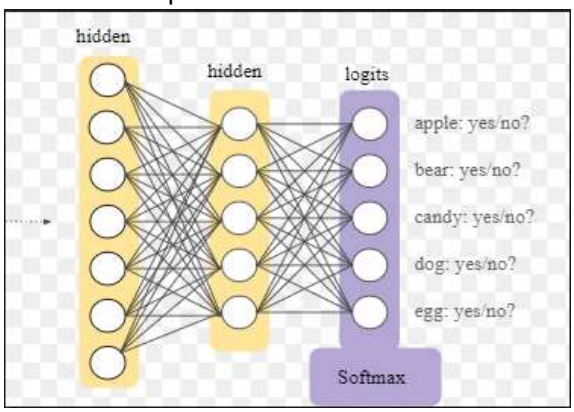
4. Softmax

The output probabilities from the last dense layer cannot be interpreted as the probability of the input being one of the n classes. Essentially, the softmax function is taken to be from the bernouilli distribution but instead of two classes, it is extended to ten classes.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Purpose:

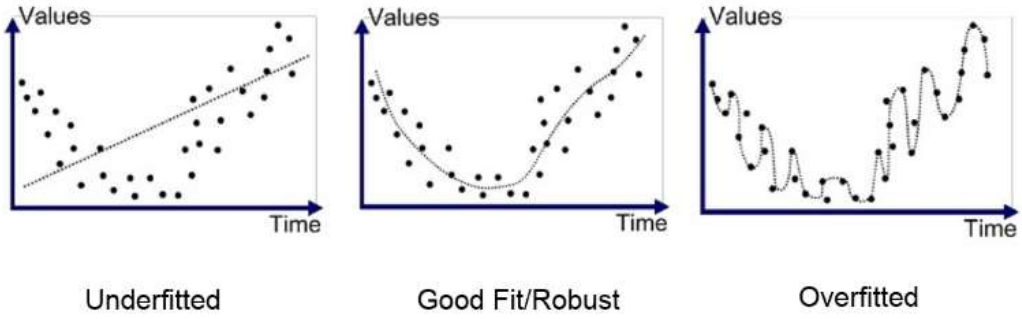
1. The softmax layer takes the input vector and converts it to a range such that the sum of all the values in the converted input vector sum to 1. Therefore, this can be interpreted as the probability that the input image belongs to a particular class.
2. The softmax is mainly used as the last layer activation for classification problems.

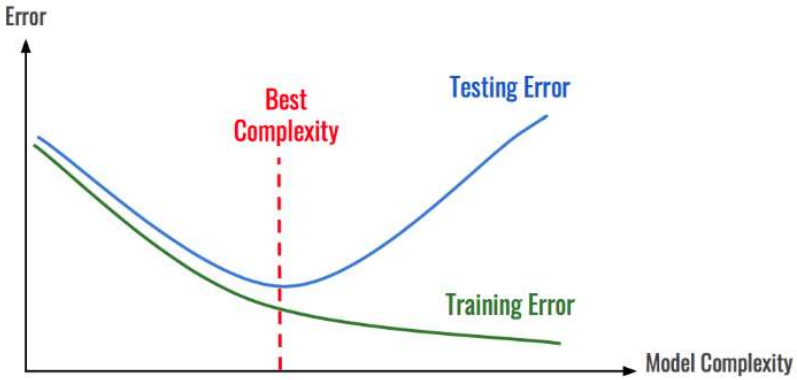


Overfitting:

A model overfits when it does not generalize broadly enough. When a model is overfit, it has trained excessively on the training set and obtains a very high accuracy on it but

performs poorly on the test set. Mathematically, it means that the weights have been tuned to meet the challenge of the training set and not to generalize on unseen data.





The best time to stop the model from training would be at the point where the training and the testing losses/ accuracies are the closest in the epoch v/s accuracy/ loss curve.

Causes of overfitting:

1. More likely with non parametric models
2. More likely with less data
3. More likely with very less or very high number of trainable parameters
4. Less variety in training data
5. Long run of epochs
6. Inappropriate learning rate

Overfitting implies that the model has trained itself to adjust to the randomness of the input data and not the randomness of the test data. This might severely affect the decision-making power of the machine in real time. Many techniques have been devised to overcome the overfitting problem. This process is known as regularization. Regularization techniques that are famous in CNNs are:

1. L1 and L2 regularization
2. Dropout
3. Early Stopping
4. Data augmentation

Data Augmentation:

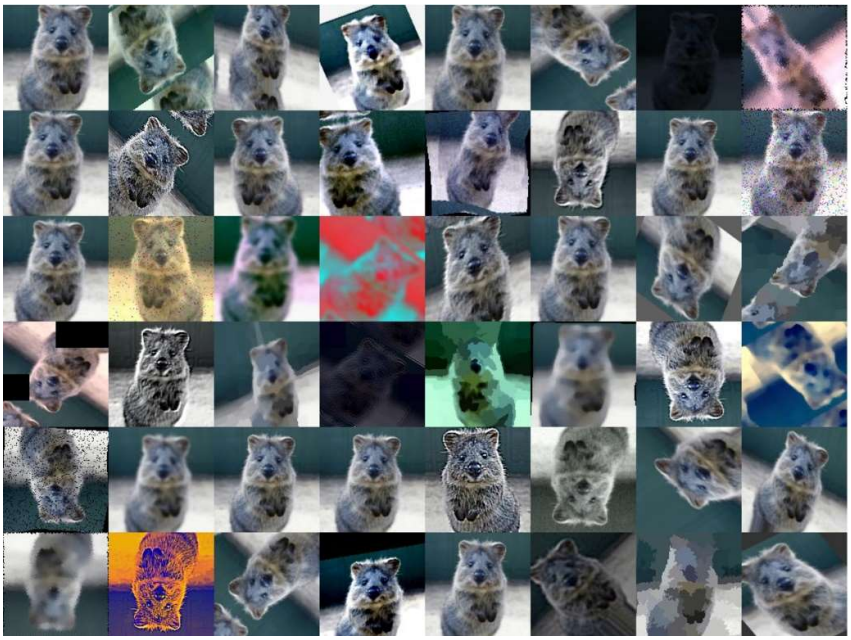
Data augmentation is used for either small datasets or when the training data does not represent its domain in terms of variety.

In data augmentation, the label is maintained but a variety of different geometric operations are performed on the input data. Some of these operations include:

1. Scaling
2. Translation
3. Rotation
4. Flipping
5. Adding impulse noise
6. Illumination
7. Perspective transform

Such manipulations of the data result in more training data and more variety to set the intuition of the training model. Note that data augmentation is only used on the training set and not he validation set.

Effect of data augmentation ref [5]



Data augmentation

Why data augmentation works-

If there are a large number of training variables, then there needs to be a proportionally large amount of data to train on. This might not be obtainable through manual means. Therefore, augmenting the data produces a variety of the same image for the network to train on.

There are two types of data augmentation: online and offline data augmentation. Offline augmentation is for small datasets where the increase in size can be afforded by double or triple the size of the input. However, online augmentation is for large datasets where such an increase in size is not needed. In this case, the mini batches are augmented.

Why CNNs work better than the traditional approach?

There are several reasons why CNNs work better than the traditional approach

1. **Number of parameters – Scaling problem:**
   Artificial neural networks use a flattened version of the previous layer as input. But images are essentially more than just one dimension. As a result, we would have to flatten the image in order to connect the neurons from that layer to the next layer. For small images, the flattened array would be of size 28 x 28. **For example,** in **the case of MNIST** which is not a very big number but take the VGGNet for example, where the input is 224 x 224 – That would be 50,127 parameters in the first layer. If the image is RGB it would be above 150000 parameters. Such connectivity is a waste of resources and might even lead to overfitting.

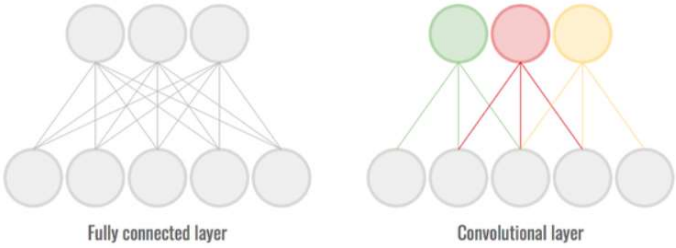
2. **Location invariance:**
   Sharing of parameters allows CNNs to take advantage of local spatial coherence. Take for example a face in an image. A CNN will have a filter that can extract different features of the face irrespective of the location of the face. Therefore, it can learn the same feature extraction for any location in that image. This would have to be encoded into a regular neural network.

3. **Designed for images/ Performance:**
   Each layer of the CONV network has more than just one dimension. Different filters are designed to extract different feature maps. As a result, this results in much better performance. Take for example, the ResNet, which can predict 1000 output classes. This would be computationally infeasible for any other traditional methods.

4. **Feature learning:**
   Traditional methods require features to be encoded. Features have to be extracted for domain specific problems but for CNNs, the network automatically learns the features while training and with the help of feature maps, saliency maps and heat maps predicts the output class to which the input image pertains. Thus, features are automatically learned.

5. **Parameter sharing:**

The CONV layer tends itself to parameter sharing. Each of the 28 x 28 inputs in the image use the slice of the depth (6 filters). This results in 6 x 5 x 5 + 6 = 156 filters instead of 28 x 28 = 784 connections. This eases the computational load and leads to better performance as well. As mentioned above, it also facilitates spatial invariance.



6. **Data driven:** A CNN is data driven. Traditional mathematically driven models will not be able to generalize to different cases as a CNN can.

Loss function:

- Optimization is the procedure of iterating through the data and adjusting the weights so that the right combination of weights can be found which generalizes the mathematical equation that is used to predict the output label for a given class of data.
- As we calculate the error involved while predicting the data, we take the gradient of the error and try to navigate down the gradient.
- The function we use to evaluate a set of weights is called the objective function. When we try to minimize the objective function, we call it the *loss* function. Therefore, we are minimizing the gradient of errors.
- To calculate the error of the model during optimization, a loss function must be chosen.
- A loss function estimates how closely the distribution of predictions made approximates the target distribution of ground truths.

Choosing a loss function depends on the problem being considered. There are different types of loss functions:

1. Mean Square Error Loss:
    - It is calculated by taking the mean squared loss between the target and the predicted values
    - The result is always positive
    - For higher magnitudes of MSE loss, there is a larger gap between the target and the predicted label.
    - If the target and the predicted label are correct, then the MSE loss is 0.
    - It is primarily used for *regression* problems.
    - There are variations of the SE loss such as the logarithmic SE loss and the absolute SE loss. However, these are all used for machine learning problems which have a continuous input range such as regression.

2. Binary Cross Entropy:

- Cross entropy calculates a score that is derived from the average difference between the ground truth label and the predicted label
- It is used for 2 class classification problems

3. Categorical Cross Entropy:
    - It is used in multiclass classification problems
    - It compares the predicted value to the target value and a penalty score is calculated on the logarithmic scale in case of wrong classification
    - A model that predicts the correct values has cross entropy of 0
    - One requirement for categorical cross entropy is that the target labels have to be one hot encoded.

Backpropagation in a CNN:

First, we will observe how backpropagation takes place in a neural network and then moce to CNNs. Backpropagation can be explained as the process by which a quantity is propagated backwards to adjust the weights so as to encourage correct classification. Mathematically, it is defined as the process by which we calculate the partial derivative of the cost with respect to the weights and biases i.e. we see how much the cost function or the loss function varies with changes in the weights and biases.

Forward propagation: A bird eye view of forward propagation reveals that it is the process of calculating intermediate stages of the network as the data propagates through the network. At the end of forward propagation, we will have a predicted label which corresponds to the multitude of computations involved to get to an output probability.

In forward propagation, we take the dot product between the node and the input data and then apply it to an activation function for non linearity. This forms the input for the node in the very next layer.

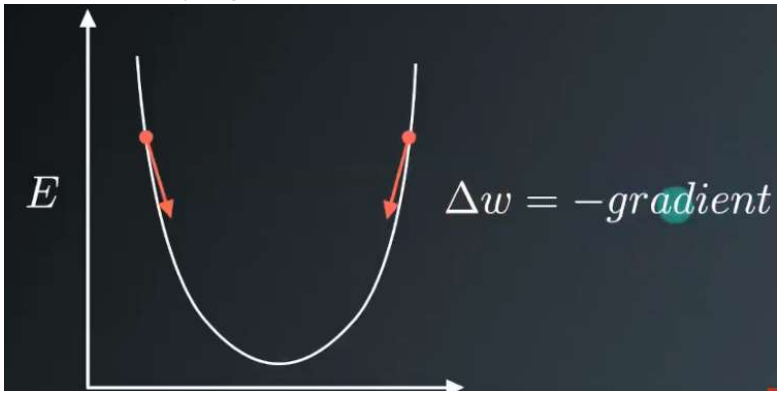
At the end of forward propagation, we can compute the error between the target and the predicted value. This error will tell us how to adjust the weights in order to encourage correct classification. Starting at a random initial value, we take a step in the direction of the least error. This is the direction of steepest gradient descent.

A neural network is a massive composite function. Each layer is a function of the previous layer i.e. composite. The chain rule states that the partial derivate of the composite function is so:

$$\frac{d}{dx}[f(g(x))] = f'(g(x))g'(x)$$

Chain rule

The purpose of backpropagation is to figure out the partial derivative of the error with respect to that individual weight in the network. Backpropagation applies the chain rule to all the weights through all possible paths in a network. It observes how the cost function changes with the change in weight.



E v/s weight

In general, backpropagation for the neural network would follow this algorithm:

1. Compute the dot product and add the bias.
2. Compute the activation.
3. Feed forward: compute the composite function through the layers
4. Calculate the error
5. Backpropagate the error
6. Calculate the gradient of the cost function and use it to optimize the next backpropagation routine.

In CNN, we have a similar algorithm, but the output of each neuron is not just h = wx + b. Instead, each neuron is a result of convolution. Therefore, before applying activation, the output of a neuron looks like so:

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

Each weight in the filter corresponds to each pixel in the output. A change in the weight, even one, will affect the output of that particular neuron. Thus, we calculate the partial derivative with respect to the weights:

$$\partial W_{11} = X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$$

In CNN, first we initialize the filter weights:

For the first convolution layer, we calculate:

$$C_p^1(i,j) = \sigma\left(\sum_{u=-2}^{2}\sum_{v=-2}^{2} I(i-u, j-v)\cdot k_{1,p}^1(u,v) + b_p^1\right)$$

where sigma is the activation function and the dot represents the convolution operation. There are two summations because the convolution is 2D. Similarly, we calculate the output neuron magnitude for the other layers:

Pool layer 1:

$$S_p^1(i,j) = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} C_p^1(2i-u, 2j-v),\ i,j = 1,2,\cdots,12$$

CONV layer 2:

$$C_q^2(i,j) = \sigma\left(\sum_{p=1}^{6}\sum_{u=-2}^{2}\sum_{v=-2}^{2} S_p^1(i-u, j-v)\cdot k_{p,q}^2(u,v) + b_q^2\right)$$

Pool layer 2:

$$S_q^2(i,j) = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} C_q^2(2i-u, 2j-v),\ i,j = 1,2,\cdots,4$$

Fully connected layer:

$$\hat{y} = \sigma(W \times f + b)$$

Loss function: This depends on what we choose. Since we have chosen categorical cross entropy, we use a variant of the binary cross entropy given by

$$H(p,q) = \mathrm{E}_p[-\log q].$$

where p and q are from the target and predicted distributions.

Back propagation is done using the chain rule:

$$\begin{aligned}\Delta W(i,j) &= \frac{\partial L}{\partial W(i,j)} \\ &= \frac{\partial L}{\partial \hat{y}(i)} \cdot \frac{\partial \hat{y}(i)}{\partial W(i,j)}\end{aligned}$$

We do this backward through the layers. The updates performed on the weights are calculated like so:

$$\begin{aligned} k_{1,p}^1 &\leftarrow k_{1,p}^1 - \alpha \cdot \Delta k_{1,p}^1 \\ b_p^1 &\leftarrow b_p^1 - \alpha \cdot \Delta b_p^1 \\ k_{p,q}^2 &\leftarrow k_{p,q}^2 - \alpha \cdot \Delta k_{p,q}^2 \\ b_q^2 &\leftarrow b_q^2 - \alpha \cdot \Delta b_q^2 \end{aligned}$$

where k1 and k2 refer to the weights in the first and the second CONV layers and alpha refers to the learning rate which is between 0 and 1.

To summarize, the following is the algorithm for backpropagation in CNNs:

1. Calculate the convoluted output neuron and the resulting activation
2. Feedforward: For each node, compute the output and the activation through the forward pass

3. Compute the output error:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

4. Backpropagate the error :

$$\delta^l_{x,y} = \delta^{l+1} * ROT180(w^{l+1}_{x,y})\sigma'(z^l_{x,y})$$

5. Calculate the gradient of the cost function:

$$\frac{\partial C}{\partial w^l_{a,b}} = \delta^l_{a,b} * \sigma(ROT180(z^{l-1}_{a,b}))$$

The rotation of weights is a result of the derivation of the delta error in CNNs.

(b)

Train Le-Net5 on MNIST dataset

1. **Abstract and Motivation:**

   LeNet5, a seven layer CNN designed by LeCun in 1998 was used to classify digits and was applied by several banks and post offices for the recognition of handwritten digits. Here, we test LeNet on MNIST and tweak the parameters to check the accuracy of the designed network. We also analyse the results using confusion matrices and classification reports and post layer feature maps.

2. **Approach and Procedures:**

   Choice of parameters in LeNet:

   kernel size is (5,5): This is a common choice. A smaller kernel might give better accuracy but a bigger kernel such as this does not compromise accuracy while also reducing computation time.
   stride is 1: A higher stride will cause loss of information
   max pool is (2x2) with stride 2: higher settings will be destructive to the integrity of the image.
   The dense layer settings are optimal settings. There is a particular number of connections in the dense layer beyond which the accuracy will not improve but the computational cost will rise. 120 and 84 are trial and error parameters which are chosen after repetitive testing. The last sense layer has only 10 connections because there are 10 output class labels.

   The input size is 32 x 32: The reason behind this is that the distinctive features should appear at the center of the receptive fields This is so that potential discriminating features are not lost.

   The following 10 settings were adopted. **A summary of the settings are presented towards the end with representative 5 settings chosen along with relevant and required statistics.**

   Setting 1 : Original LeNet Setting

   Setting 2: Lesser filters in first CONV layer, more filters in second CONV layer

   Setting 3: More filters in first CONV layer, lesser filters in second CONV layer

   Setting 4: More filters in both CONV layers

   Setting 5: Decrease kernel size

   Setting 6: Increase dense units

   Setting 7: Decrease pool size

   Setting 8: Learning rate

Setting 9: Number of epochs

Setting 10: Batch Normalization and Dropout.

The following techniques were adopted additionally:

1. Early stopping: This is almost employed at all settings with a 'patience' of 3. It is employed to avoid overfitting. If the validation accuracy tends to not show any improvement over 3 epochs, the training is stopped and the best model is saved. Note that while reporting, two accuracies are reported. **The early stopping accuracy is the best model validation accuracy.** This is the best model accuracy for unseen data. The model and weights are saved and tested on unseen data. **The stopped model accuracy** is the accuracy after all the model has stopped i.e. shown no improvement in validation accuracy. This will not be as good as the early stopped model. Nonetheless, it is recorded and presented.

2. Batch normalization and dropout: Batch normalization is used to normalized all the responses after a single convolutional layer to reduce the internal covariate shift. It improves speed and performance of the network. Dropout renders a percent of the total neurons as inactive during training. This is used to prevent overfitting.

3. **Results:**

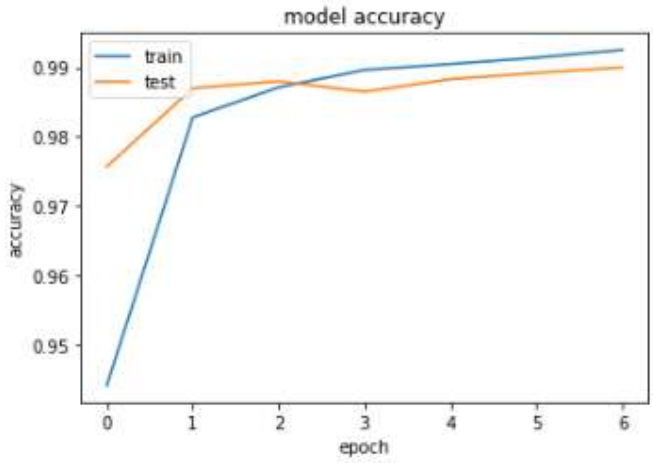
Setting 1: Original LeNet Setting

Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	6	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	16	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 8
Batch- size: None

Epoch – Accuracy Plot:



The model has run for 6 epochs. From the plot, we observe that the model start overfitting around epoch number 3. The model performs well on test data with an accuracy of above 98%.

Royston Marian Mascarenhas, Spring 19, USC

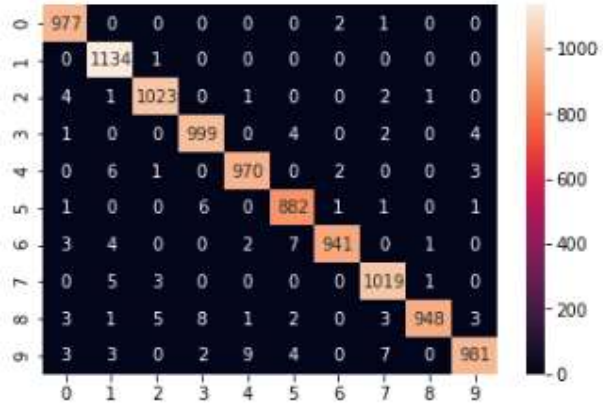
Accuracy:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 [==============================] - 50s 835us/step - loss: 0.1709 - mean_absolute_error: 0.0166 - acc: 0.9471 - val_
loss: 0.0514 - val_mean_absolute_error: 0.0050 - val_acc: 0.9831
Epoch 2/15
60000/60000 [==============================] - 46s 773us/step - loss: 0.0576 - mean_absolute_error: 0.0049 - acc: 0.9827 - val_
loss: 0.0604 - val_mean_absolute_error: 0.0046 - val_acc: 0.9815
Epoch 3/15
60000/60000 [==============================] - 44s 736us/step - loss: 0.0436 - mean_absolute_error: 0.0035 - acc: 0.9874 - val_
loss: 0.0393 - val_mean_absolute_error: 0.0033 - val_acc: 0.9880
Epoch 4/15
60000/60000 [==============================] - 43s 721us/step - loss: 0.0364 - mean_absolute_error: 0.0029 - acc: 0.9896 - val_
loss: 0.0350 - val_mean_absolute_error: 0.0029 - val_acc: 0.9888
Epoch 5/15
60000/60000 [==============================] - 44s 737us/step - loss: 0.0331 - mean_absolute_error: 0.0024 - acc: 0.9905 - val_
loss: 0.0336 - val_mean_absolute_error: 0.0026 - val_acc: 0.9894
Epoch 6/15
60000/60000 [==============================] - 44s 735us/step - loss: 0.0290 - mean_absolute_error: 0.0020 - acc: 0.9923 - val_
loss: 0.0479 - val_mean_absolute_error: 0.0035 - val_acc: 0.9861
Epoch 7/15
60000/60000 [==============================] - 44s 733us/step - loss: 0.0270 - mean_absolute_error: 0.0019 - acc: 0.9925 - val_
loss: 0.0552 - val_mean_absolute_error: 0.0027 - val_acc: 0.9877
Epoch 8/15
60000/60000 [==============================] - 44s 742us/step - loss: 0.0248 - mean_absolute_error: 0.0018 - acc: 0.9931 - val_
loss: 0.0537 - val_mean_absolute_error: 0.0026 - val_acc: 0.98777 - mean_absolute_error: 0.0017 - acc
Epoch 00008: early stopping
```

Training accuracy: Early stopping accuracy - 99.05% Stopped model accuracy – 99.31%

Test accuracy: Early stopping accuracy – 98.94% Stopped model accuracy – 98.77%

Confusion Matrix :



Classification Report:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	996
1	1.00	0.99	0.99	1139
2	0.98	1.00	0.99	1018
3	0.99	0.99	0.99	1013
4	0.98	0.99	0.99	970
5	0.99	0.99	0.99	892
6	0.98	1.00	0.99	943
7	0.99	0.99	0.99	1034
8	0.98	0.99	0.99	971
9	0.99	0.97	0.98	1024
avg / total	0.99	0.99	0.99	10000

Royston Marian Mascarenhas, Spring 19, USC

Setting 1A : Note: Adam optimizer gives accuracy (98.92%)

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 49s 824us/step - loss: 0.1888 - acc: 0.9421 - val_loss: 0.0967 - val_acc: 0.9689
Epoch 2/12
60000/60000 [==============================] - 52s 863us/step - loss: 0.0652 - acc: 0.9793 - val_loss: 0.0661 - val_acc: 0.9785
Epoch 3/12
60000/60000 [==============================] - 52s 862us/step - loss: 0.0464 - acc: 0.9851 - val_loss: 0.0423 - val_acc: 0.9854
Epoch 4/12
60000/60000 [==============================] - 53s 878us/step - loss: 0.0355 - acc: 0.9884 - val_loss: 0.0515 - val_acc: 0.9832
Epoch 5/12
60000/60000 [==============================] - 51s 858us/step - loss: 0.0283 - acc: 0.9912 - val_loss: 0.0346 - val_acc: 0.9877
Epoch 6/12
60000/60000 [==============================] - 52s 869us/step - loss: 0.0237 - acc: 0.9922 - val_loss: 0.0362 - val_acc: 0.9892
```

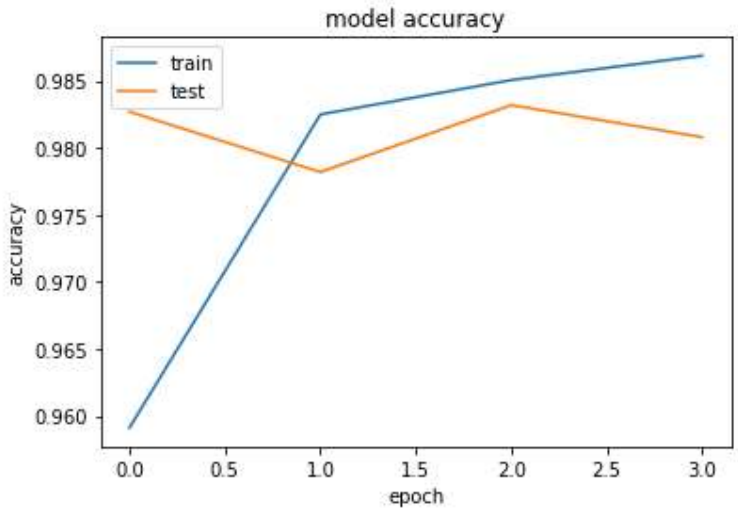
Setting 2: Filters: Lesser filters in the first CONV layer, more filters in the second CONV layer. More dense units.

Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	6	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	16	-	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 4
Batch- size: None

Epoch – Accuracy Curve:



The model has run for 3 epochs due to early stopping. From the plot, we observe that the model start overfitting around epoch number 2 which is very early. This might be because of reduced number of training variables since the filters are less in number, especially in the second CONV layer. The model performs well on test data with an accuracy of above 98%.

Royston Marian Mascarenhas, Spring 19, USC

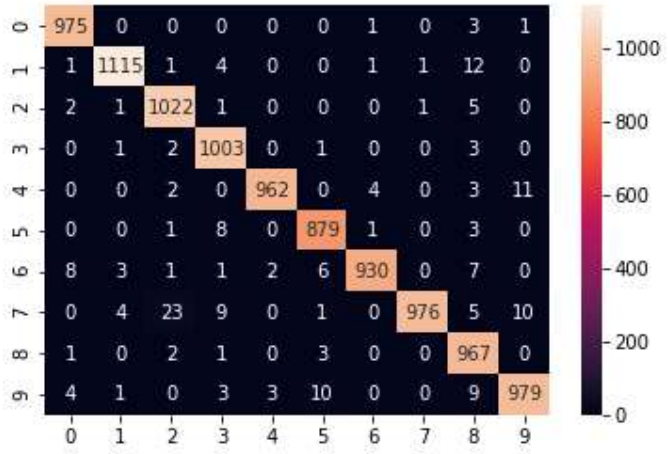
Accuracy:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 [==============================] - 64s 1ms/step - loss: 0.1338 - mean_absolute_error: 0.0120 - acc: 0.9591 - val_lo
ss: 0.0534 - val_mean_absolute_error: 0.0046 - val_acc: 0.9827
Epoch 2/15
60000/60000 [==============================] - 57s 945us/step - loss: 0.0614 - mean_absolute_error: 0.0045 - acc: 0.9825 - val_
loss: 0.0593 - val_mean_absolute_error: 0.0057 - val_acc: 0.9782
Epoch 3/15
60000/60000 [==============================] - 50s 826us/step - loss: 0.0560 - mean_absolute_error: 0.0038 - acc: 0.9851 - val_
loss: 0.0568 - val_mean_absolute_error: 0.0043 - val_acc: 0.9832
Epoch 4/15
60000/60000 [==============================] - 49s 812us/step - loss: 0.0518 - mean_absolute_error: 0.0033 - acc: 0.9869 - val_
loss: 0.0759 - val_mean_absolute_error: 0.0043 - val_acc: 0.9808
Epoch 00004: early stopping
```

Training accuracy: Early stopping accuracy – 95.91 Stopped model accuracy – 98.69

Test accuracy: Early stopping accuracy – 98.27 Stopped model accuracy – 98.08

Confusion matrix:



Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	991
1	0.98	0.99	0.99	1125
2	0.99	0.97	0.98	1054
3	0.99	0.97	0.98	1030
4	0.98	0.99	0.99	967
5	0.99	0.98	0.98	900
6	0.97	0.99	0.98	937
7	0.95	1.00	0.97	978
8	0.99	0.95	0.97	1017
9	0.97	0.98	0.97	1001
avg / total	0.98	0.98	0.98	10000

Setting 3: Filters: More filters in the first CONV layer, less filters in the second CONV layer

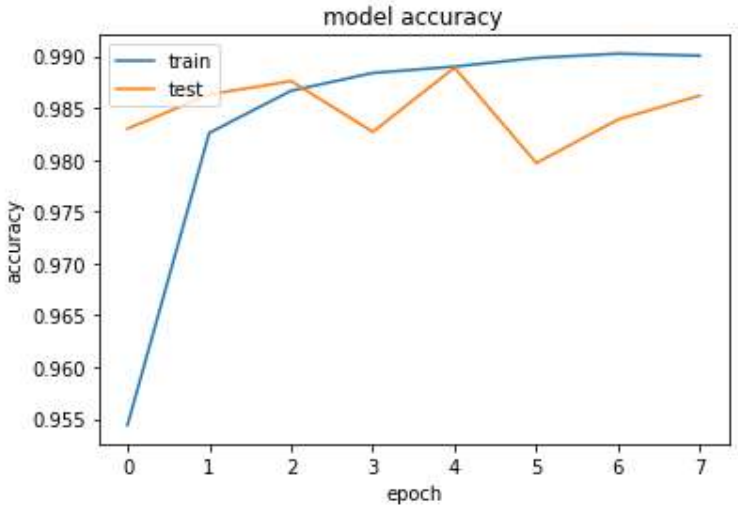
Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	18	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	10	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 8
Batch- size: 16

```
Total params: 46,112
Trainable params: 46,112
Non-trainable params: 0
```

Epoch – Accuracy Curve:



The model has run for 7 epochs. From the plot, we observe that the model start overfitting around epoch number 5. Reduced training variables might be the cause for overfitting, although the number is not as less as the previous case. The model performs well on test data with an accuracy of above 98%.

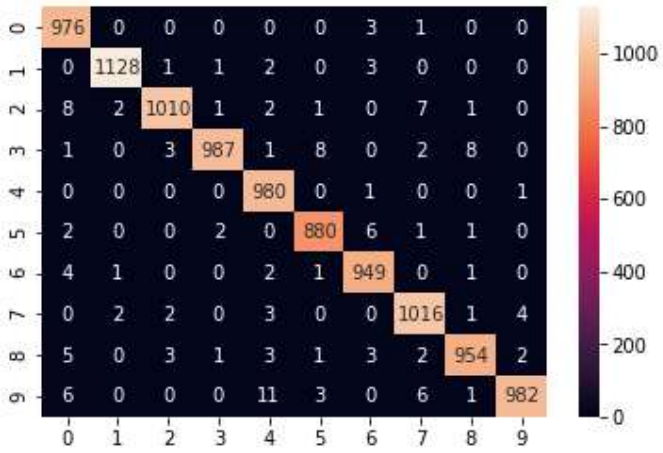
Accuracy:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 [==============================] - 87s 1ms/step - loss: 0.1485 - mean_absolute_error: 0.0134 - acc: 0.9544 - val_lo
ss: 0.0535 - val_mean_absolute_error: 0.0049 - val_acc: 0.9830 loss: 0.1511 - me
Epoch 2/15
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0591 - mean_absolute_error: 0.0046 - acc: 0.9826 - val_lo
ss: 0.0420 - val_mean_absolute_error: 0.0037 - val_acc: 0.9863
Epoch 3/15
60000/60000 [==============================] - 92s 2ms/step - loss: 0.0480 - mean_absolute_error: 0.0033 - acc: 0.9867 - val_lo
ss: 0.0495 - val_mean_absolute_error: 0.0028 - val_acc: 0.9876
Epoch 4/15
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0452 - mean_absolute_error: 0.0029 - acc: 0.9884 - val_lo
ss: 0.0558 - val_mean_absolute_error: 0.0044 - val_acc: 0.9827
Epoch 5/15
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0441 - mean_absolute_error: 0.0027 - acc: 0.9890 - val_lo
ss: 0.0435 - val_mean_absolute_error: 0.0025 - val_acc: 0.9889lu
Epoch 6/15
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0421 - mean_absolute_error: 0.0024 - acc: 0.9898 - val_lo
ss: 0.1041 - val_mean_absolute_error: 0.0046 - val_acc: 0.9797
Epoch 7/15
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0440 - mean_absolute_error: 0.0023 - acc: 0.9902 - val_lo
ss: 0.0647 - val_mean_absolute_error: 0.0055 - val_acc: 0.9839
Epoch 8/15
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0450 - mean_absolute_error: 0.0023 - acc: 0.9900 - val_lo
ss: 0.0866 - val_mean_absolute_error: 0.0030 - val_acc: 0.9862
Epoch 00008: early stopping
```

Training accuracy: Early stopping accuracy – 98.9 Stopped model accuracy - 99

Test accuracy: Early stopping accuracy – 98.89 Stopped model accuracy – 98.62

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0	1.00	0.97	0.98	1002
1	0.99	1.00	0.99	1133
2	0.98	0.99	0.98	1019
3	0.98	0.99	0.99	992
4	1.00	0.98	0.99	1004
5	0.99	0.98	0.99	894
6	0.99	0.98	0.99	965
7	0.99	0.98	0.98	1035
8	0.98	0.99	0.98	967
9	0.97	0.99	0.98	989
avg / total	0.99	0.99	0.99	10000

Setting 4: Filters: Increase the number of filters in each CONV layer

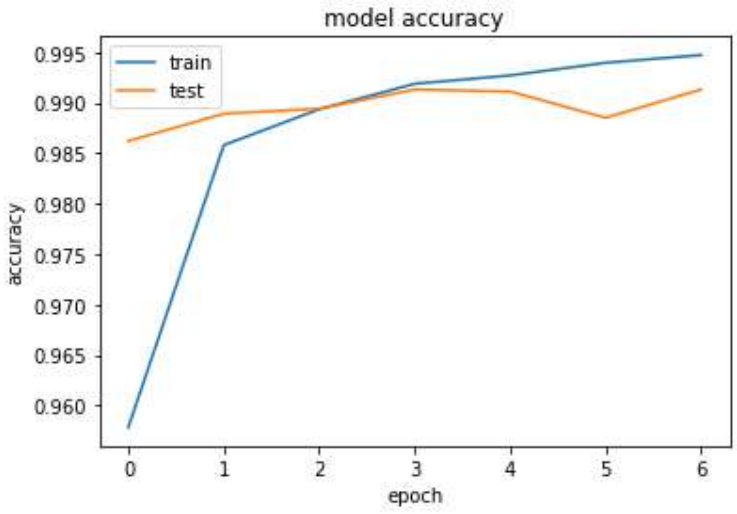
Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	18	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	25	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 7
Batch- size: None

```
Total params: 97,877
Trainable params: 97,877
Non-trainable params: 0
```

Epoch – Accuracy Curve:



The model has run for 6 epochs. From the plot, we observe that the model start overfitting around epoch number 5. The model performs well on test data with an accuracy of above 98%. The model has performed exceedingly well because of increased number of features.

Royston Marian Mascarenhas, Spring 19, USC

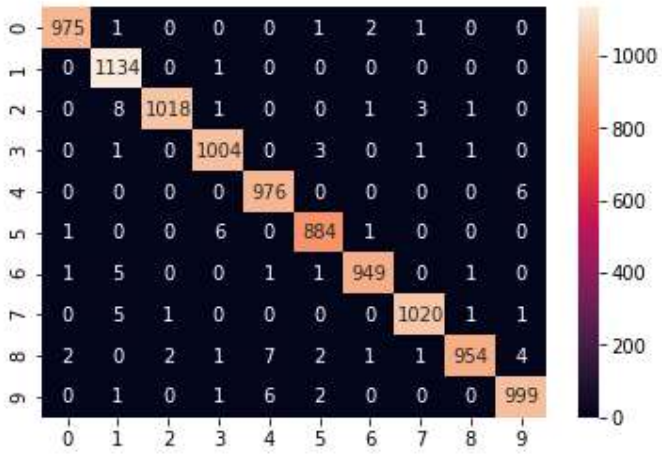
Accuracy:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 [==============================] - 76s 1ms/step - loss: 0.1352 - mean_absolute_error: 0.0129 - acc: 0.9578 - val_lo
ss: 0.0455 - val_mean_absolute_error: 0.0044 - val_acc: 0.9862
Epoch 2/15
60000/60000 [==============================] - 68s 1ms/step - loss: 0.0461 - mean_absolute_error: 0.0039 - acc: 0.9858 - val_lo
ss: 0.0354 - val_mean_absolute_error: 0.0031 - val_acc: 0.9889
Epoch 3/15
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0344 - mean_absolute_error: 0.0027 - acc: 0.9894 - val_lo
ss: 0.0343 - val_mean_absolute_error: 0.0027 - val_acc: 0.9894
Epoch 4/15
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0278 - mean_absolute_error: 0.0021 - acc: 0.9919 - val_lo
ss: 0.0315 - val_mean_absolute_error: 0.0025 - val_acc: 0.9913
Epoch 5/15
60000/60000 [==============================] - 74s 1ms/step - loss: 0.0263 - mean_absolute_error: 0.0019 - acc: 0.9927 - val_lo
ss: 0.0313 - val_mean_absolute_error: 0.0022 - val_acc: 0.9911
Epoch 6/15
60000/60000 [==============================] - 69s 1ms/step - loss: 0.0216 - mean_absolute_error: 0.0015 - acc: 0.9940 - val_lo
ss: 0.0426 - val_mean_absolute_error: 0.0025 - val_acc: 0.9885
Epoch 7/15
60000/60000 [==============================] - 75s 1ms/step - loss: 0.0196 - mean_absolute_error: 0.0013 - acc: 0.9947 - val_lo
ss: 0.0499 - val_mean_absolute_error: 0.0018 - val_acc: 0.9913
Epoch 00007: early stopping
```

Training accuracy: Early stopping accuracy – 99.19 Stopped model accuracy – 99.47

Test accuracy: Early stopping accuracy –99.13 Stopped model accuracy -99.13

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	991
1	0.98	0.99	0.99	1125
2	0.99	0.97	0.98	1054
3	0.99	0.97	0.98	1030
4	0.98	0.99	0.99	967
5	0.99	0.98	0.98	900
6	0.97	0.99	0.98	937
7	0.95	1.00	0.97	978
8	0.99	0.95	0.97	1017
9	0.97	0.98	0.97	1001
avg / total	0.98	0.98	0.98	10000

Setting 5: Kernel Size: Decrease in kernel size

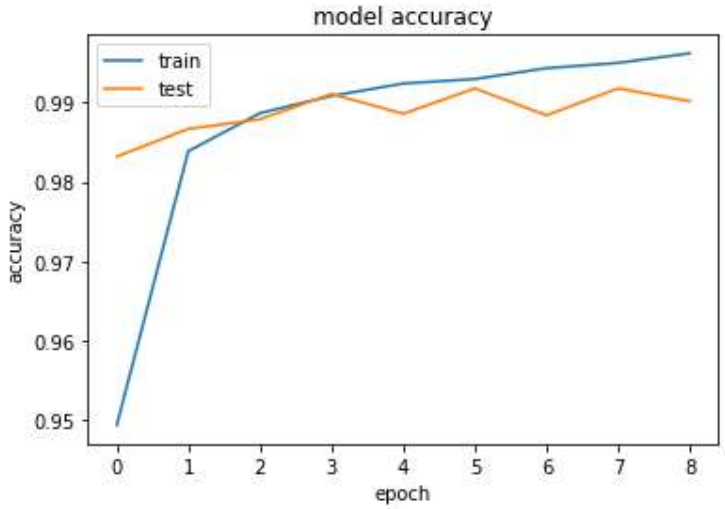
Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	18	(3,3)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	25	(3,3)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 9
Batch- size: None

```
Total params: 123,389
Trainable params: 123,389
Non-trainable params: 0
```

Epoch – Accuracy Curve:



The model has run for 8 epochs. From the plot, we observe that the model start overfitting around epoch number 7. The model performs well on test data with an accuracy of above 98%. The model has performed really well with the highest accuracy till now due to smaller kernel, accurate pixel energies.

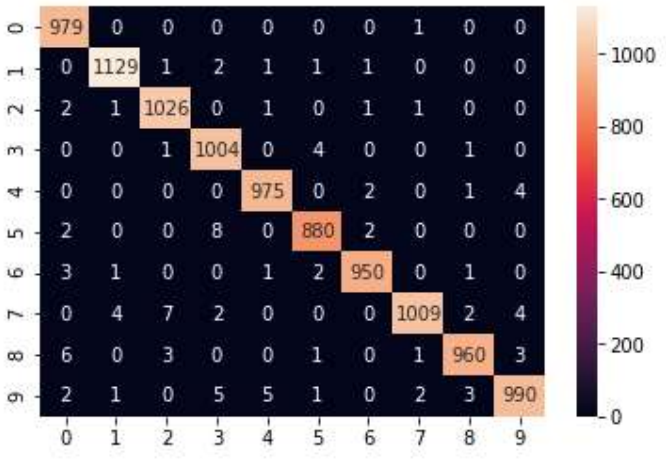
Accuracy:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 [==============================] - 47s 786us/step - loss: 0.1595 - mean_absolute_error: 0.0155 - acc: 0.9493 - val_
loss: 0.0554 - val_mean_absolute_error: 0.0048 - val_acc: 0.9832
Epoch 2/15
60000/60000 [==============================] - 45s 756us/step - loss: 0.0512 - mean_absolute_error: 0.0045 - acc: 0.9839 - val_
loss: 0.0444 - val_mean_absolute_error: 0.0034 - val_acc: 0.9867
Epoch 3/15
60000/60000 [==============================] - 45s 754us/step - loss: 0.0377 - mean_absolute_error: 0.0031 - acc: 0.9887 - val_
loss: 0.0393 - val_mean_absolute_error: 0.0034 - val_acc: 0.987978 - me - ETA: 1s - loss: 0.0373 - mean_absolute
Epoch 4/15
60000/60000 [==============================] - 44s 732us/step - loss: 0.0303 - mean_absolute_error: 0.0024 - acc: 0.9908 - val_
loss: 0.0300 - val_mean_absolute_error: 0.0024 - val_acc: 0.9911
Epoch 5/15
60000/60000 [==============================] - 45s 745us/step - loss: 0.0247 - mean_absolute_error: 0.0020 - acc: 0.9924 - val_
loss: 0.0443 - val_mean_absolute_error: 0.0027 - val_acc: 0.9886
Epoch 6/15
60000/60000 [==============================] - 46s 760us/step - loss: 0.0230 - mean_absolute_error: 0.0017 - acc: 0.9930 - val_
loss: 0.0297 - val_mean_absolute_error: 0.0021 - val_acc: 0.9918
Epoch 7/15
60000/60000 [==============================] - 46s 763us/step - loss: 0.0195 - mean_absolute_error: 0.0014 - acc: 0.9943 - val_
loss: 0.0460 - val_mean_absolute_error: 0.0028 - val_acc: 0.9884
Epoch 8/15
60000/60000 [==============================] - 46s 772us/step - loss: 0.0177 - mean_absolute_error: 0.0013 - acc: 0.9950 - val_
loss: 0.0379 - val_mean_absolute_error: 0.0019 - val_acc: 0.9918
Epoch 9/15
60000/60000 [==============================] - 45s 755us/step - loss: 0.0143 - mean_absolute_error: 9.8581e-04 - acc: 0.9962 -
val_loss: 0.0506 - val_mean_absolute_error: 0.0021 - val_acc: 0.9902
Epoch 00009: early stopping
```

Training accuracy: Early stopping accuracy – 99.3 Stopped model accuracy – 99.62

Test accuracy: Early stopping accuracy – 99.18 Stopped model accuracy – 99.02

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	991
1	0.98	0.99	0.99	1125
2	0.99	0.97	0.98	1054
3	0.99	0.97	0.98	1030
4	0.98	0.99	0.99	967
5	0.99	0.98	0.98	900
6	0.97	0.99	0.98	937
7	0.95	1.00	0.97	978
8	0.99	0.95	0.97	1017
9	0.97	0.98	0.97	1001
avg / total	0.98	0.98	0.98	10000

Setting 6: FC layer connections: More number of fully connected units in dense layer

Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	18	(3,3)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	25	(3,3)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 250	-	-	-
Dense Layer 2	units = 170	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)

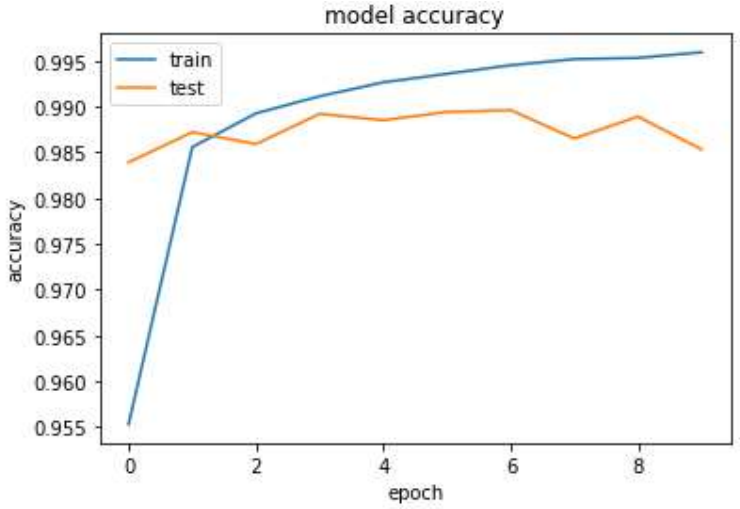
Loss: Categorical Cross Entropy

Epochs: 10

Batch- size: None

```
Total params: 273,885
Trainable params: 273,885
Non-trainable params: 0
```

Epoch – Accuracy Curve:



The model has run for 10 epochs. From the plot, we observe that the model start overfitting around epoch number 6. The model performs well on test data with an accuracy of above 98%.

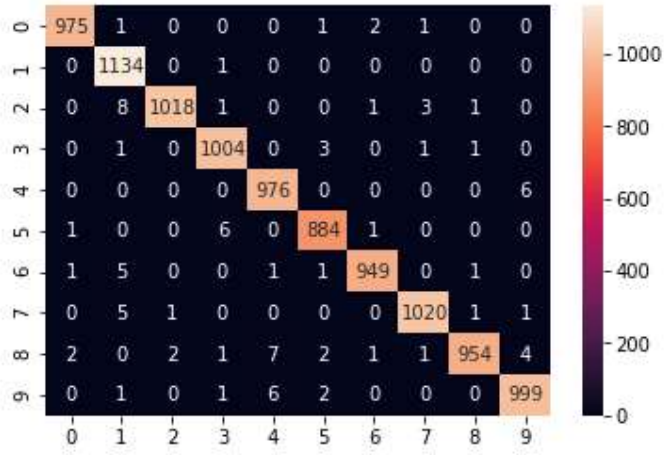
Accuracy:

```
loss: 0.0492 - val_mean_absolute_error: 0.0050 - val_acc: 0.9839
Epoch 2/15
60000/60000 [==============================] - 53s 886us/step - loss: 0.0480 - mean_absolute_error: 0.0039 - acc: 0.9856 - val_
loss: 0.0415 - val_mean_absolute_error: 0.0033 - val_acc: 0.9872
Epoch 3/15
60000/60000 [==============================] - 55s 909us/step - loss: 0.0365 - mean_absolute_error: 0.0028 - acc: 0.9893 - val_
loss: 0.0483 - val_mean_absolute_error: 0.0031 - val_acc: 0.9859
Epoch 4/15
60000/60000 [==============================] - 58s 968us/step - loss: 0.0303 - mean_absolute_error: 0.0022 - acc: 0.9911 - val_
loss: 0.0407 - val_mean_absolute_error: 0.0026 - val_acc: 0.9892
Epoch 5/15
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0261 - mean_absolute_error: 0.0018 - acc: 0.9926 - val_lo
ss: 0.0418 - val_mean_absolute_error: 0.0026 - val_acc: 0.9885
Epoch 6/15
60000/60000 [==============================] - 54s 896us/step - loss: 0.0235 - mean_absolute_error: 0.0016 - acc: 0.9936 - val_
loss: 0.0449 - val_mean_absolute_error: 0.0024 - val_acc: 0.9894 ETA: 0s - loss: 0.0233 - mean_absolute_error: 0.0016 - acc: 0.
9
Epoch 7/15
60000/60000 [==============================] - 51s 854us/step - loss: 0.0222 - mean_absolute_error: 0.0014 - acc: 0.9945 - val_
loss: 0.0535 - val_mean_absolute_error: 0.0022 - val_acc: 0.9896
Epoch 8/15
60000/60000 [==============================] - 51s 846us/step - loss: 0.0192 - mean_absolute_error: 0.0012 - acc: 0.9952 - val_
loss: 0.0882 - val_mean_absolute_error: 0.0027 - val_acc: 0.9865
Epoch 9/15
60000/60000 [==============================] - 52s 870us/step - loss: 0.0185 - mean_absolute_error: 0.0011 - acc: 0.9953 - val_
loss: 0.0611 - val_mean_absolute_error: 0.0024 - val_acc: 0.9889
Epoch 10/15
60000/60000 [==============================] - 51s 846us/step - loss: 0.0178 - mean_absolute_error: 9.5486e-04 - acc: 0.9959 -
val_loss: 0.0619 - val_mean_absolute_error: 0.0034 - val_acc: 0.9853
Epoch 00010: early stopping
```

Training accuracy: Early stopping accuracy – 99.45 Stopped model accuracy – 99.59

Test accuracy: Early stopping accuracy – 98.96 Stopped model accuracy – 98.53

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	991
1	0.98	0.99	0.99	1125
2	0.99	0.97	0.98	1054
3	0.99	0.97	0.98	1030
4	0.98	0.99	0.99	967
5	0.99	0.98	0.98	900
6	0.97	0.99	0.98	937
7	0.95	1.00	0.97	978
8	0.99	0.95	0.97	1017
9	0.97	0.98	0.97	1001
avg / total	0.98	0.98	0.98	10000

Setting 7: Max Pool Size: Increase the kernel of the max pool layer

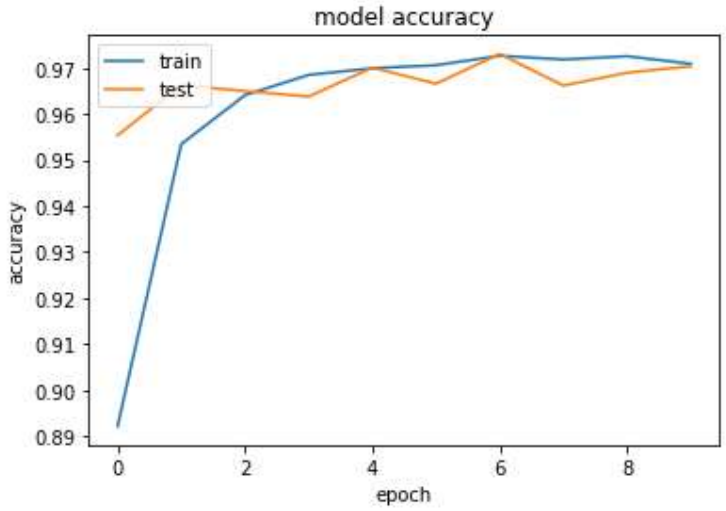
Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	6	(5,5)	1	relu
MaxPool	N/A	(3,3)	-	-
Conv2D	16	(5,5)	1	relu
MaxPool	N/A	(3,3)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 10
Batch- size: None

```
Total params: 15,626
Trainable params: 15,626
```

Epoch – Accuracy Curve:



The model has run for 6 epochs. From the plot, we observe that the model start overfitting around epoch number 3. The performance is bad because the dense layers has only 1 x 1 frames to use before flattening: not much discrimination.

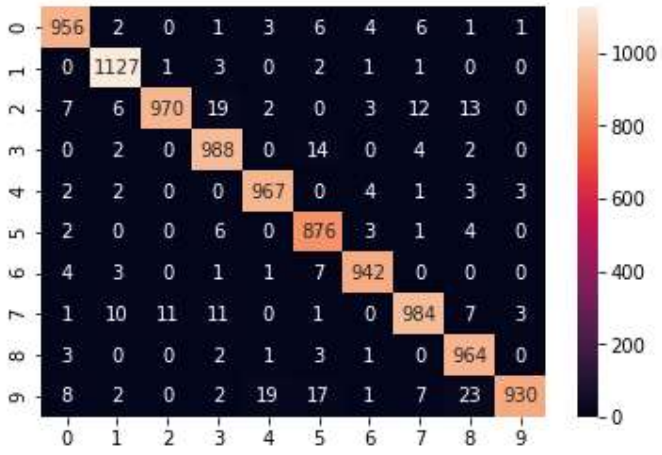
Accuracy:

```
loss: 0.1485 - val_mean_absolute_error: 0.0128 - val_acc: 0.9554
Epoch 2/15
60000/60000 [==============================] - 48s 795us/step - loss: 0.1525 - mean_absolute_error: 0.0130 - acc: 0.9535 - val_
loss: 0.1107 - val_mean_absolute_error: 0.0094 - val_acc: 0.9663
Epoch 3/15
60000/60000 [==============================] - 49s 811us/step - loss: 0.1225 - mean_absolute_error: 0.0098 - acc: 0.9641 - val_
loss: 0.1190 - val_mean_absolute_error: 0.0092 - val_acc: 0.9650
Epoch 4/15
60000/60000 [==============================] - 46s 764us/step - loss: 0.1104 - mean_absolute_error: 0.0085 - acc: 0.9685 - val_
loss: 0.1140 - val_mean_absolute_error: 0.0099 - val_acc: 0.9638
Epoch 5/15
60000/60000 [==============================] - 44s 733us/step - loss: 0.1063 - mean_absolute_error: 0.0079 - acc: 0.9700 - val_
loss: 0.1075 - val_mean_absolute_error: 0.0074 - val_acc: 0.9701
Epoch 6/15
60000/60000 [==============================] - 45s 744us/step - loss: 0.1061 - mean_absolute_error: 0.0075 - acc: 0.9706 - val_
loss: 0.1278 - val_mean_absolute_error: 0.0083 - val_acc: 0.9666
Epoch 7/15
60000/60000 [==============================] - 44s 740us/step - loss: 0.1063 - mean_absolute_error: 0.0073 - acc: 0.9727 - val_
loss: 0.0948 - val_mean_absolute_error: 0.0071 - val_acc: 0.9731
Epoch 8/15
60000/60000 [==============================] - 46s 767us/step - loss: 0.1092 - mean_absolute_error: 0.0074 - acc: 0.9719 - val_
loss: 0.1496 - val_mean_absolute_error: 0.0085 - val_acc: 0.9662
Epoch 9/15
60000/60000 [==============================] - 52s 859us/step - loss: 0.1105 - mean_absolute_error: 0.0071 - acc: 0.9726 - val_
loss: 0.1683 - val_mean_absolute_error: 0.0068 - val_acc: 0.9690
Epoch 10/15
60000/60000 [==============================] - 47s 775us/step - loss: 0.1144 - mean_absolute_error: 0.0074 - acc: 0.9709 - val_
loss: 0.1074 - val_mean_absolute_error: 0.0074 - val_acc: 0.9704
Epoch 00010: early stopping
```

Training accuracy: Early stopping accuracy – 97.27 Stopped model accuracy – 97.09

Test accuracy: Early stopping accuracy – 97.31 Stopped model accuracy – 97.04

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0	0.98	0.97	0.97	983
1	0.99	0.98	0.98	1154
2	0.94	0.99	0.96	982
3	0.98	0.96	0.97	1033
4	0.98	0.97	0.98	993
5	0.98	0.95	0.96	926
6	0.98	0.98	0.98	959
7	0.96	0.97	0.96	1016
8	0.99	0.95	0.97	1017
9	0.92	0.99	0.96	937
avg / total	0.97	0.97	0.97	10000

Setting 8: Learning rate: High and low learning rate in optimizer

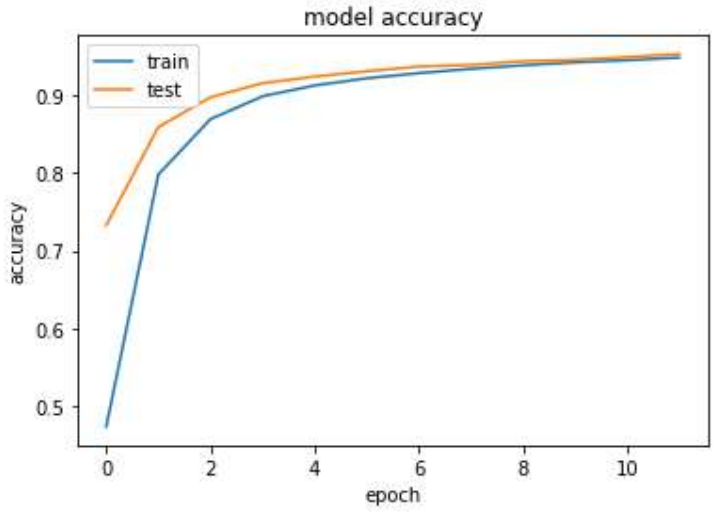
Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	6	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	16	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 12 for lr=0.00001
Batch- size: None

```
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
```

Epoch – Accuracy Curve:



Here, we do not see the usual effect of overfitting since we use a small learning rate. As a result, it takes a long time for the result to converge. Performance is not bad, just that it will take more epochs to converge. Computationally more complex.

Accuracy:

Learning rate = 1

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 50s 829us/step - loss: 14.5220 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.09
```

Learning rate = 0.00001

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 58s 959us/step - loss: 1.9412 - acc: 0.4743 - val_loss: 1.2765 - val_acc: 0.7326
Epoch 2/12
60000/60000 [==============================] - 63s 1ms/step - loss: 0.8417 - acc: 0.7976 - val_loss: 0.5613 - val_acc: 0.8584
Epoch 3/12
60000/60000 [==============================] - 59s 976us/step - loss: 0.4825 - acc: 0.8685 - val_loss: 0.3926 - val_acc: 0.8966
Epoch 4/12
60000/60000 [==============================] - 52s 860us/step - loss: 0.3662 - acc: 0.8979 - val_loss: 0.3157 - val_acc: 0.9147
Epoch 5/12
60000/60000 [==============================] - 51s 851us/step - loss: 0.3090 - acc: 0.9117 - val_loss: 0.2750 - val_acc: 0.9232
Epoch 6/12
60000/60000 [==============================] - 51s 843us/step - loss: 0.2740 - acc: 0.9209 - val_loss: 0.2473 - val_acc: 0.9299
Epoch 7/12
60000/60000 [==============================] - 51s 846us/step - loss: 0.2491 - acc: 0.9276 - val_loss: 0.2257 - val_acc: 0.9360
Epoch 8/12
60000/60000 [==============================] - 51s 849us/step - loss: 0.2302 - acc: 0.9330 - val_loss: 0.2121 - val_acc: 0.9380
Epoch 9/12
60000/60000 [==============================] - 50s 833us/step - loss: 0.2148 - acc: 0.9376 - val_loss: 0.1953 - val_acc: 0.9429
Epoch 10/12
60000/60000 [==============================] - 51s 846us/step - loss: 0.2018 - acc: 0.9415 - val_loss: 0.1857 - val_acc: 0.9445
Epoch 11/12
60000/60000 [==============================] - 52s 867us/step - loss: 0.1904 - acc: 0.9442 - val_loss: 0.1742 - val_acc: 0.9481
Epoch 12/12
60000/60000 [==============================] - 54s 898us/step - loss: 0.1804 - acc: 0.9475 - val_loss: 0.1653 - val_acc: 0.9520
```

Accuracy on training set: 94.75
Accuracy on test set: 95.2

Setting 9: Number of epochs: Disable early stopping. Run for 30 epochs.

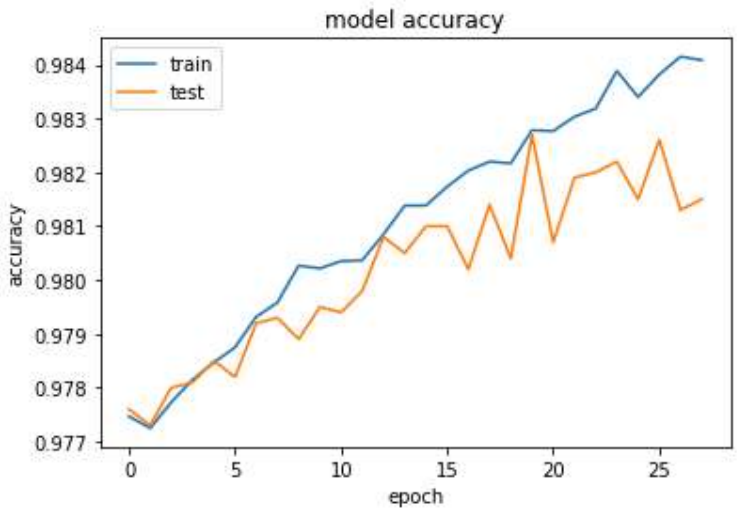
Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	6	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	16	(5,5)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 30
Batch- size: None

```
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
```

Epoch – Accuracy Curve:



The number of epochs have resulted in severe overfitting as can be observed from the curve with the rise in the training curve accuracy and the dip in the testing accuracy curve.

Accuracy:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 55s 912us/step - loss: 0.0772 - acc: 0.9775 - val_loss: 0.0696 - val_acc: 0.9776
Epoch 2/30
60000/60000 [==============================] - 41s 686us/step - loss: 0.0764 - acc: 0.9772 - val_loss: 0.0692 - val_acc: 0.9773
Epoch 3/30
60000/60000 [==============================] - 40s 665us/step - loss: 0.0751 - acc: 0.9777 - val_loss: 0.0695 - val_acc: 0.9780
Epoch 4/30
60000/60000 [==============================] - 40s 666us/step - loss: 0.0738 - acc: 0.9781 - val_loss: 0.0667 - val_acc: 0.9781
Epoch 5/30
60000/60000 [==============================] - 40s 666us/step - loss: 0.0729 - acc: 0.9785 - val_loss: 0.0654 - val_acc: 0.9785
Epoch 6/30
60000/60000 [==============================] - 41s 676us/step - loss: 0.0717 - acc: 0.9788 - val_loss: 0.0646 - val_acc: 0.9782
Epoch 7/30
60000/60000 [==============================] - 41s 688us/step - loss: 0.0706 - acc: 0.9793 - val_loss: 0.0639 - val_acc: 0.9792
Epoch 8/30
60000/60000 [==============================] - 42s 703us/step - loss: 0.0696 - acc: 0.9796 - val_loss: 0.0627 - val_acc: 0.9793
Epoch 9/30
60000/60000 [==============================] - 42s 700us/step - loss: 0.0687 - acc: 0.9803 - val_loss: 0.0630 - val_acc: 0.9789
Epoch 10/30
60000/60000 [==============================] - 42s 696us/step - loss: 0.0678 - acc: 0.9802 - val_loss: 0.0616 - val_acc: 0.9795
Epoch 11/30
60000/60000 [==============================] - 42s 708us/step - loss: 0.0668 - acc: 0.9804 - val_loss: 0.0615 - val_acc: 0.9794
Epoch 12/30
60000/60000 [==============================] - 46s 773us/step - loss: 0.0660 - acc: 0.9804 - val_loss: 0.0597 - val_acc: 0.9798
Epoch 13/30
60000/60000 [==============================] - 44s 741us/step - loss: 0.0649 - acc: 0.9808 - val_loss: 0.0590 - val_acc: 0.9808
Epoch 14/30
60000/60000 [==============================] - 43s 718us/step - loss: 0.0641 - acc: 0.9814 - val_loss: 0.0597 - val_acc: 0.9805
Epoch 15/30
60000/60000 [==============================] - 42s 697us/step - loss: 0.0634 - acc: 0.9814 - val_loss: 0.0580 - val_acc: 0.9810
Epoch 16/30
60000/60000 [==============================] - 48s 802us/step - loss: 0.0627 - acc: 0.9817 - val_loss: 0.0572 - val_acc: 0.9810
Epoch 17/30
60000/60000 [==============================] - 48s 793us/step - loss: 0.0617 - acc: 0.9820 - val_loss: 0.0574 - val_acc: 0.9802
Epoch 18/30
60000/60000 [==============================] - 48s 804us/step - loss: 0.0610 - acc: 0.9822 - val_loss: 0.0560 - val_acc: 0.9814
0s - loss: 0.0611 - acc: 0
Epoch 19/30
60000/60000 [==============================] - 48s 796us/step - loss: 0.0602 - acc: 0.9822 - val_loss: 0.0565 - val_acc: 0.9804
Epoch 20/30
60000/60000 [==============================] - 54s 904us/step - loss: 0.0595 - acc: 0.9828 - val_loss: 0.0548 - val_acc: 0.9827
Epoch 21/30
60000/60000 [==============================] - 51s 847us/step - loss: 0.0589 - acc: 0.9828 - val_loss: 0.0552 - val_acc: 0.9807
Epoch 22/30
60000/60000 [==============================] - 48s 798us/step - loss: 0.0585 - acc: 0.9830 - val_loss: 0.0541 - val_acc: 0.9819
Epoch 23/30
60000/60000 [==============================] - 46s 771us/step - loss: 0.0574 - acc: 0.9832 - val_loss: 0.0535 - val_acc: 0.9820
Epoch 24/30
60000/60000 [==============================] - 46s 759us/step - loss: 0.0567 - acc: 0.9839 - val_loss: 0.0529 - val_acc: 0.9822
Epoch 25/30
60000/60000 [==============================] - 43s 721us/step - loss: 0.0560 - acc: 0.9834 - val_loss: 0.0536 - val_acc: 0.9815
Epoch 26/30
60000/60000 [==============================] - 44s 734us/step - loss: 0.0557 - acc: 0.9838 - val_loss: 0.0522 - val_acc: 0.9826
Epoch 27/30
60000/60000 [==============================] - 43s 721us/step - loss: 0.0551 - acc: 0.9841 - val_loss: 0.0522 - val_acc: 0.9813
Epoch 28/30
60000/60000 [==============================] - 44s 735us/step - loss: 0.0544 - acc: 0.9841 - val_loss: 0.0521 - val_acc: 0.9815
Epoch 29/30
60000/60000 [==============================] - 44s 739us/step - loss: 0.0537 - acc: 0.9842 - val_loss: 0.0536 - val_acc: 0.9828
Epoch 30/30
60000/60000 [==============================] - 44s 741us/step - loss: 0.0532 - acc: 0.9845 - val_loss: 0.0508 - val_acc: 0.9831
```

Accuracy on the training set: 98.45

Accuracy on the test set: 98.31

Setting 10: Dropout and batch normalization: Use the best settings from above with dropout layers and batch normalization

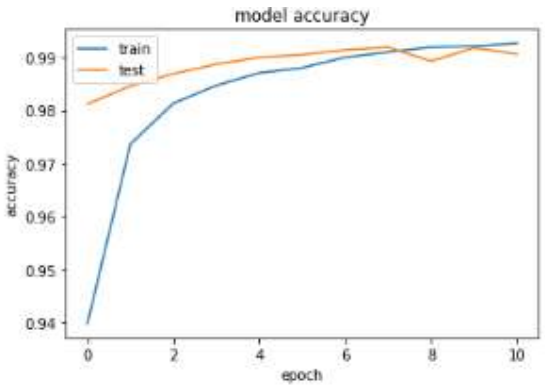
Properties:

	Filters	Kernel Size	Stride	Activation
Conv2D	18	(3,3)	1	relu
MaxPool	N/A	(2,2)	-	-
Conv2D	25	(3,3)	1	relu
MaxPool	N/A	(2,2)	-	-
Flatten	output = 400	-	-	-
Dense Layer 1	units = 120	-	-	-
Dense Layer 2	units = 84	-	-	-
Dense Layer 3	units = 10	-	-	-

Optimizer: RMS prop with default learning rate (0.001)
Loss: Categorical Cross Entropy
Epochs: 4
Batch- size: None

```
Total params: 123,561
Trainable params: 123,475
Non-trainable params: 86
```

Epoch – Accuracy Curve:



Here we can see the effect of batch normalization and dropout come into play as there is no sign of overfitting in the curve. Also, the model is performing exceptionally well with the best accuracy till date.

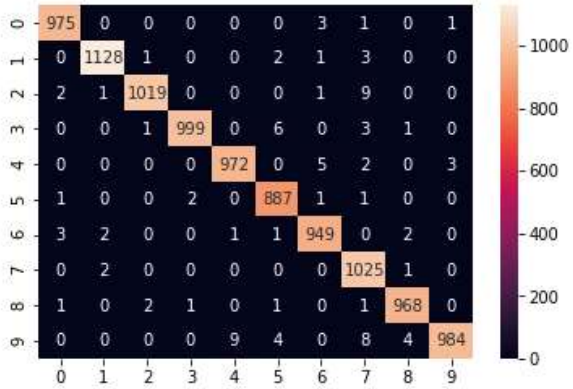
Accuracy:

```
Epoch 3/20
60000/60000 [==============================] - 230s 4ms/step - loss: 0.0660 - mean_absolute_error: 0.0057 - acc: 0.9814 - val_l
oss: 0.0437 - val_mean_absolute_error: 0.0036 - val_acc: 0.9869
Epoch 4/20
60000/60000 [==============================] - 233s 4ms/step - loss: 0.0536 - mean_absolute_error: 0.0047 - acc: 0.9847 - val_l
oss: 0.0364 - val_mean_absolute_error: 0.0032 - val_acc: 0.9887
Epoch 5/20
60000/60000 [==============================] - 229s 4ms/step - loss: 0.0440 - mean_absolute_error: 0.0038 - acc: 0.9871 - val_l
oss: 0.0348 - val_mean_absolute_error: 0.0027 - val_acc: 0.9900
Epoch 6/20
60000/60000 [==============================] - 215s 4ms/step - loss: 0.0409 - mean_absolute_error: 0.0035 - acc: 0.9880 - val_l
oss: 0.0307 - val_mean_absolute_error: 0.0026 - val_acc: 0.9905
Epoch 7/20
60000/60000 [==============================] - 213s 4ms/step - loss: 0.0344 - mean_absolute_error: 0.0030 - acc: 0.9900 - val_l
oss: 0.0284 - val_mean_absolute_error: 0.0023 - val_acc: 0.9914
Epoch 8/20
60000/60000 [==============================] - 212s 4ms/step - loss: 0.0307 - mean_absolute_error: 0.0027 - acc: 0.9910 - val_l
oss: 0.0309 - val_mean_absolute_error: 0.0023 - val_acc: 0.9919
Epoch 9/20
60000/60000 [==============================] - 2258s 38ms/step - loss: 0.0286 - mean_absolute_error: 0.0024 - acc: 0.9919 - val
_loss: 0.0389 - val_mean_absolute_error: 0.0026 - val_acc: 0.9893
Epoch 10/20
60000/60000 [==============================] - 202s 3ms/step - loss: 0.0263 - mean_absolute_error: 0.0022 - acc: 0.9921 - val_l
oss: 0.0343 - val_mean_absolute_error: 0.0021 - val_acc: 0.9918
Epoch 11/20
60000/60000 [==============================] - 195s 3ms/step - loss: 0.0250 - mean_absolute_error: 0.0021 - acc: 0.9927 - val_l
oss: 0.0323 - val_mean_absolute_error: 0.0022 - val_acc: 0.9906
Epoch 00011: early stopping
```

Training accuracy: Early stopping accuracy – 99.1 Stopped model accuracy – 99.27

Test accuracy: Early stopping accuracy –99.19 Stopped model accuracy – 99.06

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	982
1	0.99	1.00	0.99	1133
2	0.99	1.00	0.99	1023
3	0.99	1.00	0.99	1002
4	0.99	0.99	0.99	982
5	0.99	0.98	0.99	901
6	0.99	0.99	0.99	960
7	1.00	0.97	0.99	1053
8	0.99	0.99	0.99	976
9	0.98	1.00	0.99	988
avg / total	0.99	0.99	0.99	10000

Royston Marian Mascarenhas, Spring 19, USC

Summary of settings:

	Training accuracy	*Test accuracy*
Setting 1 : Original LeNet Setting	99.05	98.94
Setting 2: Lesser filters in first CONV layer, more filters in second CONV layer	95.91	98.27
Setting 3: More filters in first CONV layer, lesser filters in second CONV layer	98.9	98.89
Setting 4: More filters in both CONV layers	99.19	99.13
Setting 5: Decrease kernel size	99.3	99.18
Setting 6: Increase dense units	99.45	98.86
Setting 7: Decrease pool size	97.27	97.31
Setting 8: Learning rate	94.75	95.2
Setting 9: Number of epochs	98.45	98.31
Setting 10: Batch Normalization and Dropout	99.2	99.19

Chosen 5 representative settings:

	Training accuracy	*Test accuracy*
Setting 4: More filters in both CONV layers	99.19	99.13
Setting 5: Decrease kernel size with above setting	99.3	99.18
Setting 8: Learning rate	94.75	95.2
Setting 9: Number of epochs	98.45	98.31
Setting 10: Batch Normalization and Dropout	99.2	99.19

Mean of 5 training accuracies: 98.15799999999999
Mean of 5 testing accuracies: 98.202
Variance of training accuracies: 2.991255999999996
Variance of test accuracies: 2.363495999999997

Representative five settings:

Setting 1 : Original LeNet Setting
Part 1: Setting 2, 3, 4: Filter size
Part 2: Setting 5: kernel size
Setting 6: dense units
Setting 7: pool size
Part 3: Setting 8: Learning rate
Part 4: Setting 9: Number of epochs
Part 5: Setting 10: Batch Normalization and Dropout

Best model

Setting 10 (Best settings with batch normalization and dropout)

Training accuracy: 99.2

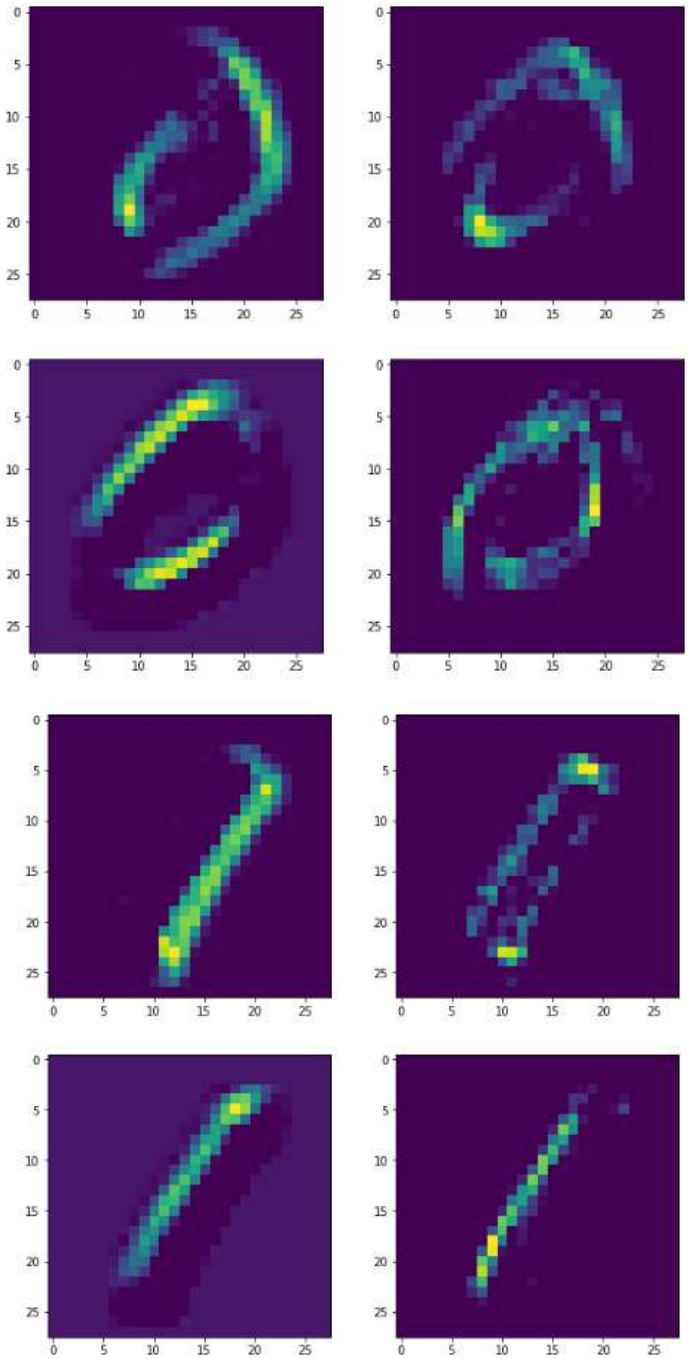
Test accuracy: 99.19

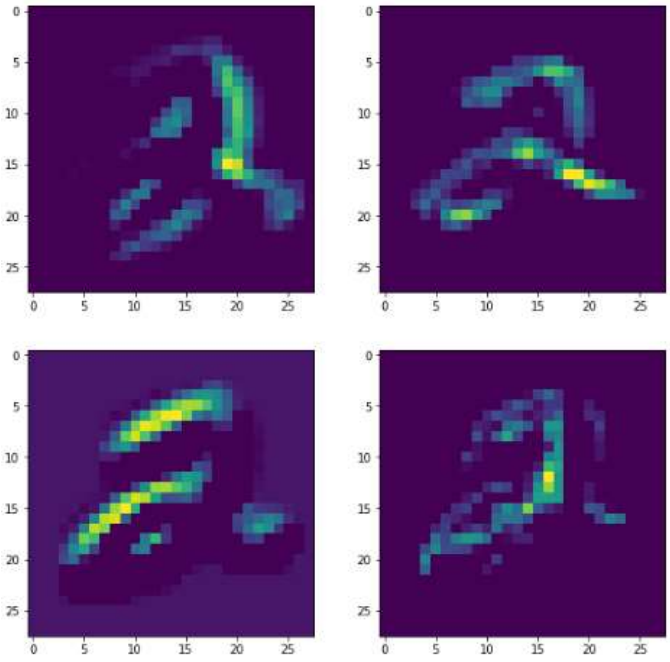
4. Discussion:

How numbers 0, 1 and 2 look after each layer in each setting (post layer feature maps)

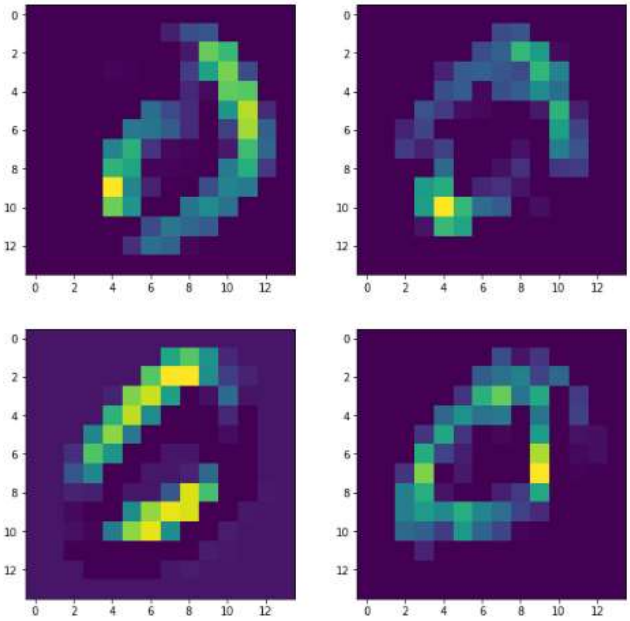
Setting 1 : Original LeNet Setting

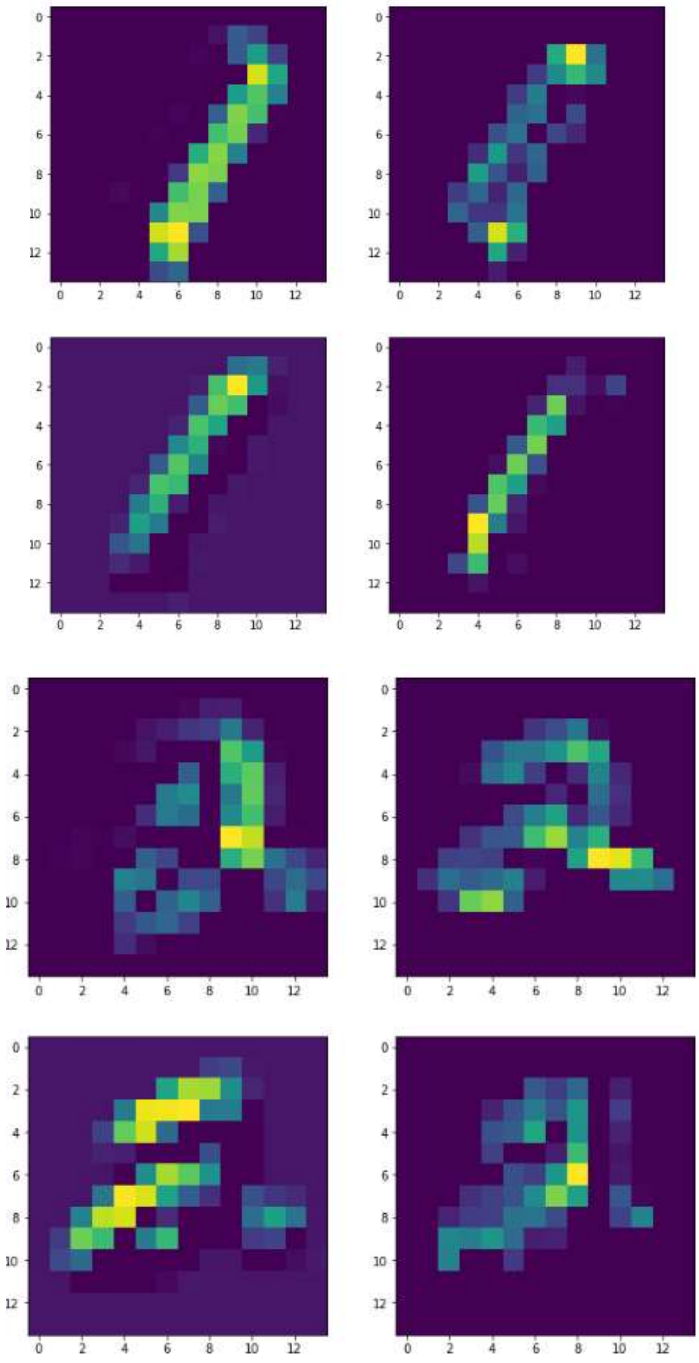
Layer 1



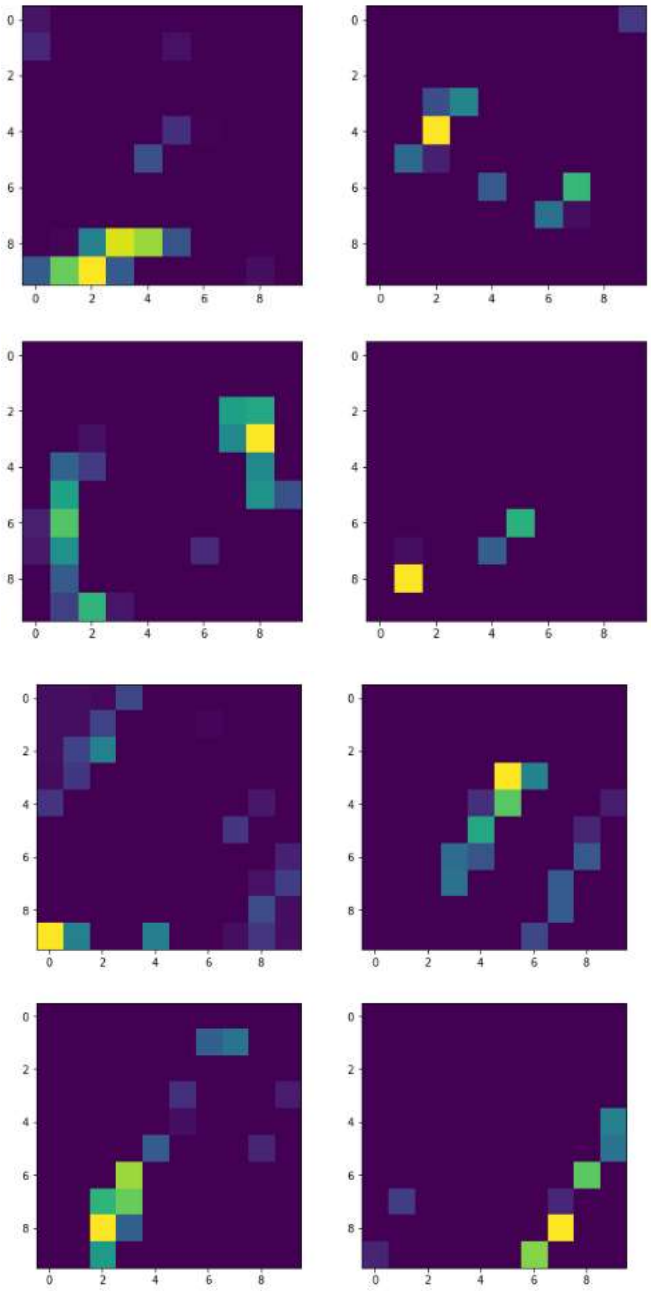


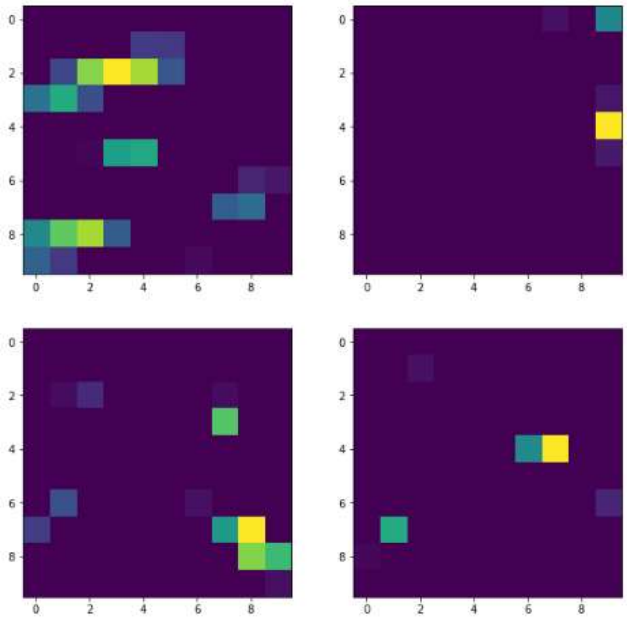
Layer 2



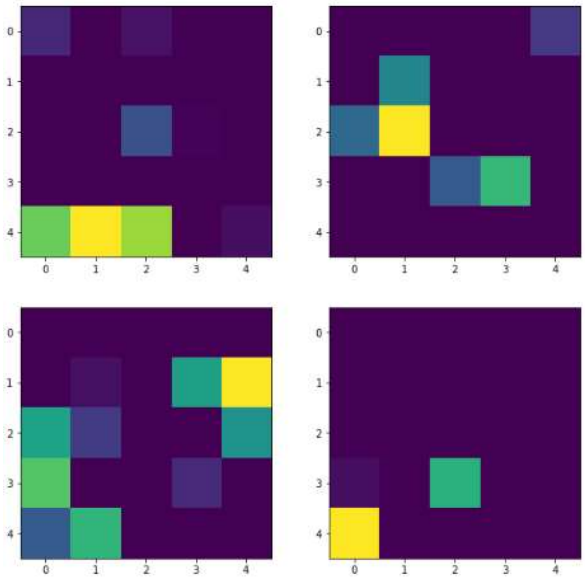


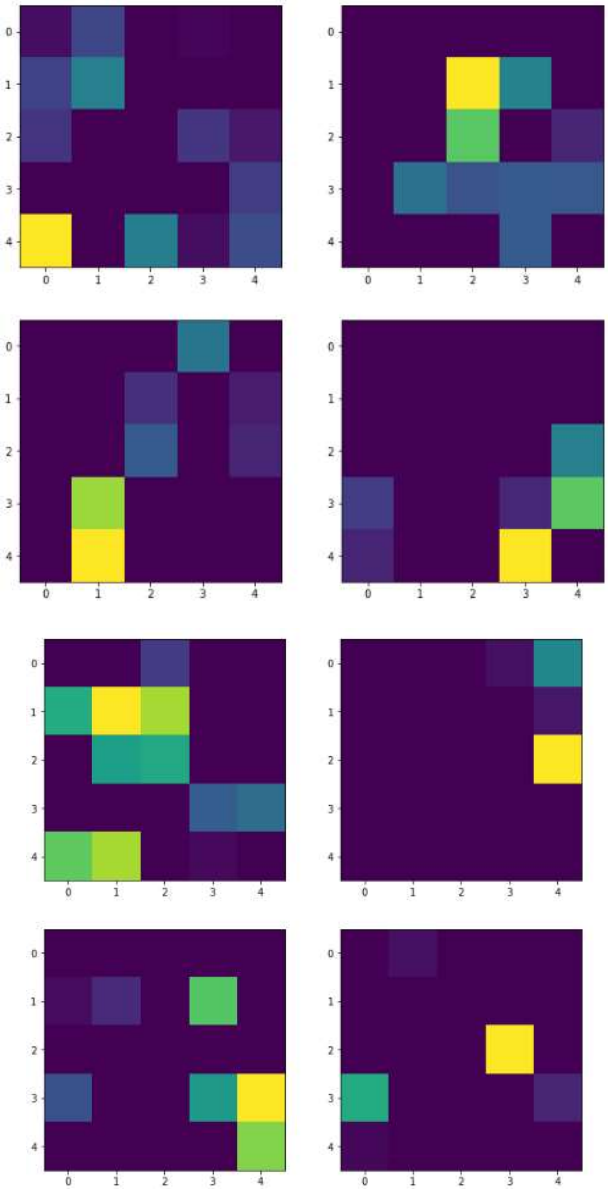
Layer 3





Layer 4



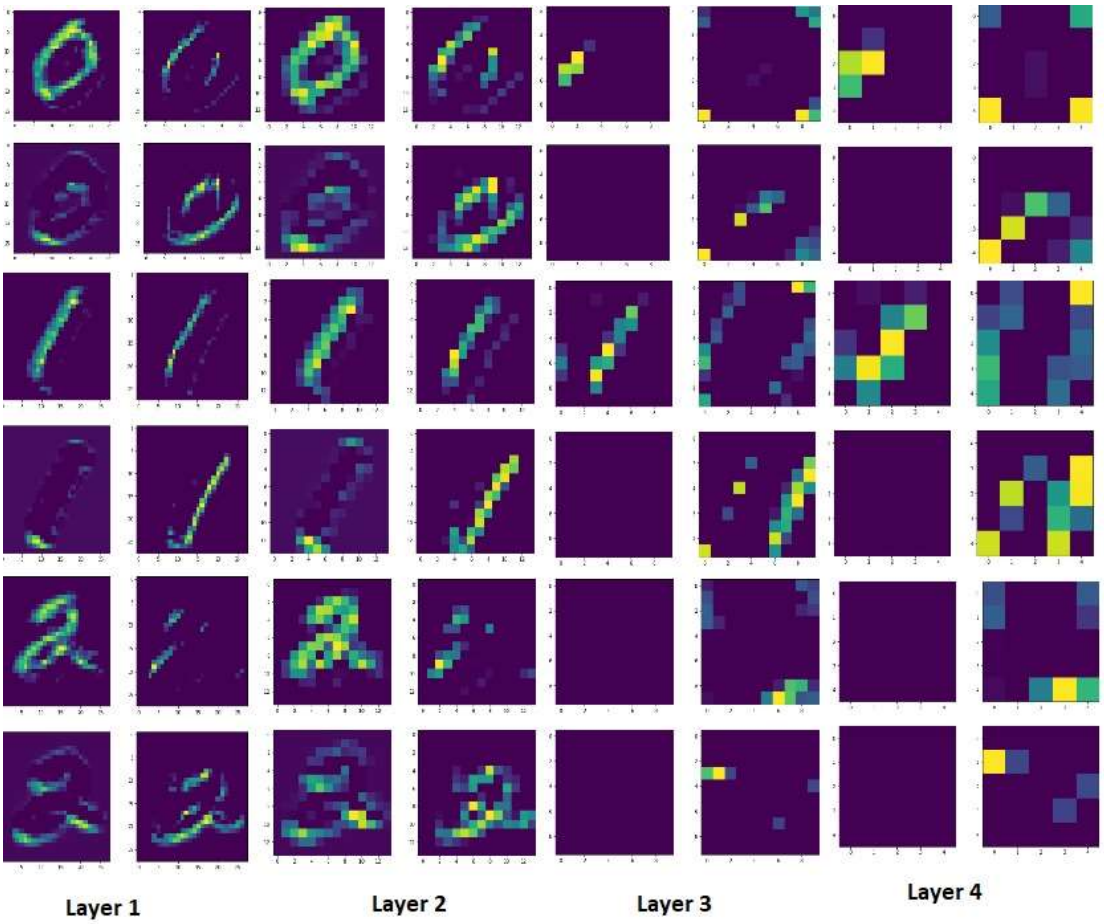


Note that the post layer feature maps have been produced in color for the first two settings to demonstrate the variety of the feature maps. In the settings that follow, the usual grayscale feature maps have been produced.

Discriminating power (aesthetic): moderate

Any effect observed: None apart from the original LeNet effect (this sets the standard for the visualizations that follow)

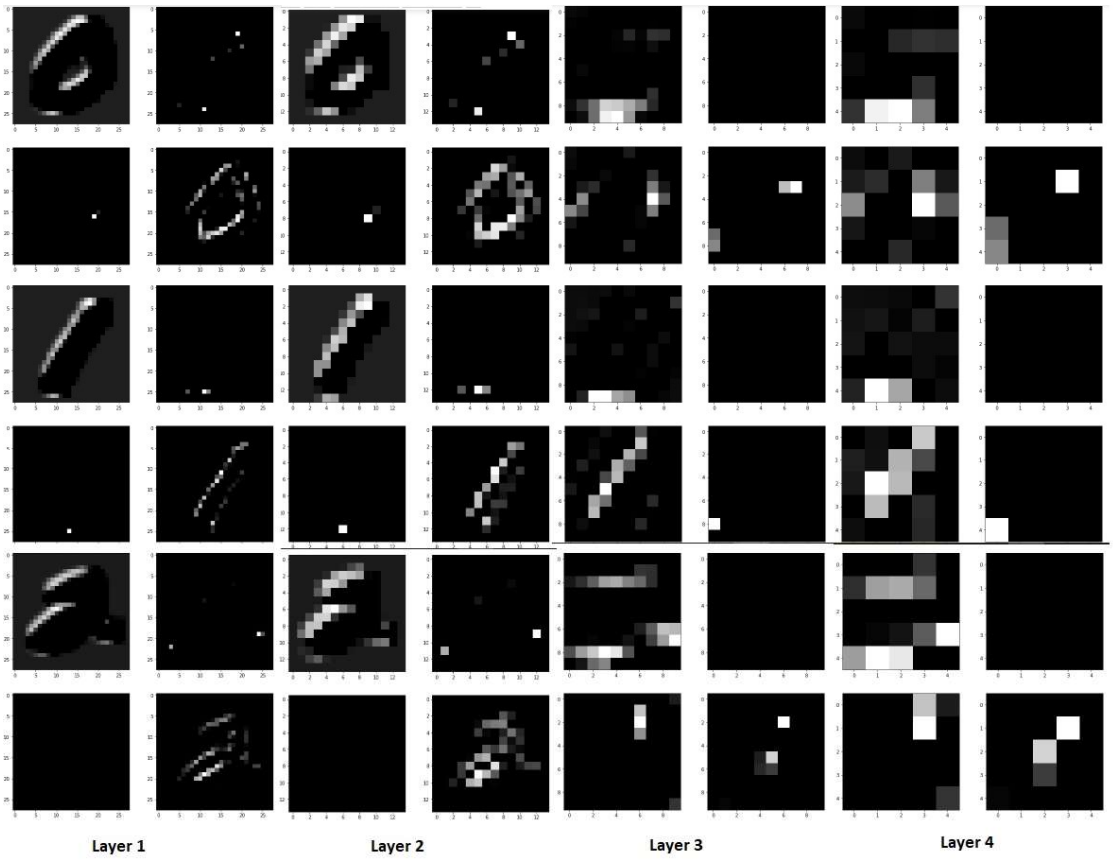
Setting 2: Lesser filters in first CONV layer, more filters in second CONV layer



Discriminating power (aesthetic): not good

Any effect observed: Lesser filters in the first CONV layer has resulted in lesser general features being captured. More filters later have resulted in more specific features being captured.

Setting 3: More filters in first CONV layer, lesser filters in second CONV layer

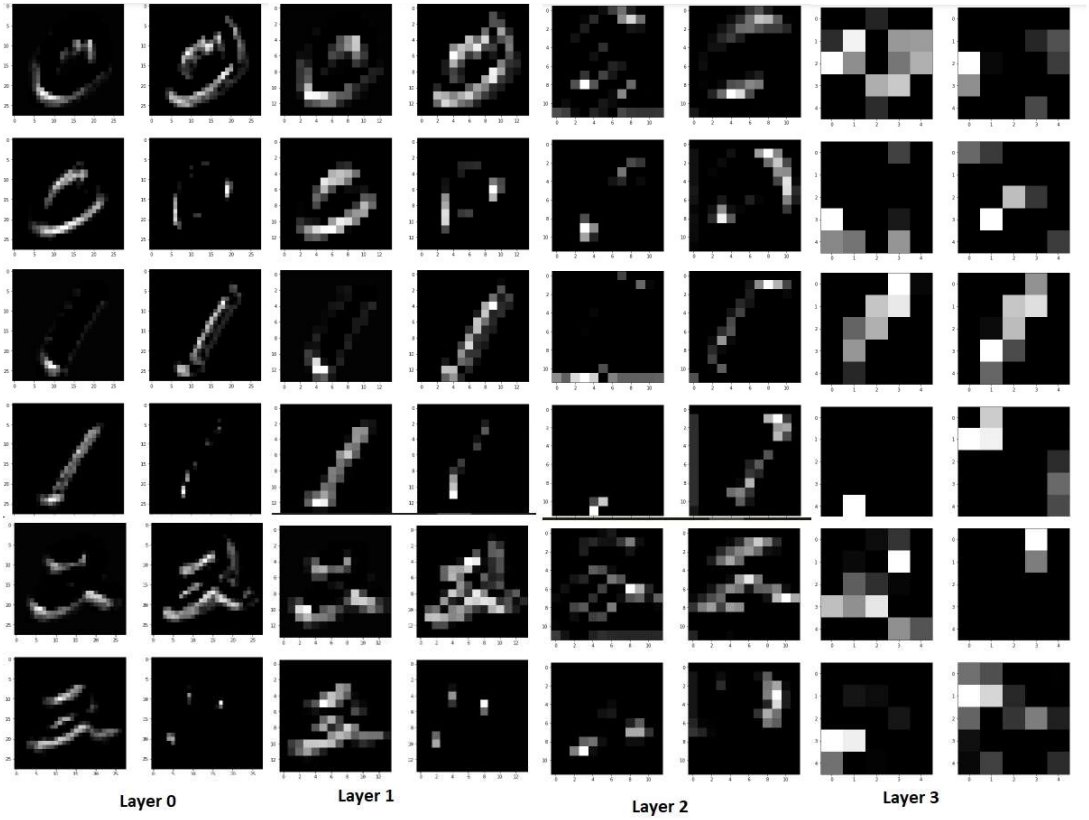


Discriminating power (aesthetic): not good

Any effect observed: More filters initially have resulted in more general features being captured. However, as we can see in layer 3 CONV – some blocks are empty due to lack of deeper layer filters to capture specific features.

Royston Marian Mascarenhas, Spring 19, USC

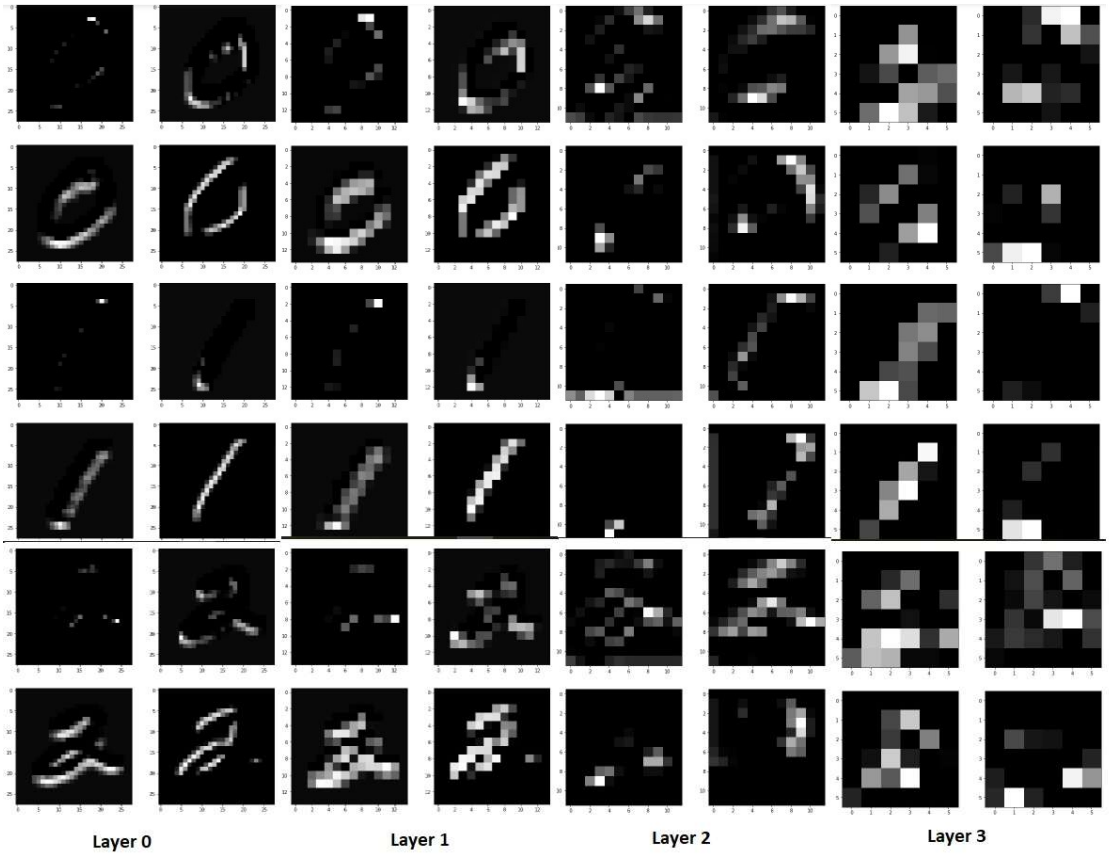
Setting 4: More filters in both CONV layers



Discriminating power (aesthetic): good

Any effect observed: more features captured gives higher accuracy due to more distinguishability

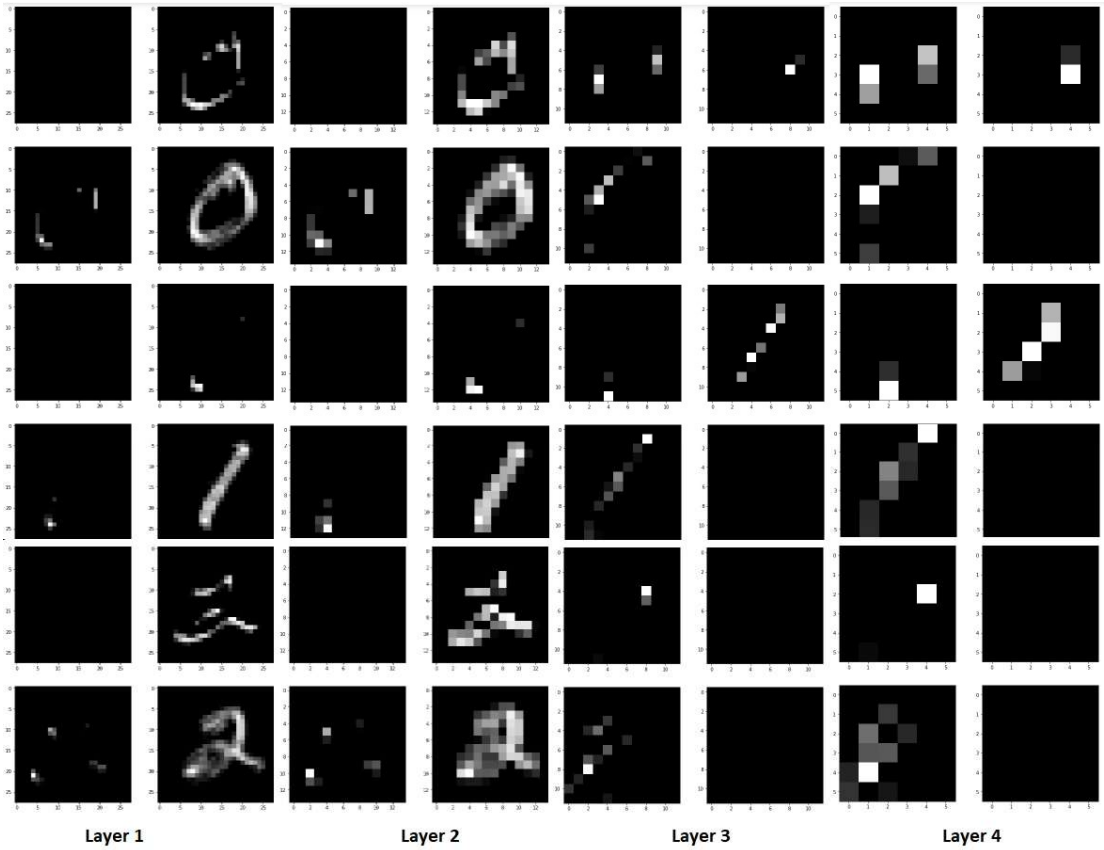
Setting 5: Decrease kernel size



Discriminating power (aesthetic): good

Any effect observed: lower kernel size has resulted in better approximation of pixel energies

Setting 6: Increase dense units

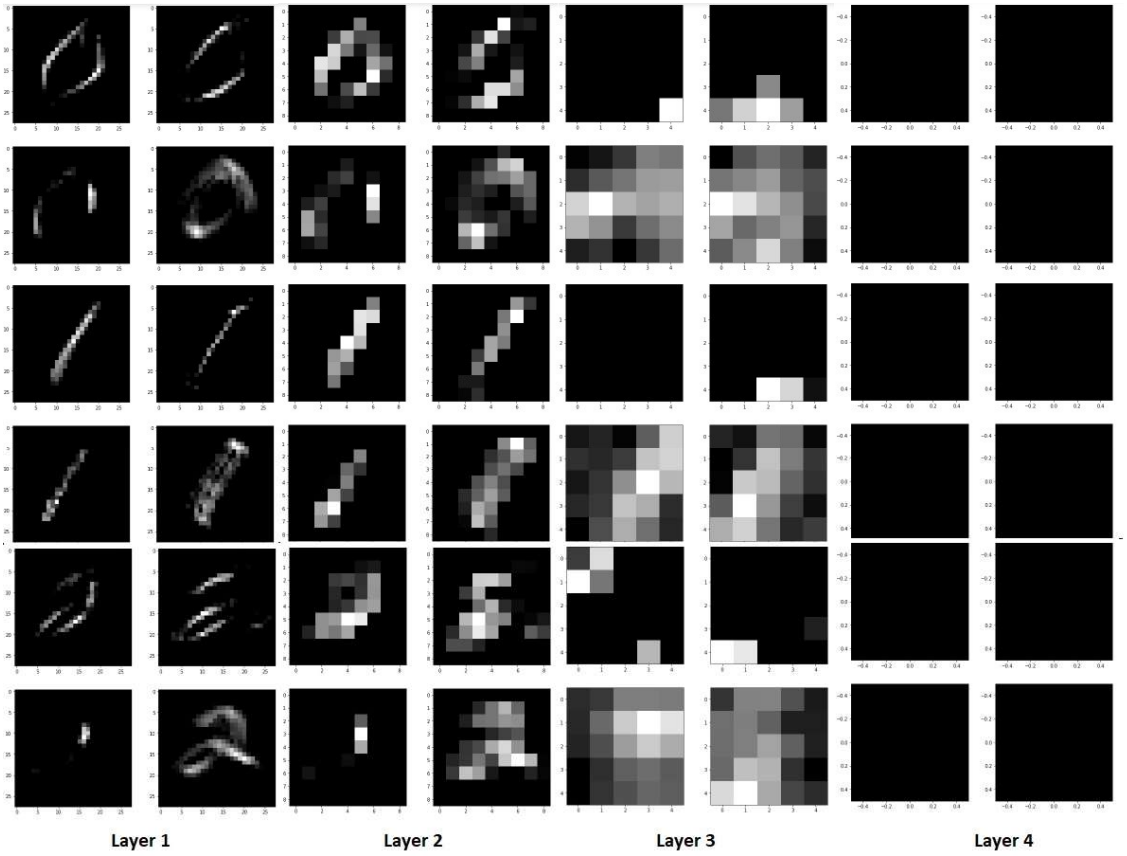


Discriminating power (aesthetic): moderate

Any effect observed: None apart from the original LeNet effect (this sets the standard for the visualizations that follow)

Royston Marian Mascarenhas, Spring 19, USC

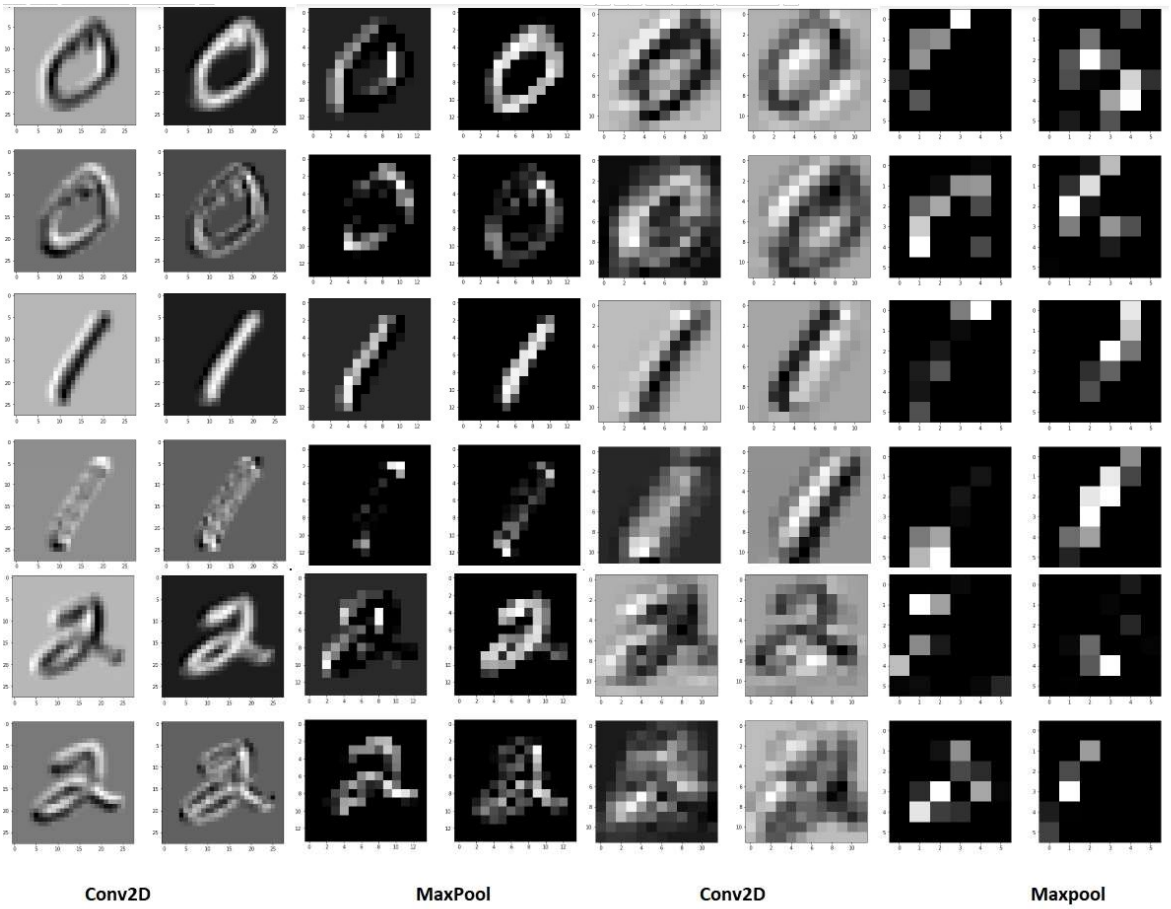
Setting 7: Decrease pool size



Discriminating power (aesthetic): bad

Any effect observed: the last max pool layer output has only one pixel that is either white or black which affects the accuracy of the network. Leakage of information is heavy

Setting 10: Batch Normalization and Dropout



Discriminating power (aesthetic): very good

Any effect observed: best distinguishability due to batch normalization, reduced overfitting due to dropout, best performance due to best settings employed.

More discussion based on experiments:

- **Batch size:**
  Batch size does not affect the accuracy as much as the other factors. Batch size determines the number of samples that are considered in one forward and backward pass. Batch size determines the number of iterations needed to complete the epoch. A higher batch size would need a higher RAM in cases of large datasets since the system should have the computational resources to process multiple images in parallel. The larger the batch size, the quicker one epoch completes but quality degrades as the batch size increases beyond a certain point.

- **Number of filters:**

Each CONV layer will have a user defined number of filters to extract features. Each filter is initialized randomly from different distributions to stop two filters from detecting the same features. The following settings explored this parameter:

- **Setting 2:**
  Here, we used less number of filters in the first layer and more number of filters in the second CONV layer. The initial CONV layers are used to learn abstract features. As we dive deeper into the network, the features will discern more specific features such as the eye, the nose and deeper would mean the whiskers of a cat. More number of filters in the first layer made the network learn more general features. It is now easy to make out that it is a number in the picture but what type of number is determined in the second CONV layer. Since lesser filters were included here, it gets harder for the network to differentiate between the numbers. Since the images were just numbers and the image size was only 28 x 28, this network did not suffer a huge blow in terms of accuracy but the decrease in accuracy can be seen as compared to the original LeNet setting.

- **Setting 3:**
  Here, we used more number of filters in the second CONV layer and lesser filters in the first CONV layer. While specific features are learned better, the general features do not provide much basis for the second CONV layer filters to learn from. Therefore, this model will also undergo a dip in accuracy as can be observed.

- **Setting 4:**
  So what happens if we increase the number of filters in both the CONV layers? More general features are learned and each filter in the second CONV layer will have as input each of the 18 filter feature maps in the first layer. This ensures that the feature maps in the network are quite thorough. As can be observed, the accuracy has improved to 99.13%.

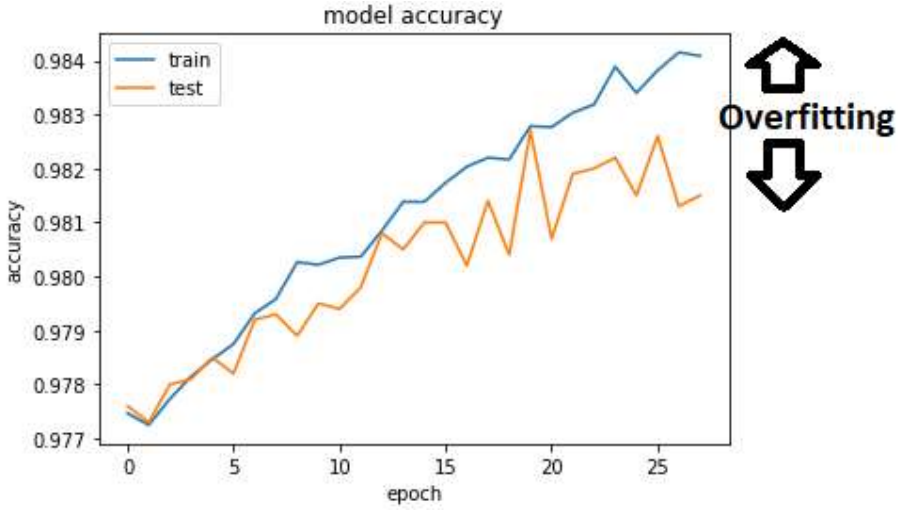
- **Kernel size:**
  The kernel size corresponds to the size of the filter that is going to propagate/convolve through/alongside the image.

  - **Setting 5:**
    A smaller kernel size means the magnitude of the learned features in each entry of the filters are more accurate. However, a smaller kernel size also takes more time to run. An optimum kernel size is preferable if the required accuracy is achieved. A higher kernel size would result in loss of information and a faster training time.

- **Overfitting and epoch number:**

  - **Setting 9:**

    Overfitting and number of epochs are closely related. Although overfitting can be due to other factors, if the model is trained beyond a certain number of epochs, the training accuracy arcs upwards and the testing accuracy drops lower. This is because the model gets generalized over the training dataset. How fast the model generalized is a product of many factors including the size and variety of the dataset, the learning rate etc.



- **Early stopping:**

  To prevent overfitting, regularization techniques are employed and one of those techniques are early stopping. In this case, patience is taken to be 3 (the number of epochs over which a drop in validation accuracy can be accepted before the model stops). This also removes the uncertainty of choosing the *right* number of epochs to run over. Instead, an upper limit is taken (in this case 15) and the model is run till validation accuracy is best and a few epochs after. The model with the best accuracy is saved and loaded later for testing.

- **Stride:**

  The stride determines how many pixels the center of the pixel filter shifts over. A smaller stride parameter realized slower training and more accurate convolved entries. A larger stride may not significantly hurt the accuracy but it will reduce the time it takes to train the network. A stride larger than 3 will affect the accuracy visibly, especially for smaller images.

- **Max Pool Kernel**

  Max pool kernel refers to the square clump of pixels considered to be reduced to one pixel, the one pixel being the highest magnitude pixel in that square clump.

  - **Setting 7:**

A larger max pooling window means that more information is going to be lost. At the end of the day, pooling is just a downsampling operation and only entries with the highest energy are chosen. Therefore, max pooling window is kept as small as possible. i.e 2x2 unless of course, the image size is really big or the domain that the images are from are majorly consistent across their area. The main purpose of max pooling is to reduce computations while preserving information. If a higher kernel size is chosen, the time taken for execution will reduce drastically.

- **Dense units:**

  - **Setting 6:**

    Dense layer connections are used so that we can eventually arrive at the output probabilities of the ten classes. Increasing the number of dense unit connections causes the model to overfit and very few dense unit connections causes loss of accuracy. Therefore, in LeNet, on a trial and error basis, the number of dense layer connections are strategically chosen as 120 for the first dense layer and 84 for the second dense layer. Indeed, when it comes to choosing best settings options, these produce the best results.

- **Number of training variables:**

The number of training variables are calculated as follows:
For a conv layer, if the input is

1. an image in grayscale, input connections= 1
2. an image with RGB, input connections = 3
3. dense layer, input connection = number of dense units
4. conv layer, input connections = number of filters

The output connections of a conv layer is always the number of filters in that layer multiplied by the size of the size of each filter.

For a dense layer, the input is the number of connections is the number of connections from the previous layer. The output is the number of connections in that layer. If the previous layer is a CONV layer, then the **flattened** shape of the previous output is the input to the dense layer.

The number of trainable parameters in each layer are calculated by input x output + bias. The bias for CONV layer is the umber of filter in that layer and the bias for dense layers is the number of connections in that layer. The calculation in case of LeNet would be

CONV layer 1 : (1 x 6x5x5) + 6 = 156
CONV layer 2 : (6 x 16x5x5) + 16 = 2416
Dense layer 1: (400 x 120) + 120 = 48120
Dense layer 2: (120 x 84) + 84 = 10164
Dense layer 3: (84 x 10) + 10 = 850

Adding all the RHS terms, we get 61,702 learnable parameters. The answer is verified with the model summary from Keras.

```
=========================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________
```

In each case, the number of training variables vary. The more the filter size, the more number of training variables and the more the number of dense unit connections, the more the dense layer connections. If the kernel increases, the number of trainable parameters increase. If the max pool increases, the number of trainable parameters decrease as the flattened input for the dense connections decrease and the max pool layer does not have trainable parameters: it is just a downsampling layer.

- **Computational complexity:**

Computational complexity depends on a lot of factors:

1. number of epochs: The primary cause for runtime, the more the number of epochs, the more the runtime.
2. filter size: more the number of filters, more the computations
3. kernel size: lesser kernel size, lesser learning variables
4. max pool kernel size: larger the kernel, lesser computations
5. learning rate: more computations with lesser learning rate
6. training variables: more the number of training variables, more the time to train the network

- **Optimizer:**

Adam optimizer is nothing but rmsprop optimizer with momentum. Thus, we get better accuracies, since past gradients help smooth out the current gradient. It has a pre parameter learning rate that improves performance with sparse gradients. It combines the advantages of Adagad and rmsprop optimizers.

- **Loss:**

Categorical cross entropy is cross entropy functionality with softmax activation. It output a probability for each class. Since this is a multi class problem, categorical cross entropy is an obvious choice.

- **Learning rate:**

  - **Setting 8:**

A very small learning rate infers slow convergence i.e. 0.001. As can be seen in the results, at 12 epochs, the accuracy is still 95%. This means that a few more epochs would be needed for absolute convergence. However, if the learning rate is really

high i.e. 1. then it would mean that the global minimum might be skipped. This might mean that the algorithm might never converge. Therefore, a very high learning rate is strictly non advisable.

- **Choice of activation function:**

  The advantages of ReLu are sparsity and reduced likelihood of vanishing gradient. While in sigmoid, the gradients of sigmoids become exceedingly small as the value of x increases, the constant gradient of ReLu results in faster learning. Sparsity is more for relu in CNNs which make it easier for computations. Therefore, relu is a direct choice.

- **Dropout and batch normalization:**
  With dropout and batch normalization, we are generalizing the model. As a result, we can observe that the initial epochs have comparatively lower accuracy due to number of dense units converted to **non trainable parameters.**

- **Performance report:**
  From the performance report, we can guage the cause of the dip in accuracy. The cause might be false positives or false negatives, noted from the dip observed in precision or recall. The F1 score gives an estimate of how good the model is. The best model has achieved an F1 score of 0.99.

- **Confusion matrix:**
  The misclassified class labels can be identified from the confusion matrix. Thus, an error in annotation or the fault of the network can be localized from the misclassified labels.

- **Scope for improvement:**
  1. Leaky relu can be used instead of relu to accommodate the range for truncated information since relu eliminates negative values which might result in loss of information.
  2. Use more regularization methods

(c)

Apply trained network to negative images

1. **Abstract and Motivation:**
   The LeNet was successful in recognizing the MNIST inputs. But will it be successful as a human is in recognizing the negative image of MNIST? This is to test the rigidity and the adaptability of a CNN. Further on, we test a network which is trained on both types of images and then test it on the same to note the difference.

2. **Approach and Procedures:**

   A two fold approach was used while trying to resolve this problem. Both approaches tackled different ways to train the algorithm.

   The first approach would take the first 60,000 images of MNIST and append it to the 60,000 negative images of MNIST and then train on that 120000 image dataset.

   The second approach would randomly shuffle the images in the 120000 image dataset.

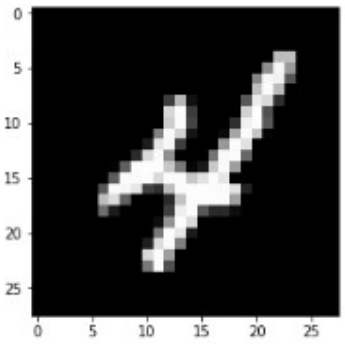
   A new network was proposed to train the network on the negative as well as the original MNIST images. The network parameters are shown below:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 18)	468
batch_normalization_1 (Batch	(None, 28, 28, 18)	72
activation_1 (Activation)	(None, 28, 28, 18)	0
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 18)	0
conv2d_2 (Conv2D)	(None, 10, 10, 25)	11275
batch_normalization_2 (Batch	(None, 10, 10, 25)	100
activation_2 (Activation)	(None, 10, 10, 25)	0
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 25)	0
flatten_1 (Flatten)	(None, 625)	0
dense_1 (Dense)	(None, 120)	75120
dropout_1 (Dropout)	(None, 120)	0
dense_2 (Dense)	(None, 84)	10164
dropout_2 (Dropout)	(None, 84)	0
dense_3 (Dense)	(None, 10)	850

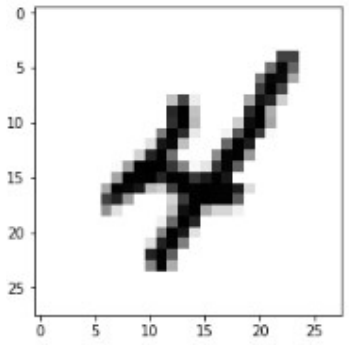
Total params: 98,049
Trainable params: 97,963
Non-trainable params: 86

3. Results:

MNIST



Negative image of MNIST



Accuracy of pre-trained network on negative images:

```
In [21]:  1  model_temp.evaluate(x_test,y_test,batch_size=None)
          10000/10000 [==============================] - 4s 366us/step
Out[21]: [8.862299317932129, 0.15109853370189666, 0.2474]
```

Training accuracy: 98.9%

Testing accuracy: 24.74%

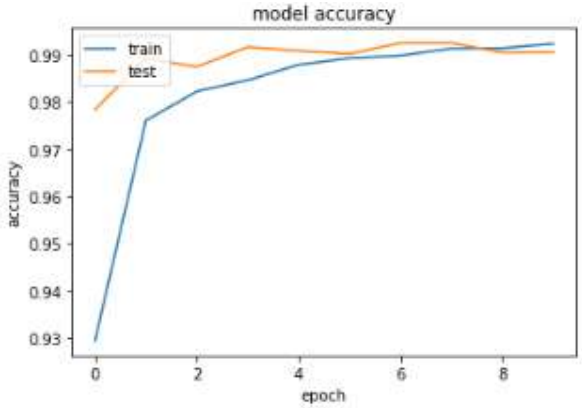
Network trained on MNIST and negative of MNIST:

Approach 1:

Epochs run for : 10
Optimizer: rmsprop
Loss: Categorical Cross Entropy

```
Train on 120000 samples, validate on 20000 samples
Epoch 1/17
120000/120000 [==============================] - 386s 3ms/step - loss: 0.2285 - mean_absolute_error: 0.0207 - acc: 0.9294 - val
_loss: 0.0755 - val_mean_absolute_error: 0.0066 - val_acc: 0.9784
Epoch 2/17
120000/120000 [==============================] - 342s 3ms/step - loss: 0.0843 - mean_absolute_error: 0.0075 - acc: 0.9761 - val
_loss: 0.0380 - val_mean_absolute_error: 0.0035 - val_acc: 0.9892
Epoch 3/17
120000/120000 [==============================] - 403s 3ms/step - loss: 0.0623 - mean_absolute_error: 0.0055 - acc: 0.9823 - val
_loss: 0.0412 - val_mean_absolute_error: 0.0036 - val_acc: 0.9875absolute_error: 0.0055 - acc
Epoch 4/17
120000/120000 [==============================] - 388s 3ms/step - loss: 0.0532 - mean_absolute_error: 0.0047 - acc: 0.9846 - val
_loss: 0.0281 - val_mean_absolute_error: 0.0026 - val_acc: 0.9917
Epoch 5/17
120000/120000 [==============================] - 361s 3ms/step - loss: 0.0432 - mean_absolute_error: 0.0038 - acc: 0.9879 - val
_loss: 0.0300 - val_mean_absolute_error: 0.0024 - val_acc: 0.9909
Epoch 6/17
120000/120000 [==============================] - 377s 3ms/step - loss: 0.0373 - mean_absolute_error: 0.0033 - acc: 0.9893 - val
_loss: 0.0324 - val_mean_absolute_error: 0.0026 - val_acc: 0.9902
Epoch 7/17
120000/120000 [==============================] - 412s 3ms/step - loss: 0.0340 - mean_absolute_error: 0.0030 - acc: 0.9898 - val
_loss: 0.0297 - val_mean_absolute_error: 0.0021 - val_acc: 0.9925
Epoch 8/17
120000/120000 [==============================] - 388s 3ms/step - loss: 0.0300 - mean_absolute_error: 0.0026 - acc: 0.9913 - val
_loss: 0.0260 - val_mean_absolute_error: 0.0020 - val_acc: 0.9925
Epoch 9/17
120000/120000 [==============================] - 382s 3ms/step - loss: 0.0292 - mean_absolute_error: 0.0025 - acc: 0.9914 - val
_loss: 0.0348 - val_mean_absolute_error: 0.0024 - val_acc: 0.9905
Epoch 10/17
120000/120000 [==============================] - 413s 3ms/step - loss: 0.0265 - mean_absolute_error: 0.0023 - acc: 0.9923 - val
_loss: 0.0382 - val_mean_absolute_error: 0.0022 - val_acc: 0.9906
Epoch 00010: early stopping
```

Epoch – Accuracy curve:



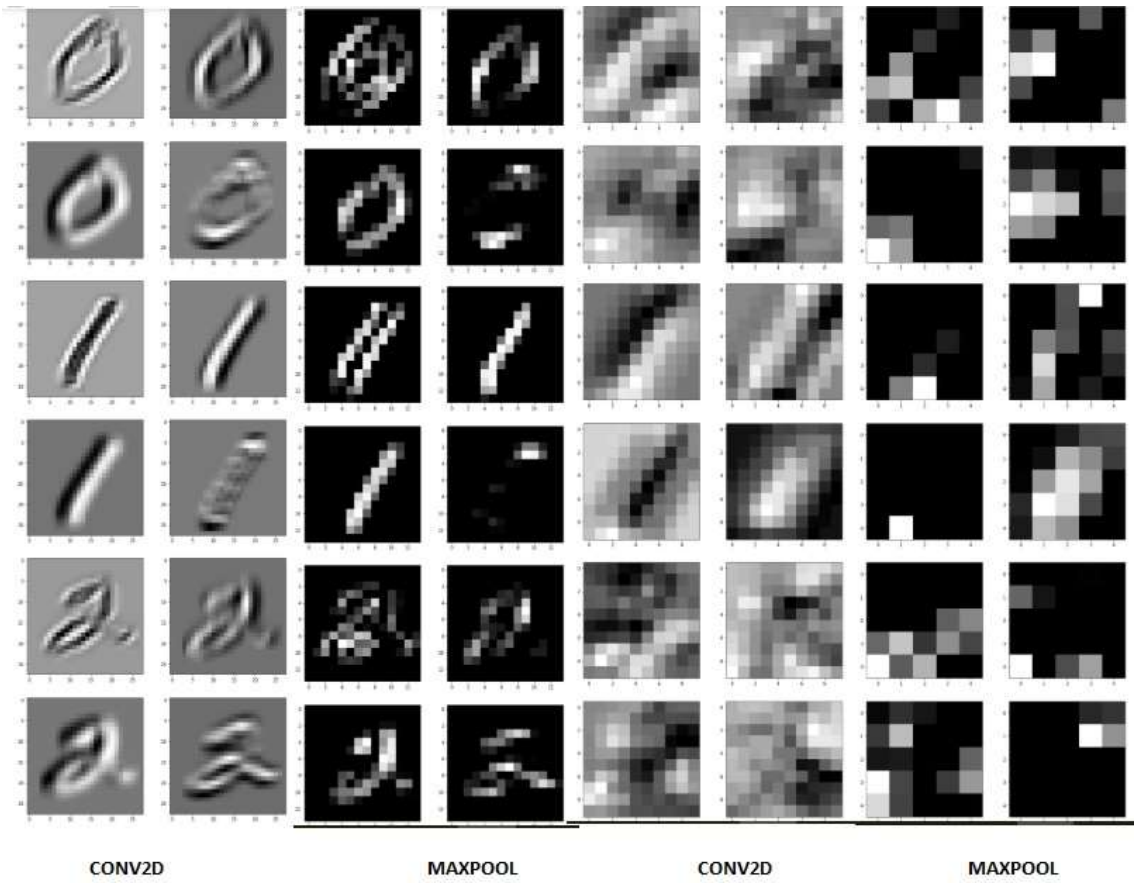
No overfitting is observed because of dropout. Commendable performance, thanks to batch normalization.

Accuracy:

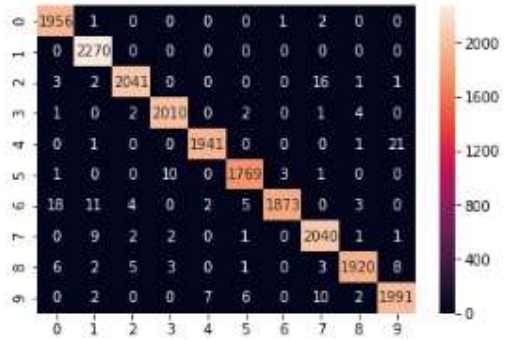
Training accuracy: 99.23

Test accuracy: 99.06

Post layer feature maps:



Confusion matrix:



Classification report:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	1985
1	1.00	0.99	0.99	2298
2	0.99	0.99	0.99	2054
3	1.00	0.99	0.99	2025
4	0.99	1.00	0.99	1950
5	0.99	0.99	0.99	1784
6	0.98	1.00	0.99	1877
7	0.99	0.98	0.99	2073
8	0.99	0.99	0.99	1932
9	0.99	0.98	0.99	2022
avg / total	0.99	0.99	0.99	20000

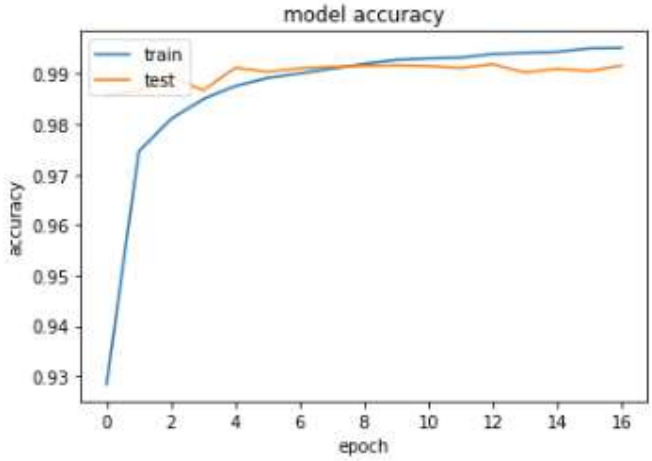
Approach 2:

Epochs run for : 17
Optimizer: rmsprop
Loss: Categorical Cross Entropy

```
Train on 120000 samples, validate on 20000 samples
Epoch 1/17
120000/120000 [==============================] - 368s 3ms/step - loss: 0.2310 - mean_absolute_error: 0.0211 - acc: 0.9284 - val
_loss: 0.0482 - val_mean_absolute_error: 0.0045 - val_acc: 0.9857
Epoch 2/17
120000/120000 [==============================] - 365s 3ms/step - loss: 0.0867 - mean_absolute_error: 0.0077 - acc: 0.9746 - val
_loss: 0.0437 - val_mean_absolute_error: 0.0042 - val_acc: 0.9866
Epoch 3/17
120000/120000 [==============================] - 392s 3ms/step - loss: 0.0644 - mean_absolute_error: 0.0058 - acc: 0.9810 - val
_loss: 0.0329 - val_mean_absolute_error: 0.0031 - val_acc: 0.9895
Epoch 4/17
120000/120000 [==============================] - 421s 4ms/step - loss: 0.0528 - mean_absolute_error: 0.0046 - acc: 0.9850 - val
_loss: 0.0411 - val_mean_absolute_error: 0.0036 - val_acc: 0.9867
Epoch 5/17
120000/120000 [==============================] - 414s 3ms/step - loss: 0.0443 - mean_absolute_error: 0.0039 - acc: 0.9876 - val
_loss: 0.0292 - val_mean_absolute_error: 0.0024 - val_acc: 0.9912
Epoch 6/17
120000/120000 [==============================] - 408s 3ms/step - loss: 0.0380 - mean_absolute_error: 0.0033 - acc: 0.9891 - val
_loss: 0.0327 - val_mean_absolute_error: 0.0026 - val_acc: 0.9904
Epoch 7/17
120000/120000 [==============================] - 420s 4ms/step - loss: 0.0344 - mean_absolute_error: 0.0030 - acc: 0.9901 - val
_loss: 0.0304 - val_mean_absolute_error: 0.0023 - val_acc: 0.9911
Epoch 8/17
120000/120000 [==============================] - 401s 3ms/step - loss: 0.0307 - mean_absolute_error: 0.0027 - acc: 0.9910 - val
_loss: 0.0335 - val_mean_absolute_error: 0.0022 - val_acc: 0.9914
Epoch 9/17
120000/120000 [==============================] - 433s 4ms/step - loss: 0.0280 - mean_absolute_error: 0.0024 - acc: 0.9919 - val
_loss: 0.0285 - val_mean_absolute_error: 0.0021 - val_acc: 0.9916
Epoch 10/17
120000/120000 [==============================] - 400s 3ms/step - loss: 0.0252 - mean_absolute_error: 0.0022 - acc: 0.9927 - val
_loss: 0.0333 - val_mean_absolute_error: 0.0021 - val_acc: 0.9917
Epoch 11/17
120000/120000 [==============================] - 362s 3ms/step - loss: 0.0246 - mean_absolute_error: 0.0021 - acc: 0.9930 - val
_loss: 0.0289 - val_mean_absolute_error: 0.0022 - val_acc: 0.9916
```

```
Epoch 13/17
120000/120000 [==============================] - 369s 3ms/step - loss: 0.0207 - mean_absolute_error: 0.0018 - acc: 0.9939 - val
_loss: 0.0308 - val_mean_absolute_error: 0.0020 - val_acc: 0.9919
Epoch 14/17
120000/120000 [==============================] - 366s 3ms/step - loss: 0.0203 - mean_absolute_error: 0.0017 - acc: 0.9941 - val
_loss: 0.0386 - val_mean_absolute_error: 0.0024 - val_acc: 0.9902
Epoch 15/17
120000/120000 [==============================] - 383s 3ms/step - loss: 0.0188 - mean_absolute_error: 0.0016 - acc: 0.9943 - val
_loss: 0.0348 - val_mean_absolute_error: 0.0021 - val_acc: 0.9910
Epoch 16/17
120000/120000 [==============================] - 360s 3ms/step - loss: 0.0182 - mean_absolute_error: 0.0015 - acc: 0.9951 - val
_loss: 0.0336 - val_mean_absolute_error: 0.0022 - val_acc: 0.9906
Epoch 17/17
120000/120000 [==============================] - 369s 3ms/step - loss: 0.0173 - mean_absolute_error: 0.0015 - acc: 0.9951 - val
_loss: 0.0325 - val_mean_absolute_error: 0.0020 - val_acc: 0.9917
Epoch 00017: early stopping
```

Epoch – Accuracy curve:



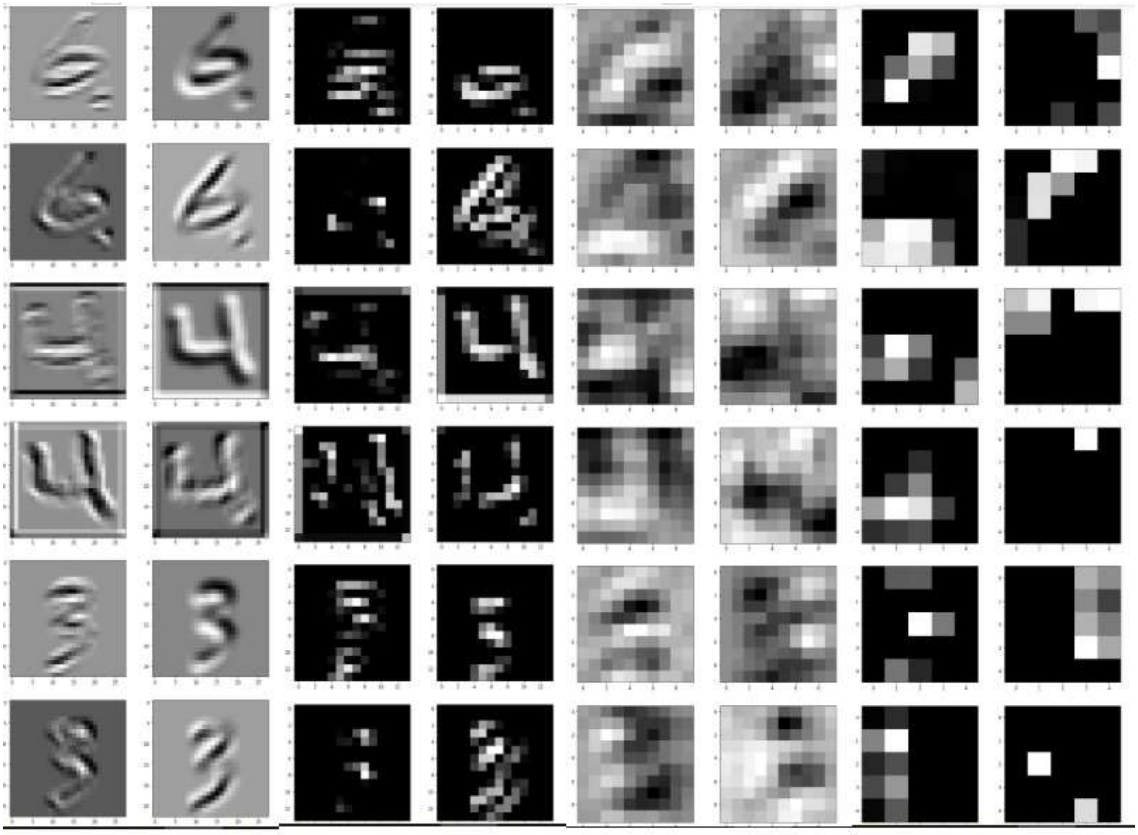
No overfitting is observed because of dropout. Commendable performance, thanks to batch normalization.

Accuracy:

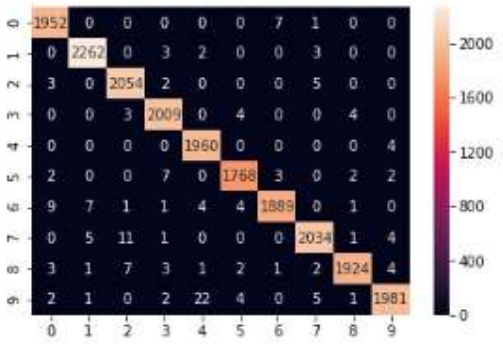
Training accuracy: 99.1

Test accuracy: 99.19

Post layer feature map:



Confusion matrix:



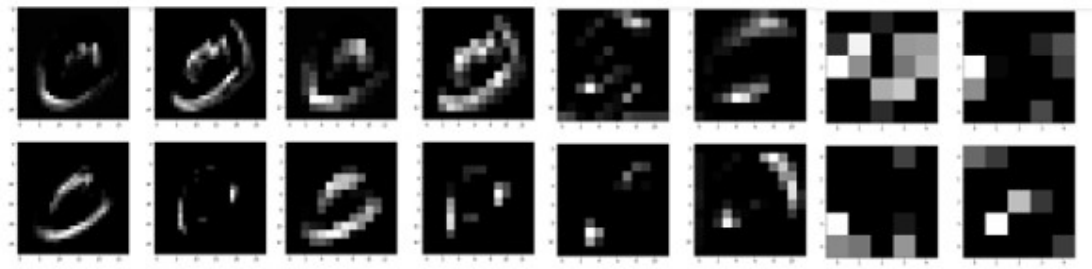
Classification report:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	1971
1	1.00	0.99	1.00	2276
2	1.00	0.99	0.99	2076
3	0.99	0.99	0.99	2028
4	1.00	0.99	0.99	1989
5	0.99	0.99	0.99	1782
6	0.99	0.99	0.99	1900
7	0.99	0.99	0.99	2050
8	0.99	1.00	0.99	1933
9	0.98	0.99	0.99	1995
avg / total	0.99	0.99	0.99	20000

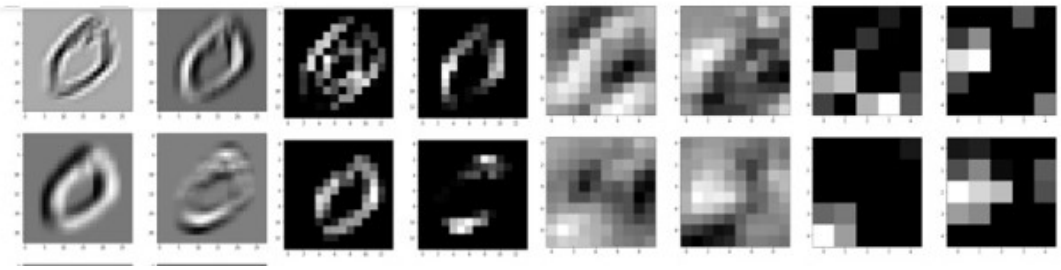
4.Discussion:

Reasons why the negative of the images did not result in good classification with pre trained network:

Feature map of pre trained network (LeNet5) :



Required feature map:



There are several reasons why there was poor accuracy in this case:

1. The datasets are not similar: Even if the network has trained on the MNIST dataset with black background, it does not possess the cognitive ability to differentiate digits on a white background.
2. The feature maps generated for both the cases are different. The second feature map is the required feature map which will produce a good accuracy. The difference between the two feature maps are clear.
3. The weights mathematical equations of different neurons are not tuned to the data which has white background as the object/area of interest and the shape of interest is completely different.

Reasons for proposed network:

1. Reasons for chosen parameters:

   kernel size (5,5)

   - good accuracy
   - less computation time than (3x3) kernel since a 3x3 kernel on a 120000 image dataset would consume a lot of time.

   stride = 1

   - optimal with no loss of information

max pool size = (2,2) with default stride

- standard size
- a higher parameter in both aspects would result in loss of information, especially since this is a small image.

dense layers as in LeNet – 120, 84 and the fixed value 10

- parameters chosen as a result of trial and error
- optimal, as in standard Le-Net

filter size

- increased in both CONV layers to capture more features.
- computation time increases but since the dataset is not as simple as the basic MNIST, I used more features for better recognition.

batch normalization

- used to speed up algorithm, improve performance
- reduce internal covariate shift in weight space

dropout

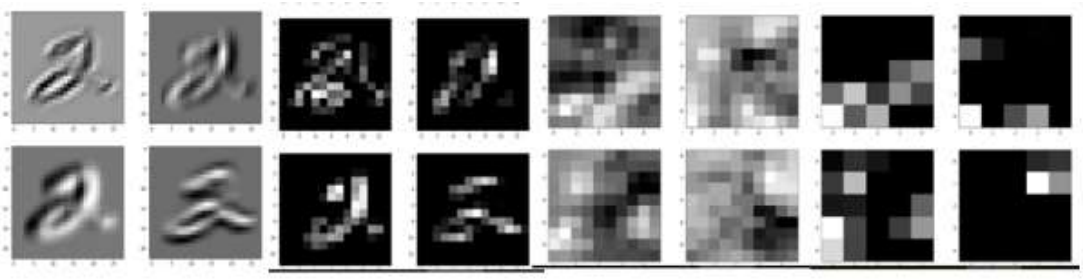
- since the data is not very varied (same shape, different colors) there is a chance of overfitting
- dropout prevents overfitting

2. The best features were picked from part (b) with additional modifications to the network architecture to obtain the best training accuracy.

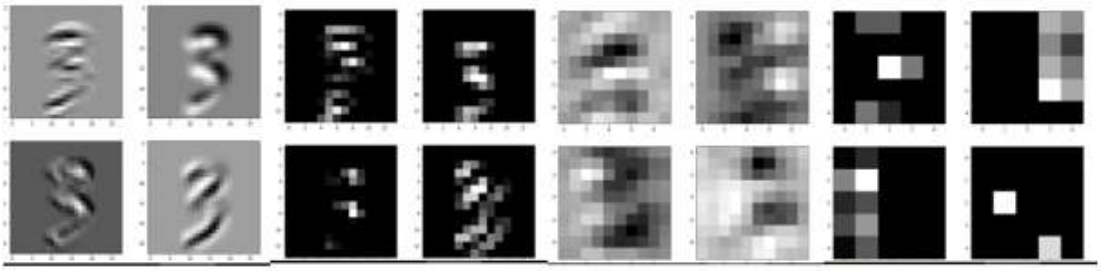
Discussion on which approach was the best:

Post layer maps: From the post layer maps, we can make out that both approaches had approximates the same amount of discrimination between the numbers. The feature maps were strong and had very good discriminating power.

Approach 1:



Approach 2:



Accuracies: The accuracy was also about the same : Above 99%. Not much can be discerned on which approach was better.

Therefore, we conclude that order in which the network was trained does not matter in a CNN as long as the dataset is large enough and the epochs are run long enough to obtain sufficient accuracy.

Note that the second approach did take a longer time to converge.

Also, the **computational complexity** is a factor while training 120000 images. A run time of almost 1 hour was recorded.

References:

[1] http://cs231n.github.io/convolutional-networks/

[2] O'Reilly Media Pictures (only for one image during theoretical explanation)

[3] freecodecamp.com Pictures (only for one image during theoretical explanation)

[4] https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[5] http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf

[6] https://pdfs.semanticscholar.org/5d79/11c93ddcb34cac088d99bd0cae9124e5dcd1.pdf

[7] https://grzegorzgwardys.wordpress.com/2016/04/22/8/

[8] LeCun's paper on LeNet5

[9] Classwork, homework file and discussion pdfs