SMAI Assignment 2

Neural Networks

By

Kaushik L V (20172138)

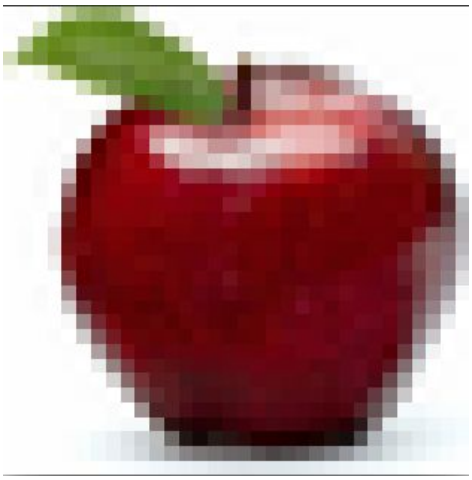
Table of Contents

1. Question 1 -

1.1. Forward pass of Convolutional Neural Network

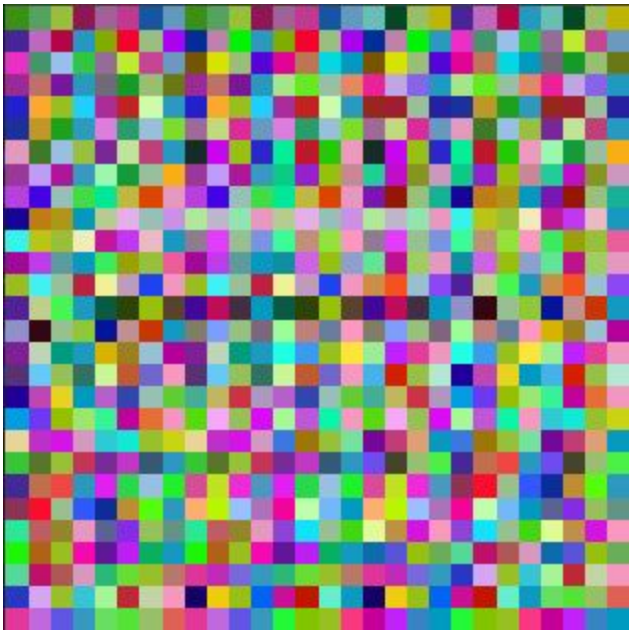
1.1.1. Input Image -

Dimension - 312 x 312 x 3



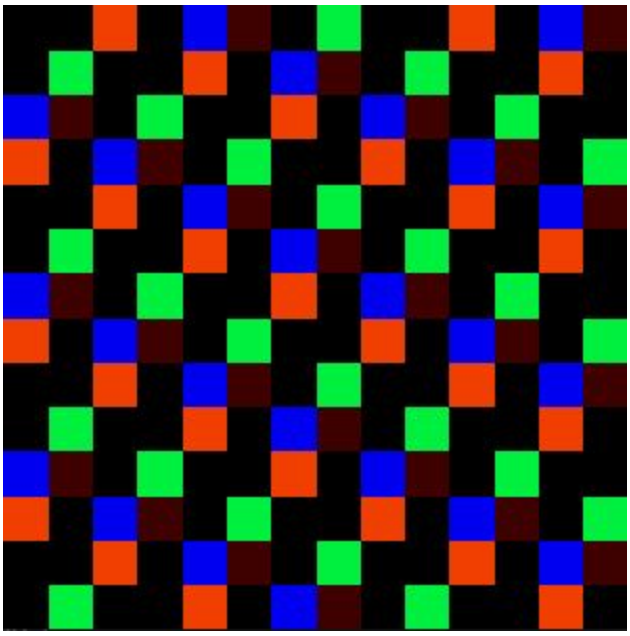
Dimension - 32 x 32 x 3

1.1.2. Output Image After First Convolution -



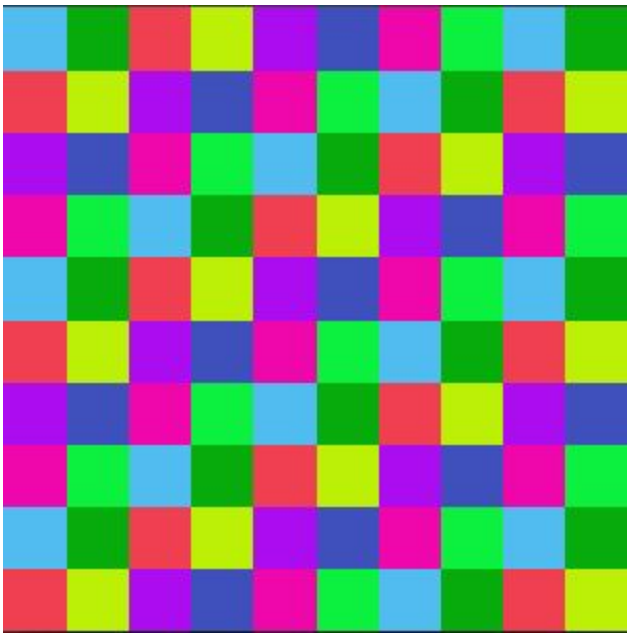
Dimension - 28 x 28 x 6

1.1.3. Output Image After First Max Pooling -



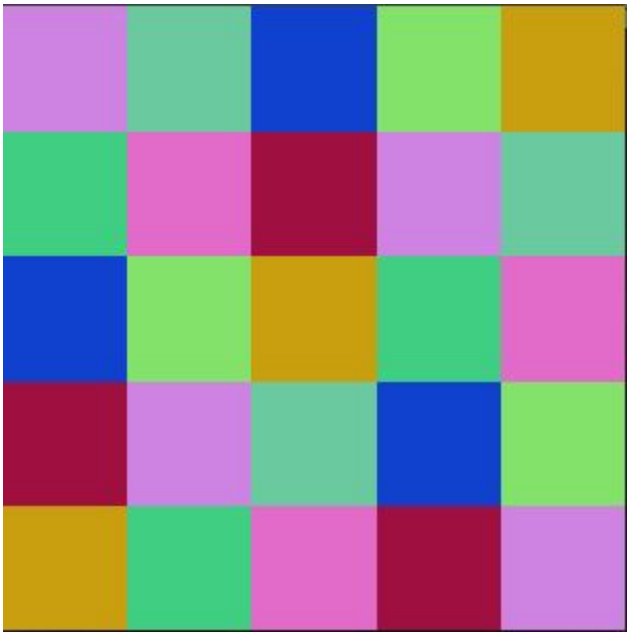
Dimension - 14 x 14 x 6

1.1.4. Output Image After Second Convolution -



Dimension - 10 x 10 x 16

1.1.5. Output Image After Second Max Pooling -



Dimension - 5 x 5 x 16

1.1.6. Code -

```
import numpy as np
import cv2
from PIL import Image


def ReLU(x):
    """
    Compute the relu
    :param x:
    :return:
    """
    return np.where(x > 0, 1.0, 0.0)
```

```
def softMax(x):
    """
    Compute softmax values for each value in x.
    """
    sMax = []
    for i in range(x.shape[0]):
        sMax.append(np.exp(x[i]) / np.sum(np.exp(x)))
    return np.array(sMax)


def convolutionLayer1(image, filters):
    """
    Convolves the image by applying the filter
    :param image:
    :param filters:
    :return: convolutionMatrix
    """
    convolutionMatrix = np.zeros((28, 28, 6))

    for fil in range(6):
        for rowStride in range(image.shape[0] - filters.shape[0] + 1):
            for colStride in range(image.shape[0] - filters.shape[0] +
1):
                convolutionMatrix[rowStride][colStride][fil] = \
                    np.sum(filters * image[rowStride:rowStride + 5,
                                           colStride:colStride + 5, :])

    return convolutionMatrix


def maxPoolLayer1(convolutionMatrix):
    """
    Performs max pooling on the convoluted matrix
    :param convolutionMatrix:
    :return: max pooled matrix
    """
    maxPoolMatrix = np.zeros((14, 14, 6))

    for fil in range(6):
        for rowStride in range(0, 28, 2):
```

```
            for colStride in range(0, 28, 2):
                maxPoolMatrix[rowStride / 2][colStride / 2][fil] = \
                        np.max(convolutionMatrix[rowStride:rowStride + 1,
                                                 colStride:colStride + 1,
:])

    return maxPoolMatrix


def convolutionLayer2(maxPoolMatrix, filters):
    """
    Convolves the given matrix by applying the filter
    :param maxPoolMatrix1:
    :param filters2:
    :return: convoluted matrix
    """
    convolutionMatrix = np.zeros((10, 10, 16))

    for fil in range(16):
        for rowStride in range(maxPoolMatrix.shape[0] - filters.shape[0]
+ 1):
            for colStride in range(maxPoolMatrix.shape[0] -
filters.shape[0] + 1):
                convolutionMatrix[rowStride][colStride][fil] = \
                    np.sum(filters * maxPoolMatrix[rowStride:rowStride +
5,
                                                   colStride:colStride +
5, :])

    return convolutionMatrix


def maxPoolLayer2(convolutionMatrix):
    """
    Perform max pooling on the given convolution matrix
    :param convolutionMatrix:
    :return: max pooled matrix
    """
    maxPoolMatrix = np.zeros((5, 5, 16))
```

```
    for fil in range(16):
        for rowStride in range(0, 10, 2):
            for colStride in range(0, 10, 2):
                maxPoolMatrix[rowStride / 2][colStride / 2][fil] = \
                        np.max(convolutionMatrix[rowStride:rowStride + 1,
                                                 colStride:colStride + 1,
:])

    return maxPoolMatrix


def forwardPassOfNeuralNetwork(inputs):
    """
    Perform the forward pass of the neural network
    :param inputs:
    :return:
    """
    # Initializing variables
    hiddenLayer1 = 120
    hiddenLayer2 = 84
    outputLayer = 10

    weightsHiddenLayer1 = np.random.uniform(-0.1, 0.1,
size=(hiddenLayer1, inputs.shape[0]))
    weightsHiddenLayer2 = np.random.uniform(-0.1, 0.1,
size=(hiddenLayer2, hiddenLayer1))
    weightsOutputLayer = np.random.uniform(-0.1, 0.1, size=(outputLayer,
hiddenLayer2))

    # Forward Pass of Neural Network
    hiddenLayer1Output = np.matmul(weightsHiddenLayer1, inputs)
    hiddenLayer2Output = np.matmul(weightsHiddenLayer2,
hiddenLayer1Output)
    outputLayerOutput = np.matmul(weightsOutputLayer, hiddenLayer2Output)

    # Performing softmax of the output
    softMaxOutputs = softMax(outputLayerOutput)
    print(softMaxOutputs)
```

```
if __name__ == "__main__":
    imagePath = raw_input("Enter the path of the image: ")
    image = cv2.imread(imagePath)


    # Resize the image to 32 x 32 x 3
    imageResized = np.resize(image, (32, 32, 3))

    img = Image.fromarray(imageResized, 'RGB')
    img = img.resize((312, 312))
    img.show()

    filters1 = np.random.randn(5, 5, 3)

    # Perform the First convolution
    convolutionMatrix1 = convolutionLayer1(imageResized, filters1)

    # Display the intermediate image
    img1 = Image.fromarray(convolutionMatrix1, 'RGB')
    img1 = img1.resize((312, 312))
    img1.show()

    # Apply the ReLU function on the convolution outputs
    reluMatrix1 = ReLU(convolutionMatrix1)

    # Perform the First Max Pooling
    maxPoolMatrix1 = maxPoolLayer1(reluMatrix1)

    # Display the intermediate image
    img2 = Image.fromarray(maxPoolMatrix1, 'RGB')
    img2 = img2.resize((312, 312))
    img2.show()

    # Perform the Second convolution
    filters2 = np.random.randn(5, 5, 6)
    convolutionMatrix2 = convolutionLayer2(maxPoolMatrix1, filters2)

    # Display the intermediate image
    img3 = Image.fromarray(convolutionMatrix2, 'RGB')
    img3 = img3.resize((312, 312))
```

```
img3.show()

# Apply ReLU function on the second convolution
reluMatrix2 = ReLU(convolutionMatrix2)

# Perform the Second Max Pooling
maxPoolMatrix2 = maxPoolLayer2(convolutionMatrix2)

# Display the intermediate image
img4 = Image.fromarray(maxPoolMatrix2, 'RGB')
img4 = img4.resize((312, 312))
img4.show()

# Perform the forward pass of the Neural Network
inputs = maxPoolMatrix2.flatten()
forwardPassOfNeuralNetwork(inputs)
```

1.2. Practical aspects in Deep Networks

1.2.1. **Q1: [0.5 point]**

What are the number of parameters in convolution layers with K filters each of size 3*w*h.

Ans: In the first layer, the filters are of size (3 x 5 x 5), hence the parameters corresponding to the first convolution layers are **456**

1.2.2. **Q2: [0.5 points]**

What are the number of parameters in a max pooling operation?

Ans: **Two** parameters - Size of the pooling window and Stride of the window.

1.2.3. **Q3: [0.5 point]**

Which of the operations contain most number of parameters? (a) conv (b) pool (c) Fully connected layer (FC) (d) Relu

Ans: **(c)** Fully connected Layer

1.2.4. **Q4: [0.5 point]**

Which operation consume most amount of memory? (a) initial convolution layers (b) fully connected layers at the end

Ans: **(b)** Fully Connected Layers

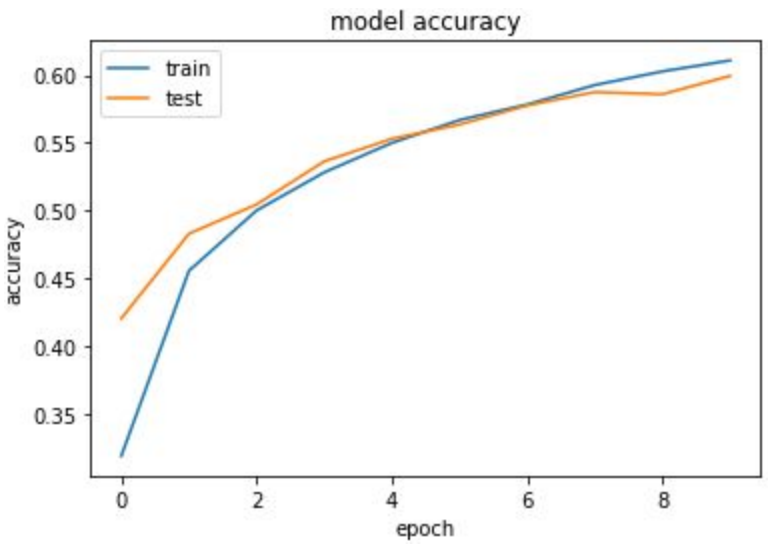
Initial Setting for the Neural Network

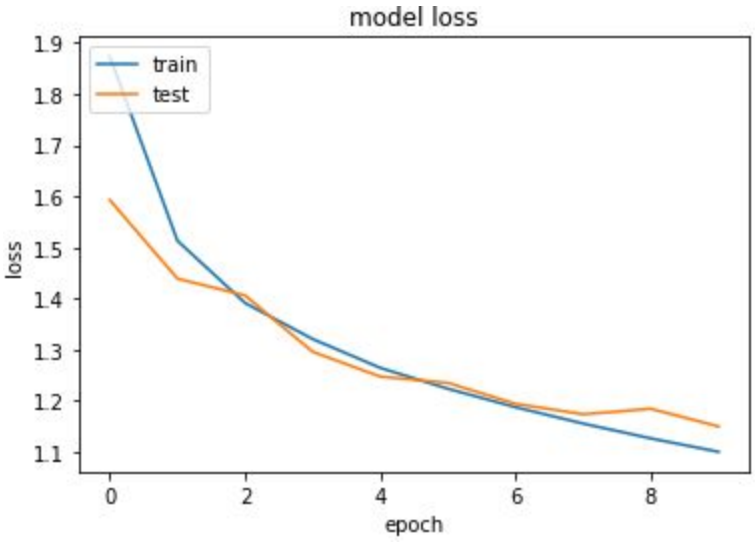
```
batchSize = 50                  #-- Training Batch Size
num_classes = 10                #-- Number of classes in CIFAR-10 dataset
num_epochs = 10                 #-- Number of epochs for training
learningRate = 0.001             #-- Learning rate for the network
lr_weight_decay = 0.95          #-- Learning weight decay. Reduce the learn
rate by 0.95 after epoch

img_rows, img_cols = 32, 32   #-- input image dimensions

Y_train = np_utils.to_categorical(y_train, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)
```

Results obtained with the above setting





Test score: 1.1499698184967042
Test accuracy: 0.5993

1.2.5. **Q5: [2 points]**

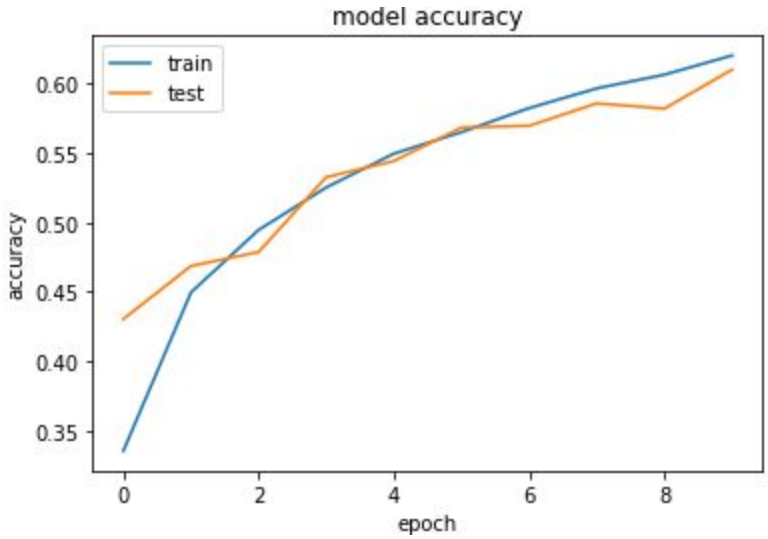
Experiment with **learning rate** (learningRate) and notice the behaviour of the learning process. Plot your observations in a graph with brief explanation. Take the values on a log scale. Vary only one parameter at a time.

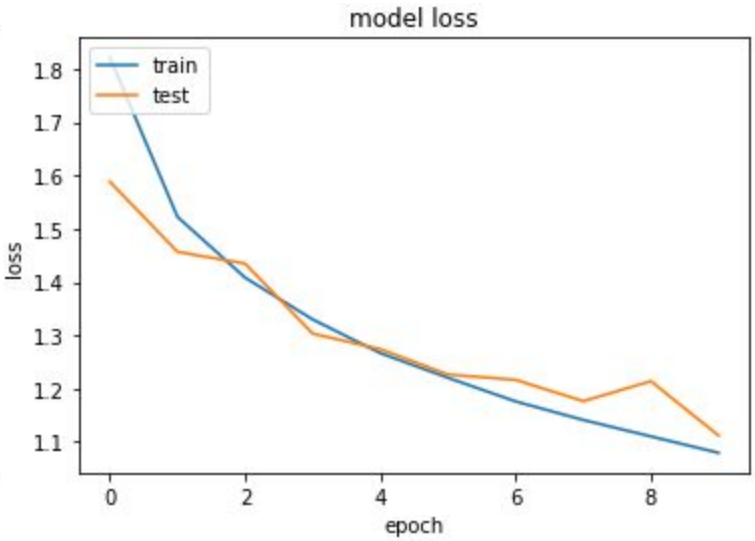
Ans:

By Observing the learning rate vs test score we can see that as the as the learning rate decreases the test score decreases.

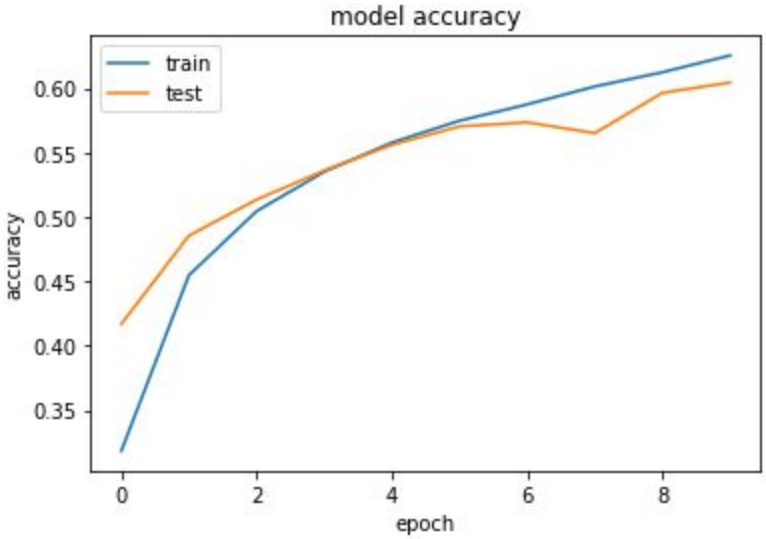
Learning Rate	**Test Score**	**Test Accuracy**
0.0001	1.1121616285324096	0.6102
0.001	1.1499698184967042	0.5993
0.01	1.1390595510482788	0.6044
0.1	1.1419799154281616	0.5946
1	1.1635176993370056	0.59

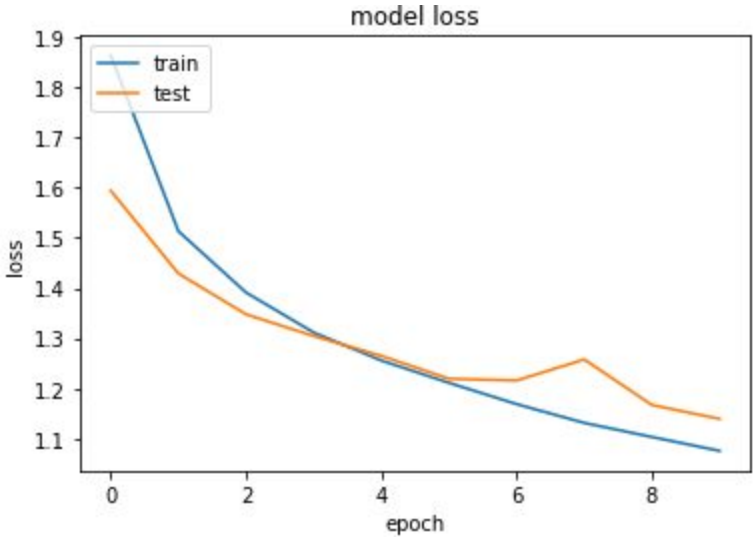
Learning Rate (0.0001)



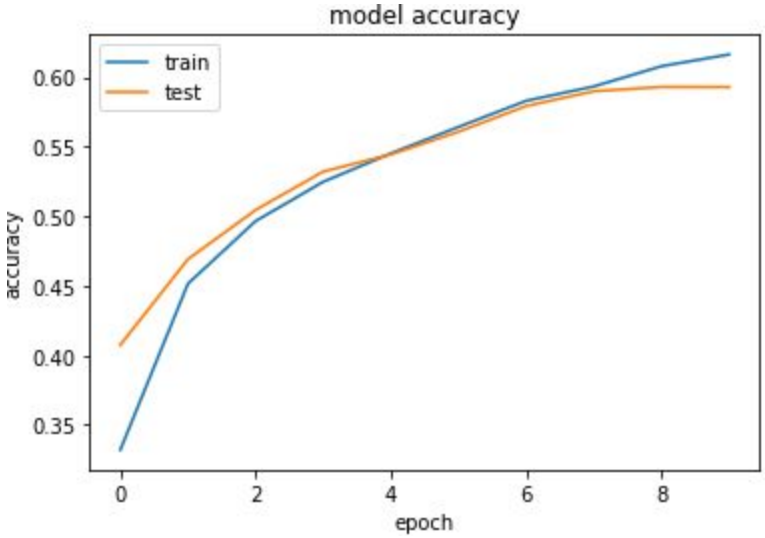


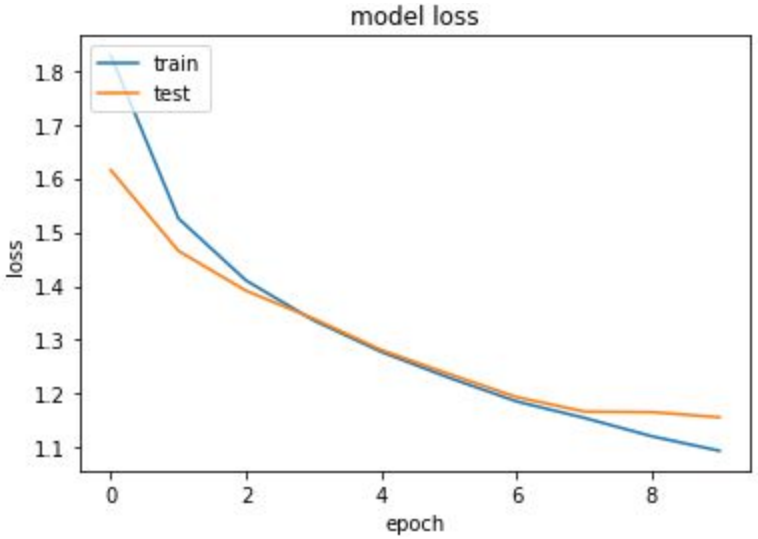
Learning Rate (0.001)



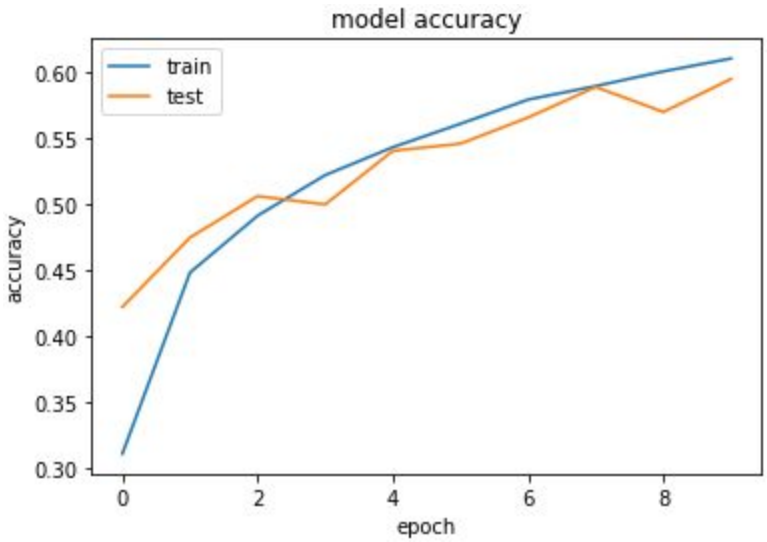


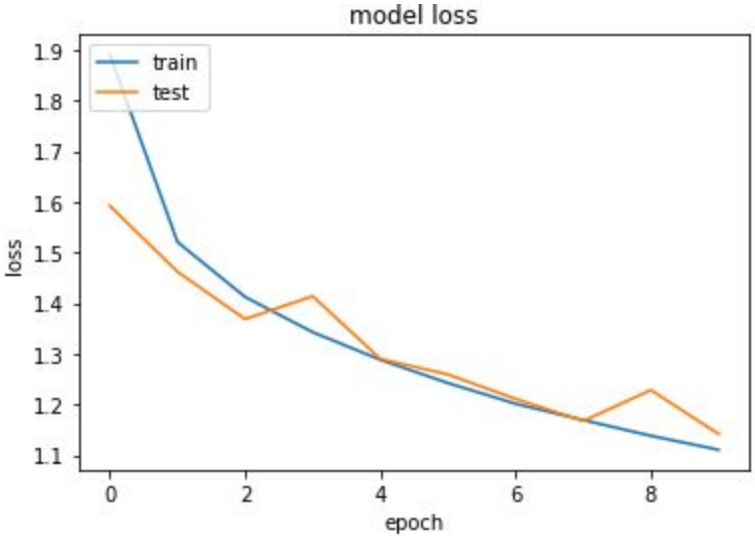
Learning Rate (0.01)



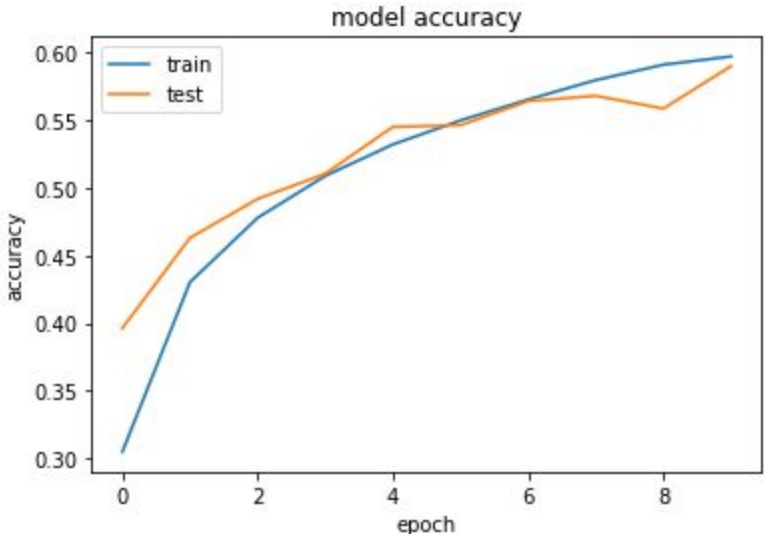


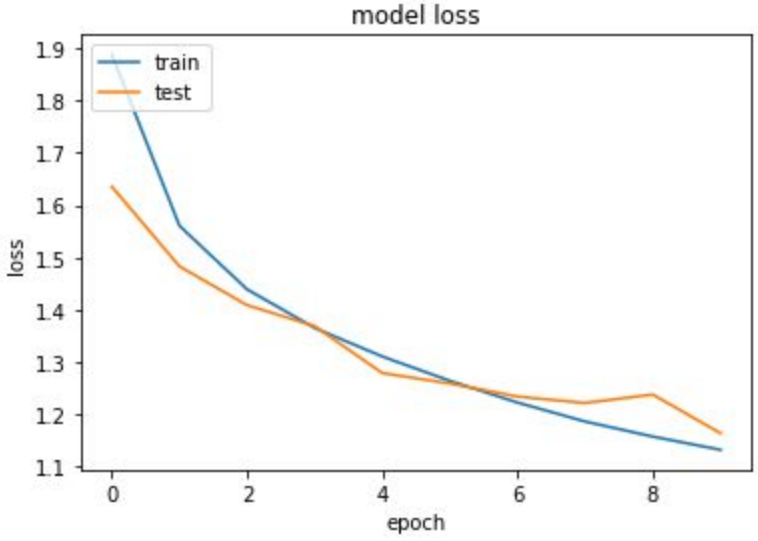
Learning Rate (0.1)





Learning Rate (1)





Overall Results

1.2.6. **Q6: [2 points]**

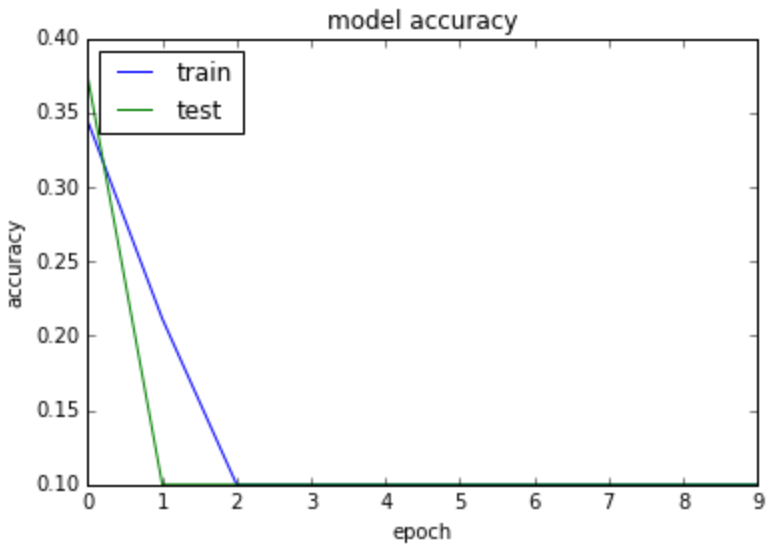
Currently, the **batch-size** is 50. Notice the training loss curve if batch size is changed to 1. Is it smooth or fluctating? Show the effect of batch-size on the learning curves in a plot. Take the values on a log scale. Vary only one parameter at a time.

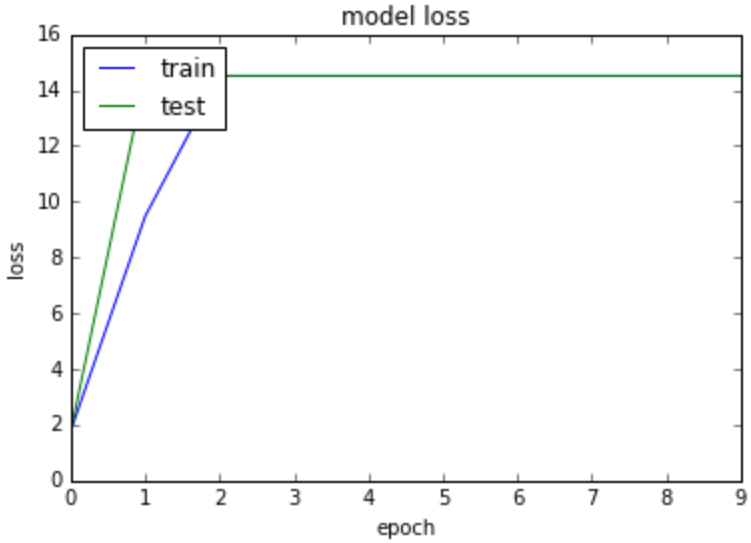
Ans:

From the graph we can see that the accuracy increases till a particular batch size but when the batch size gets too large then the accuracy decreases.

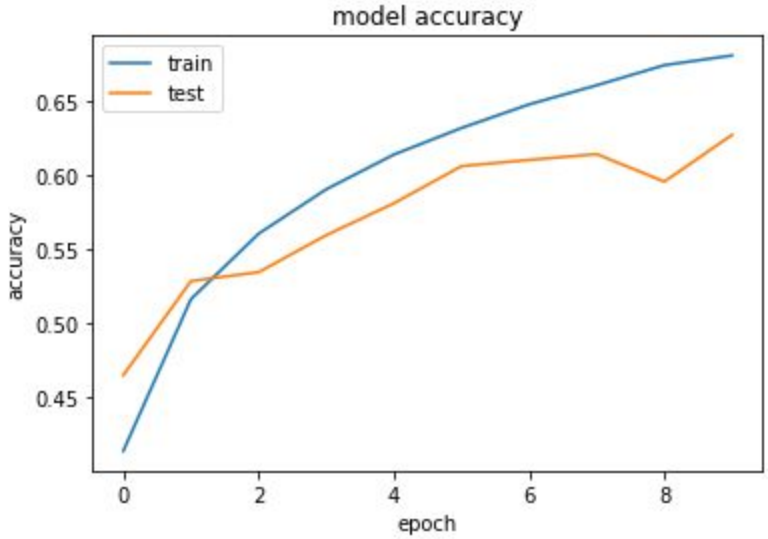
Batch Size	**Test Score**	**Test Accuracy**
1	14.506285664367676	0.1
5	1.1340943577766418	0.6272
50	1.1407447536468507	0.5958
500	1.0882536724090577	0.6256
5000	2.148634006881714	0.2143

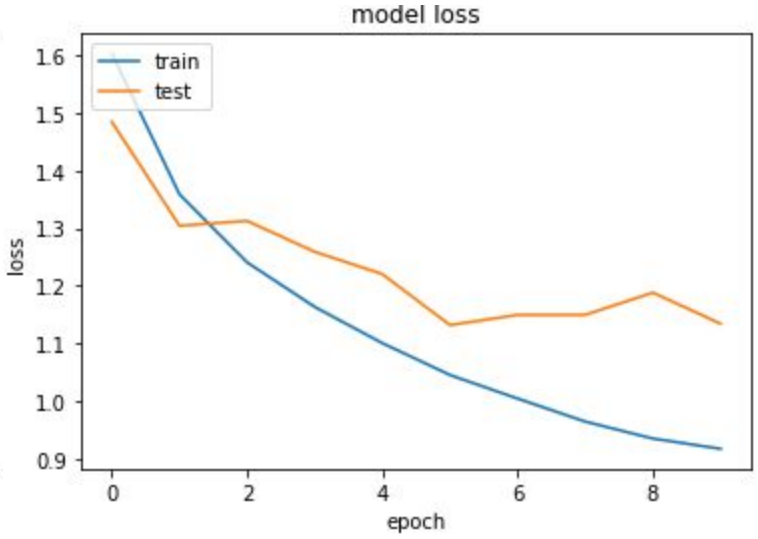
Batch Size (1)



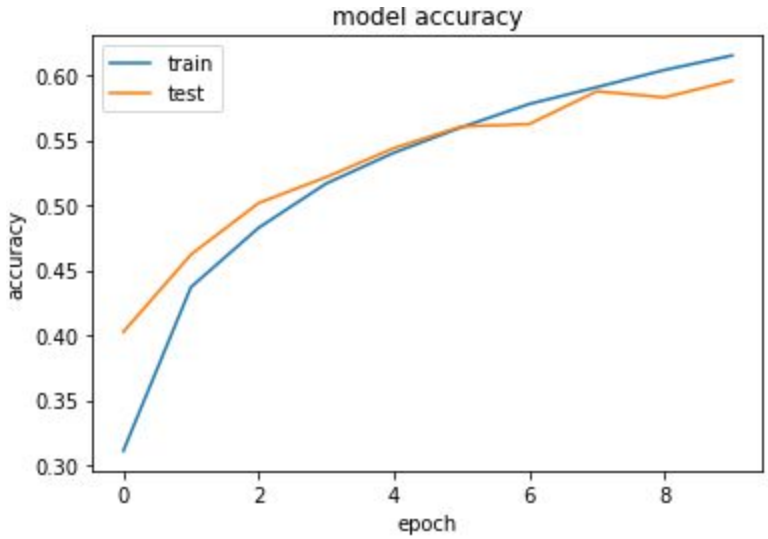


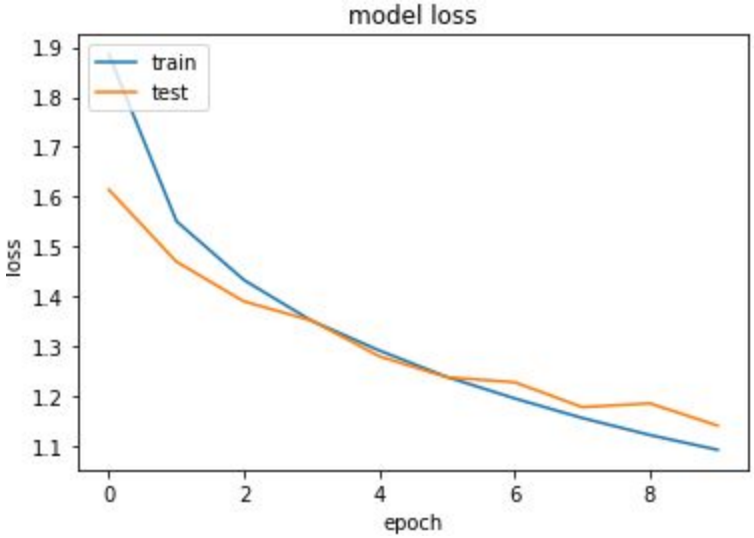
Batch Size (5)



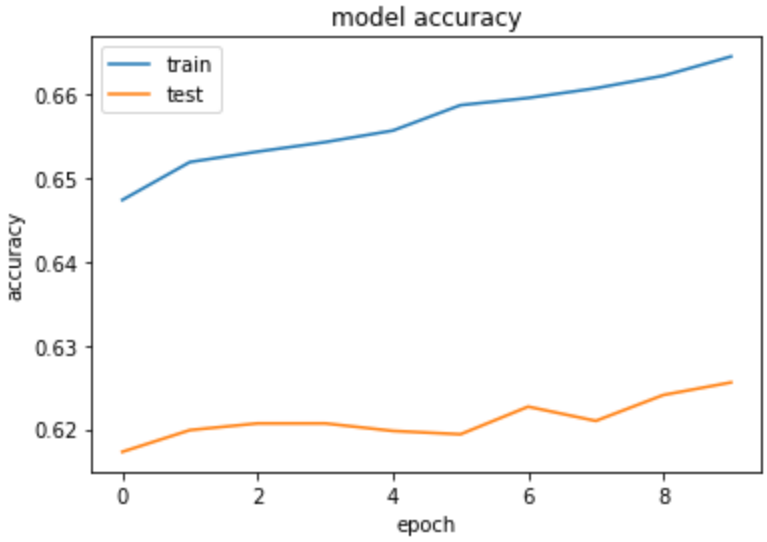


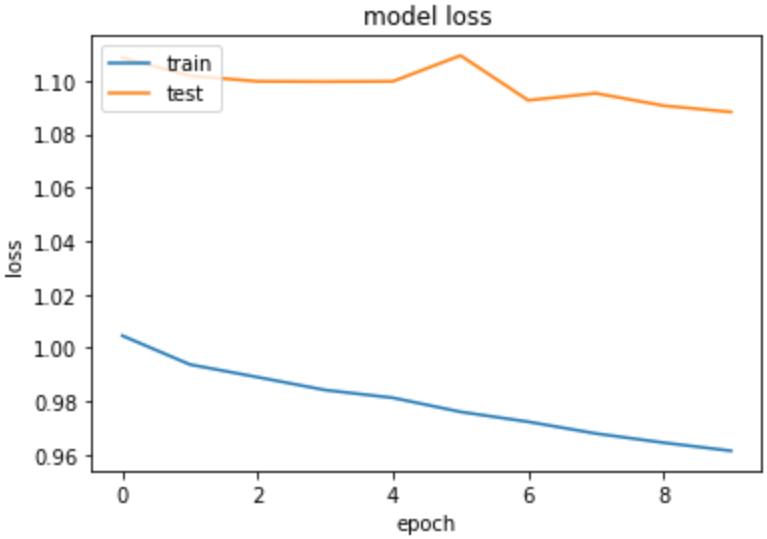
Batch Size (50)



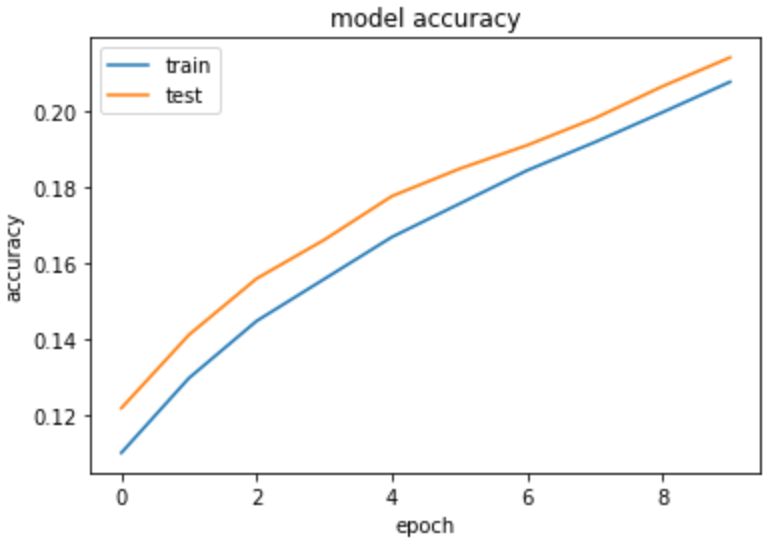


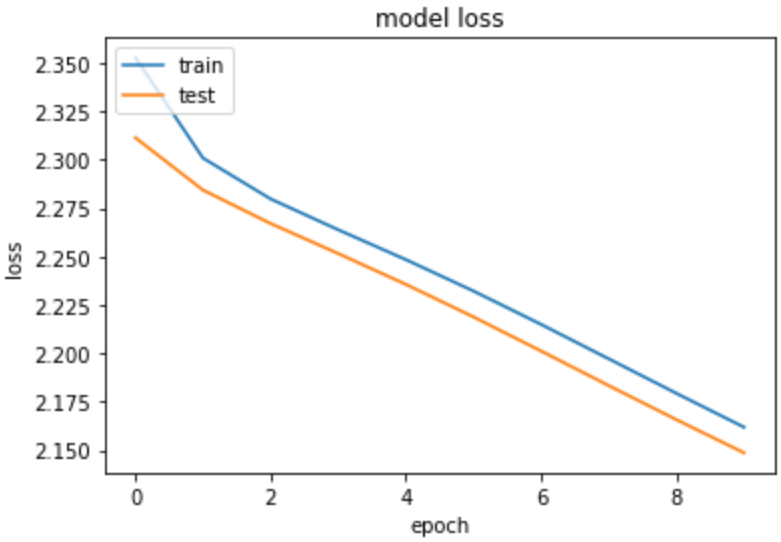
Batch Size (500)



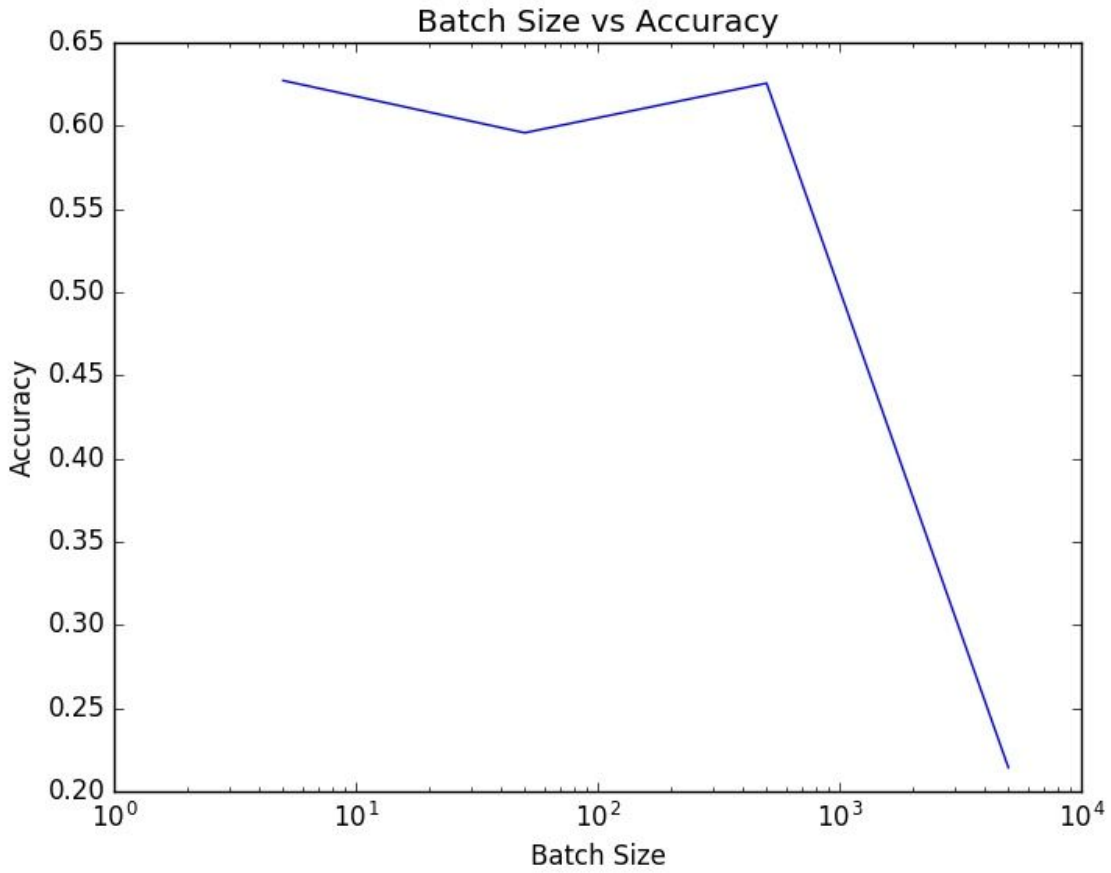


Batch Size (5000)





Overall Results



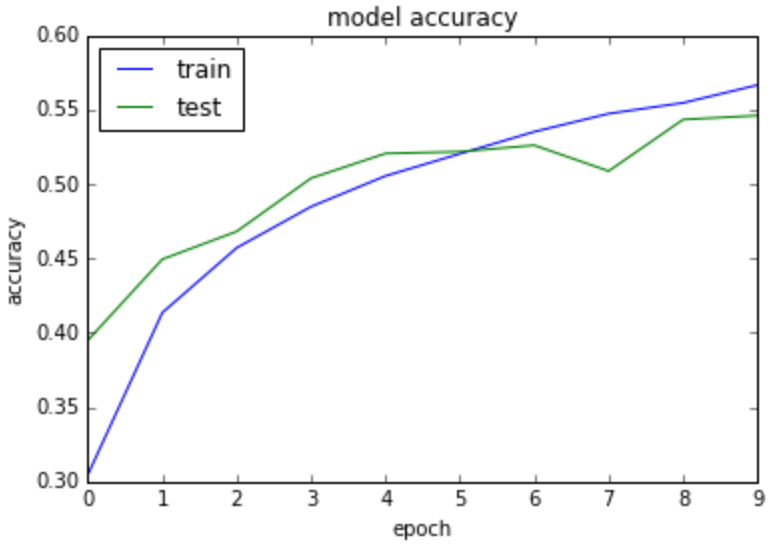
1.2.7. **Q7: [2 points]**

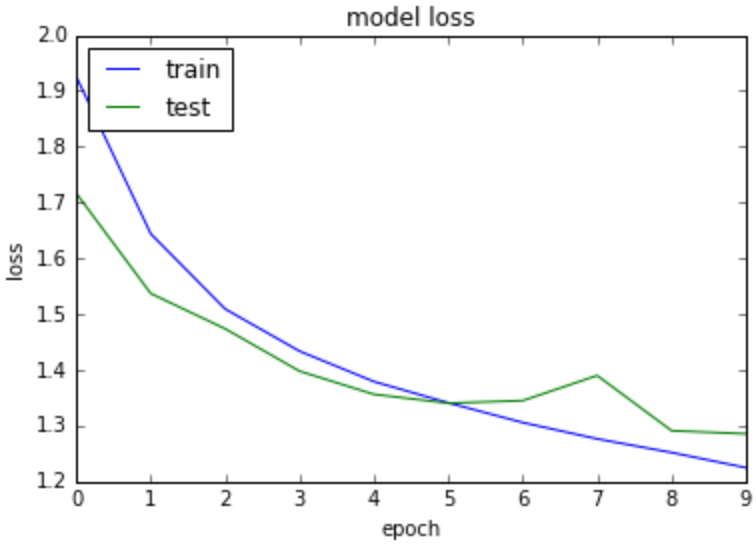
Increase the **number of convolution filters** and experiment. Present your observations using plots and brief explanations. Take the values on a log scale. Vary only one parameter at a time.

Ans:

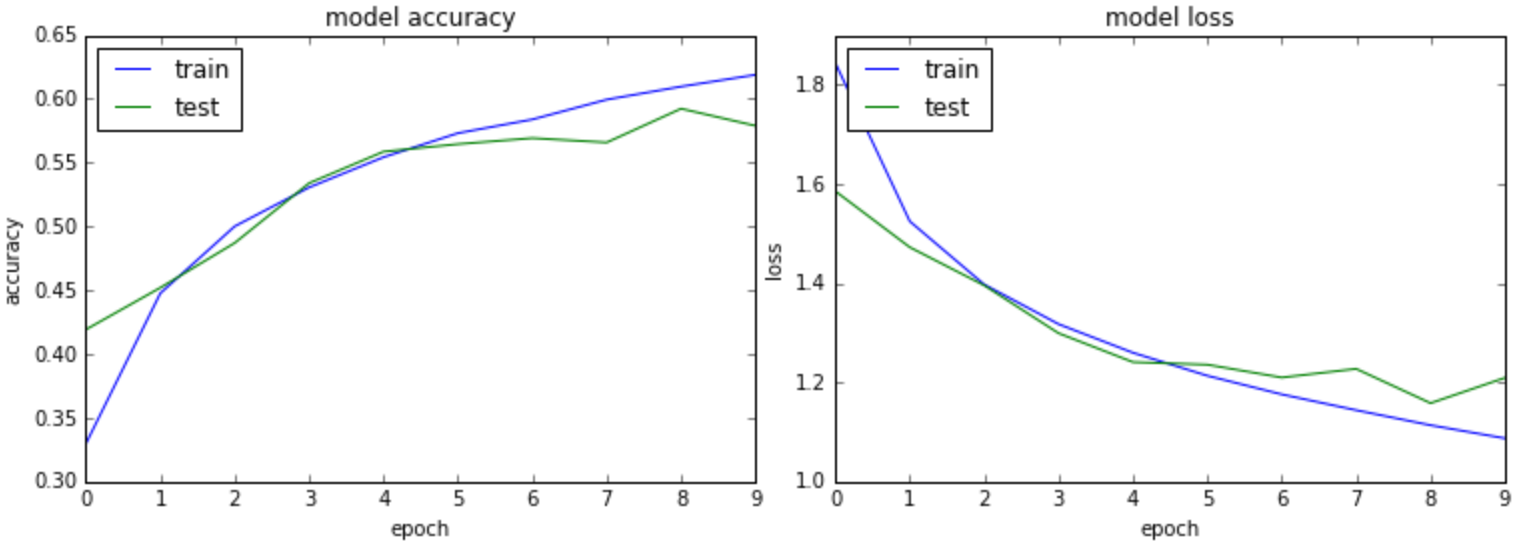
No of Convolution Filters at Layer 1	**No of Convolution Filters at Layer 2**	**Test Score**	**Test Accuracy**
3	16	1.285635710144043	0.546
6	16	1.2093229509353638	0.579
10	16	1.0924378486633302	0.6181
16	16	1.0143535257339478	0.6494
160	16	0.9815066294670105	0.6592
6	3	1.376261851119995	0.5008
6	6	1.2286624605178833	0.5675
6	10	1.2068831861495972	0.5787
6	50	1.0443928214073182	0.6336
6	100	1.0486767791748046	0.6403

Filters Layer 1 : 3 & Filters Layer 2: 16

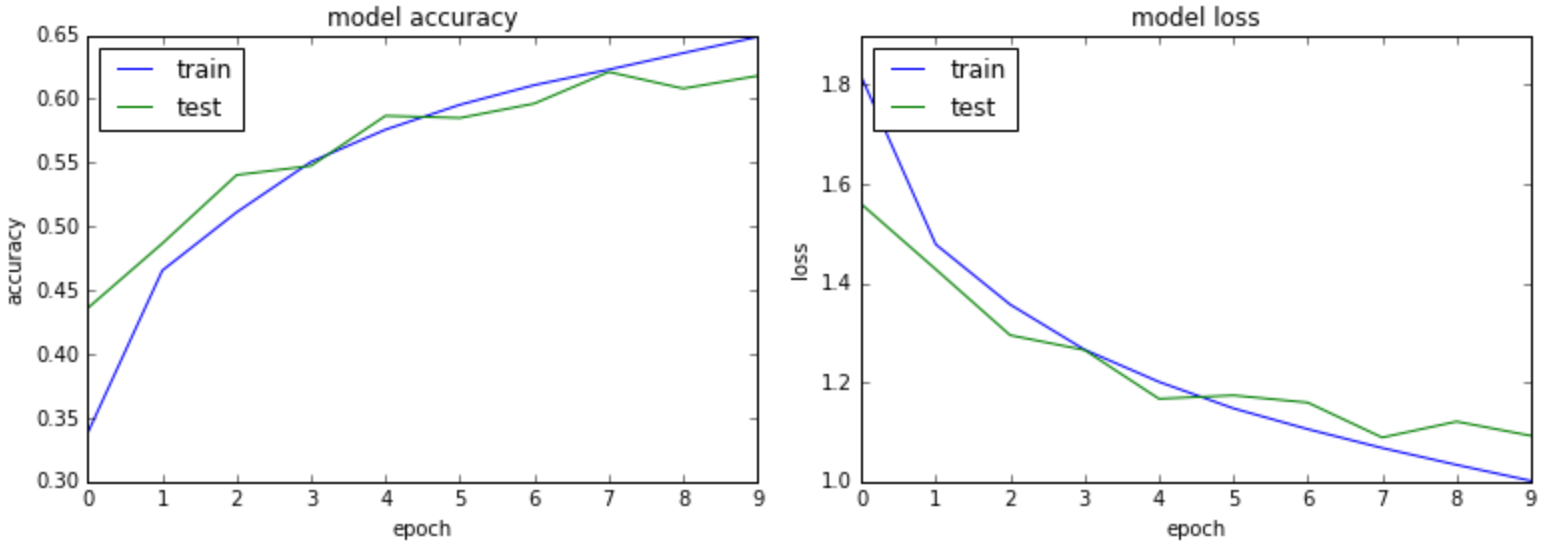




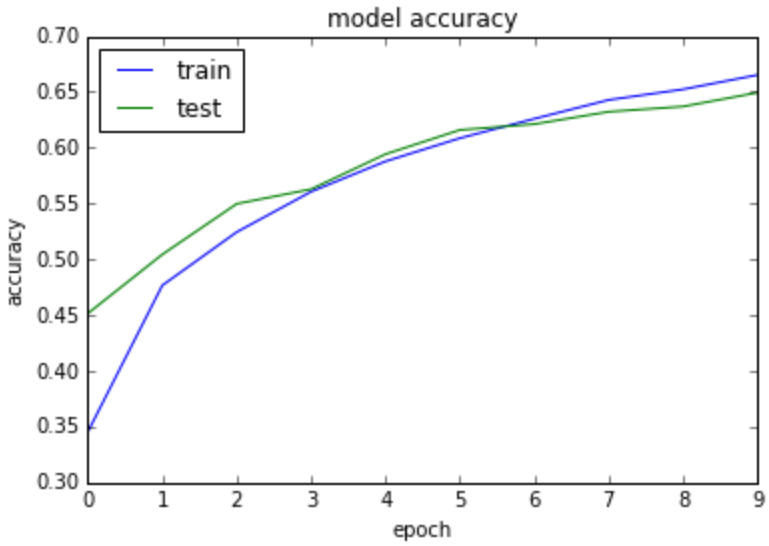
Filters Layer 1 : 6 & Filters Layer 2: 16

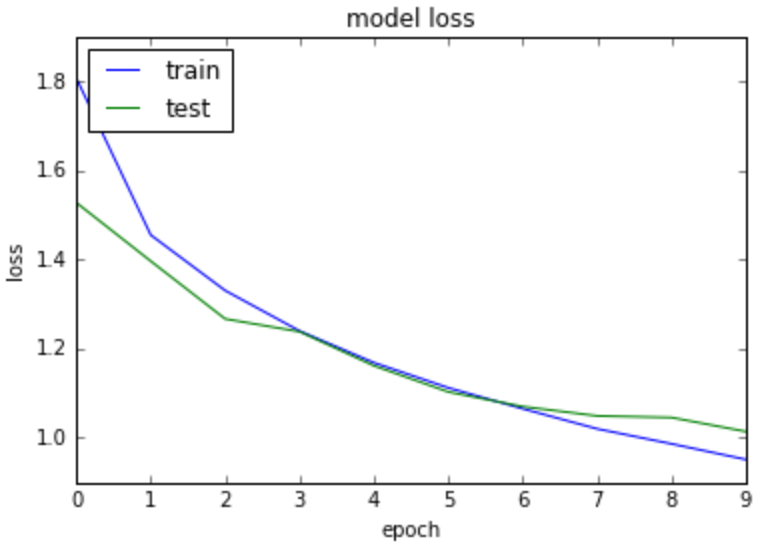


Filters Layer 1 : 10 & Filters Layer 2: 16

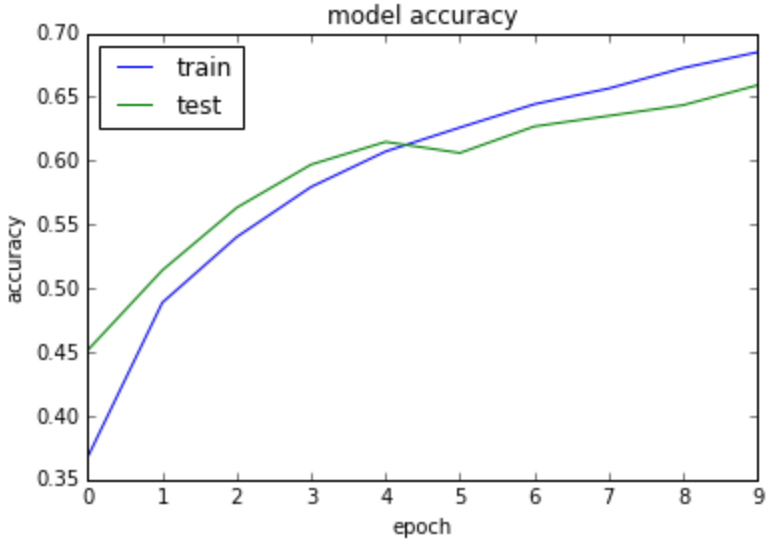


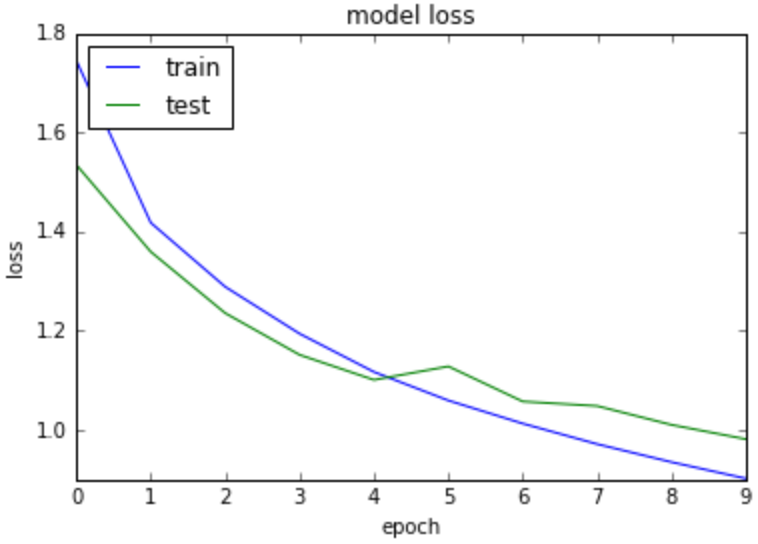
Filters Layer 1 : 16 & Filters Layer 2: 16



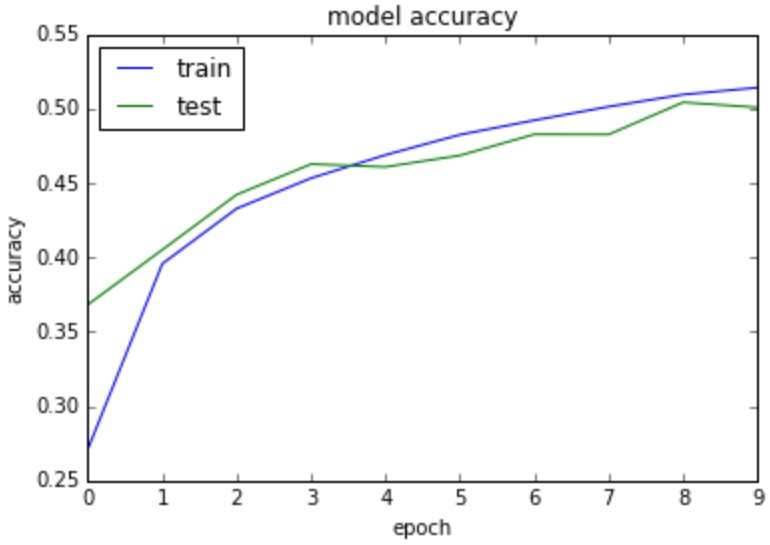


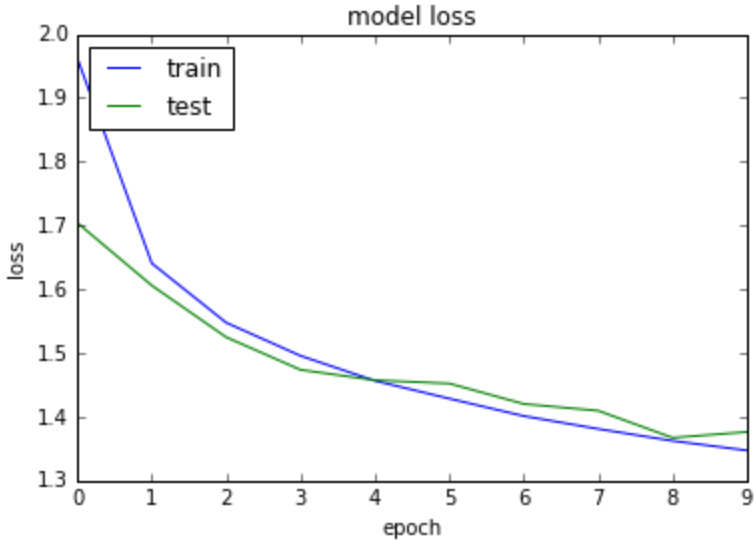
Filters Layer 1 : 160 & Filters Layer 2: 16



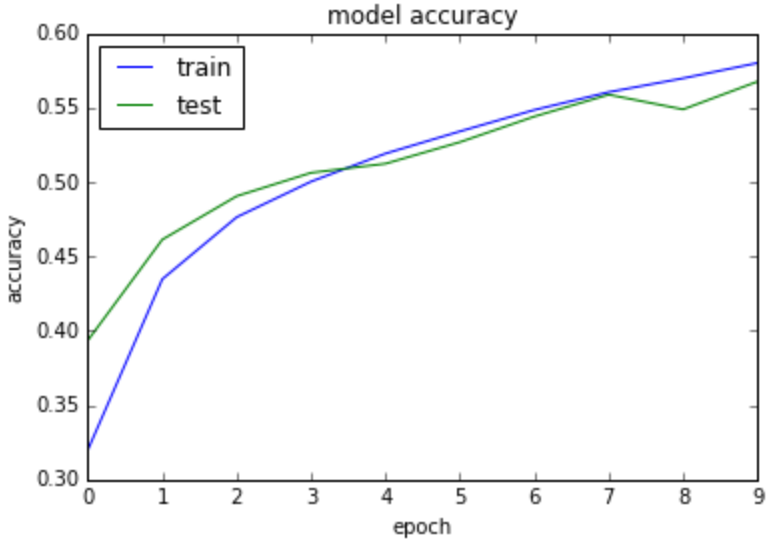


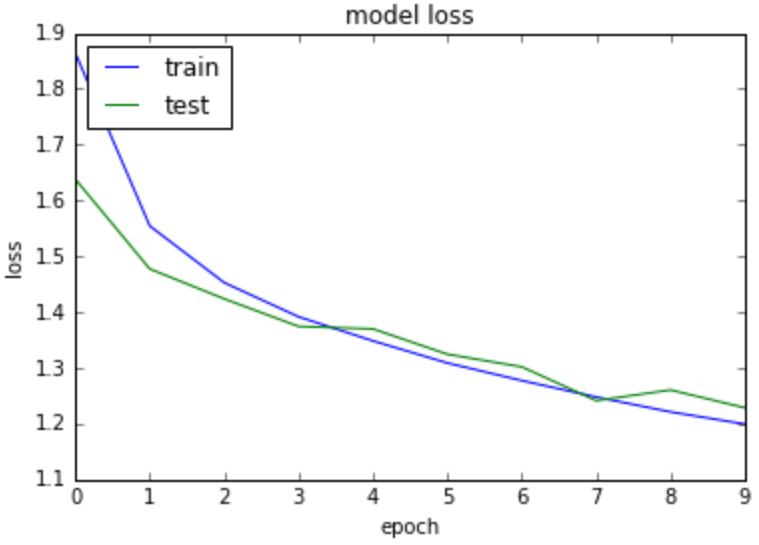
Filters Layer 1 : 6 & Filters Layer 2: 3



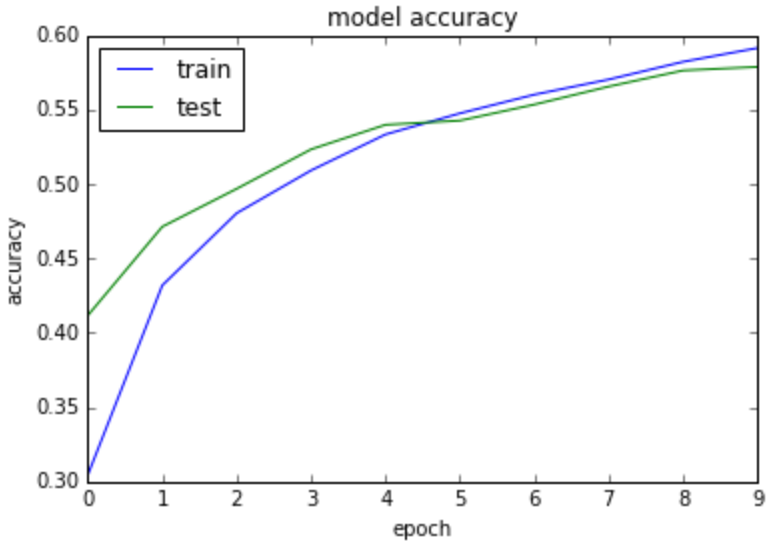


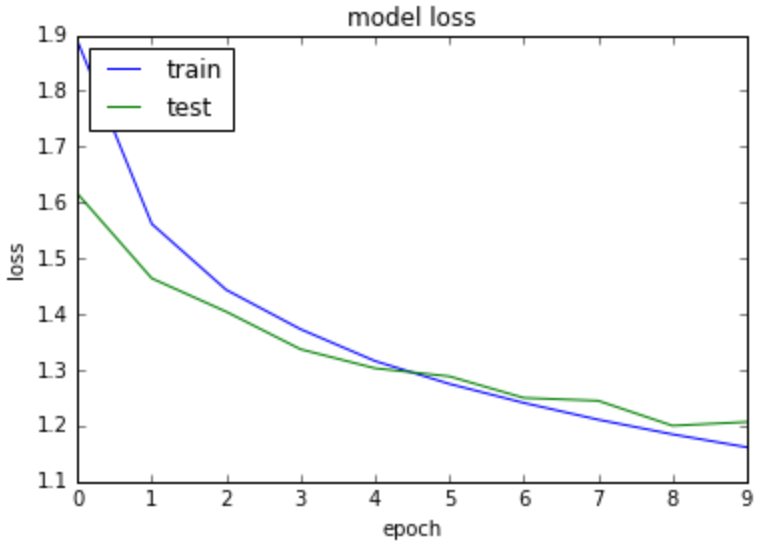
Filters Layer 1 : 6 & Filters Layer 2: 6



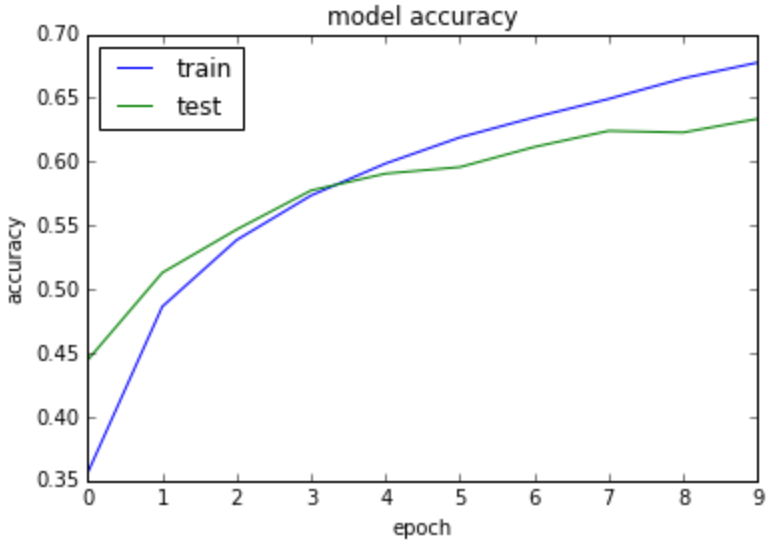


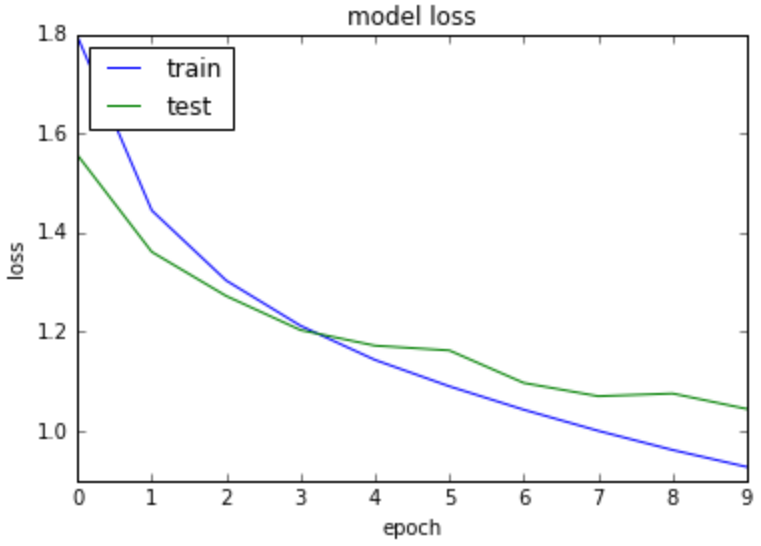
Filters Layer 1 : 6 & Filters Layer 2: 10



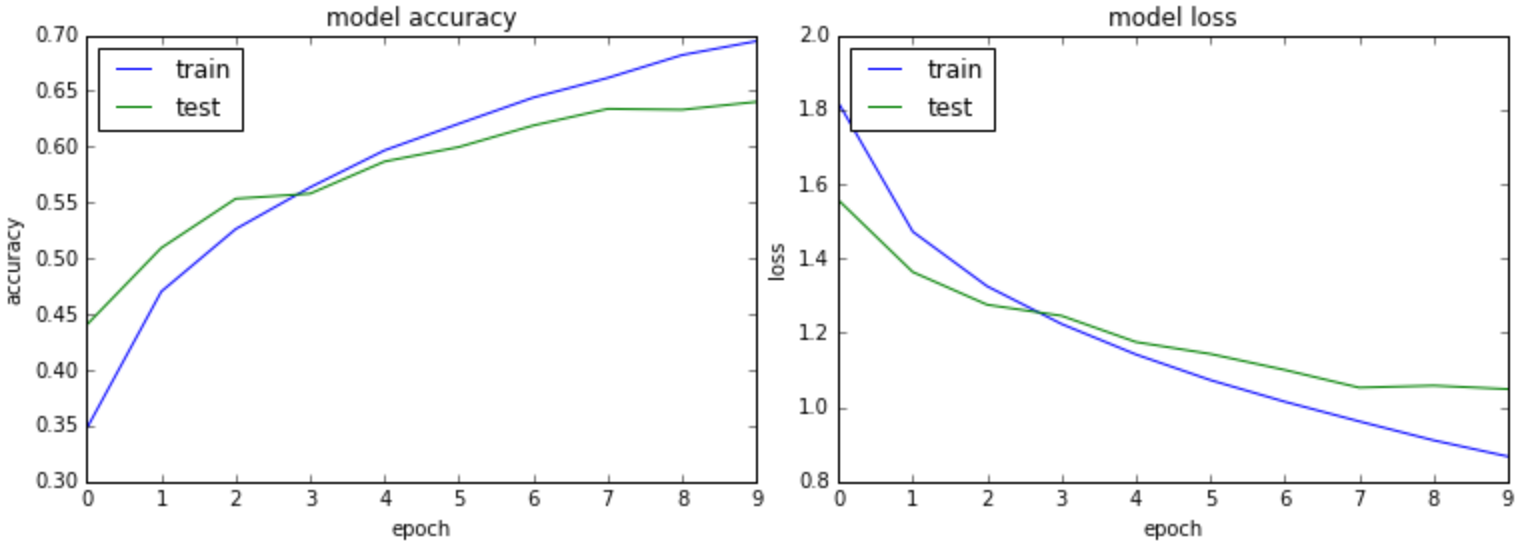


Filters Layer 1 : 6 & Filters Layer 2: 50

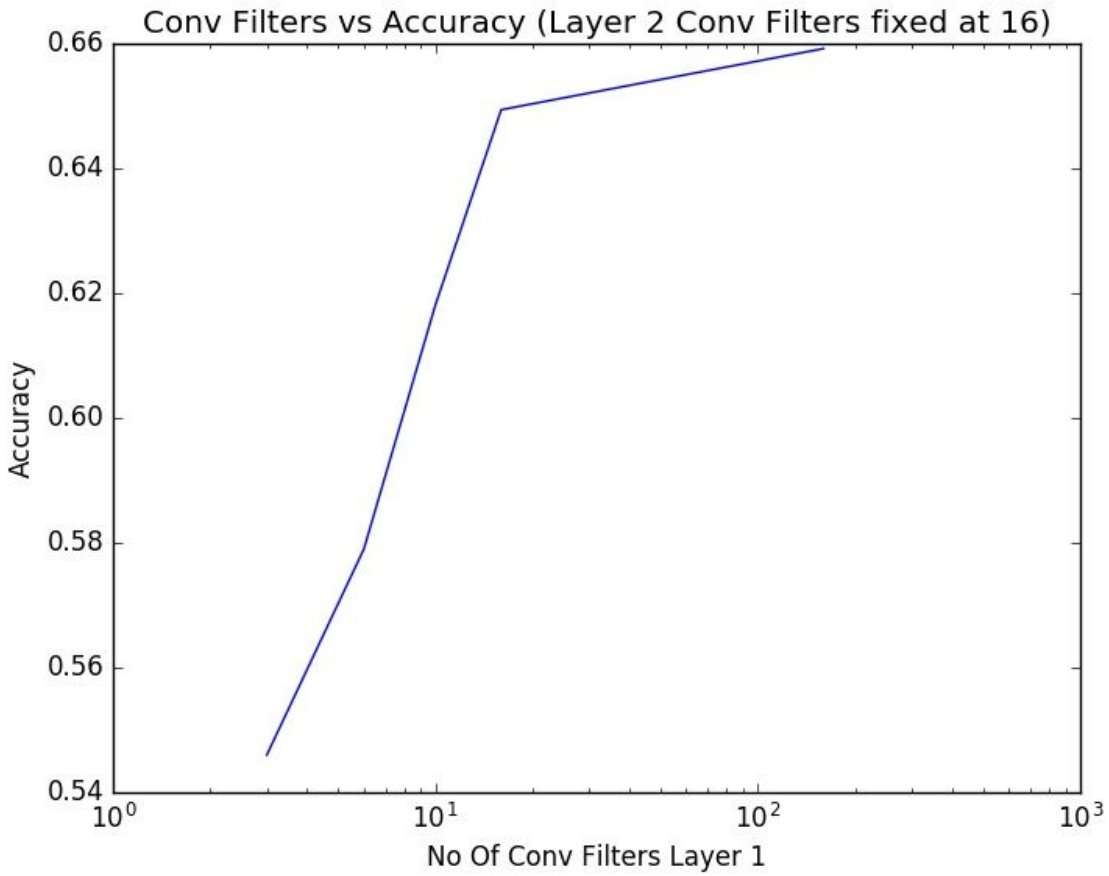




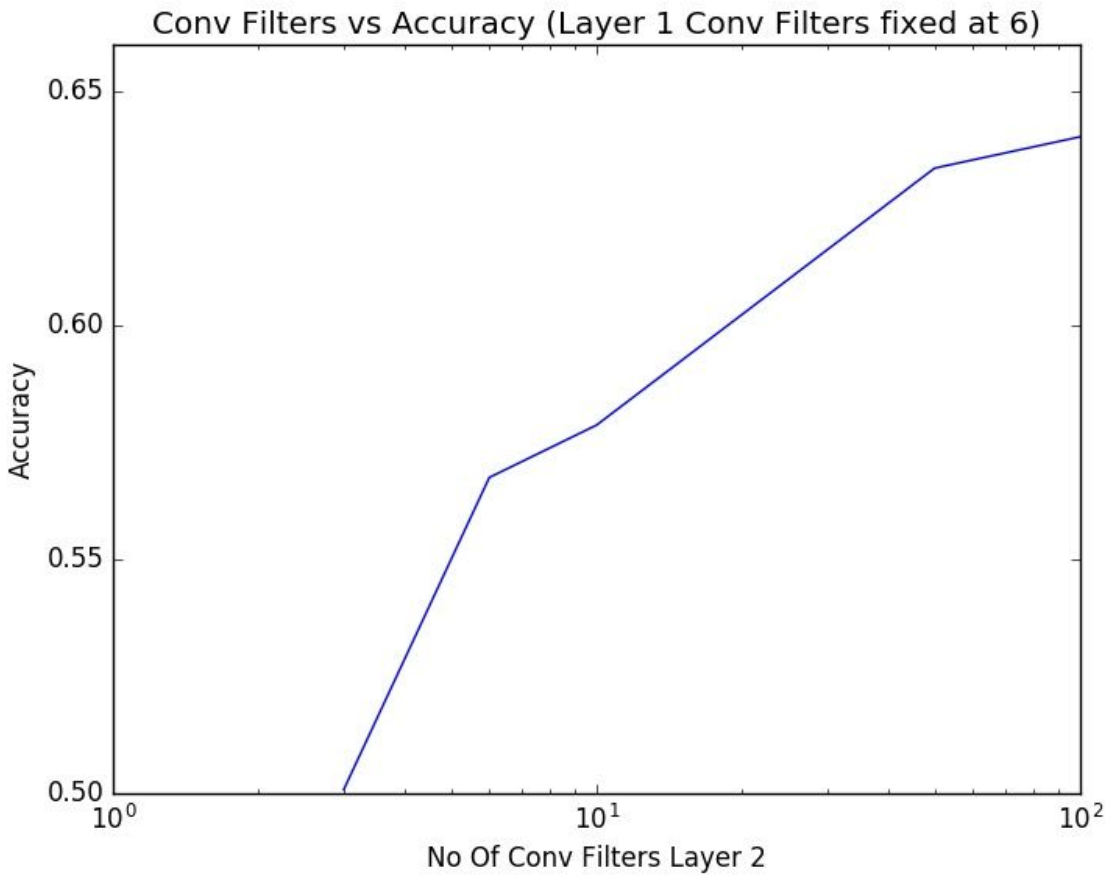
Filters Layer 1 : 6 & Filters Layer 2: 100



Overall Results when Filters of Layer 2 are fixed at 16



Overall Results when Filters of Layer 1 are fixed at 6

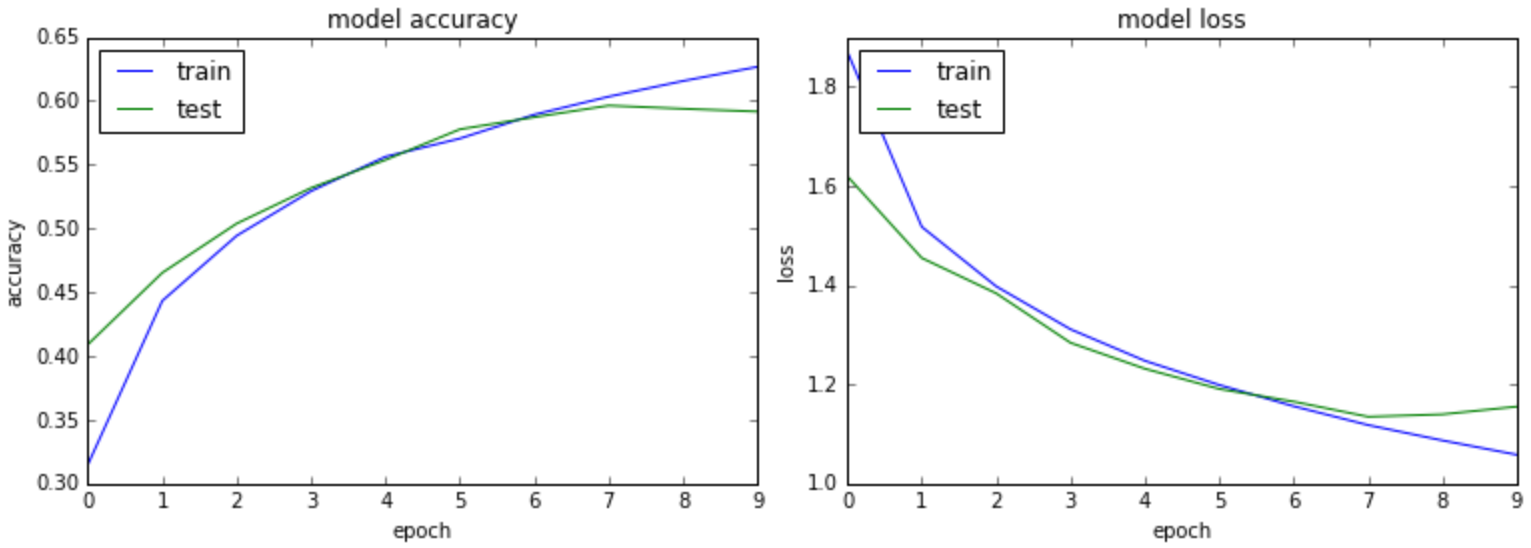


1.2.8. **Q8: [2 points]**

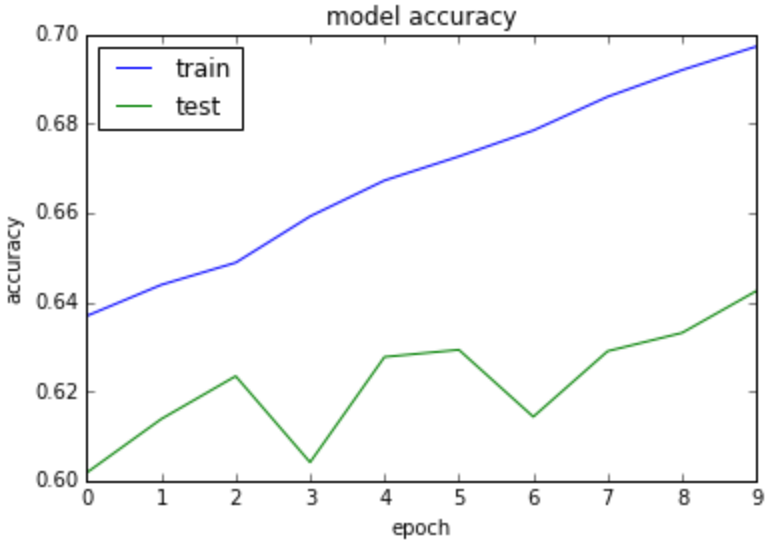
What do you observe if you increase the **number of layers** (depth of the network) ? Present your observations using plots and brief explanations.
Ans:

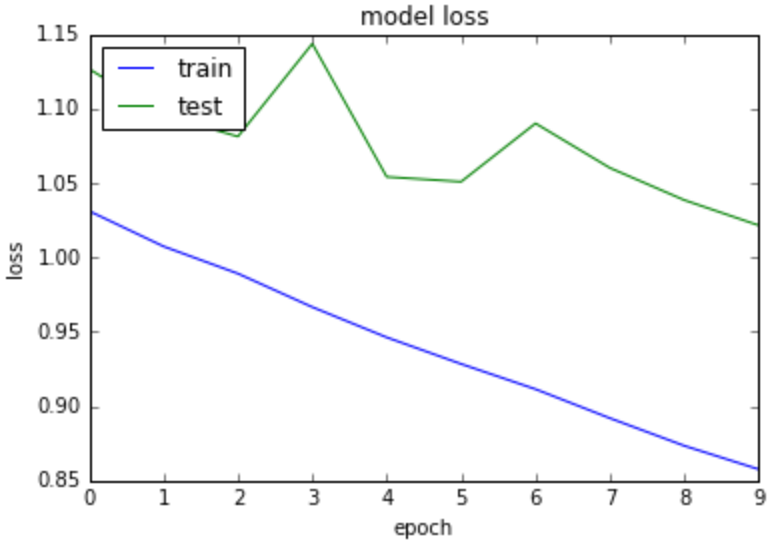
Number of Layers	**TestScore**	**Accuracy**
3 Hidden layers with **120, 100, 84** neurons	1.1552771268844604	0.5917
3 Hidden layers with **120, 84, 50** neurons	1.0215245670318605	0.6425
4 Hidden layers with **120, 100, 84, 50** neurons	1.1550739307403564	0.5977
5 Hidden layers with **120, 100, 84, 50, 30** neurons	1.185371272468567	0.581

3 Hidden layers with 120, 100, 84 neurons

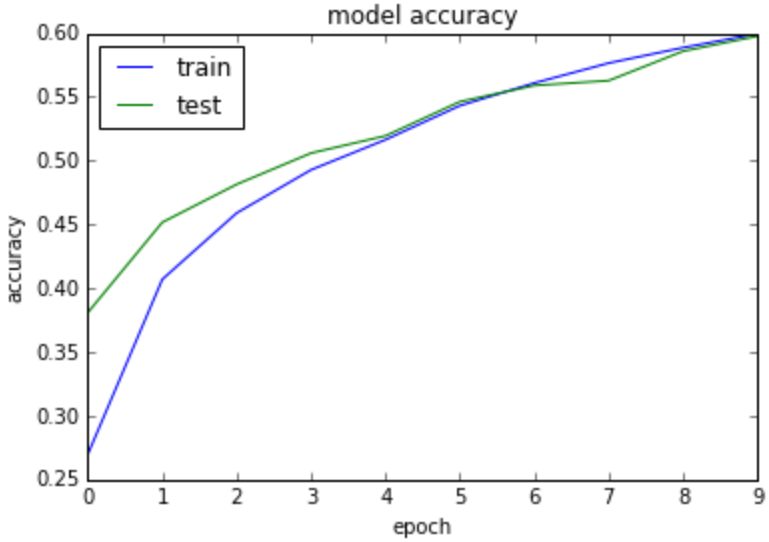


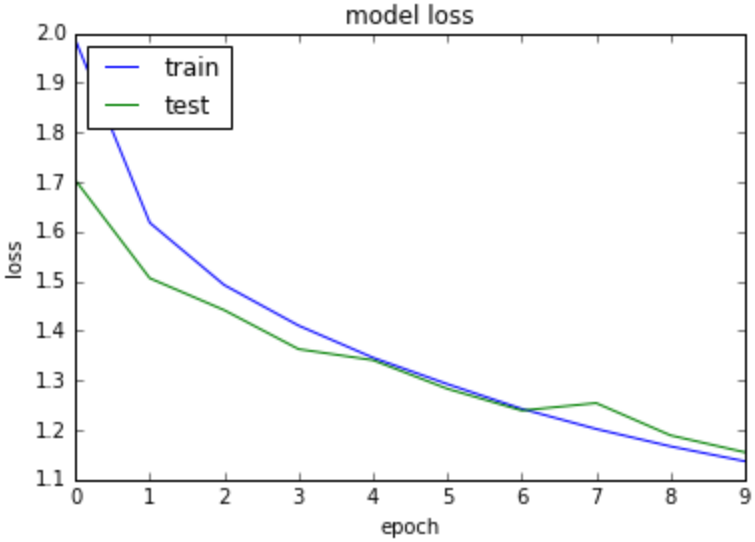
3 Hidden layers with 120, 84, 50 neurons



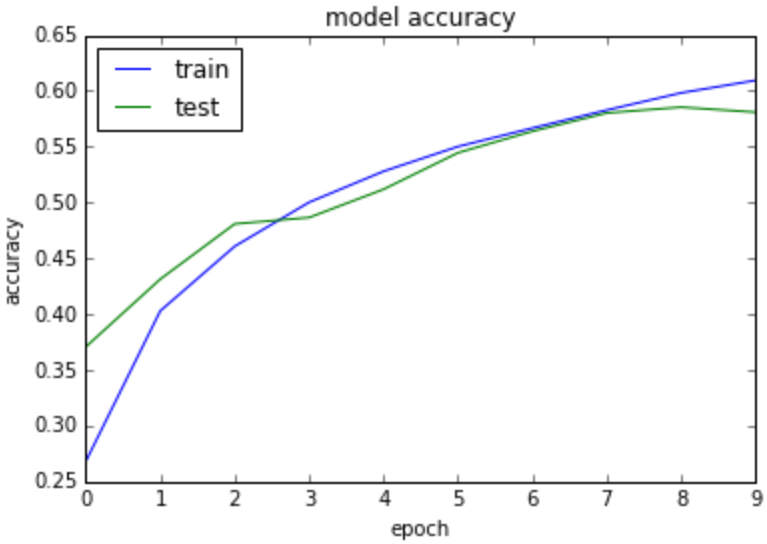


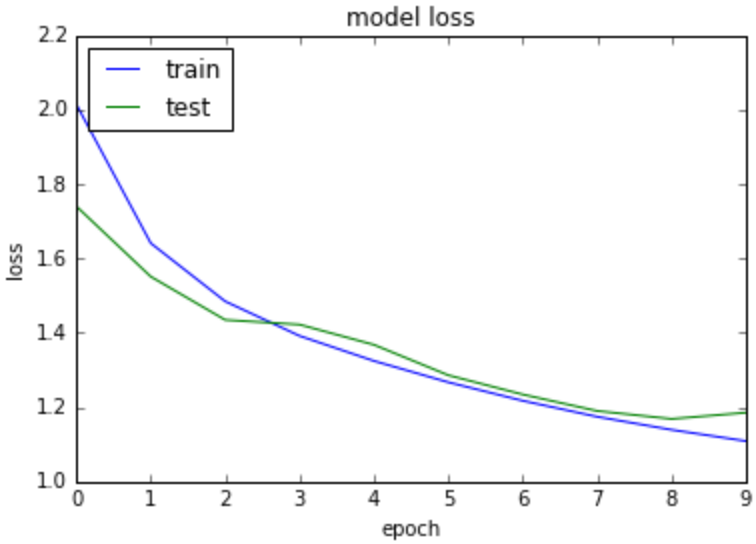
4 Hidden layers with 120, 100, 84, 50 neurons





5 Hidden layers with 120, 100, 84, 50, 30 neurons





1.2.9. **Q9: [2 points]**

What do you observe if you change the **activation functions** (tanh, relu, sigmoid) ? Present your observations using plots and brief explanations.
Ans:

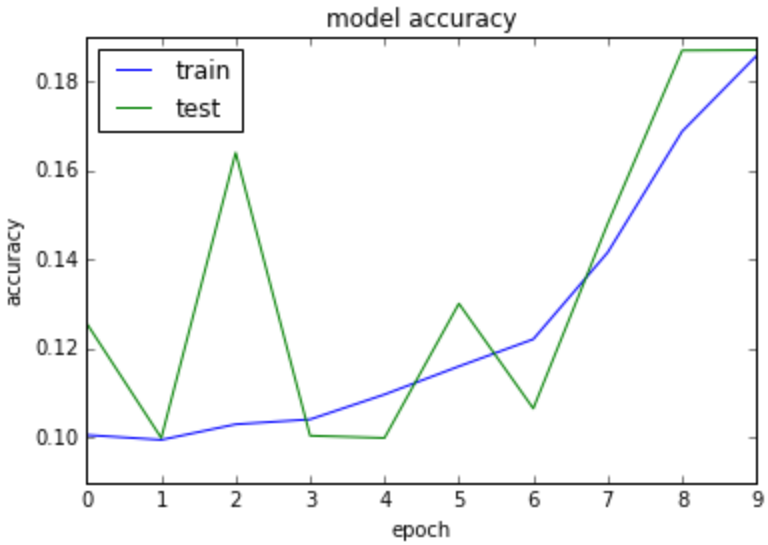
Sigmoid - Accuracy with this function is significantly lower due to vanishing gradient problem.

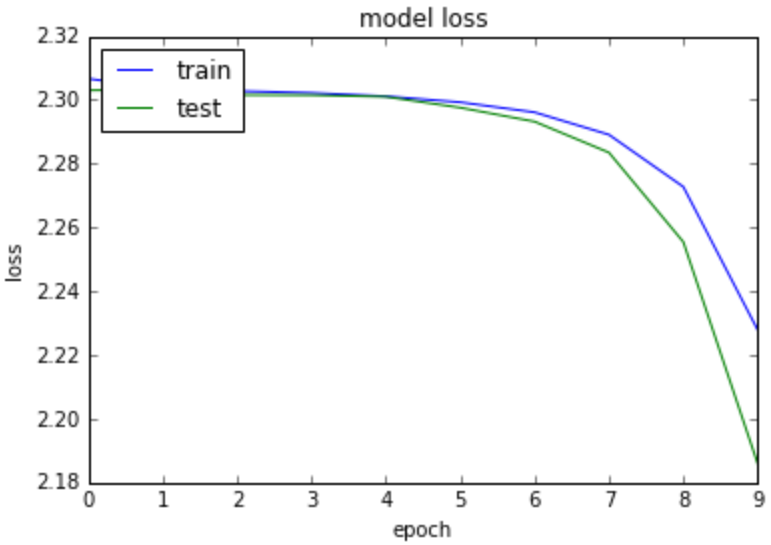
Tanh - Accuracy here is much improved because here the gradient is much stronger compared to sigmoid.

ReLU - This activation handles the vanishing gradient problem in the best way and hence the accuracy is the highest comparitively using this function.

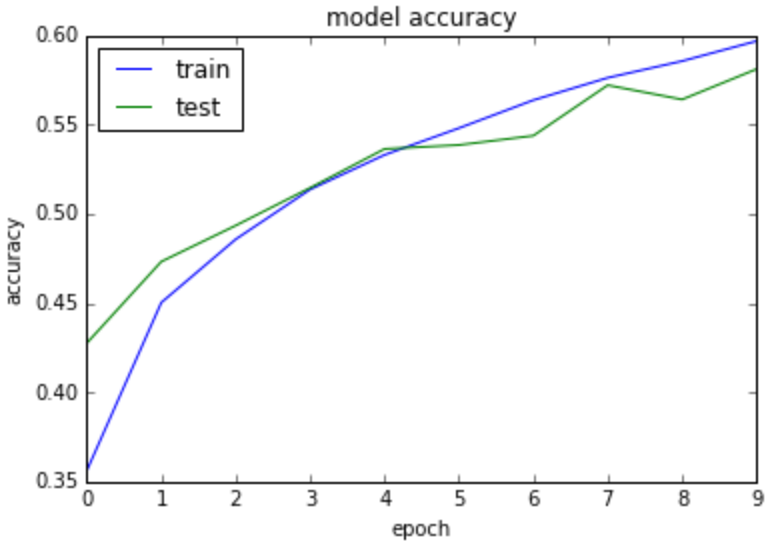
Activation Function	Test Score	Accuracy
Sigmoid	2.1858888526916505	0.1869
Tanh	1.1710123237609864	0.581
ReLU	1.1294254224777223	0.6052

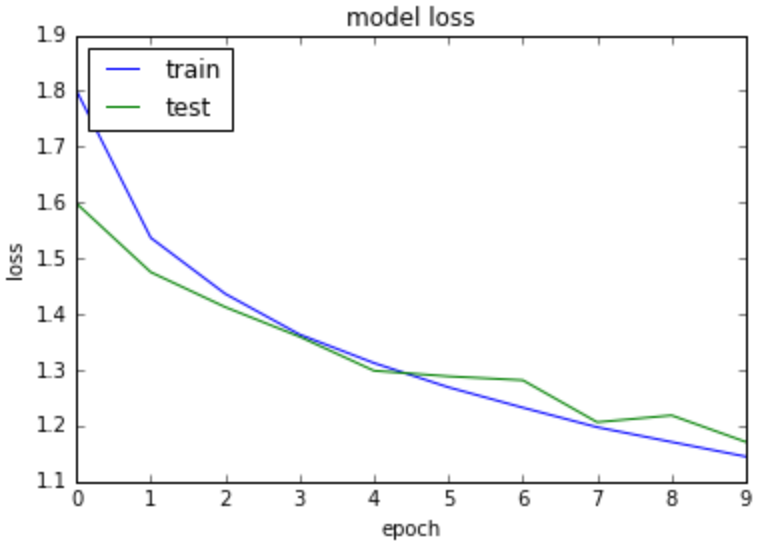
Sigmoid Activation Function



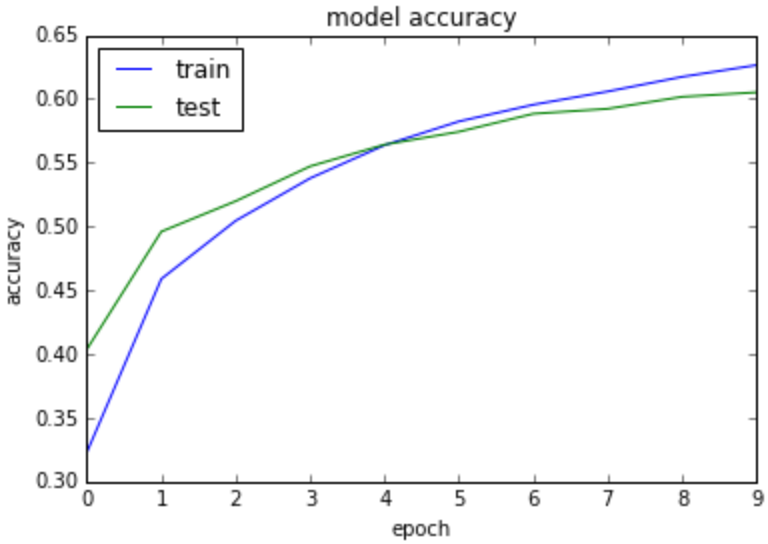


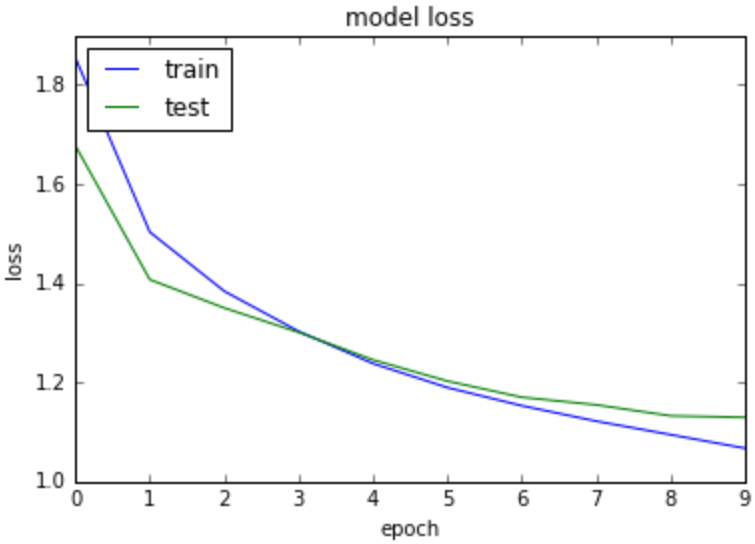
Tanh Activation Function





ReLU Activation Function





1.2.10. **Q10: [1 points]**

CNN training requires lot of training data. In the absence of large training data, a common practice is to use synthetic data using operations such as flipping, scaling, etc. Can you think of any other two operations techniques that can help to increase the training set? Demonstrate these effects with sufficient explanation.

Ans:

The Technique used to increase the training set is **Data Augmentation.**

Techniques for achieving the above are as follows -

1) Adding Salt and Pepper Noise -

   Salt and Pepper noise refers to addition of white and black dots in the image. Though this may seem unnecessary, it is important to remember that a general user who is taking image to feed into your network may not be a professional photographer. His camera can produce blurry images with lots of white and black dots. This augmentation aides the above mentioned users.

2) Translation -

   We would like our network to recognize the object present in any part of the image. Also, the object can be present partially in the corner or edges of the image. For this reason, we shift the object to various parts of the image. This may also result in addition of a background noise. The code snippet shows translating the image at four sides retaining 80 percent of the base image.

3) Flipping -

   This scenario is more important for network to remove biasness of assuming certain features of the object is available in only a particular side. Consider the case shown in image example. You don't want network to learn that tilt of banana happens only in right side as observed in the base image. Also notice that flipping produces different set of images from rotation at multiple of 90 degrees.My additional question is has anyone done some study on what is the maximum number of classes it gives good performance. Consider, data can be generated with good amount of diversity for each class and time of training is not a factor.

2. Question 2 -

2.1. Activation Functions -

2.1.1. Sigmoid -

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

Pros -

1) It is non linear
2) It gives an analog like activation function
3) It gives a smooth gradient
4) Values of this function always lie in the range of (0, 1) thus the activations will never blow up.

Cons -

1) It has the problem of “Vanishing gradients”
2) Sigmoids have slow convergence.
3) Sigmoids saturate and kill gradients.
4) Secondly , its output isn’t zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.

2.1.2. Tanh -

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

Pros -

1) It is also non linear
2) Bounded within the range (-1, 1) hence it will never blow up
3) Gradient is stronger for tanh than sigmoid (derivatives are steeper)

Cons -

1) It also suffers from Vanishing gradient problem and solution to this problem is to use ReLU function.

2.2. Results -

Dermatology

Epocs	Accuracy	
	Activation Function	
	Sigmoid	Tanh
5	47.91	48.75
10	58.33	50.00
20	70.83	47.91
30	62.50	52.91
40	52.08	45.00

Stopping Condition
(1) Using number of Epochs to stop training (2) Can also set threshold on the Error

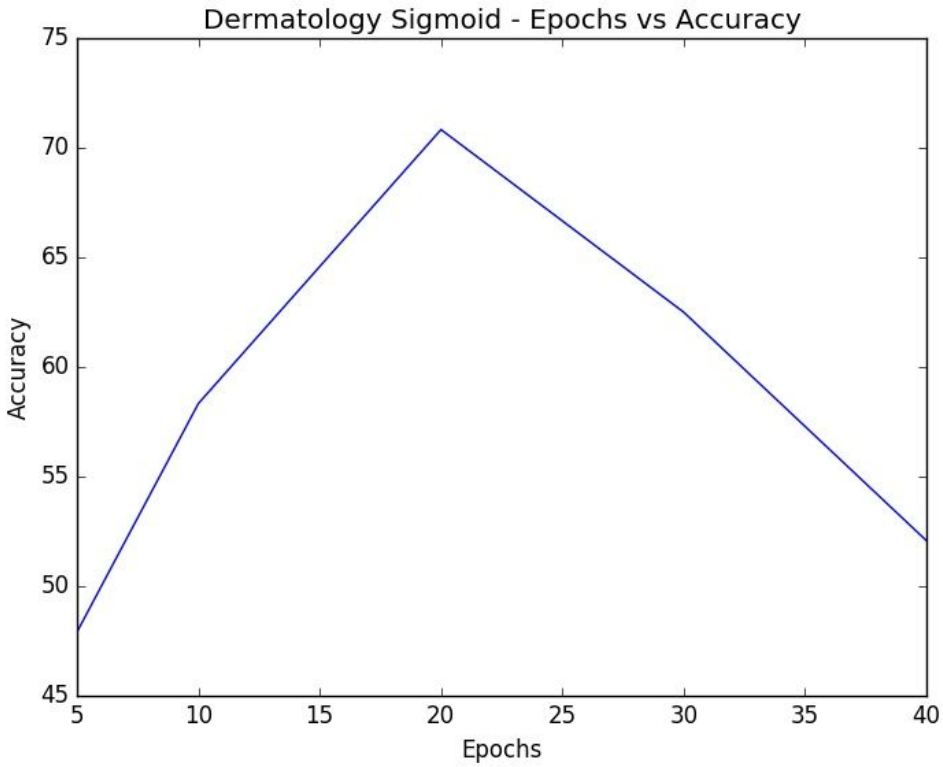
PenDigit

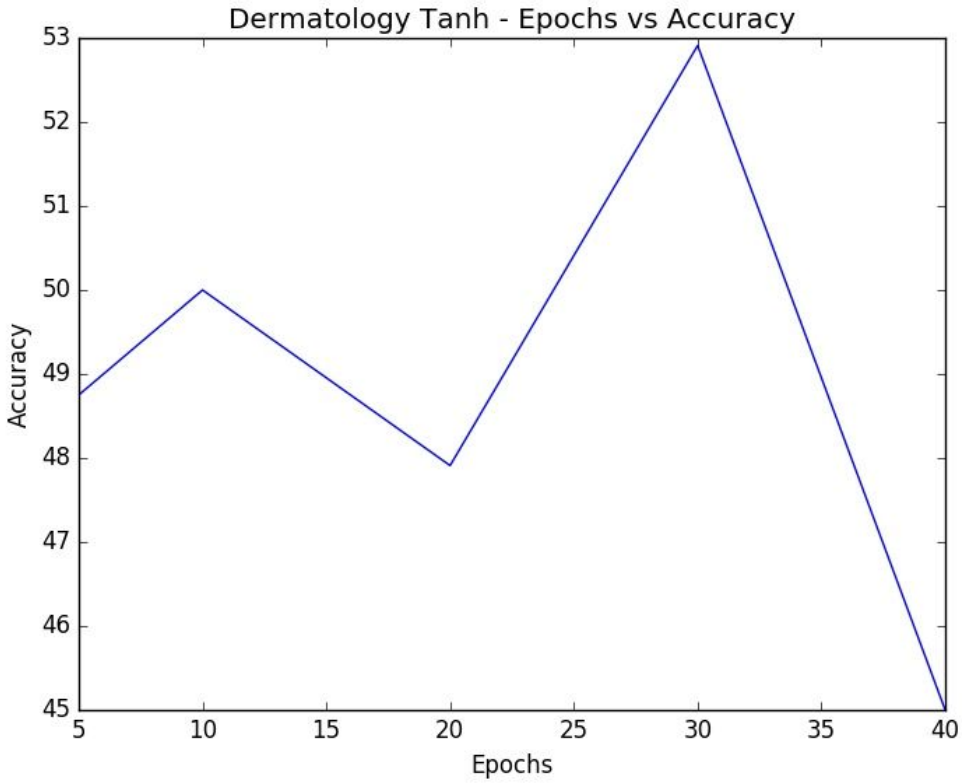
Epochs	Accuracy	
	Activation Function	
	Sigmoid	Tanh
5	27.12	27.68
10	28.66	35.38
20	24.38	32.66
30	25.51	24.60
40	24.73	25.29

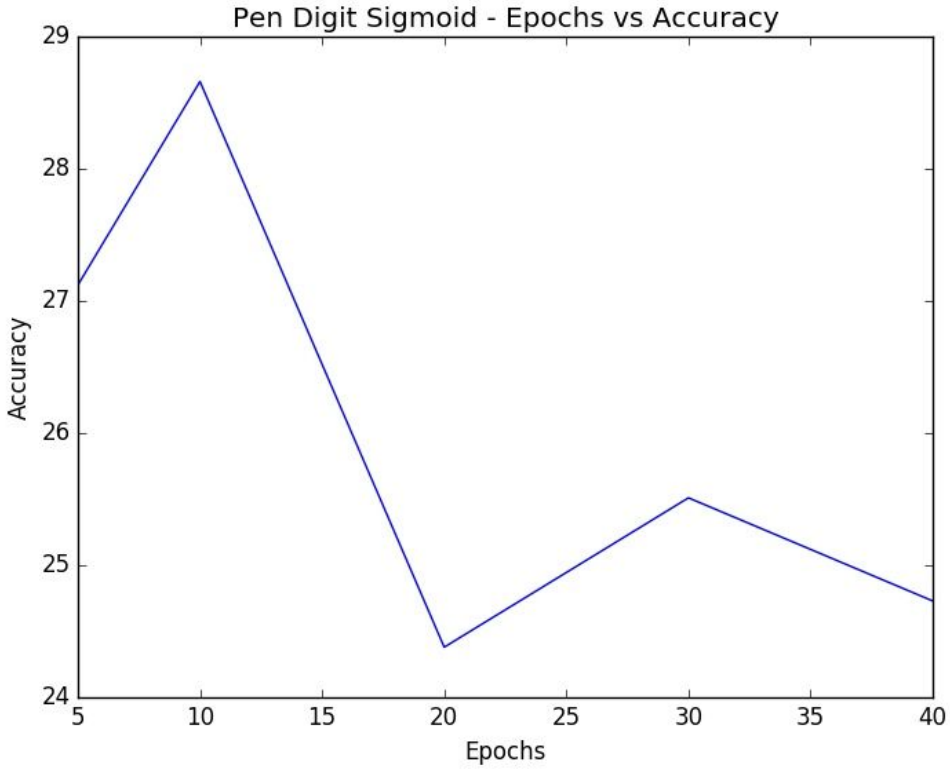
Stopping Condition
(1) Using number of Epochs to stop training (2) Can also set threshold on the Error

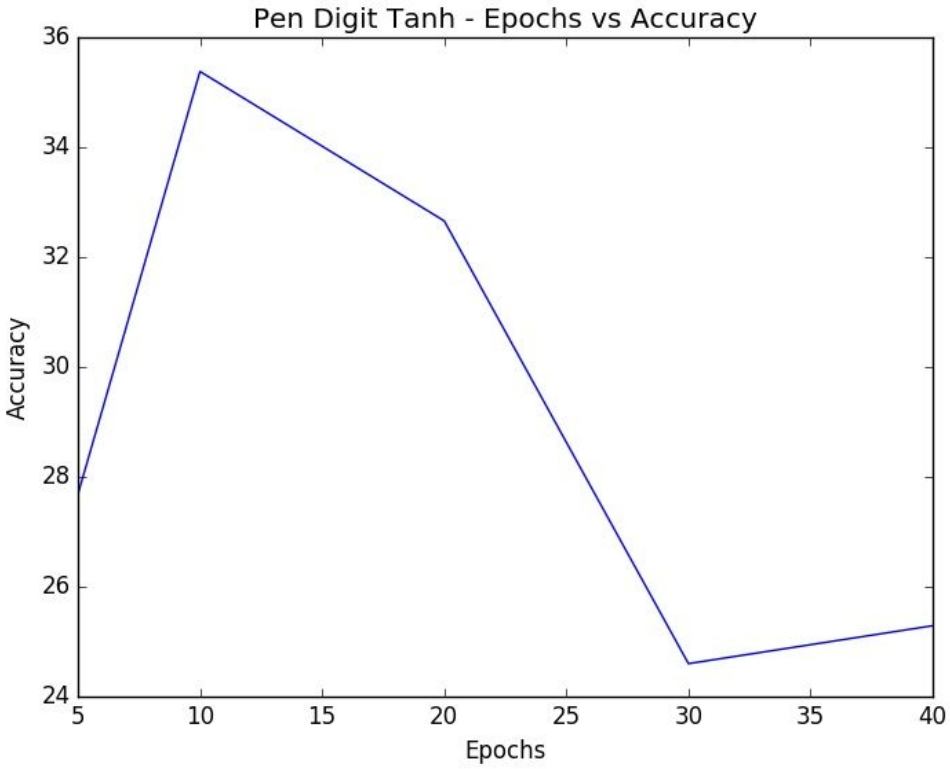
The following graphs have the trend that the the accuracy increases upto a certain number of epochs during training. If the training is done for too many epochs then this results in overfitting and hence the accuracy decreases.

2.3. Graphs -


Dermatology Sigmoid - Epochs vs Accuracy
Accuracy
Epochs
75
70
65
60
55
50
45
5
10
15
20
25
30
35
40


Dermatology Tanh - Epochs vs Accuracy
Accuracy
Epochs
53
52
51
50
49
48
47
46
45
5
10
15
20
25
30
35
40


Pen Digit Sigmoid - Epochs vs Accuracy
Accuracy
Epochs
29
28
27
26
25
24
5
10
15
20
25
30
35
40


Pen Digit Tanh - Epochs vs Accuracy
Accuracy
Epochs
36
34
32
30
28
26
24
5
10
15
20
25
30
35
40