Q1(a):

We have Implemented the model shown below:

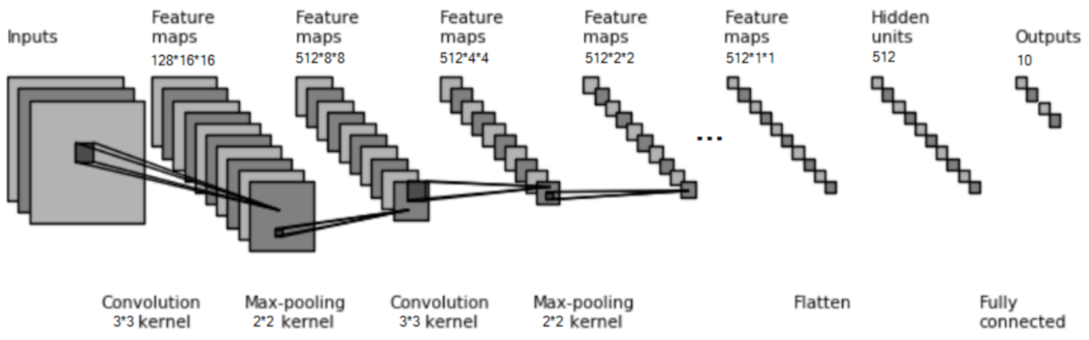
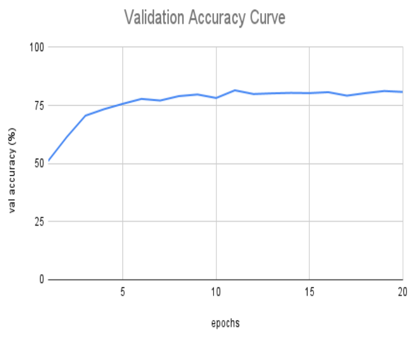
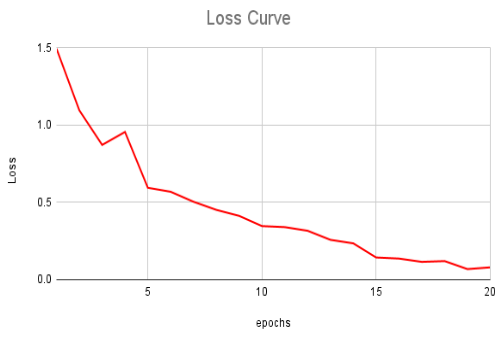
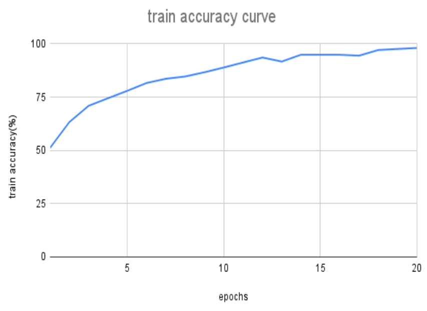


Figure 1: Figure depicting the convolutional neural networks.







After training the model for 20 epochs:
We achieve the best validation accuracy of 81.5% and test accuracy of 79.40% (on 1000 images)

Q1(b):

The model parameters can be broken down as follows:

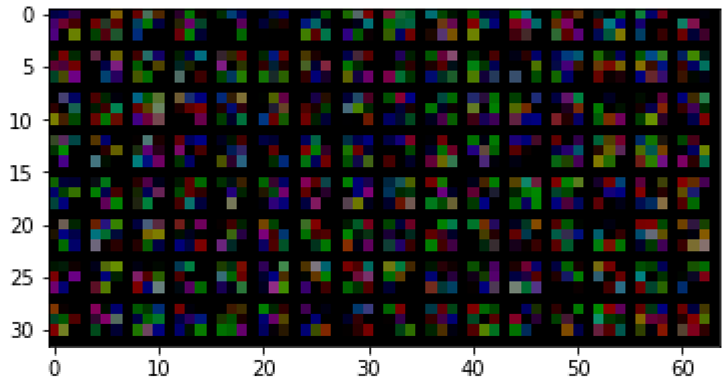
conv2d
#params: 3584
conv2d
#params: 590336
conv2d
#params: 2359808
conv2d
#params: 2359808

conv2d
#params: 2359808
Fully Connected Layer:
#params: 5130

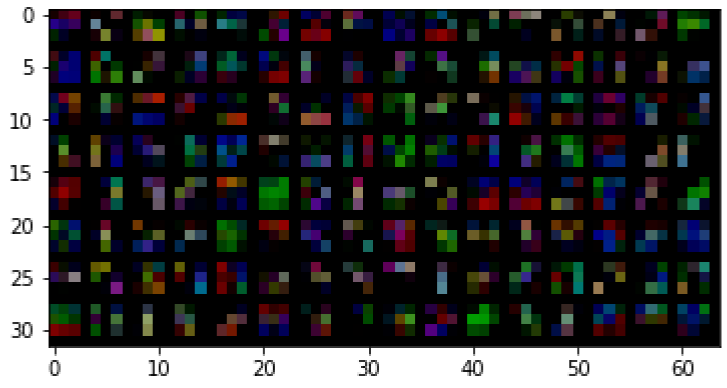
Giving a total of 7678474 parameters.

Q1(c):

Filter Weights Before training:



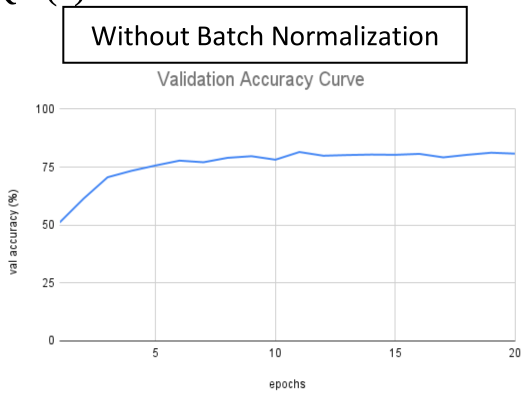
Filter weights after training:

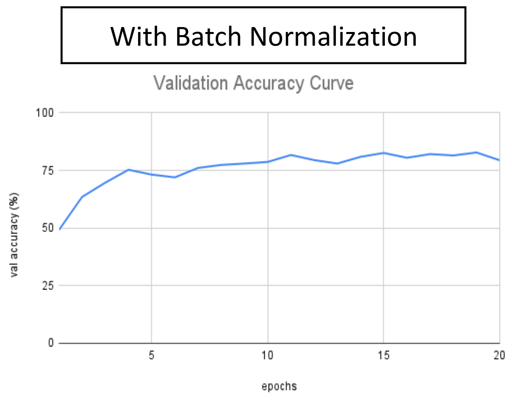


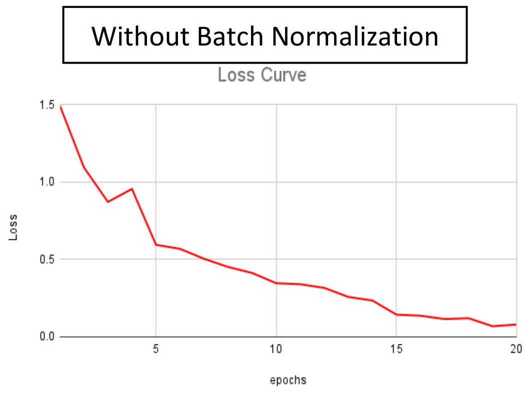
Observations:

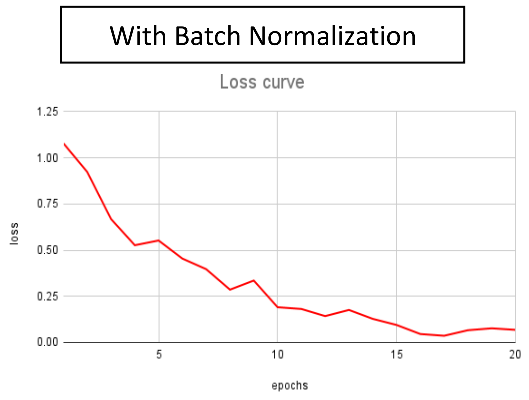
While visualizing the trained filters, we can see patterns like horizontal and vertical lines which usually are characteristics of filters that are used to extract basic structural information like edges on an image.

Q2 (a):





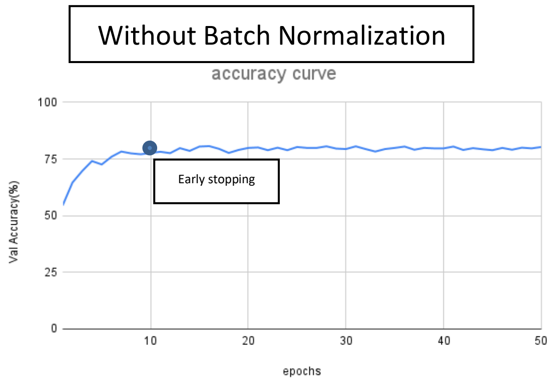


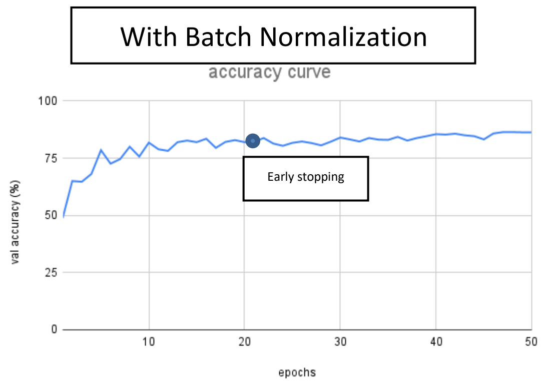


On applying Batch Normalization to the model implemented earlier we see a faster convergence in the training. The model with Batch Normalisation reached a loss value of 0.286 in just 7 epochs while it took the other model almost 15 epochs. Also after 20 epochs of training it reached a higher best validation accuracy of 82.8% in comparison to 81.5% of the model without Batch Normalisation. Also Batch normalization helps the weights to not explode in the training process.

Q2 (b):

For this experiment we have kept the range of shift in accuracy percentage, delta = 1% and a patience of 3.





Models	Test accuracy without early stopping	Test accuracy with early stopping
Without batch normalization	79.9	77.2
With batch normalization	85.8	81.1

It also stops the model from overfitting as we can see both of our scenarios.

Q3 (a):

We have applied torchvision.transforms.RandomHorizontalFlip() and torchvision.transforms.ColorJitter() as our augmentation transforms and have tested it with our batch normalizationed model.

The following represents the results obtained with different variations in the parameters.

Variation with probability parameter of torchvision.transforms.RandomHorizontalFlip() and brightness parameter of torchvision.transforms.ColorJitter() :

(Keeping the other parameters of torchvision.transforms.ColorJitter() constant at 0.5)

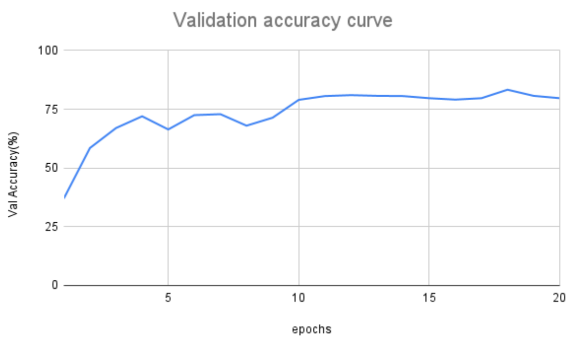
p	Brightness	Test accuracy
0.5	0.5	81.8%
0.7	0.5	81.0%
0.5	0.7	80.3%
0.7	**0.7**	**81.9%**

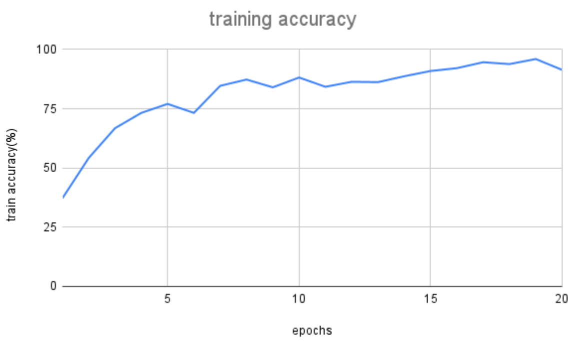
Thus we can conclude that for p=0.7 and brightness=0.7 we get the best results

Q3 (b):

At dropout of 10%:

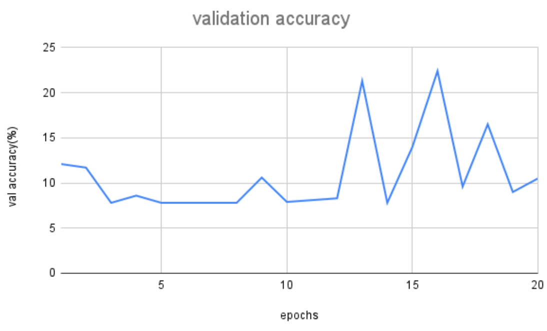
After training the model for 20 epochs we achieved the best validation accuracy of 83.3%, the best training accuracy of 95.97% and a test accuracy of 82.3%

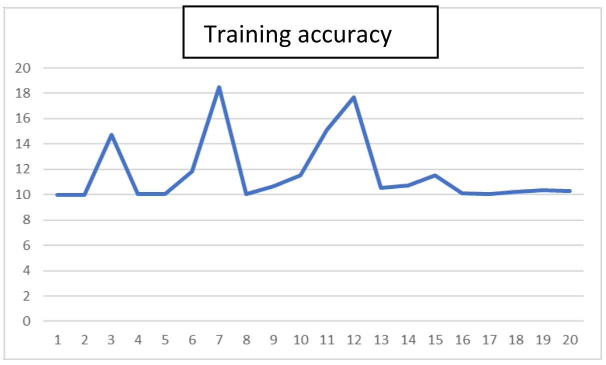




At dropout of 50%

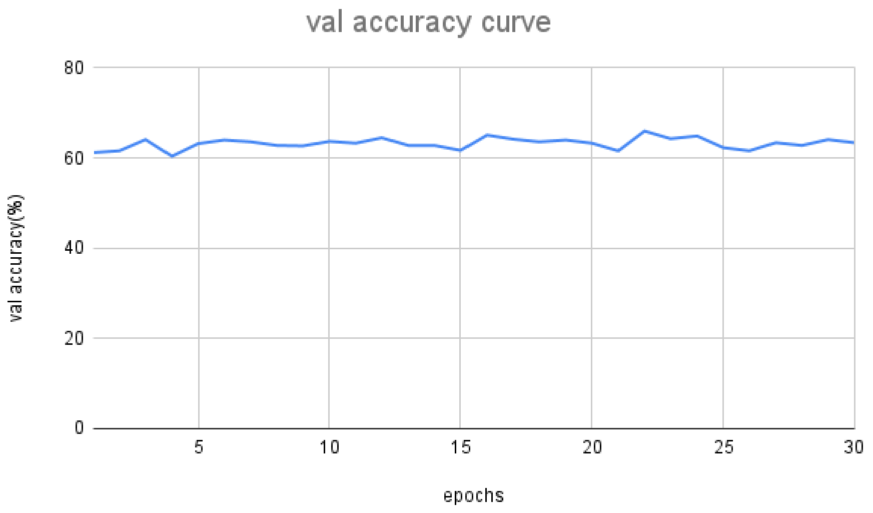
After training the model for 20 epochs we achieved the best validation accuracy of 13.0%, the best training accuracy of 18.5% and a test accuracy of 9.1%





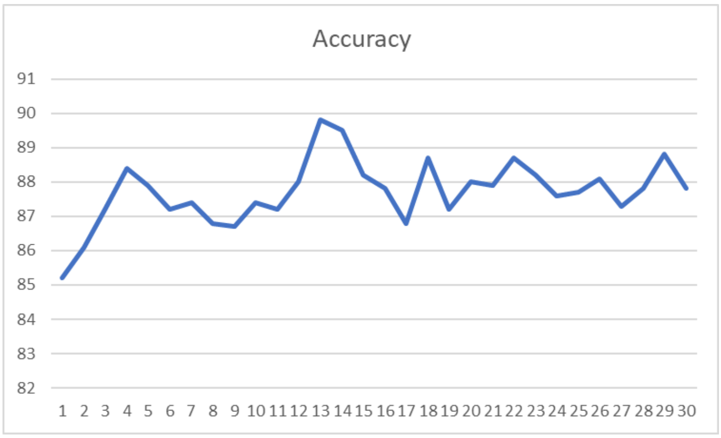
Q4(a):

With fine tune and pretrained true, the model will use the previous vgg weights but donot update it however the newly added layers gets updated while training. The initial validation accuracy starts from 61.2% and the best validation accuracy after training for 30 epochs is 65.1% and test accuracy of 62%%

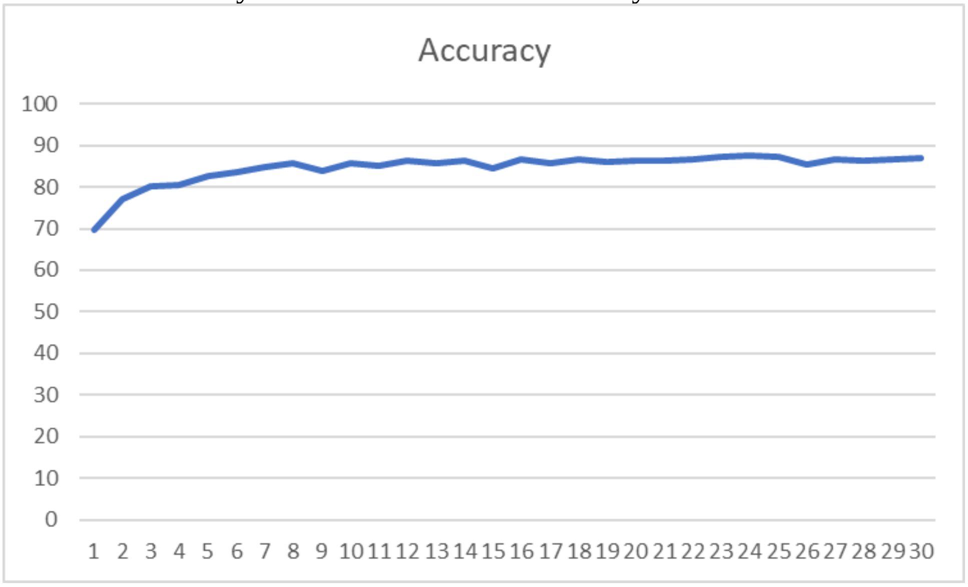


Q4(b):

With fine tune false and pretrained true, the model will use the previous vgg weights and update it along with the newly added layers while training. The initial validation accuracy starts from 85.2% and the best validation accuracy after training for 30 epochs is 89.8% and test accuracy of 87.9%



With fine tune false and pretrained false, the model will randomly assign the weights to vgg layers and newly added layers. The model has to update the vgg weights and also newly added layers weights. The initial validation accuracy starts from 69.8 percent and the best validation accuracy is 87.7% and the test accuracy is 85.1%.


Accuracy