ELL793 REPORT

ASSIGNMENT 2(CONVOLUTION NEURAL NETWORKS)

SARANSH AGARWAL (2019MT60763)
MANISH BORTHAKUR (2019MT60493)

INTRODUCTION

Neural Networks and related Deep Learning techniques are state of the art methods used today for a wide variety of applications in fields such as Computer Vision, NLP, Finance, Diagnostics, etc. In this assignment, consisting of two parts, we use their power to classify the MNIST handwritten digit dataset and the CIFAR-10 dataset.

PART 1

Train VGG-16 CNN from scratch using MNIST dataset. The MNIST handwritten digit dataset consist of 28x28 grayscale images. It is made up of about 60,000 images for the purpose of training and about 10,000 for the purpose of validation.

The input to VGG16 is 224x224 rgb image. We have resized the MNIST image from 28x28 grayscale to 224x224 rgb image.

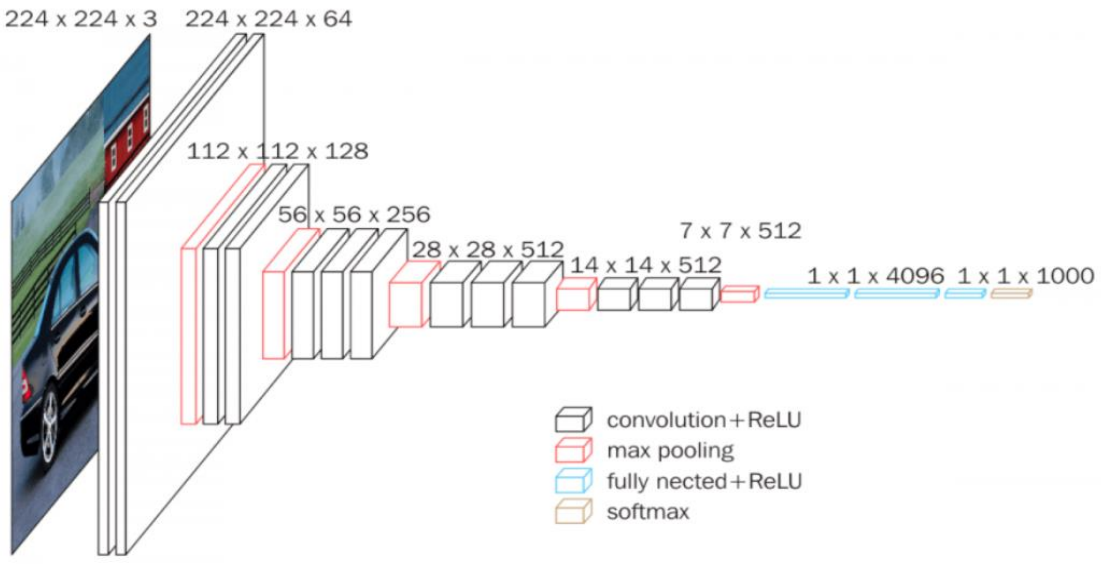


Figure 1: VGG16 architecture

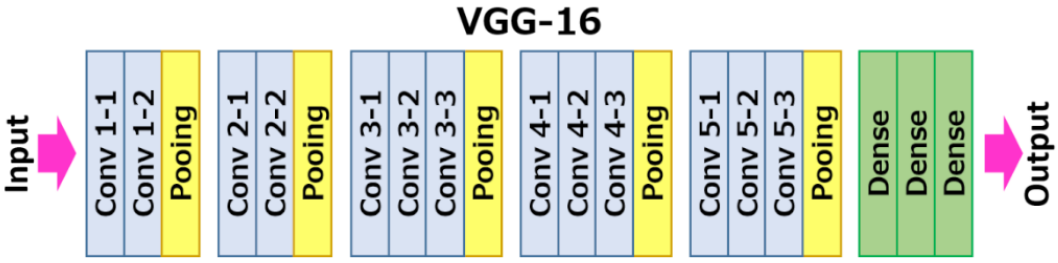


Figure 2: VGG16 model

```
class VGG16(nn.Module):
    def __init__(self, num_classes, act_func):
        super(VGG16, self).__init__()
        self.layer1 = nn.Sequential(nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(64),act_func)
        self.layer2 = nn.Sequential(nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(64),act_func, nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer3 = nn.Sequential(nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(128),act_func)
        self.layer4 = nn.Sequential(nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(128),act_func,nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer5 = nn.Sequential(nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(256),act_func)
        self.layer6 = nn.Sequential(nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(256),act_func)
        self.layer7 = nn.Sequential(nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(256),act_func,nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer8 = nn.Sequential(nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(512),act_func)
        self.layer9 = nn.Sequential(nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(512),act_func)
        self.layer10 = nn.Sequential(nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(512),act_func,nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer11 = nn.Sequential(nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(512),act_func)
        self.layer12 = nn.Sequential(nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(512),act_func)
        self.layer13 = nn.Sequential(nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),nn.BatchNorm2d(512),act_func,nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.fc = nn.Sequential(nn.Dropout(0.5),nn.Linear(512, 4096),act_func)
        self.fc1 = nn.Sequential(nn.Dropout(0.5),nn.Linear(4096, 4096),act_func)
        self.fc2= nn.Sequential(nn.Linear(4096, num_classes),nn.Softmax())
```

Figure 3: Wrapper code of VGG16 layers

PART 1 (a)

(Which Activation function is better?) Fix the number of epochs (to 50 epochs) and fixate any learning rate, optimizer, batch size, train this network, with layers having:

- only RELU activation,
- only leaky RELU activation,
- only sigmoid activation,
- only tanh activation.

In part a we have varied the activation function and fixed other parameters as follows:

- No of epochs = 50
- Learning rate = 0.005
- Optimizer = SGD
- Batch Size = 1024

ONLY RELU ACTIVATION

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking activation function as RELU.

PLOT OF LOSSES VS EPOCHS

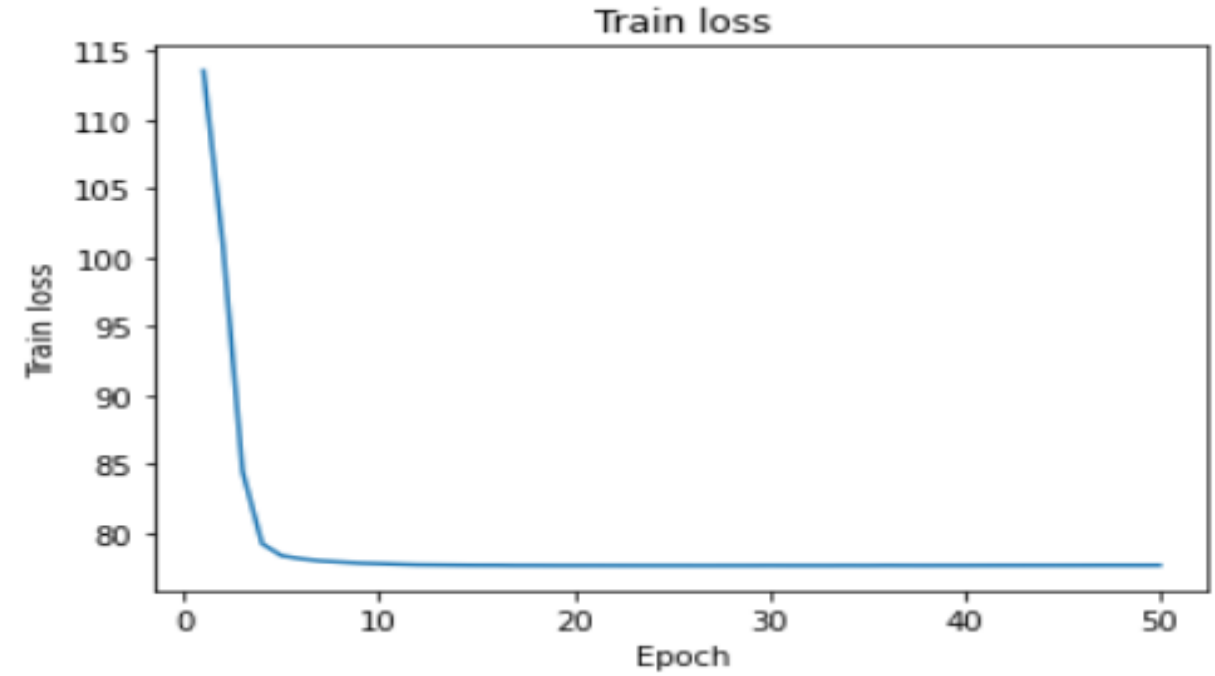


Figure 4: Training loss Vs Epoch

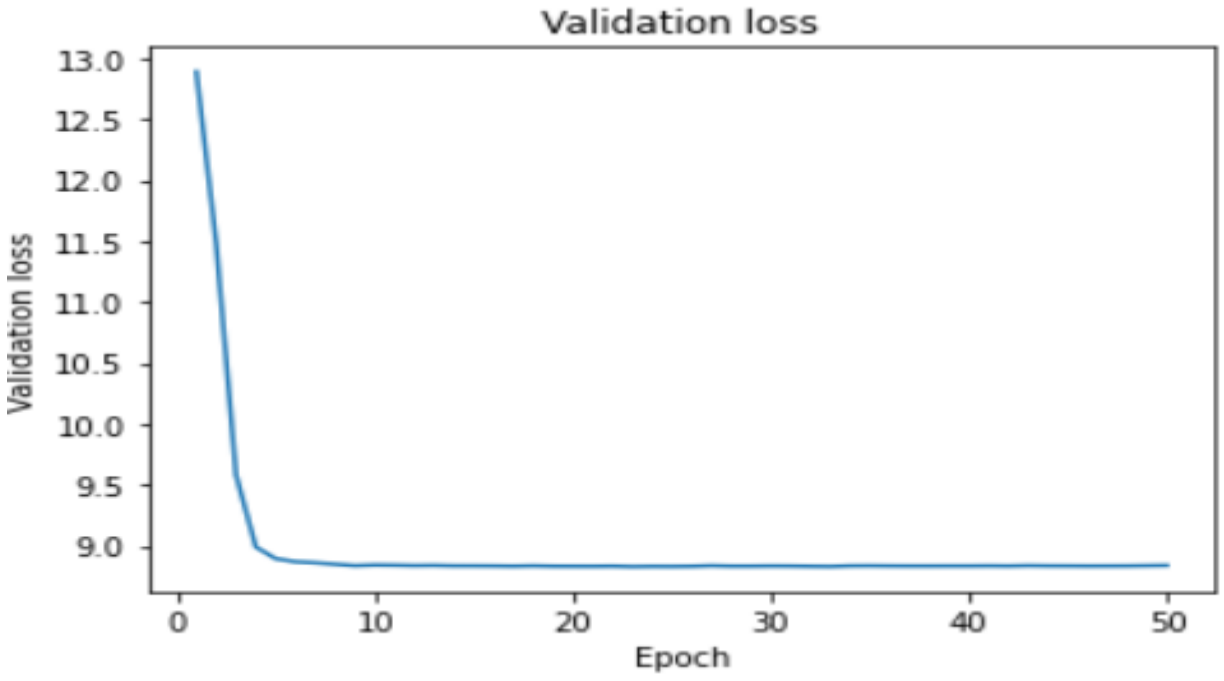


Figure 5: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

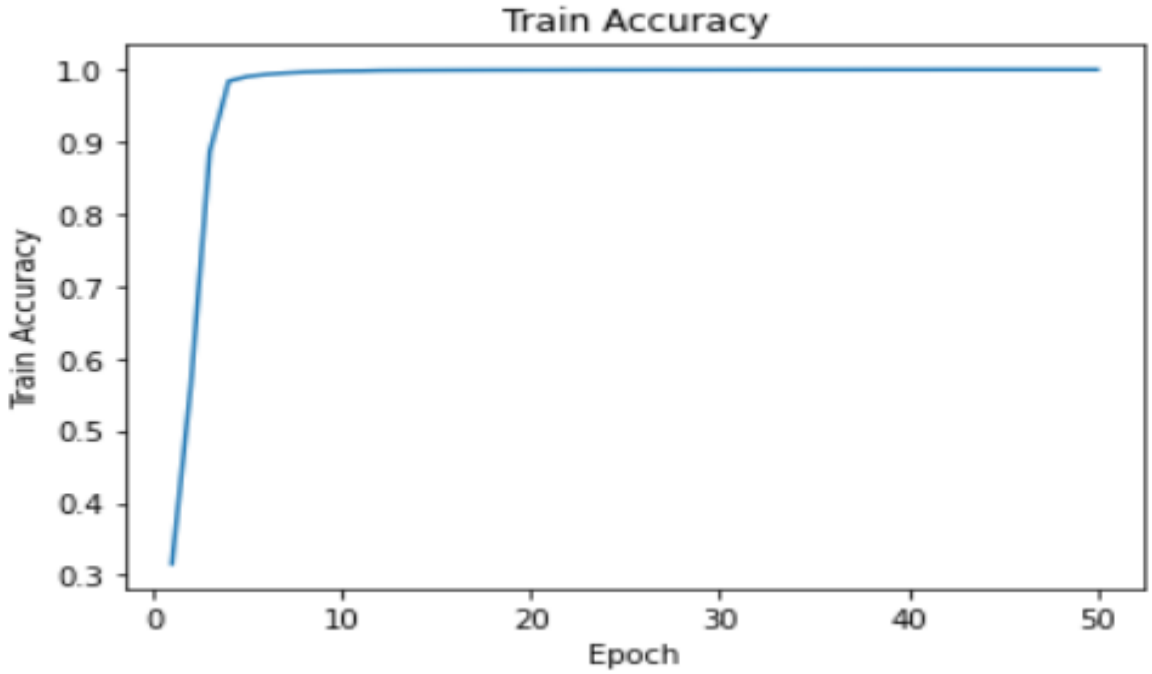


Figure 6: Training Accuracy Vs Epoch

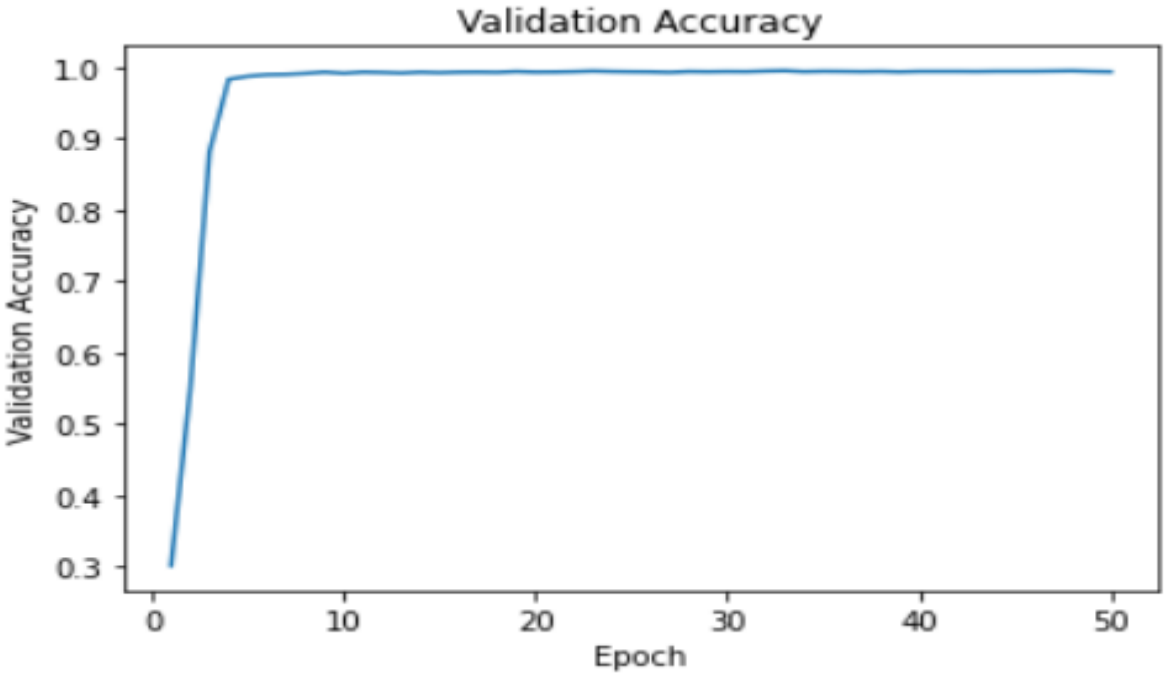


Figure 7: Validation Accuracy Vs Epoch

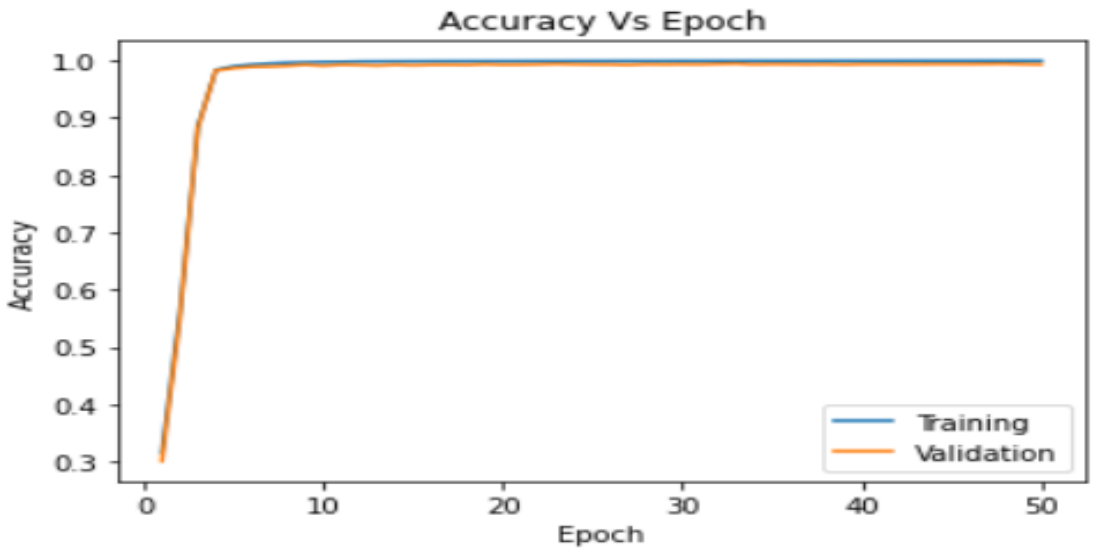


Figure 8: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.71

Testing Accuracy = 99.52%

ONLY LEAKY RELU ACTIVATION

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking activation function as leaky RELU.

PLOT OF LOSSES VS EPOCHS

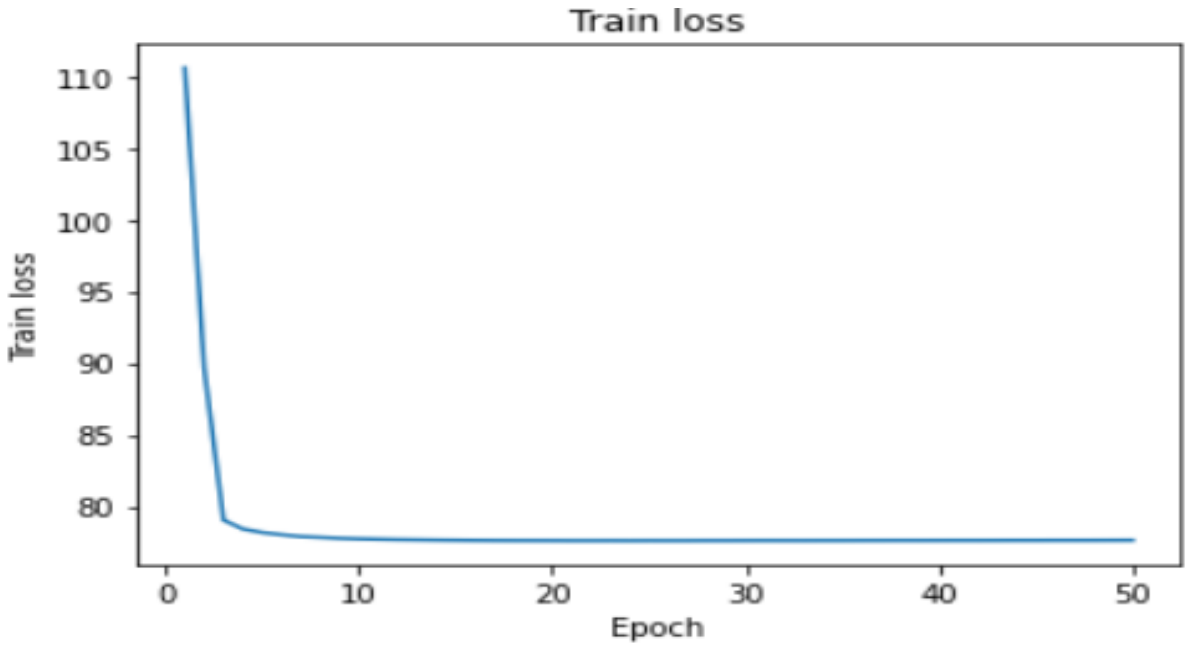


Figure 9: Training loss Vs Epoch

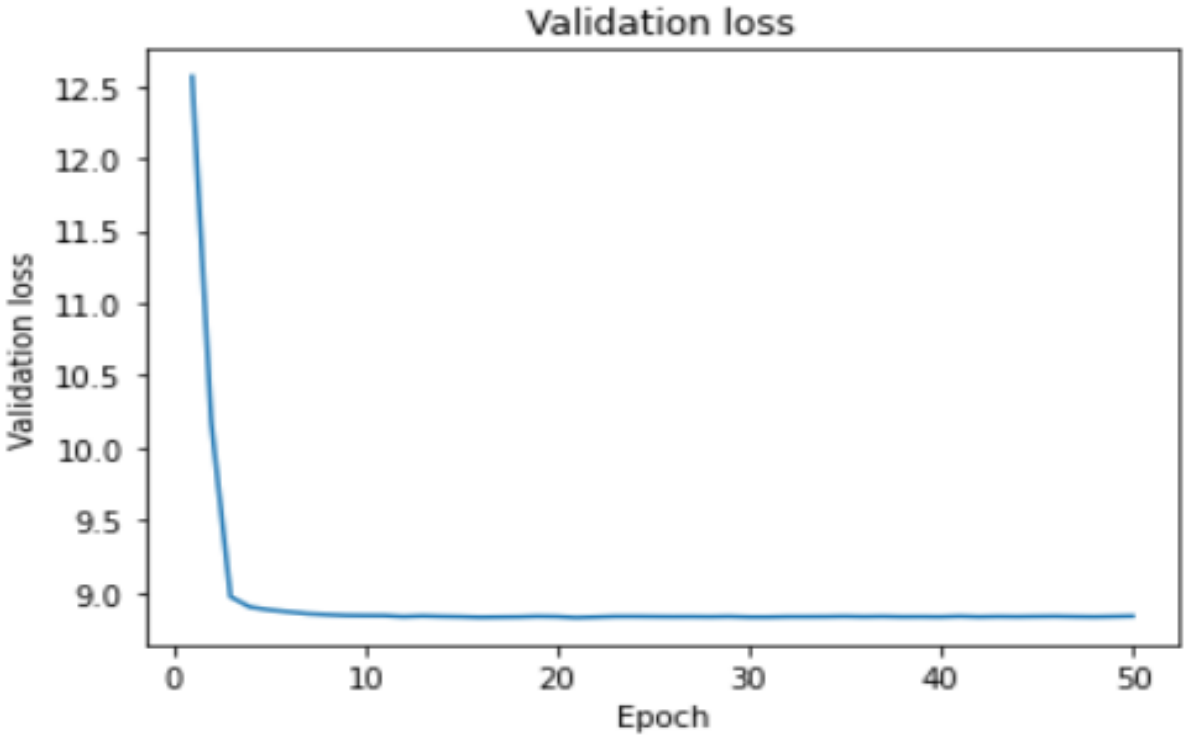


Figure 10: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

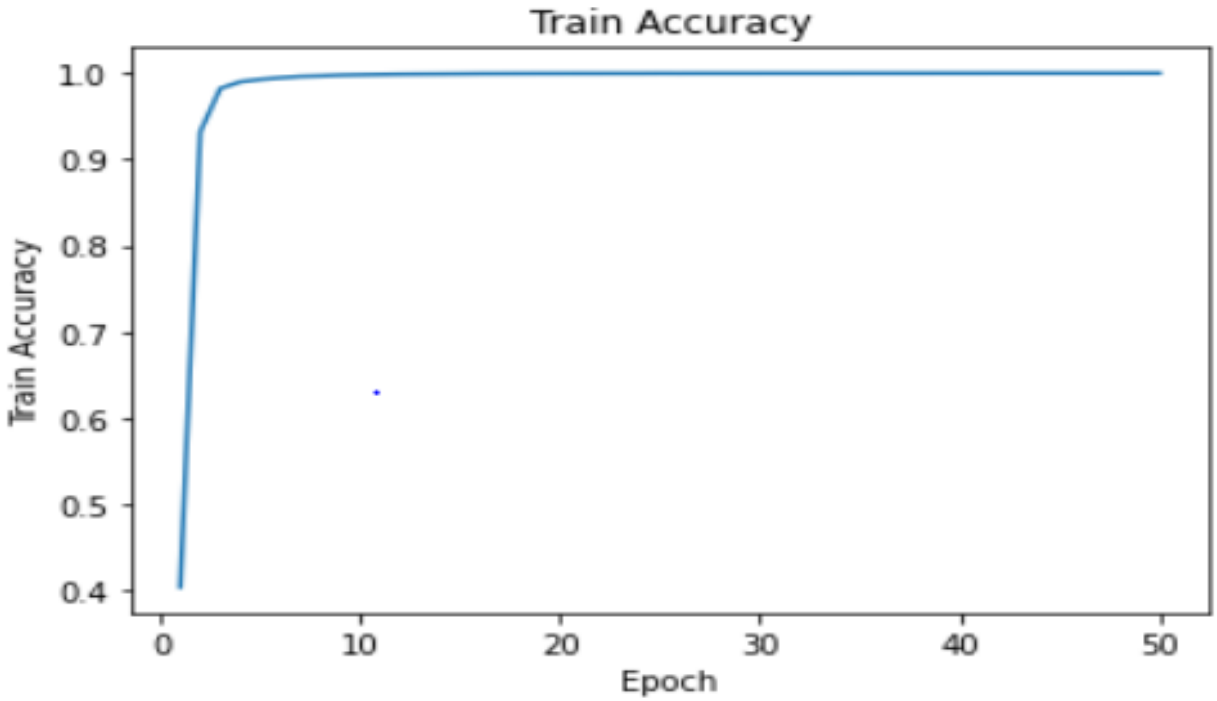


Figure 11: Training Accuracy Vs Epoch

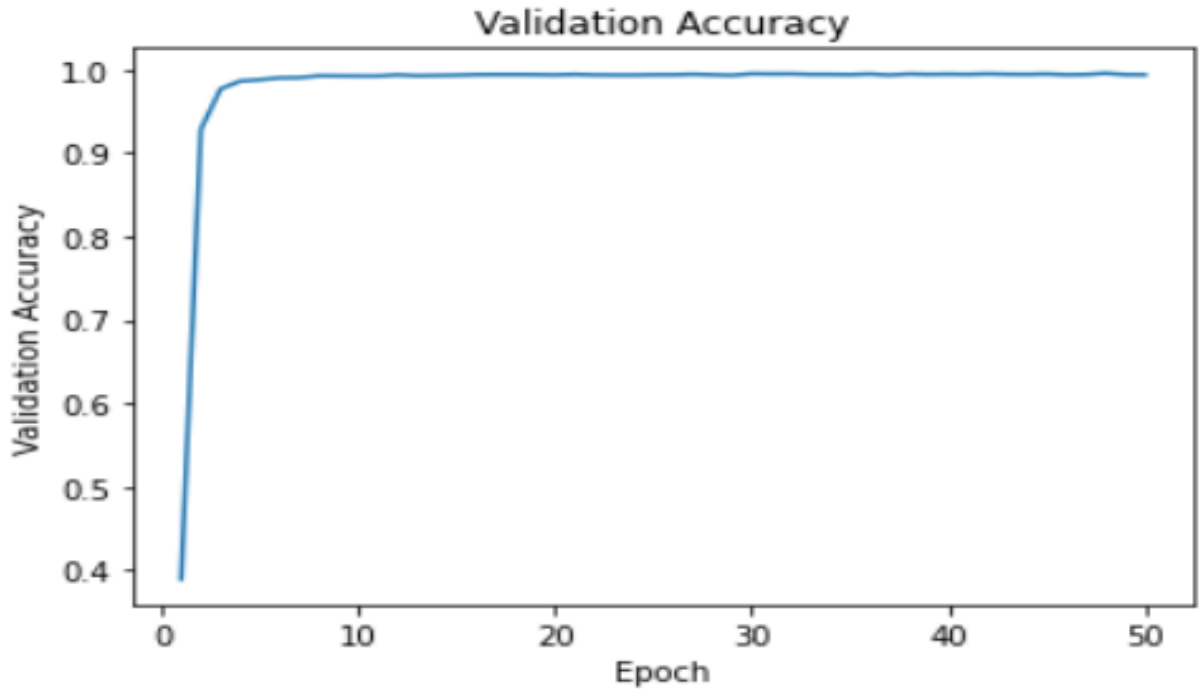


Figure 12: Validation Accuracy Vs Epoch

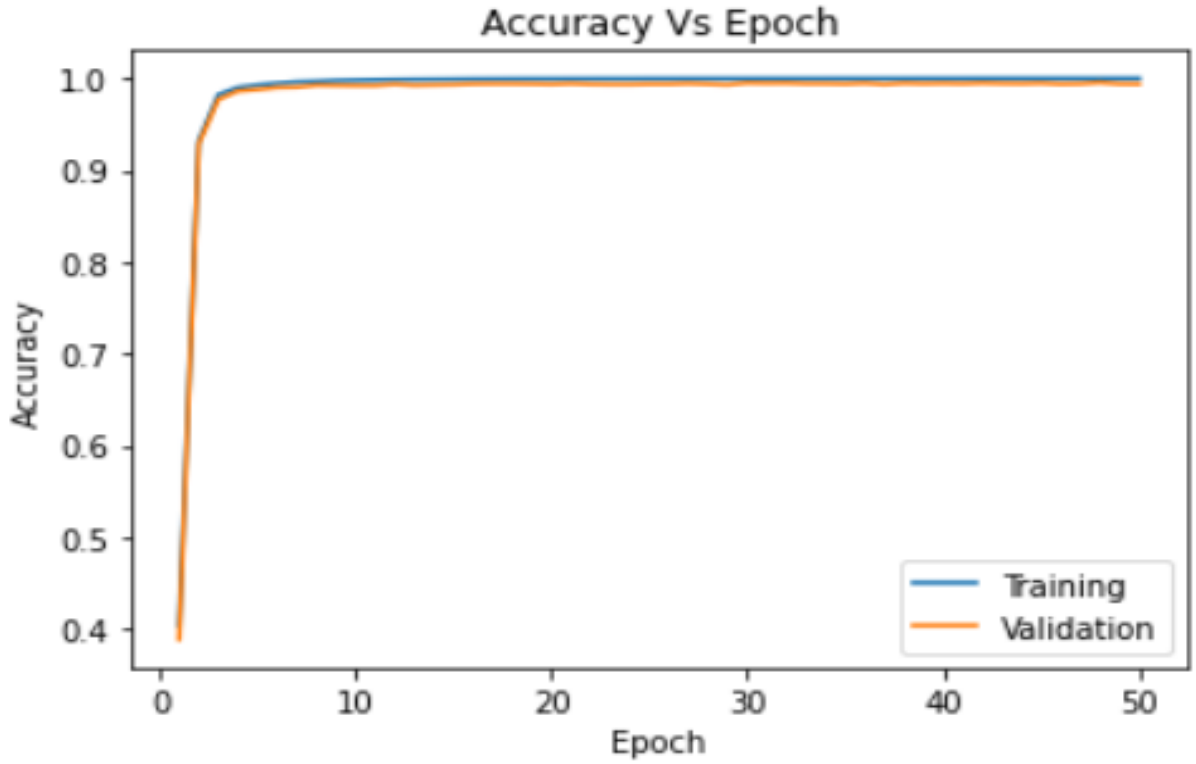


Figure 13: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.71

Testing Accuracy = 99.52%

ONLY SIGMOID ACTIVATION

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking activation function as Sigmoid.

PLOT OF LOSSES VS EPOCHS

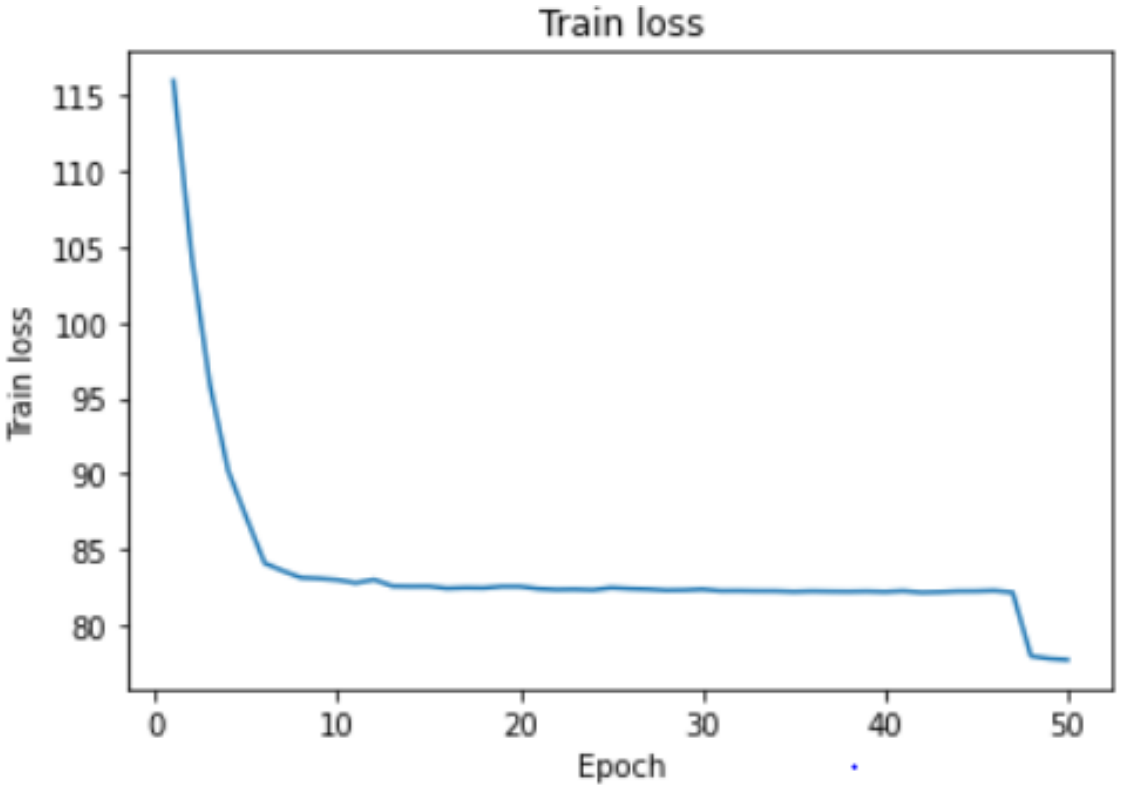


Figure 14: Training loss Vs Epoch

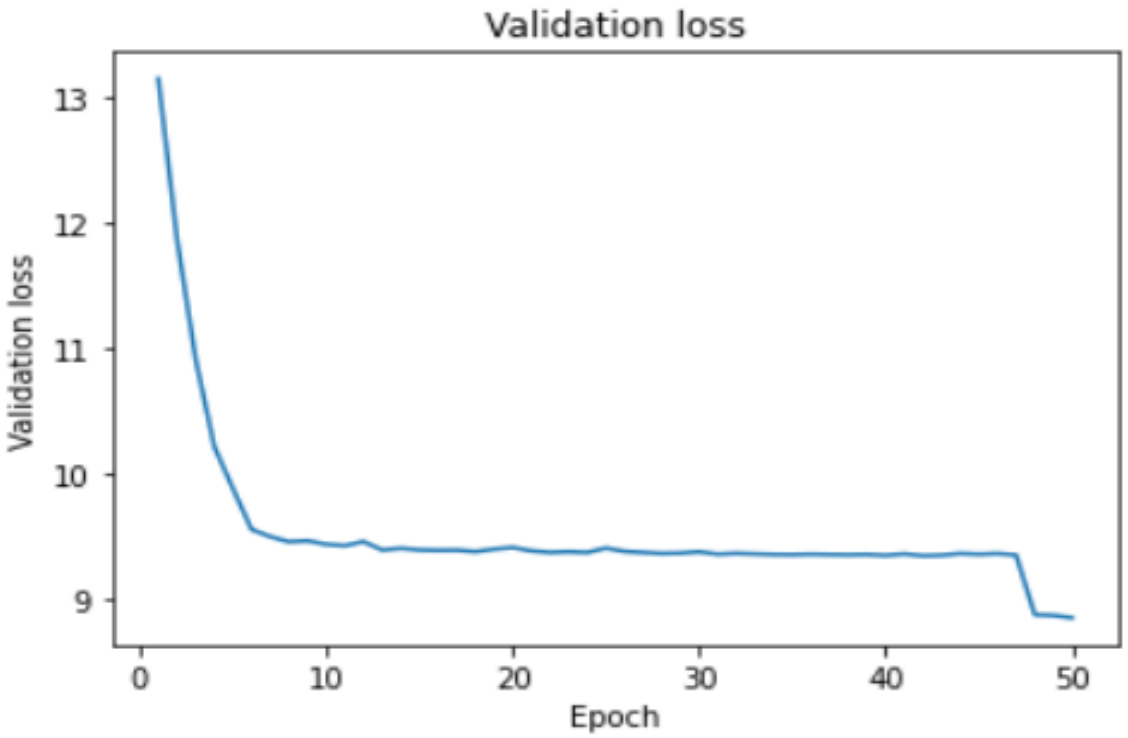


Figure 15: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

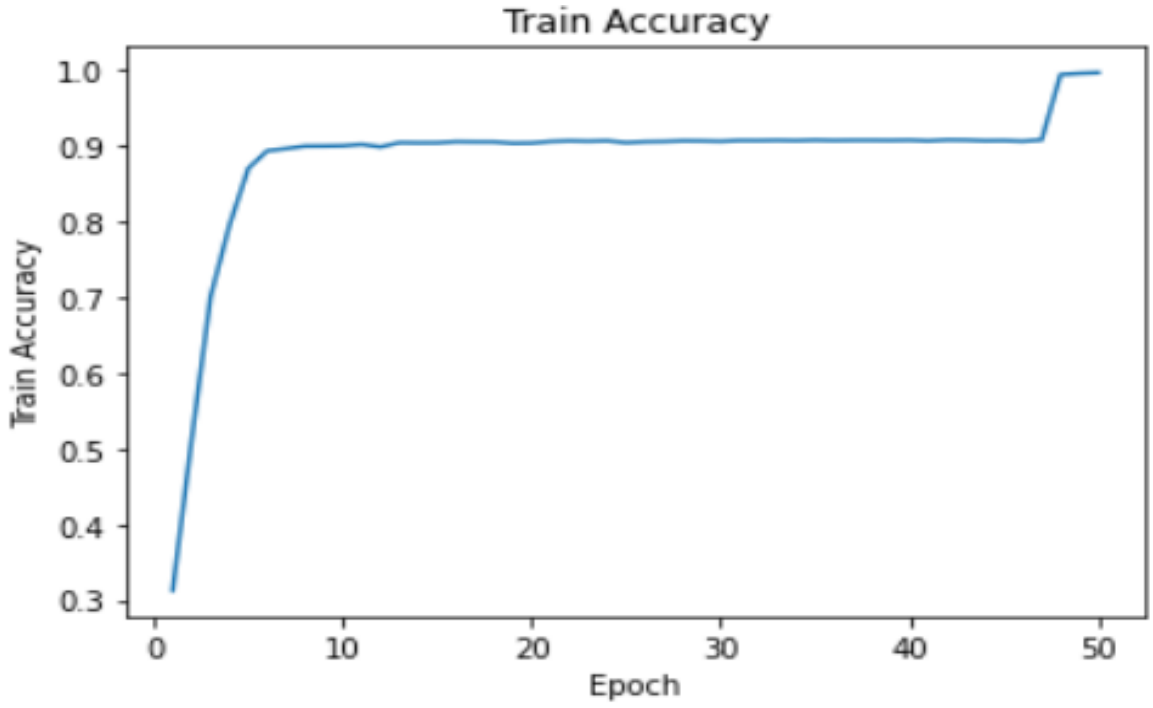


Figure 16: Training Accuracy Vs Epoch

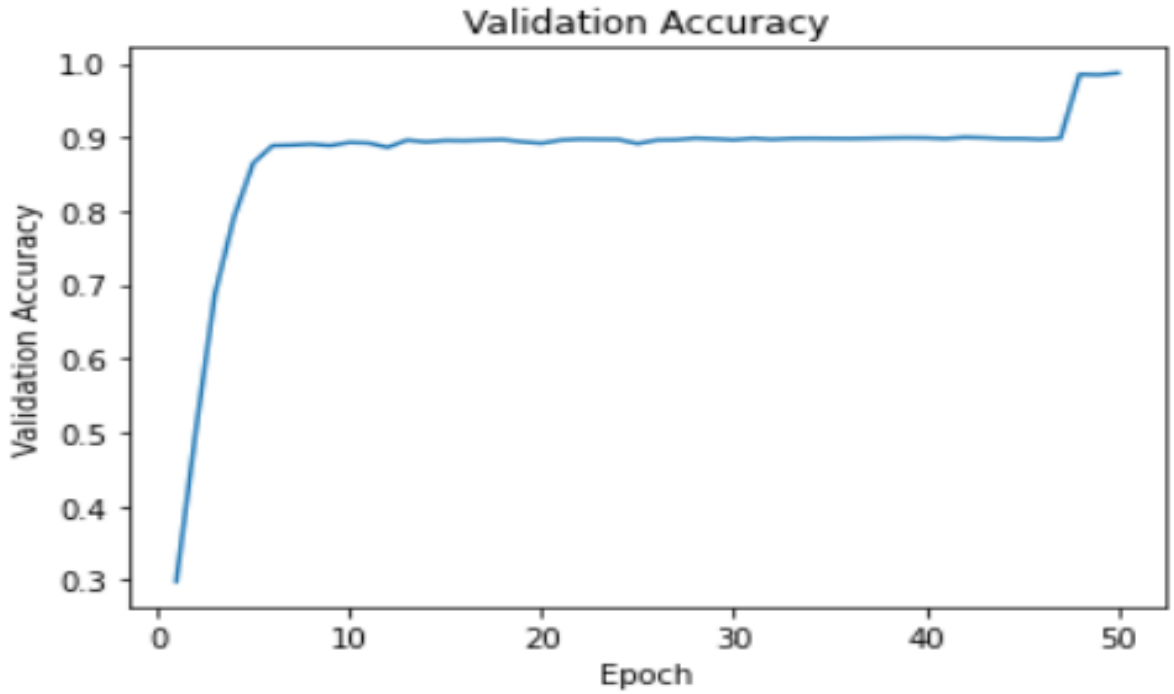


Figure 17: Validation Accuracy Vs Epoch

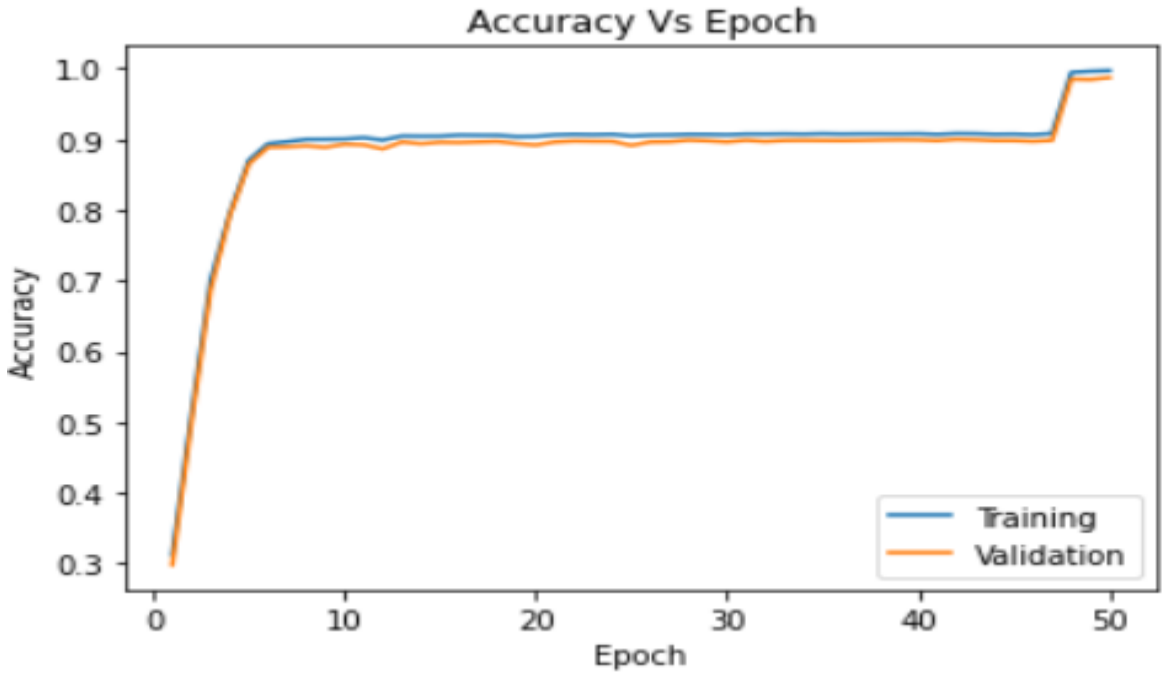


Figure 18: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.76

Testing Accuracy = 98.64%

ONLY TANH ACTIVATION

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking activation function as Tanh.

PLOT OF LOSSES VS EPOCHS

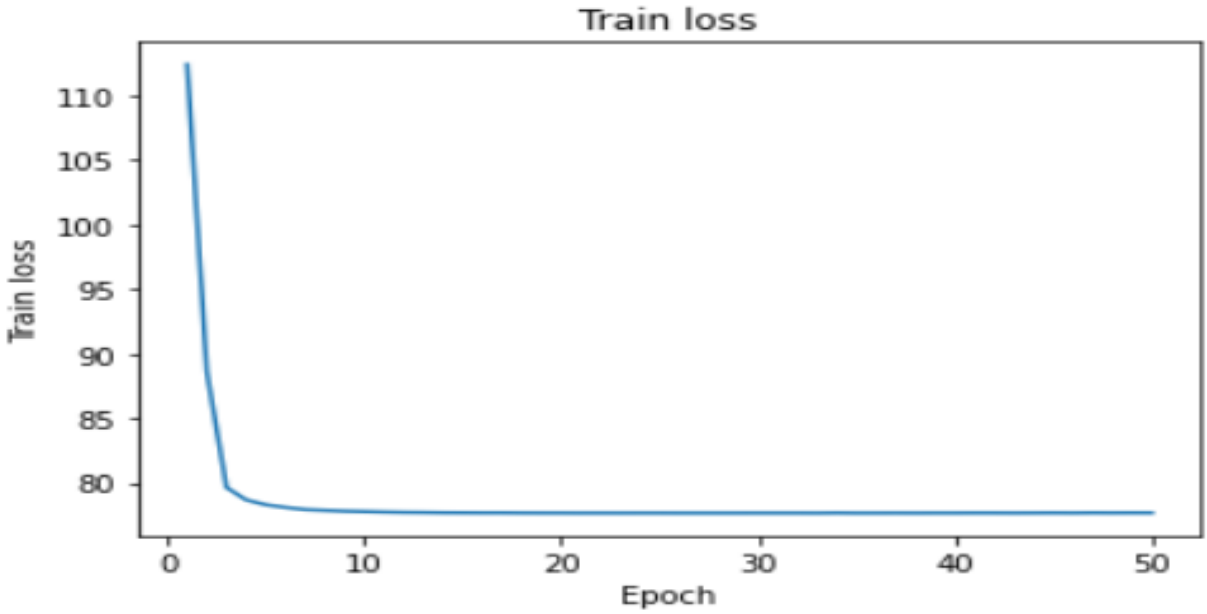


Figure 19: Training loss Vs Epoch

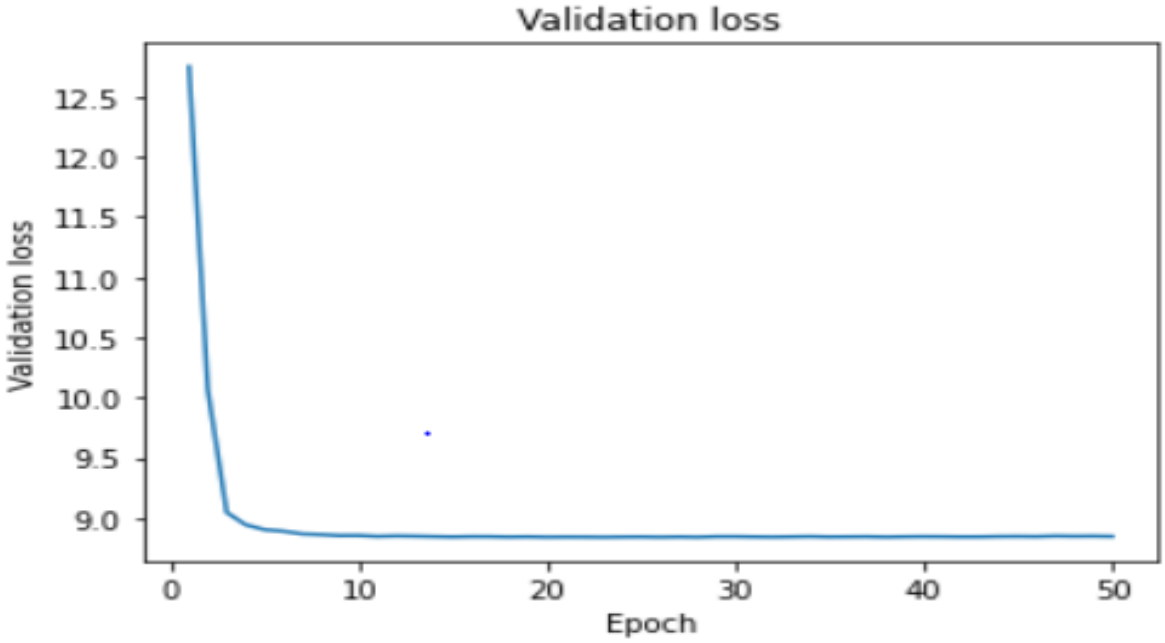


Figure 20: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

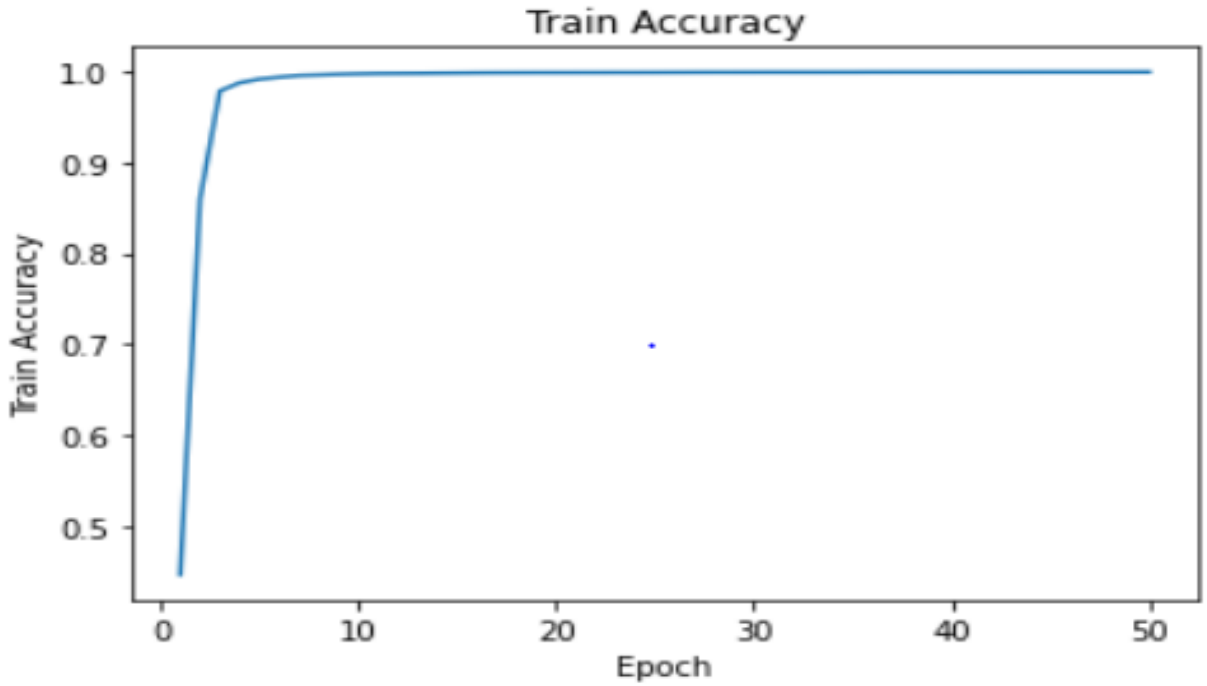


Figure 21: Training Accuracy Vs Epoch

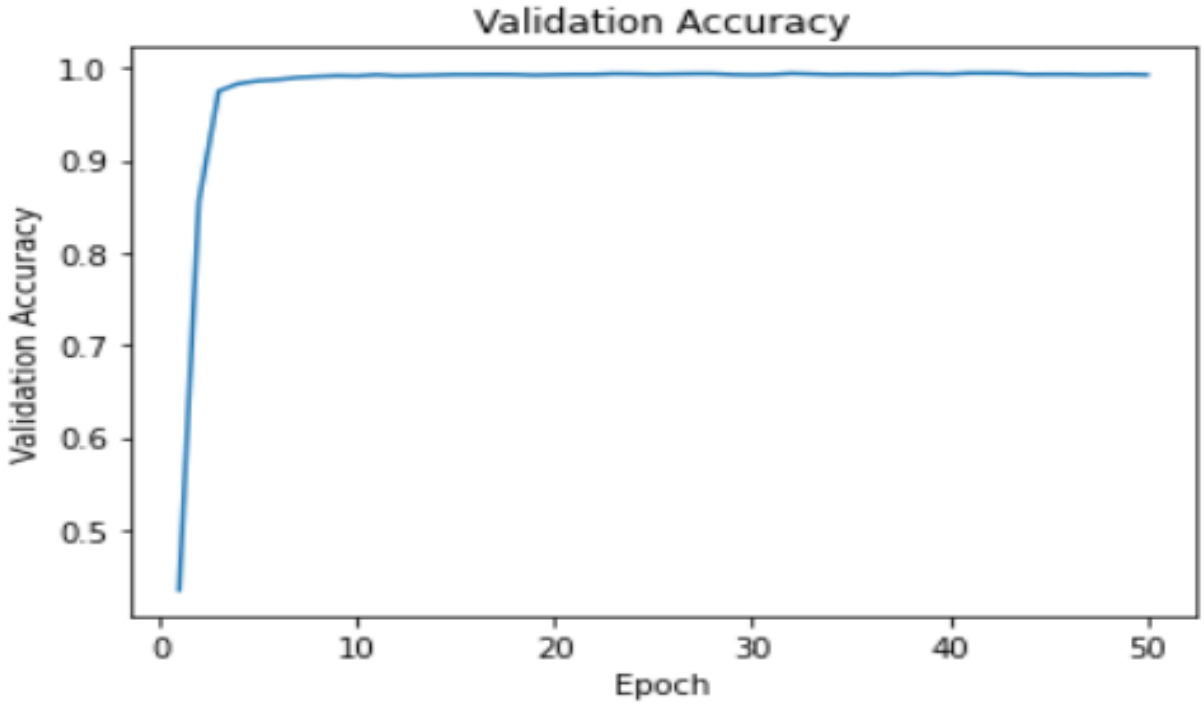


Figure 22: Validation Accuracy Vs Epoch

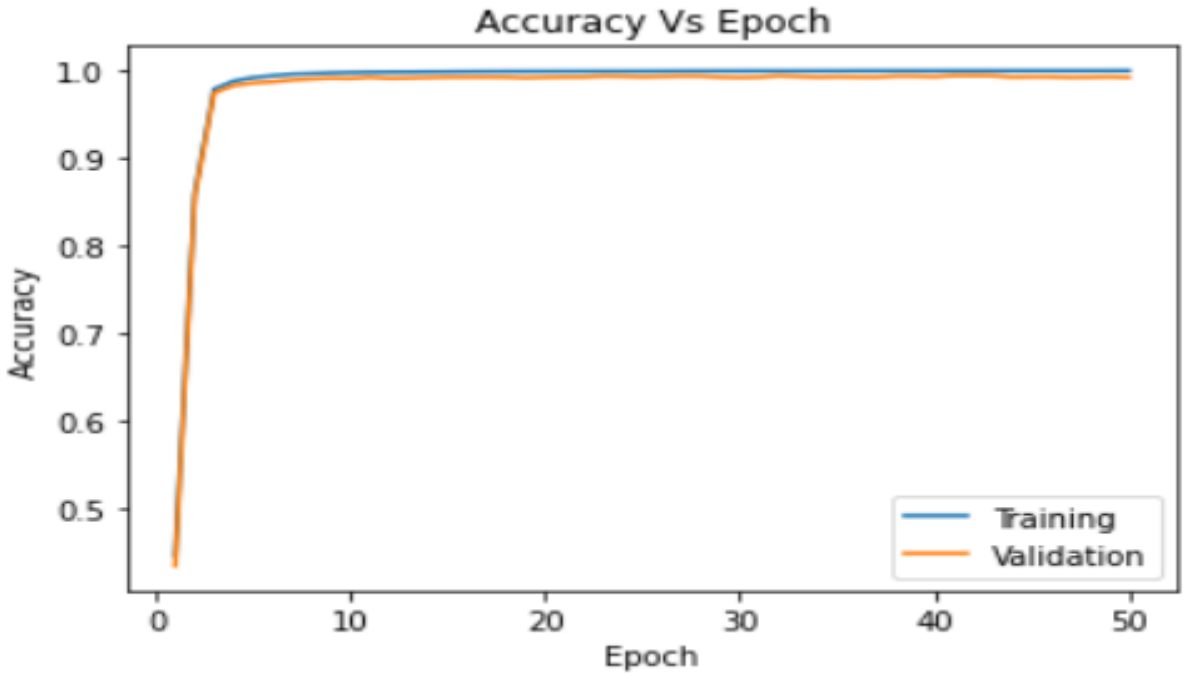


Figure 23: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.74

Testing Accuracy = 99.30%

COMPARISON FOR VARIOUS ACTIVATION FUNCTIONS

Activation	Training Accuracy	Validation Accuracy	Testing Loss	Testing Accuracy
Relu	99.98	99.47	14.71	99.52
Leaky Relu	99.98	99.55	14.71	99.52
Sigmoid	99.71	98.72	14.76	98.64
Tanh	99.97	99.40	14.74	99.30

Figure 24: COMPARISON FOR VARIOUS ACTIVATION FUNCTIONS

CONCLUSION

According to our analysis, Leaky ReLU is the most effective activation function for the validation set compared to the ones mentioned earlier. This is because Leaky ReLU is less susceptible to the vanishing gradient problem than tanh and sigmoid, which often results in slow learning due to activation function gradients becoming too small. Unlike ReLU, Leaky ReLU's small non-zero gradient when the input is negative gives it an edge over ReLU. Moreover, we have observed that the sigmoid activation function takes the longest time to converge due to the significant number of vanishing gradients it faces.

PART 1 (b)

(What learning rate is better?) Fix the optimizer, batch size, and activation function, the number of epochs (to 50 epochs), train this network first using different learning rates (of your choice). In part b we have varied the learning rate and fixed other parameters as follows:

- No of epochs = 50
- Activation Function = RELU
- Optimizer = SGD
- Batch Size = 1024

We have varied learning rate as 0.001, 0.005 and 0.01.

LEARNING RATE AS 0.001

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking learning rate as 0.001.

PLOT OF LOSSES VS EPOCHS

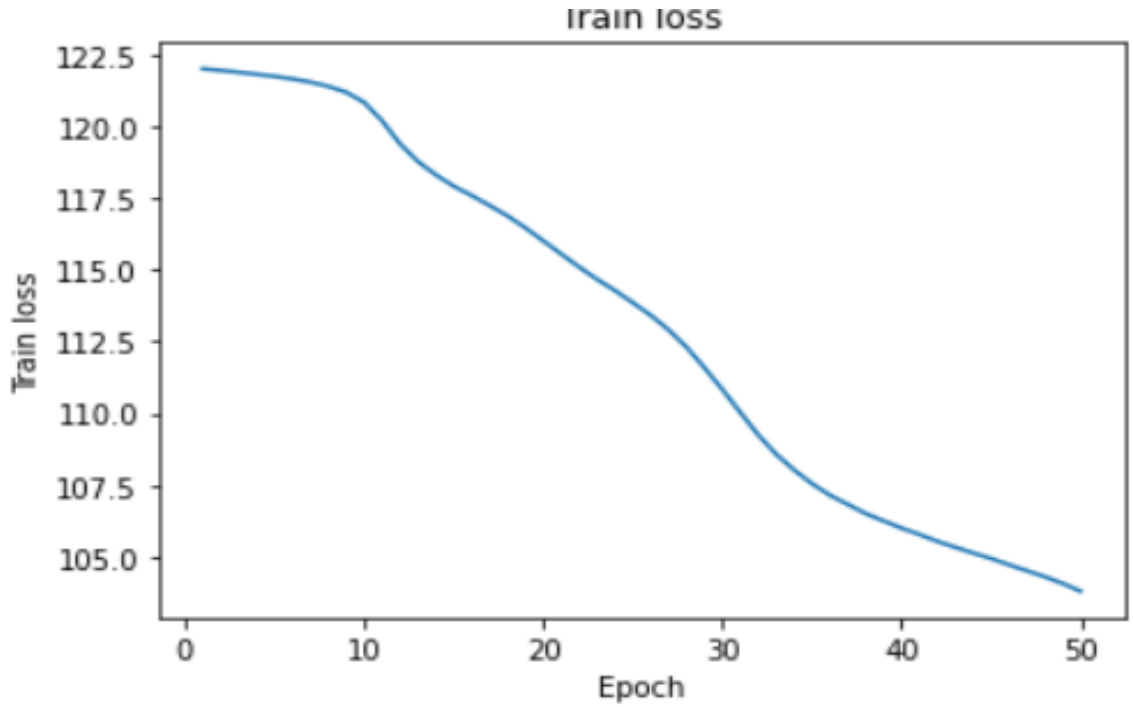


Figure 25: Training loss Vs Epoch

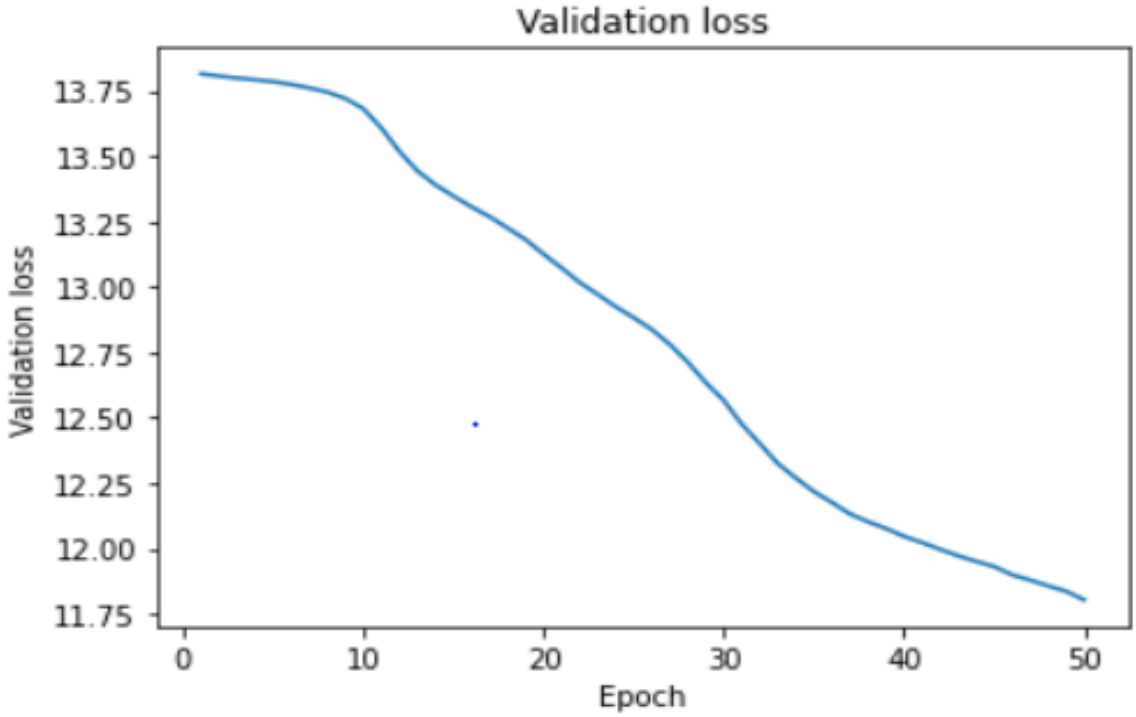


Figure 26: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

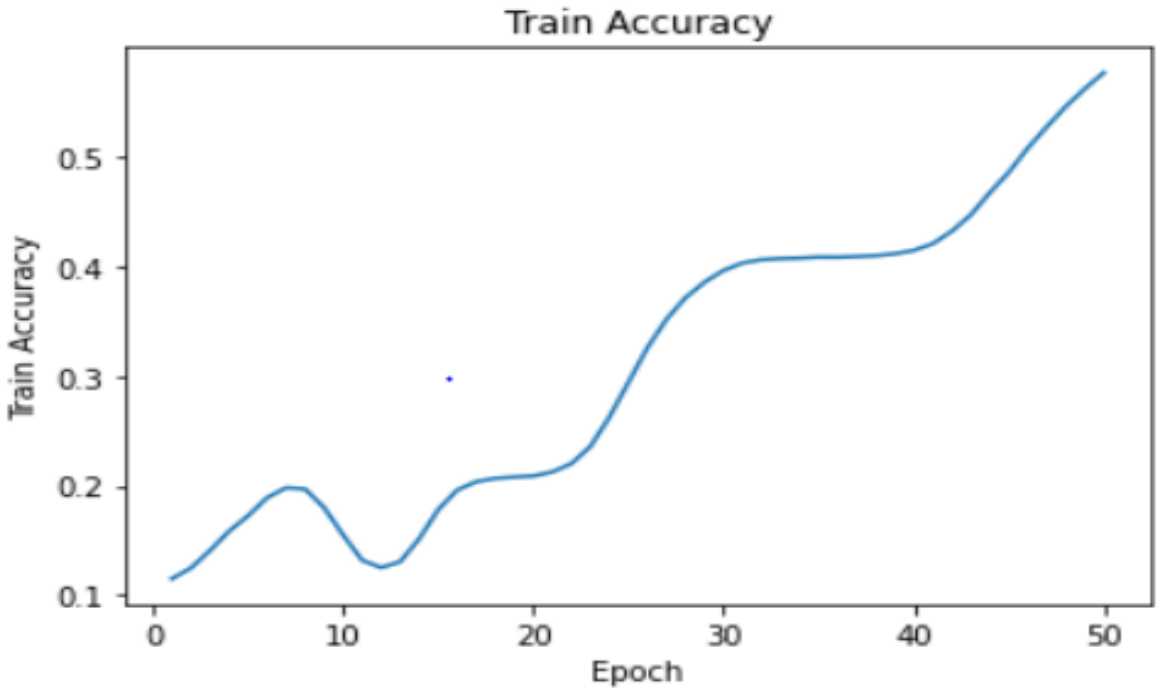


Figure 27: Training Accuracy Vs Epoch

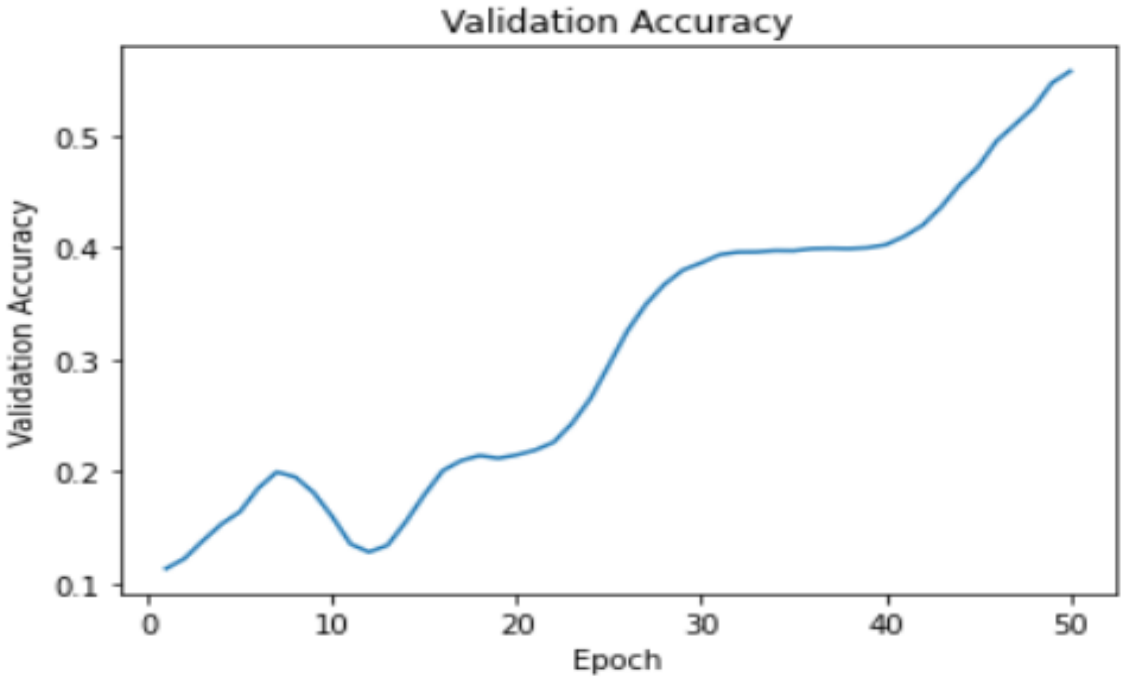


Figure 28: Validation Accuracy Vs Epoch

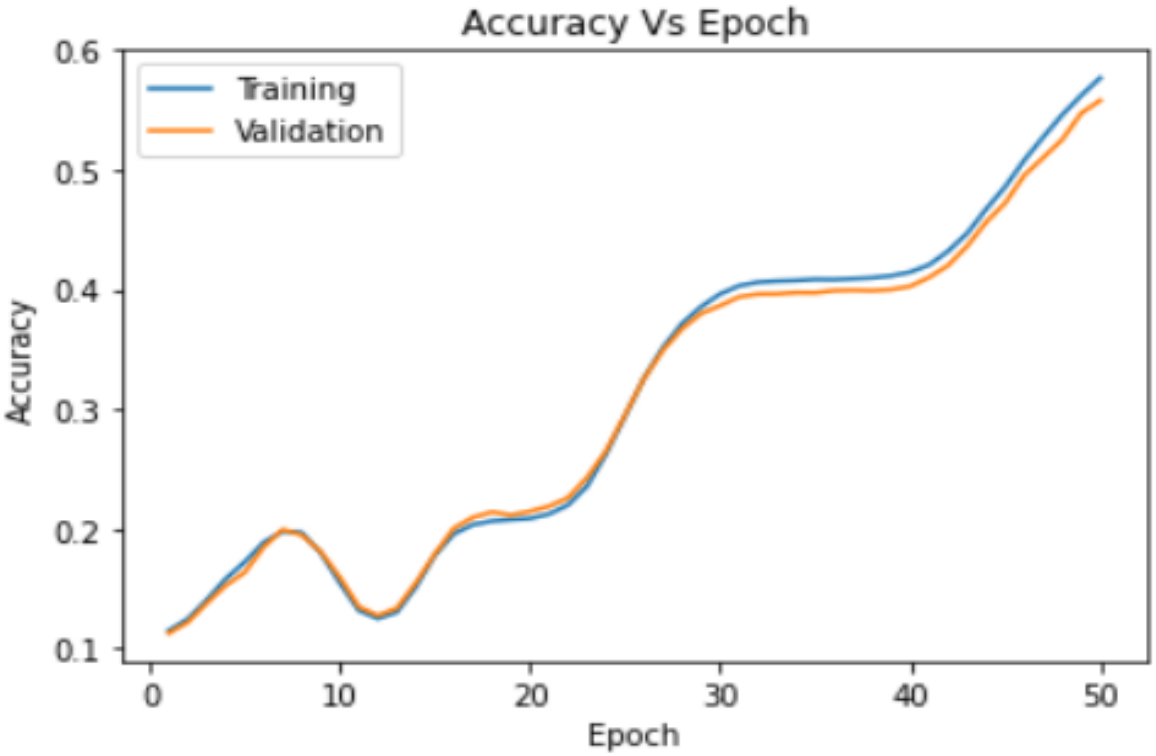


Figure 29: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 19.61

Testing Accuracy = 57.04%

LEARNING RATE AS 0.005

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking learning rate as 0.005.

PLOT OF LOSSES VS EPOCHS

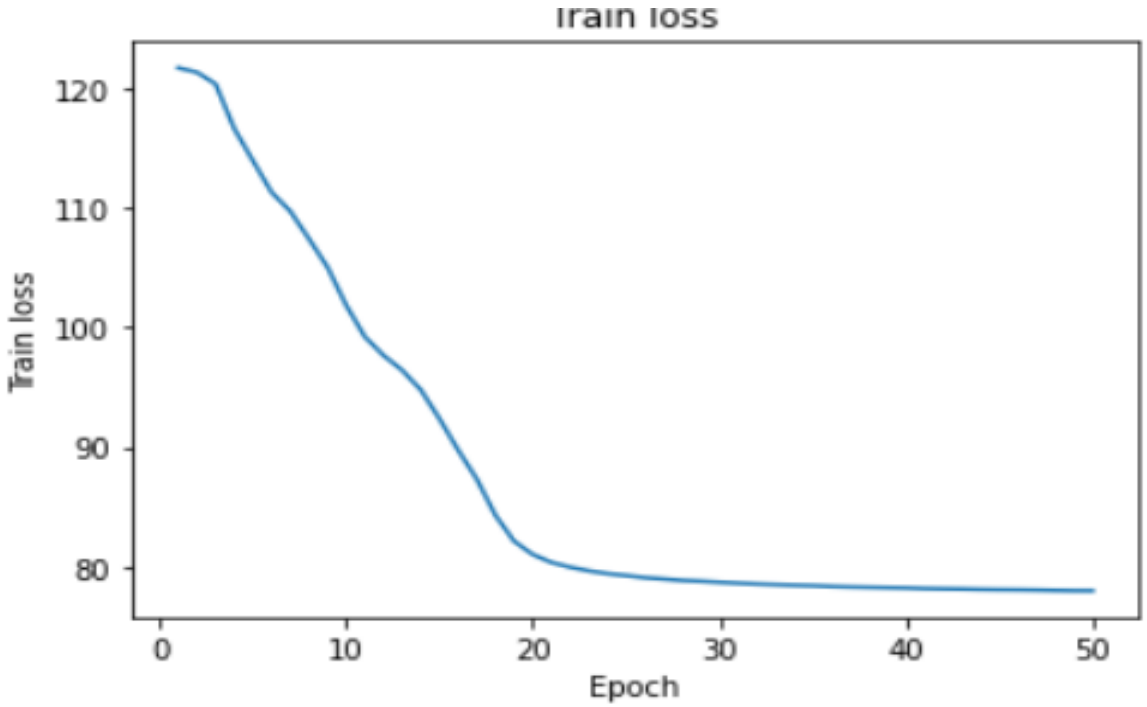


Figure 30: Training loss Vs Epoch

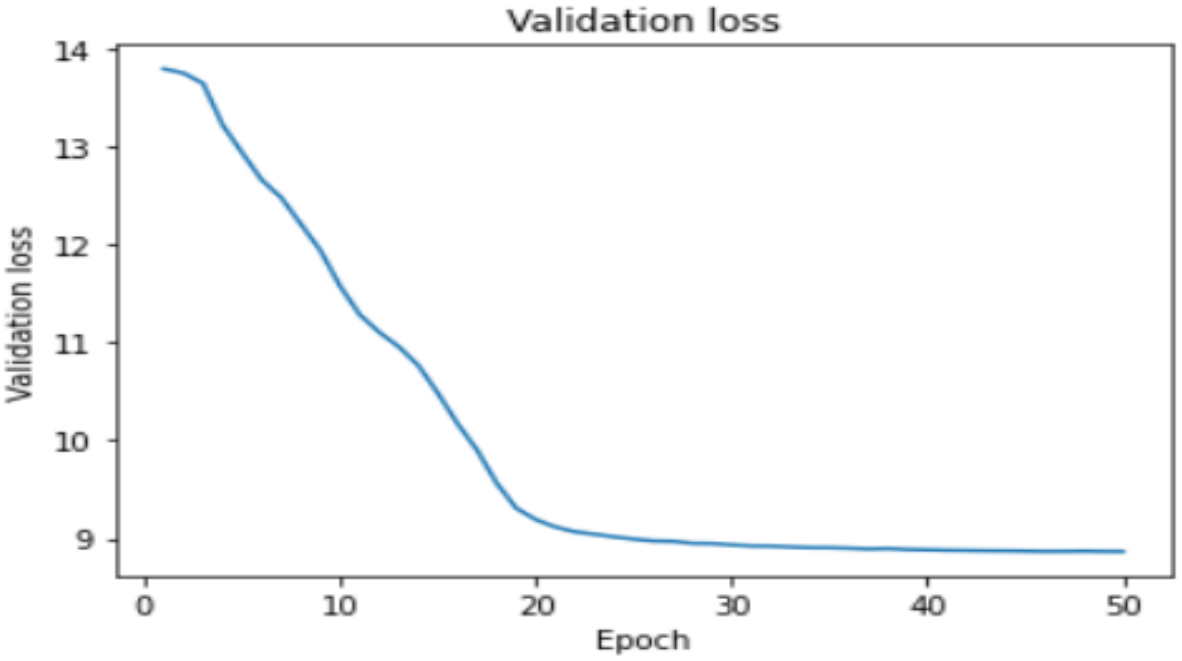


Figure 31: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

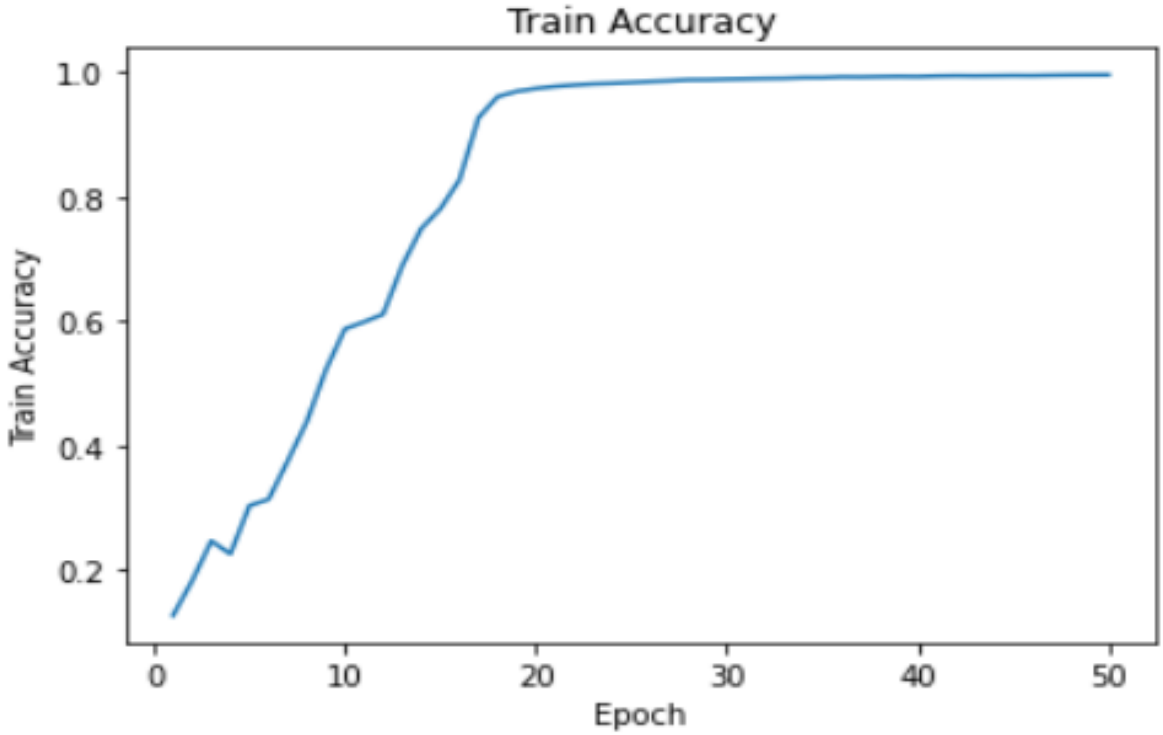


Figure 32: Training Accuracy Vs Epoch

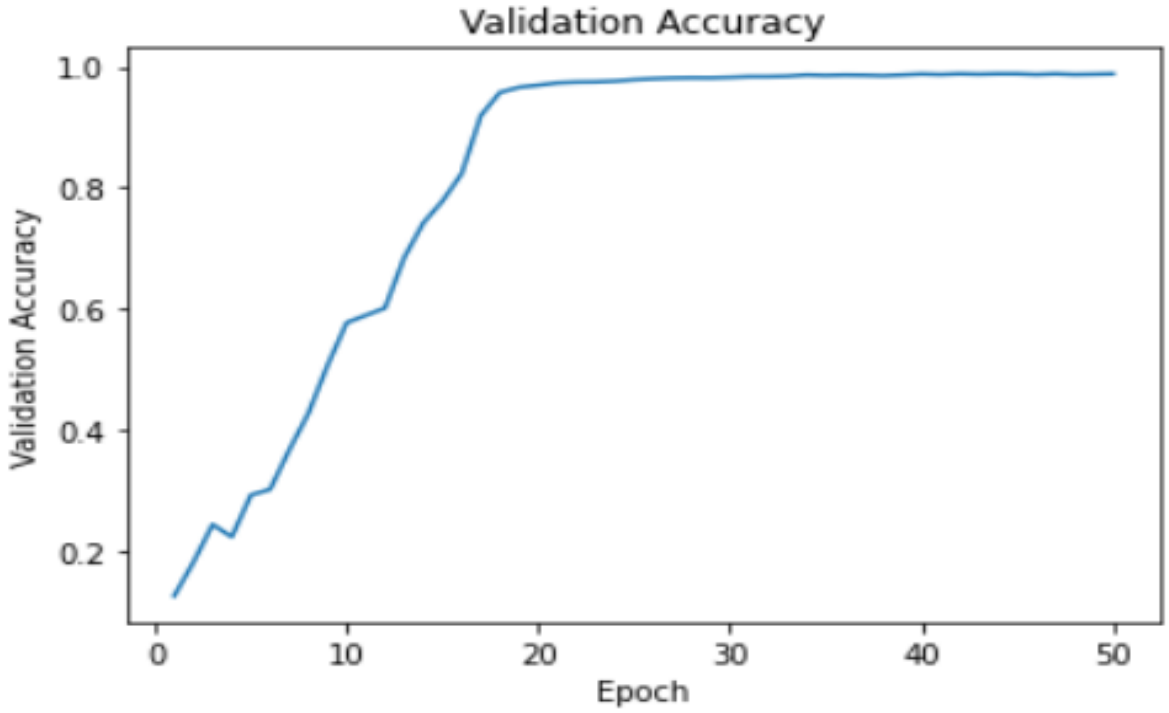


Figure 33: Validation Accuracy Vs Epoch

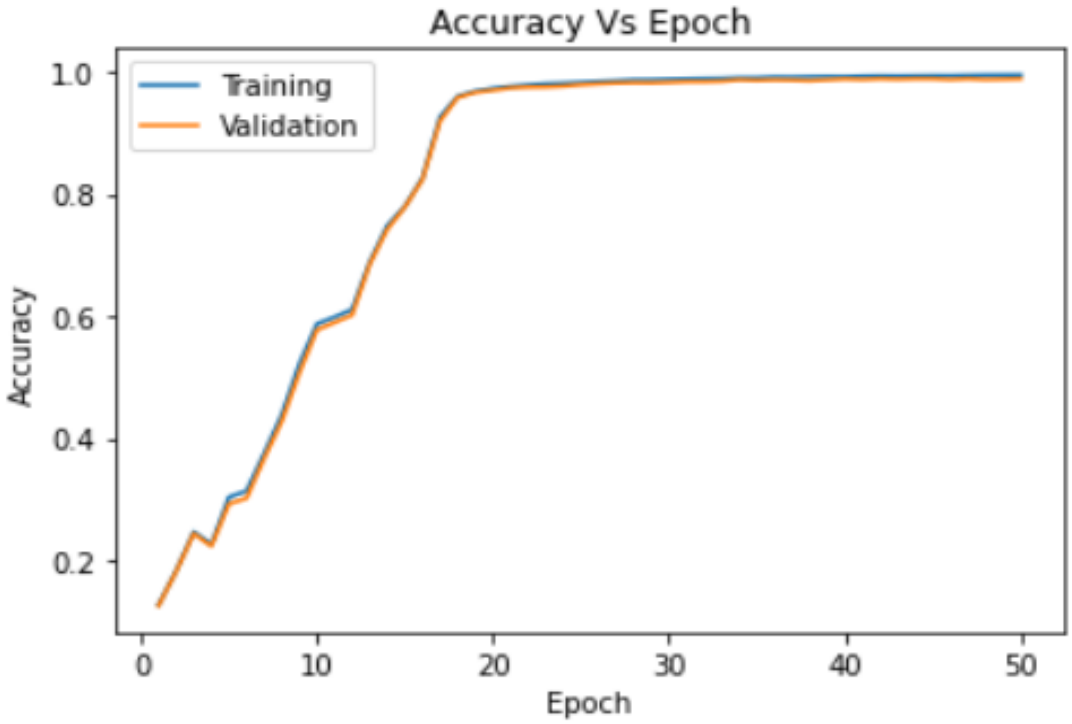


Figure 34: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.75

Testing Accuracy = 98.99%

LEARNING RATE AS 0.01

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking learning rate as 0.01.

PLOT OF LOSSES VS EPOCHS

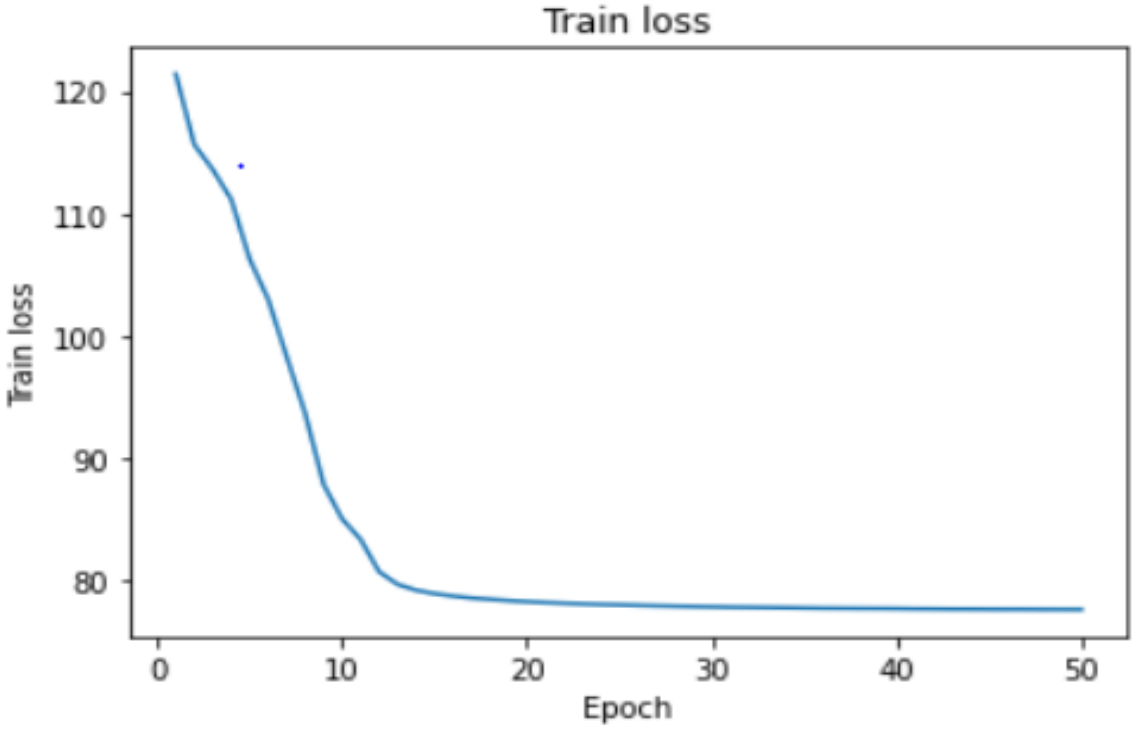


Figure 35: Training loss Vs Epoch

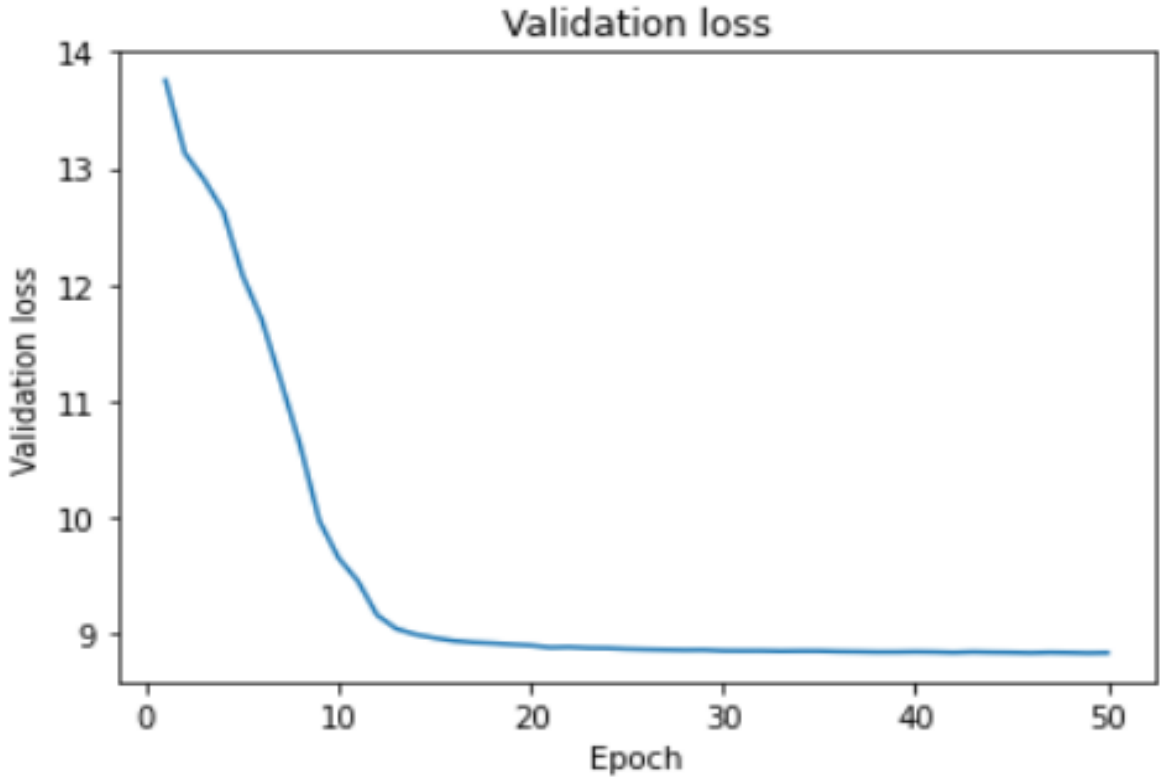


Figure 36: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

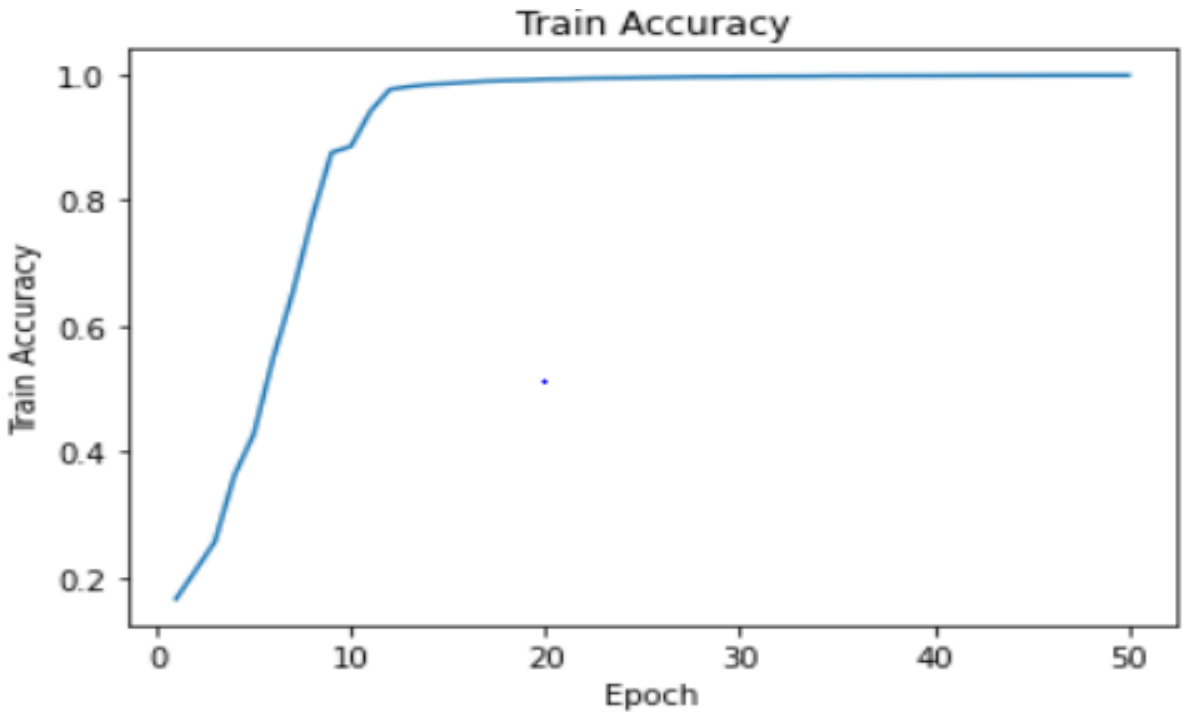


Figure 37: Training Accuracy Vs Epoch

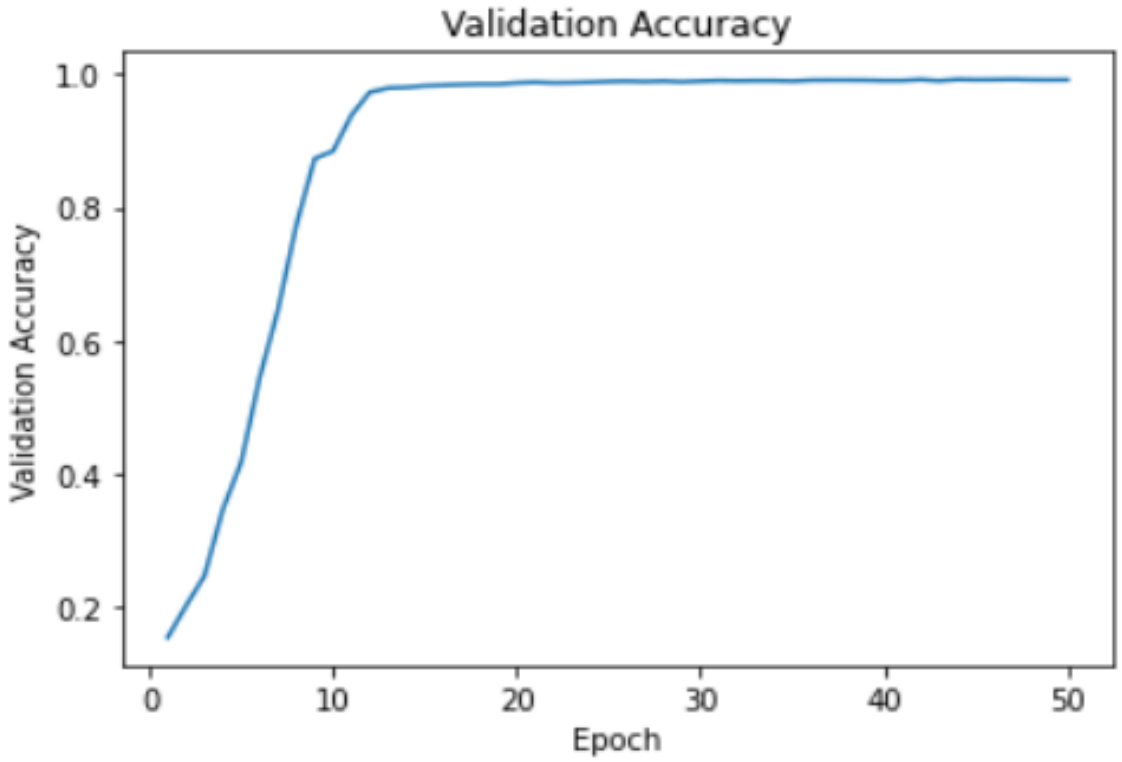


Figure 38: Validation Accuracy Vs Epoch

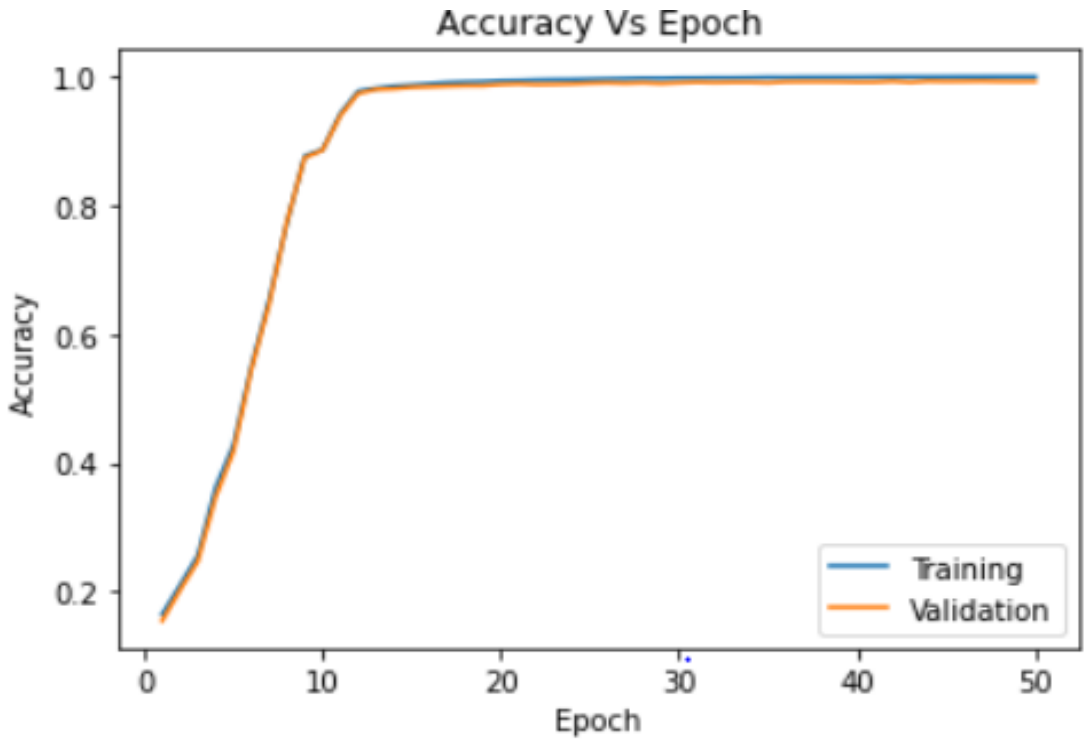


Figure 39: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.71

Testing Accuracy = 99.19

COMPARISON BY VARYING LEARNING RATE

Learning rate	Training Accuracy	Validation Accuracy	Testing Loss	Testing Accuracy
0.001	57.68	55.80	19.61	57.04
0.005	99.49	99.87	14.75	98.99
0.01	99.12	99.84	14.71	99.19

Figure 40: Comparison By Varying learning rate

CONCLUSION

Based on our experiments, the learning rate of 0.01 resulted in the best performance for our model. We noticed that when we set the learning rate to 0.001, the model's parameter updates were too small, leading to slow convergence towards the optimal values. Conversely, when we set the learning rate to 0.005, we observed intermediate results. It's worth noting that the appropriate learning rate for a given model depends on several factors, such as the size and complexity of the dataset, the type of optimizer used, and the network's architecture. Therefore, it's crucial to experiment with different learning rates and carefully monitor the model's performance to determine the optimal learning rate.

PART 1 (c)

(Role of optimizer) Fix the number of epochs, learning rate, batch size, activation function, train this network first using the following optimizers:

- Vanilla Gradient descent
- Momentum based gradient descent (momentum value of your choice)
- Adam.
- RMS prop

In part c we have varied the optimizer and fixed other parameters as follows:

- No of epochs = 50
- Activation Function = RELU
- Batch Size = 1024

Vanilla Gradient descent

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking Vanilla Gradient descent as optimizer and learning rate as 0.01.

PLOT OF LOSSES VS EPOCHS

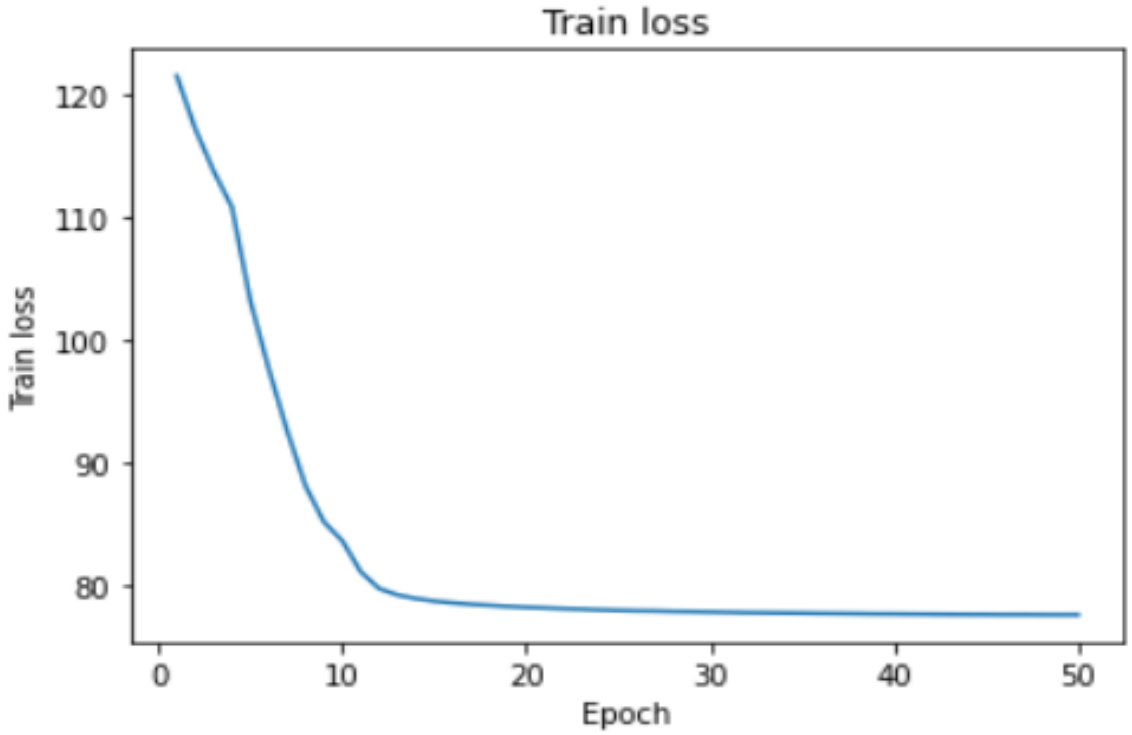


Figure 41: Training loss Vs Epoch

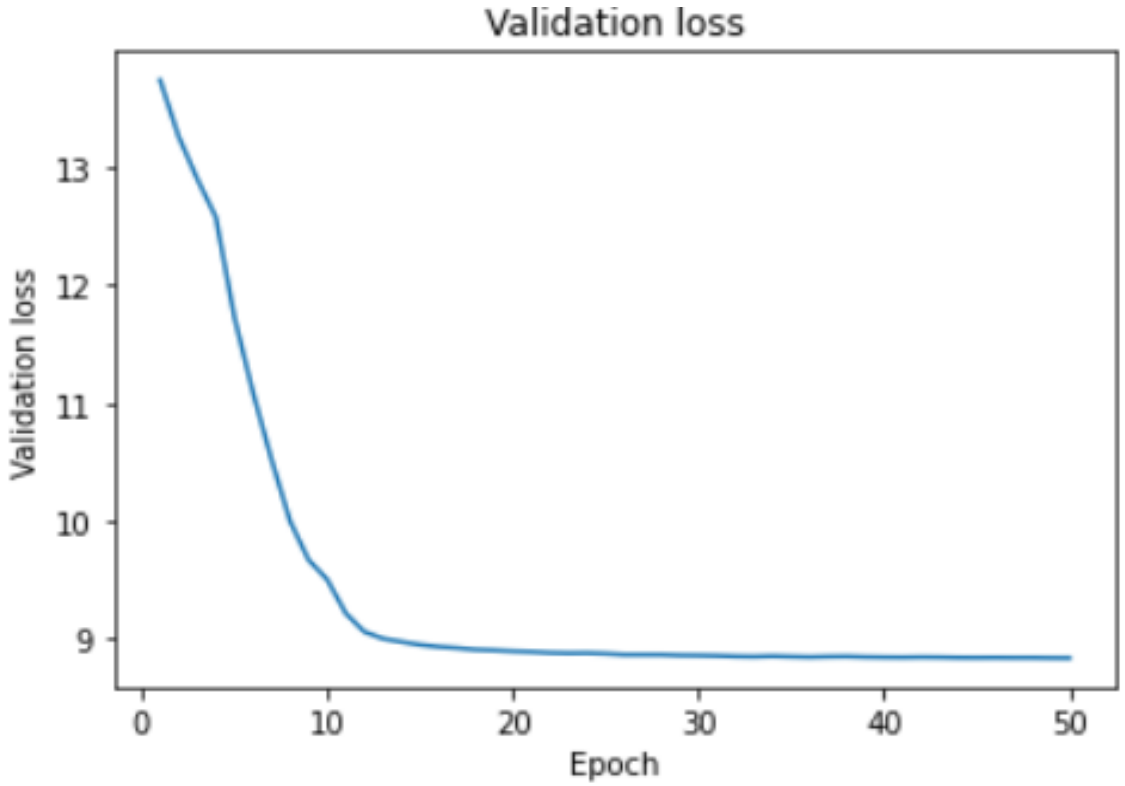


Figure 42: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

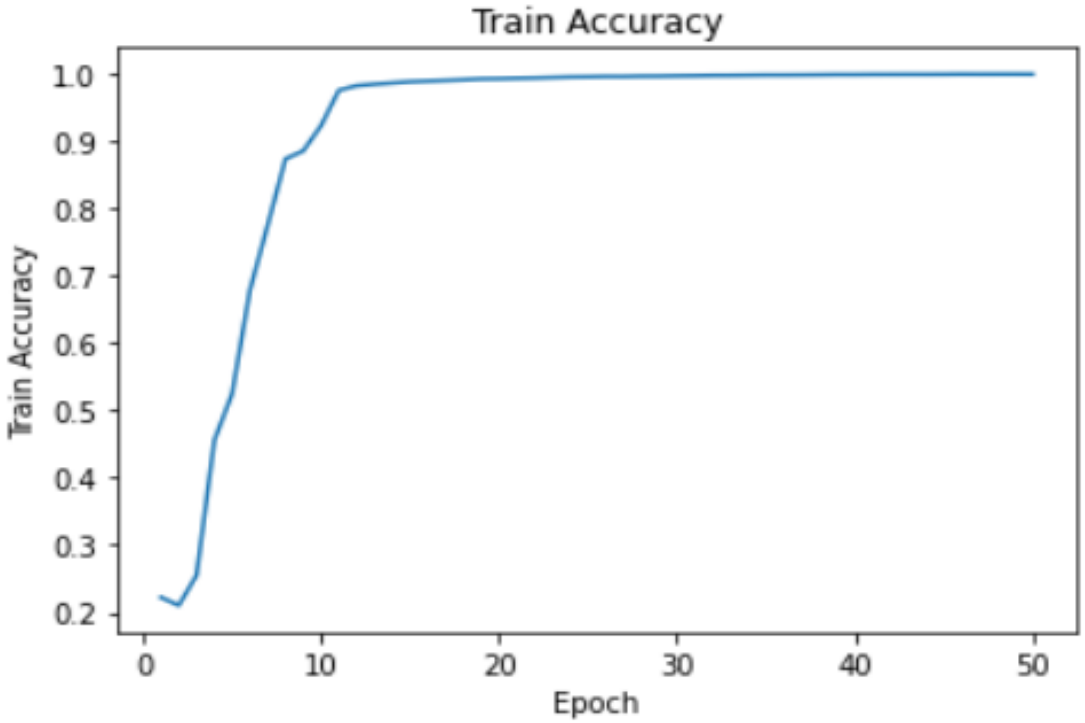


Figure 43: Training Accuracy Vs Epoch

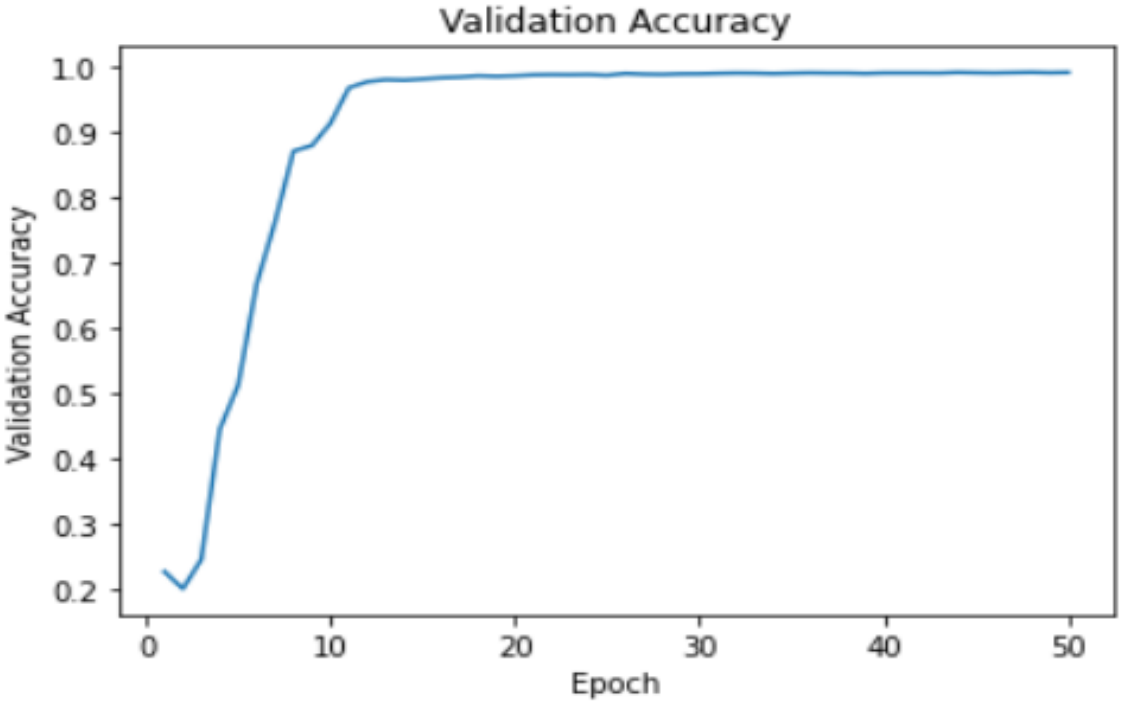


Figure 44: Validation Accuracy Vs Epoch

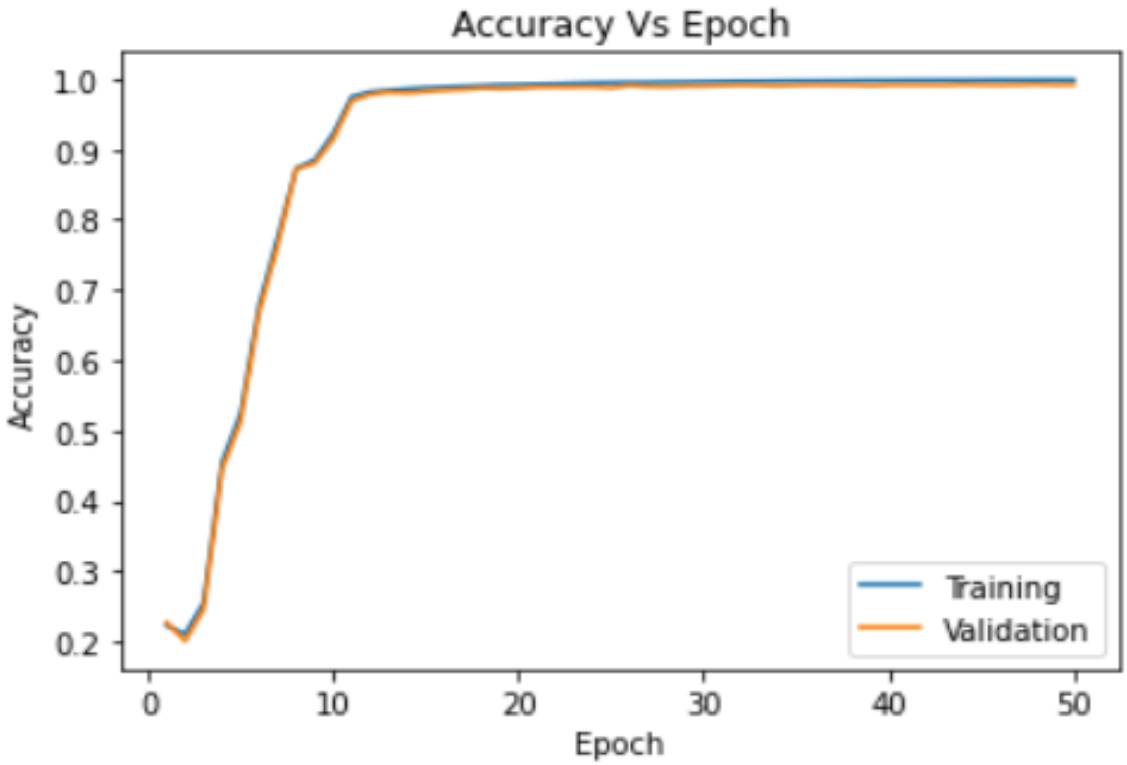


Figure 45: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.71

Testing Accuracy = 99.19

Momentum based gradient descent

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking Momentum based gradient descent as optimizer and learning rate as 0.01.

PLOT OF LOSSES VS EPOCHS

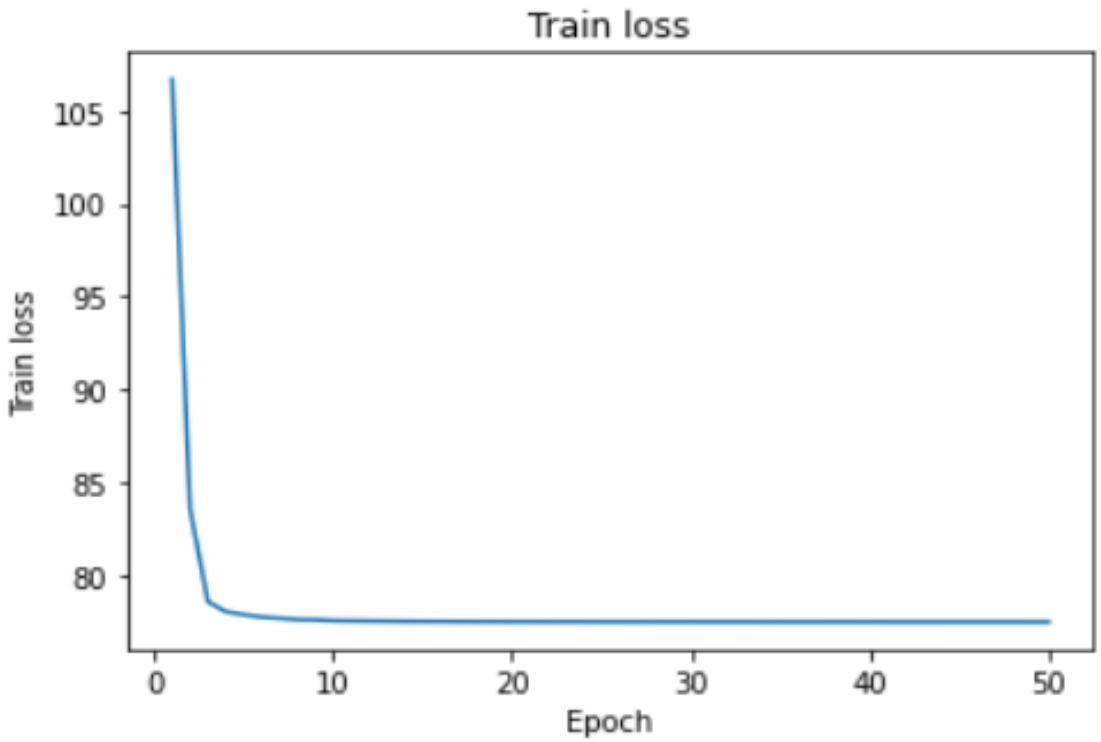


Figure 46: Training loss Vs Epoch

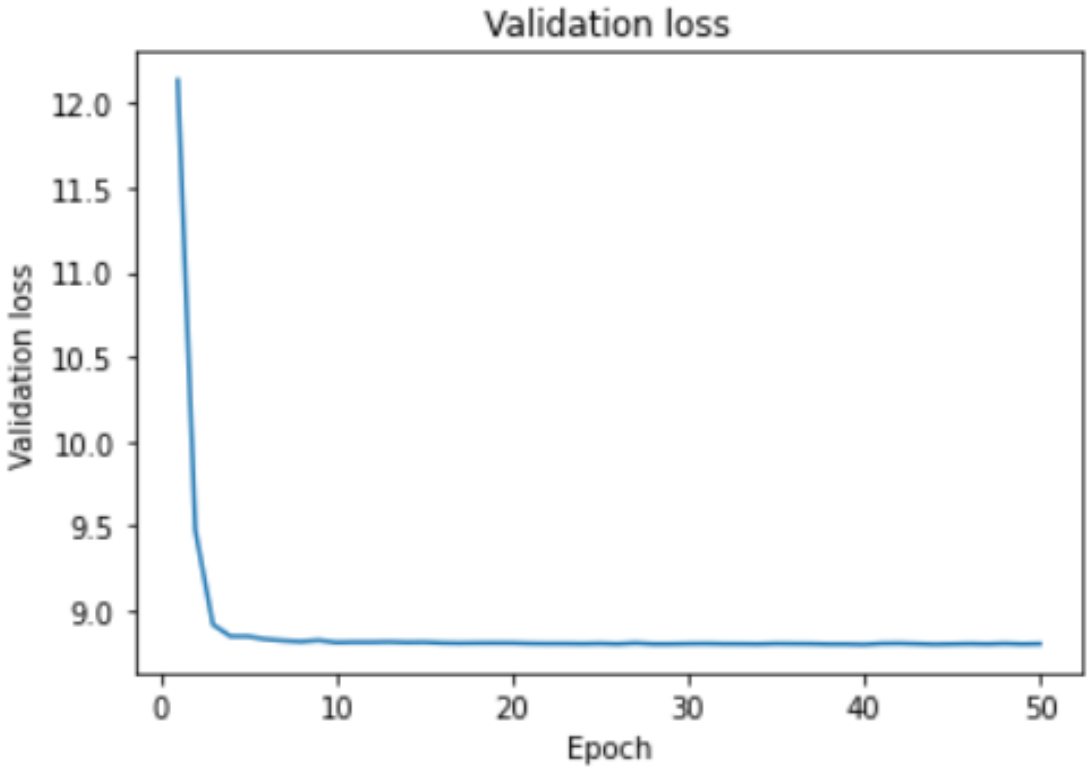


Figure 47: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

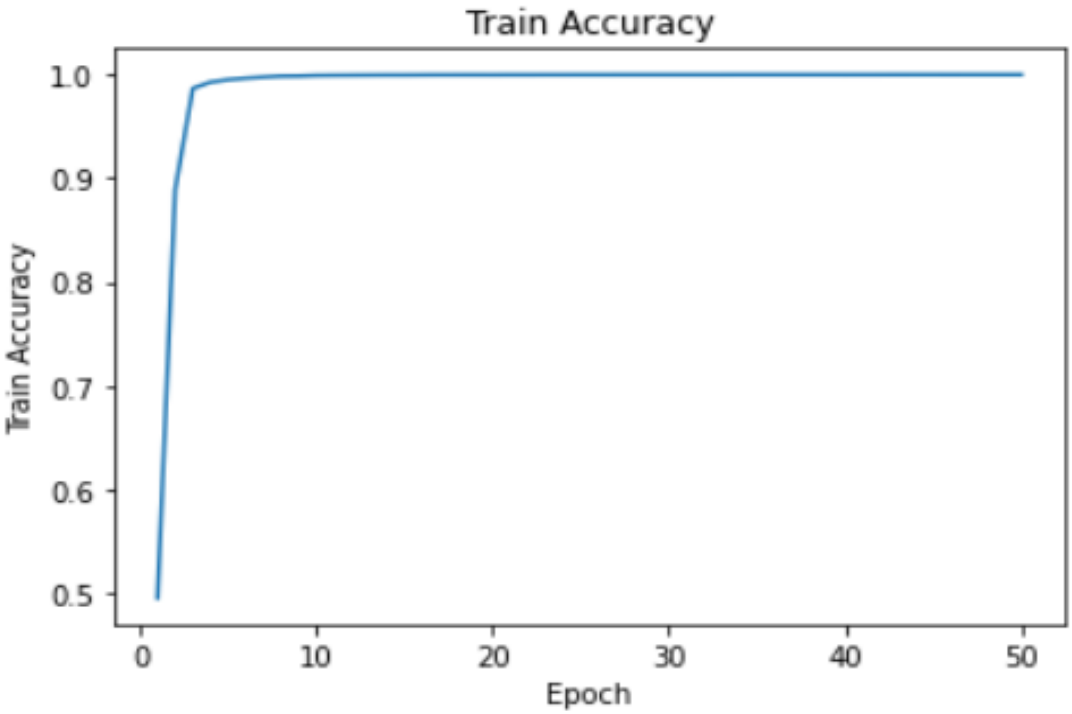


Figure 48: Training Accuracy Vs Epoch

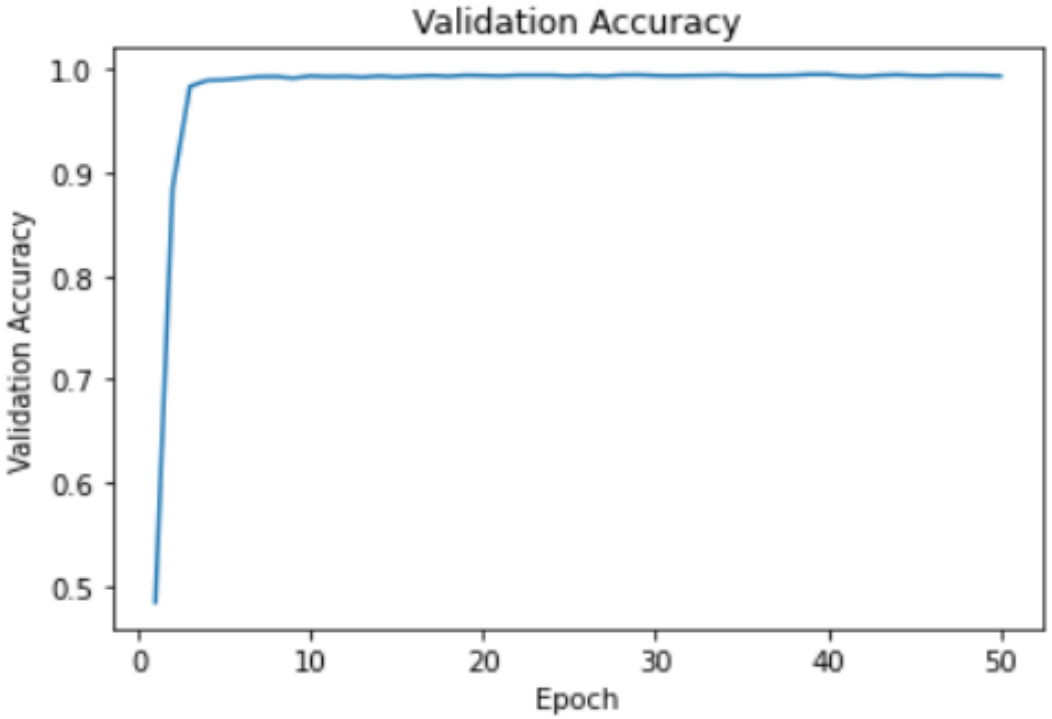


Figure 49: Validation Accuracy Vs Epoch

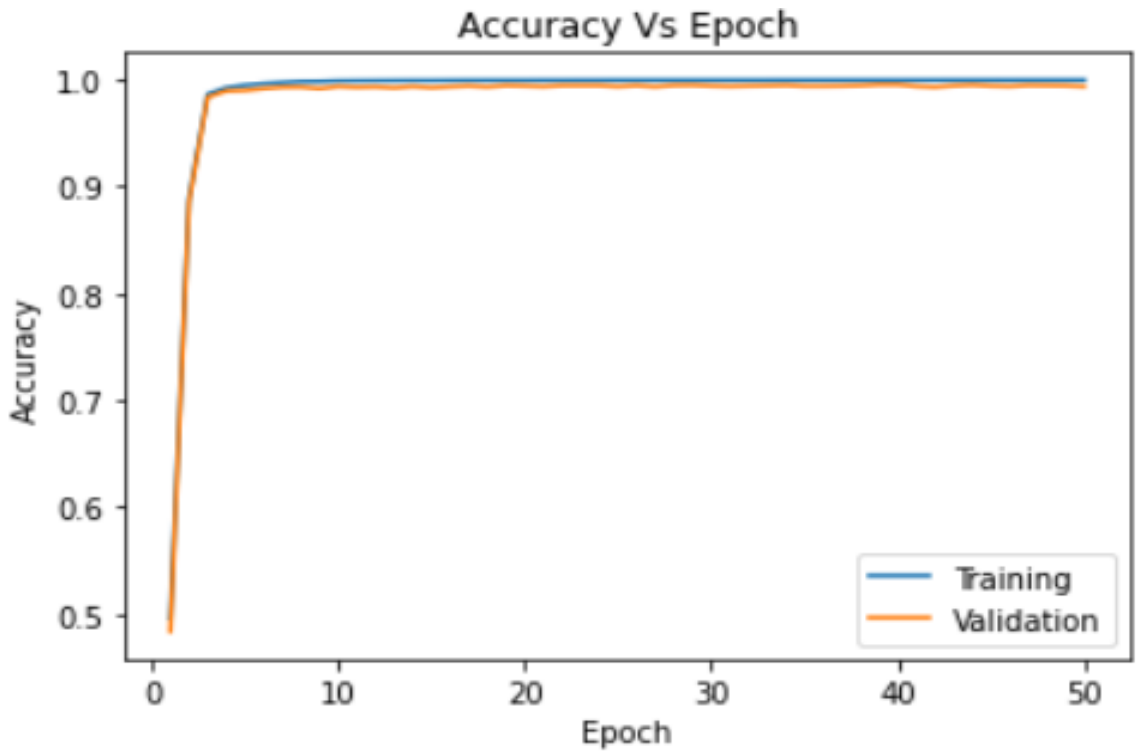


Figure 50: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.60

Testing Accuracy = 99.50

ADAM

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking Adam as optimizer and learning rate as 0.00005.

PLOT OF LOSSES VS EPOCHS

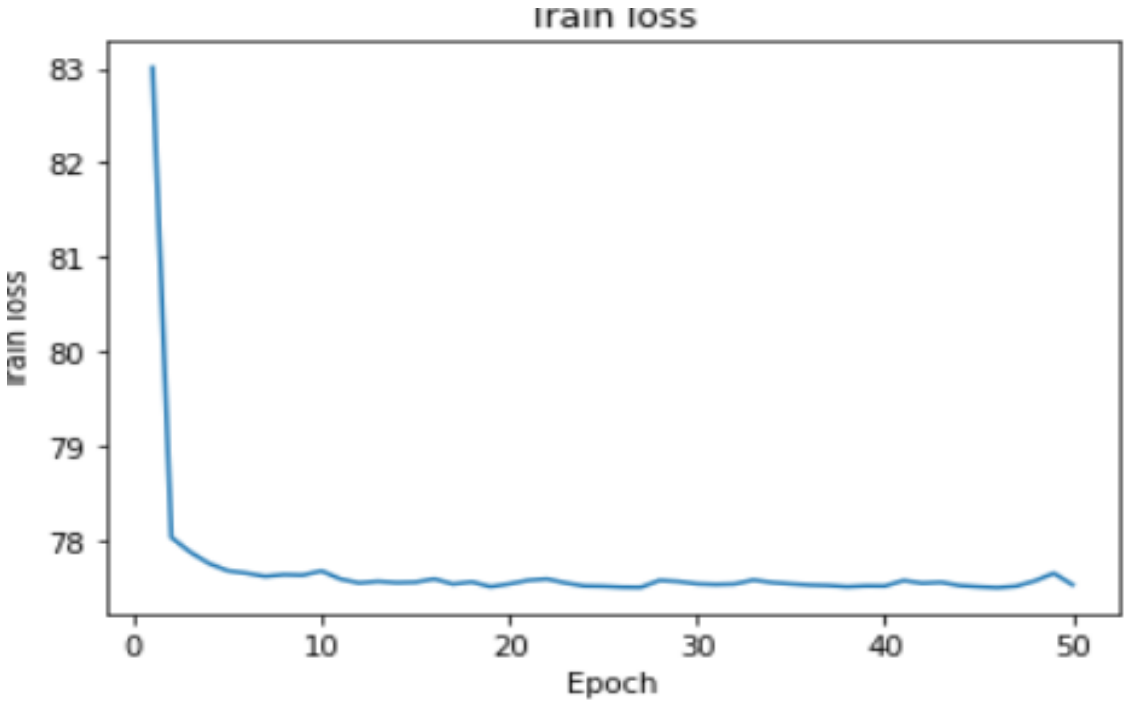


Figure 51: Training loss Vs Epoch

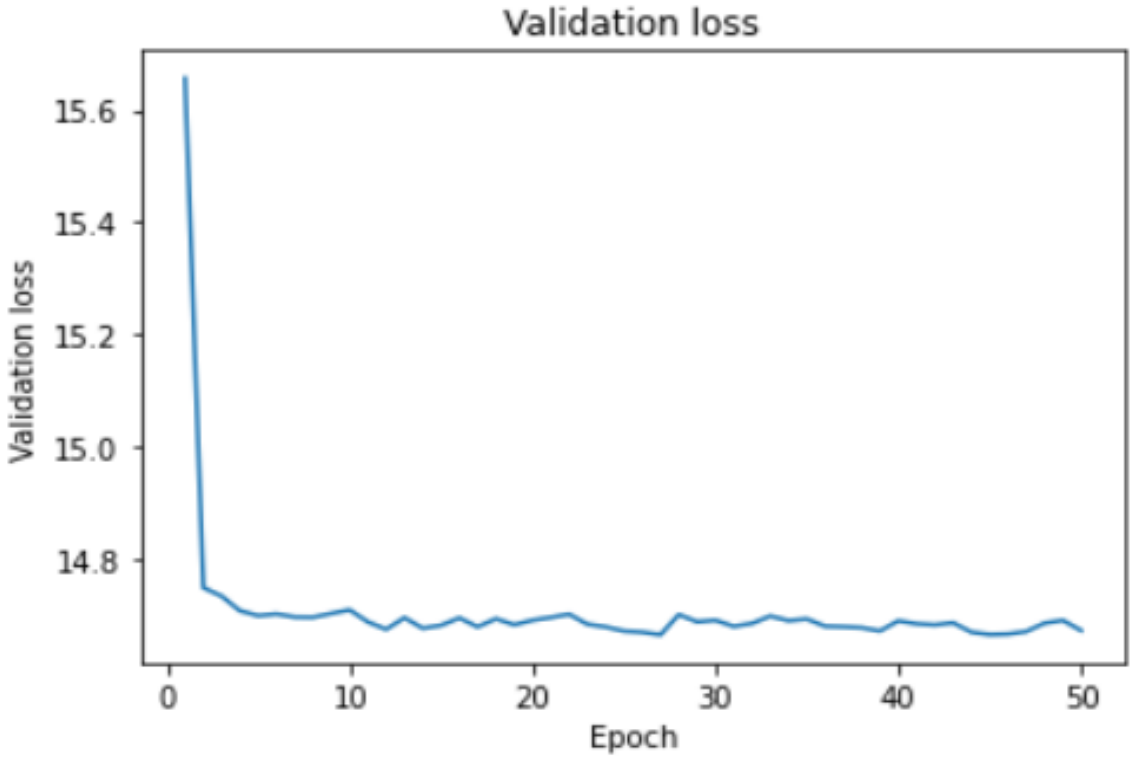


Figure 52: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

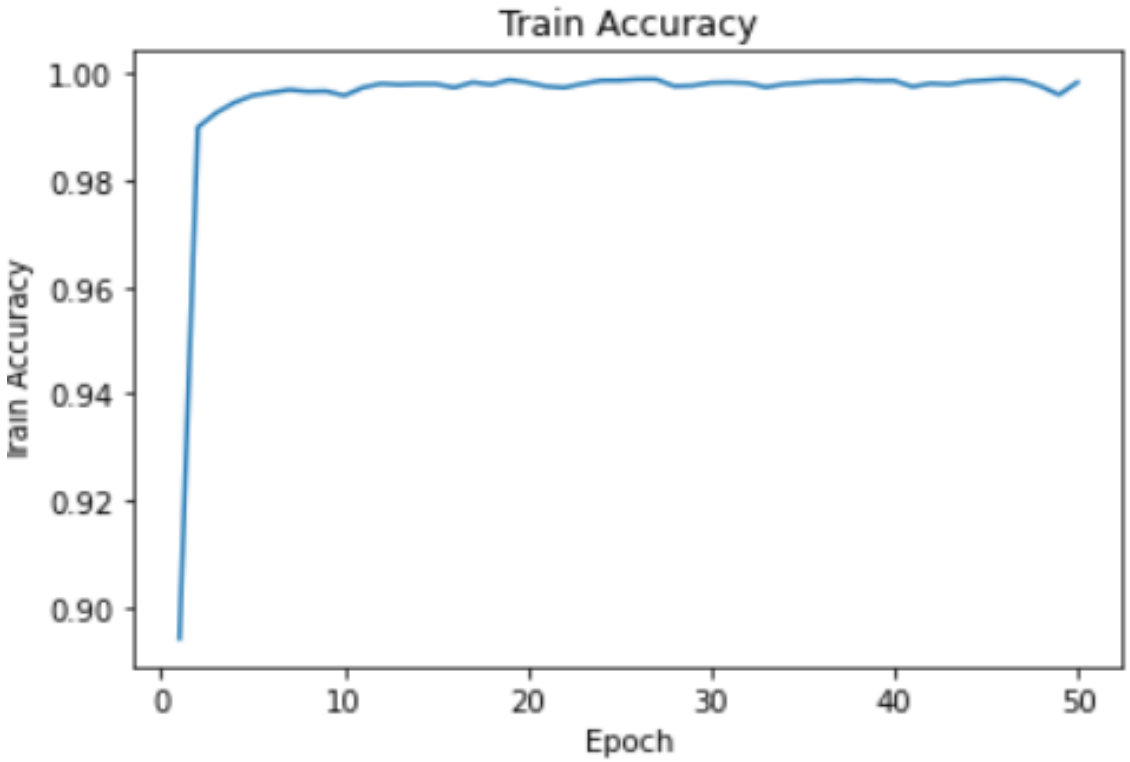


Figure 53: Training Accuracy Vs Epoch

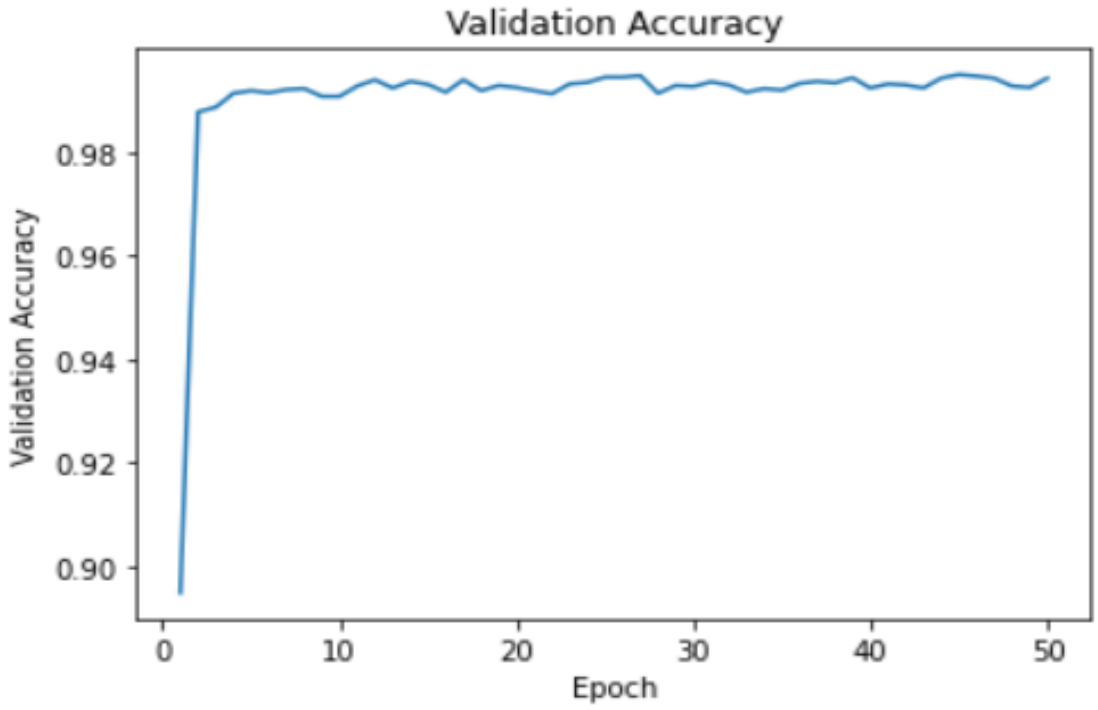


Figure 54: Validation Accuracy Vs Epoch

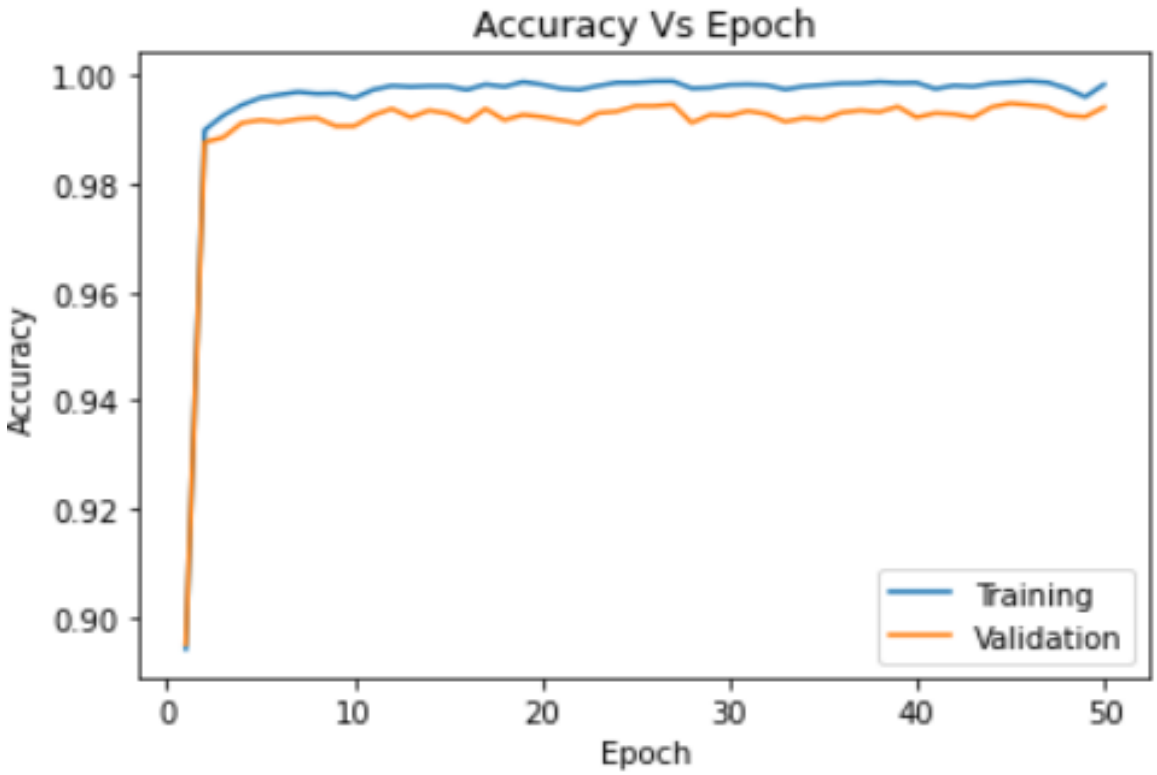


Figure 55: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.66

Testing Accuracy = 99.46

RMS prop

We have plotted training loss, validation loss, training accuracy, validation accuracy by taking RMS prop as optimizer and learning rate as 0.00005.

PLOT OF LOSSES VS EPOCHS

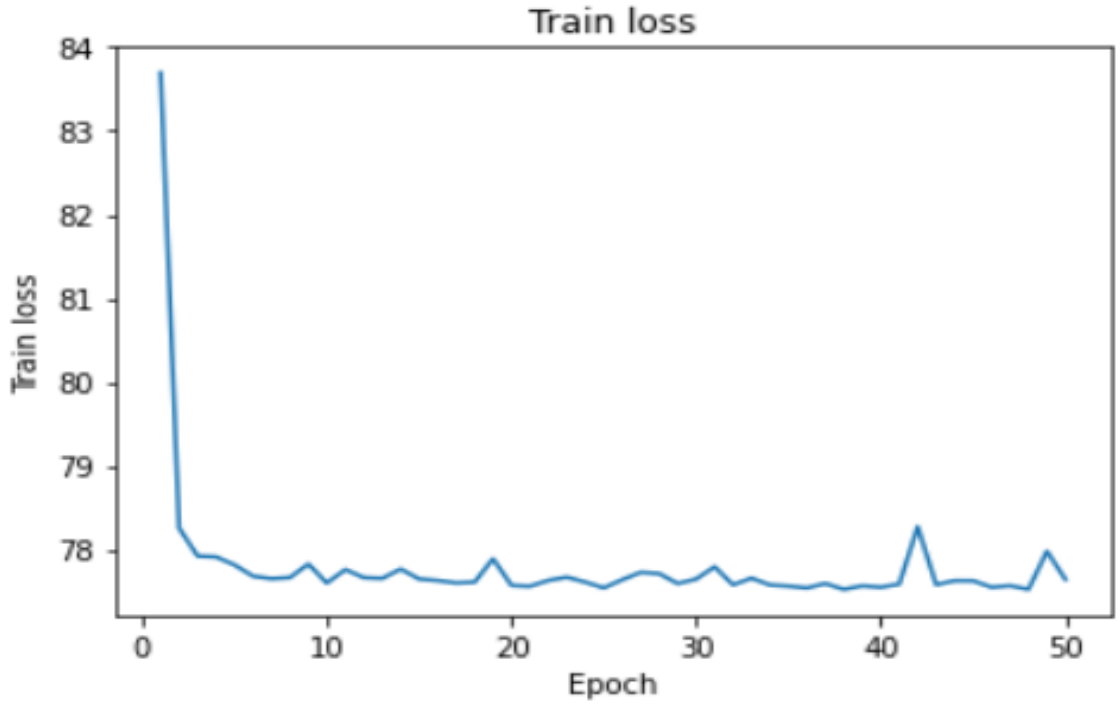


Figure 56: Training loss Vs Epoch

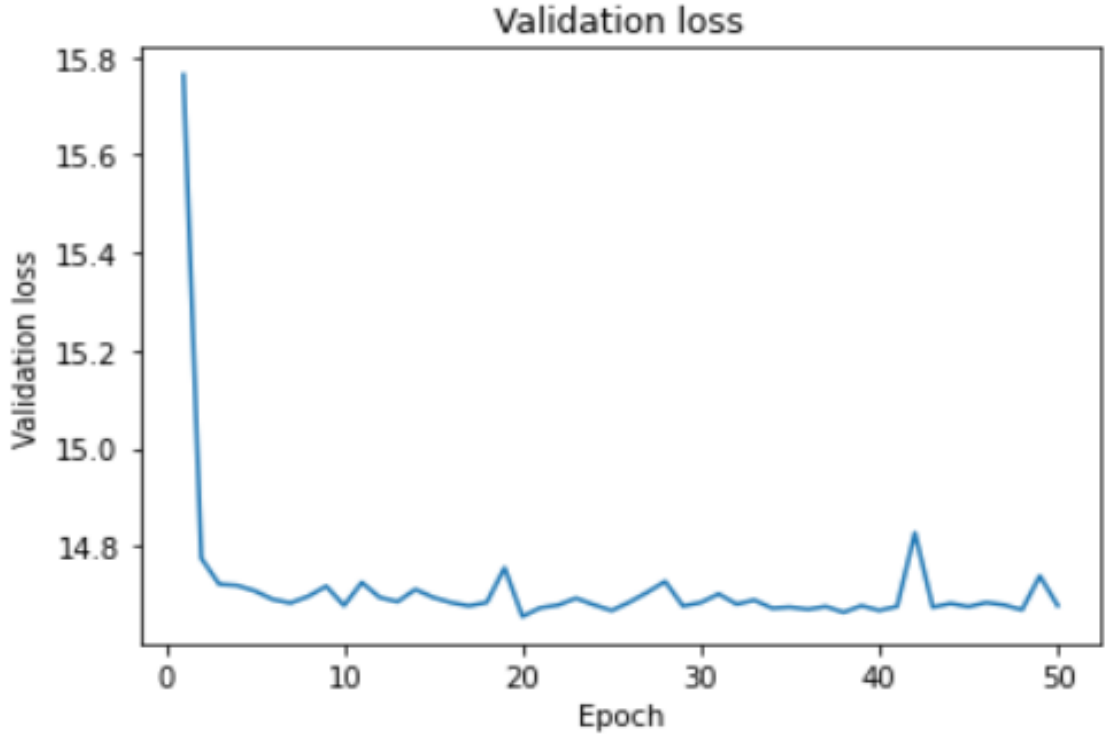


Figure 57: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

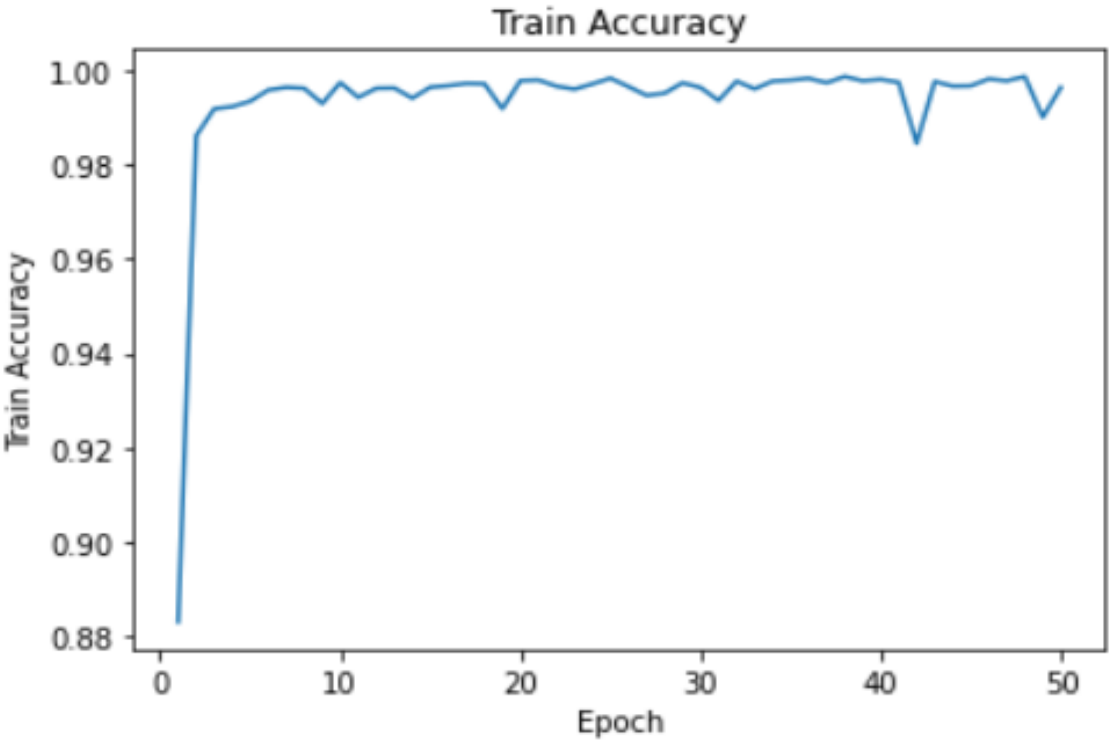


Figure 58: Training Accuracy Vs Epoch

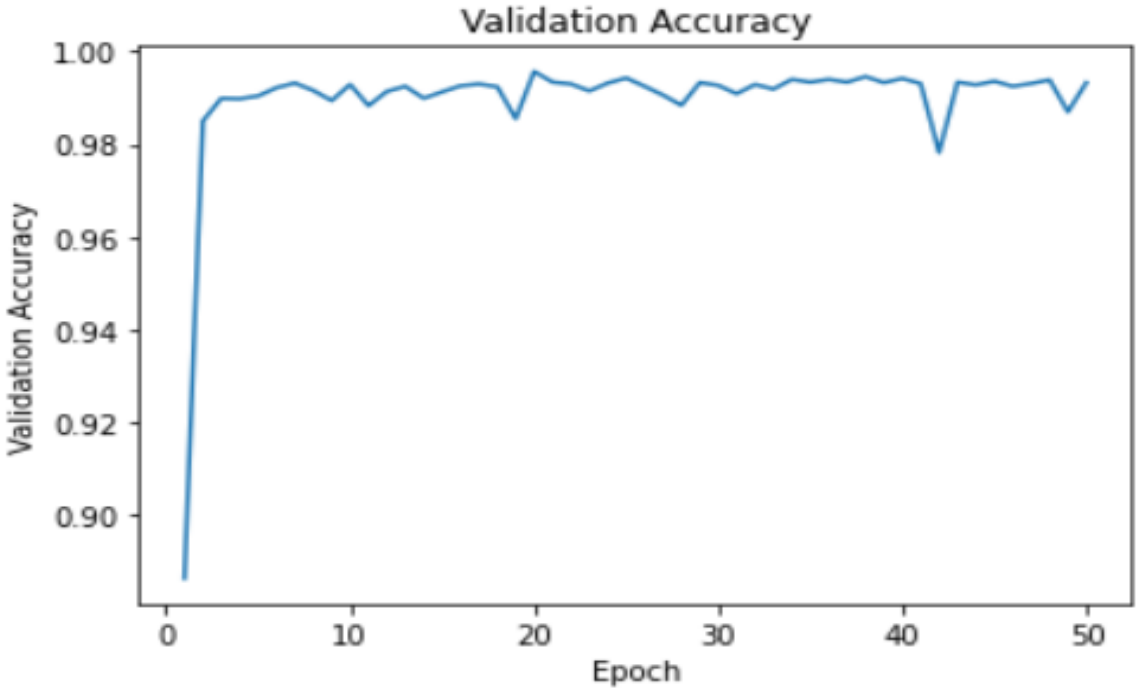


Figure 59: Validation Accuracy Vs Epoch

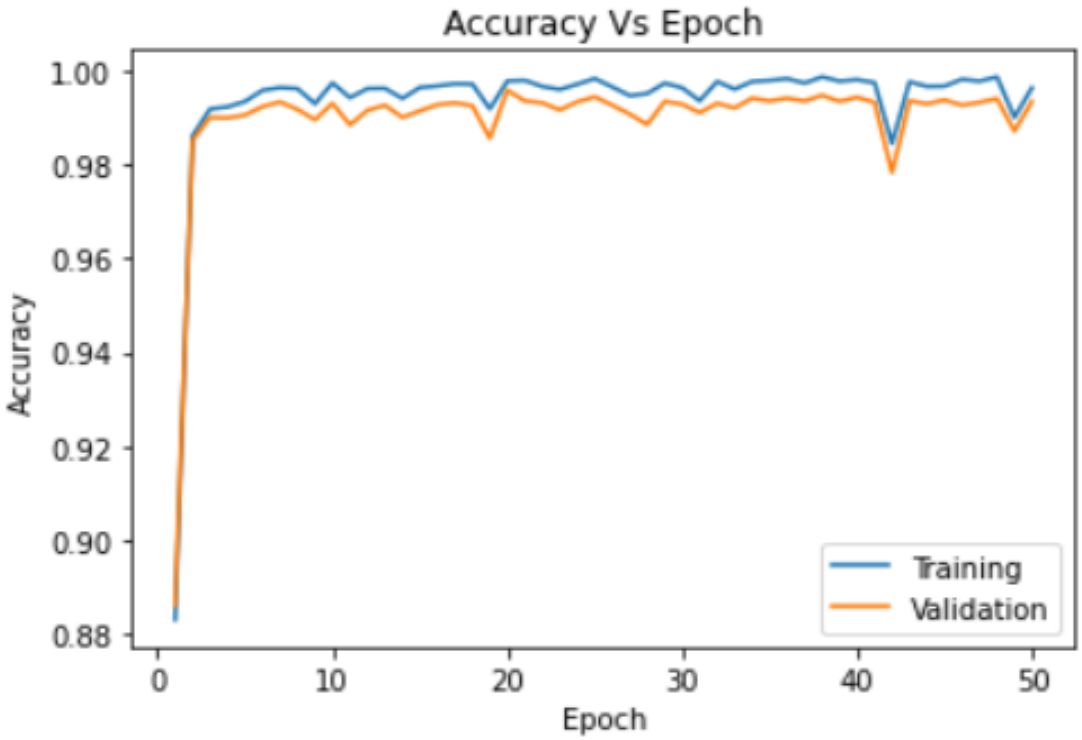


Figure 60: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 14.68

Testing Accuracy = 99.31

COMPARISON BY VARYING OPTIMIZER

OPTIMIZER	Training Accuracy	Validation Accuracy	Testing Loss	Testing Accuracy
Vanilla gradient descent	99.85	99.20	14.71	99.19
Momentum Gradient descent	99.96	99.47	14.60	99.50
Adam	99.98	99.42	14.66	99.46
RMS prop	99.96	99.33	14.68	99.31

Figure 61: Comparison by varying optimizer

CONCLUSION

The most effective model was discovered by experimenting with various values of learning rate, activation function, and optimizer. It was determined that VGG16, utilizing ReLU activation, a learning rate of 0.01, and an optimization technique based on Momentum-based gradient descent, achieved the best results. When evaluated on the validation dataset comprising 10,000 images, this model achieved an accuracy of 99.50%.

PART 1 (d)

(Transfer Learning) So far, we have understood the best/optimal hyperparameters that yield the best performance in your set of experiments. Save the weights/configuration of this model (this is what we will call the pretrained model). In this experiment, rather than classifying the images into classes 0 to 9, we will slightly change the problem statement in the following way:

- The dataset with digits 0-9 has mainly two types of information: either the handwritten digit is made using a curved stroke or a straight stroke (or maybe both). Hence, divide the dataset into two classes viz., the set of digits with curved stroke(s) C: {0,2,3,5,6,8,9} and the set of digits with straight stroke(s), S: {1,4,7}.
- Modify the last layer of the network so that now it can classify the digit into class 'C' or class 'S'. Initialize the network with the weights of the pre trained model (except the weights of the penultimate layer that must be trained). In other words, retrain only the last layer of the network keeping the weight of all other layers same as were in the pretrained model.

In this part we have pretrained the model with following parameters:

- No of epochs = 50
- Activation Function = RELU
- Optimizer = RMS prop
- Learning rate = 0.00005
- Batch Size = 1024

PLOT OF LOSSES VS EPOCHS

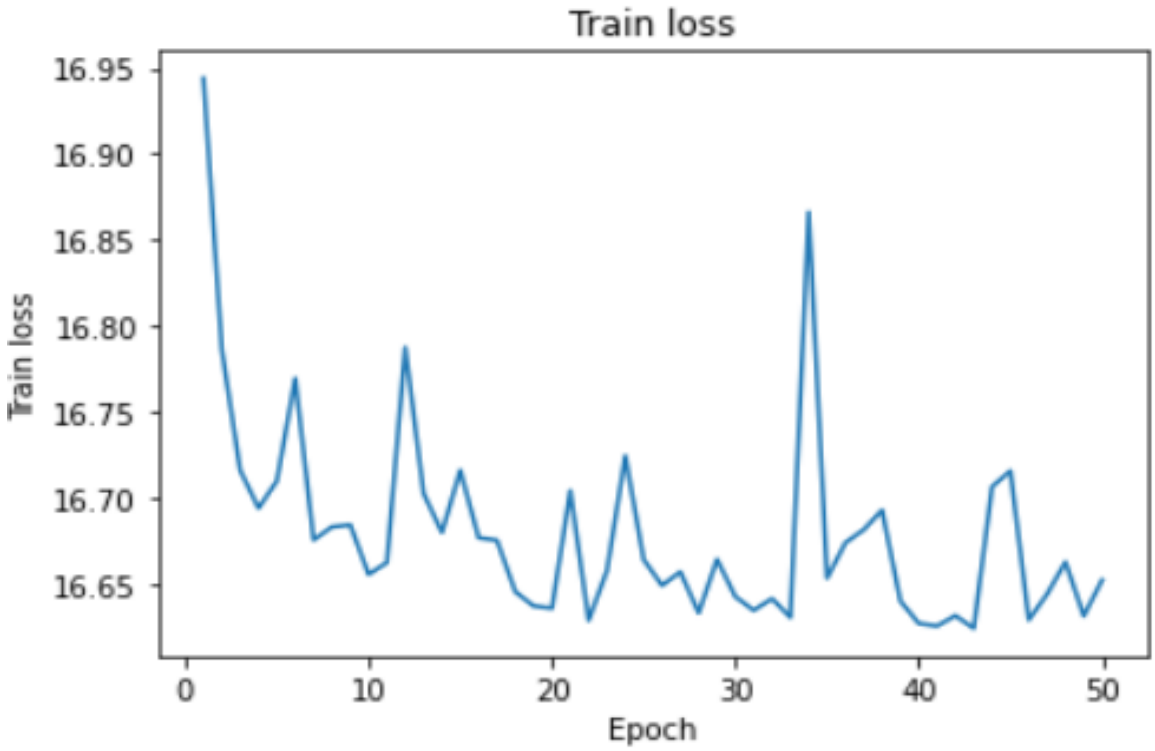


Figure 62: Training loss Vs Epoch

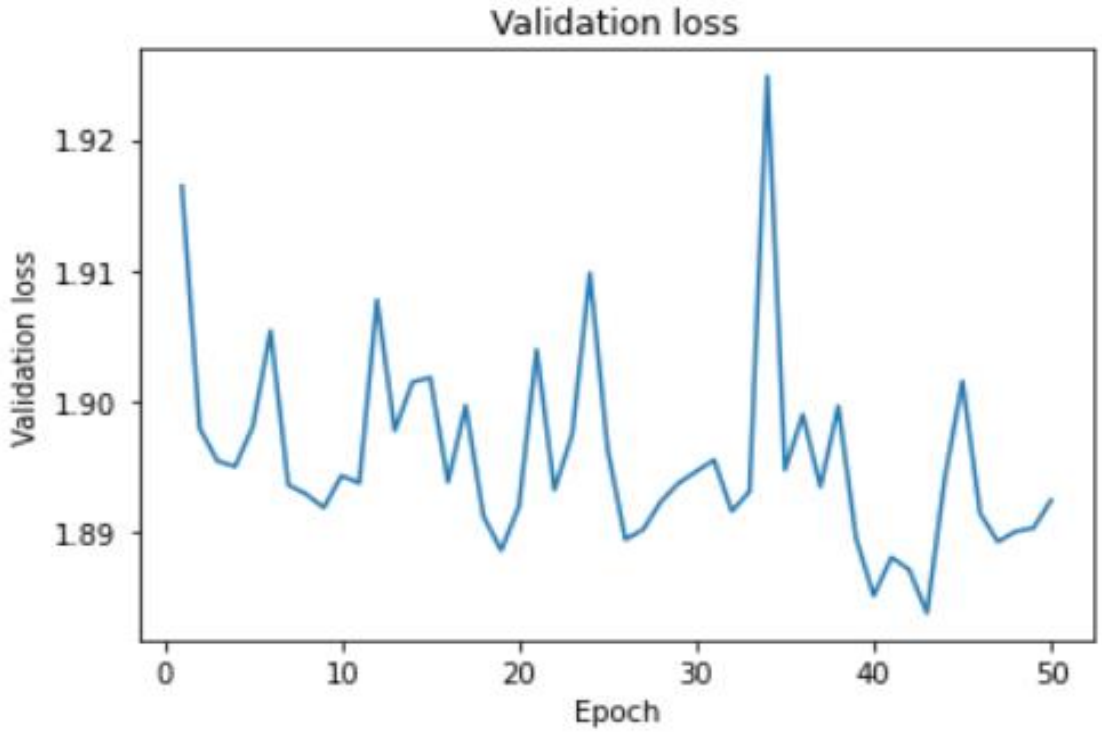


Figure 63: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

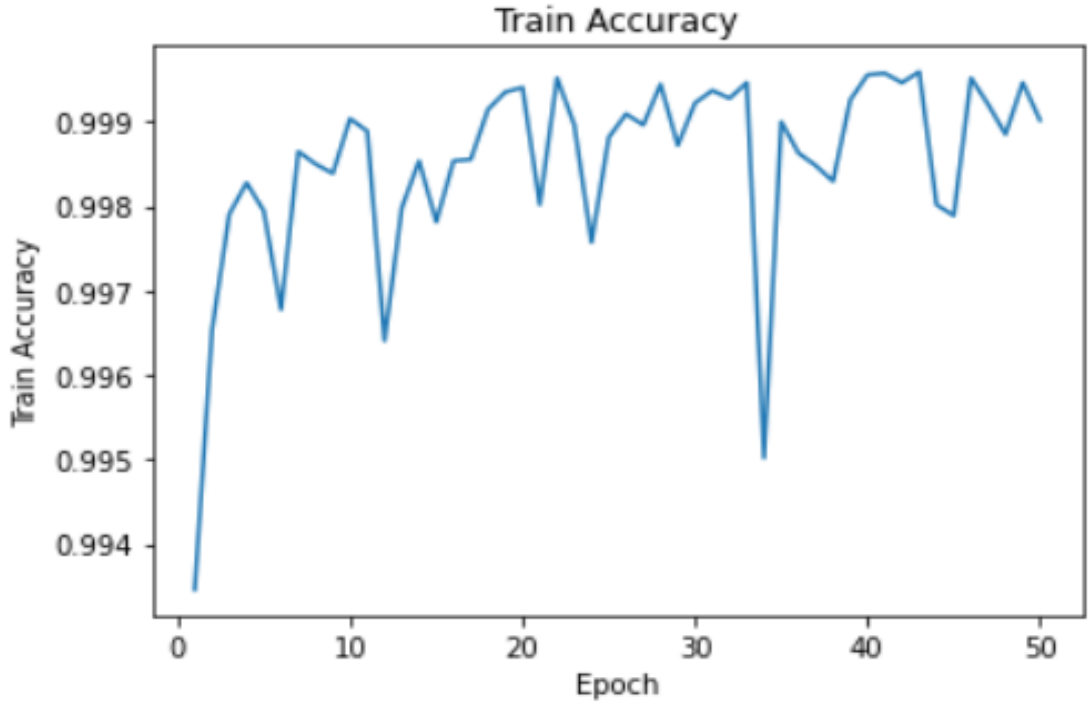


Figure 64: Training Accuracy Vs Epoch

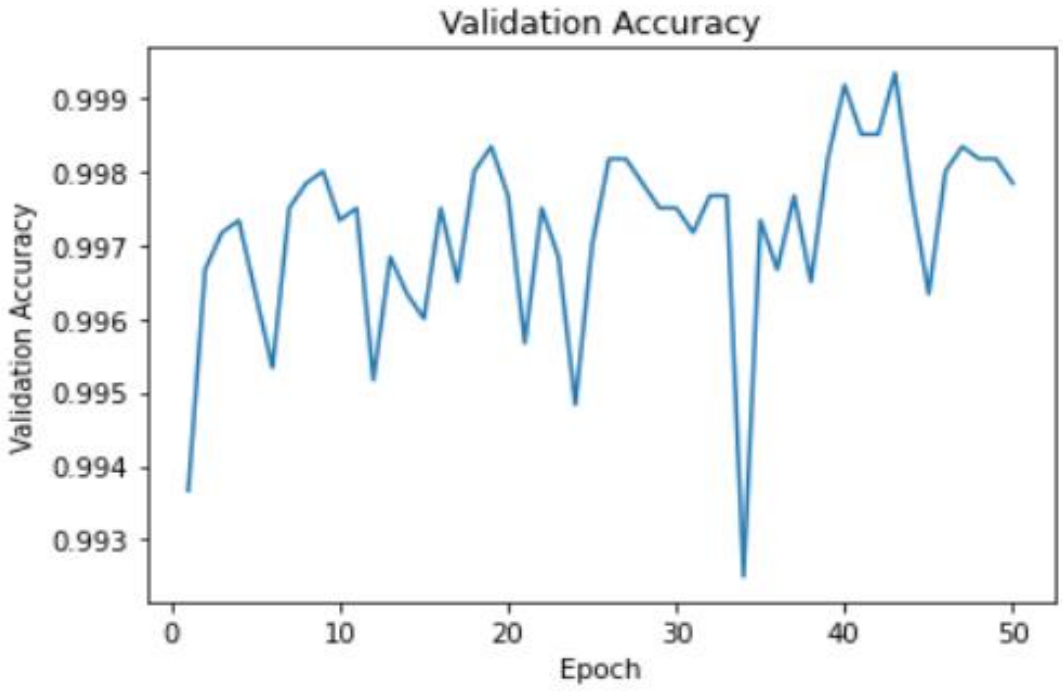


Figure 65: Validation Accuracy Vs Epoch

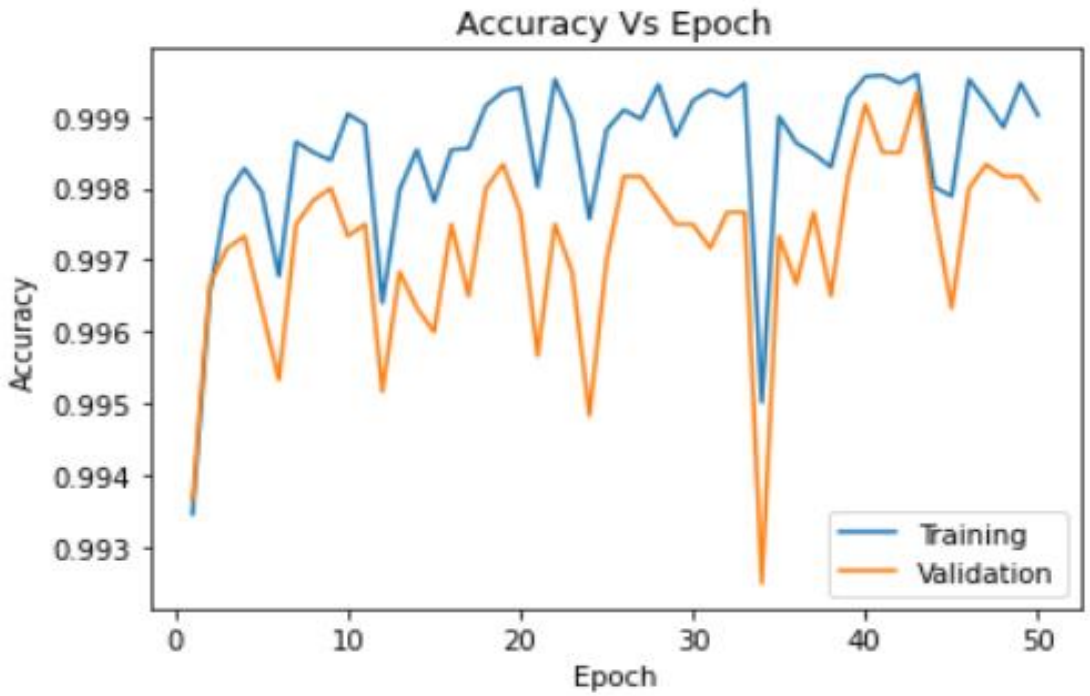


Figure 66: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 3.1174

Testing Accuracy = 99.53%

CONCLUSION

The plots above illustrate that the pre-trained model converges at a faster rate than models with all layers trainable. The highest accuracy achieved by the best pre-trained model on the MNIST test dataset, which contains 10,000 images, was 99.53%. It is worth noting that this model finished training after only a few epochs, and afterward, there were only minor fluctuations in accuracy, with all values remaining above 99.3%.

PART 2

Train Resnet-18 CNN from scratch using CIFAR-10 dataset. The CIFAR-10 dataset consists of 28x28 grayscale images. It is made up of about 50,000 images for the purpose of training and about 10,000 for the purpose of validation.

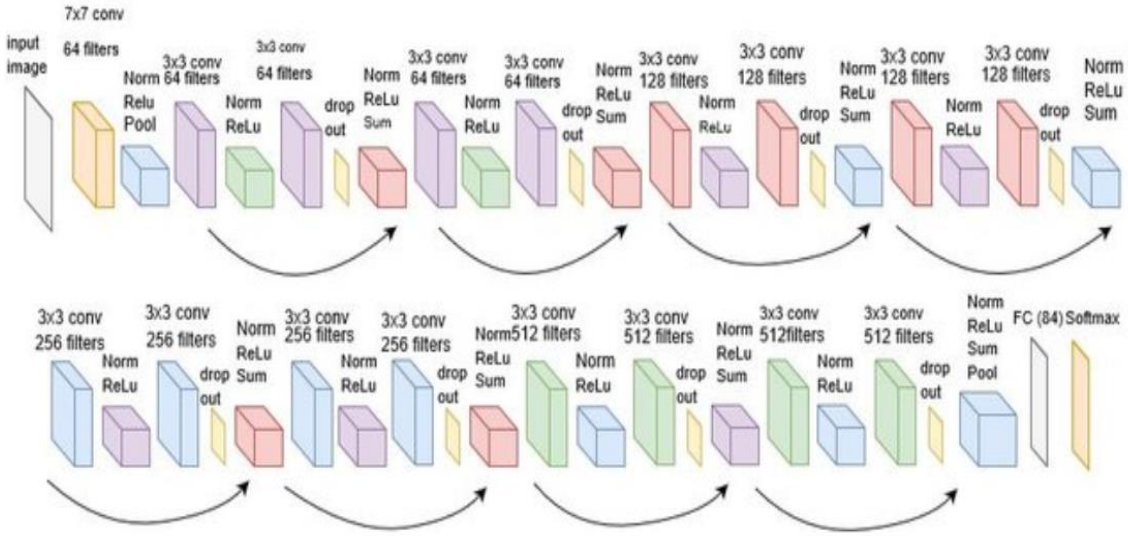


Figure 67: Resent-18 model

Part2 a)

Initialize the ResNet-18 network with pre-trained weights from ImageNet and then try to use these weights to improve the training for the CIFAR-10 dataset. Try to come up with different ways of using these weights to improve the performance and play with the hyper-parameters to get the best performance. Document the results of your experiments.

Training Model from Scratch

We have trained the Resnet -18 model from scratch. We have fixed the following parameters as follows:

- No of epochs = 50
- Learning rate = 0.05
- Optimizer = SGD
- Batch Size = 1024

PLOT OF LOSSES VS EPOCHS

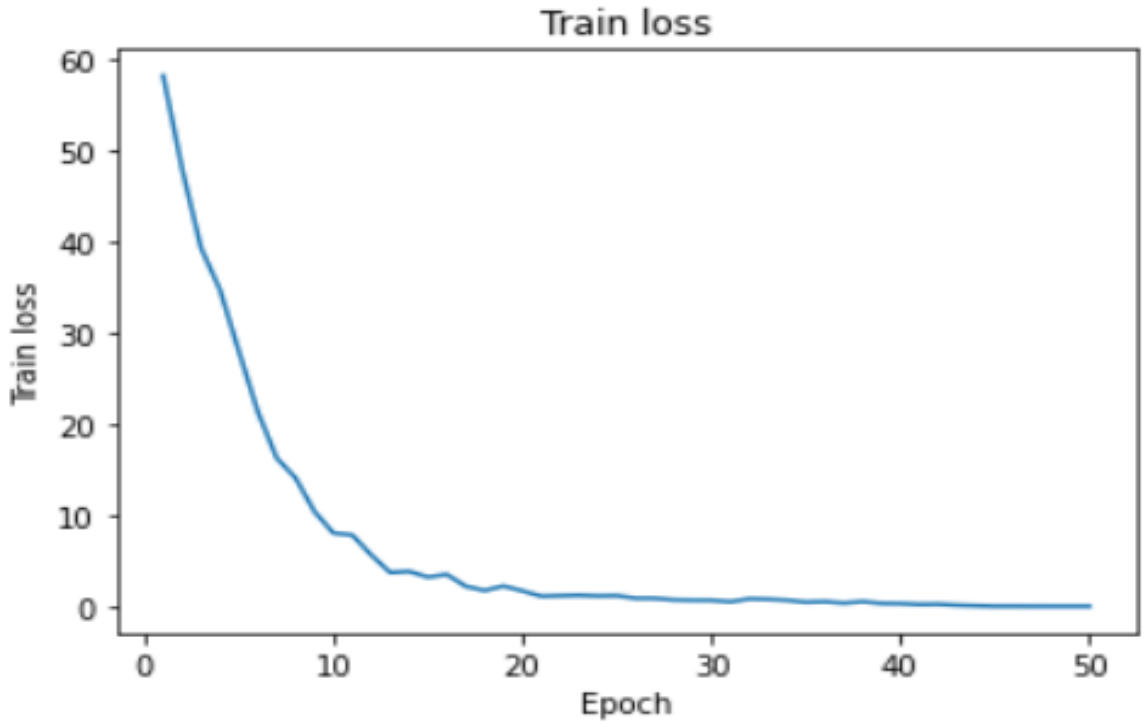


Figure 68: Training loss Vs Epoch

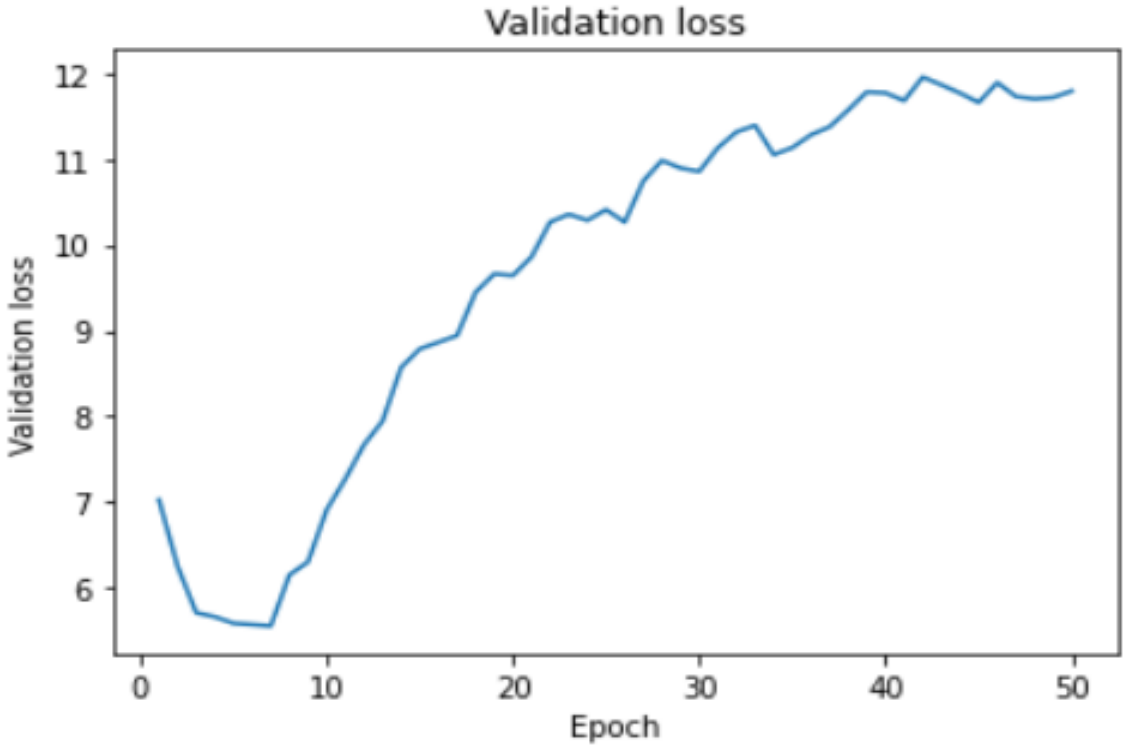


Figure 69: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

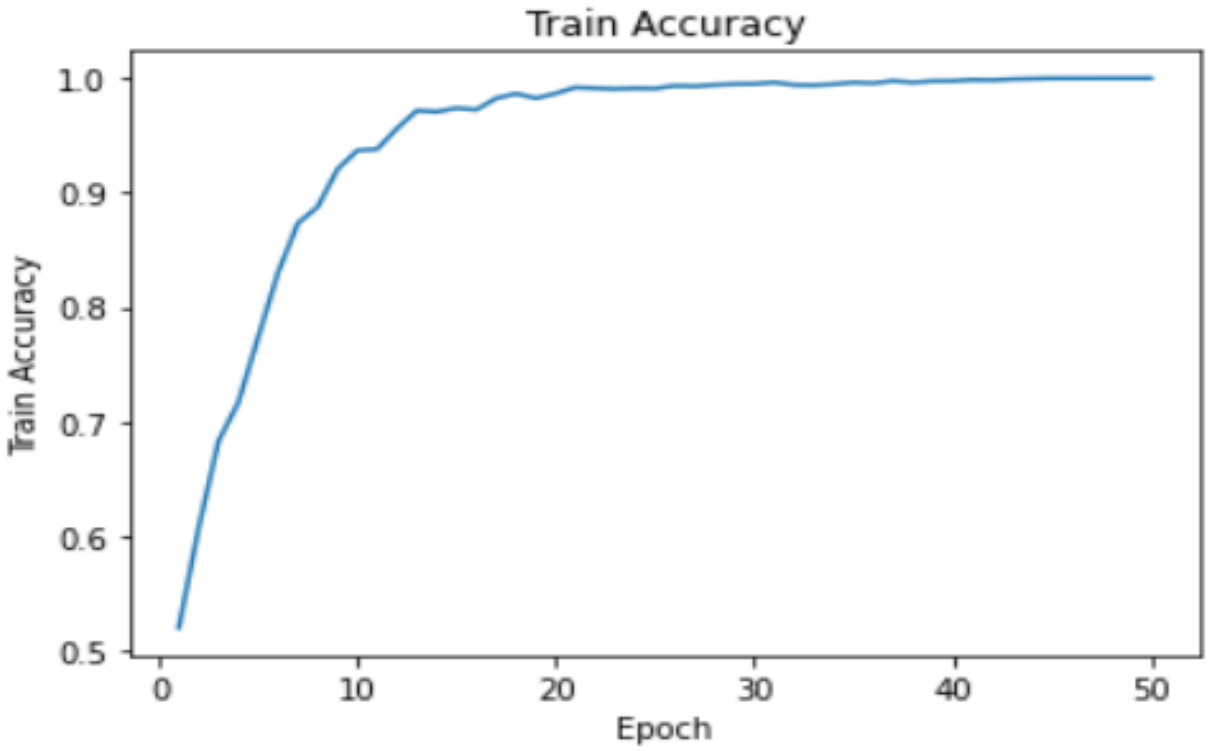


Figure 70: Training Accuracy Vs Epoch

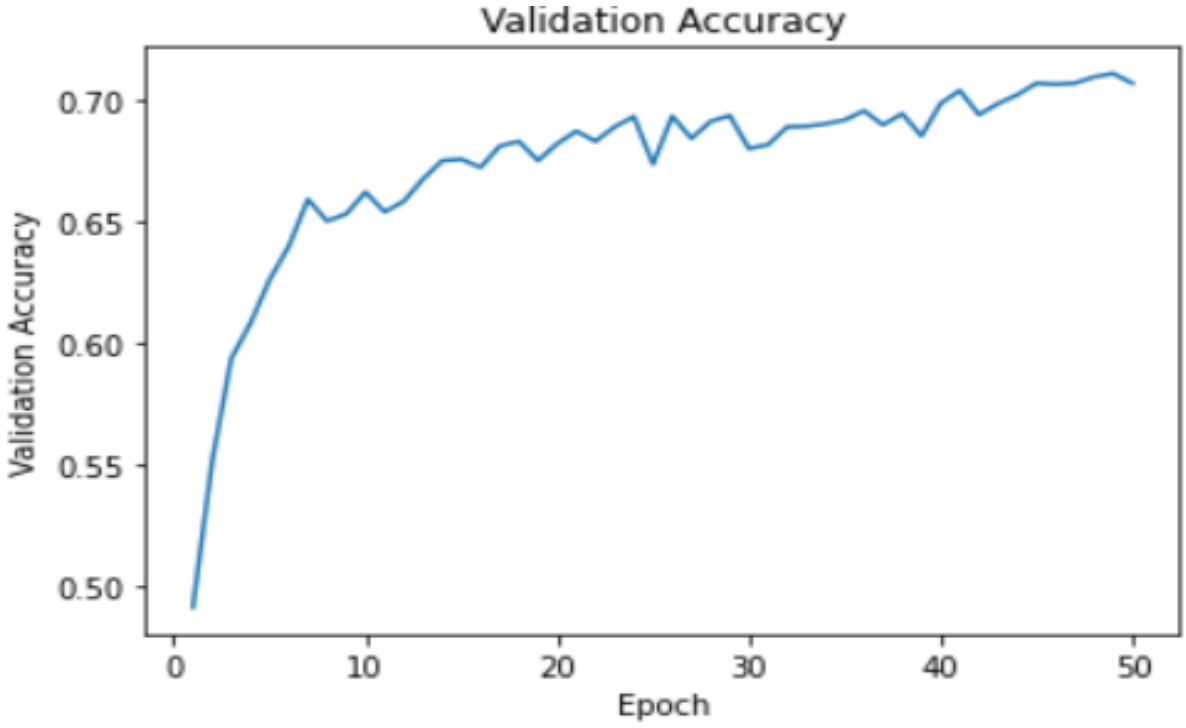


Figure 71: Validation Accuracy Vs Epoch

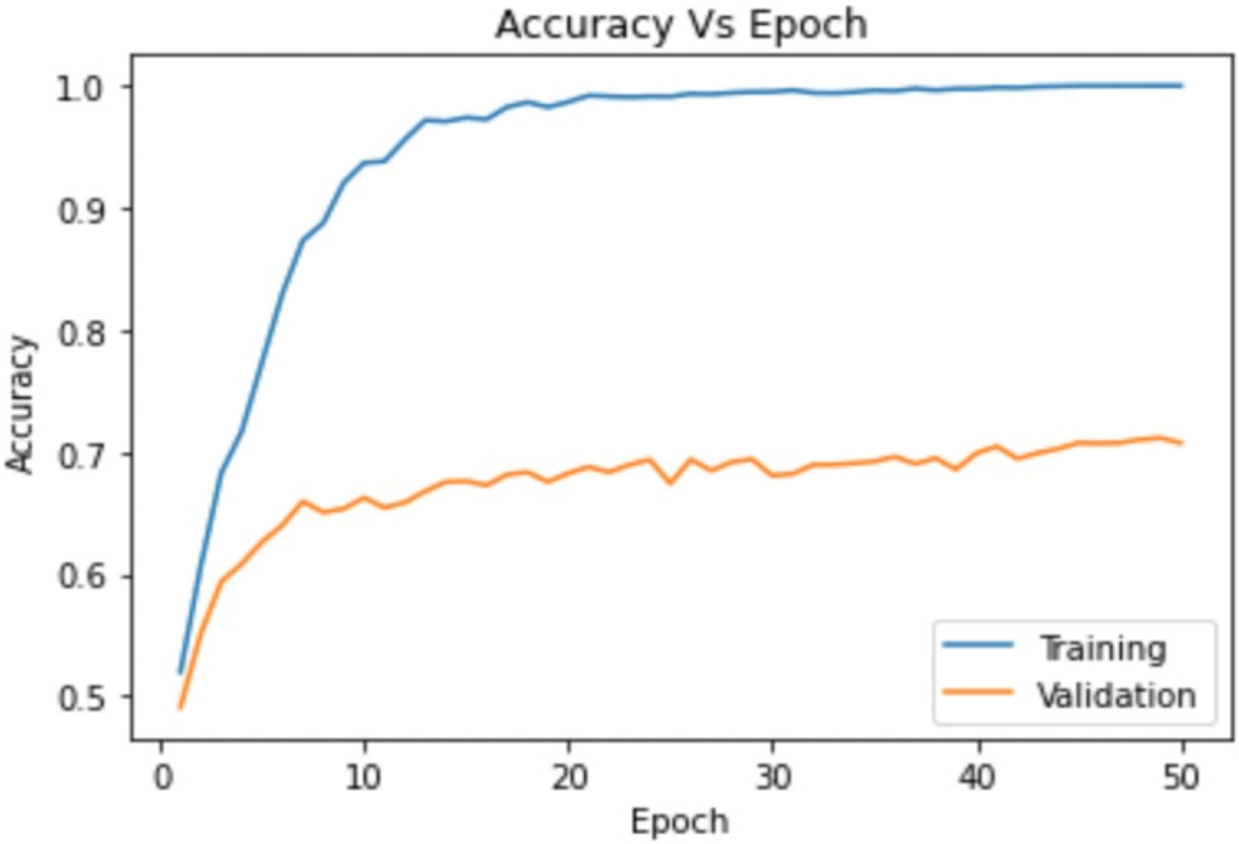


Figure 72: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 23.80

Testing Accuracy = 70.15%

Now we have use different ways of using these weights to improve the performance.

Method 1

Using Pretrained Model

We are using the pre-trained model of the Resnet -18 model for training.

PLOT OF LOSSES VS EPOCHS

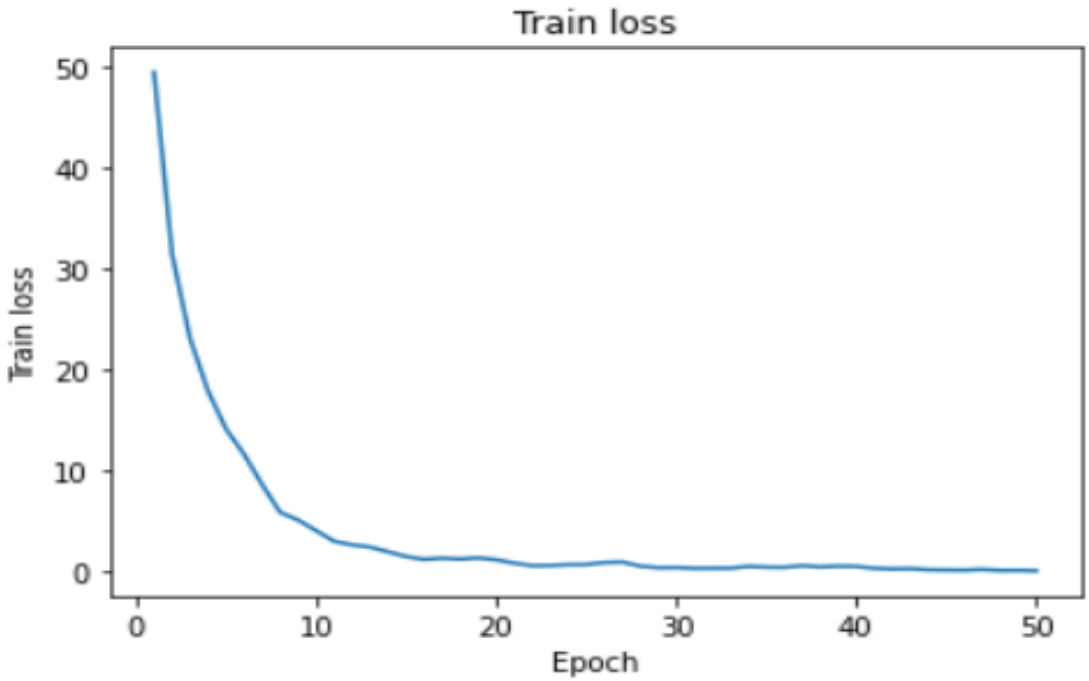


Figure 73: Training loss Vs Epoch

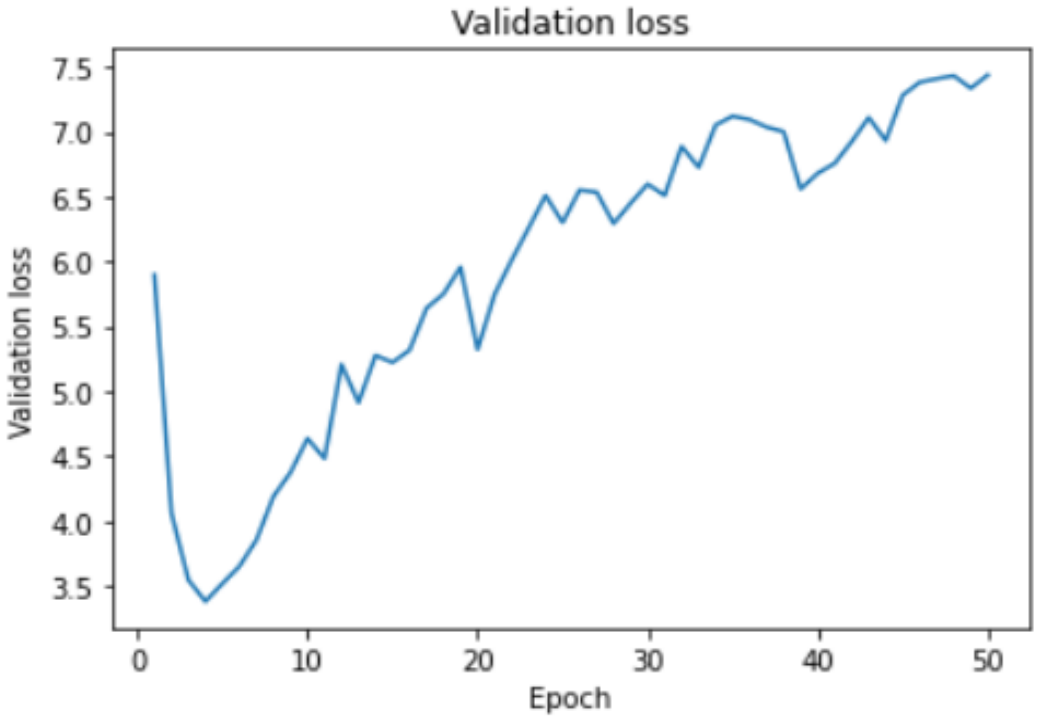


Figure 74: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

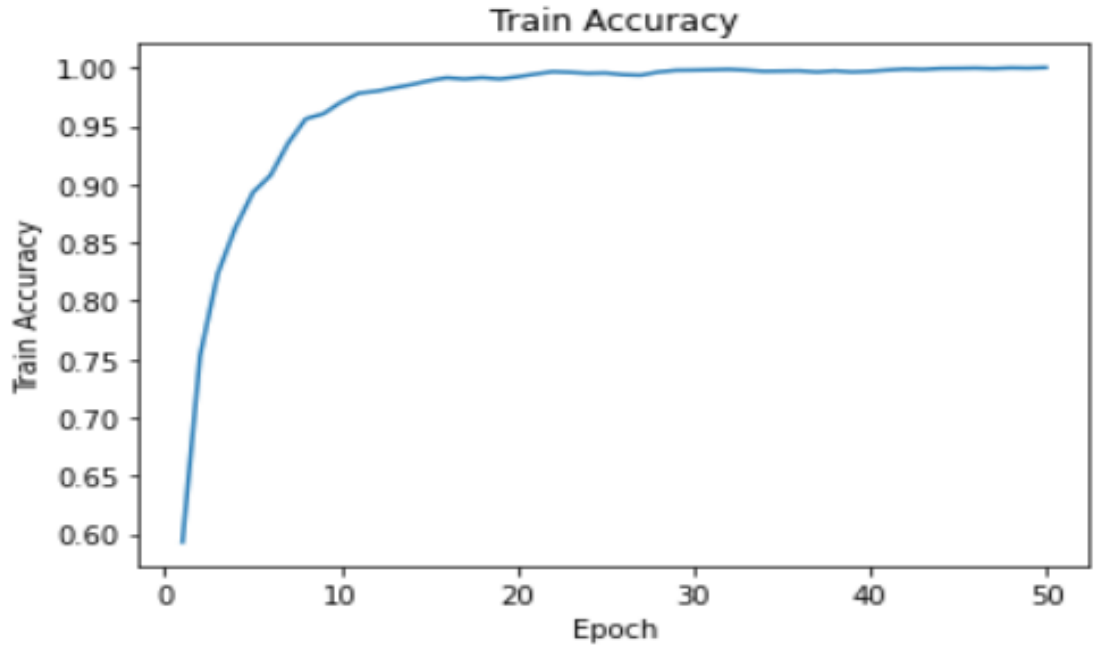


Figure 75: Training Accuracy Vs Epoch

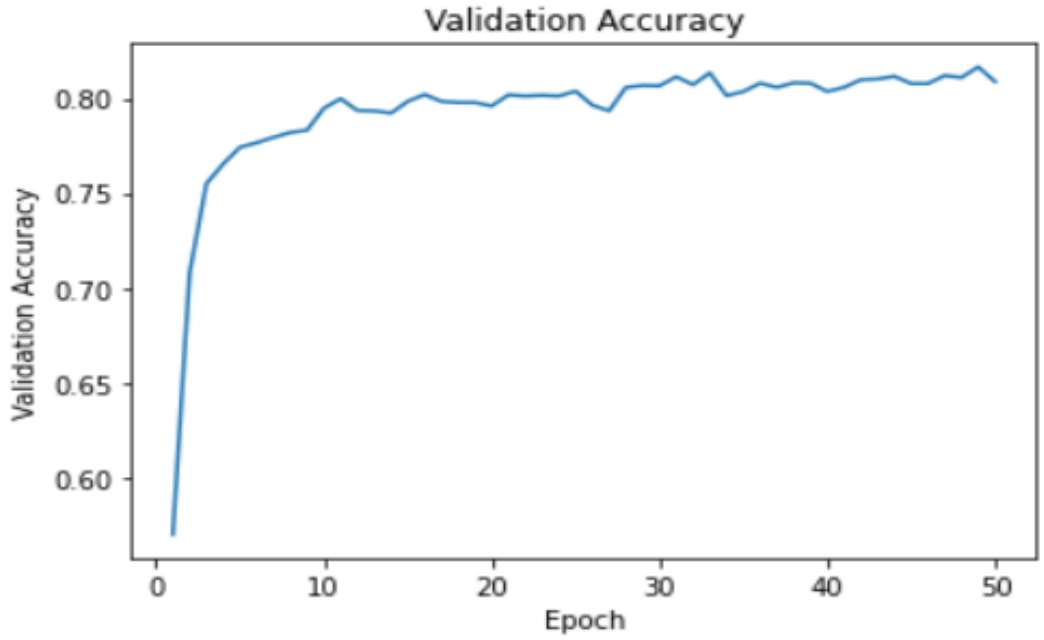


Figure 76: Validation Accuracy Vs Epoch

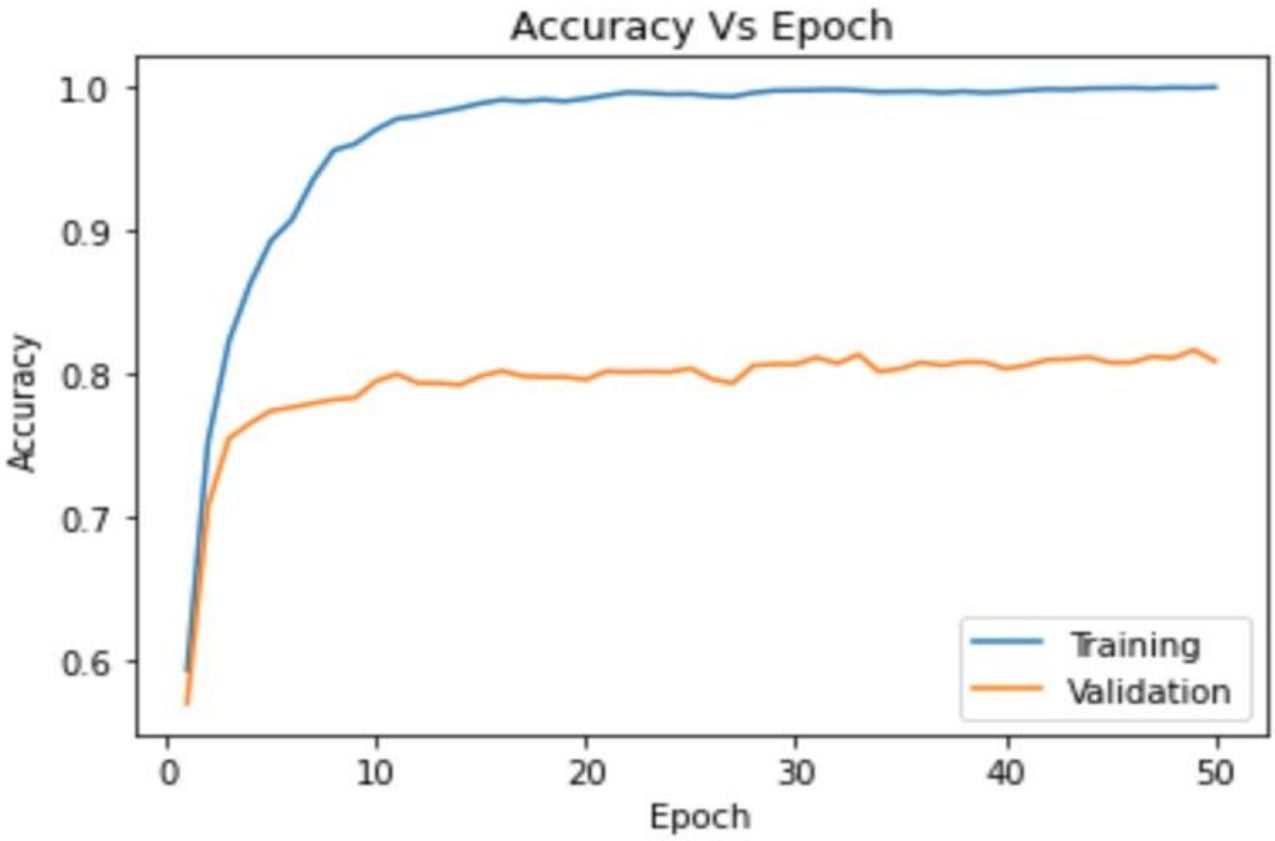


Figure 77: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 16.17

Testing Accuracy = 80.88%

Method 2

Using Pretrained Model and Training only final layer

PLOT OF LOSSES VS EPOCHS

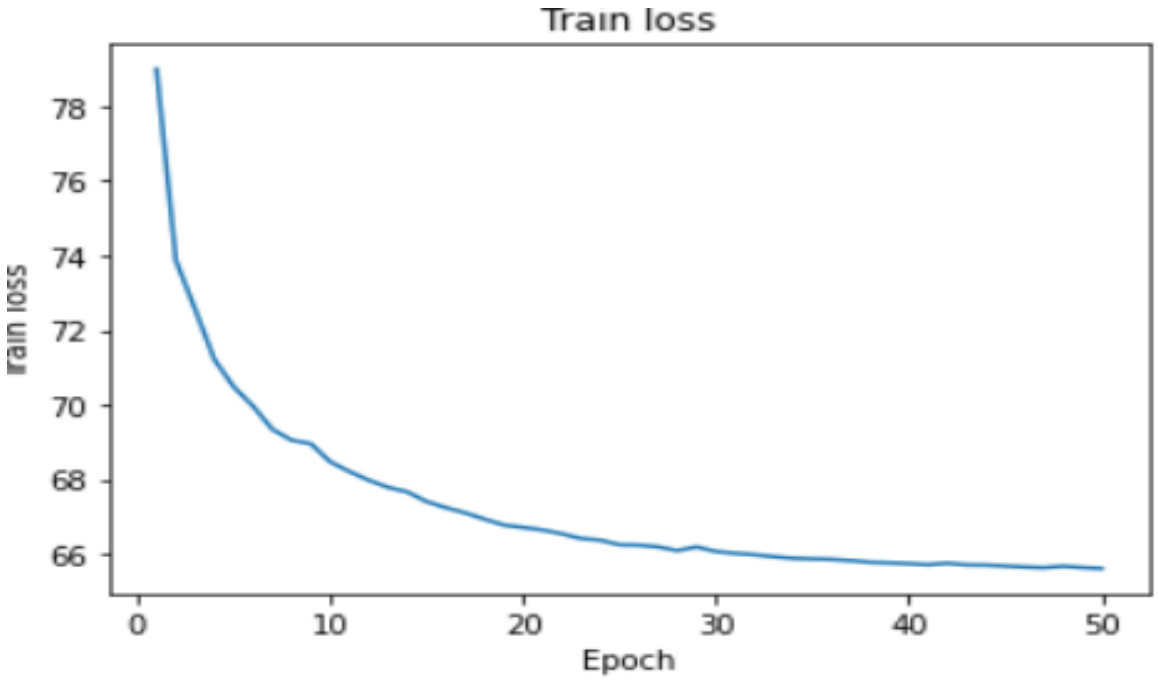


Figure 78: Training loss Vs Epoch

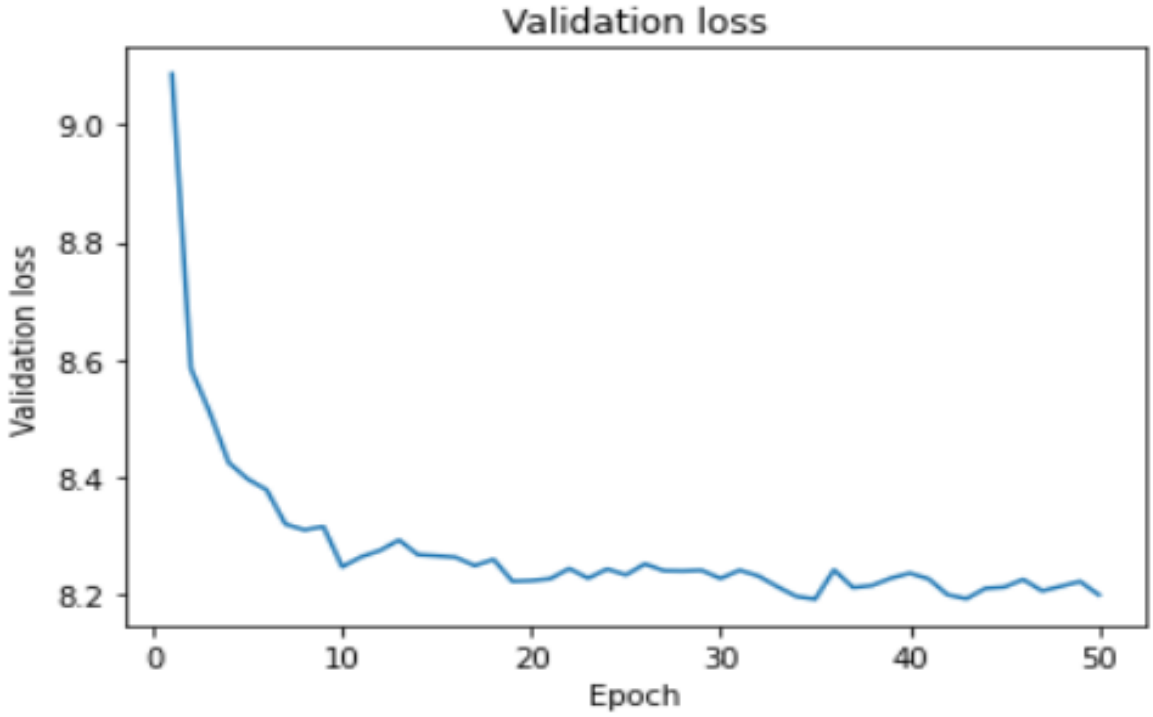


Figure 79: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

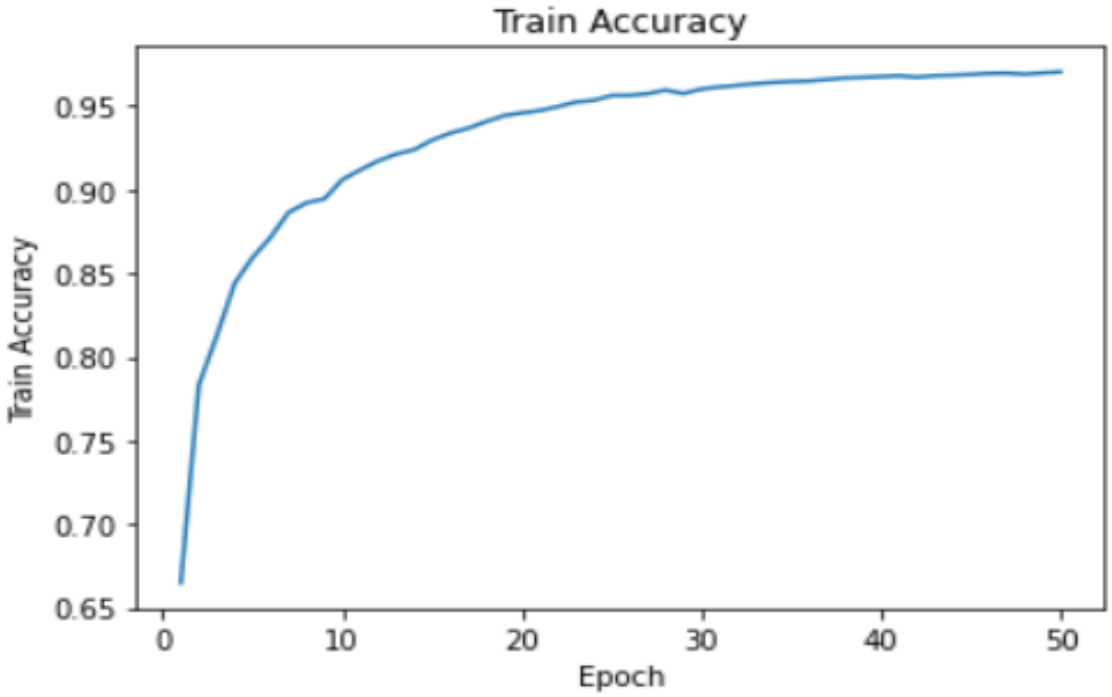


Figure 80: Training Accuracy Vs Epoch

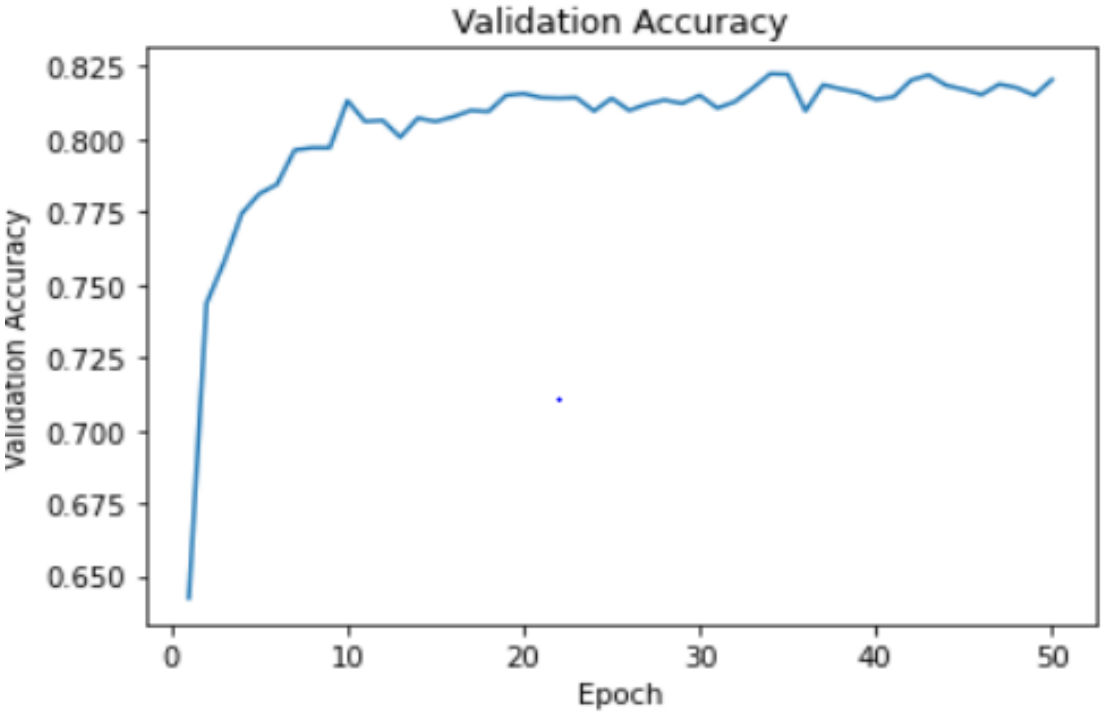


Figure 81: Validation Accuracy Vs Epoch

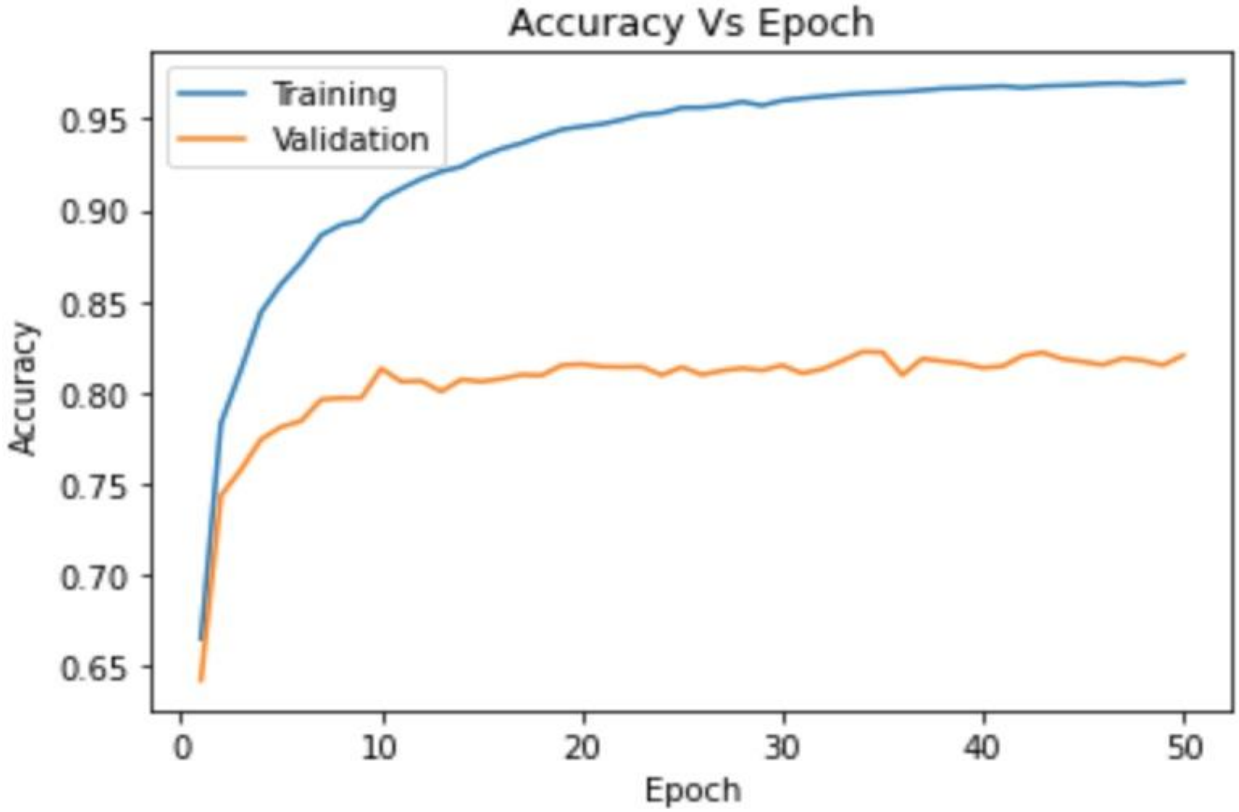


Figure 82: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 16.37

Testing Accuracy = 82.33%

RESULTS

Method	**Training Accuracy**	**Validation Accuracy**	**Testing Loss**	**Testing Accuracy**
Training from scratch	100	71.74	23.08	70.15
Training using pretrained model	100	81.66	16.17	80.88
Training only final and using pretrained model	97.03	82.04	16.37	82.33

Figure 83: Comparison

CONCLUSION

We see that training a model from scratch gave a testing accuracy of 70.15%, whereas using pre-trained models gave a boost of more than 10%. The training accuracy of training the complete pre-trained model (100%) was higher than training just the final layer of a pre-trained model (97.03%), whereas the testing accuracies were opposite (80.88% and 82.33% respectively). This shows that training the complete pre-trained model caused more overfitting than training only the last layer. We can also conclude that all models are overfitted since the training accuracies are much higher than the testing accuracies.

PART2 B)

Train the network from scratch with the Tiny-CIFAR-10 dataset. Try using as many data augmentation techniques as you can think of to try to improve the performance. Try dropout after different layers and with different dropout rates.

Training Without Augmentation

We have trained the Resnet -18 model from scratch without Augmentation.

PLOT OF LOSSES VS EPOCHS

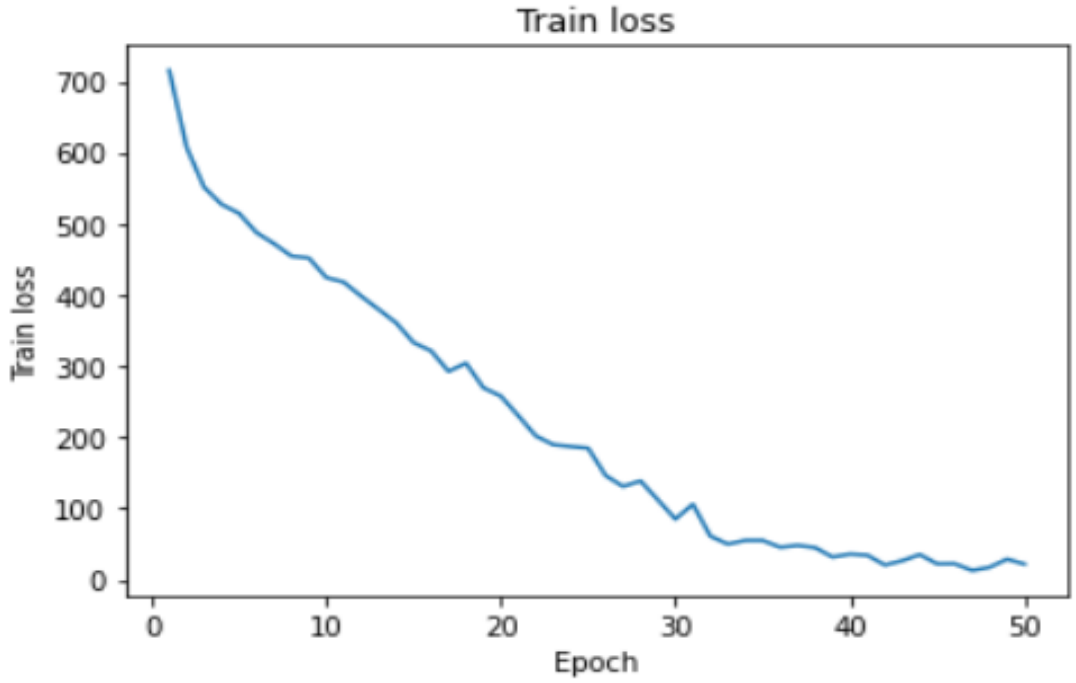


Figure 84: Training loss Vs Epoch

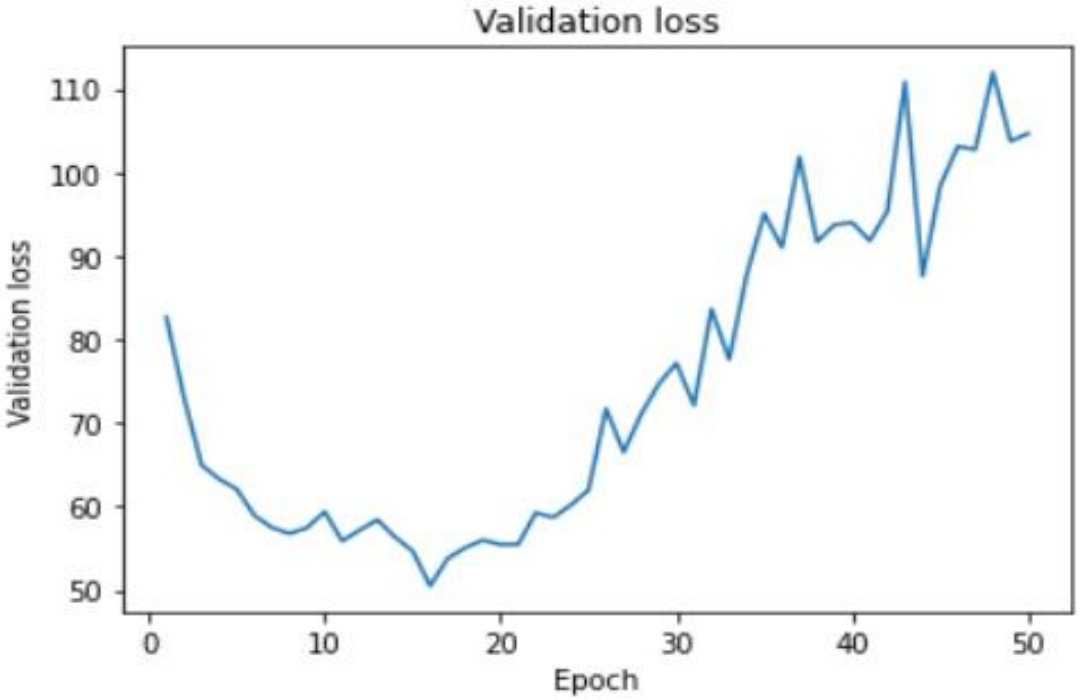


Figure 85: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

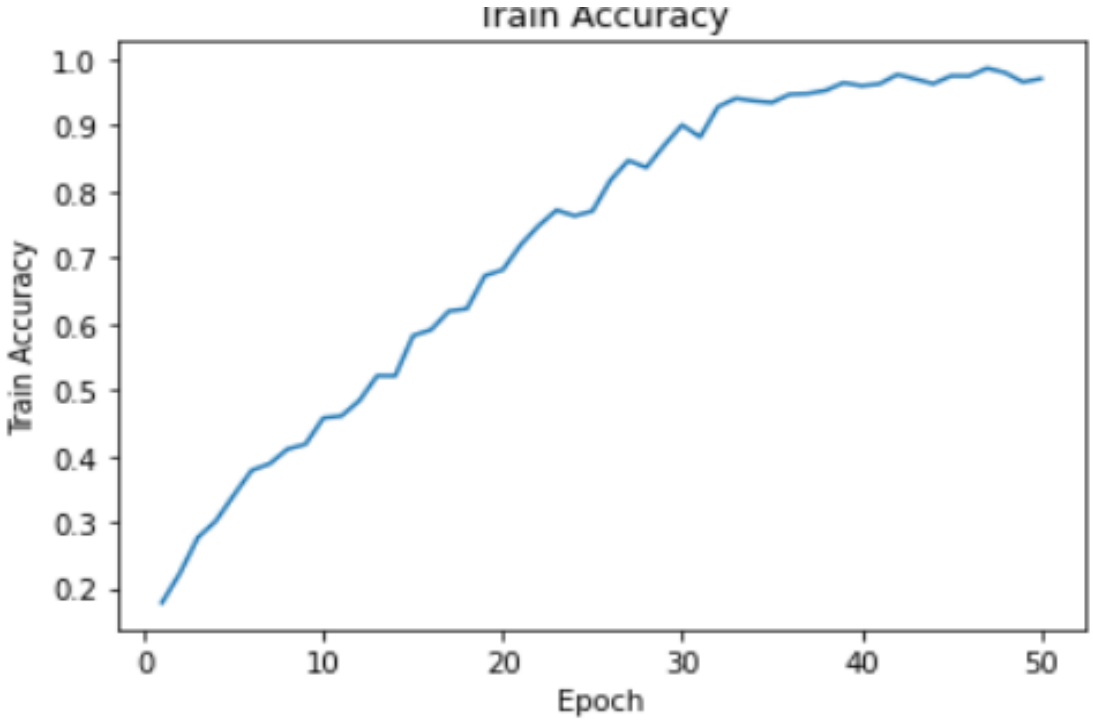


Figure 86: Training Accuracy Vs Epoch

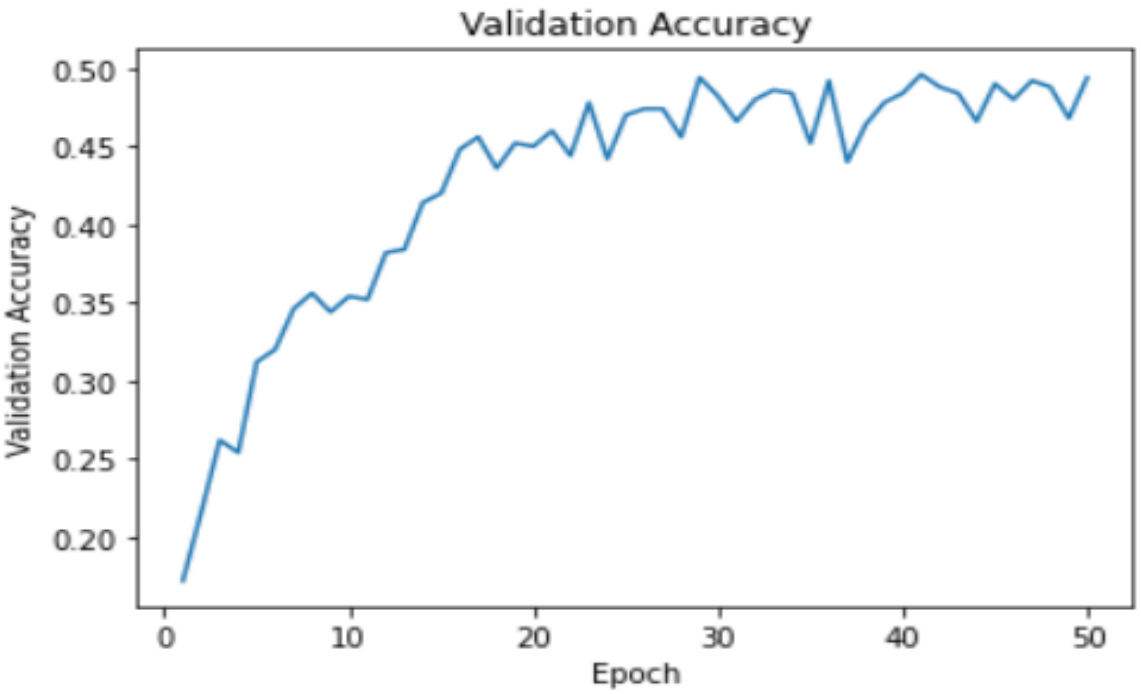


Figure 87: Validation Accuracy Vs Epoch

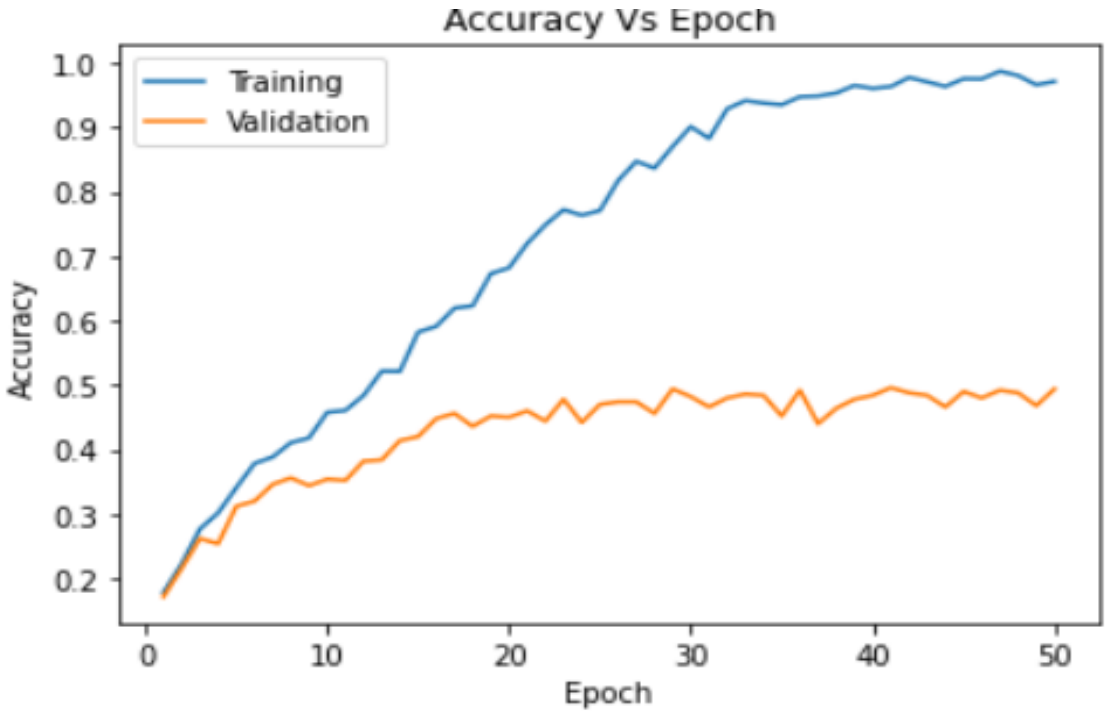


Figure 88: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 2021.22

Testing Accuracy = 50.46

Training With Augmentation

We have trained the Resnet -18 model from scratch using Augmentation.

Augmentation methods

- **RandomResizedCrop**: Crop a random portion of image and resize it to a given size.
- **RandomHorizontalFlip**: Horizontally flip the given image randomly with a given probability.
- **RandomAffine**: Random affine transformation of the image keeping center invariant.
- **ColorJitter:** Randomly change the brightness, contrast, saturation and hue of an image.
- **Normalize:** Normalize a tensor image with mean and standard deviation.

ORGINAL IMAGE

Image after Random Horizontal Flip

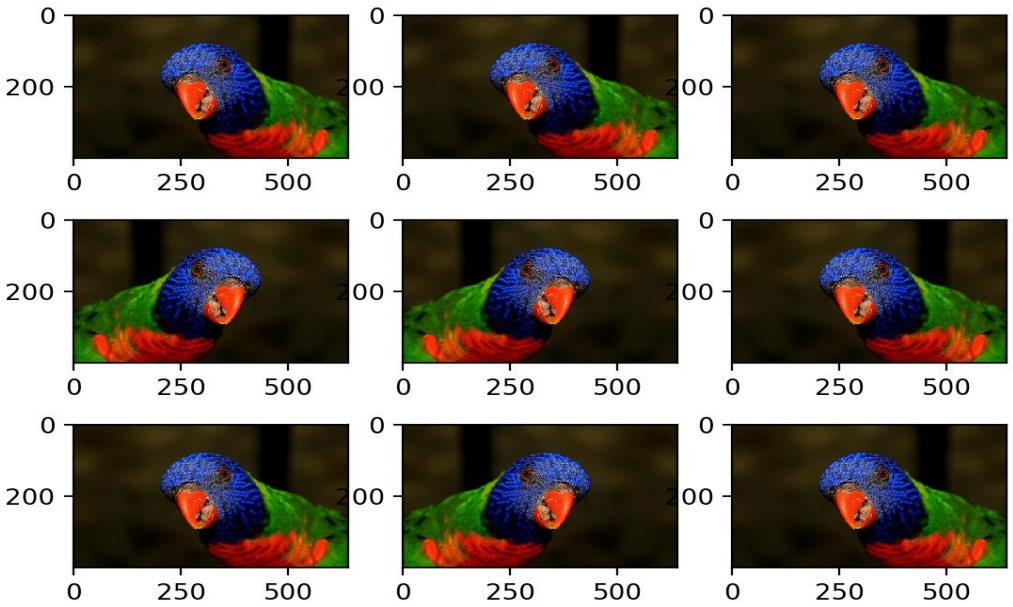
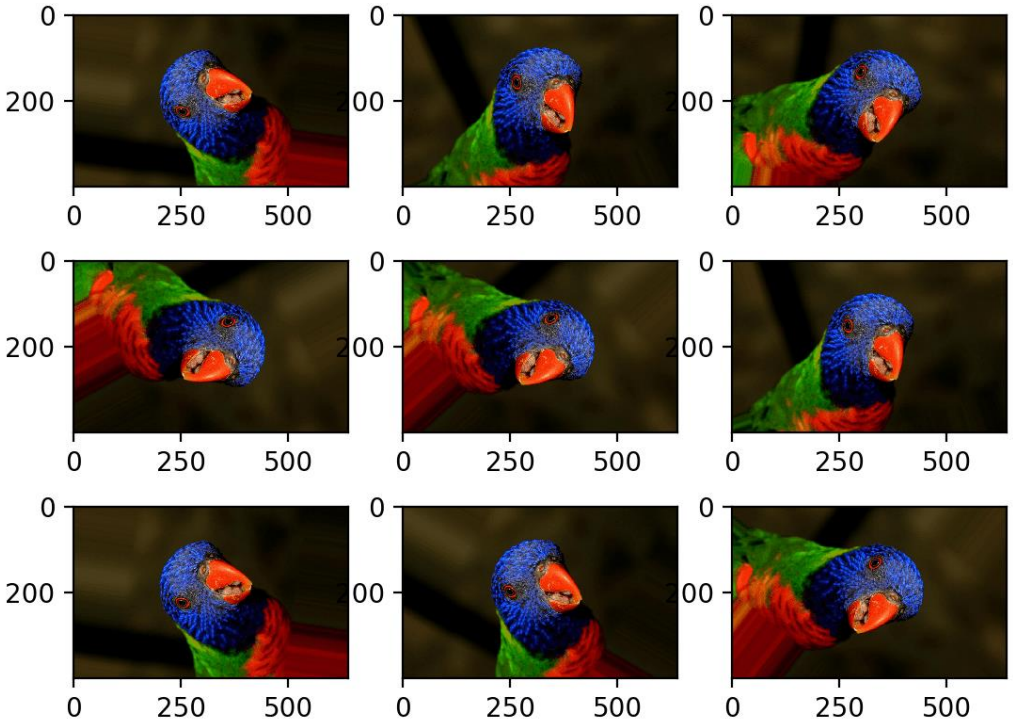
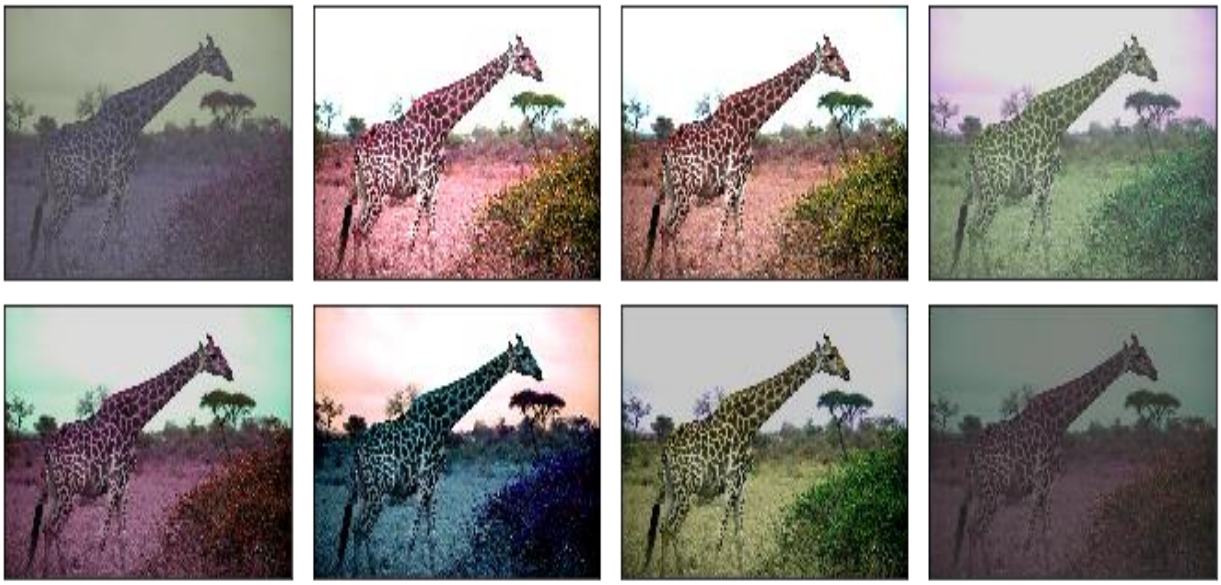


Image after Random Rotation



Colour Jitter



PLOT OF LOSSES VS EPOCHS

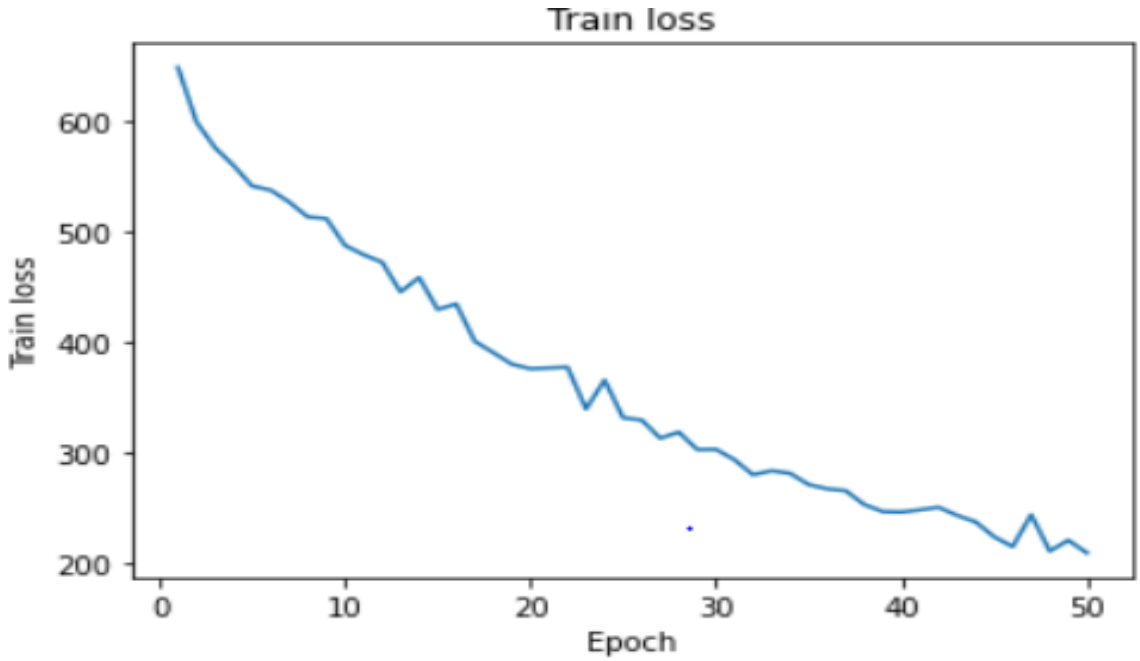


Figure 89: Training loss Vs Epoch

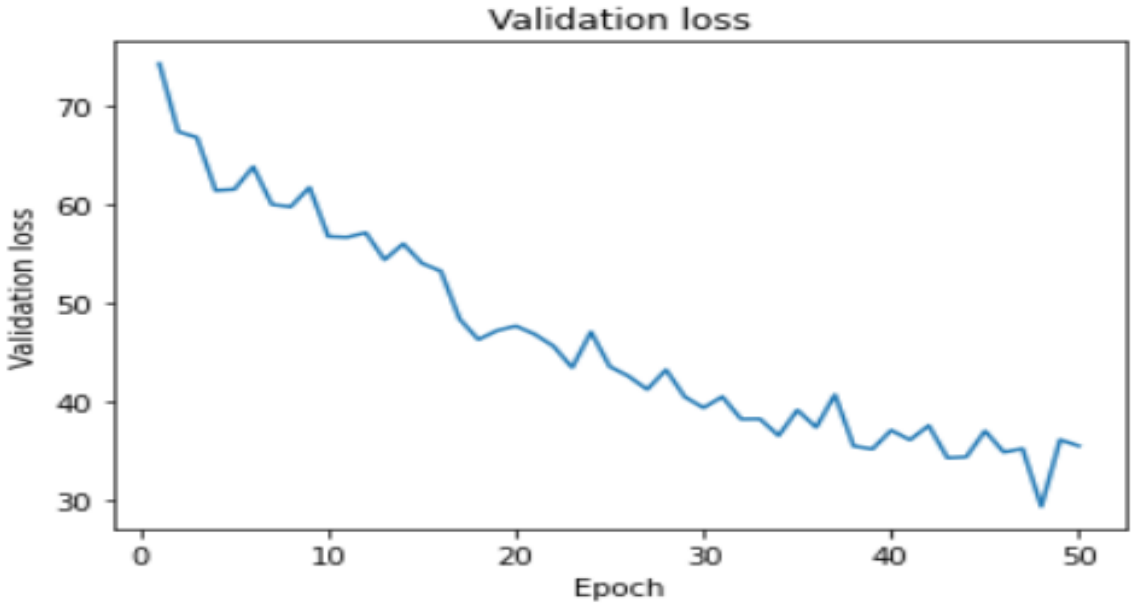


Figure 90: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

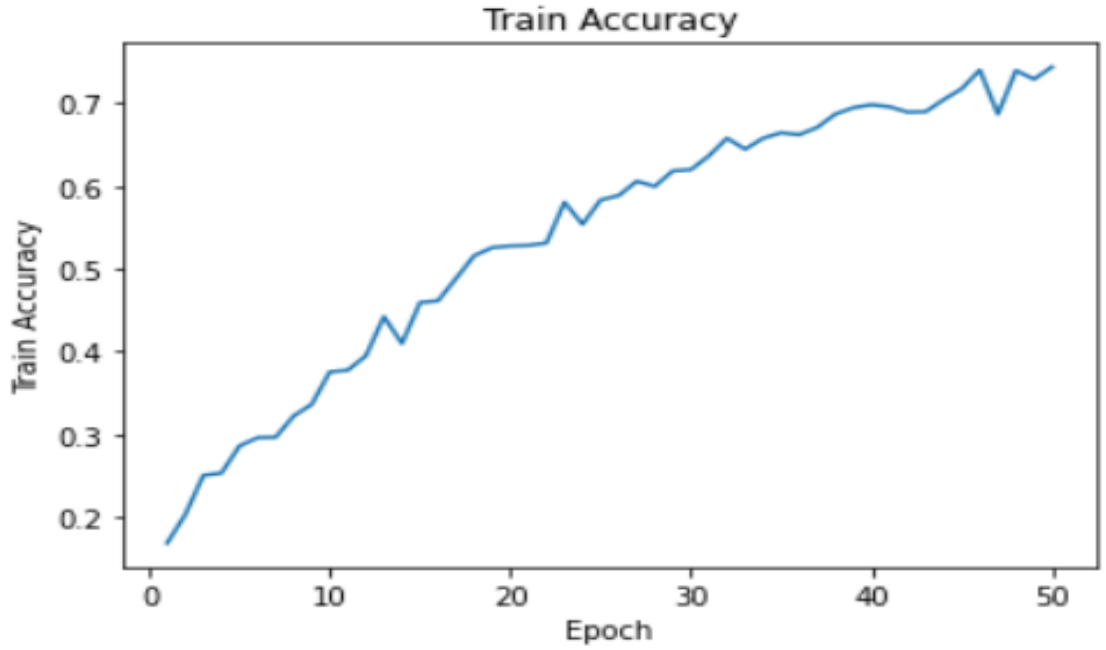


Figure 91: Training Accuracy Vs Epoch

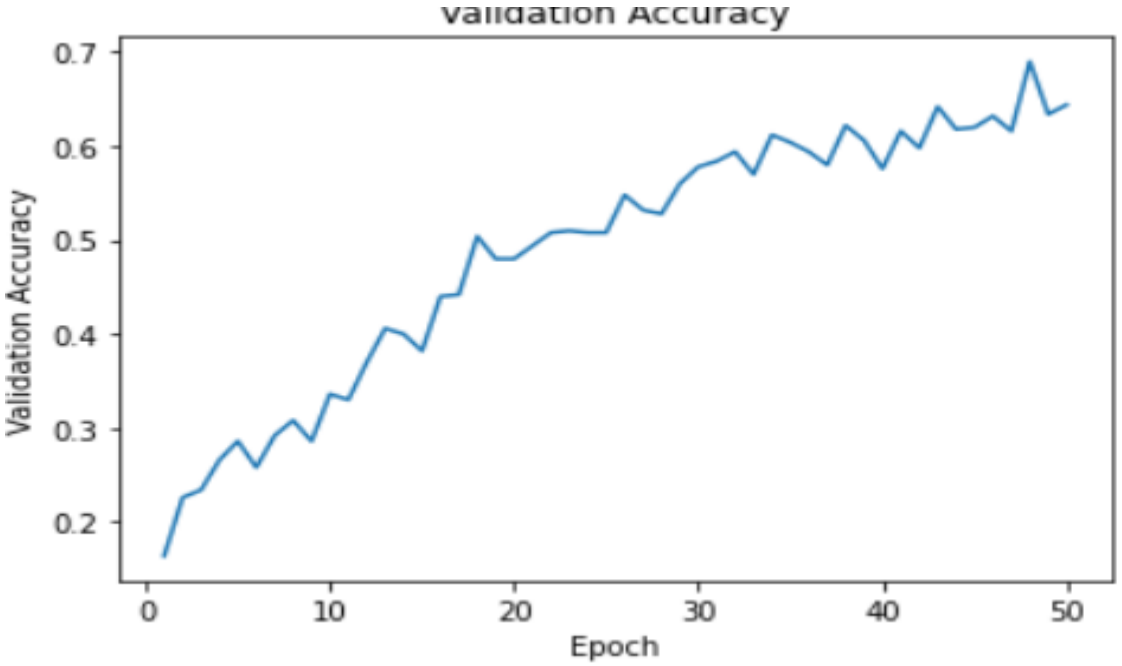


Figure 92: Validation Accuracy Vs Epoch

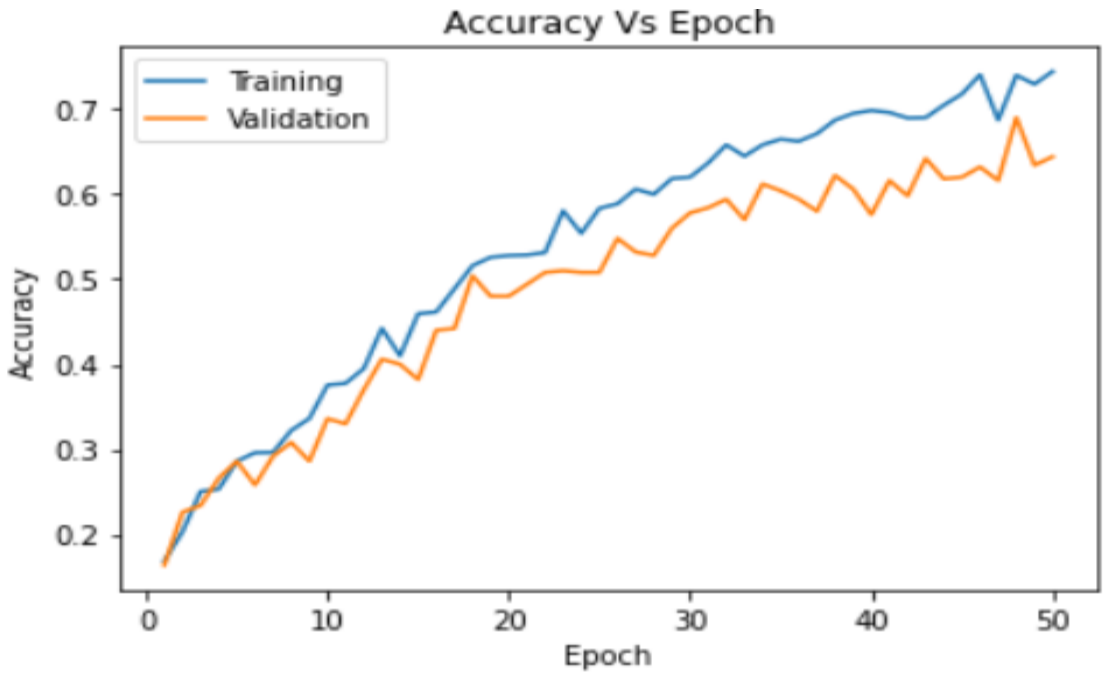


Figure 93: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 499.12

Testing Accuracy = 73.34

Trying dropout after different layers and with different dropout rates

ADDING DROPOUT VARIOUS LAYERS

Dropout = 0.5

PLOT OF LOSSES VS EPOCHS

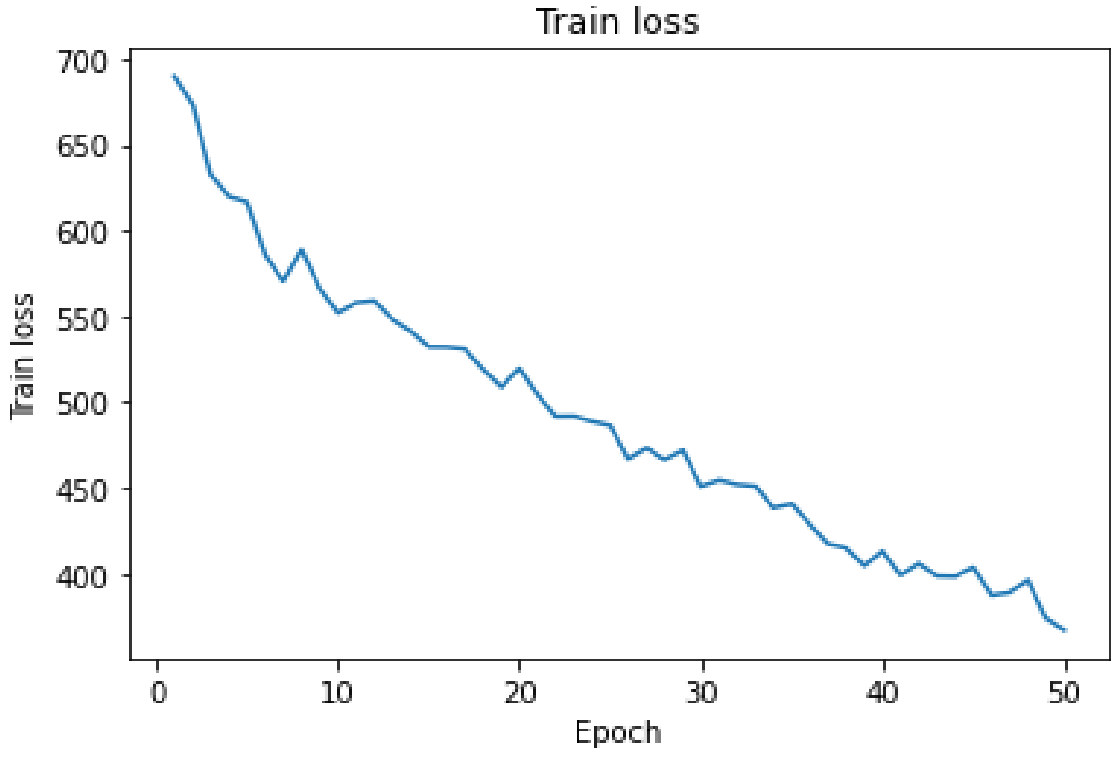


Figure 94: Training loss Vs Epoch

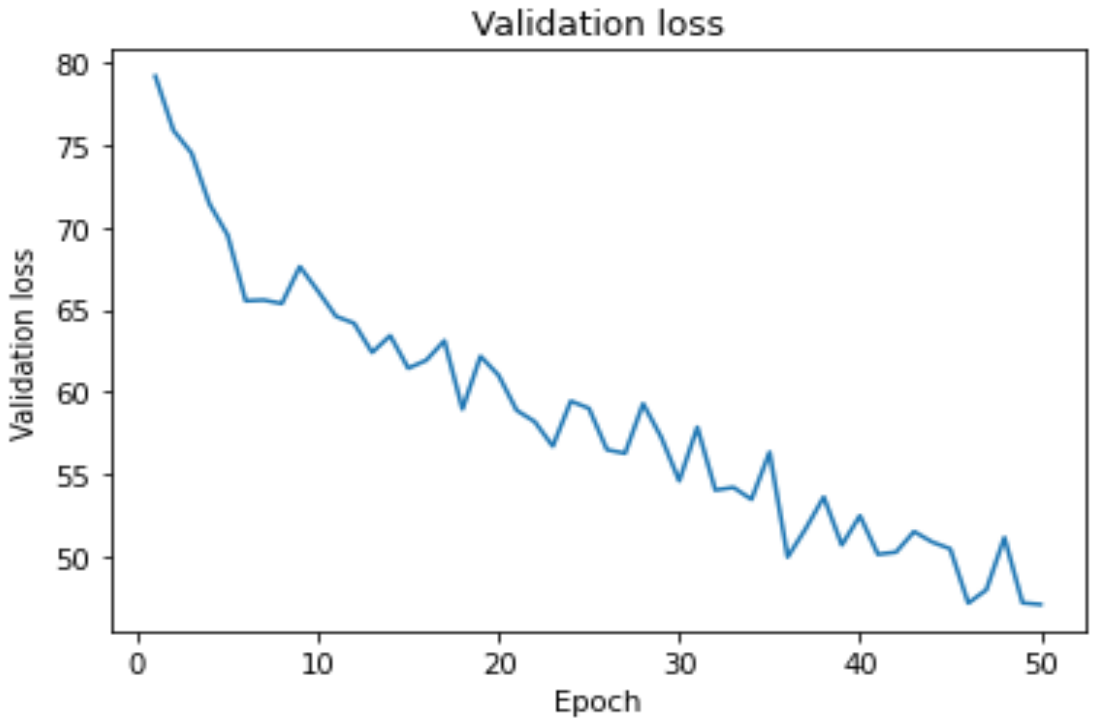


Figure 95: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

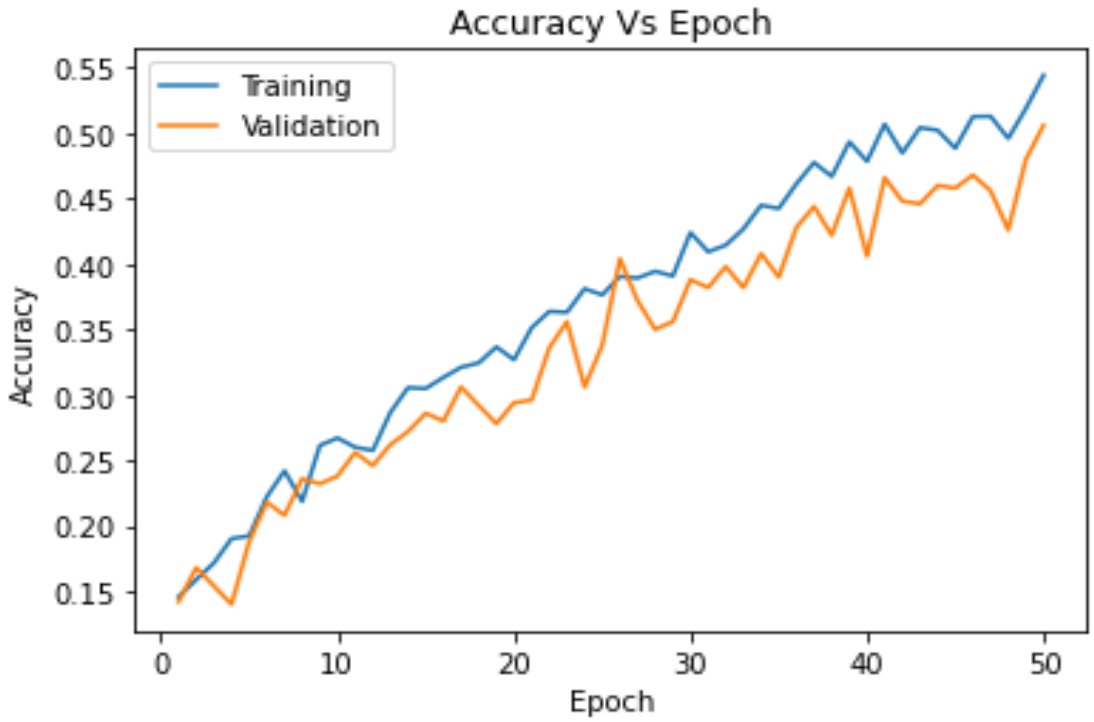


Figure 96: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 751

Testing Accuracy = 58.16

Dropout = 0.8

PLOT OF LOSSES VS EPOCHS

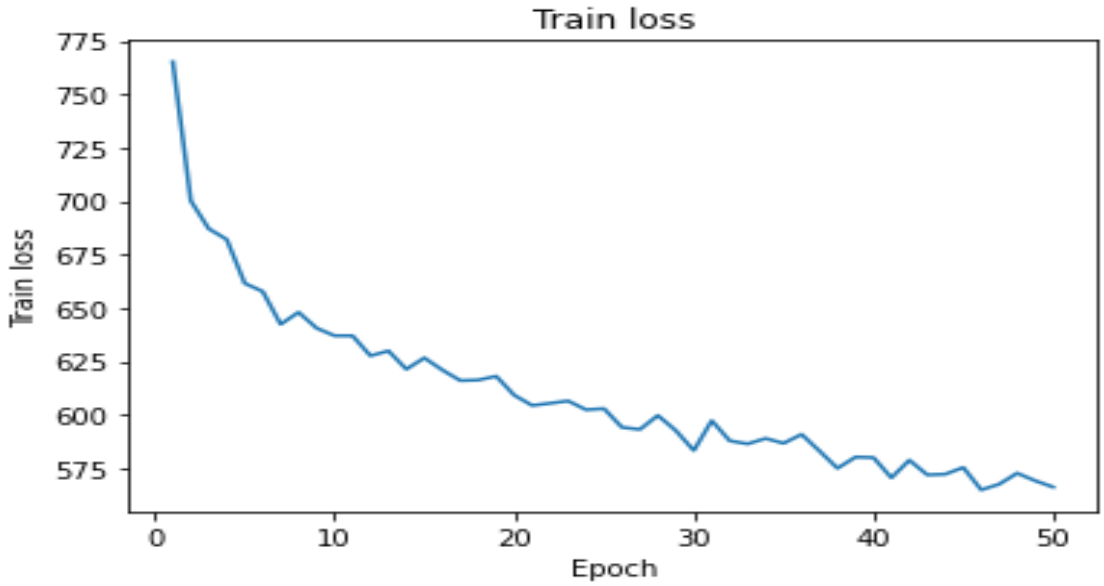


Figure 97: Training loss Vs Epoch

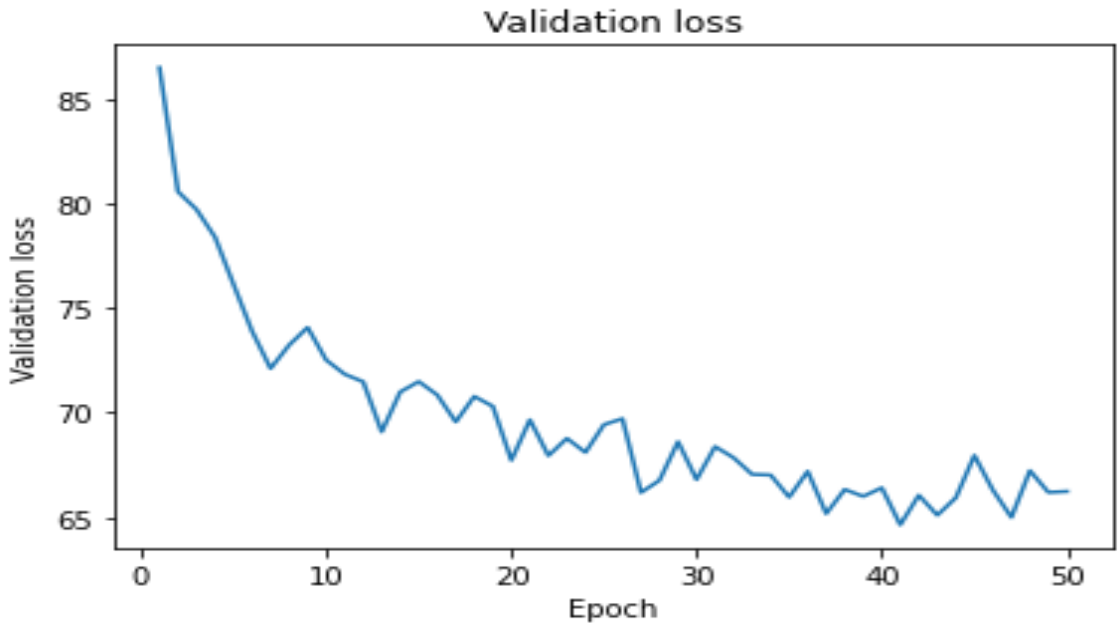


Figure 98: Validation loss Vs Epoch

PLOT OF ACCURACY VS EPOCHS

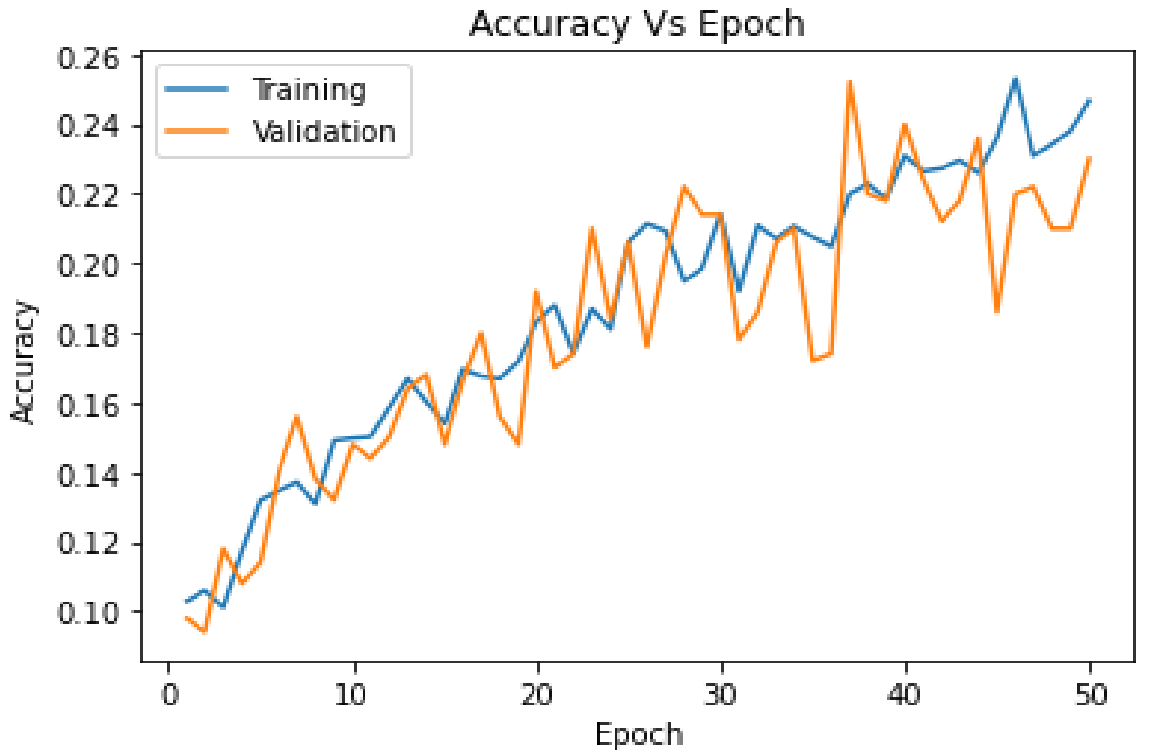


Figure 99: Training and Validation Accuracy Vs Epoch

TESTING ACCURACY AND TESTIING LOSS

Testing loss = 1201.11

Testing Accuracy = 27.51

RESULTS

Method	**Training Accuracy**	**Validation Accuracy**	**Testing Loss**	**Testing Accuracy**
Training without augmentation	100	52.06	2021	50.46
Training with augmentation	76	75.08	499.12	73.34

Figure 100: Comparison

Training with augmentation and varying dropout

Dropout	Training Accuracy	Validation Accuracy	Testing Loss	Testing Accuracy
0.5	54.22	51	751	58.16
0.8	24.67	23.00	1201.11	27.51

Figure 101: Comparison

CONCLUSION

We find that without augmentation, the training accuracy is 100% and the testing accuracy is 50.46%. This strongly suggests that there is high overfitting here. This is supported by the fact that the TinyCIFAR dataset is very small.

After using data augmentation, we get a training accuracy of 76% and a testing accuracy of 73.34%. We get a worse training accuracy but a boost in testing accuracy of more than 23%. There isn't much overfitting here.

We tried to apply dropout in the 2nd and 4th layers of the model with 2 different p values: 0.5 and 0.8. We see that now the training and validation accuracies closely follow each other, suggesting minimum overfitting. But this also leads to low testing accuracies along with low training accuracies. With p=0.5, the training and testing accuracies are 54.22% and 58.16% and with p=0.8, the training and testing accuracies are 24.67% and 27.51%. This shows that even though dropout can reduce overfitting, it can also lead to lower accuracies.

Hence in our experiments, data augmentation yielded the best results.

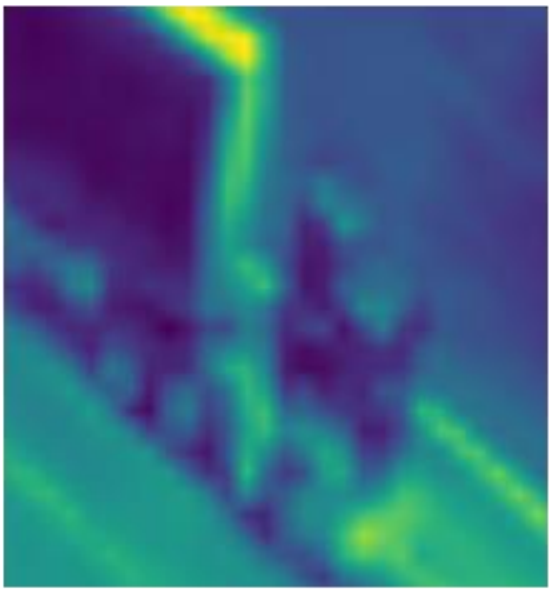
PART2 C)

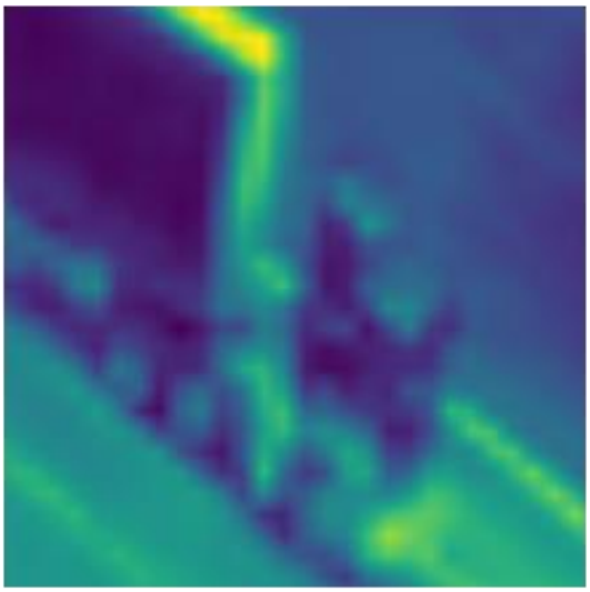
Visualize the activations of the CNN for a few test examples in each of the above cases. How are the activation in the first few layers different from the later layers

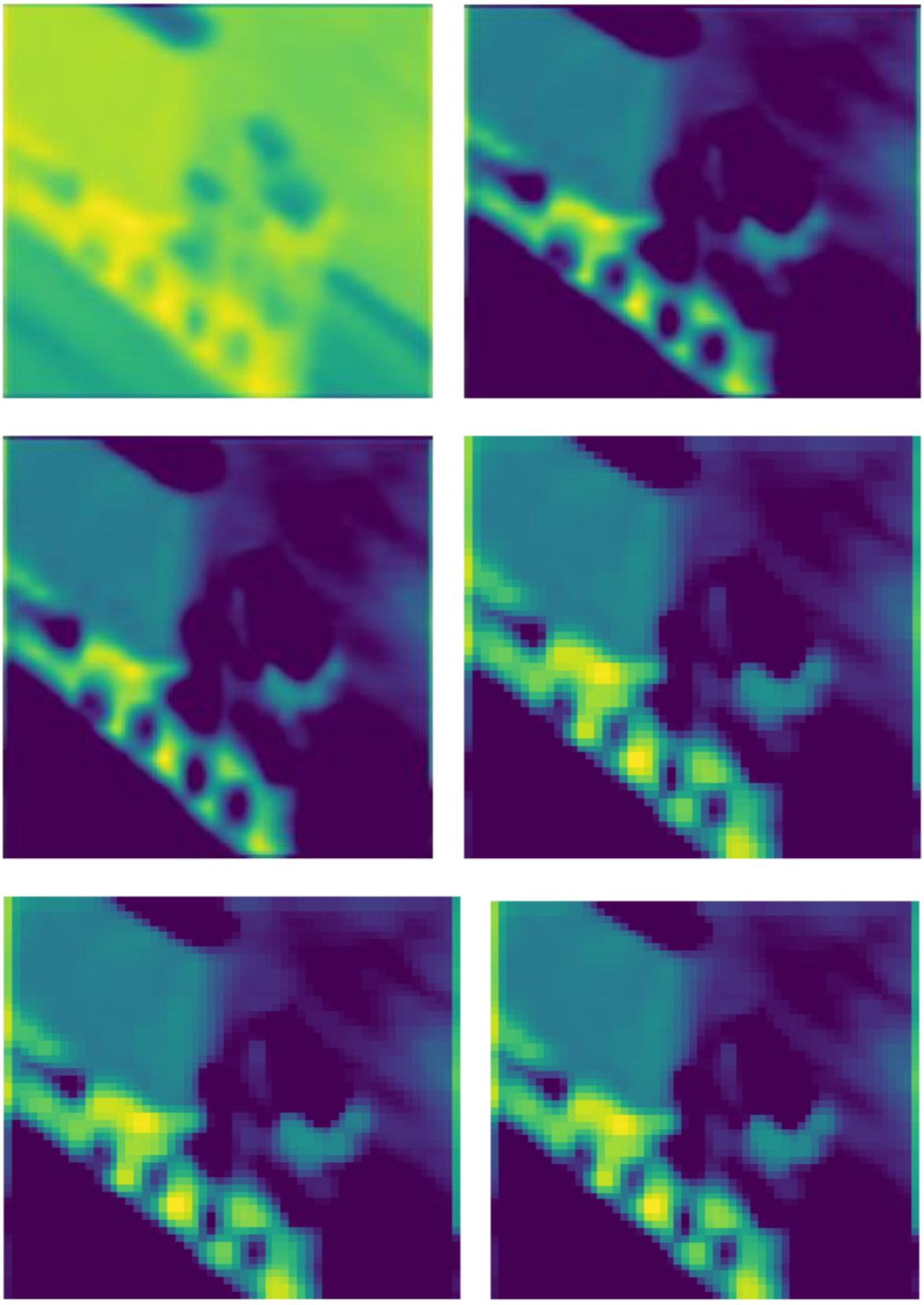
In order to examine the functioning of various convolutional layers, we stored the output images produced by each layer and proceeded to depict the output tensor by compressing and plotting it. In this regard, we present the visualization of CNN activation generated by a pre-trained ResNet model which we fine-tuned by training only the final layer, when applied to an image of a truck. As we can observe, the earlier layers of the model are only able to identify simple features like straight lines and curves. Nevertheless, as we delve deeper into the

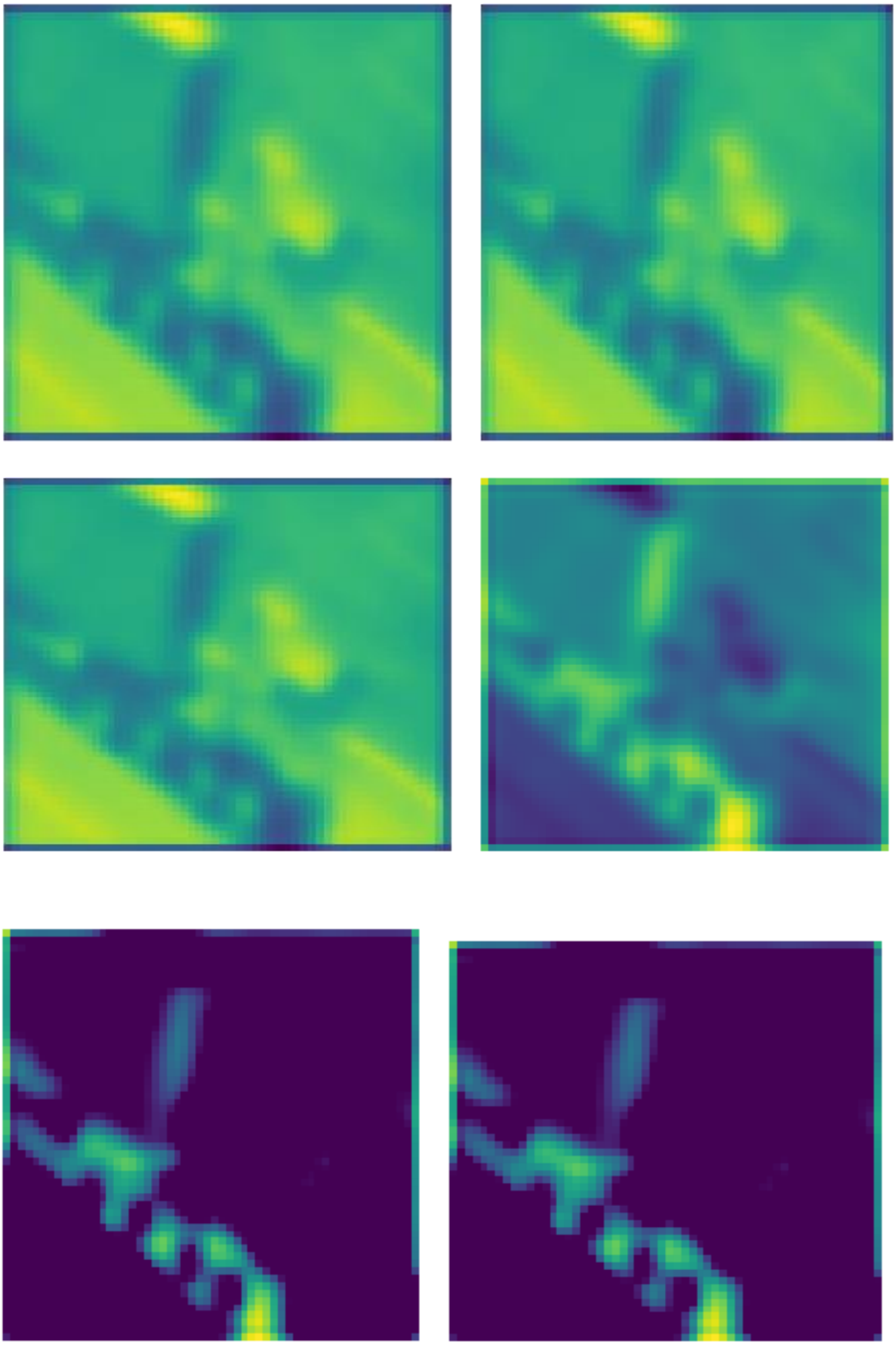
subsequent layers, the model is able to discern a boundary between the truck and its surrounding environment.

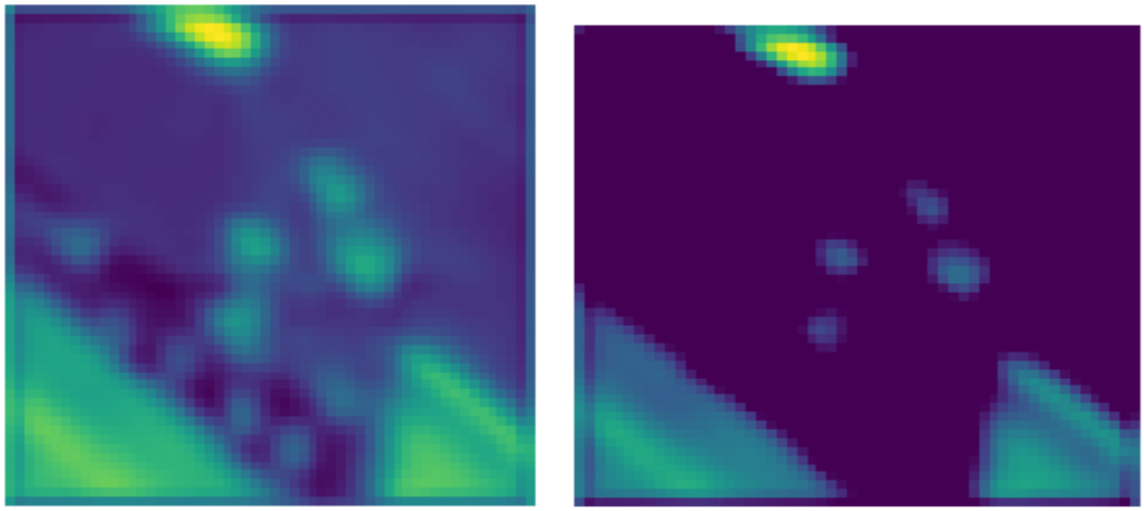
Figure 102: Original Image











Displayed below is the CNN activation visualization of a ResNet model trained from scratch on the TinyCIFAR dataset with data augmentation, on an image of a dog.

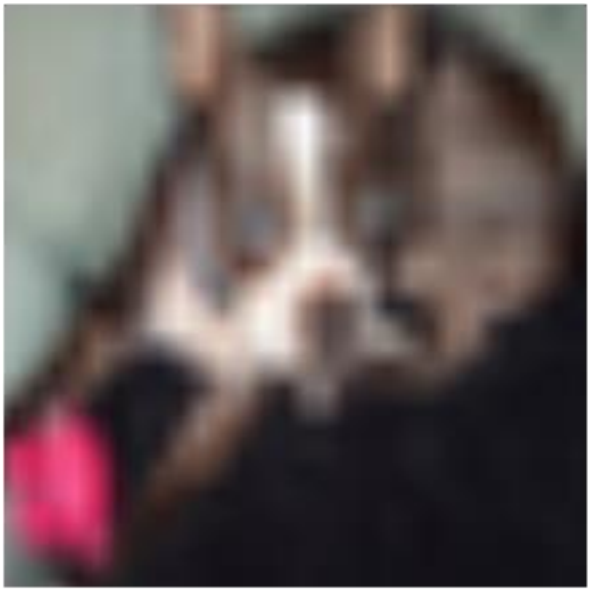
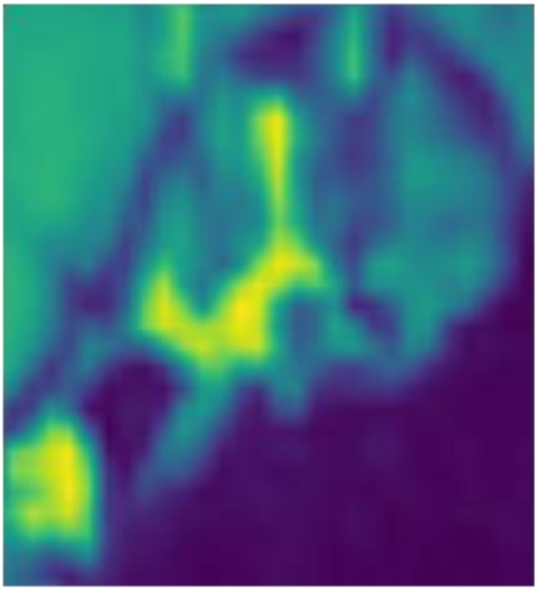
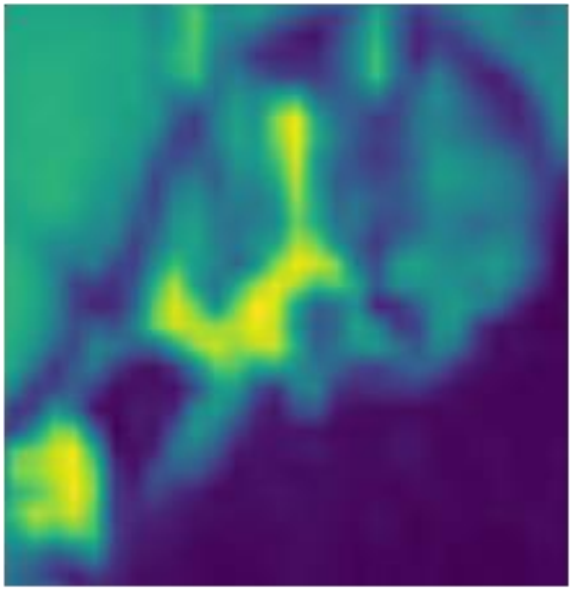
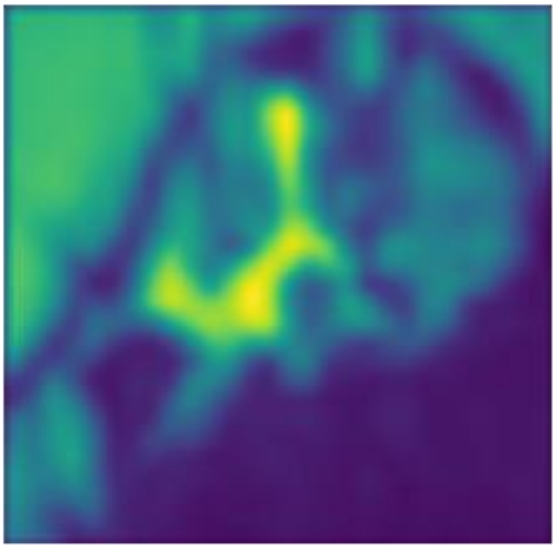
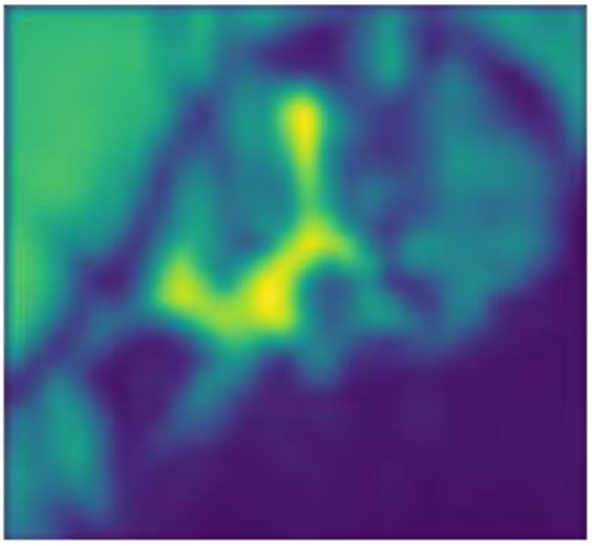


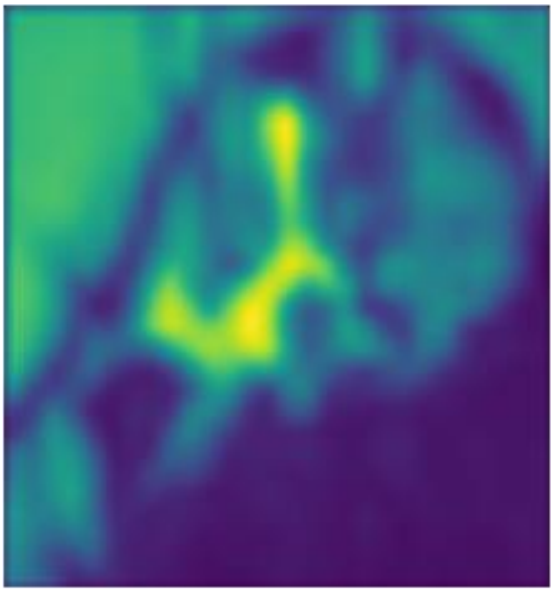
Figure 103: Original Image

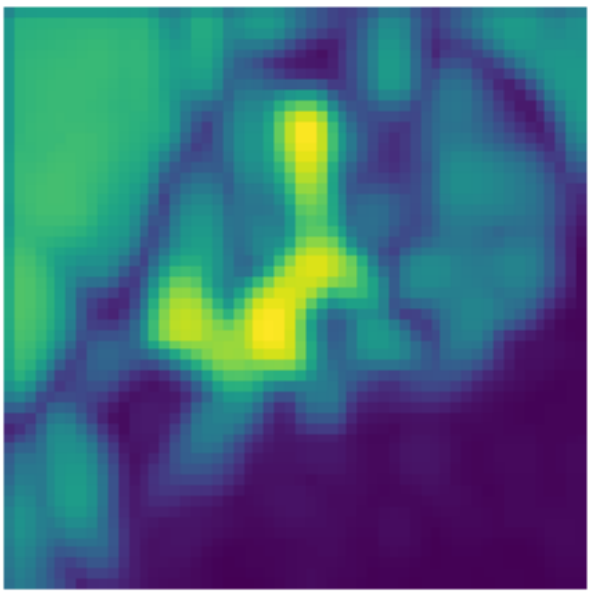


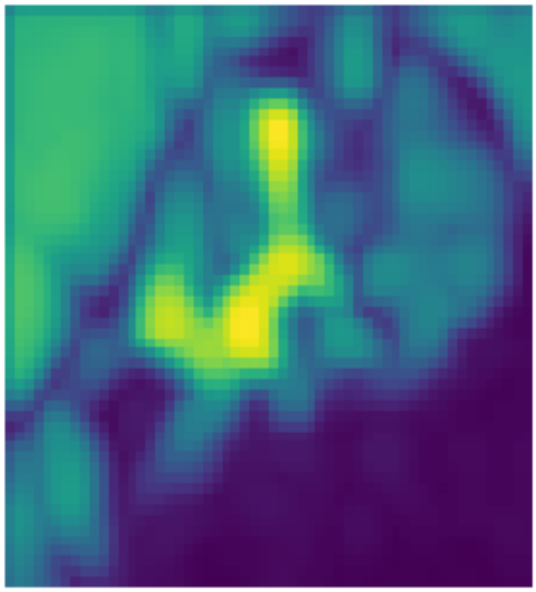
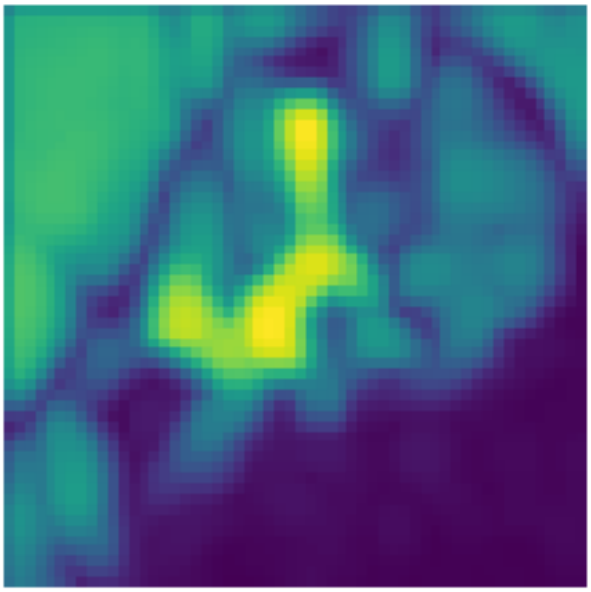
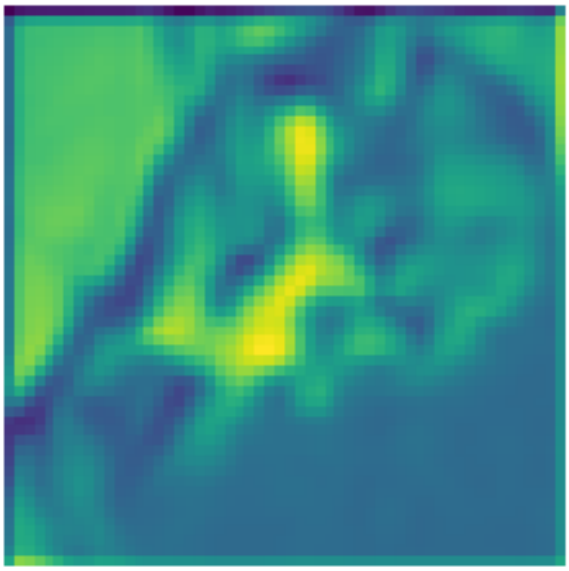
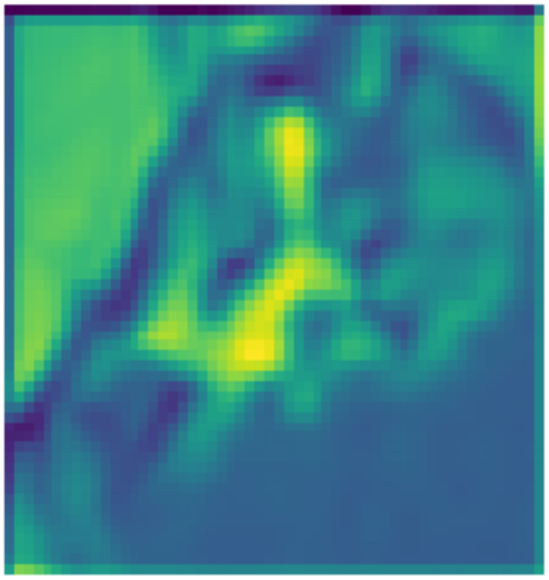
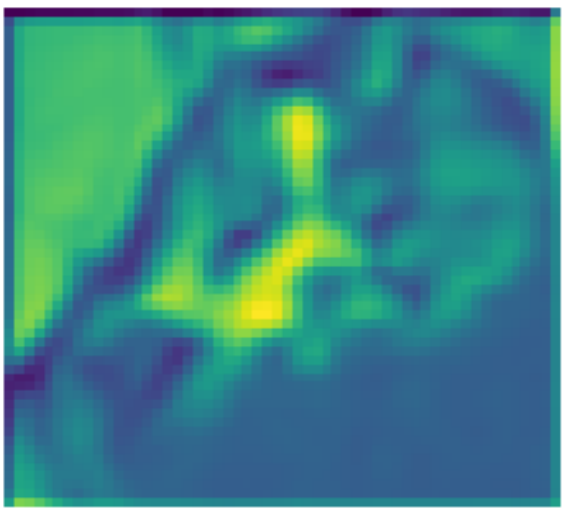
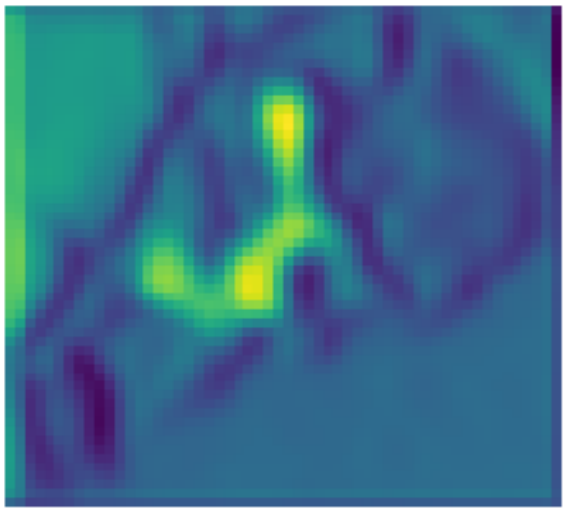


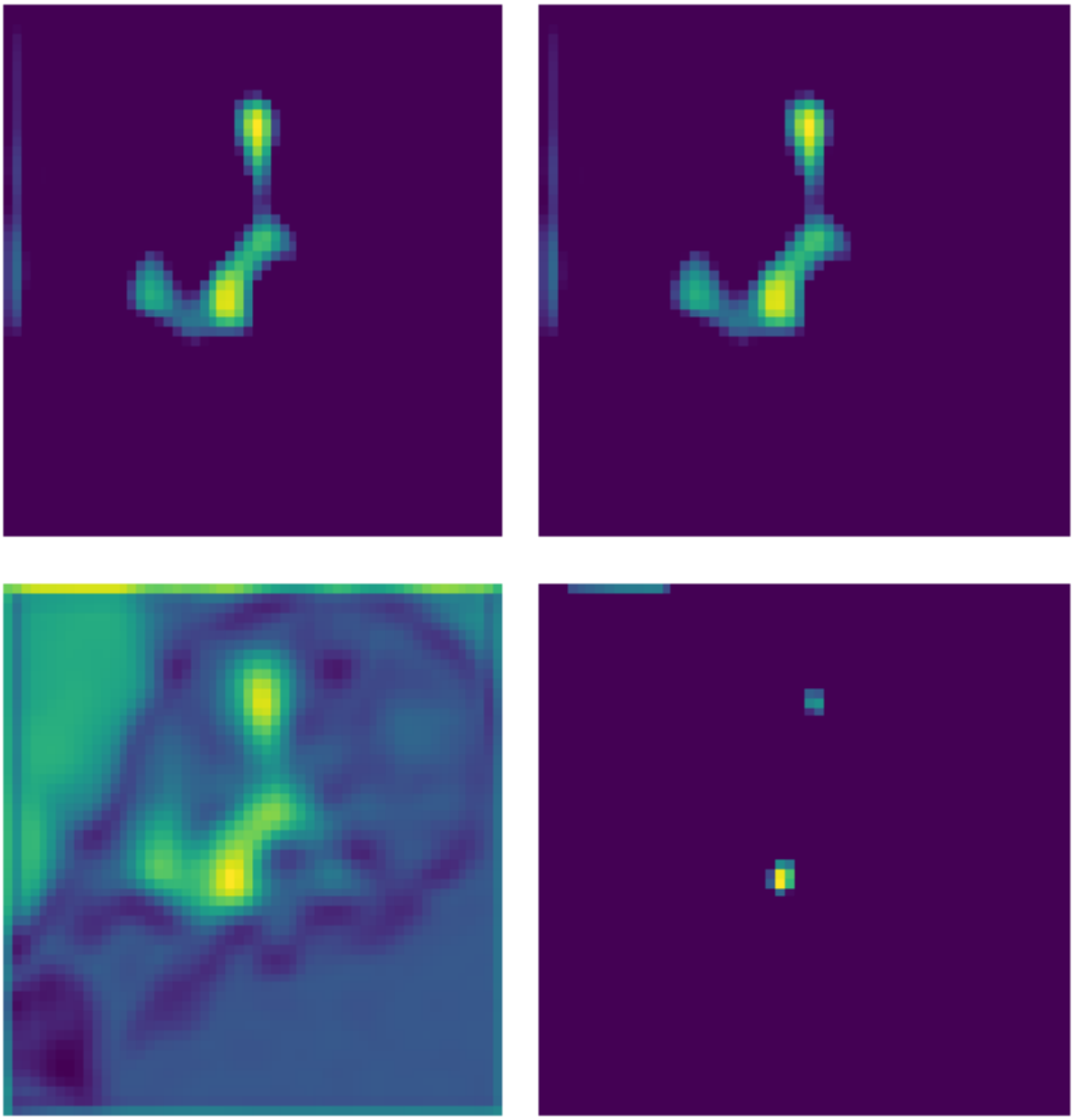










THANK YOU