Ronak Bhanushali and Rucha Pendharkar 1st April 2024

Project V - Recognition using Deep Networks

Introduction

The goal of this project is to build and train a deep network capable of completing a recognition task. These recognition tasks were mainly implemented using the MNIST Digit and MNIST Fashion Datasets. The network was implemented using pytorch. The following tasks were performed

Tasks

1. Build and train a network to recognize digits
2. Examine the Network
3. Transfer the learnings on Greek letters
4. Design your own Experiment
5. Extensions

Task 1: Build and train a network to recognize digits

A) Get the MNIST digit data set
The MNIST dataset used. To make the experiment repeatable, the random seed was set to 3.

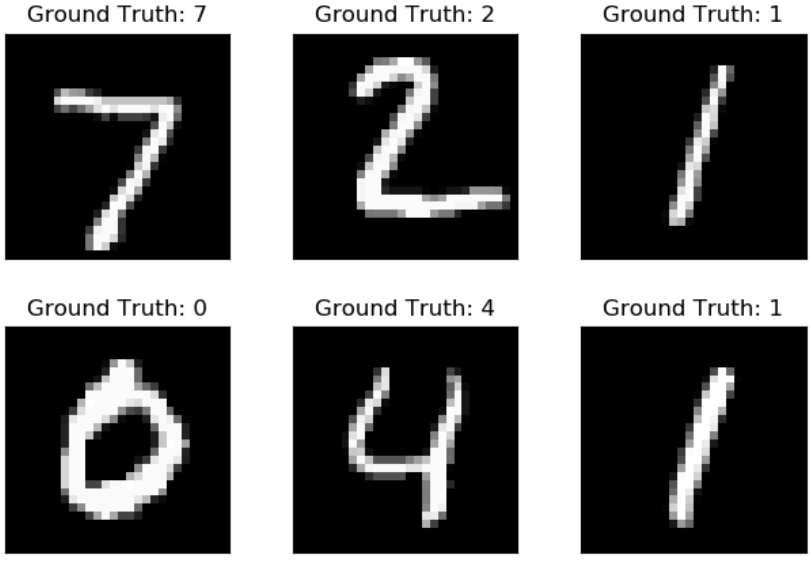


Fig 1: First six sample digits from the test set

B) Build a network model

```
mnistNet(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (fcnn1): Linear(in_features=980, out_features=50, bias=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fcnn2): Linear(in_features=50, out_features=10, bias=True)
)
```

Fig 3: Layer Definitions in pytorch

```
input torch.Size([2, 1, 28, 28])
convolution torch.Size([2, 10, 28, 28])
maxpool torch.Size([2, 10, 14, 14])
relu torch.Size([2, 10, 14, 14])
conv torch.Size([2, 20, 14, 14])
dropout torch.Size([2, 20, 14, 14])
maxpool torch.Size([2, 20, 7, 7])
relu torch.Size([2, 20, 7, 7])
FCNN torch.Size([2, 50])
Relu torch.Size([2, 50])
FCNN torch.Size([2, 10])
Softmax torch.Size([2, 10])
```

Fig 4: Output size of each layer

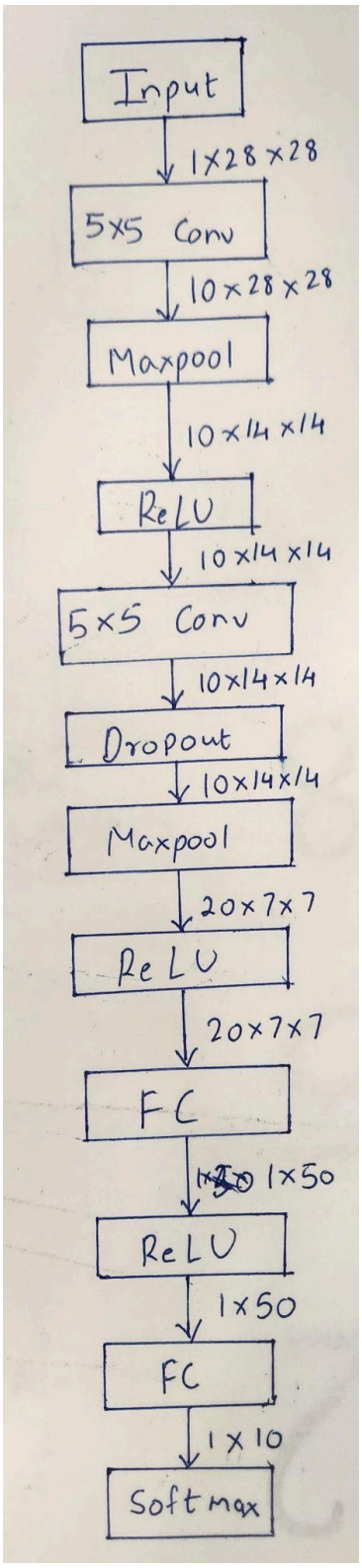


Fig 2: Network Architecture

C) Train the model

After around 100000 examples are seen, the training seems to converge and the loss stops decreasing

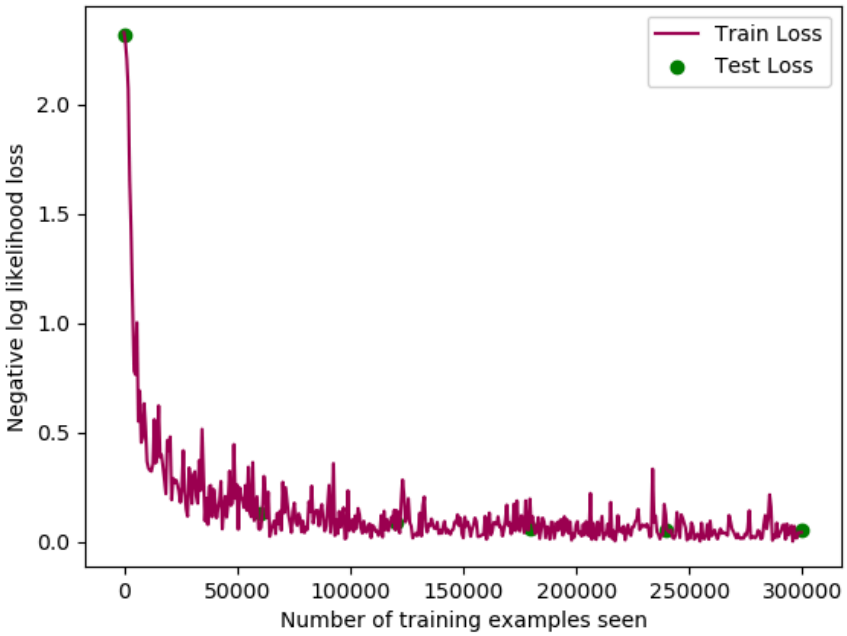


Fig 5: Training plot

D) Save the network

The network was saved as checkpoint.pth and checkpoint_best.pth where checkpoint_best.pth corresponds to the checkpoint with lowest training loss

E) Read the network and run it on the test set

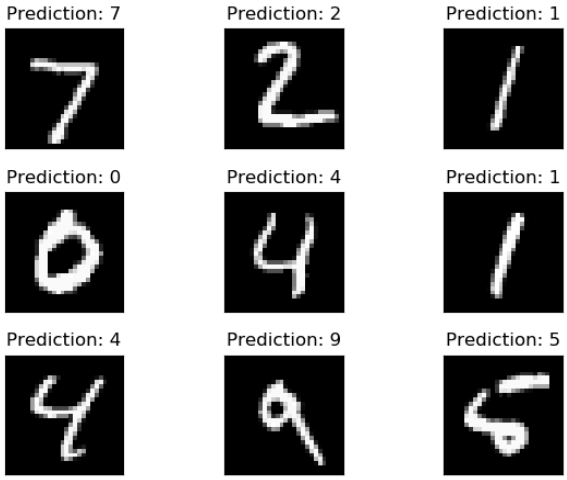


Fig 6: Predictions on the test set

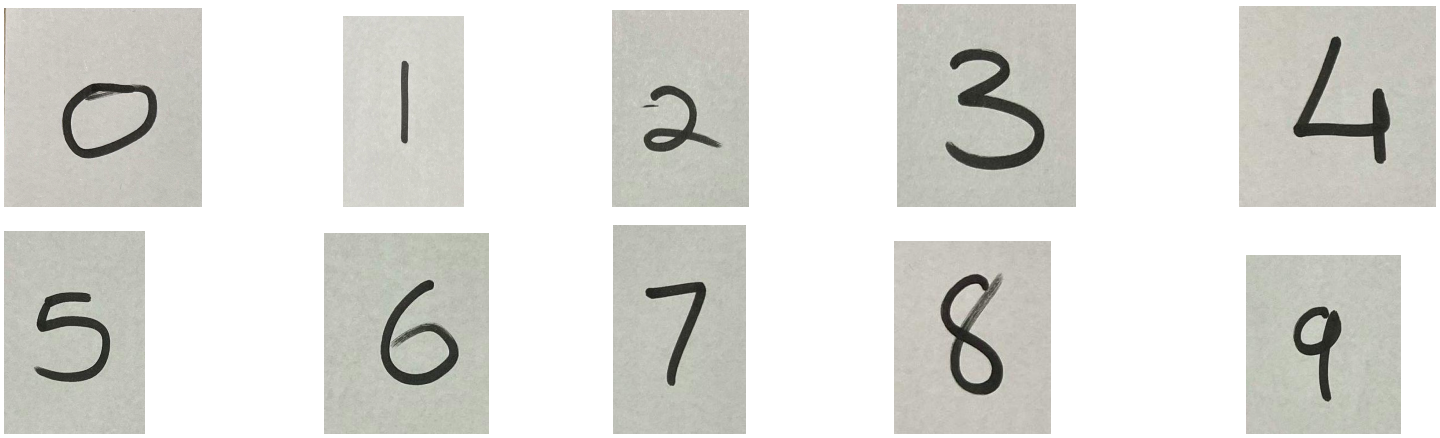
```
tensor([   -19.77,    -18.02,    -13.57,    -12.76,    -24.07,    -19.53,
           -36.40,     -0.00,    -18.99,    -17.31])
Best Match:  tensor(7)
Ground Truth:  tensor(7)
tensor([   -13.14,    -10.41,     -0.00,    -18.09,    -24.89,    -21.42,
           -14.29,    -21.02,    -12.32,    -23.51])
Best Match:  tensor(2)
Ground Truth:  tensor(2)
tensor([   -13.17,     -0.00,    -12.40,    -18.77,     -9.66,    -15.64,
           -11.71,    -10.10,    -11.18,    -14.38])
Best Match:  tensor(1)
Ground Truth:  tensor(1)
tensor([    -0.00,    -20.05,    -11.61,    -17.15,    -18.18,    -13.46,
           -11.13,    -15.07,    -13.34,    -13.94])
Best Match:  tensor(0)
Ground Truth:  tensor(0)
tensor([   -17.32,    -14.29,    -18.69,    -18.36,     -0.00,    -14.83,
           -15.89,    -14.35,    -16.11,     -5.87])
Best Match:  tensor(4)
Ground Truth:  tensor(4)
tensor([   -15.77,     -0.00,    -16.44,    -23.46,    -10.11,    -20.75,
           -16.99,    -12.07,    -13.18,    -16.88])
Best Match:  tensor(1)
Ground Truth:  tensor(1)
tensor([   -27.02,    -14.62,    -19.48,    -25.96,     -0.00,    -16.33,
           -20.13,    -11.12,     -7.55,     -9.69])
Best Match:  tensor(4)
Ground Truth:  tensor(4)
tensor([   -18.47,     -7.79,    -12.35,    -10.27,     -4.77,     -9.01,
           -16.84,    -12.60,     -9.45,     -0.01])
Best Match:  tensor(9)
Ground Truth:  tensor(9)
tensor([   -16.23,    -19.69,    -16.78,    -18.50,    -18.76,     -0.00,
            -7.10,    -20.48,     -9.73,     -9.76])
Best Match:  tensor(5)
Ground Truth:  tensor(5)
tensor([   -23.63,    -23.34,    -24.76,    -13.58,    -13.34,    -16.61,
           -30.30,     -9.14,    -12.01,     -0.00])
Best Match:  tensor(9)
Ground Truth:  tensor(9)
```

Fig 7: Output values for each photo and best match value

The accuracy of test set is seen to be 98%

F) Test the network on new inputs

The following digits were hand drawn:



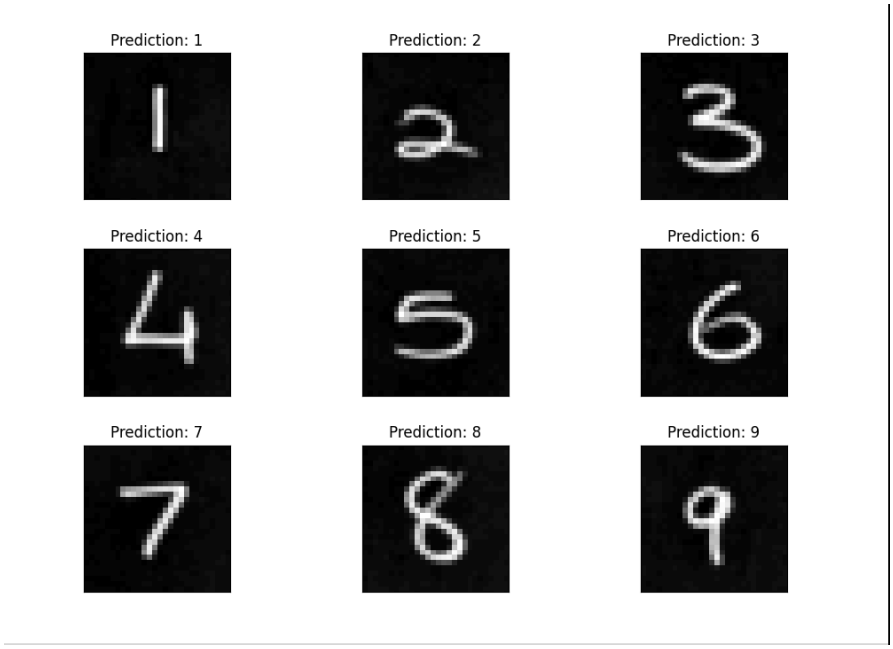


Fig 8: Handwritten output and model prediction

```
tensor([-1.79, -5.98, -4.30, -2.12, -5.41, -2.19, -0.66, -5.46, -3.08, -4.29])
Best Match:  tensor(6)
Ground Truth:  0
tensor([-4.36, -0.37, -4.01, -4.38, -2.96, -3.81, -3.93, -3.98, -1.98, -4.21])
Best Match:  tensor(1)
Ground Truth:  1
tensor([-5.46, -3.20, -0.84, -1.44, -5.58, -1.72, -4.01, -4.02, -2.91, -4.31])
Best Match:  tensor(2)
Ground Truth:  2
tensor([-12.52,  -8.53,  -7.33,  -0.03, -11.09,  -4.32, -10.16,  -8.60,  -4.57,
        -6.62])
Best Match:  tensor(3)
Ground Truth:  3
tensor([-7.50, -5.86, -4.56, -7.56, -0.06, -4.88, -4.48, -7.65, -8.23, -3.79])
Best Match:  tensor(4)
Ground Truth:  4
tensor([-7.95, -8.37, -7.50, -2.19, -8.05, -0.17, -6.64, -6.38, -5.15, -3.46])
Best Match:  tensor(5)
Ground Truth:  5
tensor([-3.08, -6.19, -4.34, -3.41, -5.41, -2.91, -0.25, -6.71, -3.11, -3.98])
Best Match:  tensor(6)
Ground Truth:  6
tensor([-5.09, -4.61, -2.88, -1.20, -6.97, -5.64, -7.59, -1.16, -1.25, -3.89])
Best Match:  tensor(7)
Ground Truth:  7
tensor([-8.00, -7.08, -5.75, -3.49, -9.21, -2.18, -3.72, -7.63, -0.22, -3.75])
Best Match:  tensor(8)
Ground Truth:  8
tensor([-8.24, -4.63, -8.18, -3.67, -3.64, -4.76, -7.84, -4.45, -1.63, -0.33])
Best Match:  tensor(9)
Ground Truth:  9
```

Fig 9 Output values for each photo and best match value

Task 2: Examine your network

A) Analyze the first layer

The weights were extracted and visualized as follows

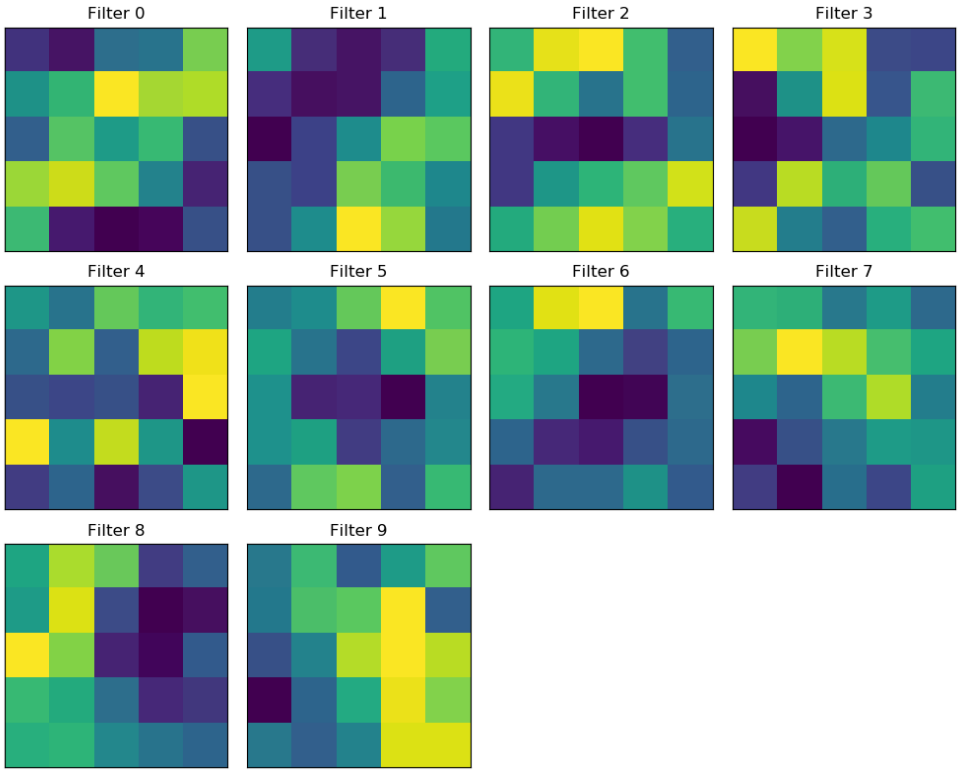


Fig10 : Filter weights visualized.

B) Show the effect of filters

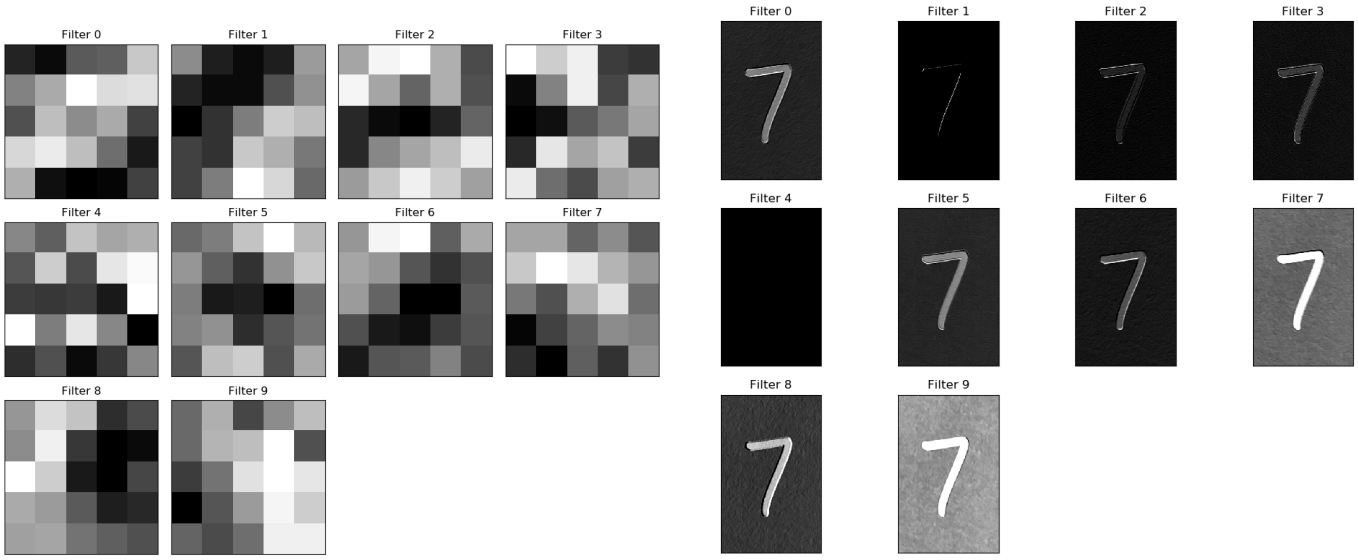


Fig11: Effect of filters

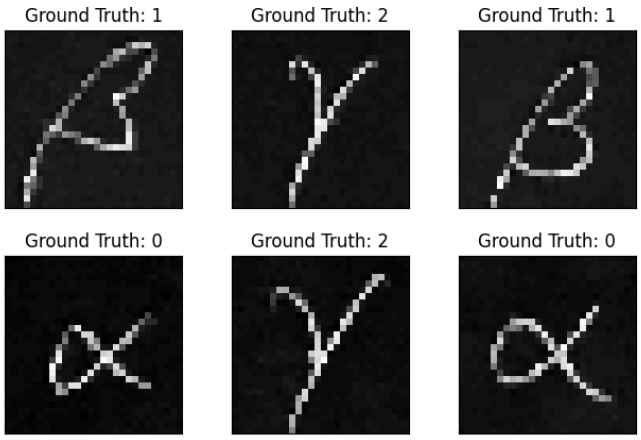
It looks like the model creates filters that try to detect the edge and texture in the image. For example, filter 1 looks like it is trying to detect lines angled at 45 degrees whereas filter 2 tries to detect horizontal lines. Filter 5 somewhat resembles the kernel of a Gabor filter when visualized and the output seems to highlight the texture of the image. Looking at the output, it is clear that some filters focus on edges, some focus on texture and some highlight different things. Since the process of training is stochastic and iterative, the filters are not perfect and might even be a mix of multiple filters.

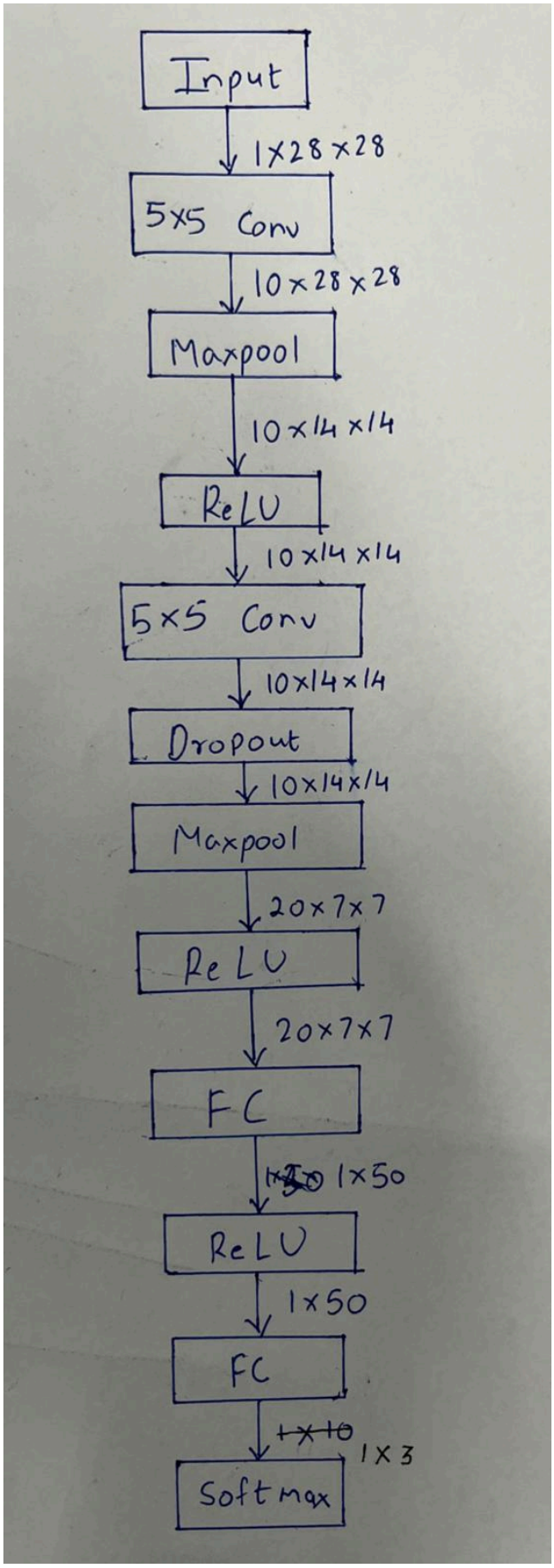
<u>Task 3: Transfer learning on Greek Letters</u>

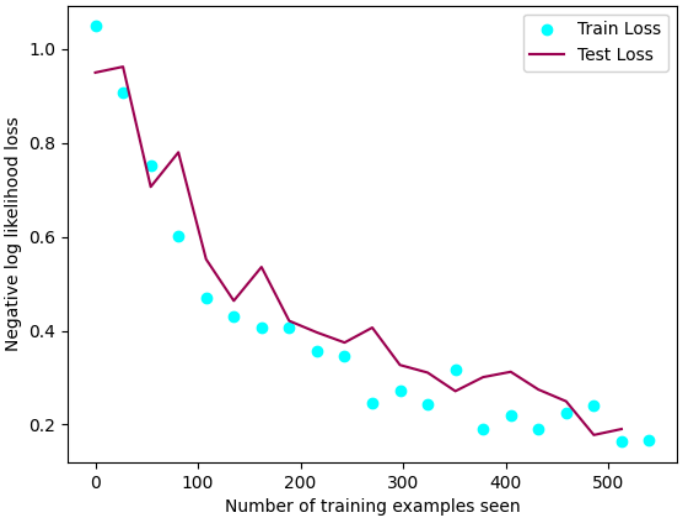
For this task, we changed the last layer's number of output to 3 since we only have 3 classes.

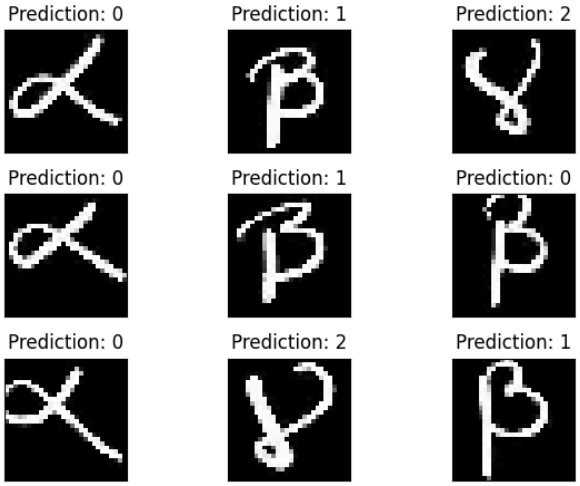
```
mnistNet(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (fcnn1): Linear(in_features=980, out_features=50, bias=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fcnn2): Linear(in_features=50, out_features=3, bias=True)
)
```

Fig12: Model Printout for new model










```
tensor([-2.25, -0.78, -0.83])
Best Match:  tensor(1)
Ground Truth:  tensor(1)
tensor([    -0.00,    -15.63,    -18.26])
Best Match:  tensor(0)
Ground Truth:  tensor(0)
tensor([-5.65, -0.40, -1.12])
Best Match:  tensor(1)
Ground Truth:  tensor(1)
tensor([   -11.45,    -10.35,     -0.00])
Best Match:  tensor(2)
Ground Truth:  tensor(2)
tensor([-0.10, -2.39, -6.79])
Best Match:  tensor(0)
Ground Truth:  tensor(0)
tensor([-4.89, -0.01, -6.16])
Best Match:  tensor(1)
Ground Truth:  tensor(1)
tensor([-0.59, -0.84, -4.12])
Best Match:  tensor(0)
Ground Truth:  tensor(1)
tensor([-0.03, -5.44, -3.88])
Best Match:  tensor(0)
Ground Truth:  tensor(0)
tensor([-9.57, -4.20, -0.02])
Best Match:  tensor(2)
Ground Truth:  tensor(2)
tensor([-2.18, -0.17, -3.08])
Best Match:  tensor(1)
Ground Truth:  tensor(1)

Test set: Avg. loss: 0.6585, Accuracy: 12/15 (80%)
```

As indicated by the picture, the difference between the previous and new network was setting the output of the fully connected layer to 3 instead of 10. The accuracy on custom hand written letters was 80%. It takes only about 3 epochs to reach 100% accuracy. This makes sense as we have a pre-trained backbone and we only fine tune on this new dataset.

Task 4: Design your own experiment

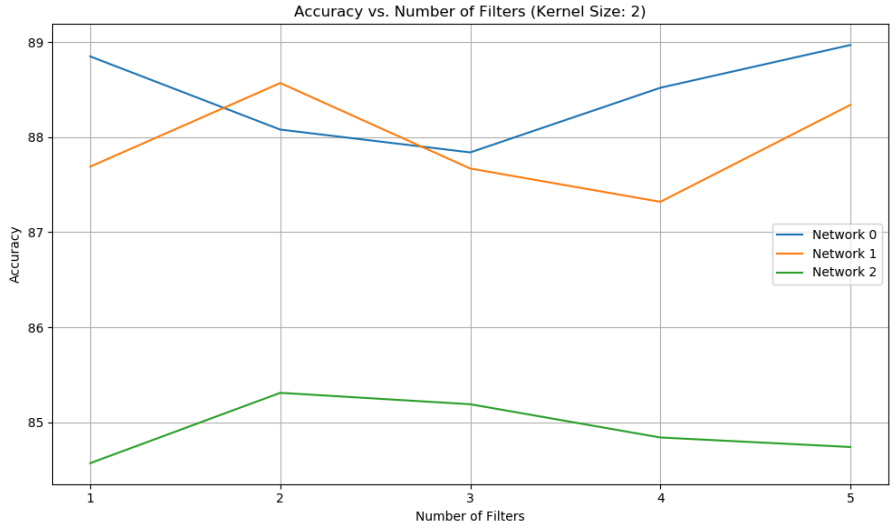
A) Develop a plan - For our experiment it was decided to vary three dimensions. The model architecture, the kernel size and the number of filters were varied. The MNIST Fashion Dataset was used for this task.

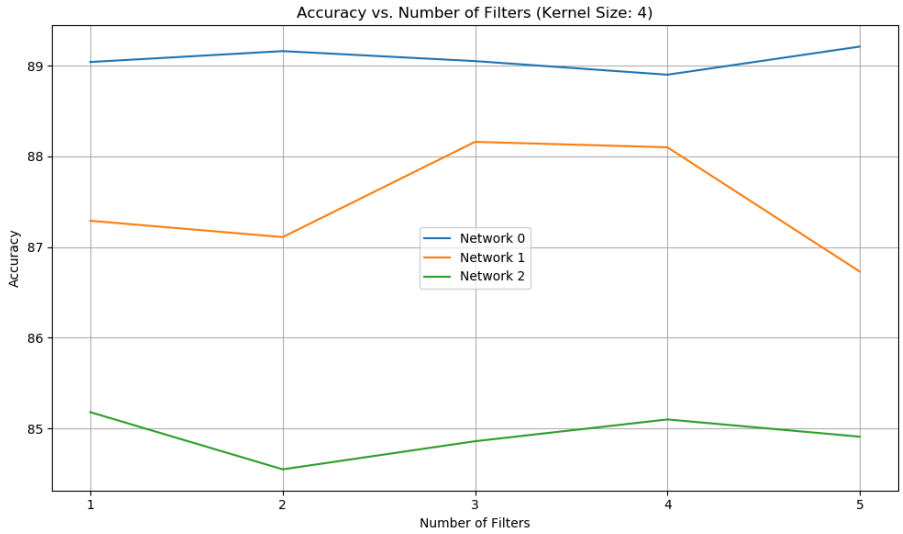
   i) Model Architecture: mnistNet, mnistNet3 consists of two convolutional layers, whereas mnistNet2 contains three convolutional layers. mnistNet2 and mnistNet3 incorporate a dropout layer with a dropout probability of 0.5 after the second convolutional layer.
   ii) Kernel Size: The size of the kernel was varied between 2,3,4,5,6.
   iii) Number of filters: The number of filters were varied between 5,10,20,50,90

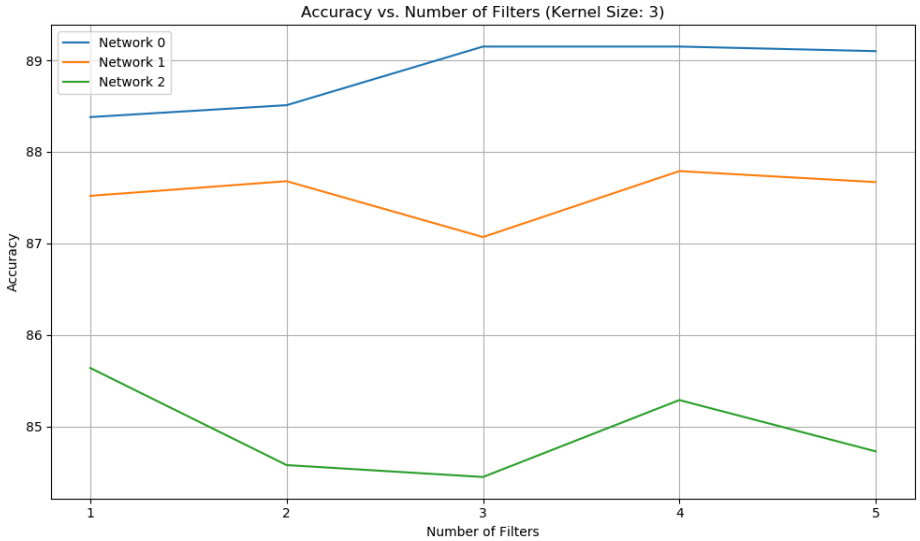
B) Predict the results - Accuracy of classification was used as the metric for evaluation and comparison of the effects of varying these parameters.

   Hypothesis:
   i) Model Architecture : We thought that adding more convolutional layers and a dropout layer would increase the flexibility of the model and would increase the overall accuracy. We also changed the architecture by replacing max pool with avg pool to preserve
   ii) Kernel Size: We thought that increasing the kernel size would affect the accuracy
   iii) Number of filters: We thought that increasing the filters will yield better accuracy by making the model more flexible.

C) Execute your plan - The following were the results obtained for the above experiment.







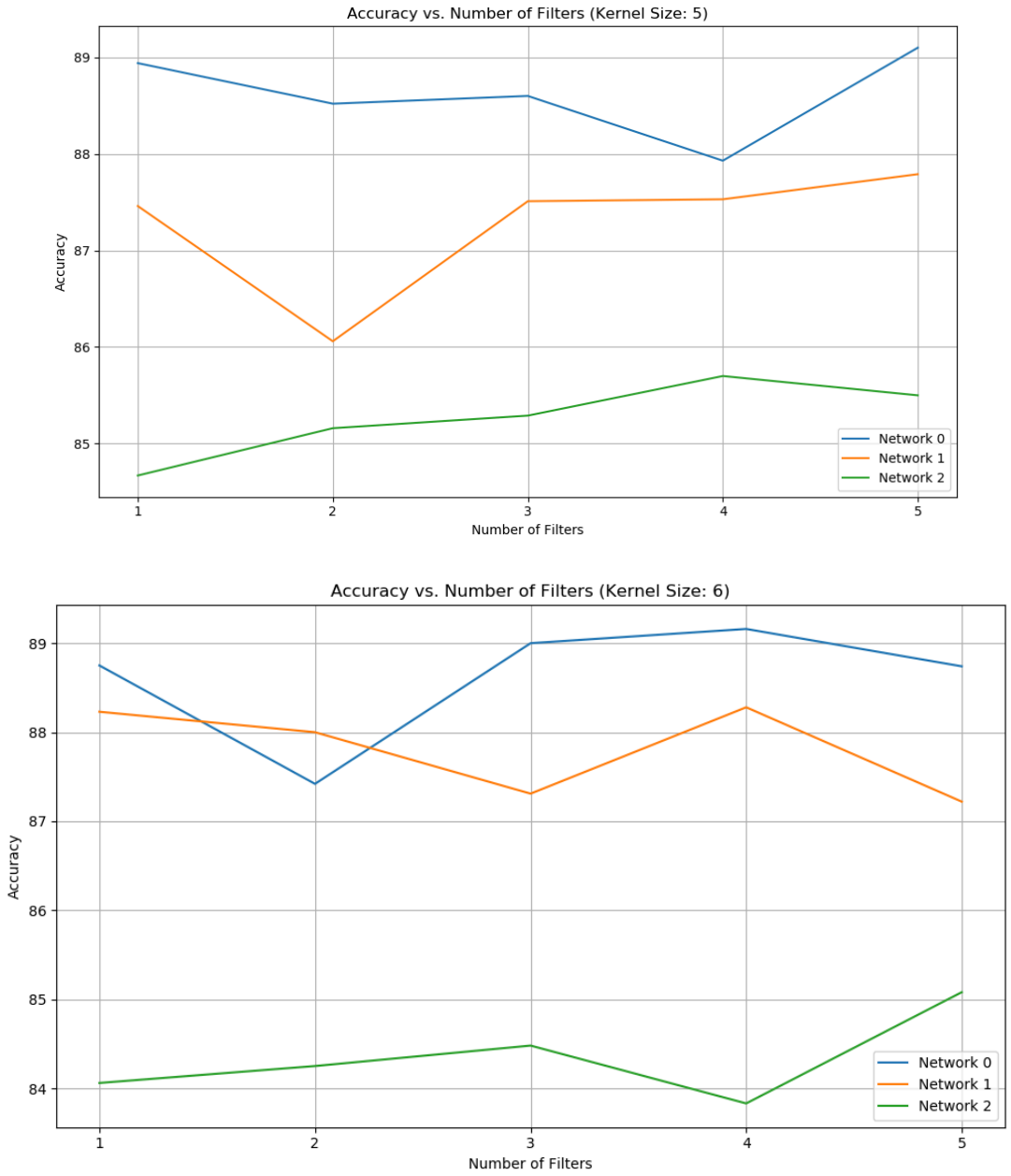


Fig 14: Results of task 4 experimentation

From the graphs, it can be concluded that the maximum accuracy is obtained when Network 0 (i.e the network defined in task 1) is trained with a kernel size of 5 and Number of filters are 90. The maximum accuracy is **89.21%**

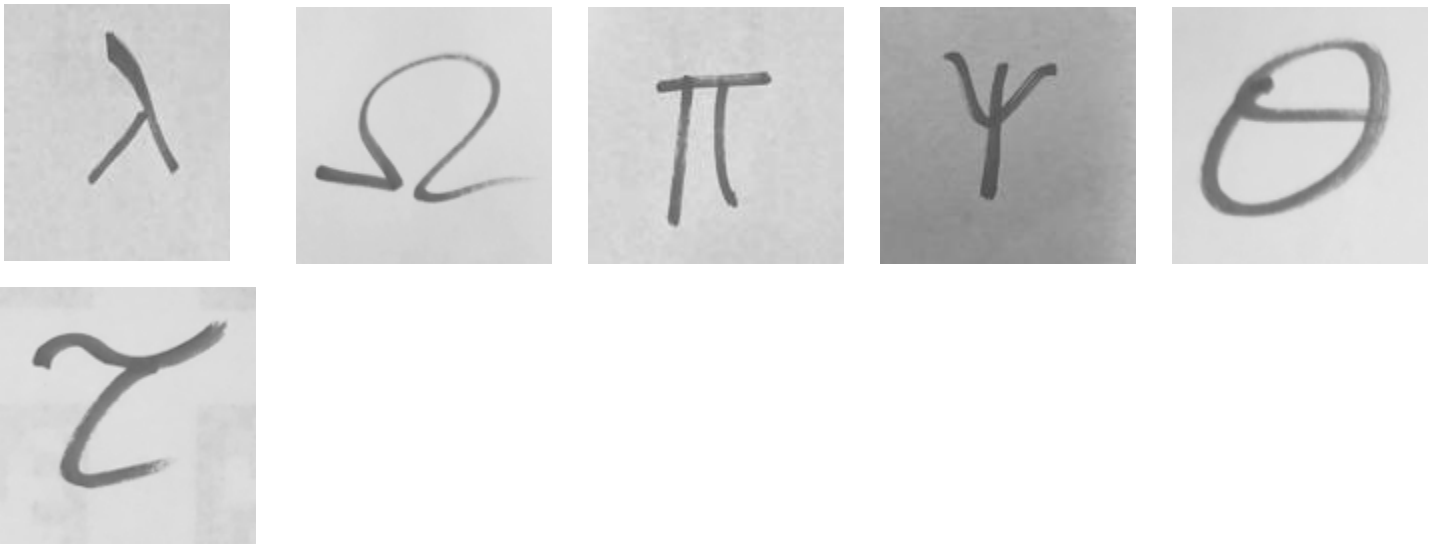
The only visible trend is with the model architecture. Contrary to what we believed, having more filters did not result in higher accuracy.

The trends in number of filters did not quite affect the accuracy too much. The trends look more or less due to stochasticity. The reason might be the that the extra filters are redundant and dont affect the performance. The kernel size seems to increase accuracy up to size 5 after which it starts to decrease.

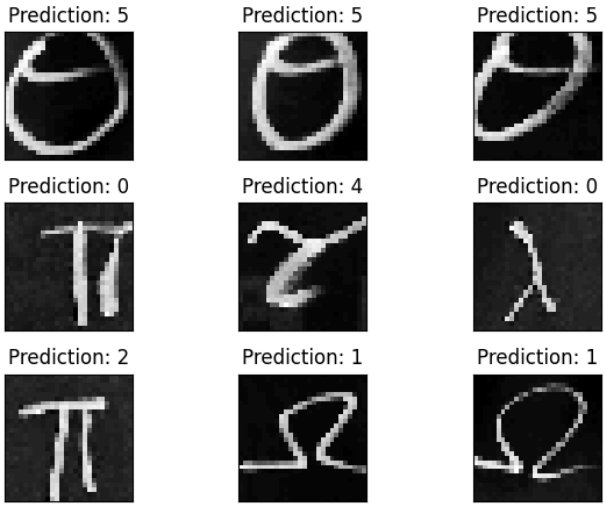
Extensions

Extension 1: Trying more greek letters

For this extension we decided to test 6 more greek letters. We hand-drew lambda , pi, psi, omega, theta and tau. The link to dataset - https://drive.google.com/drive/folders/1c4XRiisC3ZAQLG-Pc7yR7W6doxjPS5AN?usp=drive_link



The model was changed to have 6 output values as we have 6 classes. We trained for 20 epochs and like previously, it only takes two to three epochs to get 100% accuracy. The output is as follows-




```
tensor([-0.24, -2.28, -3.60, -4.07, -3.07, -3.98])
Best Match:  tensor(0)
Ground Truth:  tensor(0)
tensor([-3.51, -4.83, -6.20, -5.28, -3.98, -0.07])
Best Match:  tensor(5)
Ground Truth:  tensor(5)
tensor([-4.13, -3.77, -3.28, -3.12, -1.64, -0.38])
Best Match:  tensor(5)
Ground Truth:  tensor(5)
tensor([-1.78, -2.01, -3.30, -4.79, -1.28, -0.99])
Best Match:  tensor(5)
Ground Truth:  tensor(5)
tensor([-1.02, -4.08, -1.10, -1.79, -3.74, -2.33])
Best Match:  tensor(0)
Ground Truth:  tensor(2)
tensor([-2.62, -3.10, -2.99, -3.19, -0.48, -1.75])
Best Match:  tensor(4)
Ground Truth:  tensor(4)
tensor([-0.35, -3.57, -2.93, -3.09, -2.82, -2.22])
Best Match:  tensor(0)
Ground Truth:  tensor(0)
tensor([-5.68, -8.42, -0.09, -4.19, -2.96, -4.23])
Best Match:  tensor(2)
Ground Truth:  tensor(2)
tensor([-2.68, -0.11, -5.29, -5.84, -3.83, -4.66])
Best Match:  tensor(1)
Ground Truth:  tensor(1)
tensor([-3.30, -0.24, -4.22, -4.71, -2.31, -2.92])
Best Match:  tensor(1)
Ground Truth:  tensor(1)

Test set: Avg. loss: 0.4231, Accuracy: 13/13 (100%)
```

Fig13: Prediction on custom greek letters dataset

Extension 2: Varying the number of epochs of training

From task 4, the best accuracy was obtained when the number of filters = 90 and kernel size was set to index 3 i.e. 5, with the original network (task1). Using this data, the number of epochs were varied to observe the behavior of accuracy with different epochs. The results were plotted. It can be observed that the accuracy kept on increasing till the number of epochs were 40, and then reduced after 50. This is likely due to overfitting.
The maximum accuracy is **90.7%**

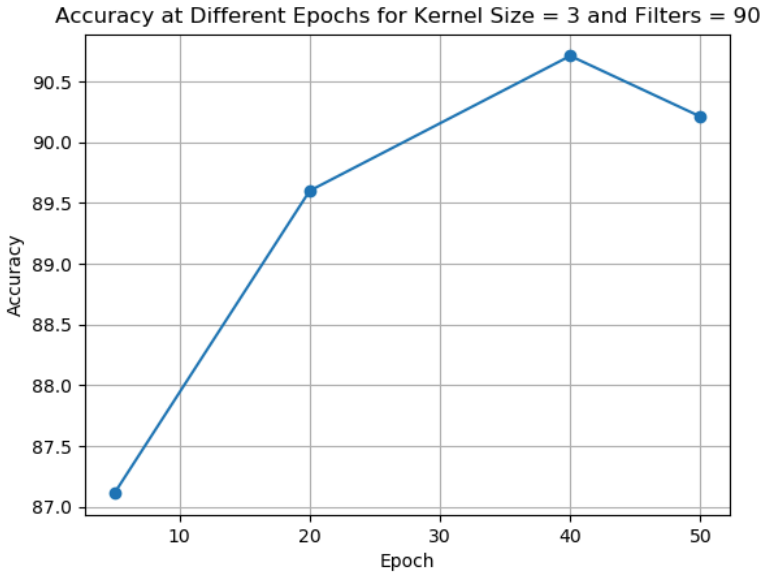


Fig15: Accuracy vs Epochs

Extension 3

For this extension, we used mobilenetv3-large as the backbone for our network. We modified the last layer such that we replace the classification layer. We got **91%** accuracy with this model with 20 epochs. New architecture-

```
class MobileNetMNIST(nn.Module):
    def __init__(self):
        super(MobileNetMNIST, self).__init__()

        self.backbone = models.mobilenet_v3_large(weights='MobileNet_V3_Large_Weights.DEFAULT')
        print('Loading pretrained MobileNet model.....')
        in_features = self.backbone.classifier[0].in_features
        self.linear_layer = nn.Sequential(nn.Dropout(p=0.5),nn.Linear(in_features, 10))
        self.batch_norm = nn.BatchNorm1d(in_features,track_running_stats=True)

    def forward(self, x):
        x = self.backbone.features(x)
        x = self.backbone.avgpool(x)
        x = torch.squeeze(x,dim=2)
        x = torch.squeeze(x,dim=2)
        x = self.batch_norm(x)
        x = self.linear_layer(x)
        x = F.log_softmax(x,dim=1)
        return x
```

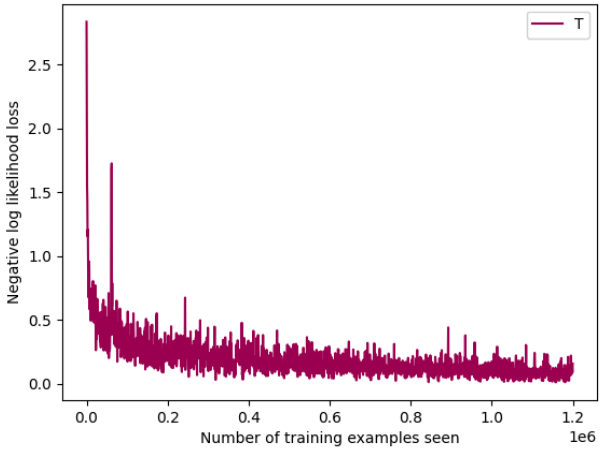


Fig 16: Mobilenetv3-large

Reflection

This project gave us a better understanding of how deep networks function. We also learnt about the Pytorch library, and experimented with different hyperparameters and their effect. We learnt the following-

1. Designing Deep Neural Networks in pytorch
2. Loading datasets from torchvision and custom datasets
3. Training models - choosing hyperparameters, saving model checkpoints, loading checkpoints
4. Transfer learning for a different dataset - freezing layers and training specific layers
5. Loading pretrained model and using it as backbone network

Acknowledgment

We would like to thank Prof Bruce Maxwell for helping us grasp these concepts quickly. His in-class notes and overall outline of the project was very useful in learning basics of deep networks effectively

References

1. CS5330 Class Recordings and Lecture Notes
2. Pytorch Documentation
3. StackOverflow for debugging
4. https://nextjournal.com/gkoehler/pytorch-mnist