Project 5: Recognition using Deep Networks

Team Mates: Yalala Mohit, Dhruv Kamalesh Kumar

Project Description:

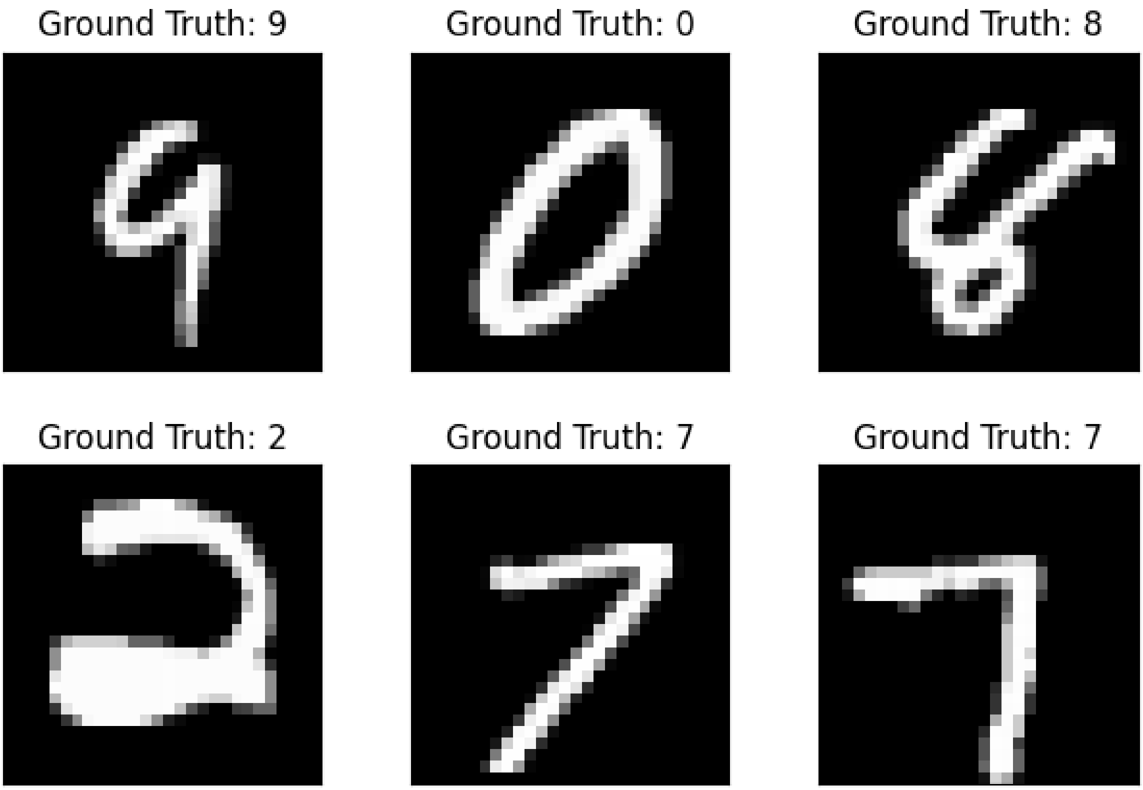
The project focuses on building, training, analyzing, and modifying a deep network for a recognition task using the MNIST digit recognition dataset. The MNIST digit recognition data set will be used to train the network, which is both simple enough to be trained without a GPU and challenging enough to provide a good example of what deep networks can do.

Tasks

1. **Build and train a network to recognize digits**

   A. **Get the MNIST digit data set**

The following are the first 6 mages in the MNIST digit dataset.



   B. **Make your network code repeatable**

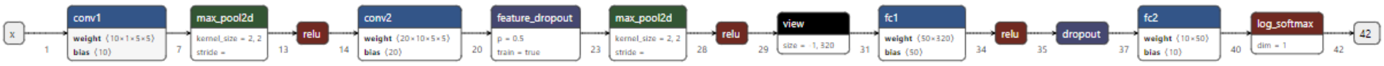
The code has been made repeatable by using,

```
torch.manual_seed(2502)

torch.backends.cudnn.enabled = False
```

C. Build a network model

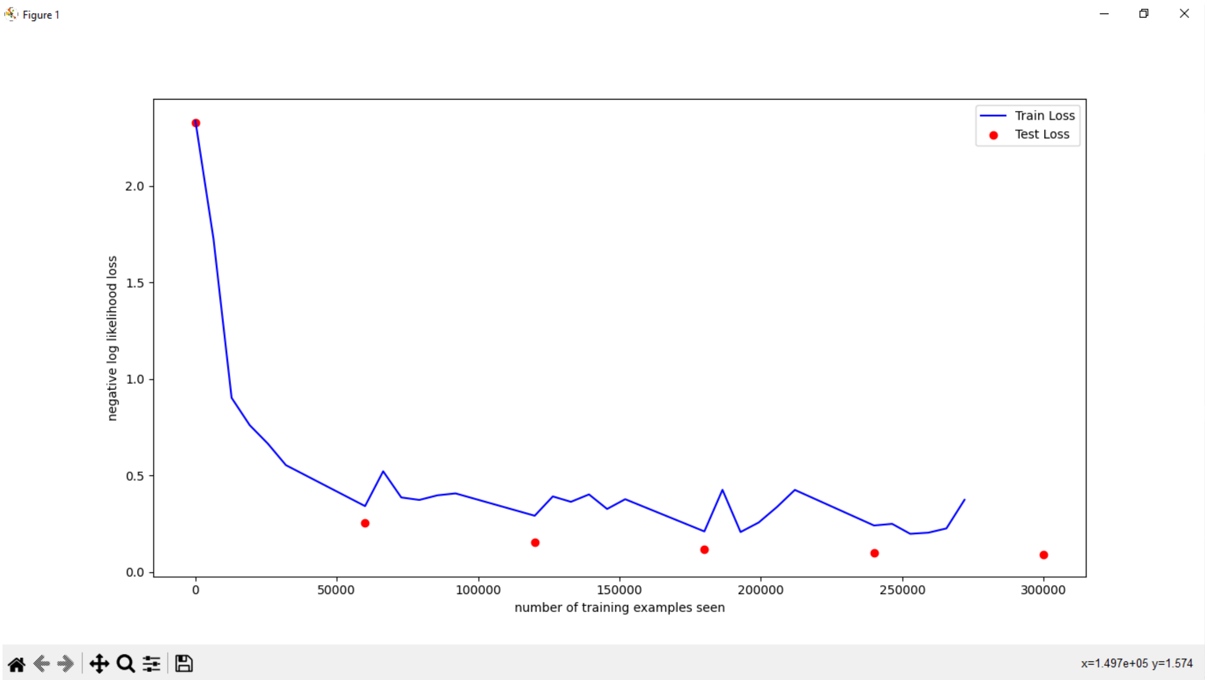
The following is the model summary, and a visualization of the model through Netron.app

```
NeuralNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
)
```



D. Train the model

The negative log likelihood loss, is attached below. Both the train loss and test loss gradually decrease over epochs, and donot diverge. Hence, we can suggest that there is no overfitting.

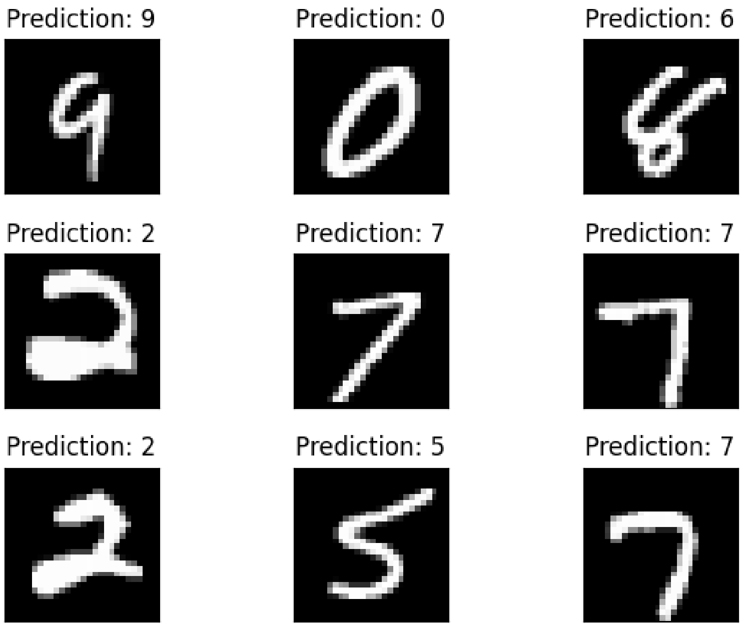


E. Save the network to a file

The network is saved in various formats. Even torch script .pt format was explored and visualized.

F. **Read the network and run it on the test set**

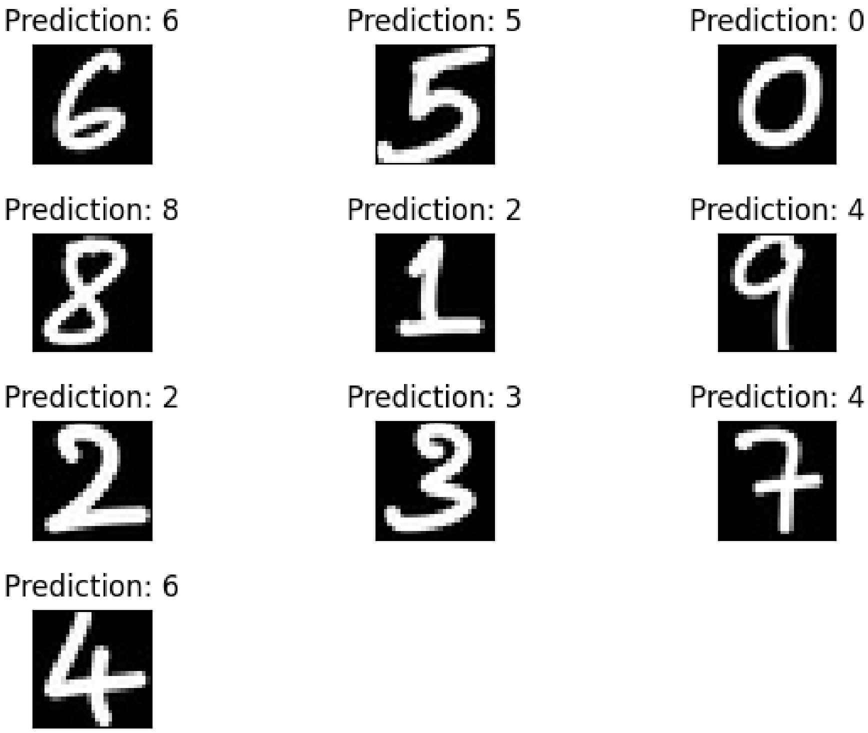
The following predictions were made, when run on the test set.



G. **Test the network on new inputs**

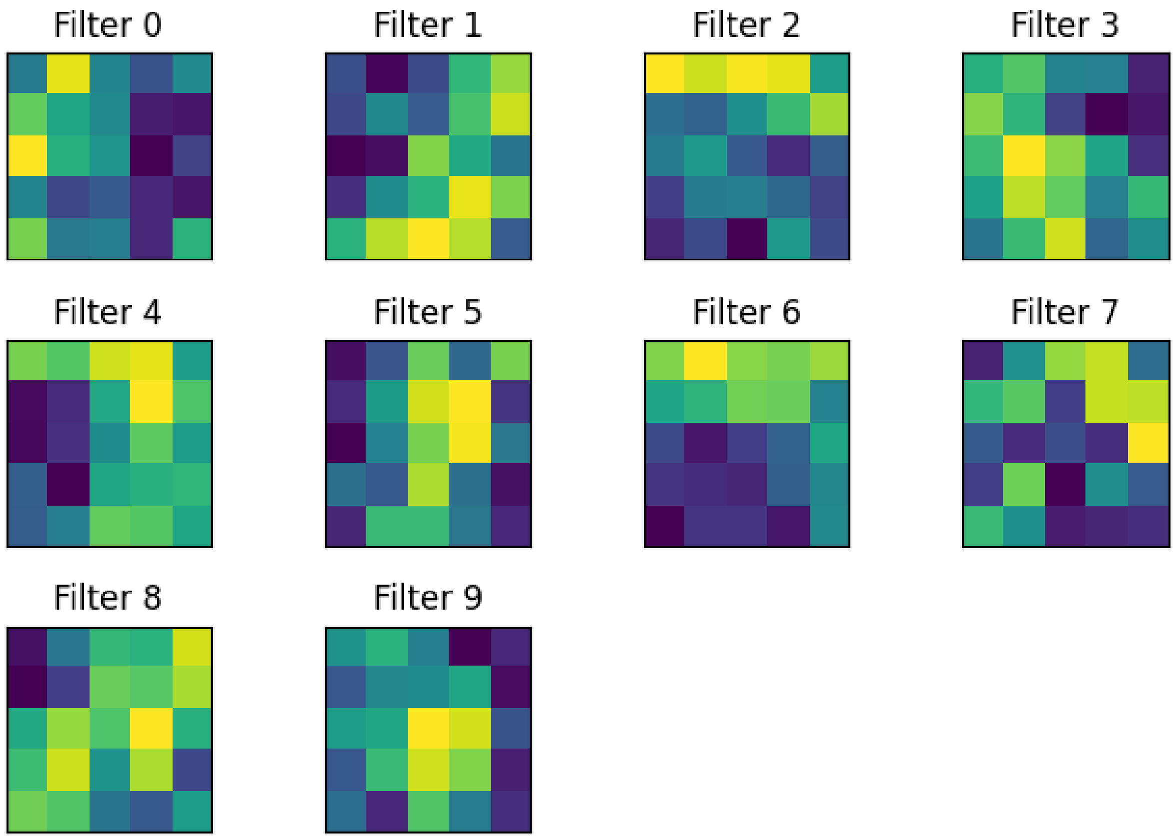
The following predictions were made on our hand written digits.

Test Accuracy = 60%

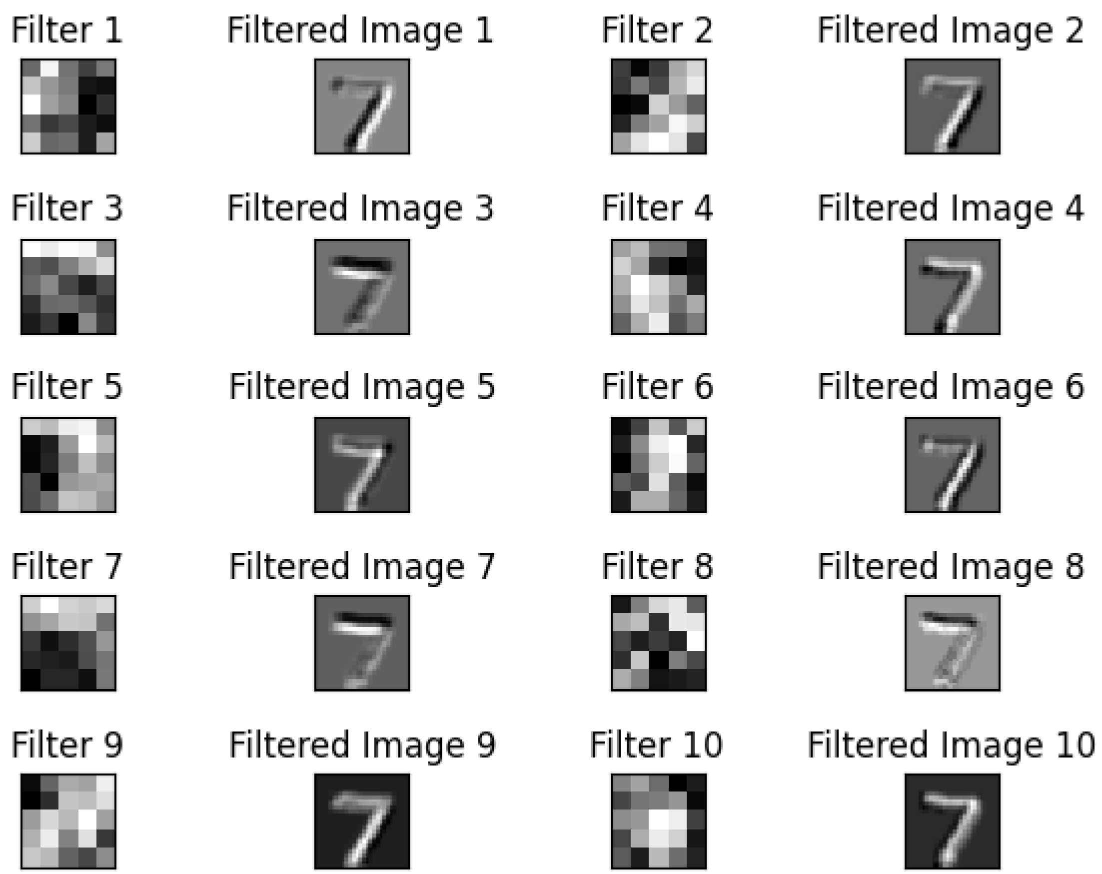


2. **Examine your network**

A. **Analyze the first layer**



B. **Show the effect of the filters**



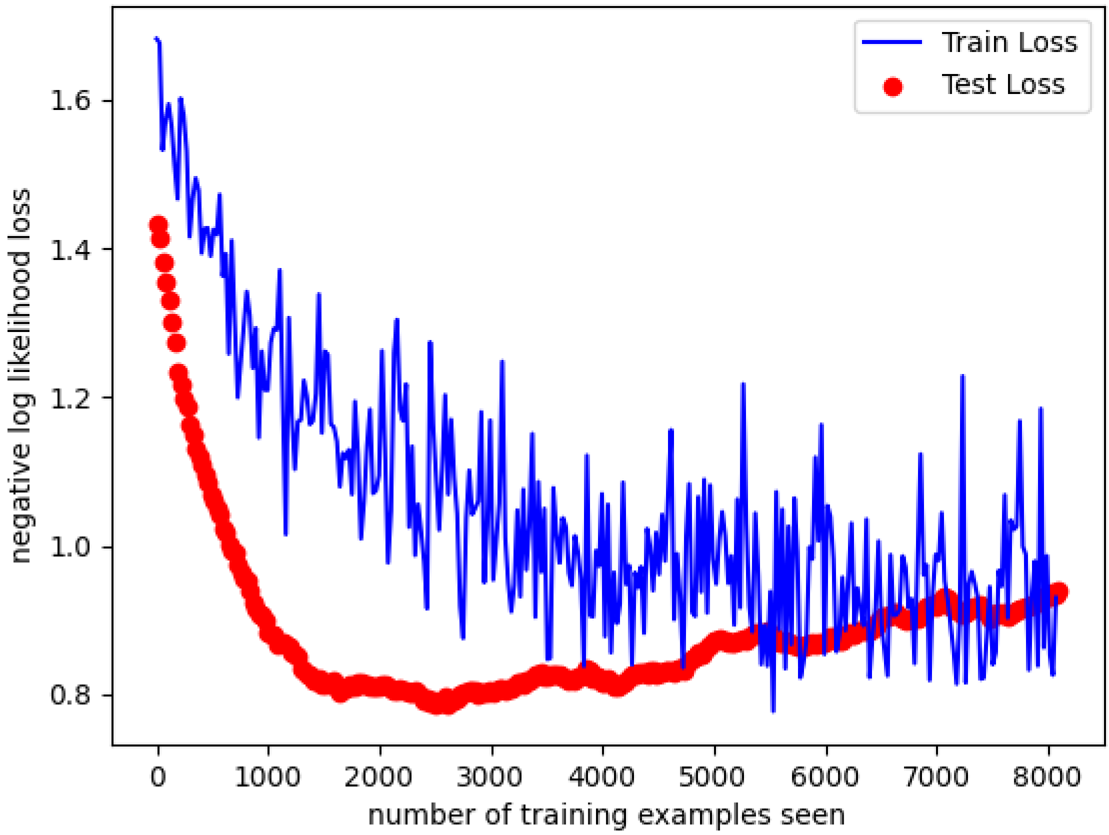
The filters after being applied to the image, highlights the number in various backgrounds, extracting the features of the number. It particularly makes sense, because they're the most important features in this aspect.

3. **Transfer Learning on Greek Letters**

How many epochs does it take using the 27 examples in order to perfectly identify them?

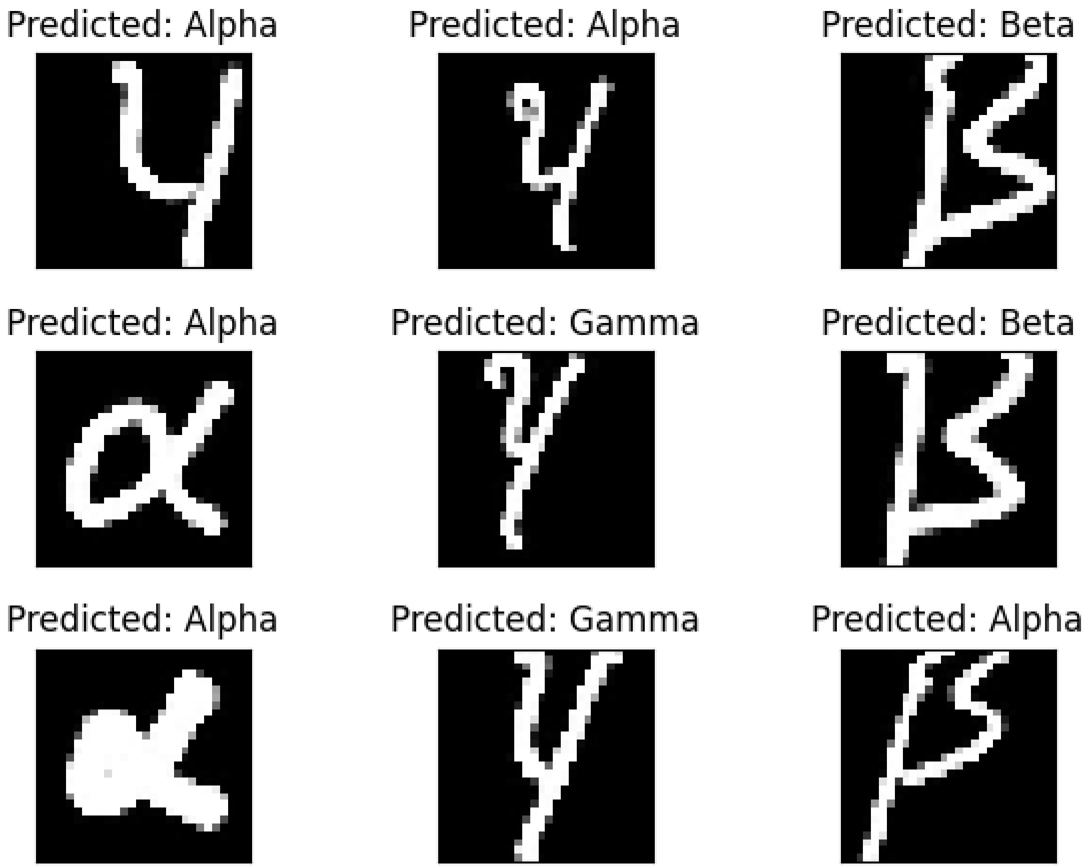
The model trains and improves accuracy until 85 epochs, but is stagnated after that , even when run until 300 epochs.

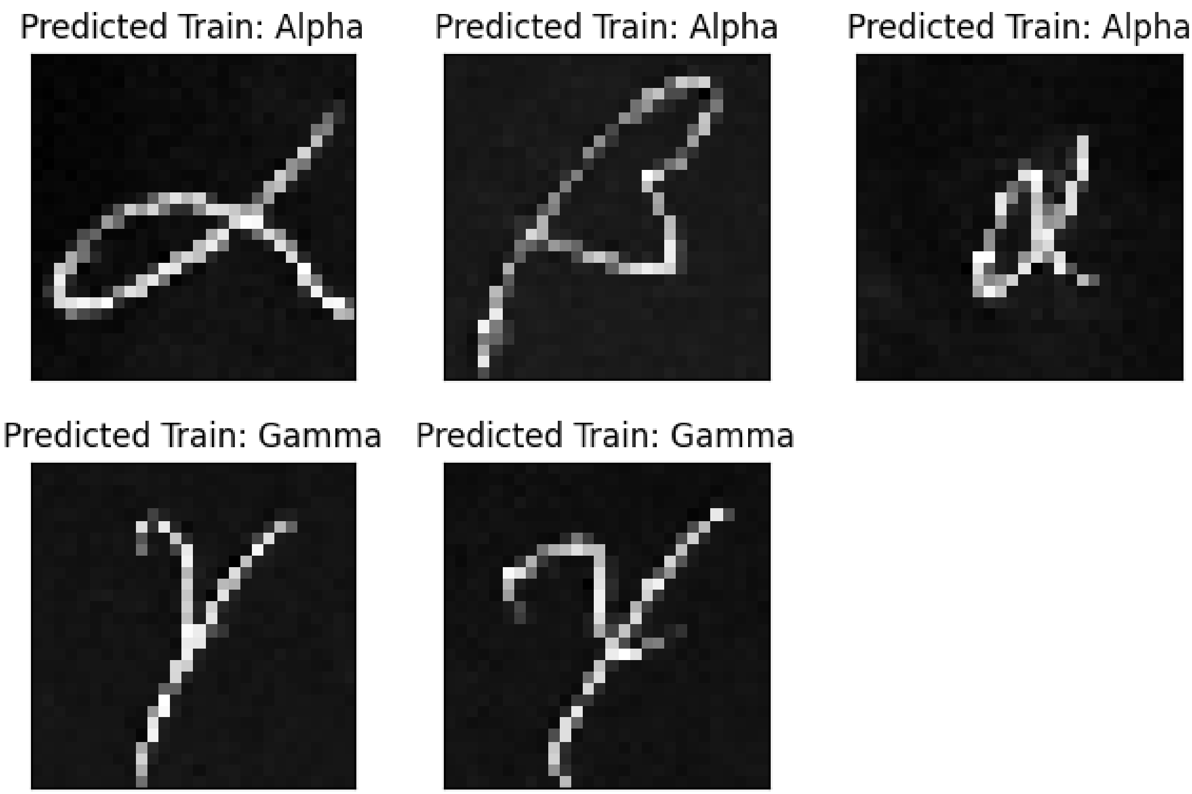
The loss plot below states that overfitting has been handled. **This can be attributed to the implementation of L2 regularization during training.**



The following are the predictions made on the test set, and hand written test set.

Test set accuracy is 79% and hand written test set accuracy is 75%


Predicted: Alpha
Predicted: Alpha
Predicted: Beta
Predicted: Alpha
Predicted: Gamma
Predicted: Beta
Predicted: Alpha
Predicted: Gamma
Predicted: Alpha


Predicted Train: Alpha
Predicted Train: Alpha
Predicted Train: Alpha
Predicted Train: Gamma
Predicted Train: Gamma

4. **Design your own experiment**

A. Develop a plan

The plan is to evaluate the following Hyperparameters. The model will evaluate all L*M*N combinations, which will correspond to 1024 models eventualy.

```python
# put all hyper params into a OrderedDict, easily expandable
params = OrderedDict(
    lr = [.01, .001],
    batch_size = [100, 1000],
    shuffle = [True],
    epochs = [5,10],
    conv_channels = [[6,12], [8,16]],
    conv_kernel_size = [3,5],
    pool_kernel_size = [2,3],
    pool_stride = [1, 2],
    dropout_rate = [0.1, 0.5],
    hidden_layers = [[120, 60], [512,256]],
    activation = [nn.ReLU(), nn.Tanh()],
)
```

B. Predict the results

The hypothesis is that the learning rate of 0.001 will work the best with 10 epochs, and batch size of 100 and hidden layer rate as [120,60], with Relu activation.

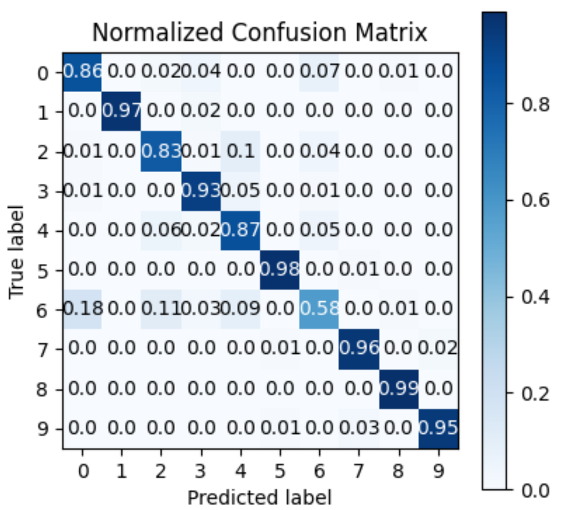
The rest of the parameters cannot be predicted.

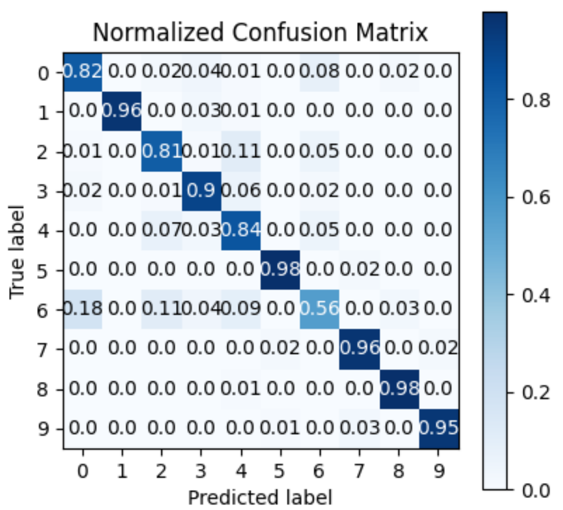
The assumption was that after a few epochs, the model was already working very well. Hence, the model doesn't need to have extremely huge parameters.

C. Execute your plan

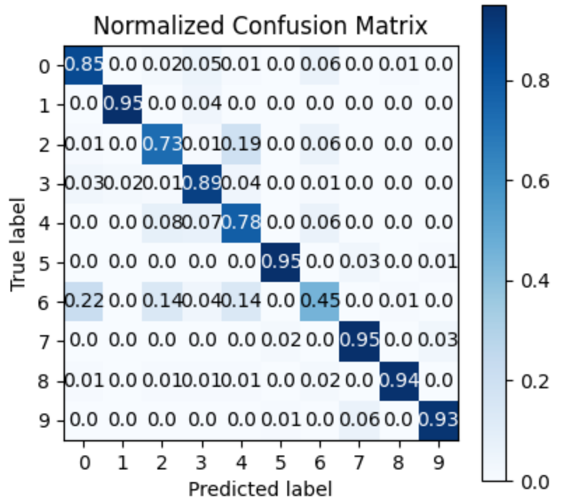
As expected the parameters that perform the best are in accordance with the hypothesis.

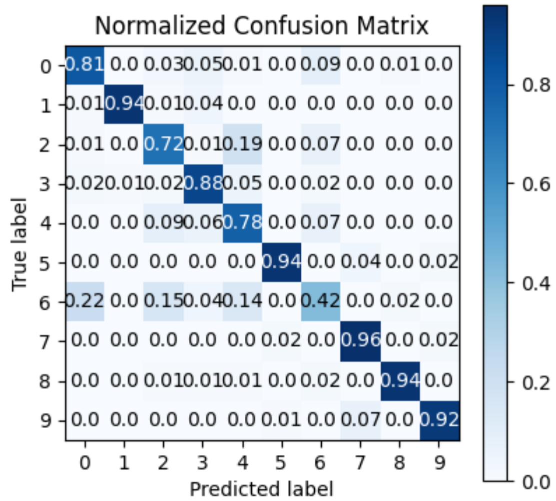
The confusion matrixes of top 5 working models have been shown below.



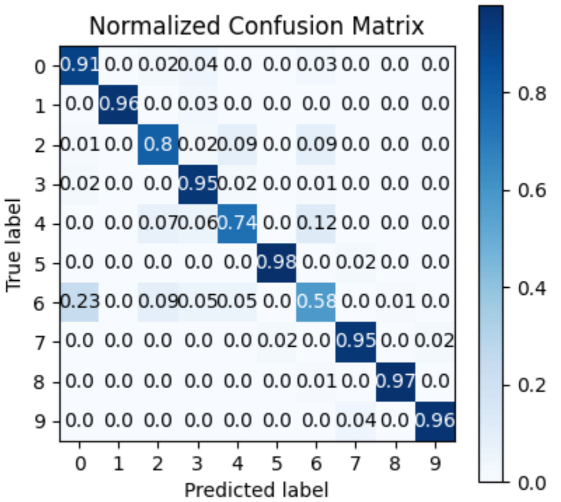


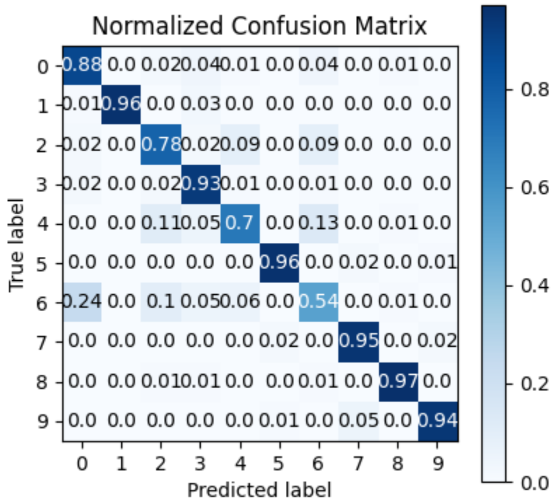
Train accuracy = 0.8926166666666666, Test accuracy = 0.8762



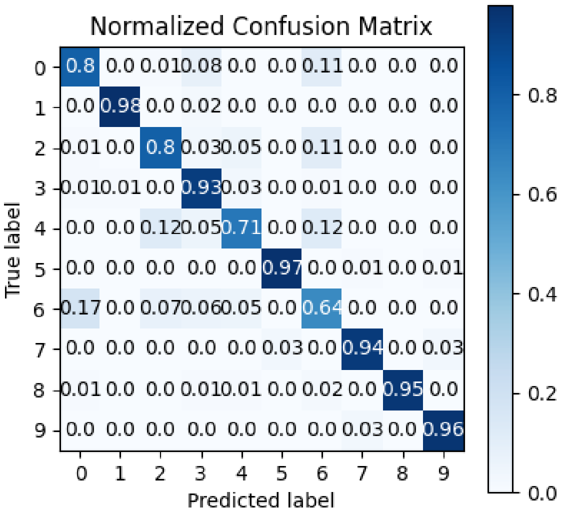


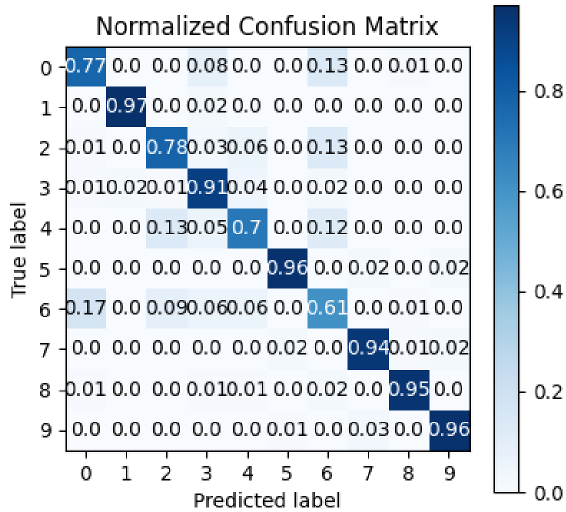
Train accuracy = 0.8439166666666666, Test accuracy = 0.8297



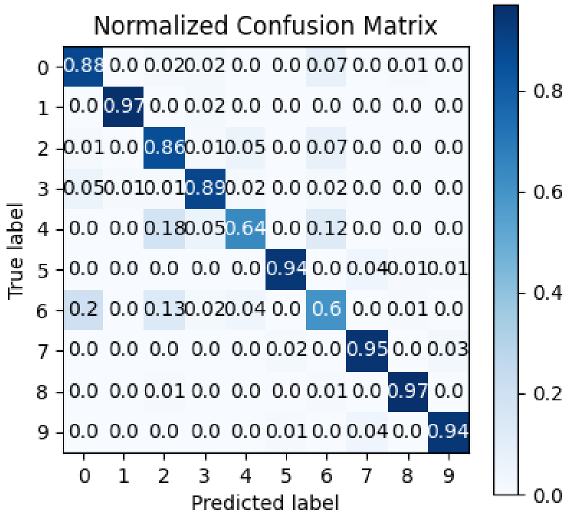


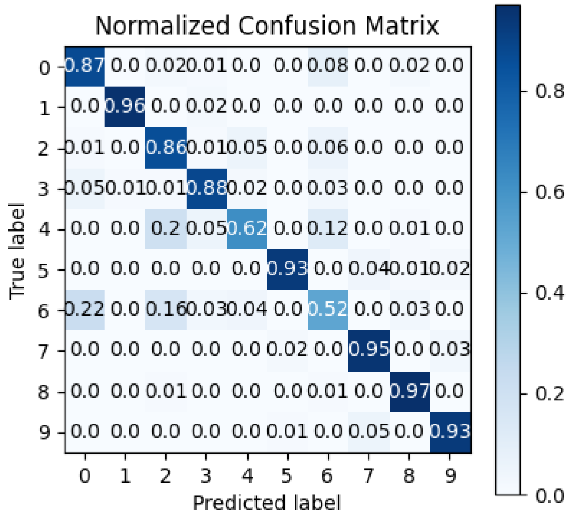
Train accuracy = 0.8798333333333334, Test accuracy = 0.8619





Train accuracy = 0.8669333333333333, Test accuracy = 0.8535





Train accuracy = 0.8633166666666666, Test accuracy = 0.8492

5. Extensions

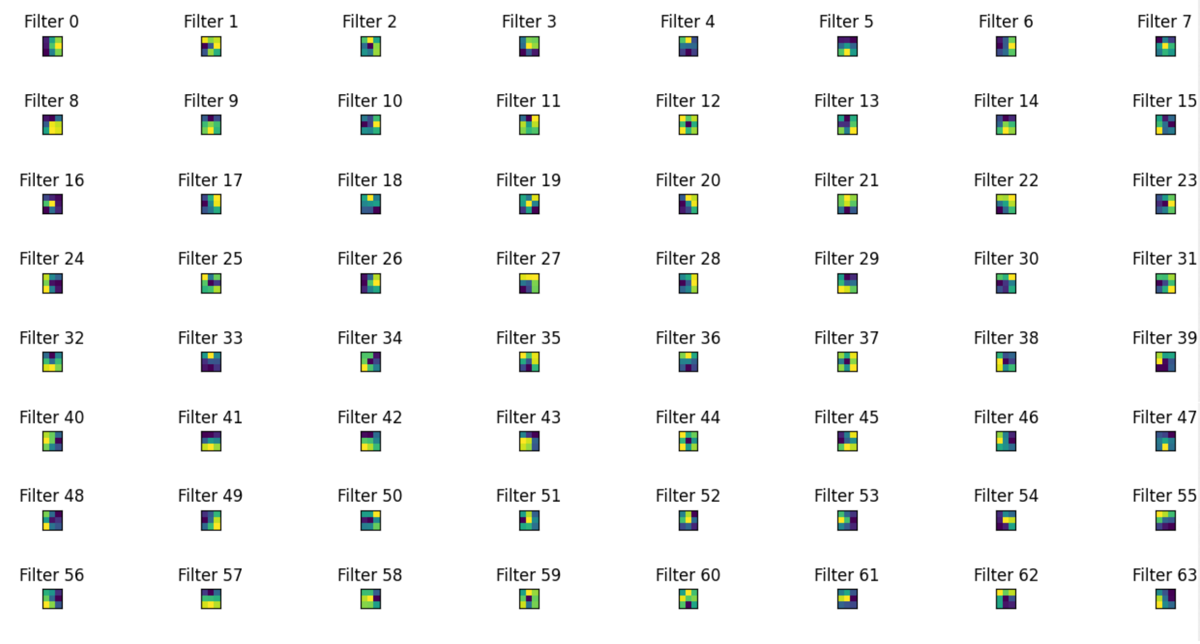
A. There are many pre-trained networks available in the PyTorch package. Try loading one and evaluate its first couple of convolutional layers as in task 2.

Model: VGG16.

Test images – 2 images from CIFAR 10 dataset.

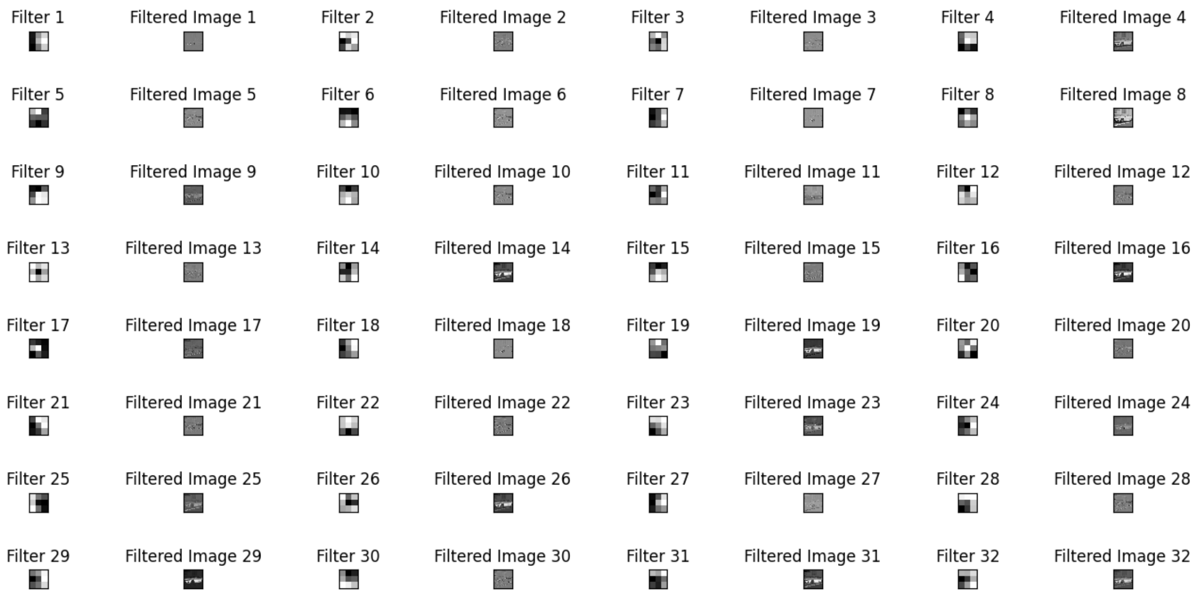
- Shape of layer 1 of the network - torch.Size([64, 3, 3, 3])
- Shape of first filter - torch.Size([3, 3])
- First filter – tensor([
- [-0.5537, 0.1427, 0.5290],
- [-0.5831, 0.3566, 0.7657],
- [-0.6902, -0.0480, 0.4841]], grad_fn=<SelectBackward0>)
- Shape of layer 3 of the network - torch.Size([64, 64, 3, 3])
- Shape of first filter - torch.Size([3, 3])
- First filter - tensor([
- [-0.0306, -0.0985, -0.1326],
- [0.0068, -0.0835, -0.1670],
- [0.0310, -0.0658, -0.1317]], grad_fn=<SelectBackward0>)

Conv 1- filters

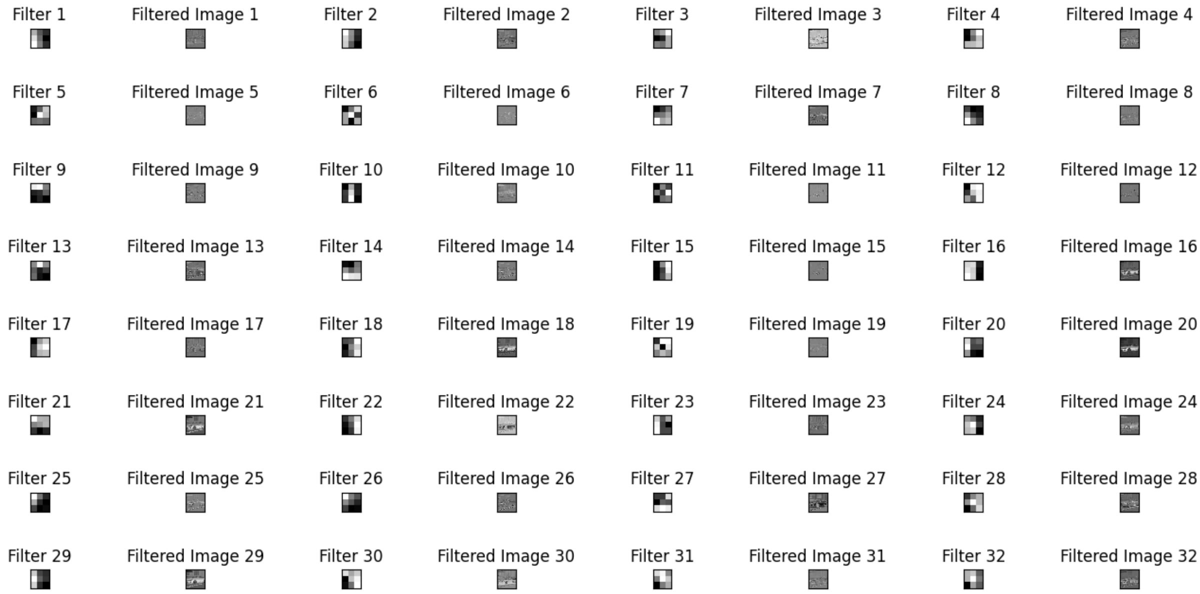


Conv2 -Filters

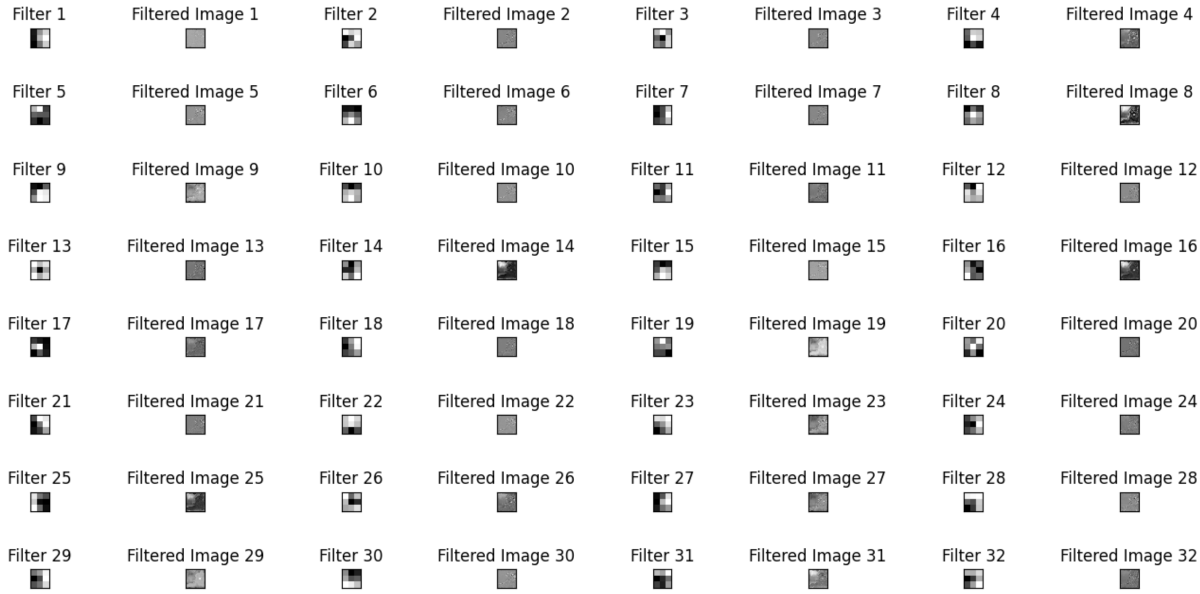
Conv1- filter applied to image – Car from CIFAR10 dataset



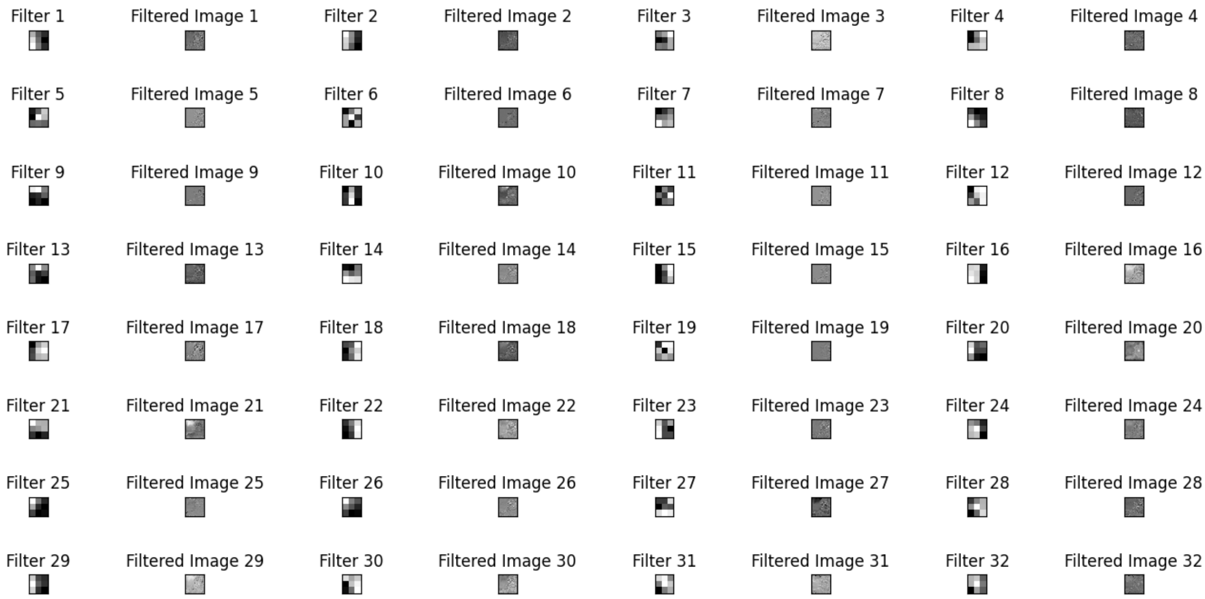
Conv2- filter applied to image – Car from CIFAR10 dataset



Conv1- filter applied to image – Dog from CIFAR10 dataset



Conv2- filter applied to image – Dog from CIFAR10 dataset



B. **Replace the first layer of the MNIST network with a filter bank of your choosing (e.g. Gabor filters) and retrain the rest of the network, holding the first layer constant. How does it do?**

We applied 3 different kinds of filters,

- **Gabor Filter**
- **Laplacian Filter**
- **Gaussian Filter**

The Gabor Filter when applied to first conv layer and frozen performs really bad, with an accuracy of 11% which was not anticipated.

```
Gabor model training
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.304324
Train Epoch: 1 [10000/60000 (17%)]      Loss: 2.293156
Train Epoch: 1 [20000/60000 (33%)]      Loss: 2.295112
Train Epoch: 1 [30000/60000 (50%)]      Loss: 2.292838
Train Epoch: 1 [40000/60000 (67%)]      Loss: 2.296666
Train Epoch: 1 [50000/60000 (83%)]      Loss: 2.301371

Test set: Avg. loss: 2.3012, Accuracy: 1135/10000 (11%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 2.301706
Train Epoch: 2 [10000/60000 (17%)]      Loss: 2.300179
Train Epoch: 2 [20000/60000 (33%)]      Loss: 2.295363
Train Epoch: 2 [30000/60000 (50%)]      Loss: 2.308613
Train Epoch: 2 [40000/60000 (67%)]      Loss: 2.303169
Train Epoch: 2 [50000/60000 (83%)]      Loss: 2.301089

Test set: Avg. loss: 2.3011, Accuracy: 1135/10000 (11%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 2.298507
Train Epoch: 3 [10000/60000 (17%)]      Loss: 2.313024
Train Epoch: 3 [20000/60000 (33%)]      Loss: 2.304189
Train Epoch: 3 [30000/60000 (50%)]      Loss: 2.291450
Train Epoch: 3 [40000/60000 (67%)]      Loss: 2.296810
Train Epoch: 3 [50000/60000 (83%)]      Loss: 2.298209

Test set: Avg. loss: 2.3011, Accuracy: 1135/10000 (11%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 2.300630
Train Epoch: 4 [10000/60000 (17%)]      Loss: 2.300376
Train Epoch: 4 [20000/60000 (33%)]      Loss: 2.310190
Train Epoch: 4 [30000/60000 (50%)]      Loss: 2.293983
Train Epoch: 4 [40000/60000 (67%)]      Loss: 2.292970
Train Epoch: 4 [50000/60000 (83%)]      Loss: 2.303795
```

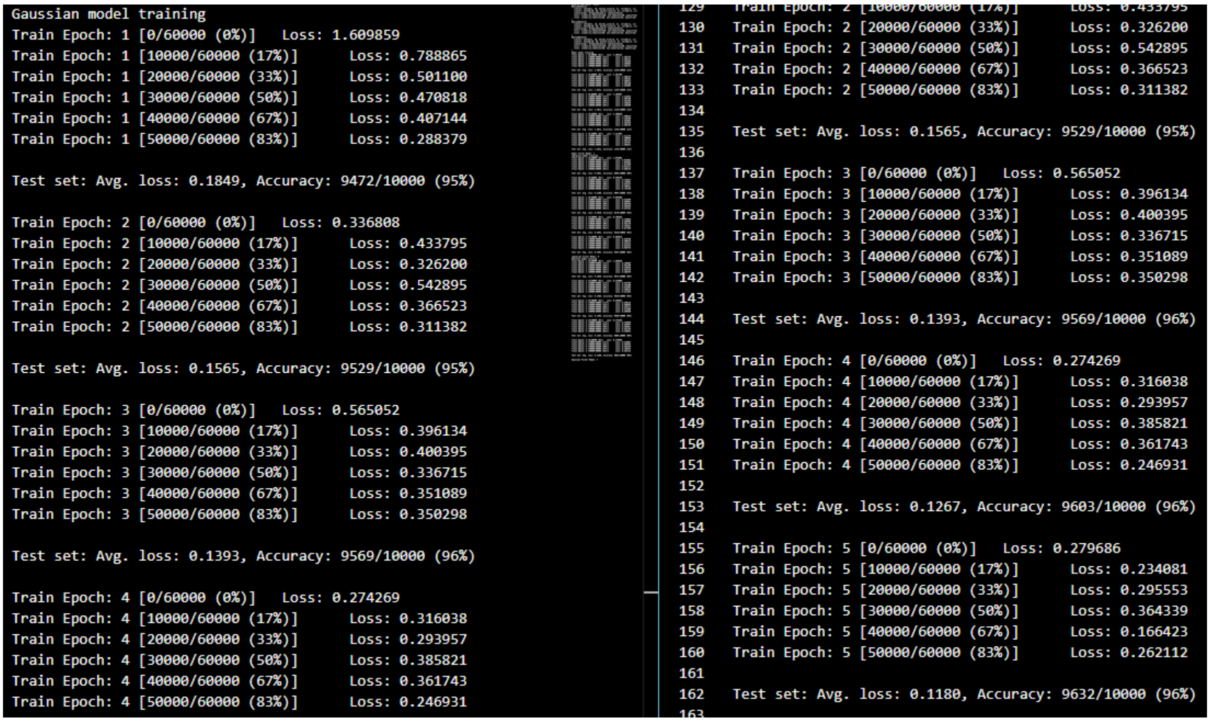
```
Train Epoch: 2 [10000/60000 (17%)]      Loss: 2.300179
Train Epoch: 2 [20000/60000 (33%)]      Loss: 2.295363
Train Epoch: 2 [30000/60000 (50%)]      Loss: 2.308613
Train Epoch: 2 [40000/60000 (67%)]      Loss: 2.303169
Train Epoch: 2 [50000/60000 (83%)]      Loss: 2.301089

Test set: Avg. loss: 2.3011, Accuracy: 1135/10000 (11%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 2.298507
Train Epoch: 3 [10000/60000 (17%)]      Loss: 2.313024
Train Epoch: 3 [20000/60000 (33%)]      Loss: 2.304189
Train Epoch: 3 [30000/60000 (50%)]      Loss: 2.291450
Train Epoch: 3 [40000/60000 (67%)]      Loss: 2.296810
Train Epoch: 3 [50000/60000 (83%)]      Loss: 2.298209

Test set: Avg. loss: 2.3011, Accuracy: 1135/10000 (11%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 2.300630
Train Epoch: 4 [10000/60000 (17%)]      Loss: 2.300376
Train Epoch: 4 [20000/60000 (33%)]      Loss: 2.310190
Train Epoch: 4 [30000/60000 (50%)]      Loss: 2.293983
Train Epoch: 4 [40000/60000 (67%)]      Loss: 2.292970
Train Epoch: 4 [50000/60000 (83%)]      Loss: 2.303795

Test set: Avg. loss: 2.3011, Accuracy: 1135/10000 (11%)

Train Epoch: 5 [0/60000 (0%)]   Loss: 2.311693
Train Epoch: 5 [10000/60000 (17%)]      Loss: 2.311175
Train Epoch: 5 [20000/60000 (33%)]      Loss: 2.308060
Train Epoch: 5 [30000/60000 (50%)]      Loss: 2.304125
Train Epoch: 5 [40000/60000 (67%)]      Loss: 2.297257
Train Epoch: 5 [50000/60000 (83%)]      Loss: 2.302872

Test set: Avg. loss: 2.3011, Accuracy: 1135/10000 (11%)
```

The Laplacian filter performs fairly better with an accuracy of 93%

```
Laplacian model training
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.078504
Train Epoch: 1 [10000/60000 (17%)]      Loss: 0.943062
Train Epoch: 1 [20000/60000 (33%)]      Loss: 1.045826
Train Epoch: 1 [30000/60000 (50%)]      Loss: 1.013499
Train Epoch: 1 [40000/60000 (67%)]      Loss: 0.922272
Train Epoch: 1 [50000/60000 (83%)]      Loss: 0.686135

Test set: Avg. loss: 0.3931, Accuracy: 8892/10000 (89%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 0.559149
Train Epoch: 2 [10000/60000 (17%)]      Loss: 0.724622
Train Epoch: 2 [20000/60000 (33%)]      Loss: 0.720831
Train Epoch: 2 [30000/60000 (50%)]      Loss: 0.678157
Train Epoch: 2 [40000/60000 (67%)]      Loss: 0.700722
Train Epoch: 2 [50000/60000 (83%)]      Loss: 0.723783

Test set: Avg. loss: 0.3109, Accuracy: 9097/10000 (91%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 0.697680
Train Epoch: 3 [10000/60000 (17%)]      Loss: 0.523949
Train Epoch: 3 [20000/60000 (33%)]      Loss: 0.569957
Train Epoch: 3 [30000/60000 (50%)]      Loss: 0.607950
Train Epoch: 3 [40000/60000 (67%)]      Loss: 0.568528
Train Epoch: 3 [50000/60000 (83%)]      Loss: 0.527512

Test set: Avg. loss: 0.2870, Accuracy: 9178/10000 (92%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 0.705846
Train Epoch: 4 [10000/60000 (17%)]      Loss: 0.739026
Train Epoch: 4 [20000/60000 (33%)]      Loss: 0.686989
Train Epoch: 4 [30000/60000 (50%)]      Loss: 0.552179
Train Epoch: 4 [40000/60000 (67%)]      Loss: 0.764357
Train Epoch: 4 [50000/60000 (83%)]      Loss: 0.617486
```

```
Train Epoch: 2 [10000/60000 (17%)]      Loss: 0.724622
Train Epoch: 2 [20000/60000 (33%)]      Loss: 0.720831
Train Epoch: 2 [30000/60000 (50%)]      Loss: 0.678157
Train Epoch: 2 [40000/60000 (67%)]      Loss: 0.700722
Train Epoch: 2 [50000/60000 (83%)]      Loss: 0.723783

Test set: Avg. loss: 0.3109, Accuracy: 9097/10000 (91%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 0.697680
Train Epoch: 3 [10000/60000 (17%)]      Loss: 0.523949
Train Epoch: 3 [20000/60000 (33%)]      Loss: 0.569957
Train Epoch: 3 [30000/60000 (50%)]      Loss: 0.607950
Train Epoch: 3 [40000/60000 (67%)]      Loss: 0.568528
Train Epoch: 3 [50000/60000 (83%)]      Loss: 0.527512

Test set: Avg. loss: 0.2870, Accuracy: 9178/10000 (92%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 0.705846
Train Epoch: 4 [10000/60000 (17%)]      Loss: 0.739026
Train Epoch: 4 [20000/60000 (33%)]      Loss: 0.686989
Train Epoch: 4 [30000/60000 (50%)]      Loss: 0.552179
Train Epoch: 4 [40000/60000 (67%)]      Loss: 0.764357
Train Epoch: 4 [50000/60000 (83%)]      Loss: 0.617486

Test set: Avg. loss: 0.2641, Accuracy: 9226/10000 (92%)

Train Epoch: 5 [0/60000 (0%)]   Loss: 0.584815
Train Epoch: 5 [10000/60000 (17%)]      Loss: 0.549709
Train Epoch: 5 [20000/60000 (33%)]      Loss: 0.409578
Train Epoch: 5 [30000/60000 (50%)]      Loss: 0.547813
Train Epoch: 5 [40000/60000 (67%)]      Loss: 0.644324
Train Epoch: 5 [50000/60000 (83%)]      Loss: 0.555774

Test set: Avg. loss: 0.2487, Accuracy: 9257/10000 (93%)
```

The Gaussian filter works the best with an accuracy of 95%.

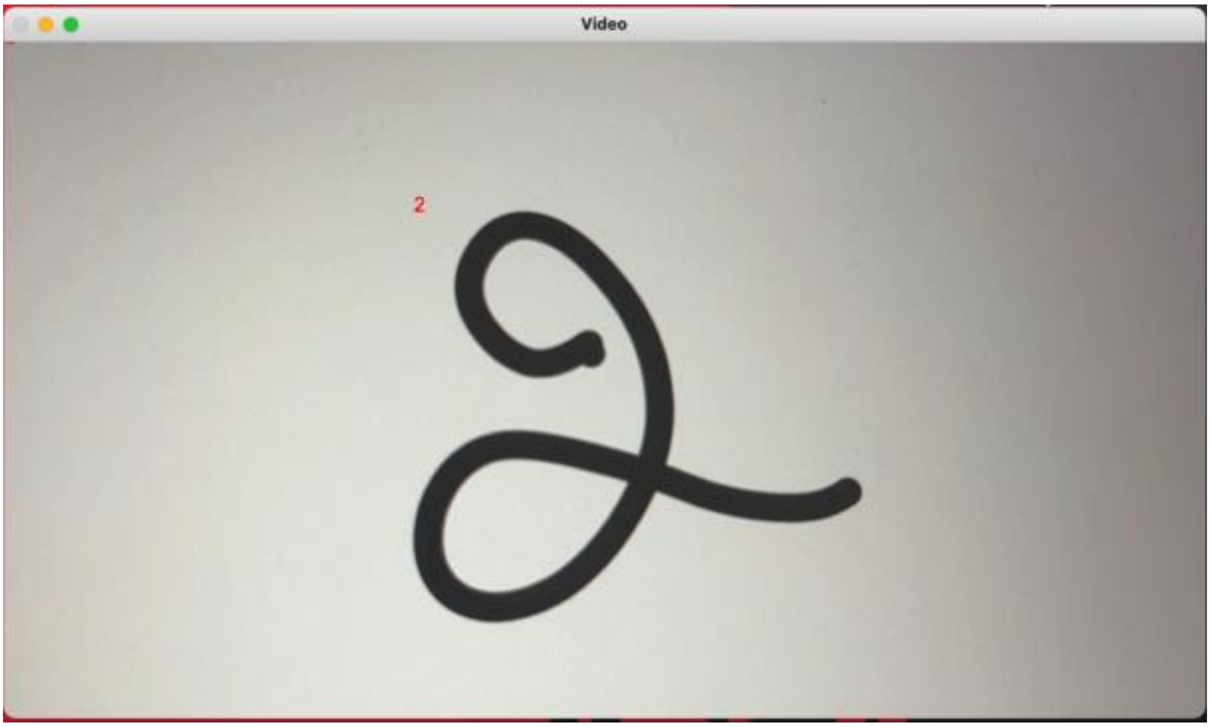


```
Gaussian model training
Train Epoch: 1 [0/60000 (0%)]   Loss: 1.609859
Train Epoch: 1 [10000/60000 (17%)]      Loss: 0.788865
Train Epoch: 1 [20000/60000 (33%)]      Loss: 0.501100
Train Epoch: 1 [30000/60000 (50%)]      Loss: 0.470818
Train Epoch: 1 [40000/60000 (67%)]      Loss: 0.407144
Train Epoch: 1 [50000/60000 (83%)]      Loss: 0.288379

Test set: Avg. loss: 0.1849, Accuracy: 9472/10000 (95%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 0.336808
Train Epoch: 2 [10000/60000 (17%)]      Loss: 0.433795
Train Epoch: 2 [20000/60000 (33%)]      Loss: 0.326200
Train Epoch: 2 [30000/60000 (50%)]      Loss: 0.542895
Train Epoch: 2 [40000/60000 (67%)]      Loss: 0.366523
Train Epoch: 2 [50000/60000 (83%)]      Loss: 0.311382

Test set: Avg. loss: 0.1565, Accuracy: 9529/10000 (95%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 0.565052
Train Epoch: 3 [10000/60000 (17%)]      Loss: 0.396134
Train Epoch: 3 [20000/60000 (33%)]      Loss: 0.400395
Train Epoch: 3 [30000/60000 (50%)]      Loss: 0.336715
Train Epoch: 3 [40000/60000 (67%)]      Loss: 0.351089
Train Epoch: 3 [50000/60000 (83%)]      Loss: 0.350298

Test set: Avg. loss: 0.1393, Accuracy: 9569/10000 (96%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 0.274269
Train Epoch: 4 [10000/60000 (17%)]      Loss: 0.316038
Train Epoch: 4 [20000/60000 (33%)]      Loss: 0.293957
Train Epoch: 4 [30000/60000 (50%)]      Loss: 0.385821
Train Epoch: 4 [40000/60000 (67%)]      Loss: 0.361743
Train Epoch: 4 [50000/60000 (83%)]      Loss: 0.246931
```

```
Train Epoch: 2 [10000/60000 (17%)]      Loss: 0.433795
Train Epoch: 2 [20000/60000 (33%)]      Loss: 0.326200
Train Epoch: 2 [30000/60000 (50%)]      Loss: 0.542895
Train Epoch: 2 [40000/60000 (67%)]      Loss: 0.366523
Train Epoch: 2 [50000/60000 (83%)]      Loss: 0.311382

Test set: Avg. loss: 0.1565, Accuracy: 9529/10000 (95%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 0.565052
Train Epoch: 3 [10000/60000 (17%)]      Loss: 0.396134
Train Epoch: 3 [20000/60000 (33%)]      Loss: 0.400395
Train Epoch: 3 [30000/60000 (50%)]      Loss: 0.336715
Train Epoch: 3 [40000/60000 (67%)]      Loss: 0.351089
Train Epoch: 3 [50000/60000 (83%)]      Loss: 0.350298

Test set: Avg. loss: 0.1393, Accuracy: 9569/10000 (96%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 0.274269
Train Epoch: 4 [10000/60000 (17%)]      Loss: 0.316038
Train Epoch: 4 [20000/60000 (33%)]      Loss: 0.293957
Train Epoch: 4 [30000/60000 (50%)]      Loss: 0.385821
Train Epoch: 4 [40000/60000 (67%)]      Loss: 0.361743
Train Epoch: 4 [50000/60000 (83%)]      Loss: 0.246931

Test set: Avg. loss: 0.1267, Accuracy: 9603/10000 (96%)

Train Epoch: 5 [0/60000 (0%)]   Loss: 0.279686
Train Epoch: 5 [10000/60000 (17%)]      Loss: 0.234081
Train Epoch: 5 [20000/60000 (33%)]      Loss: 0.295553
Train Epoch: 5 [30000/60000 (50%)]      Loss: 0.364339
Train Epoch: 5 [40000/60000 (67%)]      Loss: 0.166423
Train Epoch: 5 [50000/60000 (83%)]      Loss: 0.262112

Test set: Avg. loss: 0.1180, Accuracy: 9632/10000 (96%)
```

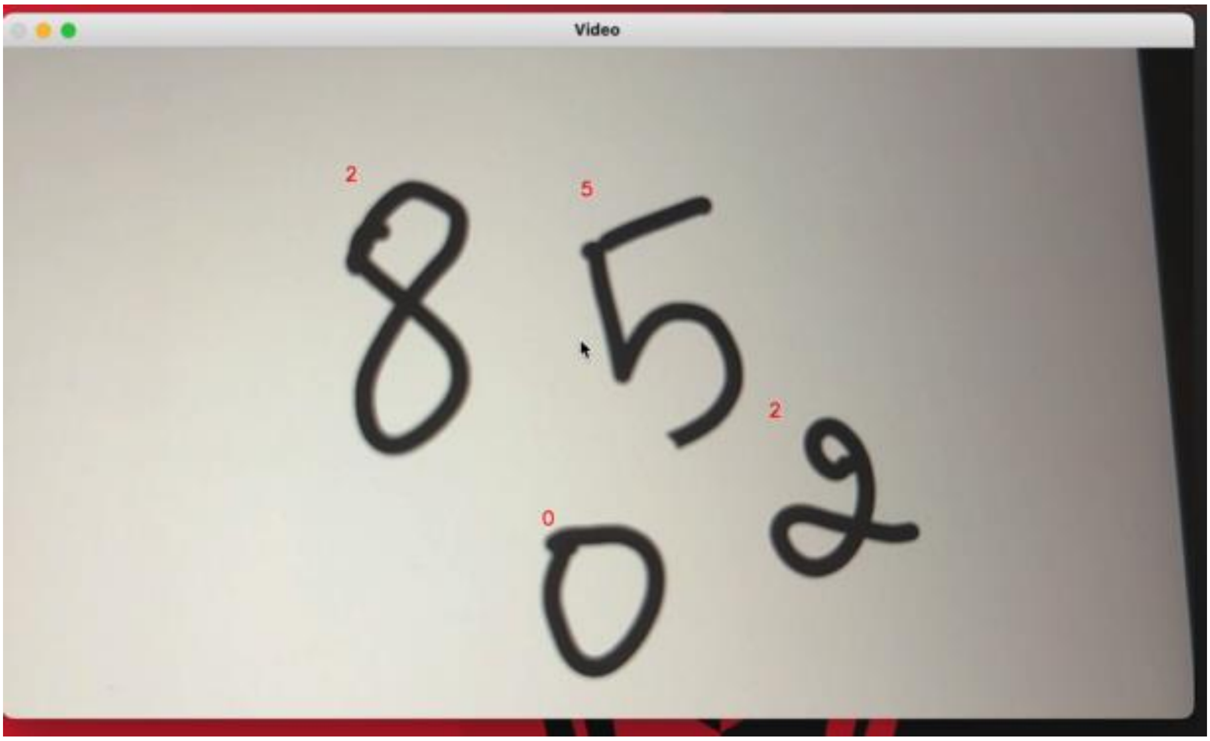
C. Build a live video digit recognition application using the trained network.

The application was developed, to recognize a single digit in the live video stream.



D. **Build a live video Multi digit recognition application using the trained network.**

The previous extension was again extended to recognize Multiple digits in the same frame.



E. **Evaluate more dimensions on task 4 – 1024 Models.**
A total of 1024 models were evaluated, with the best performing model having an accuracy of 87% on the Fashion MNIST dataset. The results for all runs with all hyperparamteres can be seen in results.csv and results.json

6. Short Reflection

The project allowed us to understand all the intricasies involved in pytorch. We were both extensive users of tensorflow and were new to pytorch, but now we're able to understand and apply all techniques in Pytorch too. The library is very effective and provided great flexibility.
Additionally, it was very useful to learn hyper parameter tuning, and test out various hypothesis.

7. References

- PyTorch documentation - https://pytorch.org/docs/stable/index.html
- PyTorch tutorials - https://pytorch.org/tutorials/
- torchvision documentation - https://pytorch.org/vision/stable/index.html
- MNIST dataset website - http://yann.lecun.com/exdb/mnist/
- Matplotlib documentation - https://matplotlib.org/stable/contents.html
- OpenCV documentation - https://docs.opencv.org/master/
- NumPy documentation - https://numpy.org/doc/stable/
- Pandas documentation - https://pandas.pydata.org/docs/
- Seaborn documentation - https://seaborn.pydata.org/documentation.html
- SciPy documentation - https://docs.scipy.org/doc/
- Jupyter Notebook documentation - https://jupyter-notebook.readthedocs.io/en/stable/
- Google Colab documentation - https://colab.research.google.com/notebooks/intro.ipynb
- GitHub documentation - https://docs.github.com/en
- Git documentation - https://git-scm.com/doc