CS5330: PRCV
Project 5: Recognition using Deep Networks
Report

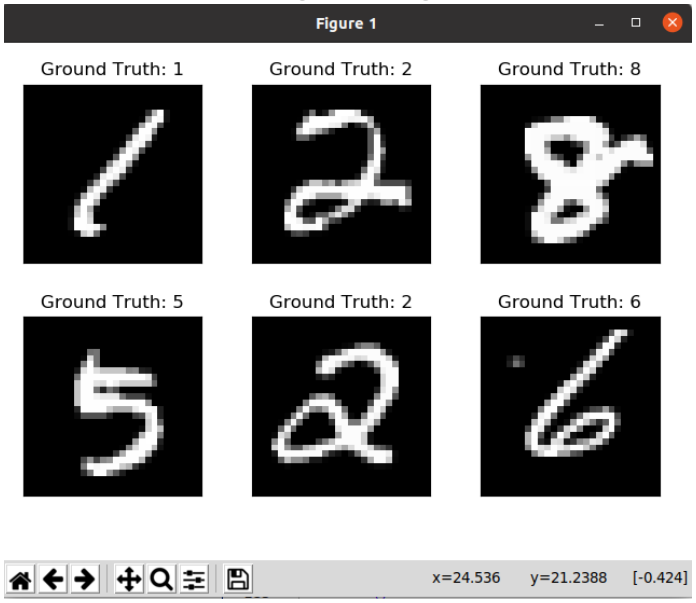
Tejaswini Dilip Deore

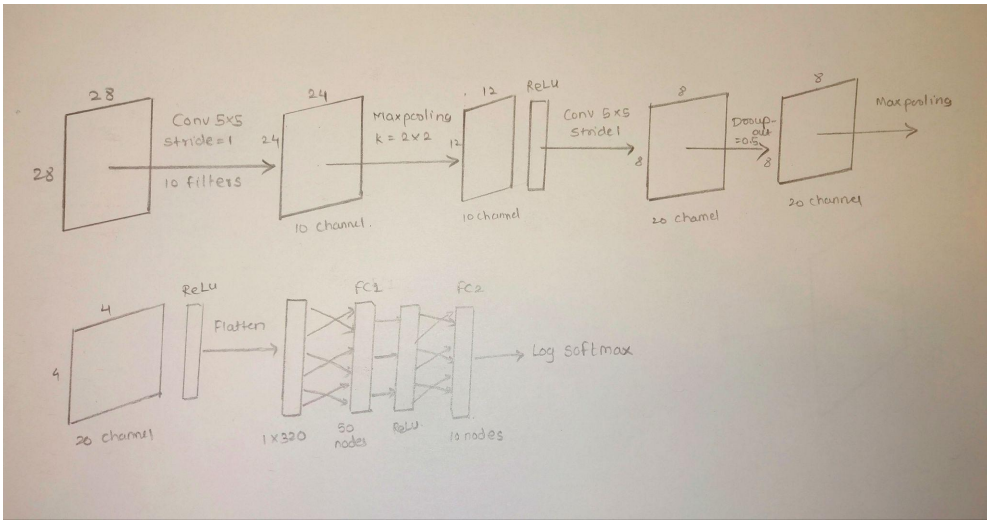
Project Description:

This project involves building, training, analyzing, and modifying a deep network for a recognition task. The project uses MNIST digit recognition data set and PyTorch to build and train a network for digit recognition. This network is then re-used to perform transfer learning on Greek letters (alpha, beta and gamma). The project also includes and experiment which involves changing different aspects of the network (number of epochs, batch size, number of convolution layers, filter size and dropout rate).

As an extension, the project evaluates more than 3 network dimensions. It also evaluates the first two convolution layers of ResNet18 and analyzes 64 filters and analyzes the network performance after replacing the first layer of the MNIST network with Gabor filters.

1. Build and train a network to recognize digits:




28
28
Conv 5x5
stride=1
10 filters
24
24
10 channel
Maxpooling
k = 2x2
12
12
10 channel
ReLU
Conv 5x5
stride 1
8
8
20 channel
Drop-out
=0.5
8
8
20 channel
Maxpooling
4
4
20 channel
ReLU
Flatten
1x320
FC1
50
nodes
ReLU
FC2
10 nodes
Log softmax

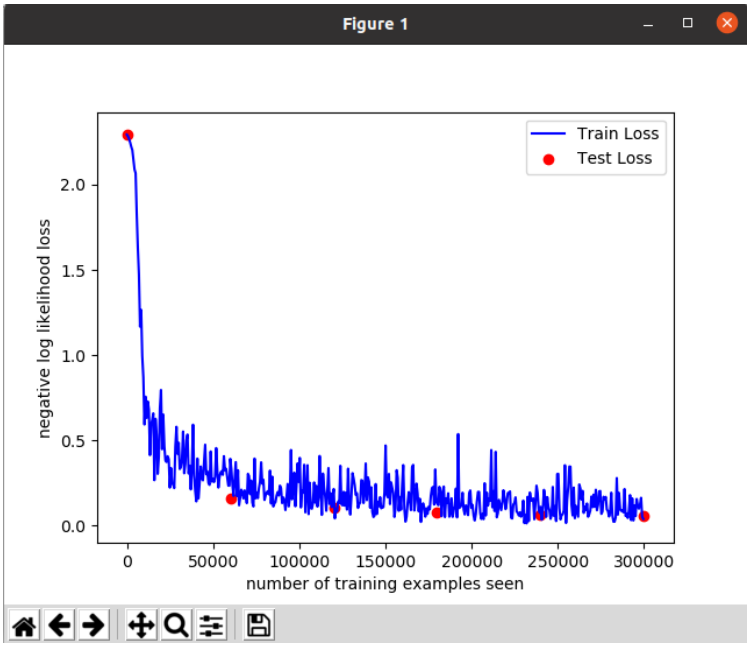
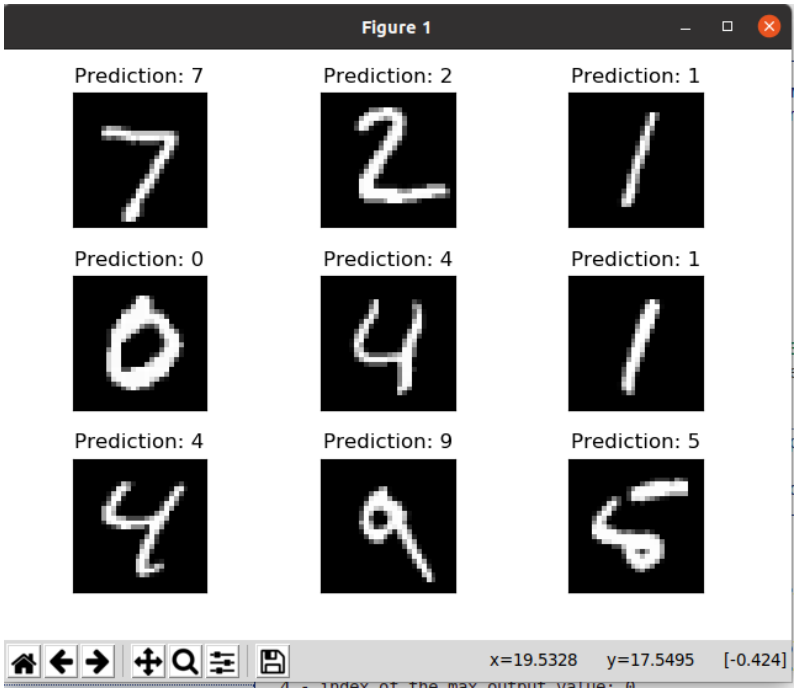
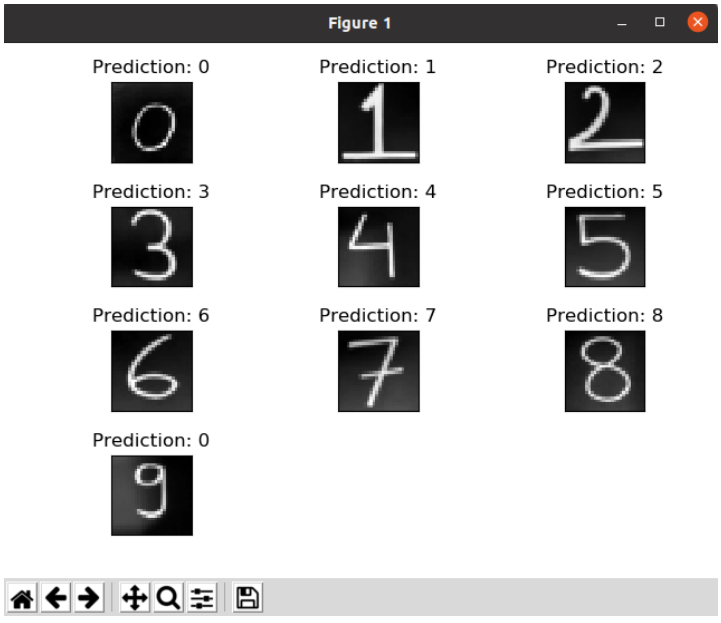


Figure 1
Train Loss
Test Loss
negative log likelihood loss
2.0
1.5
1.0
0.5
0.0
0
50000
100000
150000
200000
250000
300000
number of training examples seen




```
tejaswini@tejaswini-Dell-G15:~/PRCV/Project5$ python3 task1_Test.py
1 - output: tensor([[-1.7550e+01, -1.8199e+01, -1.0724e+01, -1.2678e+01, -1.9541e+01,
         -2.0132e+01, -3.3235e+01, -4.0769e-05, -1.4985e+01, -1.1083e+01]])
1 - index of the max output value: 7
1 - prediction label: 7
2 - output: tensor([[-7.8880e+00, -7.0340e+00, -1.2921e-03, -1.1380e+01, -1.8322e+01,
         -1.9586e+01, -1.0798e+01, -1.6321e+01, -1.2834e+01, -2.2834e+01]])
2 - index of the max output value: 2
2 - prediction label: 2
3 - output: tensor([[-1.3214e+01, -5.4535e-04, -9.7363e+00, -1.0298e+01, -9.0555e+00,
         -1.4353e+01, -1.1055e+01, -8.3385e+00, -9.5280e+00, -1.2110e+01]])
3 - index of the max output value: 1
3 - prediction label: 1
4 - output: tensor([[-1.3113e-05, -2.2098e+01, -1.2678e+01, -1.7592e+01, -1.9877e+01,
         -1.3335e+01, -1.2165e+01, -1.5272e+01, -1.5888e+01, -1.2796e+01]])
4 - index of the max output value: 0
4 - prediction label: 0
5 - output: tensor([[-1.8126e+01, -1.6752e+01, -1.3620e+01, -1.7192e+01, -1.1944e-04,
         -1.8672e+01, -1.4201e+01, -1.4974e+01, -1.7266e+01, -9.0524e+00]])
5 - index of the max output value: 4
5 - prediction label: 4
6 - output: tensor([[-1.4738e+01, -2.3982e-04, -1.1262e+01, -1.1007e+01, -1.0427e+01,
         -1.6586e+01, -1.4244e+01, -9.0974e+00, -9.7117e+00, -1.1862e+01]])
6 - index of the max output value: 1
6 - prediction label: 1
7 - output: tensor([[-2.0919e+01, -1.0945e+01, -1.4066e+01, -1.3052e+01, -3.6030e-03,
         -1.2727e+01, -1.7911e+01, -8.8112e+00, -8.3030e+00, -5.7521e+00]])
7 - index of the max output value: 4
7 - prediction label: 4
8 - output: tensor([[-1.4936e+01, -1.3722e+01, -8.2668e+00, -7.5919e+00, -6.1818e+00,
         -7.6318e+00, -1.8805e+01, -9.7791e+00, -7.8891e+00, -3.7528e-03]])
8 - index of the max output value: 9
8 - prediction label: 9
9 - output: tensor([[ -9.3311, -18.6436, -11.8668, -14.8775, -12.6022,  -0.1202,  -2.2036,
         -17.5313,  -7.4536,  -6.1417]])
9 - index of the max output value: 5
9 - prediction label: 5
10 - output: tensor([[-1.5934e+01, -2.0789e+01, -1.4804e+01, -1.1365e+01, -1.0811e+01,
         -1.3742e+01, -2.3116e+01, -6.7195e+00, -7.0022e+00, -2.1527e-03]])
10 - index of the max output value: 9
10 - prediction label: 9
```

Figure below shows the test results of a handwritten digit dataset. The images of this set were converted to grayscale, resized (28x28) and inverted to match the MNIST dataset images which are white on black. The accuracy of 98.05% is achieved.

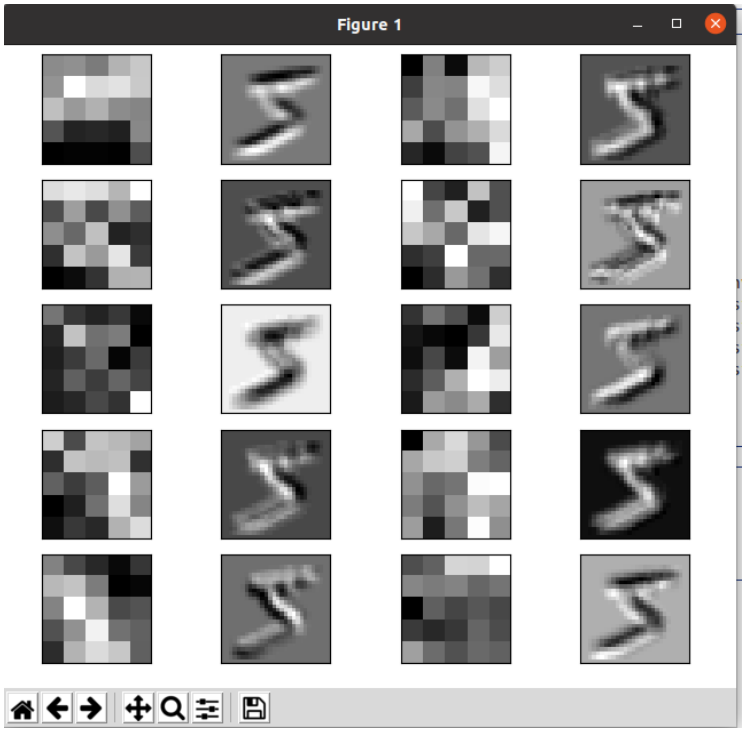



```
1 - output: tensor([[-0.5095, -3.7165, -2.3127, -3.5751, -3.8592, -3.3535, -2.9724, -3.2889,
        -3.1392, -2.8127]])
1 - index of the max output value: 0
1 - prediction label: 0
2 - output: tensor([[-8.3932, -0.1175, -4.7835, -3.9342, -6.1690, -6.3248, -8.7731, -2.8238,
        -4.1564, -5.6262]])
2 - index of the max output value: 1
2 - prediction label: 1
3 - output: tensor([[-2.5971, -2.6488, -0.2060, -5.6144, -6.2123, -5.6944, -3.5996, -7.1768,
        -5.5840, -9.1321]])
3 - index of the max output value: 2
3 - prediction label: 2
4 - output: tensor([[-12.0884,  -8.7077,  -5.0144,  -0.0220,  -4.8371,  -8.5655, -12.3784,
         -8.7887,  -5.2253,  -6.6812]])
4 - index of the max output value: 3
4 - prediction label: 3
5 - output: tensor([[-7.6302, -6.4397, -5.4604, -5.4218, -0.0910, -5.6291, -7.4308, -5.3073,
        -5.4245, -2.7695]])
5 - index of the max output value: 4
5 - prediction label: 4
6 - output: tensor([[-3.6514, -6.1892, -3.0219, -1.6987, -5.4768, -0.4852, -4.6665, -2.9769,
        -2.9179, -5.0879]])
6 - index of the max output value: 5
6 - prediction label: 5
7 - output: tensor([[ -5.5460, -10.1870,  -5.4955,  -7.4566,  -4.7531,  -3.0541,  -0.0896,
        -12.4602,  -3.8983,  -6.9278]])
7 - index of the max output value: 6
7 - prediction label: 6
8 - output: tensor([[-8.8952, -1.6932, -2.1182, -1.7211, -2.0410, -7.5177, -9.5654, -1.1030,
        -3.0836, -4.7533]])
8 - index of the max output value: 7
8 - prediction label: 7
9 - output: tensor([[-2.7302, -5.0428, -2.1167, -1.5613, -3.4610, -2.7670, -3.7294, -4.4647,
        -0.7772, -4.7596]])
9 - index of the max output value: 8
9 - prediction label: 8
10 - output: tensor([[-1.0880, -2.2551, -1.1851, -2.0468, -4.1690, -3.6914, -4.9360, -3.6101,
        -3.7398, -3.6897]])
10 - index of the max output value: 0
10 - prediction label: 0
```

2. Examine your network:

Figure below shows filters from the first layer of the network and the structure of the network.

Figure below shows filtered images. The obtained results make sense for the given filters. If we look at the 1st filter (1,1), we can observe that the top pixels are bright while the bottom ones are dark, hence it is highlighting horizontal edges. Similarly, the second filter (1,2) has dark pixels on the left and bright pixels on the right, highlighting vertical edges.



3. Transfer Learning on Greek Letters:

This task re-uses the MNIST digit recognition network you built in task 1 to recognize three different greek letters: alpha, beta, and gamma. The figure below shows training loss of the network.

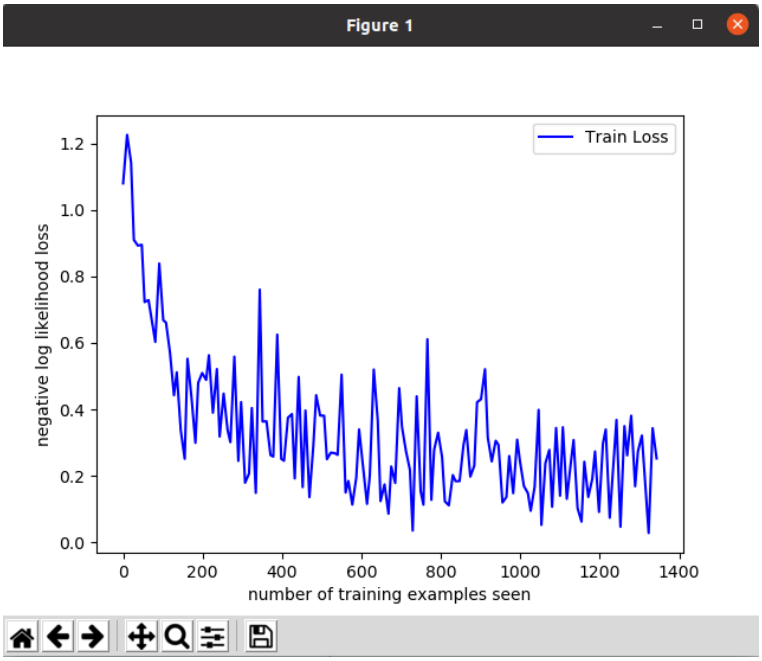
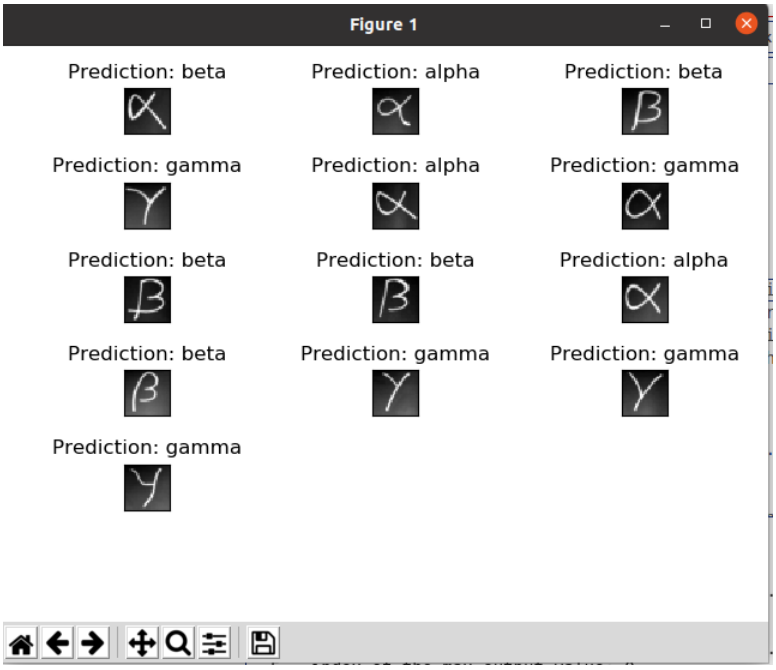


Figure below shows a modified network with updated last layer.

```
tejaswini@tejaswini-Dell-G15:~/PRCV/Project5$ python3 task3.py
MyNetwork(
  (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=3, bias=True)
)
```

The figure below shows results of testing the model on additional handwritten greek letter images. This test set consists of 13 images. The network identifies 12/13 letters correctly. To achieve high accuracy, a larger dataset is required.



4. Design your own experiment:

This involves 64 experiments with 5 dimensions as follows:
epoch sizes: 3, 5
training batch sizes: 64, 128
the number of convolution layers: additional 1 - 4 convolution layers
convolution layer filter size: 3, 5
dropout rate: 0.3, 0.5

Hypothesis:

As the epoch size increases, the accuracy is expected to improve since the weights are trained more times. Similarly, doubling the training batch size should also result in improved accuracy due to the increased amount of training data provided. When the number of convolution layers is increased, the accuracy is also expected to improve since more features can be analyzed. For optimal performance, it is recommended to use a mid-size convolution layer filter, as both larger and smaller filters may decrease accuracy. To address overfitting, an appropriate dropout rate

should be chosen. For the MNIST dataset, both 0.3 and 0.5 are acceptable rates, although the difference in performance may not be significant for smaller datasets such as MNIST.

Evaluation:
The results are following my hypothesis. Although, since MNIST is a smaller dataset, results aren't that significantly noticeable. The results from all the 64 experiments were saved in a csv file and compared.
The maximum accuracy achieved is 98.03% with following dimensions:
epoch sizes: 5
training batch sizes: 64
the number of convolution layers: 1
convolution layer filter size: 5
dropout rate: 0.3

6. Extensions:

A. Evaluate more dimensions:

For implementation of this, refer to explanation of task 4, where 5 dimensions have been evaluated.

B. Load and evaluate first two conv layers of ResNet18:

Figure below shows 64 filters of ResNet18.

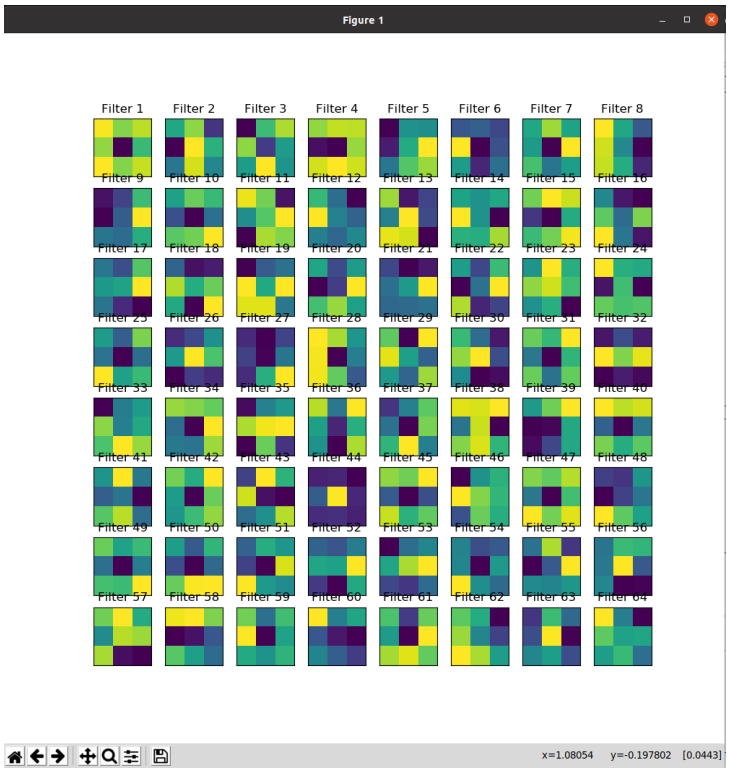
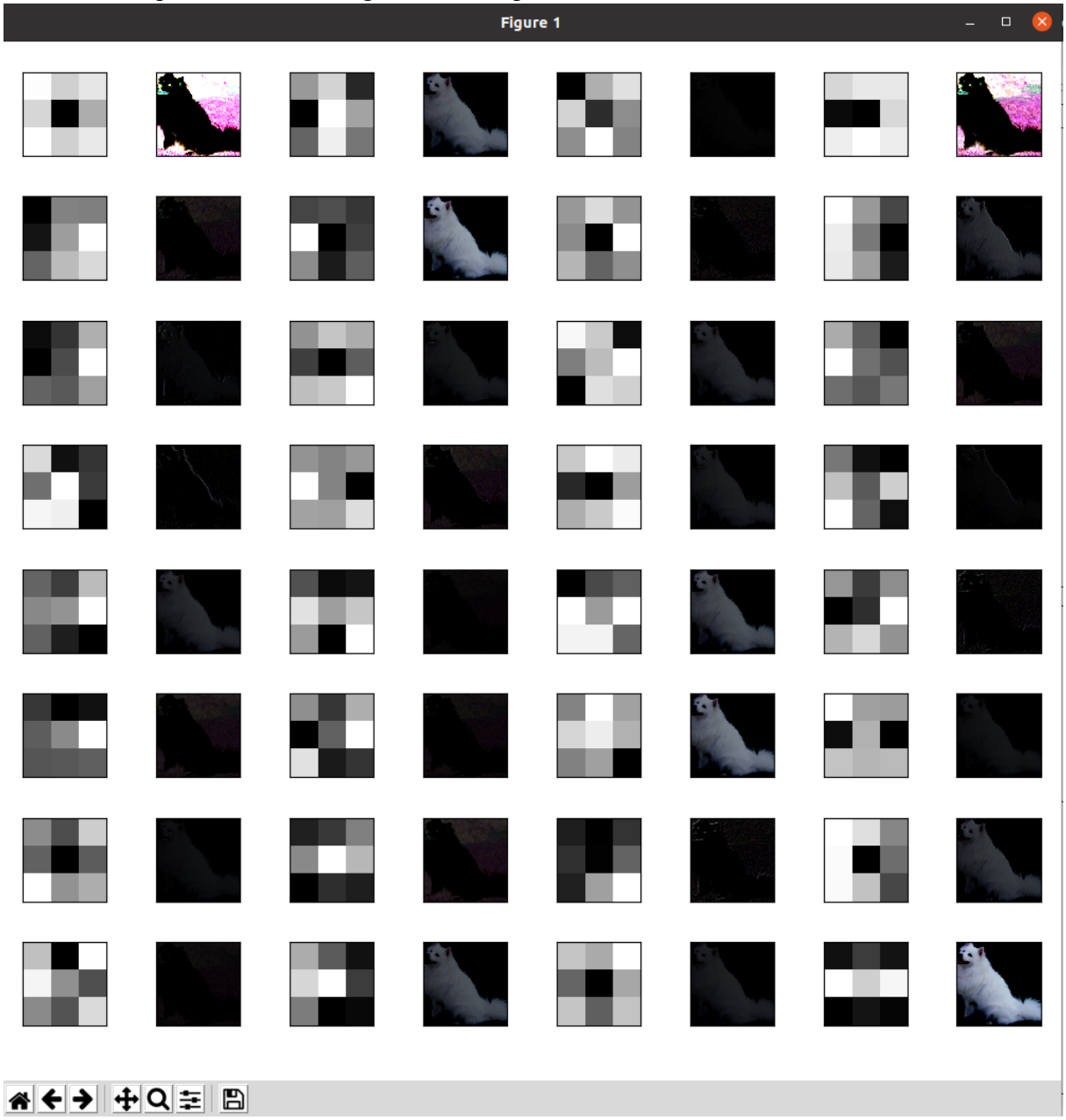


Figure below shows dog images with application of first 32 filters (64 filters were not clearly visible on plot). The filters appear to be separating foreground from background as we can see that the dog is being separated from the background. The model focuses

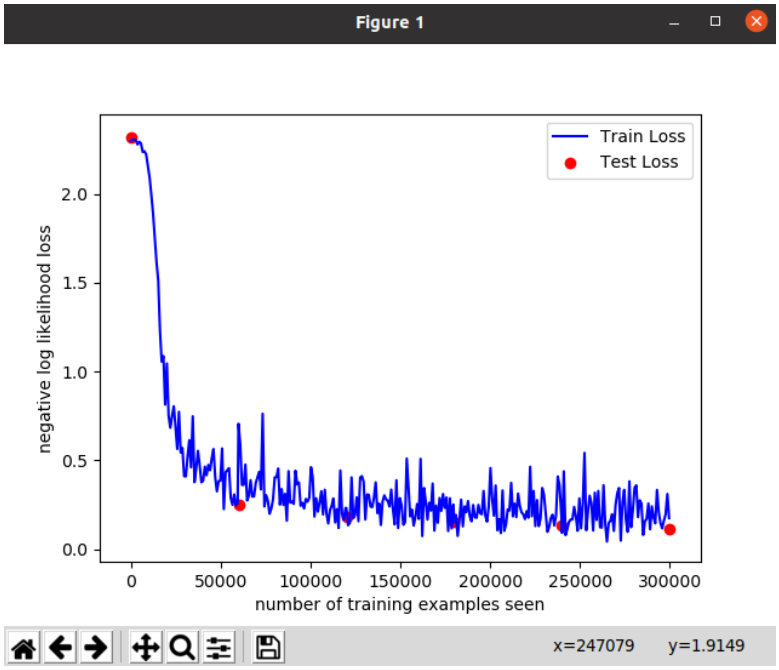
more on the area of the image where the weight values are more when doing the element-wise product of the weights with the pixel values.



C. Replace the first layer of the MNIST network with Gabor filters:

Achieved accuracy: 95.91%

While using the Gabor filter bank as the first conv layer in the MNIST digit recognition network, the accuracy decreased slightly, however, the performance remained good. This could be because the filters in the first conv layer of the MNIST digit recognition network primarily extract edges, which is also a task accomplished by the Gabor filter.




```
Test set: Avg. loss: 0.1254, Accuracy: 9592/10000 (96%)
```

7. Learnings:

1. Using PyTorch
2. Better understanding of CNN architecture and significance of each layer
3. Building and training a network
4. Using a pre-trained network for testing
5. Observed effects of changing different aspects of network
6. Learned how to visualize filters from conv layers

8. Acknowledgment:

- PyTorch: https://pytorch.org/tutorials/beginner/basics/intro.html
- MNIST: https://nextjournal.com/gkoehler/pytorch-mnist
- CNN Architecture: https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243
- ResNet18: https: //pytorch.org/hub/pytorch_vision_ResNet18/