# CS5330_Project5

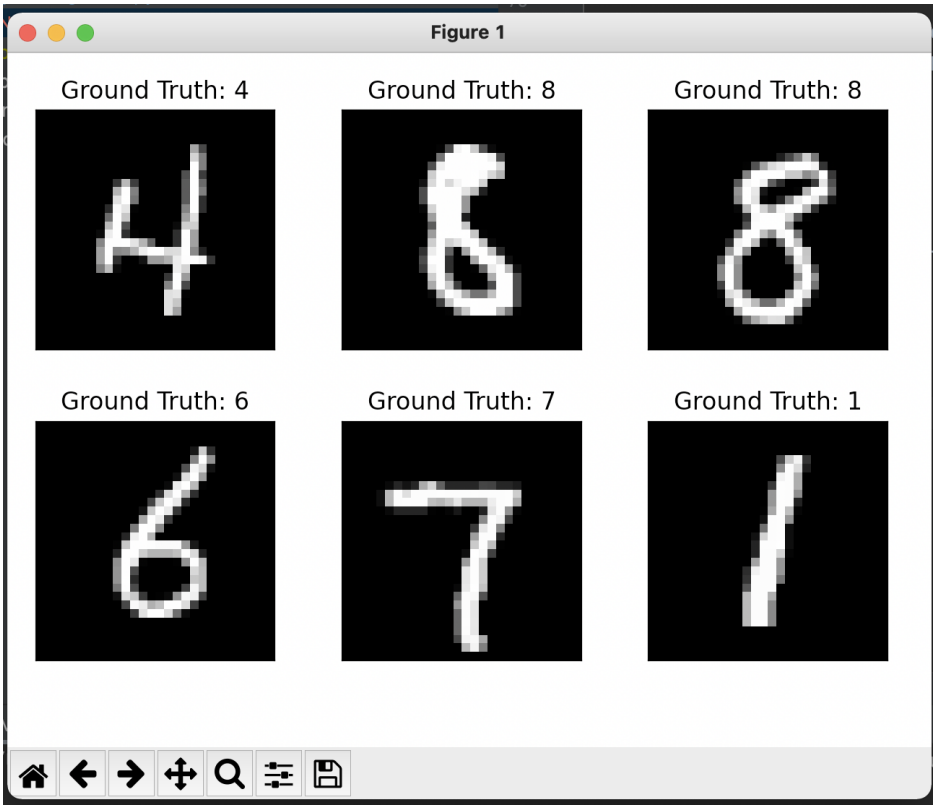
## A short description of the overall project in your own words. (200 words or less)

This project uses the MNIST digit recognition dataset and PyTorch to build, train, analyze, and modify a deep network.

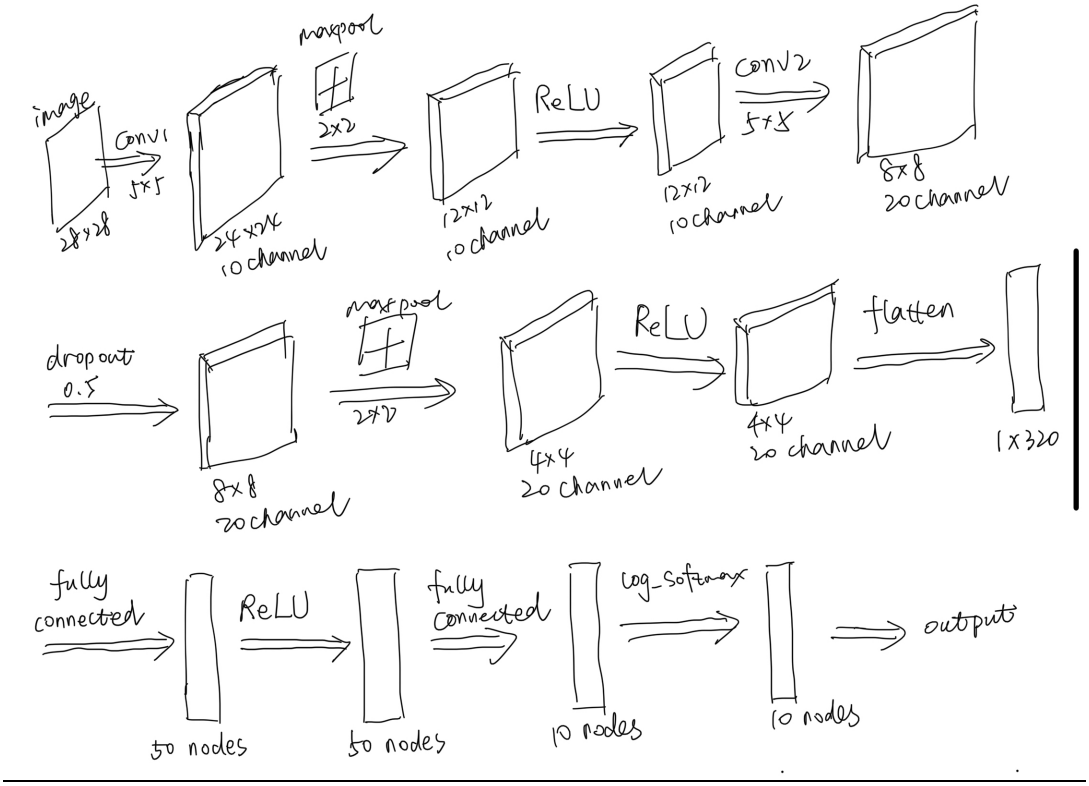
There are four tasks in this project, the first task requires us to read the dataset and build a deep network. It familiarizes us with different layers like the convolution layer, pooling layer, dropout layer, fully connected layer, etc. The second one looks inside the network and sees how the convolution filters process the data. The third task asks us to build a digit embedding space for written symbols. The last task requires us to develop experiments to understand the effect of changing different aspects of the network and analyze the results.

## Any required images along with a short description of the meaning of the image.

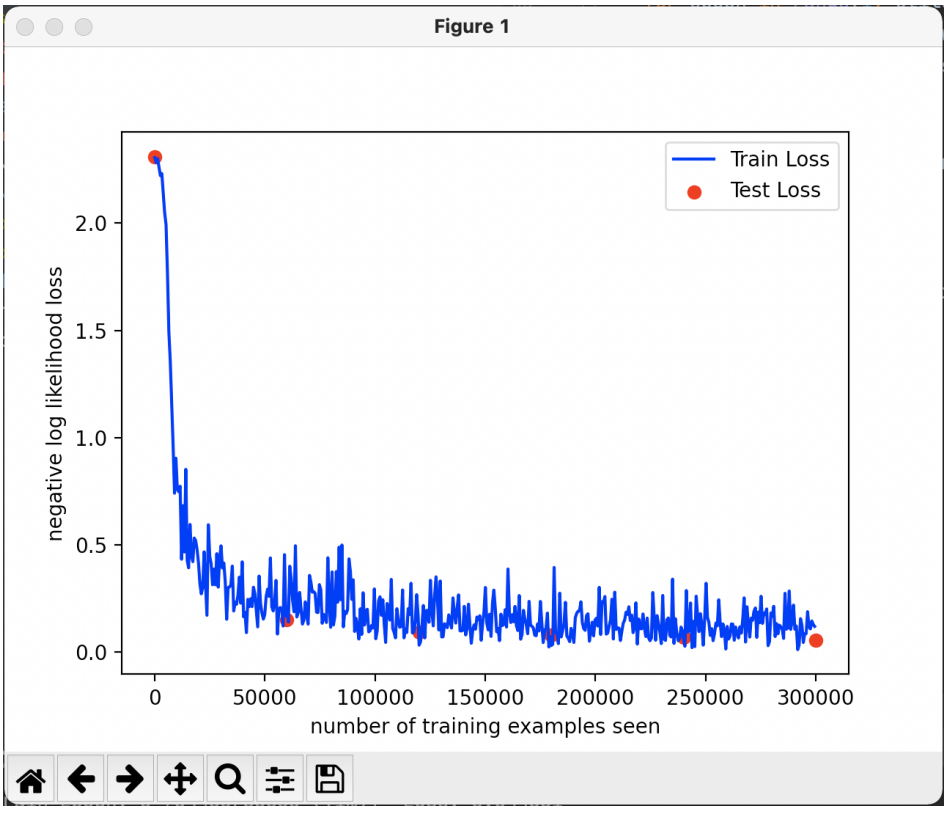
### Task 1A - Plot of the First Six Example Digits




### Task 1C - Diagram of the Network




## Task 1D - Plot of the Training and Testing Accuracy

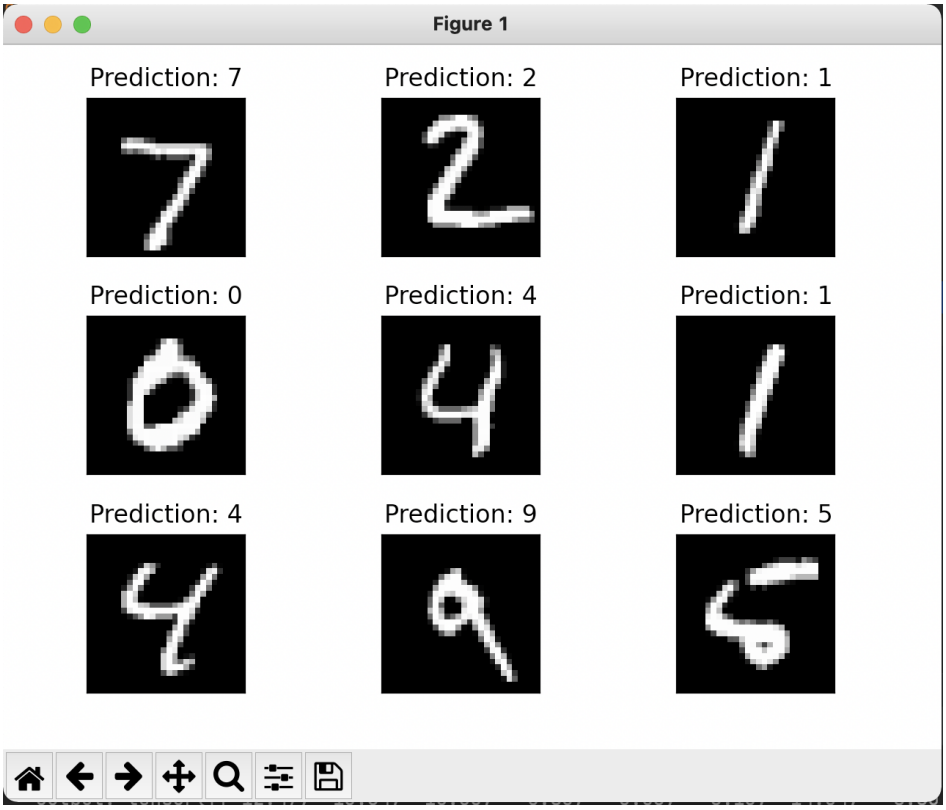



## Task 1F - Read the network and run it on the test set(first 10 as examples)

Print out the 10 output values, the index of the max output value, and the correct label of the digit.

```
/Users/yueyangwu/.local/share/virtualenvs/proj5-JiuvsETS/bin/python /Users/yueyangwu/Desktop/CS5330/hw/proj5/MNISTTest.py
1 - output: tensor([[-1.75e+01, -1.78e+01, -9.44e+00, -1.06e+01, -1.89e+01, -1.88e+01,
         -3.13e+01, -1.12e-04, -1.42e+01, -1.18e+01]])
1 - index of the max output value: 7
1 - prediction label: 7
2 - output: tensor([[-9.43e+00, -6.95e+00, -1.14e-03, -1.15e+01, -1.68e+01, -1.90e+01,
         -1.15e+01, -2.31e+01, -9.46e+00, -2.24e+01]])
2 - index of the max output value: 2
2 - prediction label: 2
3 - output: tensor([[-1.49e+01, -9.63e-04, -8.74e+00, -1.15e+01, -8.72e+00, -1.05e+01,
         -1.07e+01, -7.79e+00, -8.78e+00, -1.22e+01]])
3 - index of the max output value: 1
3 - prediction label: 1
4 - output: tensor([[-2.34e-05, -1.96e+01, -1.22e+01, -1.69e+01, -2.10e+01, -1.35e+01,
         -1.13e+01, -1.80e+01, -1.50e+01, -1.23e+01]])
4 - index of the max output value: 0
4 - prediction label: 0
5 - output: tensor([[-1.33e+01, -2.17e+01, -1.04e+01, -1.43e+01, -1.49e-03, -1.52e+01,
         -1.31e+01, -1.35e+01, -1.24e+01, -6.54e+00]])
5 - index of the max output value: 4
5 - prediction label: 4
6 - output: tensor([[-1.84e+01, -1.61e-04, -1.21e+01, -1.43e+01, -1.05e+01, -1.41e+01,
         -1.55e+01, -9.12e+00, -1.11e+01, -1.45e+01]])
6 - index of the max output value: 1
6 - prediction label: 1
7 - output: tensor([[-2.07e+01, -1.45e+01, -1.60e+01, -1.45e+01, -3.87e-03, -1.19e+01,
         -1.85e+01, -1.07e+01, -5.69e+00, -7.70e+00]])
7 - index of the max output value: 4
7 - prediction label: 4
8 - output: tensor([[-1.96e+01, -1.70e+01, -1.01e+01, -6.50e+00, -6.17e+00, -9.73e+00,
         -2.30e+01, -1.11e+01, -7.80e+00, -4.13e-03]])
8 - index of the max output value: 9
8 - prediction label: 9
9 - output: tensor([[-1.34e+01, -2.29e+01, -1.48e+01, -1.43e+01, -1.58e+01, -6.37e-03,
         -5.36e+00, -1.74e+01, -6.55e+00, -8.39e+00]])
9 - index of the max output value: 5
9 - prediction label: 5
10 - output: tensor([[-1.83e+01, -2.20e+01, -1.62e+01, -1.27e+01, -9.76e+00, -1.44e+01,
         -2.64e+01, -6.30e+00, -6.78e+00, -3.04e-03]])
10 - index of the max output value: 9
10 - prediction label: 9
```

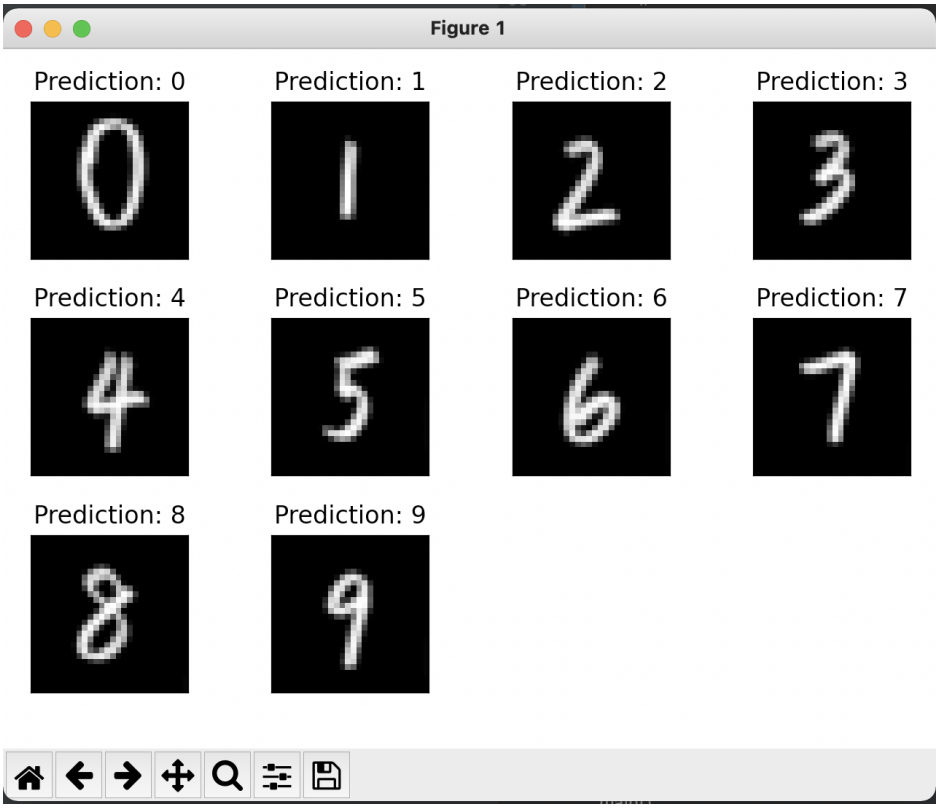
A plot of the prediction of the first 9 digits in the test set




## Task 1G - Test on Custom Digit Images

The custom digits are black digits written on white background. Therefore, when loading the image, I resized it to 28x28, turned it into a single-channel image, and inverted the foreground and background.

It turns out that the model works well on the custom digits.




## Task 2 - Exam the Network

Print the model

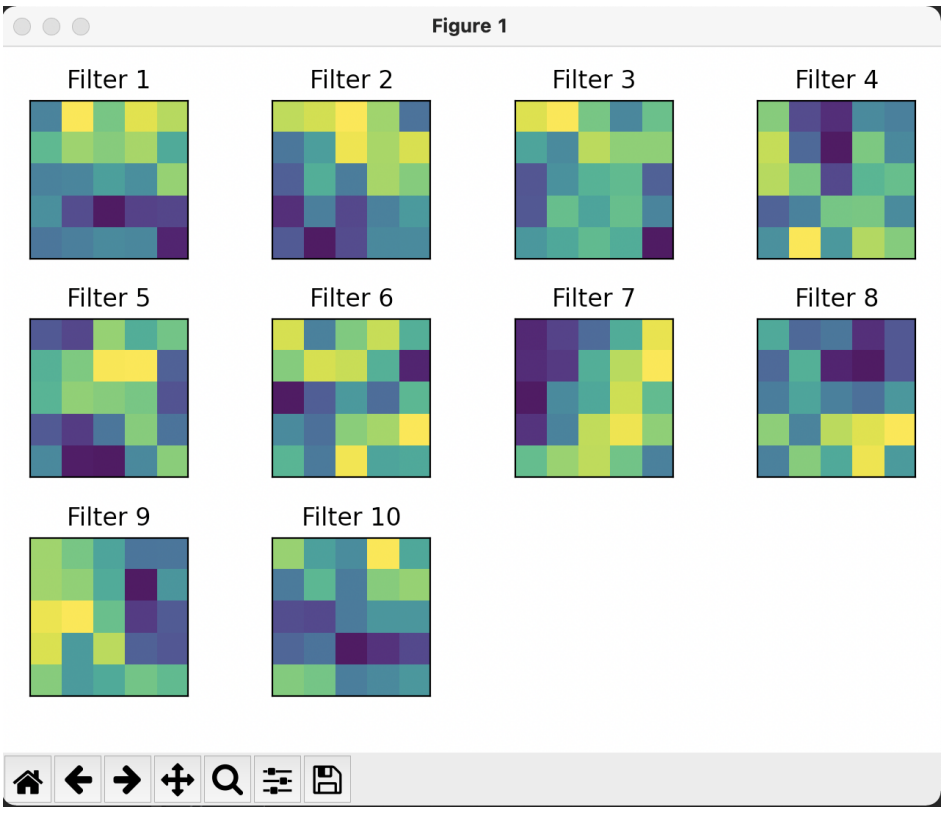
```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)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

## Task 2A - Analyze the First Layer

Print the filter weights and their shape

```
filter 1
tensor([[-0.0764,  0.2618,  0.1102,  0.2314,  0.1849],
        [ 0.0705,  0.1564,  0.1289,  0.1681,  0.0284],
        [-0.0814, -0.0727, -0.0019, -0.0460,  0.1484],
        [-0.0455, -0.1954, -0.2811, -0.2152, -0.2078],
        [-0.1089, -0.0880, -0.0579, -0.0689, -0.2600]], requires_grad=True)
torch.Size([5, 5])


filter 2
tensor([[ 0.2772,  0.3053,  0.3684,  0.2319, -0.1345],
        [-0.1219,  0.0117,  0.3462,  0.2456,  0.3145],
        [-0.1921,  0.0713, -0.1051,  0.2414,  0.1882],
        [-0.3062, -0.0955, -0.2512, -0.0905, -0.0043],
        [-0.2072, -0.3545, -0.2437, -0.0640, -0.0554]], requires_grad=True)
torch.Size([5, 5])


filter 3
tensor([[ 3.3132e-01,  3.7428e-01,  1.8578e-01, -3.3814e-02,  1.6370e-01],
        [ 6.7877e-02, -2.2120e-02,  2.8327e-01,  2.2381e-01,  2.2281e-01],
        [-1.6959e-01,  1.4662e-04,  1.1245e-01,  1.4105e-01, -1.4063e-01],
        [-1.6372e-01,  1.5422e-01,  5.9930e-02,  1.5438e-01, -4.0222e-02],
        [ 2.1901e-02,  7.7581e-02,  1.4302e-01,  9.1226e-02, -2.9666e-01]],
       requires_grad=True)
torch.Size([5, 5])


filter 4
tensor([[ 0.1476, -0.1324, -0.1742, -0.0141, -0.0360],
        [ 0.2872, -0.0827, -0.2040,  0.1367, -0.0171],
        [ 0.1952,  0.1334, -0.1363,  0.0872,  0.1099],
        [-0.0917, -0.0322,  0.1294,  0.1335, -0.0103],
        [ 0.0012,  0.2616,  0.0181,  0.1922,  0.1469]], requires_grad=True)
torch.Size([5, 5])


filter 5
tensor([[-0.1324, -0.1710,  0.2351,  0.1068,  0.1833],
        [ 0.1193,  0.2008,  0.3620,  0.3661, -0.1132],
        [ 0.1272,  0.2294,  0.2134,  0.1920, -0.1459],
        [-0.1296, -0.1947, -0.0589,  0.2146, -0.0662],
        [-0.0072, -0.2487, -0.2583, -0.0090,  0.2190]], requires_grad=True)
torch.Size([5, 5])


filter 6
tensor([[ 0.2332, -0.0327,  0.1484,  0.2184,  0.0761],
        [ 0.1529,  0.2330,  0.2212,  0.0813, -0.1884],
        [-0.2067, -0.1037,  0.0247, -0.0725,  0.0970],
        [-0.0114, -0.0675,  0.1586,  0.1872,  0.2723],
        [ 0.0921, -0.0470,  0.2621,  0.0512,  0.0627]], requires_grad=True)
torch.Size([5, 5])


filter 7
tensor([[-0.3137, -0.2586, -0.1561,  0.0634,  0.3349],
        [-0.3048, -0.2765,  0.0698,  0.2648,  0.3642],
        [-0.3484, -0.0566,  0.0480,  0.2971,  0.1197],
        [-0.2917, -0.0814,  0.2762,  0.3445,  0.2044],
        [ 0.1263,  0.2210,  0.2731,  0.1528, -0.0875]], requires_grad=True)
torch.Size([5, 5])


filter 8
tensor([[ 0.0846, -0.1254, -0.0841, -0.2595, -0.1689],
        [-0.1224,  0.1060, -0.2793, -0.3062, -0.1734],
        [-0.0628,  0.0692, -0.0556, -0.1032,  0.0187],
        [ 0.2290, -0.0441,  0.2928,  0.3439,  0.3868],
        [-0.0542,  0.2171,  0.0859,  0.3676,  0.0338]], requires_grad=True)
torch.Size([5, 5])


filter 9
tensor([[ 0.2030,  0.1399,  0.0097, -0.1461, -0.1455],
        [ 0.2060,  0.1819,  0.0337, -0.3695, -0.0435],
        [ 0.3152,  0.3373,  0.1127, -0.2965, -0.2220],
        [ 0.2858, -0.0222,  0.2427, -0.2050, -0.2317],
        [ 0.1651, -0.0220,  0.0324,  0.1311,  0.0935]], requires_grad=True)
torch.Size([5, 5])


filter 10
tensor([[ 0.1867,  0.0111, -0.0511,  0.3167,  0.0357],
        [-0.1016,  0.0868, -0.0992,  0.1626,  0.1856],
        [-0.2122, -0.2242, -0.1020, -0.0212, -0.0194],
        [-0.1583, -0.1225, -0.3158, -0.2688, -0.2243],
        [ 0.1573,  0.1353, -0.0914, -0.0602, -0.0184]], requires_grad=True)
torch.Size([5, 5])
```

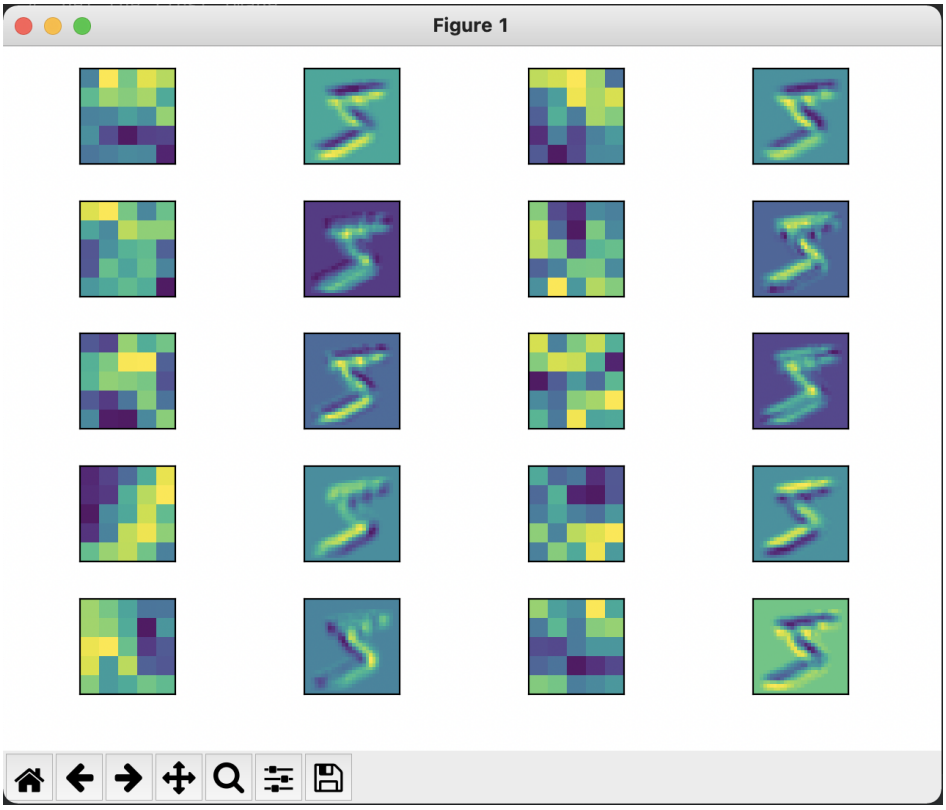
Visualize the 10 filters




## Task 2B - Effect of the Filters

The first image filtered by the ten filters.

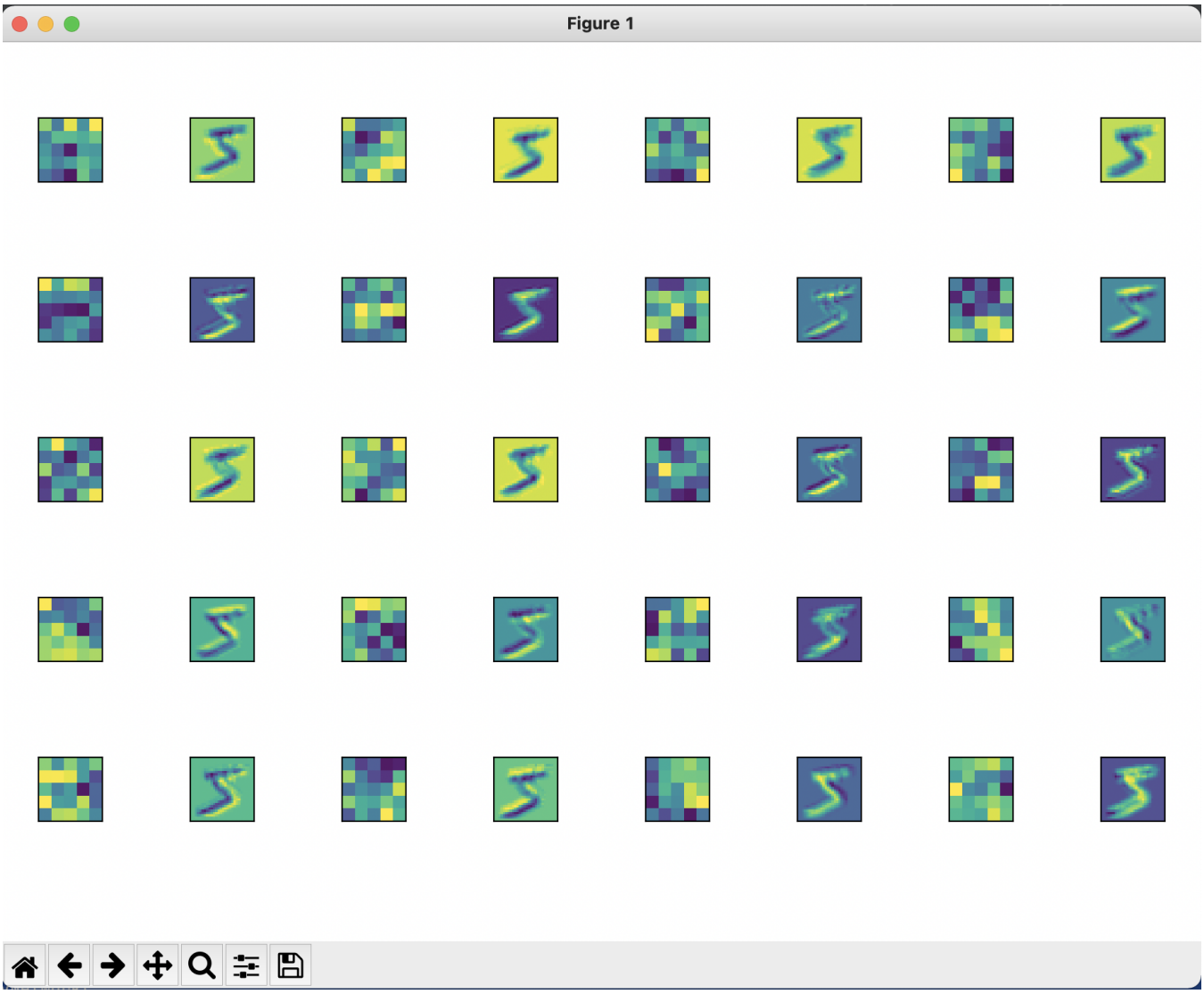
The results make sense since the filters detect edges of different orientations. For example, in the first filter in the first row, the top side is brighter and the bottom side is darker, and it highlights the horizontal edges. Also, in the forth filter in the first column, the upper left is darker and the lower right is brighter, and it highlights the slope on the left side of the digit.




## Task 2C - Build a Truncated Model

The first image filtered by the 20 filters

The second layer filters are harder to interpret but we can still see that a lot of them make sense since they can detect different edges of the digit. For example, in the 7th filter in the second row, the upper side is darker and the lower side is brighter, and it detects the horizontal edges. Also, in the last filter in the first column, the left part is darker and only a small part of the right is brighter, so it detects the edges to the right of the digits.




## Task 3B - Truncated Model Gives 50 Output

The system prints out the shape of the output of applying the truncated model to the first training image

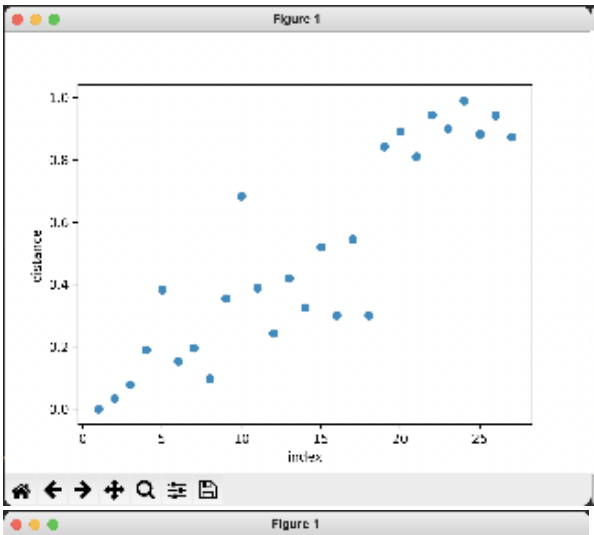
```
torch.Size([1, 50])
```

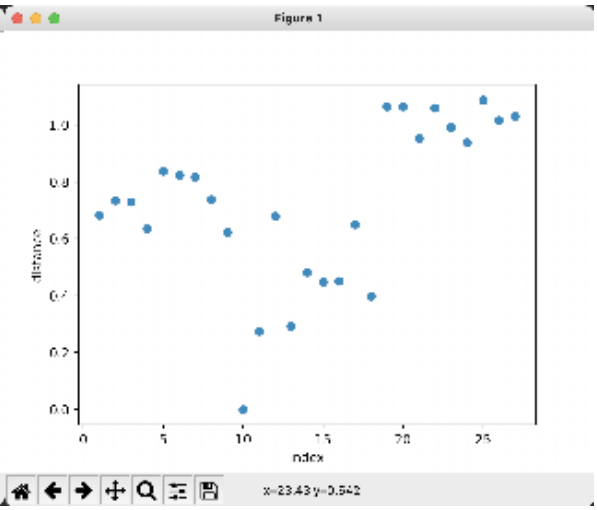
## Task 3D - Compute Distances in the Embedding Space

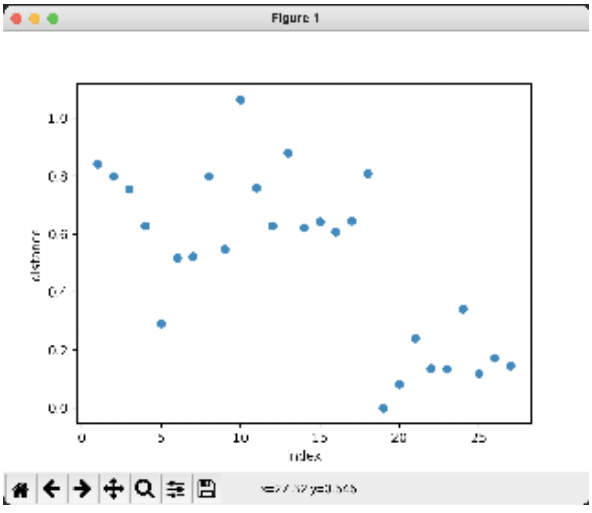
All the 27 SSD values are printed, the order is alpha, beta, and gamma. For each symbol, I choose the first one as an example. So the expected result is that the distances between the example and the next 8 symbols should be smaller than the distances between the example and the other symbols. From the printed results, we can see that the expectation is met. Therefore, the KNN classifier should work well for this task.

[0.0, 0.03348553688800136, 0.0783387505594457, 0.18929814061308348, 0.3820393156155557, 0.1533369727391809, 0.19524672085213549, 0.0969682468586565, 0.3550970748421516, 0.6836820768696216, 0.3882468335085891, 0.24218215023352382, 0.4199797553254959, 0.325673448124198, 0.5197041860581597, 0.30044423399149056, 0.5451795211719657, 0.29956704248674276, 0.8424081948928387, 0.891336293216519, 0.8105074372565739, 0.9440952344296016, 0.8991809288266441, 0.9900789177447962, 0.8818938922359085, 0.9430278800350695, 0.8734271507859585]
[0.6836820768696216, 0.7344461128468837, 0.7301659150888352, 0.6360688119251674, 0.8380202343607789, 0.8230722231552126, 0.8175284768643873, 0.737682112084363, 0.6234130355760843, 0.0, 0.2753147281567294, 0.679284743614764, 0.293200486581945, 0.4816058297244124, 0.4476004547758336, 0.45094216700888784, 0.6506968606610095, 0.39721395034388013, 1.0644637075171781, 1.0650538350898768, 0.9532995431640126, 1.0596620964616335, 0.9917726715029211, 0.9393135215585048, 1.0879900457849907, 1.0167373542903824, 1.0317102020457336]
[0.8424081948928387, 0.7995850214788216, 0.7560527757508541, 0.6290863417555917, 0.29137963775588577, 0.5179487327419613, 0.522781051192599, 0.799878367596282, 0.5479002643725848, 1.0644637075171781, 0.76072182074345, 0.6284909588453367, 0.8811247848928956, 0.6223828124654004, 0.6425082480689625, 0.6085349333144892, 0.6467026266293288, 0.8088472162602165, 0.0, 0.08256612860711016, 0.2402165337196765, 0.13625443797430092, 0.1348601919999845, 0.34220148405045725, 0.11806458618310806, 0.17178482664123212, 0.14543151242325913]

The plots help visualize the results, the results form three clusters.

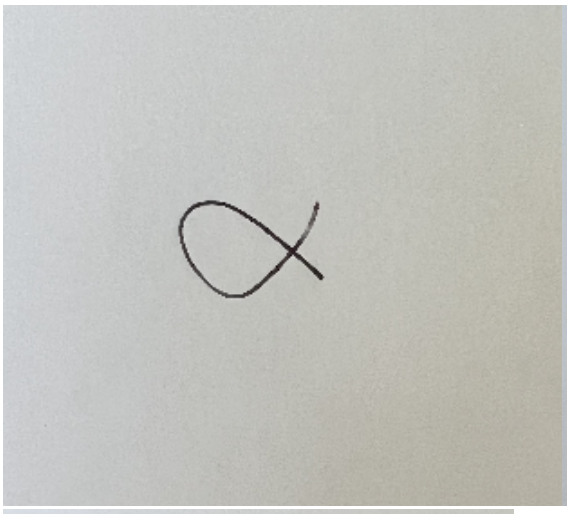


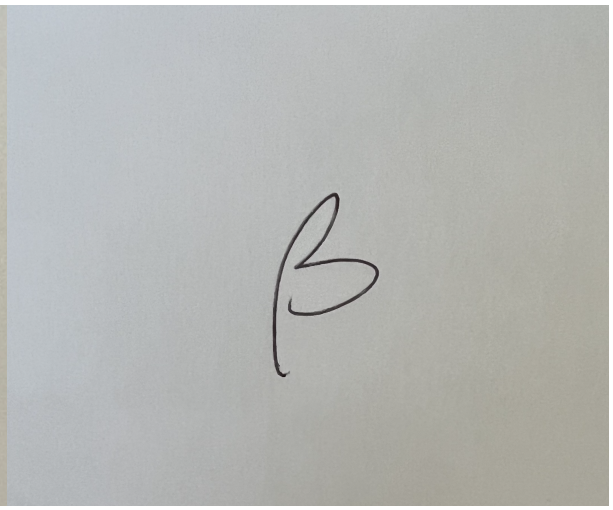



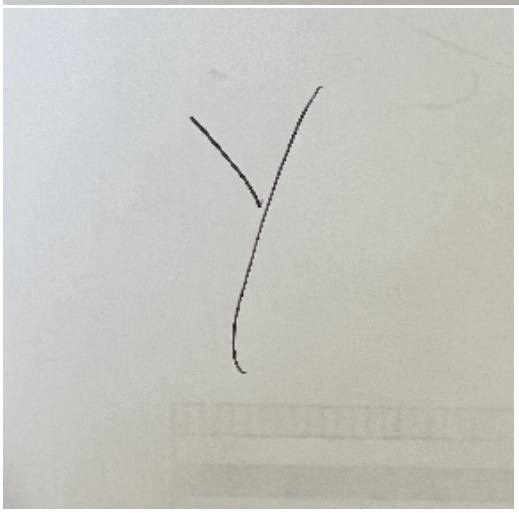




## Task 3E - Custom Greek Symbol

This task uses  the following three examples, and the SSDs are calculated



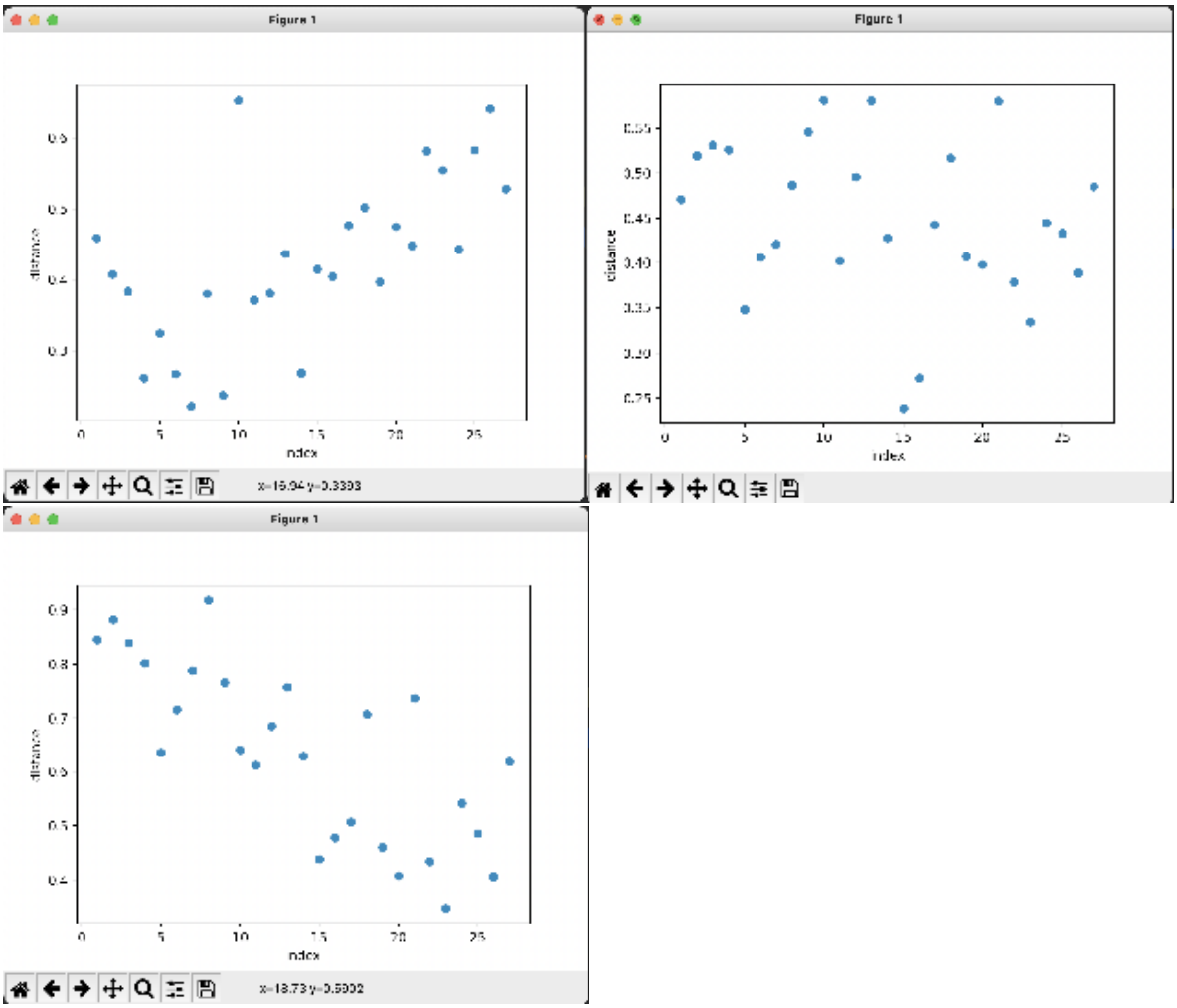




Below are the 27 SSD values for the three example symbols

[0.45918934591449684, 0.407524718843259, 0.38371889250059726, 0.26161653147014235, 0.3247085028111106, 0.26757795208813207, 0.2223092955937834, 0.3799507117287692, 0.23768056913841074, 0.6523857975938373, 0.3713818984187077, 0.38113845571453936, 0.43647532104558706, 0.268977494911628, 0.4149414618718084, 0.40439563249270977, 0.47660575095680346, 0.5020198279336407, 0.3966908280690232, 0.47517428082664, 0.44805219930379697, 0.5811789487783614, 0.5541080287282221, 0.44306141010848477, 0.5822757293764906, 0.6409434360580366, 0.528348453532259]
[0.47072898980339406, 0.5194207203697712, 0.5312500647438101, 0.5261258777877174, 0.3484611303784563, 0.4065166932751326, 0.4210263126327717, 0.4866436036159067, 0.5460744093087622, 0.5810730517454119, 0.40233507826433457, 0.4958768508246674, 0.5806547768002326, 0.4279920142755861, 0.23881550836961146, 0.27244597313020286, 0.4431279877695474, 0.5168928116260336, 0.40713518073725125, 0.3983556916522924, 0.5798400634429581, 0.379010181452433, 0.3345141316611195, 0.4450730618046031, 0.4331386255105665, 0.38912158079522885, 0.4853370515533164]
[0.844916905074211, 0.8822016576521271, 0.8390652152296937, 0.8017704512155397, 0.6363615886145377, 0.7156146004866802, 0.7876991748753485, 0.9180514379636213, 0.7660909592785584, 0.6412071135808813, 0.6124138878078791, 0.6853396716386684, 0.7572088035174591, 0.628992024246501, 0.4379599043596358, 0.4785803087952207, 0.5071812916681926, 0.7077519918768731, 0.4603501213127487, 0.4075460987179099, 0.7373281169180207, 0.43401216290212324, 0.34763791986611897, 0.5414396095538182, 0.48605707644176327, 0.40586611601476186, 0.6189526328598064]

Plot the values, and we can see that, similarly to the result of task 3D, the custom symbols match the corresponding symbols in the embedding space.




## Task 4 - Experiment with the Deep Network

The experiment evaluates 72 networks from 5 dimensions. To be more specific:

epoch sizes: 3, 5

training batch sizes: 64, 128

the number of convolution layers: add an additional 1 - 3 convolution layers

convolution layer filter size: 3, 5, 7

dropout rate: 0.3, 0.5

### My hypotheses:

When the epoch size increases, the accuracy should also be increased because the weight will be trained more times.

When the training batch size doubles, the accuracy should also be increased because there are more training data provided.

When the number of convolution layers increases, the accuracy should also be increased since more features are analyzed.

A mid-size convolution layer filter should work the best, larger or smaller filters will both decrease the accuracy.

A proper dropout rate should be chosen to reduce overfitting. both 0.3 and 0.5 should be acceptable for the MNIST dataset.

However, for a small dataset like MNIST, the difference may not be very obvious.

### The Evaluation Results:

Generally speaking, the evaluation results support the hypotheses. However, since the MNIST dataset is a small one, the difference is not very obvious.

Here are the complete results in txt format: experiment_results.txt

# A description and example images of any extensions.

## Extension 1 - Evaluate More Dimensions in Task 4

5 dimensions are explored instead of 3. For details, see Task 4.

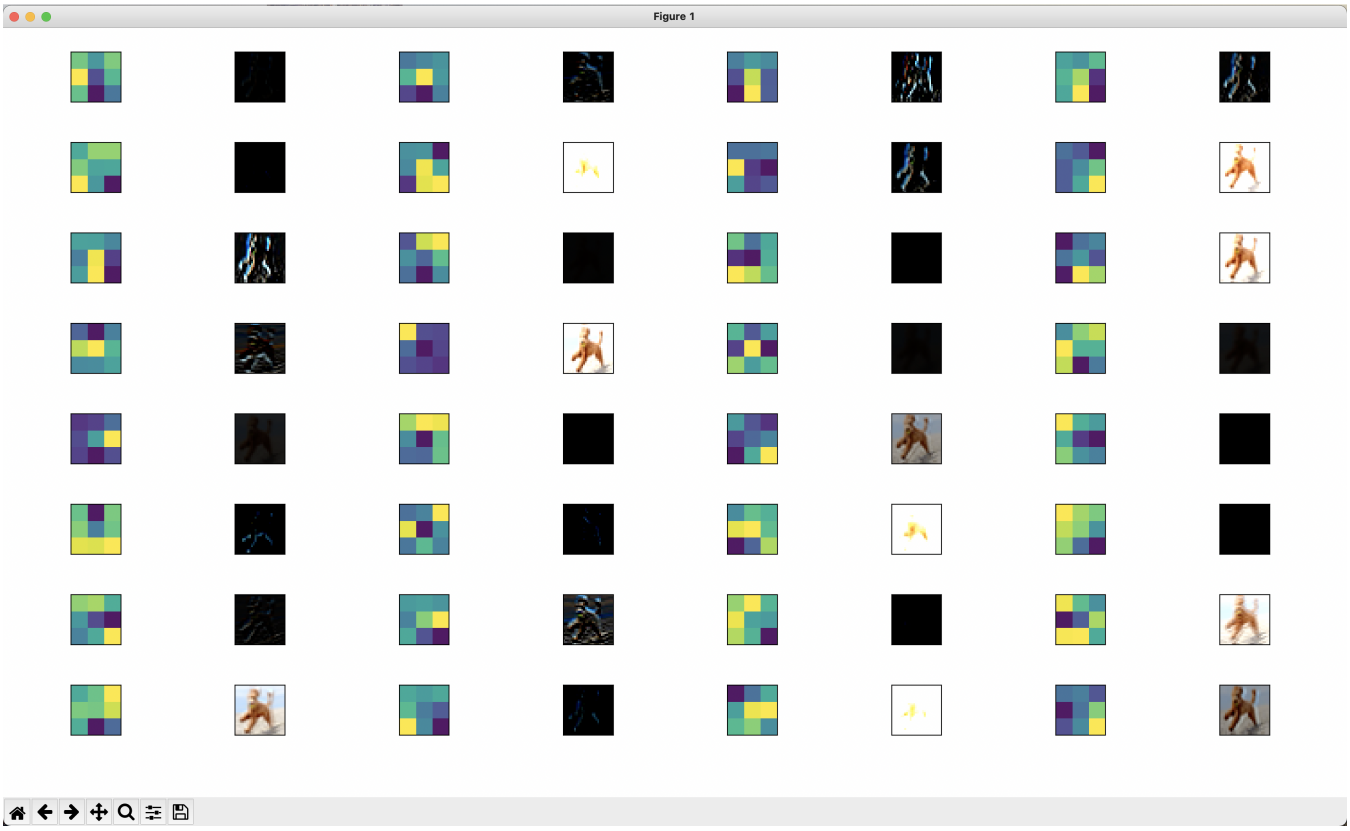
## Extension 2 - Load a pre-trained network from PyTorch and evaluate the first layer.

Mobilenet is loaded for this task, and the first layer is evaluated.

The first layer has 32 3x3 filters.

```
Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
```

Apply the 32 filters to an image of a dog, and get the following results. Some of the filters make sense since they can detect the edges in different orientations(similar to SobelX and SobelY). Other filters are similar to Gaussian blur and blur the image.



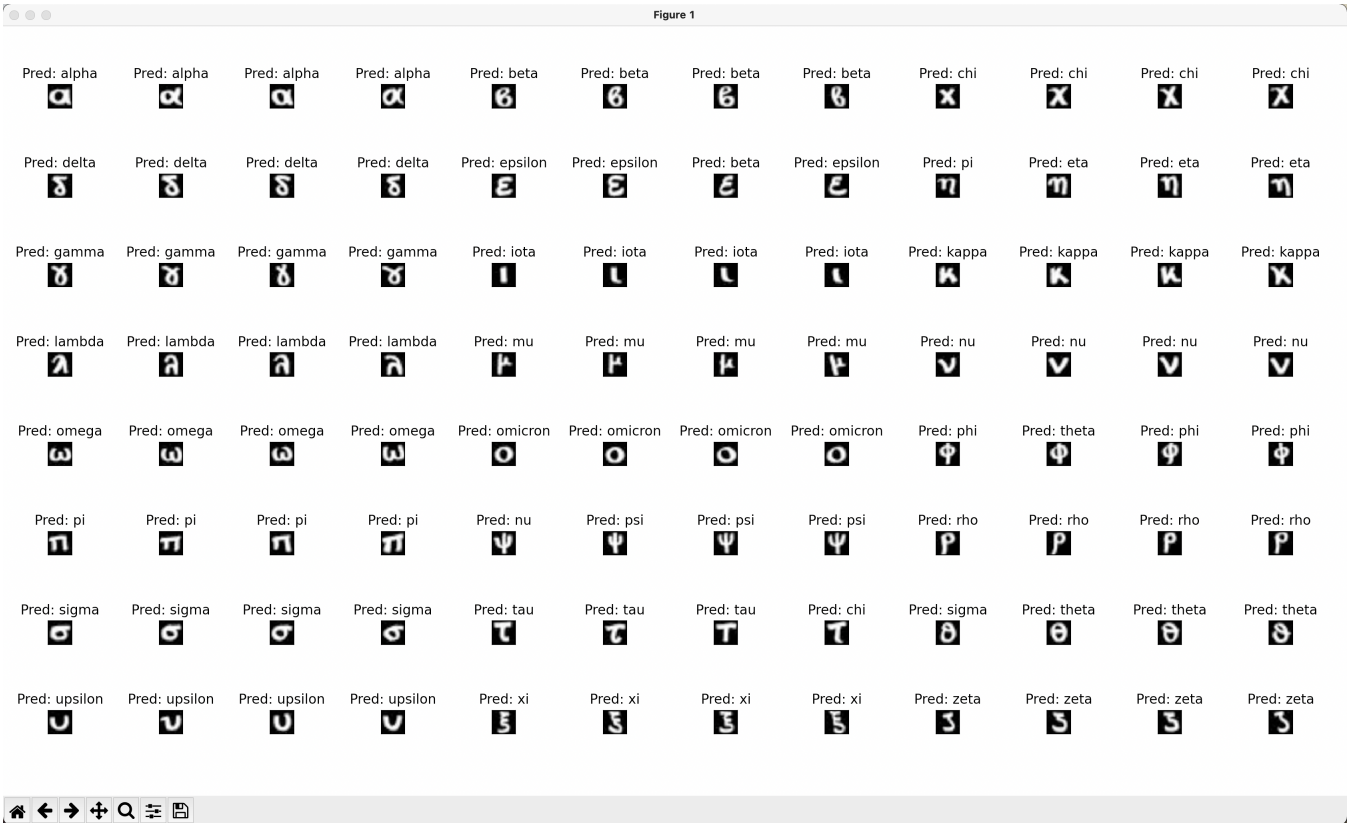
## Extension 3 - More Greek Letters and KNN Classifier

This extension builds a KNN classifier to recognize all the 24 greek letters.

For a full dataset of Greek letters, I use the dataset from Kaggle. The training dataset consists of 240 images of Greek letters(10 for each letter), and the test dataset consist of 96 images(4 for each letter)

https://www.kaggle.com/datasets/katianakontolati/classification-of-handwritten-greek-letters

Below is the plot of the prediction value for all the 96 testing data when K=5 in KNN. The overall accuracy is 94.79%(91/96).




# A short reflection of what you learned.

First of all, I learned how to use PyTorch. Also, this is my first time working with a deep network and I learned a lot from this project. First, I learned how to build and train a deep learning network. By building truncated models and looking into different convolution layers, I get a better understanding of how the filters and layers process the image. By modifying the parameters in the network, I learned the effect of changing different aspects of the network.

# Acknowledgment of any materials or people you consulted for the assignment.


- PyTorch Tutorial https://pytorch.org/tutorials/beginner/basics/intro.html
- PyTorch Documentation https://pytorch.org/docs/stable/index.html
- MNIST Recognition Tutorial https://nextjournal.com/gkoehler/pytorch-mnist
- Intro to CNNs https://www.tomasbeuzen.com/deep-learning-with-pytorch/chapters/chapter5_cnns-pt1.html
- Matplotlib Documentation https://matplotlib.org/3.5.0/index.html
- Write to CVS with Python https://www.pythontutorial.net/python-basics/python-write-csv-file/
- Mobilenet network https://pytorch.org/hub/pytorch_vision_mobilenet_v2/