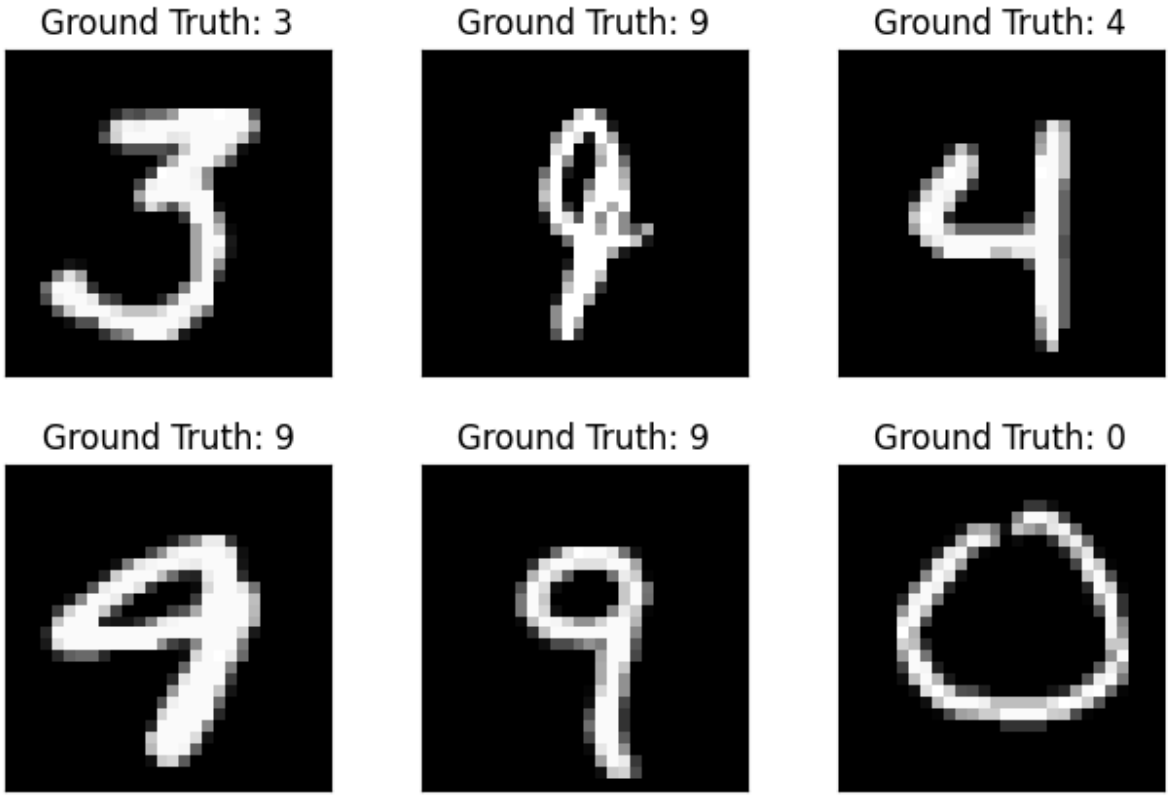
CS5330
Project 5
Instructor: Bruce A. Maxwell
Contributors: Matej Zecic

Description:

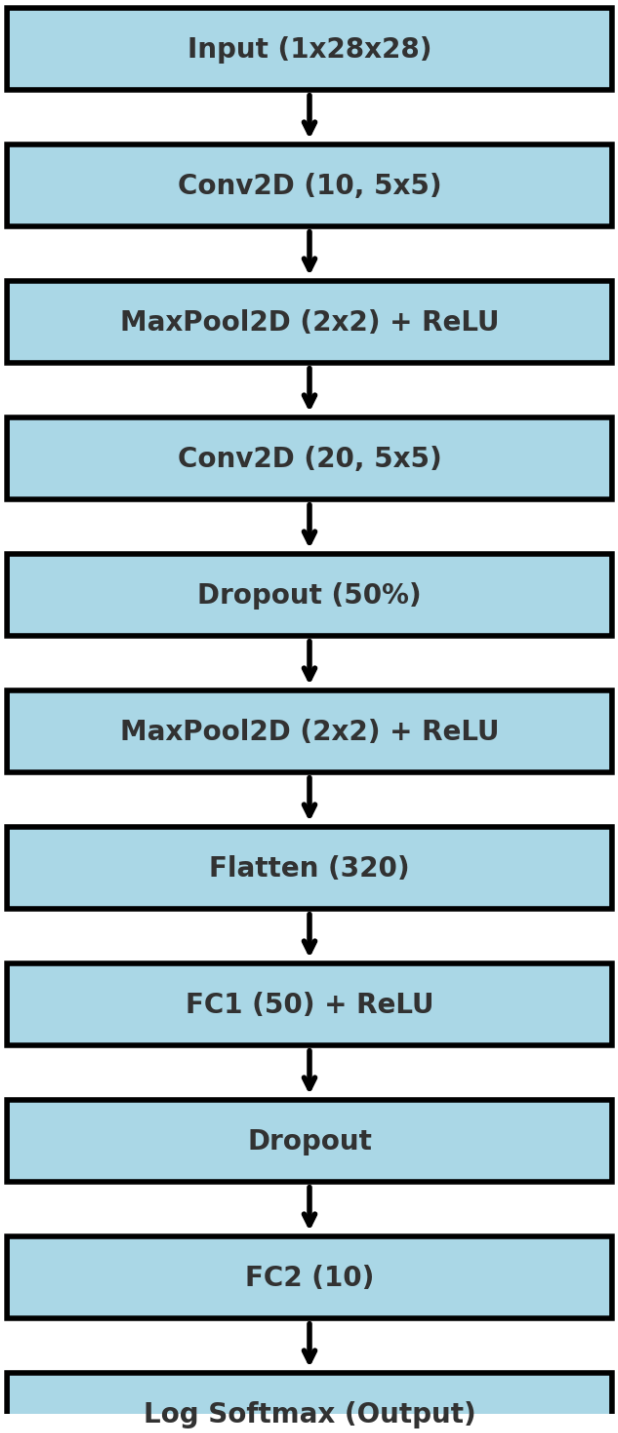
This project is about training a neural network on the MNIST digits dataset. It's about learning how to load, train, analyze the data on a model for a recognition task.

1. Build a network to recognize digits.
    A. For this task, I'm using matplotlib to plot the first six example digits in the dataset.



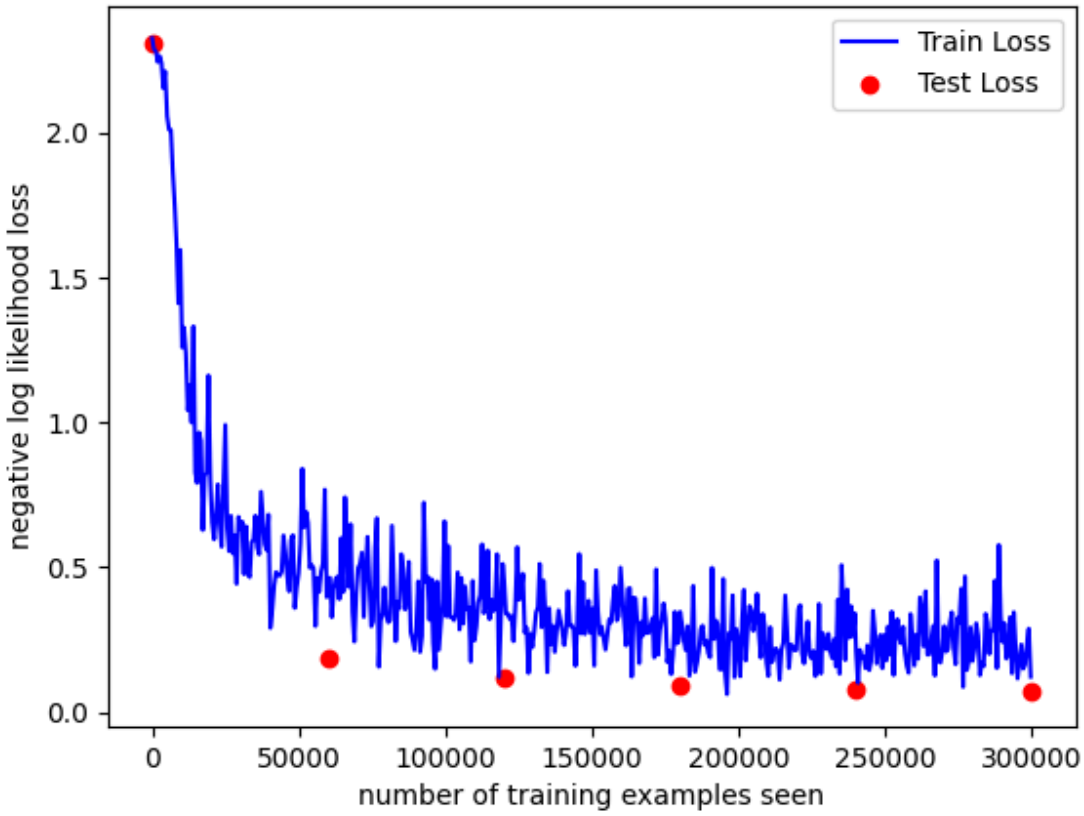
    B. Based on the tutorial and the class constructed for this task given the problem requirements, here is my network diagram:

Forward Pass Through the Neural Network

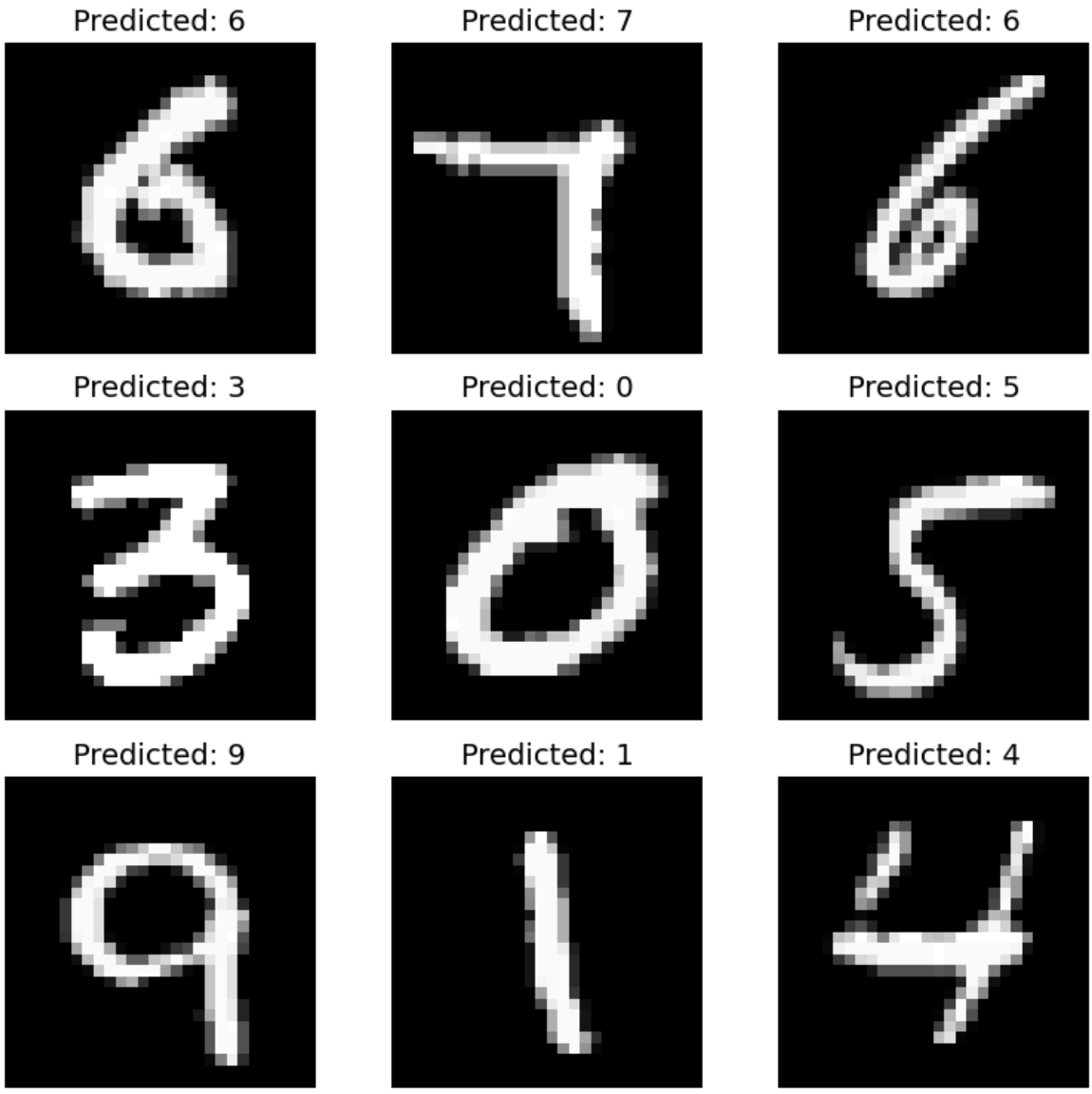


C. For this task I'm plotting the results after 5 epochs of training. After the first epoch, we see that the loss significantly diminishes. It continues to diminish and converges in about 2, 3 epochs in, with little results after the third epoch.

Results:



E. In this task, I'm creating a new file model_test.py, initializing previously saved model weights and setting up the network in eval mode. Then I'm printing results after running the network on the first 10 examples of the test set. Here are my results:


Predictions for First 9 Test Examples
Predicted: 6
Predicted: 7
Predicted: 6
Predicted: 3
Predicted: 0
Predicted: 5
Predicted: 9
Predicted: 1
Predicted: 4

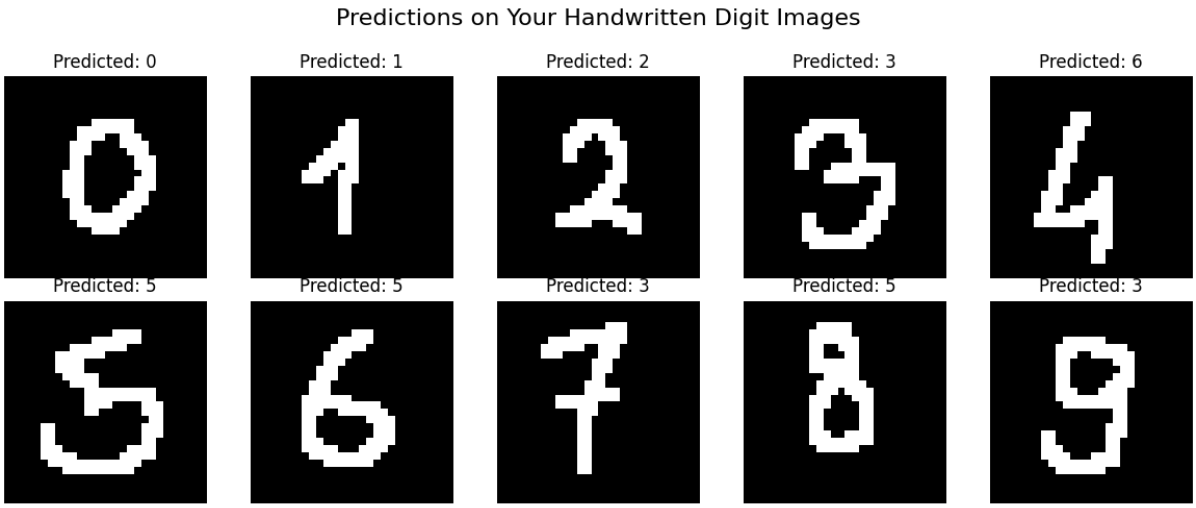
```
Example  Output Values (rounded)                                                    Predicted  True Label
       ['-8.05', '-15.83', '-13.96', '-11.83', '-16.43', '-2.33', '-0.10', '-24.20', '-9.26', '-15.44'] 6
   6
       ['-17.14', '-12.81', '-9.97', '-8.02', '-15.20', '-15.98', '-26.09', '-0.00', '-11.87', '-9.52'] 7
   7
       ['-8.33', '-11.84', '-10.21', '-13.20', '-11.77', '-5.37', '-0.01', '-20.66', '-8.27', '-16.37'] 6
   6
       ['-29.62', '-19.76', '-17.82', '0.00', '-30.55', '-17.11', '-33.07', '-22.20', '-17.97', '-18.73'] 3
     3
       ['-0.00', '-21.44', '-12.03', '-17.31', '-23.37', '-17.82', '-12.81', '-20.25', '-15.86', '-16.68'] 0
    0
       ['-18.08', '-30.62', '-30.24', '-11.98', '-27.99', '-0.00', '-21.23', '-30.69', '-15.35', '-14.99'] 5
    5
       ['-12.87', '-20.79', '-11.96', '-8.28', '-6.43', '-10.20', '-17.83', '-7.07', '-10.38', '-0.00'] 9
   9
       ['-19.49', '-0.00', '-10.68', '-12.56', '-11.17', '-14.34', '-15.45', '-10.57', '-9.24', '-14.31'] 1
     1
       ['-23.96', '-17.47', '-13.98', '-16.21', '-0.00', '-14.31', '-15.28', '-17.55', '-17.40', '-11.54'] 4
    4
       ['-10.49', '-11.33', '-11.07', '-9.43', '-15.14', '-2.73', '-5.93', '-19.74', '-0.07', '-12.23'] 8
   8
(venvmateizecic@Mateis-MacBook-Pro cs5330-Project-5 %
```

In my previous tasks with training, I get around 98% accuracy. This is reflected in results here with all of the digits predicted correctly.

F.

In this task I drew my own digits 0-9 in ms paint. I resized the canvas to 28x28 and drew the digits with black ink in the middle of the screen. I further made sure they are resized in python and converted the colors so it matches the mnist database.

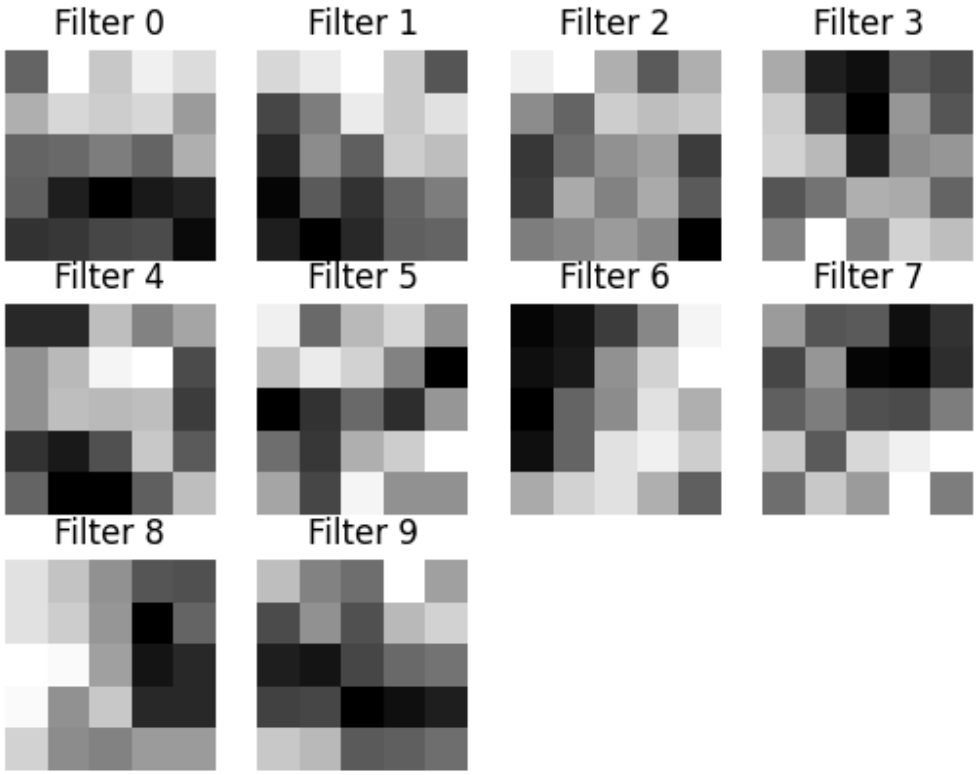
Here are my results:



Total 50% accuracy. It performs significantly worse than the original testing data from mnist. This is interesting as I don't see a significant difference than the numbers found in mnist. I think this may mean that my model, although with 98% accuracy for mnsit testing data, is overfitted, where the numbers are slightly more exaggerated and uneven in some cases.

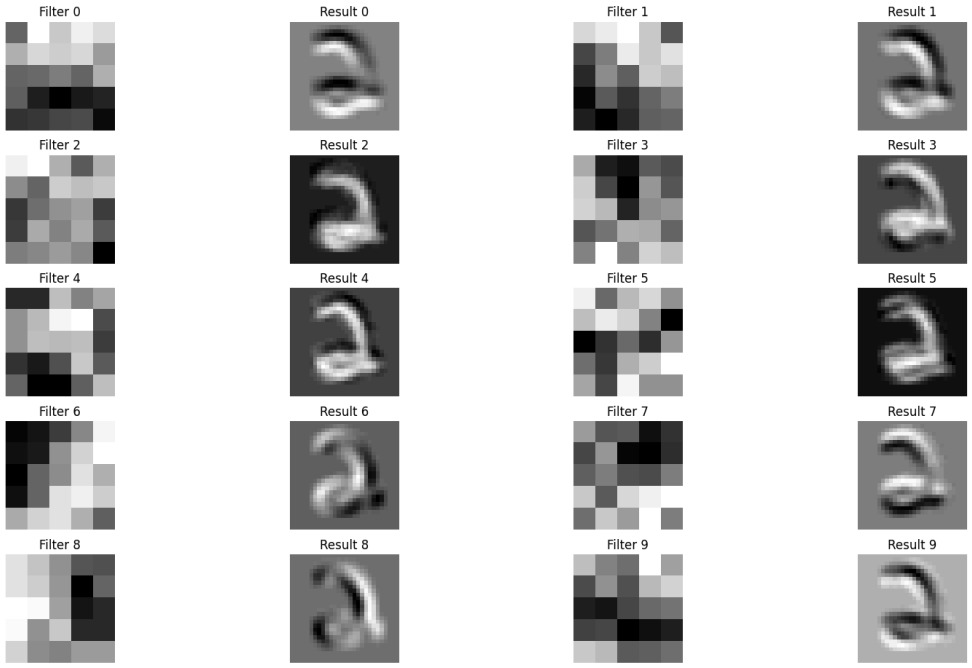
2. Examine your network

    A. In this task, I'm loading the previously used model, accessing weights from the first conv1 layer and printing the first 10.



Some of these filters look roughly like sobel y, for example filter 0, or sobel x, for example filter 6.

    B. In this task, I'm taking the first number in the training set and showing the effects of the filters by plotting the filter and the result of the filter next to it for the first 10 in conv1 layer.

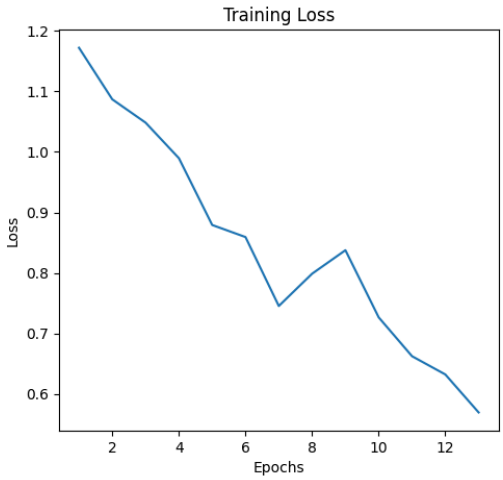


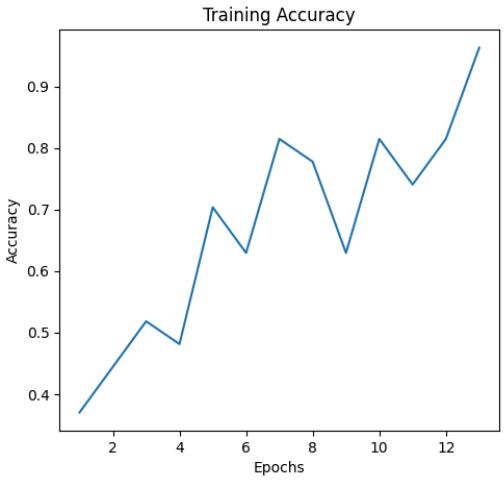
I think that these filters attempt to extract the edges in a particular angle direction from the image to get the idea of what the shape of the number is. For example, the first filter gets a sense of the upper boundary of the number in the image by passing in a 5x5 kernel that has higher values in the top left and right quadrants.
As we continue onto the second filter, the same principle applies, but this time the higher values are mostly in the top right quadrant and then some in top left and some bottom right. As we see in the result of the second filter, this is reflected by showing the top boundaries of an image at a 45% angle.
I find filter 2 and 5 interesting, as they show results that differ from others. Filter 2, or the third filter, might attempt to capture the overall width of the number in the image, or how much space the number takes up width wise, by having lower values in the 5x5 kernel at left and right edge, and similar values in the center.

3. Transfer learning on Greek Letters

In this task I'm reusing the trained network from task 1, loading the weights, freezing the gradient, and changing the last layer with a linear layer with 3 nodes. Then I'm retraining that layer until I achieve good accuracy for identifying Greek letters alpha, beta, and gamma.

Here are my results:

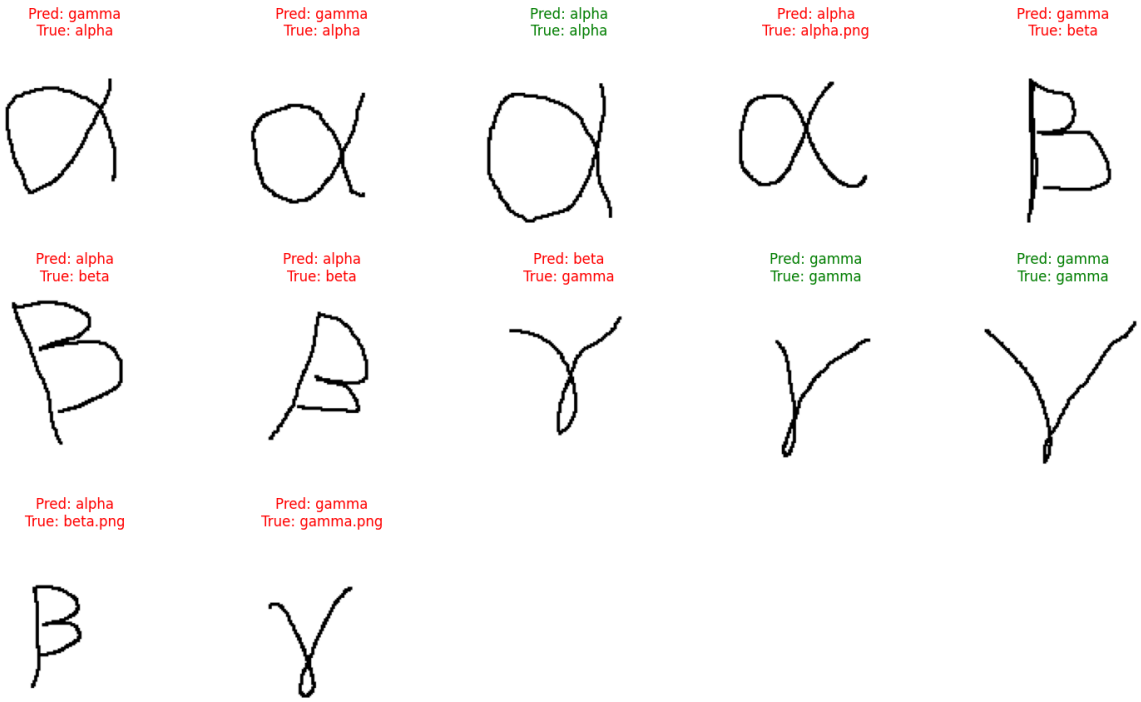





```
Starting training for Greek letters classification...
Epoch 1/20: Loss: 1.1722, Accuracy: 37.04%
Epoch 2/20: Loss: 1.0869, Accuracy: 44.44%
Epoch 3/20: Loss: 1.0484, Accuracy: 51.85%
Epoch 4/20: Loss: 0.9896, Accuracy: 48.15%
Epoch 5/20: Loss: 0.8792, Accuracy: 70.37%
Epoch 6/20: Loss: 0.8593, Accuracy: 62.96%
Epoch 7/20: Loss: 0.7453, Accuracy: 81.48%
Epoch 8/20: Loss: 0.7987, Accuracy: 77.78%
Epoch 9/20: Loss: 0.8375, Accuracy: 62.96%
Epoch 10/20: Loss: 0.7265, Accuracy: 81.48%
Epoch 11/20: Loss: 0.6623, Accuracy: 74.07%
Epoch 12/20: Loss: 0.6321, Accuracy: 81.48%
Epoch 13/20: Loss: 0.5693, Accuracy: 96.30%
Reached high accuracy after 13 epochs. Stopping early.
Training completed in 13 epochs.
Final accuracy: 96.30%
Model saved to 'results/greek_model.pth'
```

I took 13 epochs to get high accuracy on 27 examples of greek letters.

Handwritten prediction results:



Unfortunately, I don't get good results, I get 25% accuracy. I tried making each letter slightly different with each handwriting to simulate a real world scenario. For my neural network modification, I freeze gradients on trained parts on mnist and load in the third layer as a fully connected layer per instructions.

```
Overall accuracy: 25.00% (3/12)

Detailed results:
1. alpha_4.png: True: alpha, Pred: gamma ✗
2. alpha_3.png: True: alpha, Pred: gamma ✗
3. alpha_2.png: True: alpha, Pred: alpha ✓
4. alpha.png: True: alpha.png, Pred: alpha ✗
5. beta_4.png: True: beta, Pred: gamma ✗
6. beta_2.png: True: beta, Pred: alpha ✗
7. beta_3.png: True: beta, Pred: alpha ✗
8. gamma_4.png: True: gamma, Pred: beta ✗
9. gamma_2.png: True: gamma, Pred: gamma ✓
10. gamma_3.png: True: gamma, Pred: gamma ✓
11. beta.png: True: beta.png, Pred: alpha ✗
12. gamma.png: True: gamma.png, Pred: gamma ✗
.venvmatejzecic@Matejs-MacBook-Pro cs5330-Project-5 %
```

Link for my handwritten greek letters (contains 4 examples for each letter):
https://drive.google.com/file/d/1-Uy9tnuNJNDvrr_dnkD7TbOxvMCDzlsG/view?usp=drive_link

4. Design your own experiment

For this experiment I used the MNIST dataset as suggested and automated the process by rotating through a number of different dimensions of the network one by one, kept the best scores per dimension, and kept rotating until I found the best parameters. Here's my hypothesis, plan, and results.

1. Plan for Network Optimization

My plan was to evaluate the effect of changing different aspects of a CNN architecture on the performance of the Fashion MNIST classification task. I chose to explore the following dimensions:

1. **Number of Convolutional Layers**: Testing 1, 2, or 3 layers to understand how network depth affects learning capacity
2. **Filter Configurations**: Testing small (8,16), medium (16,32), and large (32,64) filter counts to see how feature extraction capacity affects performance
3. **Kernel Sizes**: Testing 3×3 and 5×5 filters to understand the effect of receptive field size
4. **Hidden Layer Configurations**: Testing different hidden layer sizes (50, 100, 200) and structures (single vs. double layer)
5. **Dropout Rates**: Testing low (0.2), medium (0.5), and high (0.8) dropout to understand regularization effects
6. **Padding**: Testing different padding values (0, 1, 2) to understand how preserving spatial dimensions affects performance
7. **FC Dropout**: Testing the effect of including vs. excluding dropout after fully connected layers
8. **Activation Functions**: Testing ReLU, tanh, and Leaky ReLU to understand the effect of activation choice
9. **Learning Rates**: Testing 0.001, 0.01, and 0.1 to understand optimization dynamics

NOTE: These dimensions might get reduced by a few to save time training.

For evaluation, I used:

- **Primary Metric**: Test accuracy (measured as percentage correct on the Fashion MNIST test set)
- **Secondary Metrics**: Training time, epochs to convergence

I used a linear search approach where I optimized one parameter at a time while keeping others constant. This allowed me to systematically explore the parameter space without the exponential complexity of grid search.

2. Hypotheses

Before running the experiments, I made the following predictions:

1. **Number of Convolutional Layers**: I expected that 2 convolutional layers would perform better than 1, but 3 might lead to overfitting on this relatively simple dataset.
2. **Filter Configurations**: I predicted that larger filter counts would improve performance up to a point, but very large filters might lead to overfitting and increased computation without proportional accuracy gains.
3. **Kernel Sizes**: I expected that 3×3 kernels would perform better than 5×5 kernels because they would capture fine-grained features important for distinguishing similar clothing items.
4. **Hidden Layer Configurations**: I hypothesized that a moderate hidden layer size (100) would be optimal, with diminishing returns for larger sizes. I also expected that two smaller layers might perform better than one large layer due to increased representation power.
5. **Dropout Rates**: I expected a moderate dropout (0.5) to provide the best balance between regularization and maintaining important features.
6. **Padding**: I predicted that some padding (value 1) would improve performance by preserving spatial information at the edges of the image.
7. **FC Dropout**: I expected that including dropout after FC layers would improve generalization.
8. **Activation Functions**: I predicted that ReLU would outperform tanh and Leaky ReLU due to its simplicity and ability to avoid the vanishing gradient problem.
9. **Learning Rates**: I expected a moderate learning rate (0.01) to provide the best balance between convergence speed and stability.

3. Results and Analysis

[Here you should include the actual results from your experiments. Add tables showing the configurations tested and their performance metrics. Include visualizations of how each parameter affected accuracy.]

Summary of Results by Dimension

Experiment Summary:
Total configurations tested: 54
Best test accuracy: 90.15%
Best configuration:
  conv_layers: 1

fc_hidden_sizes: [200]
dropout_rates: [0.5]
fc_dropout: False
activation_func: leaky_relu
learning_rate: 0.1

These results aren't too far off from my hypothesis, but I do notice an increase in hidden layers and significant increase in learning rate, for the best accuracy of 90.15% to be achieved.

Extensions:

For my extension, I used 6 dimensions instead of three for part 3 of the assignment. It took a long time to run, but it allowed me to better explore what are the best possible parameters for highest accuracy on the fashion MNIST dataset.

Reflection of what I learned:

I learned a lot about neural networks in this assignment and throughout the lessons while working on this assignment. I learned about how to repurpose a network that's already trained, (transfer learning) and train a last layer to my specific case to get good results. Additionally, I learned how to change parameters and explore the best options for a specific dataset through automation and trial and error.

Resources:

https://docs.python.org/3/
https://pytorch.org/docs/stable/index.html
https://nextjournal.com/gkoehler/pytorch-mnist