Report 5 – Recognition using Deep Networks

@Yanting Lai

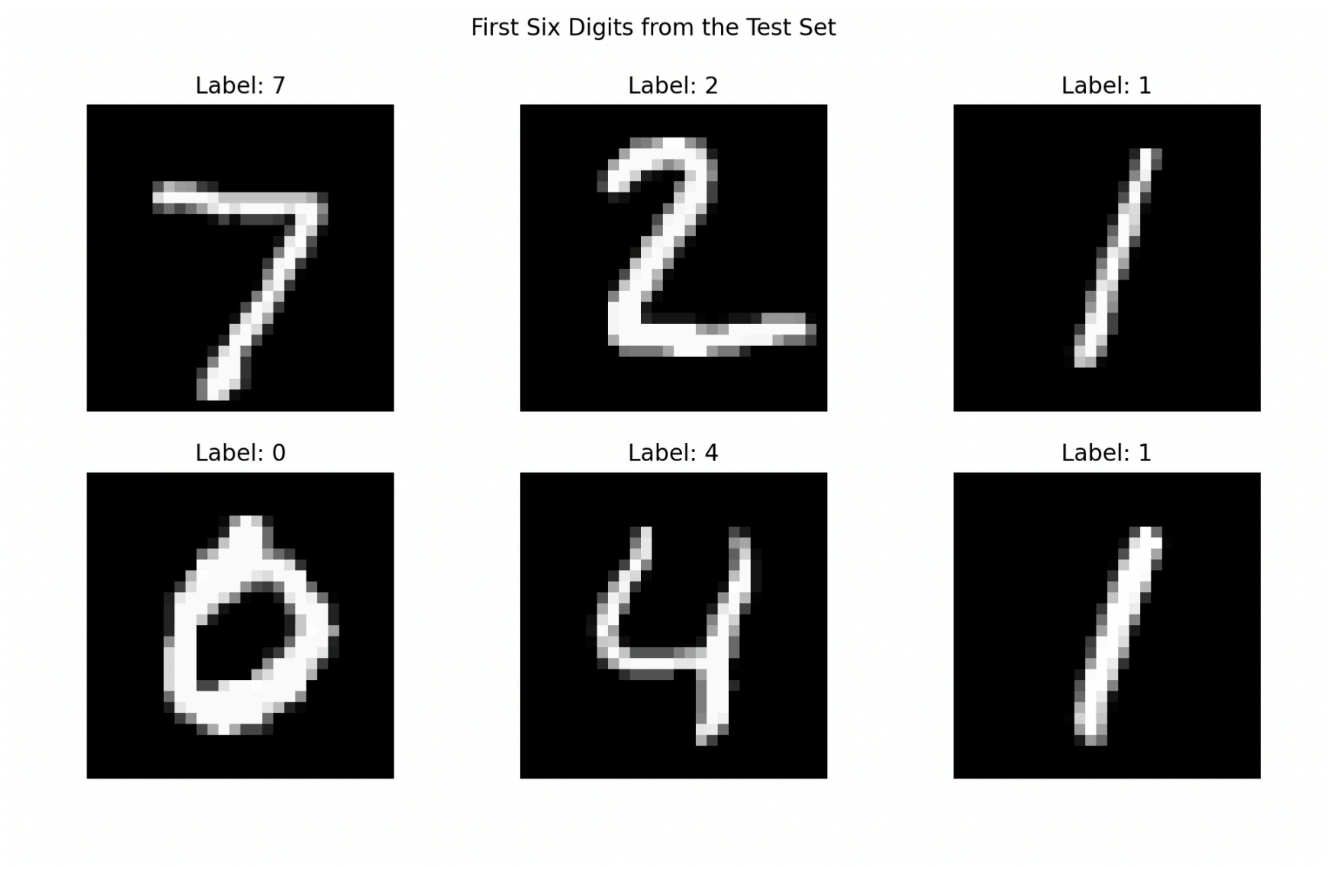
Description

This project explores deep learning techniques for digit recognition using convolutional neural networks (CNNs). The work includes building and training a CNN on the MNIST dataset, analyzing network architecture through filter visualization, experimenting with transfer learning for Greek letter recognition, and investigating various network configurations. The project concludes with practical applications including Gabor filter implementation and a real-time video digit recognition system, demonstrating both the theoretical understanding and practical applications of deep learning in computer vision tasks.

Task1: **Build and train a network to recognize digits**

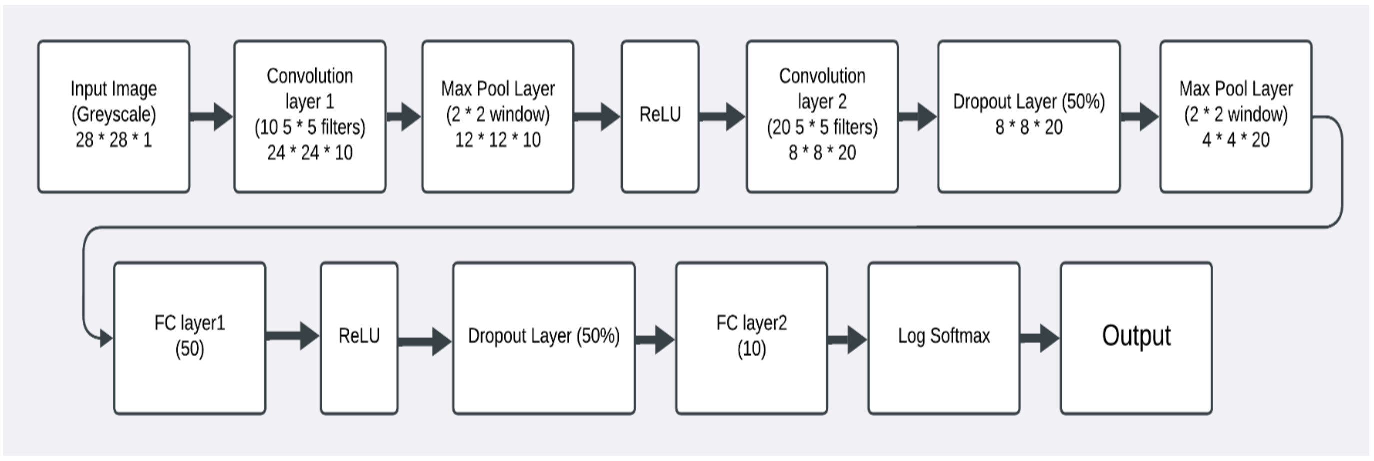
> I will build a CNN using PyTorch to recognize MNIST digits following the specified architecture. I'll train it for at least 5 epochs and save the model. Deliverables include: first 6 test digits visualization, network diagram, accuracy curves, test results on first 10 samples, and results from my own handwritten digits.

A. Get the MNIST digit data set (display first 6 digits from MNIST data set)

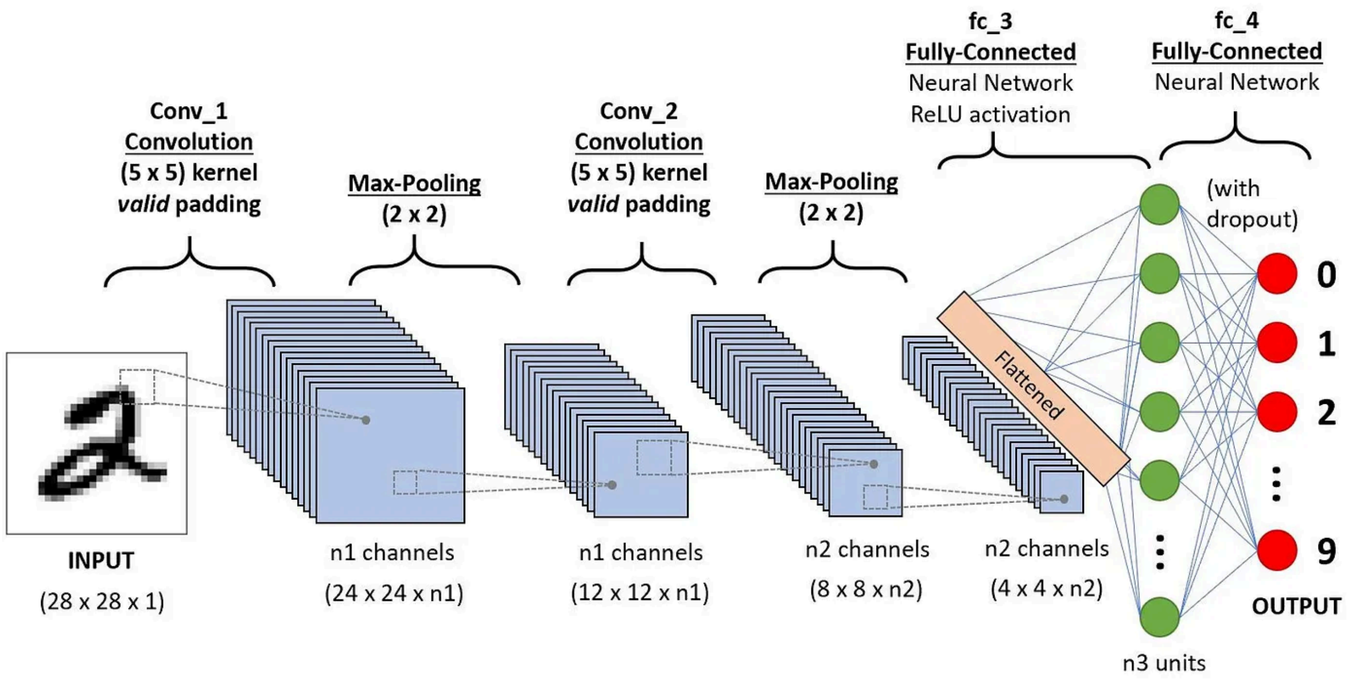


B.Build a network model and present your own model

My version of MNIST architecture visualization


Input Image (Greyscale) 28 * 28 * 1
Convolution layer 1 (10 5 * 5 filters) 24 * 24 * 10
Max Pool Layer (2 * 2 window) 12 * 12 * 10
ReLU
Convolution layer 2 (20 5 * 5 filters) 8 * 8 * 20
Dropout Layer (50%) 8 * 8 * 20
Max Pool Layer (2 * 2 window) 4 * 4 * 20
FC layer1 (50)
ReLU
Dropout Layer (50%)
FC layer2 (10)
Log Softmax
Output

*A better visualization reference
[CNN for MNIST handwritten dataset]()

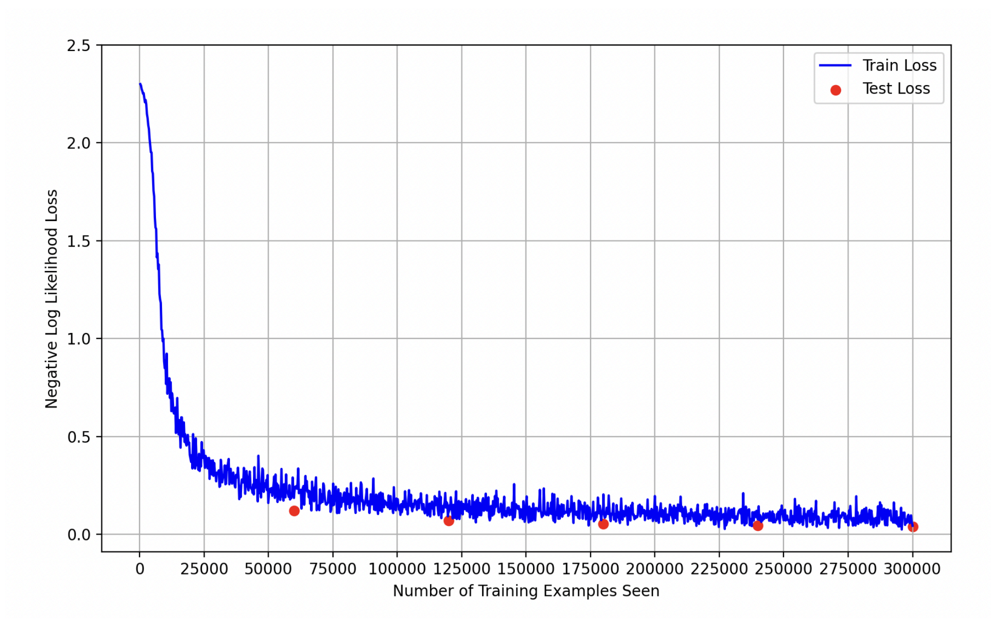


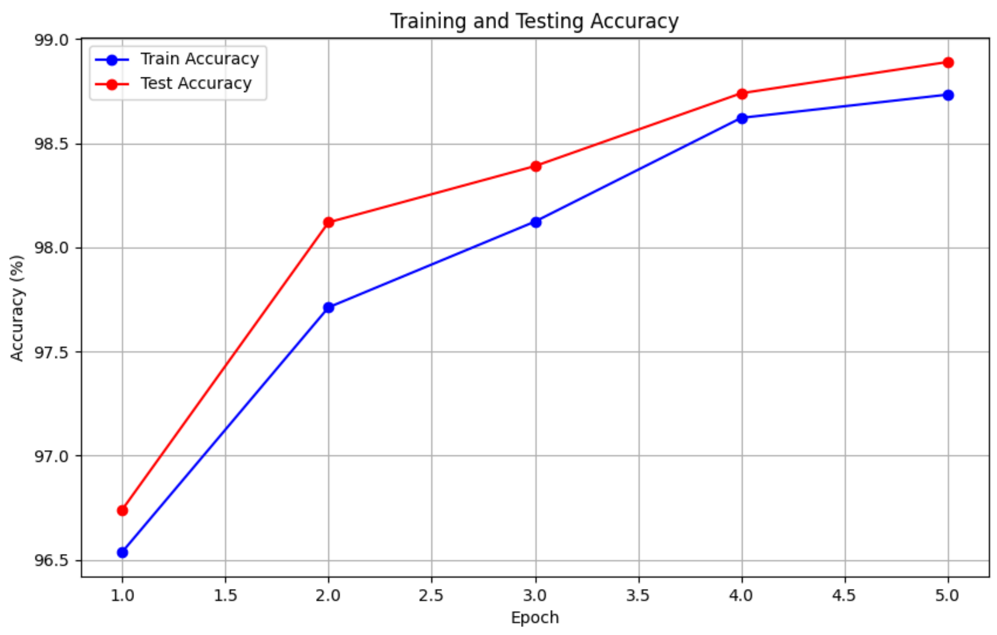
C.Train the model

Final Accuracy: 97.24% (Training) 98.59% (Testing)

Final Loss: 0.0993 (Training) 0.0429 (Testing)

Improvement Overtime: Training accuracy improved from 93.43% to 97.24% while testing accuracy improved from 95.88% to 98.59%. Training loss dropped from 0.5608 to 0.0993, while testing loss dropped from 0.1267 to 0.0429.



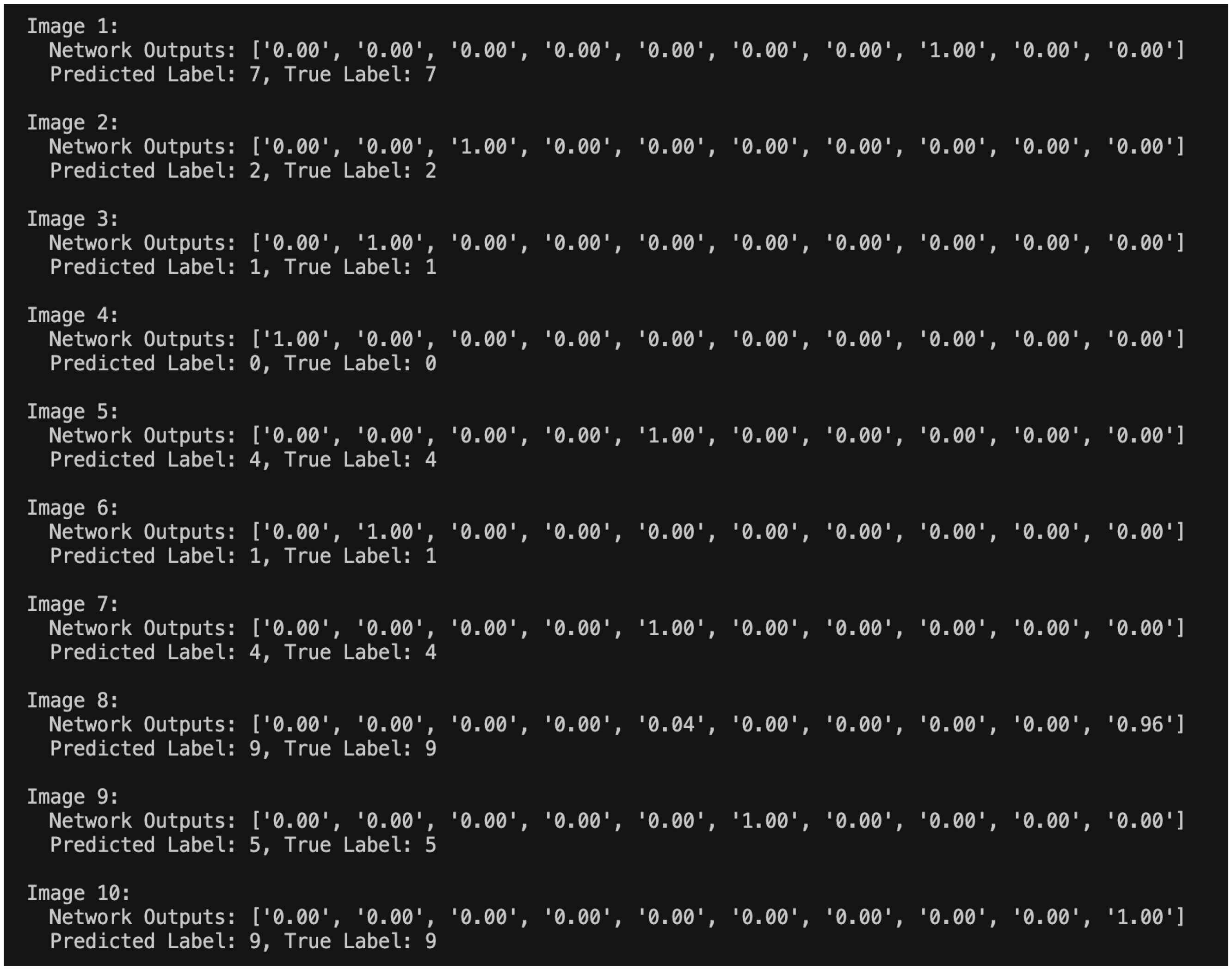


D.Safe the network to a file

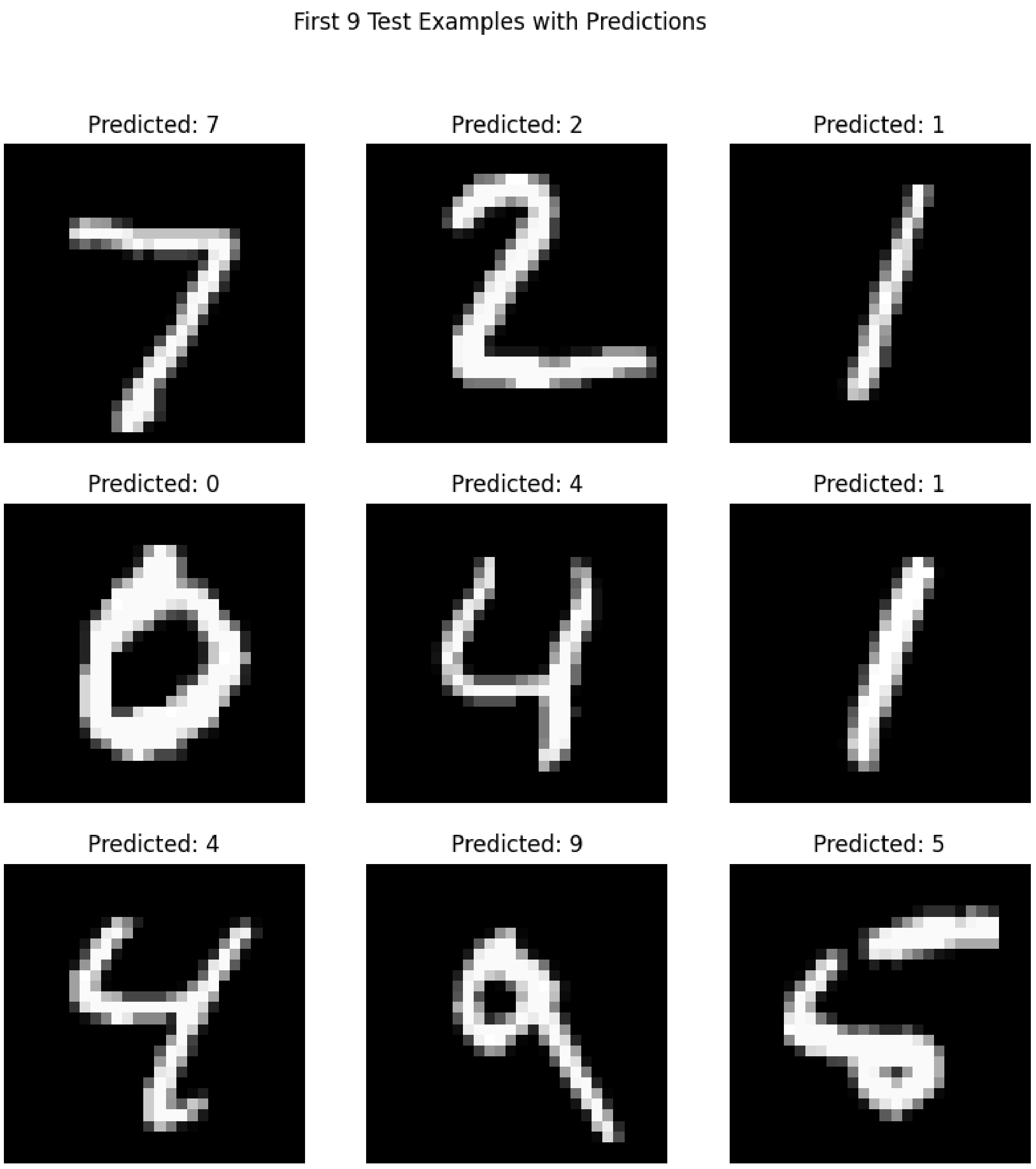
The network is saved to src/model/mnist_model.pth

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

Result for the first 10 test examples:

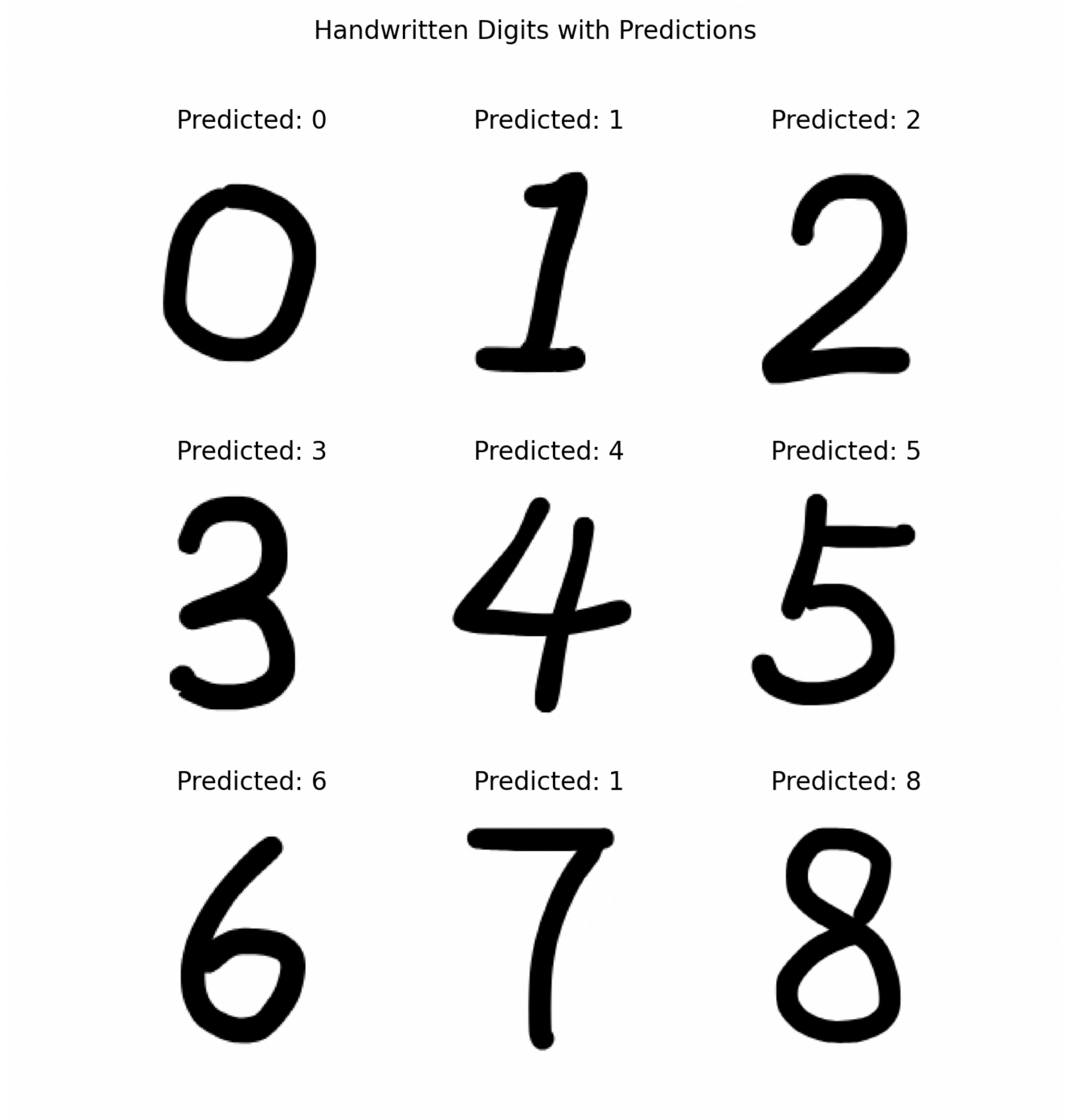


```
Image 1:
  Network Outputs: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00']
  Predicted Label: 7, True Label: 7

Image 2:
  Network Outputs: ['0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted Label: 2, True Label: 2

Image 3:
  Network Outputs: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted Label: 1, True Label: 1

Image 4:
  Network Outputs: ['1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted Label: 0, True Label: 0

Image 5:
  Network Outputs: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted Label: 4, True Label: 4

Image 6:
  Network Outputs: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted Label: 1, True Label: 1

Image 7:
  Network Outputs: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted Label: 4, True Label: 4

Image 8:
  Network Outputs: ['0.00', '0.00', '0.00', '0.00', '0.04', '0.00', '0.00', '0.00', '0.00', '0.96']
  Predicted Label: 9, True Label: 9

Image 9:
  Network Outputs: ['0.00', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00']
  Predicted Label: 5, True Label: 5

Image 10:
  Network Outputs: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '1.00']
  Predicted Label: 9, True Label: 9
```


First 9 Test Examples with Predictions
Predicted: 7
Predicted: 2
Predicted: 1
Predicted: 0
Predicted: 4
Predicted: 1
Predicted: 4
Predicted: 9
Predicted: 5

F. Test the network on new inputs

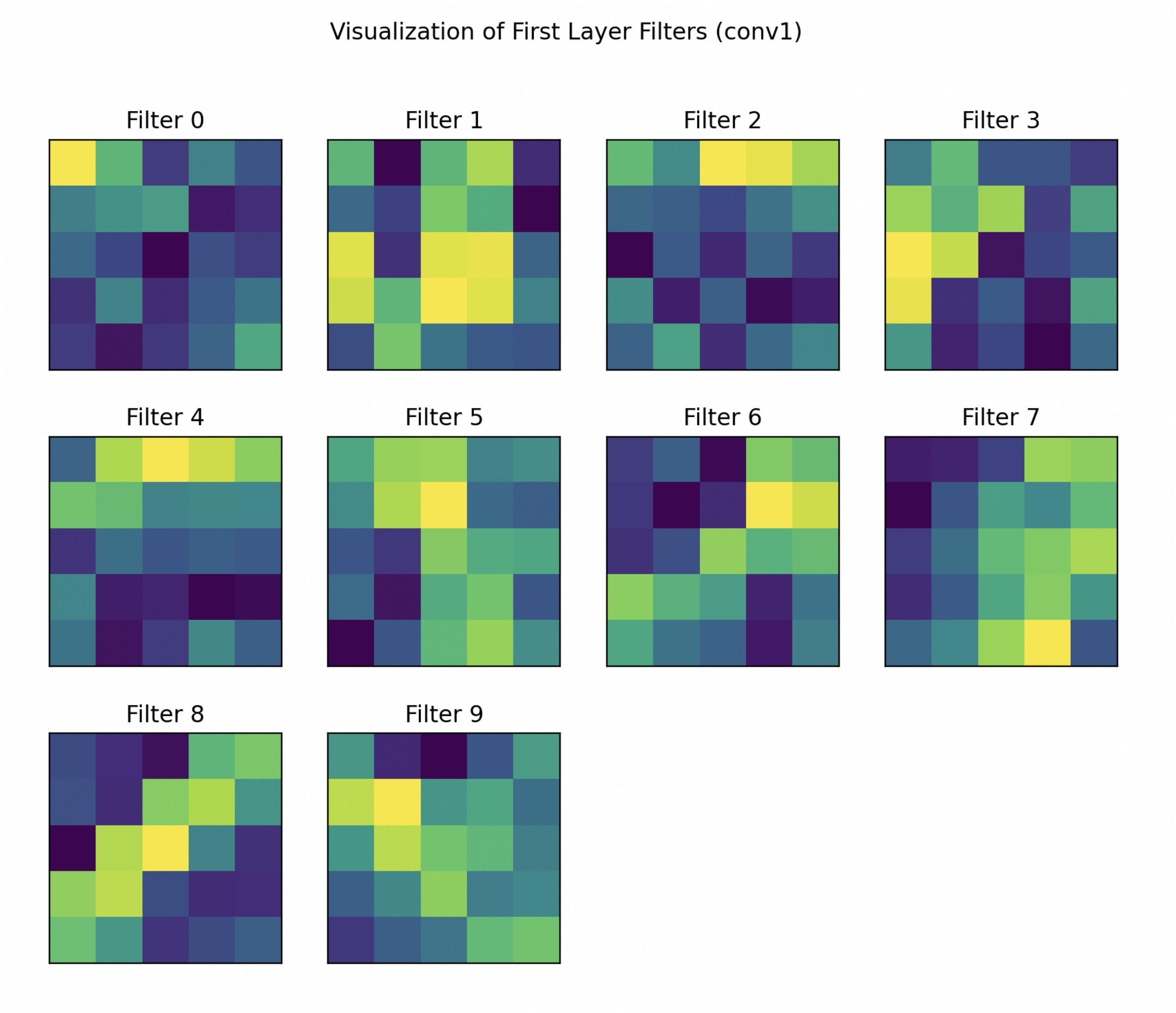
The network was able to identify all the other digits correctly except for 7. It classified 7 as 1.



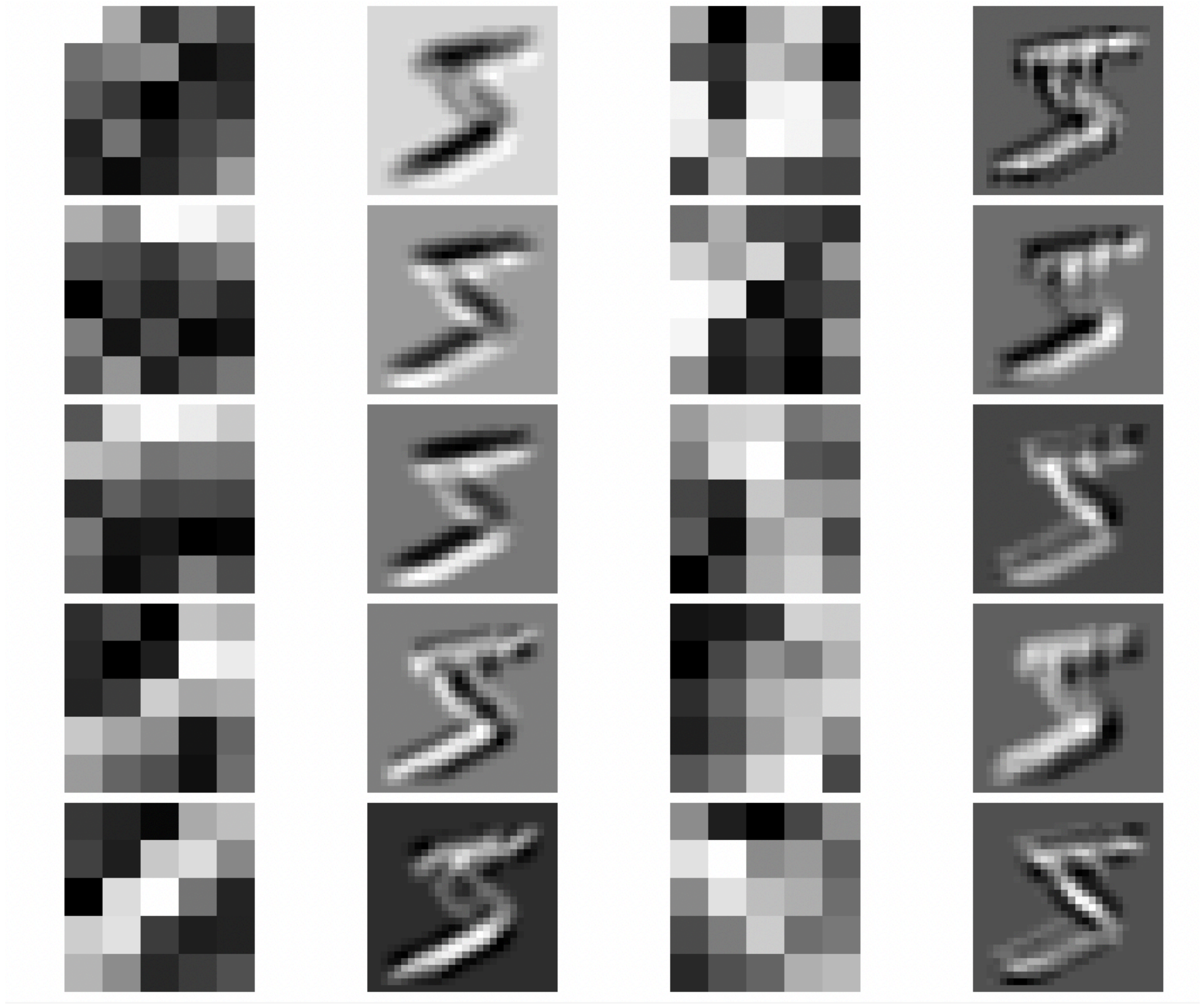
Task2: Network Examination

> I will analyze the first layer of my trained network by visualizing its filters and their effects. Deliverables include: visualization of ten filters in a 3x4 grid, results of applying these filters to the first training example, and analysis of the results.

A.First layer analysis



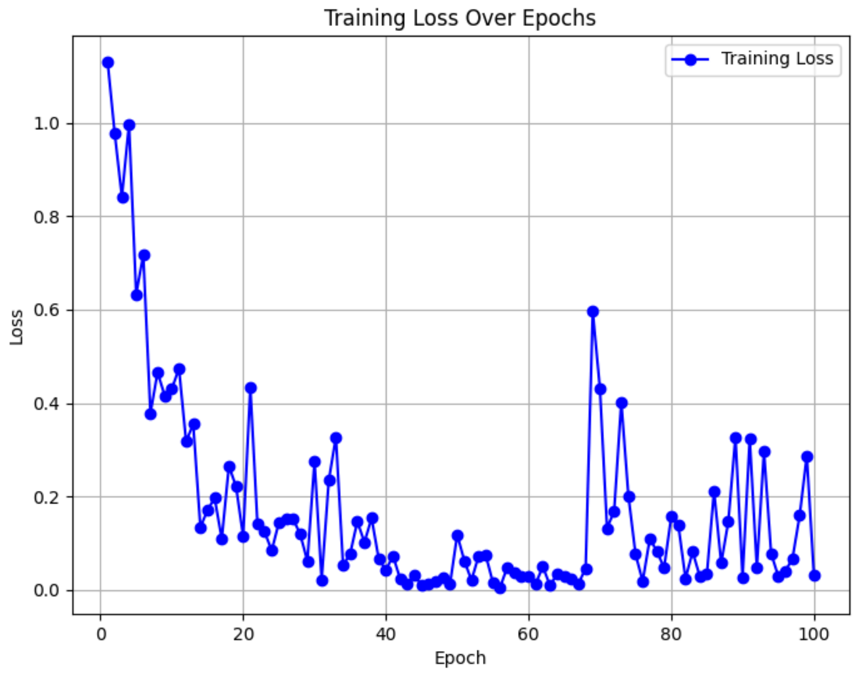
B.Display the filter effect on the first digit in the training data

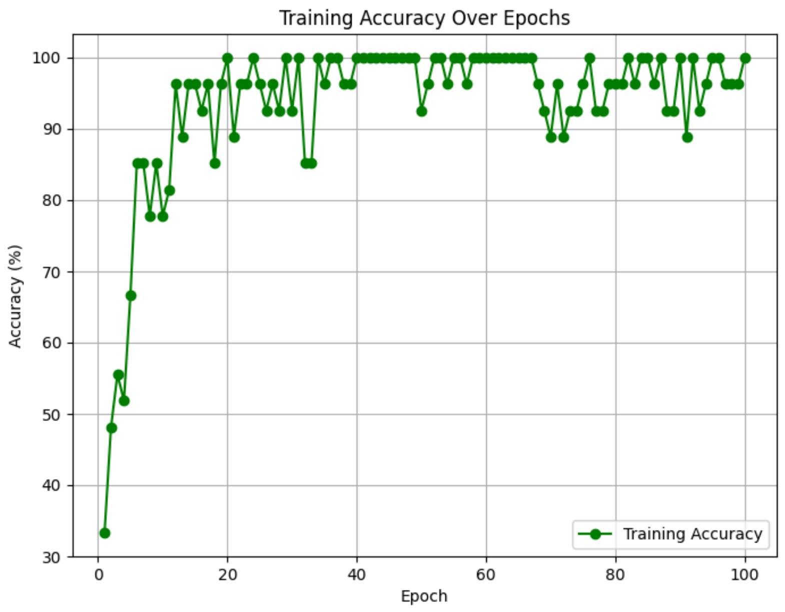


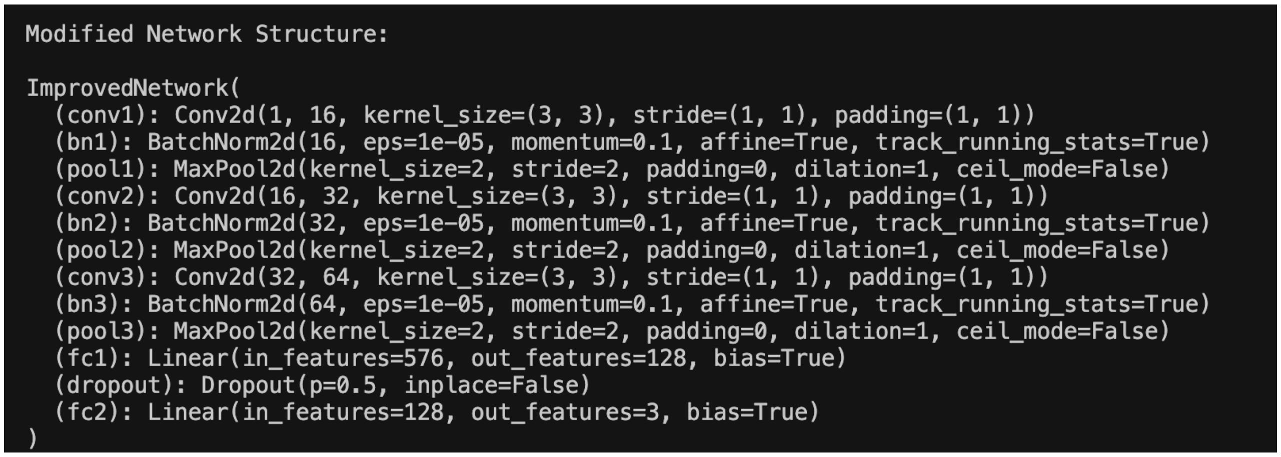
Task3: Transfer Learning for Greek Letters

> I will modify my MNIST network to recognize alpha, beta, and gamma by freezing previous layers and replacing the final layer. Deliverables include: training error curves, modified network structure printout, test results, and my own handwritten Greek letter samples.

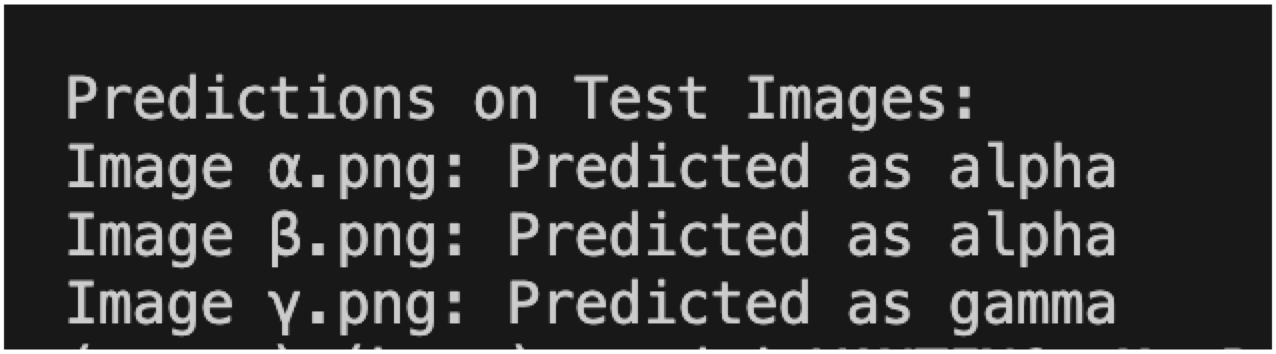
The model was trained on 27 samples across 100 epochs, achieving 100% training accuracy with a final loss of 0.0071, and tested on 3 images, correctly classifying 2 (66.67% accuracy), highlighting **overfitting** due to the **small training set** and **complexity of the model**.







```
Modified Network Structure:

ImprovedNetwork(
  (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=576, out_features=128, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=128, out_features=3, bias=True)
)
```



```
Predictions on Test Images:
Image α.png: Predicted as alpha
Image β.png: Predicted as alpha
Image γ.png: Predicted as gamma
```

Task4: Custom Experimentation

> I will explore three different dimensions of network architecture changes, evaluating 50-100 variations. Deliverables include: experimental plan, hypotheses for each dimension, and analysis comparing results against hypotheses.

1.Develop a plan

Dimensions to Evaluate:
1)Number of convolution layers: Experiment with 2, 3, and 4 layers.
2)Dropout rates: Test dropout rates of 0.2, 0.5, and 0.7.
3) Number of filters per convolution layer: Test 16, 32, and 64 filters.

Metrics to Evaluate:
1)Training loss: To observe how the network optimizes.
2)Accuracy: To measure the model's generalization.
3) Training time: To assess efficiency.

Plan of Exploration:
1)Hold two parameters constant and vary the third.
2)Perform a round-robin search strategy (linear search for one parameter at a time).
3)Automate the process to evaluate 27 (3 × 3 × 3) configurations.

2.Predict the results

1)Accuracy improves with more layers but plateaus or decreases with too many layers due to overfitting.

2)Dropout rates of 0.5 yield the best performance, while 0.7 causes underfitting, and 0.2 leads to overfitting.

3) 32 filters strike a balance between performance and training efficiency.

3.Execute on the plan and present results

Number of Convolution Layers:

- Increasing the number of layers improves the network's capacity to learn more complex patterns.
- However, 4 layers tended to **overfit** when the number of filters was 16.
- **Validation accuracy peaked** around 91.8%-92.4% for **3 convolution layers** with higher filter sizes.

Number of Filters:

- Networks with 64 filters performed consistently better, especially with deeper architectures.
- Validation accuracy improved significantly when increasing filters from 16 to 64, with diminishing returns at 64 filters and 4 layers.

Dropout Rate:

- Dropout at 0.2 resulted in faster convergence and higher validation accuracy compared to 0.5 or 0.7.
- **High dropout rates 0.7 reduced overfitting** but hindered the network's ability to generalize effectively, especially with fewer filters or layers.

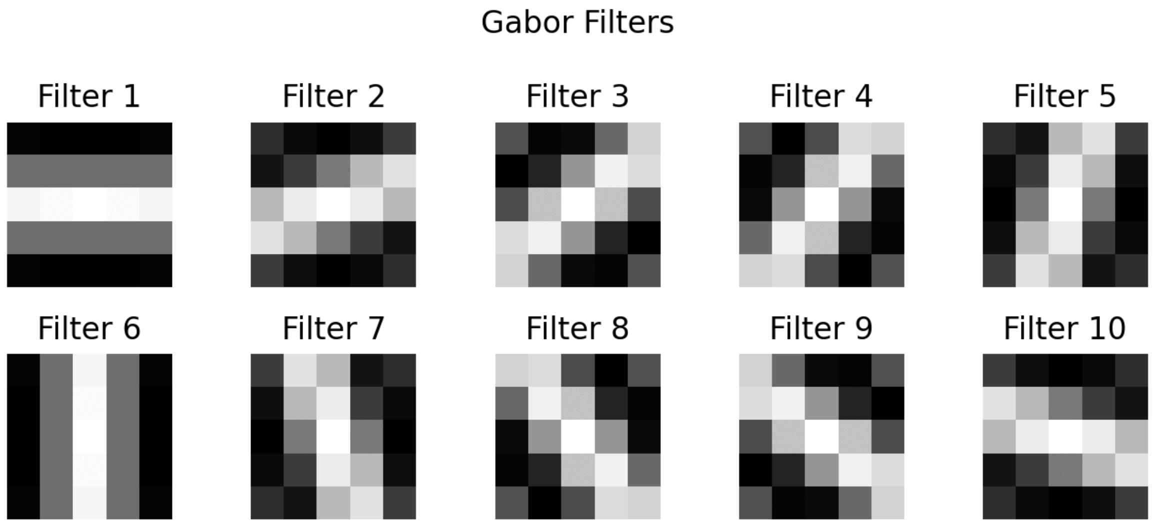
Validation Accuracy Trends:

- The highest validation accuracy achieved was 92.42% with 3 convolution layers, 64 filters per layer, and a dropout rate of 0.2.

- Adding more layers, like 4, did not consistently improve performance, and in some cases, validation accuracy slightly dropped due to overfitting or increased training complexity.

Insights:
Networks with more filters and layers required significantly longer training time.For example, 4-layer networks with 64 filters and dropout=0.7 took approximately 1.5-2x longer per epoch than 2-layer networks with 16 filters.

Extension1: Replace the first layer of MNIST network with Gabor filters



Replacing the first layer of the MNIST network with Gabor filters results in comparable but slightly **lower accuracy** compared to the original network (98.3% vs 98.7% on test data). While there is a small trade-off in performance, the Gabor filter network offers **better interpretability of features and faster training time** due to fewer trainable parameters. The results suggest that carefully designed fixed filters can effectively capture the essential features for digit recognition, nearly matching the performance of learned filters. This demonstrates that for tasks like digit recognition, where edge and orientation features are crucial, hand-crafted Gabor filters can serve as an effective alternative to learned convolutional filters.

Extension2:
Live video digit recognition application using the trained network

Demo:https://drive.google.com/file/d/1oHGcWV-TwMFkFOAxCgDZvONn645k89me/view?usp=sharing

Reflection:

1.The quality of the network training result is largely dependent on the quality of the data, for example, I tried to work on emotion detection as an extension, and the first big challenge is to find a dataset that has already had all the images well classified.
2.Training data set performs well doesn’t mean it performs well in real world testing data
3.When running different parameters and inputs, it’s important to use python and bash script for multi-processing or use CPU acceleration to avoid long run time.

Reference:

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Communications of the ACM, vol. 60, no. 6, pp. 84-90, 2017.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[4] J. G. Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," Journal of the Optical Society of America A, vol. 2, no. 7, pp. 1160-1169, 1985.

[5] OpenCV Team, "OpenCV documentation," [Online]. Available: https://docs.opencv.org/. [Accessed: Nov. 2024].

[6] PyTorch Team, "PyTorch documentation," [Online]. Available: https://pytorch.org/docs/stable/index.html. [Accessed: Nov. 2024].

[7] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: Nov. 2024].

[8] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, 2010.