Guorun Huang, Project5 Report
Pattern Recognition and Computer Vision, 25Fall

1. Project Overview

In this project, I built and trained a convolutional neural network for MNIST digit recognition using PyTorch, including convolution, pooling, dropout, and fully connected layers. The trained model was saved and later reused for evaluation. I analyzed the first convolutional layer by visualizing its learned filters and applying them to sample images using OpenCV's filter2D, confirming that the filters captured meaningful edge and stroke patterns.

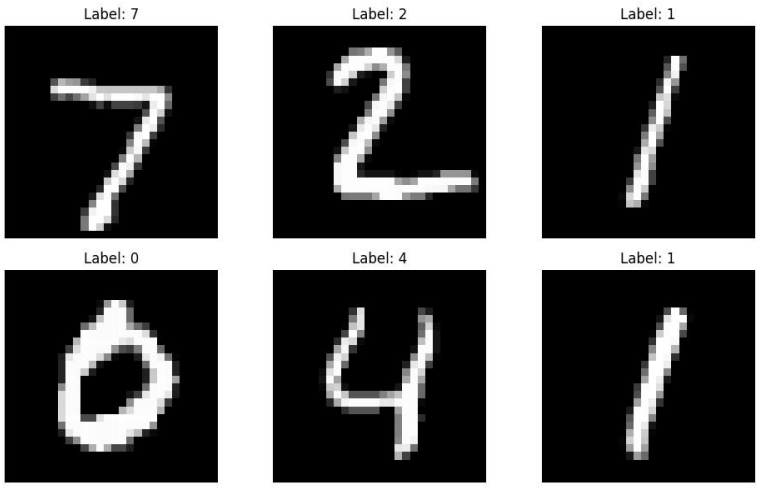
I then performed transfer learning on Greek letters and examined misclassification behavior by relating errors to the extracted filter responses. Finally, I designed a series of controlled experiments, exploring variations in network depth, filter sizes, dropout rates, activations, and optimizers. I also replaced the first layer with handcrafted filters—Sobel, Laplacian, Gaussian, and Gabor—and retrained only the remaining layers. Gaussian produced the strongest results, with 0° Gabor performing second best, demonstrating how different low-level feature extractors influence network performance.

2. Results

1 MNIST digits recognition

A. obtain the dataset

The MNIST digit data consists of a training set of 60k 28x28 labeled digits and a test set of 10k 28x28 labeled digits. The data set can be imported as torchvision.datasets.MNIST. The plots of the first six sample digits from the test set are shown below.

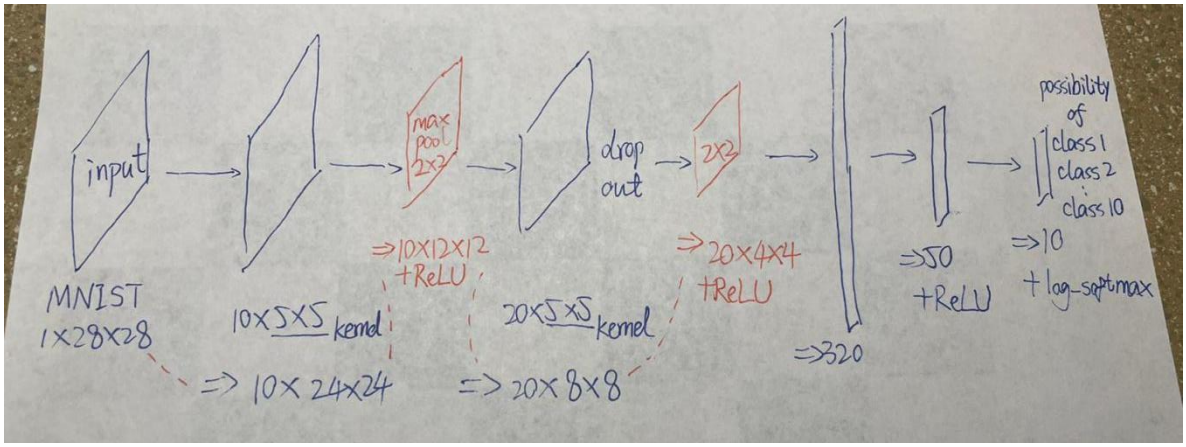


B. establish the network

Created a network with the following layers.

- A convolution layer with 10 5x5 filters
- A max pooling layer with a 2x2 window and a ReLU function applied.

- A convolution layer with 20 5x5 filters
- A dropout layer with a 0.5 dropout rate (50%)
- A max pooling layer with a 2x2 window and a ReLU function applied
- A flattening operation followed by a fully connected Linear layer with 50 nodes and a ReLU function on the output
- A final fully connected Linear layer with 10 nodes and the log_softmax function applied to the output.



A diagram of network:

```
Creating network...
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (dropout): 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)
)
```

For MNIST (the input has 1 channel), the first convolution layer has: 10 filters, each of size $1 \times 5 \times 5$

The weight shape is (10, 1, 5, 5).

The output of the first layer has 10 channels, so the second layer must be defined as:

`nn.Conv2d(in_channels=10, out_channels=20, kernel_size=5)`

Thus, the second convolution layer has: 20 filters, each of size $10 \times 5 \times 5$

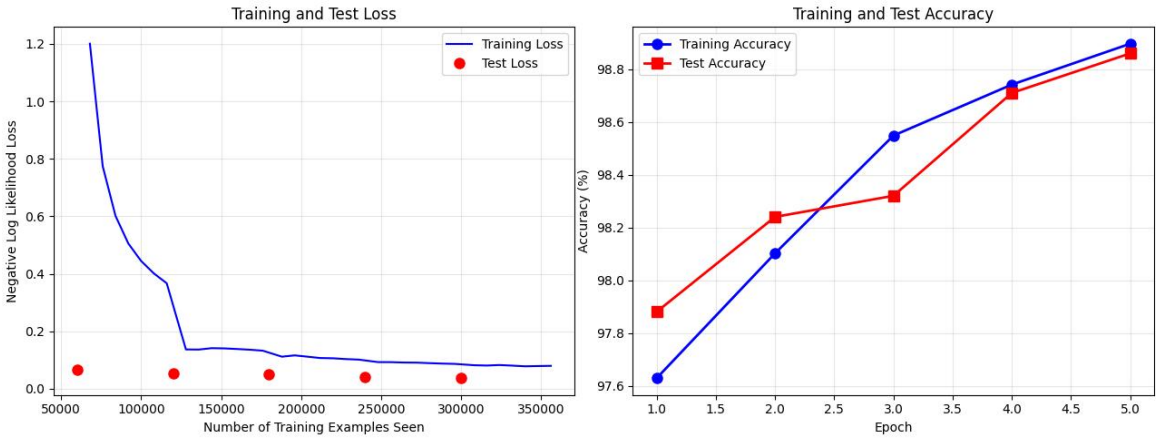
The weight shape is (20, 10, 5, 5).

Each filter is applied to the 10 output channels from the first layer, and the results are

summed element-wise across channels:
output_k = sum over i = 1..10 of conv2d(feature_map_i, kernel_k_i)

Summary:
If the input has C channels, then each filter contains C kernels, each of size k × k
Output = Σ(convolution results across all channels) + bias
No averaging is performed.
Kernels are not shared across different input channels.

C. train
Evaluating the model on both the training and test sets after each epoch. Collect the accuracy scores and plot the training and testing accuracy in a graph.



(x axis is number of training example seen, unit = 25000, y is negative log negative loss, unit = 0.5, print train loss as a blue curve line, for every 1000 train example seen added, and print test loss as a red point)

Reflection：
The reason why the test loss is around ~0.08 at 60k, 120k, 180k, and 240k example seen(though with a decline trend) is that we are using MNIST — a rather easy dataset. The model usually converges after seeing only 5k–20k samples (less than 1 epoch). In other words, **during the first few thousand training samples**, the loss drops rapidly from high(like the training loss) to below 0.1. Training for another 5 epochs will not significantly reduce the test loss any further. The model has already almost perfectly fit MNIST.

Tthe test loss doesn't start high (e.g., from ~2.3) like the training loss. This is normal behavior for convolutional networks training on MNIST. Early in training, the model starts with random weights. The training loss drops quickly (you can see this). The test loss is first computed at the end of epoch 1. By the time epoch 1 finishes, the model has already processed all 60,000 training samples. At this point the model is already performing quite well (test loss ~0.08). So the first test-loss point is already after 60,000 training steps, meaning it is naturally low.

D. save trained nn

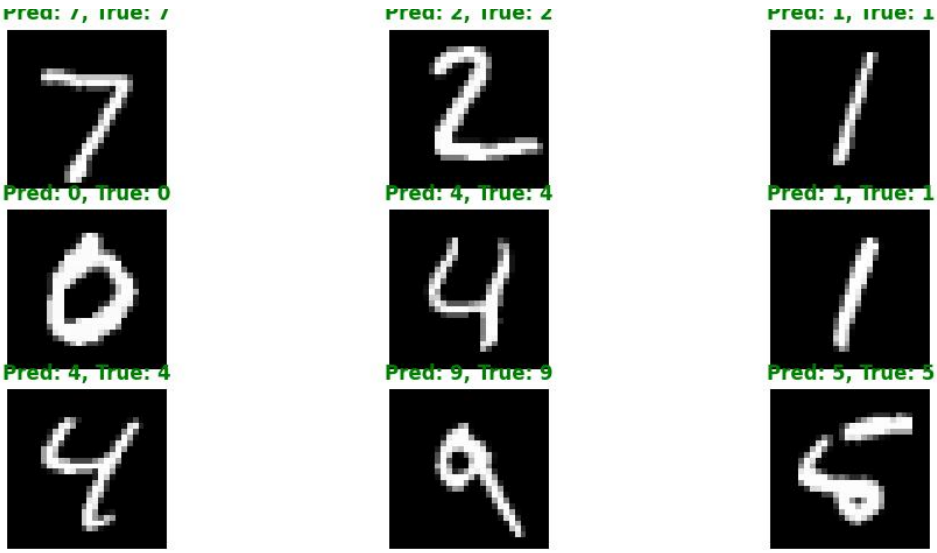
```
--- Epoch 5/5 ---
Epoch 5, Examples: 308000/60000, Loss: 0.0818
Epoch 5, Examples: 316000/60000, Loss: 0.0807
Epoch 5, Examples: 324000/60000, Loss: 0.0825
Epoch 5, Examples: 332000/60000, Loss: 0.0803
Epoch 5, Examples: 340000/60000, Loss: 0.0780
Epoch 5, Examples: 348000/60000, Loss: 0.0788
Epoch 5, Examples: 356000/60000, Loss: 0.0796
Training Set - Average loss: 0.0367, Accuracy: 59338/60000 (98.90%)
Test Set - Average loss: 0.0365, Accuracy: 9886/10000 (98.86%)

Plotting results...
Training results saved to 'training_results.png'

Saving model...
Model saved to 'mnist_model.pth'
Model size: 88.69 KB

Training complete!
```

E. run on the test

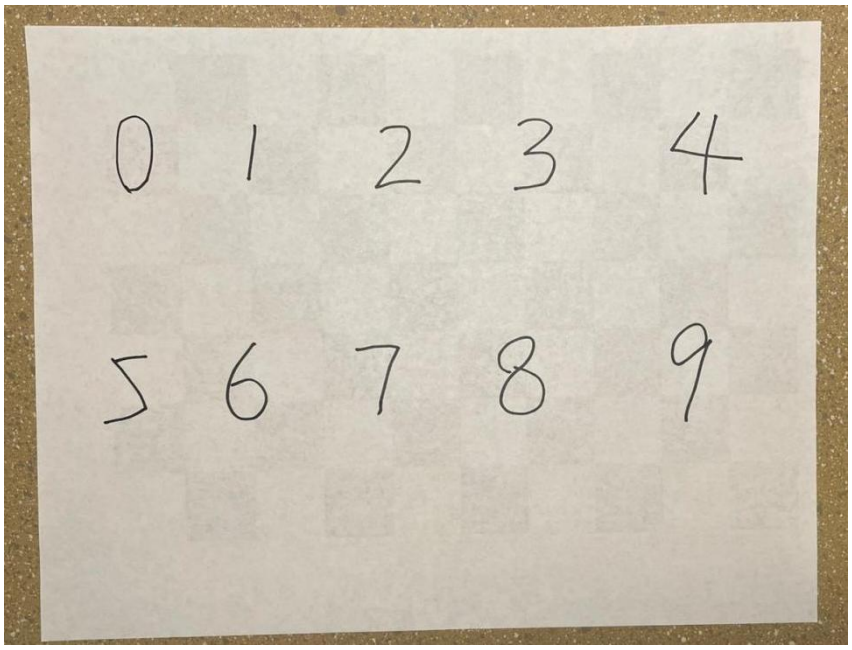
Analysis why the 10 network output values (use only 2 decimal places) **look like one-hot**:
The MNIST model's output is from log_softmax, the output might look like:
[-15.32, -10.55, -0.01, -12.33, ...]
Apply torch.exp() to convert it to probabilities:
[0.00..., 0.00..., 0.99..., 0.00..., ...]
If we want to see the details, remove rounding or print more decimal places.
For example: print([float(f"{v:.6f}") for v in probs])
we will get something like:
[0.000012, 0.000001, 0.998712, 0.000003, ...]

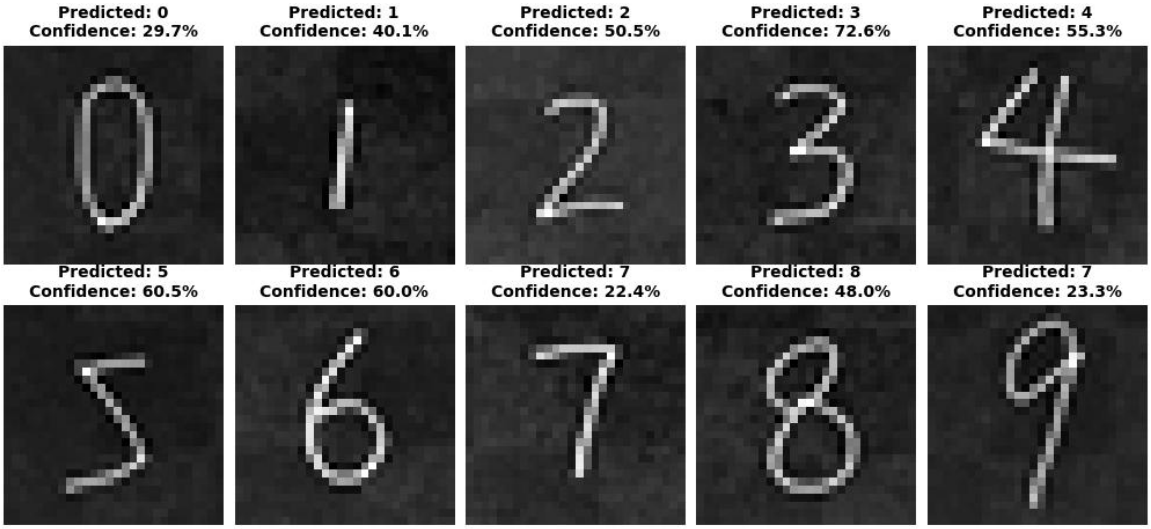
```
Using device: cpu

Loading MNIST test dataset...

Loading trained model...
Model loaded from mnist_model.pth

================================================================================
Testing first 10 examples from test set
================================================================================

Example 1:
Network outputs: [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00]
Predicted digit: 7 (max output at index 7)
True label: 7
Result: ✓ CORRECT

Example 2:
Network outputs: [0.00, 0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00]
Predicted digit: 2 (max output at index 2)
True label: 2
Result: ✓ CORRECT

Example 3:
Network outputs: [0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00]
Predicted digit: 1 (max output at index 1)
True label: 1
Result: ✓ CORRECT

Example 4:
Network outputs: [1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00]
Predicted digit: 0 (max output at index 0)
```

```
Network outputs: [0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00]
Predicted digit: 4 (max output at index 4)
True label: 4
Result: ✓ CORRECT

Example 8:
Network outputs: [0.00, 0.00, 0.00, 0.00, 0.02, 0.00, 0.00, 0.00, 0.00, 0.98]
Predicted digit: 9 (max output at index 9)
True label: 9
Result: ✓ CORRECT

Example 9:
Network outputs: [0.00, 0.00, 0.00, 0.00, 0.00, 0.98, 0.02, 0.00, 0.00, 0.00]
Predicted digit: 5 (max output at index 5)
True label: 5
Result: ✓ CORRECT

Example 10:
Network outputs: [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00]
Predicted digit: 9 (max output at index 9)
True label: 9
Result: ✓ CORRECT

================================================================================
Accuracy on first 10 examples: 10/10 (100%)
================================================================================


Plotting first 9 examples...
First 9 predictions saved to 'first_nine_predictions.png'

Testing complete!
```



F. custom input (handwritten)

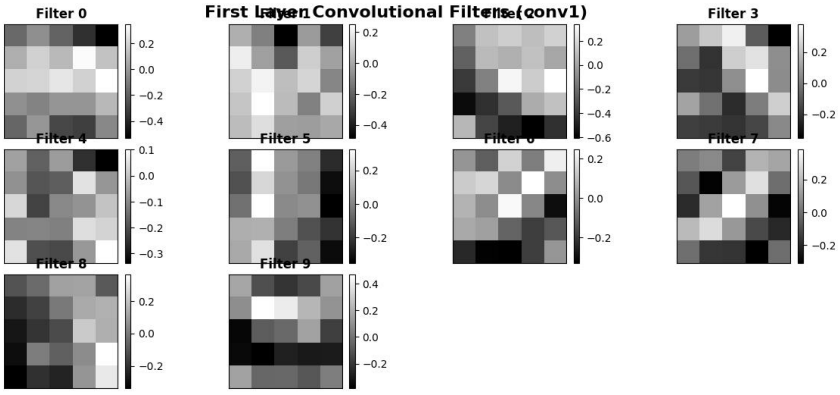


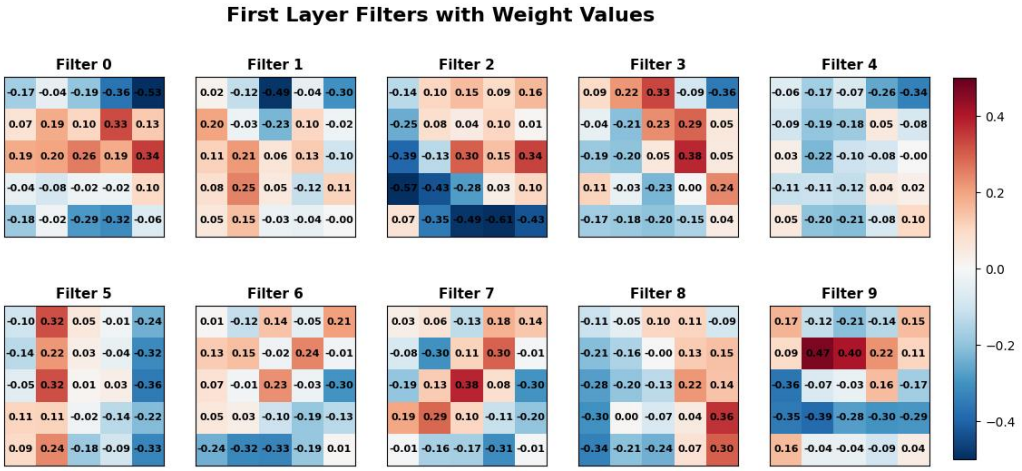


The digit 9 was not recognized well and was classified as 7. This may be because the stroke was not thick enough, or the loop was too small or too close together. As a result, during the early convolution/filtering stages, the shape of the 9 may have been **blurred out, leaving only a horizontal and vertical stroke, which caused it to be misclassified as a 7**. The other digits were recognized well.

2 Examine your network

A. print the shape of frist layer of nn

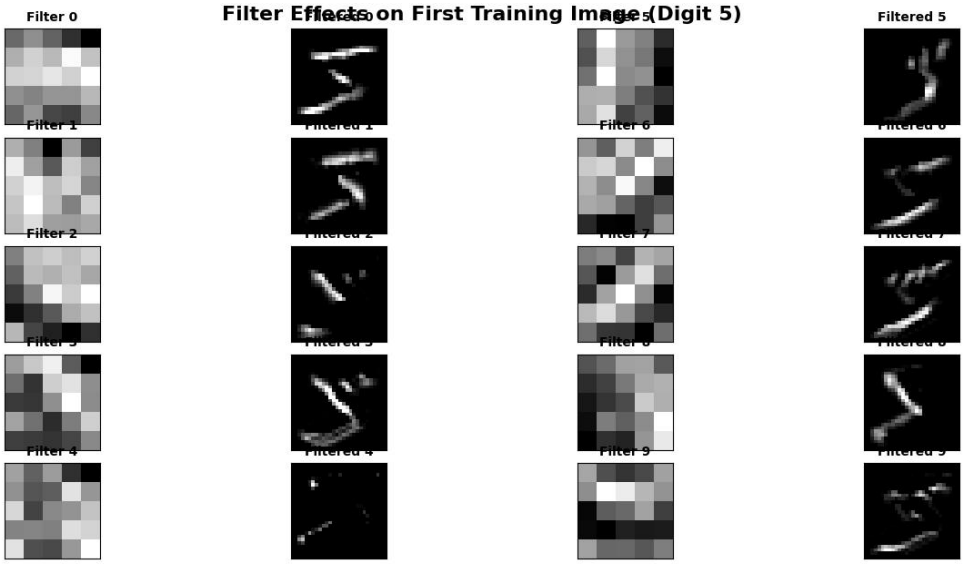




I visualized each filter as a 5×5 block and added color to it, reddish colors represent positive weights, and bluish colors represent negative weights.
I also added a color bar to make the interpretation more intuitive.
Positive weights:
→ High input pixel values produce a **larger convolution output**
→ The model is actively **looking for that pattern**
Negative weights:
→ Low input pixel values produce a larger convolution output
→ Detects the **suppresses foreground** / inverse structure / emphasizes background
Each convolution kernel:
= A **local pattern detector** learned by the model

B. visualize

Use OpenCV's filter2D function to apply the 10 filters to the first training example image.
Generate a plot of the 10 filtered images.

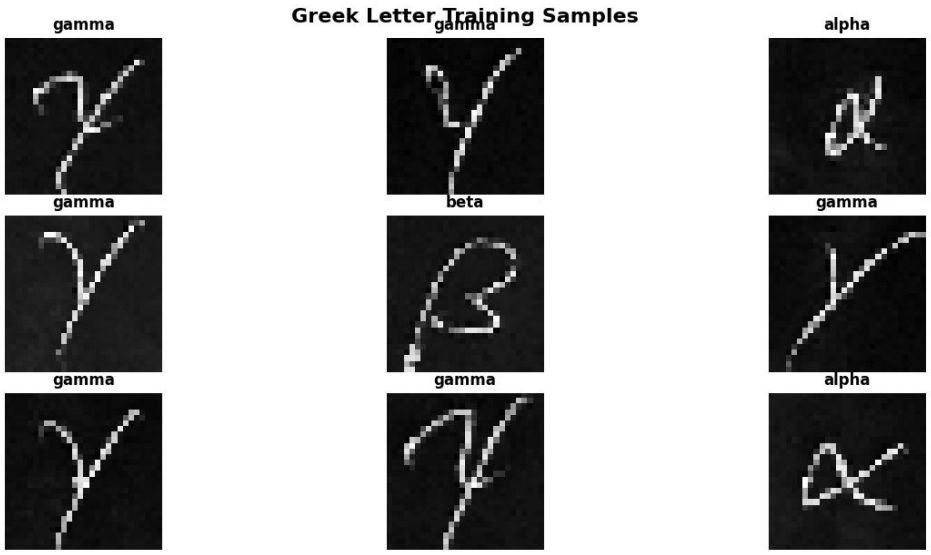


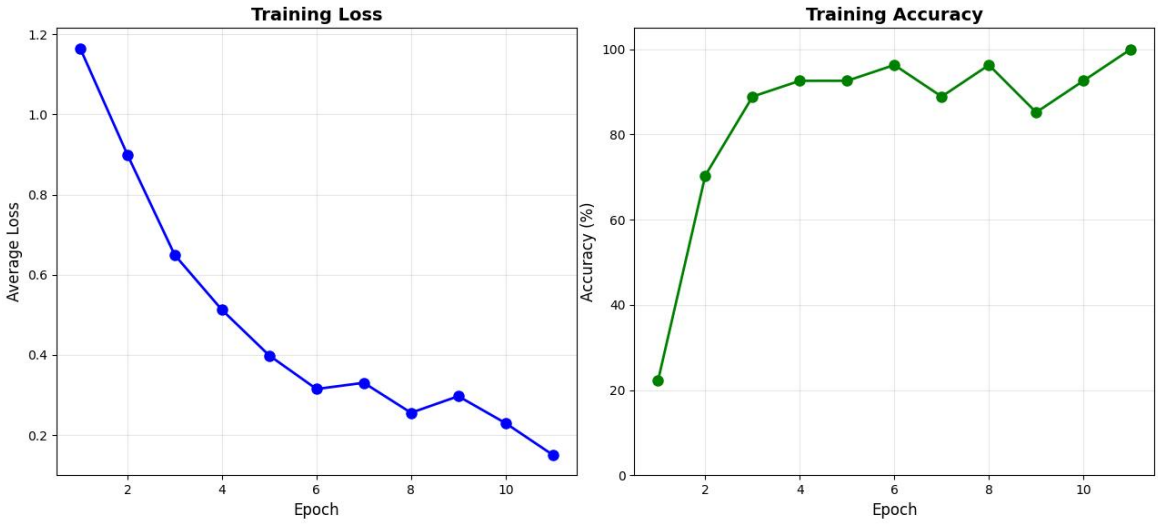
The filtered images make sense. Each filter is **a learned pattern detector**.
The first-layer filters typically learn to detect very basic visual structures: **vertical strokes, horizontal strokes, diagonal edges, corners**. Each filtered output shows how strongly each pixel matches the filter's pattern.
We can see some filters highlight the left side of the digit, some highlight the top curve, some activate around corners, some seem to detect the thick strokes, **some mostly stay dark (meaning they are not useful for this particular digit)**.

3 Greek letters

A. 7*3 imgs, 20 epochs

We need 3 epochs to make the training accuracy to achieve 90%, and we need 10 epochs to make the training accuracy to be close to 100%.

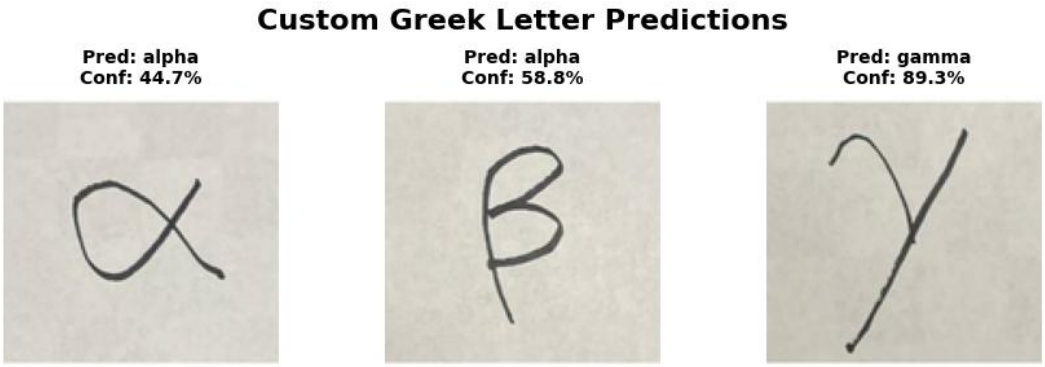




B. custom α, β, γ

Include in your report a plot of the training error, a printout of your modified network, and the results on the additional data. In your readme, please include a link to a zip file or

drive with your additional examples in your submission.




```
Trained model saved to 'greek_model.pth'

Testing on custom images from './custom_greek'...

Testing on 3 custom images...
================================================================================
alpha_processed.png:
  Prediction: alpha
  Confidence: 44.69%
  Probabilities: alpha=44.69%  beta=41.66%  gamma=13.65%

beta_processed.png:
  Prediction: alpha
  Confidence: 58.77%
  Probabilities: alpha=58.77%  beta=19.14%  gamma=22.10%

gamma_processed.png:
  Prediction: gamma
  Confidence: 89.29%
  Probabilities: alpha=6.00%  beta=4.71%  gamma=89.29%

Custom predictions saved to 'custom_greek_predictions.png'
================================================================================

Transfer learning complete!

Summary:
  - Epochs needed: 12
```

Reflection:

The neural network classified β as α mainly because the low-level convolutional filters tend to detect: horizontal edges, vertical edges, diagonal edges, curved edges.

Many of the edge directions in β are the same as those in α, the difference lies only in how these edges are combined. If this combination of edges is degraded by **blurring, resizing, or noise**, the model can no longer distinguish between the two letters.

4 Experiments

Experimental design is a very important part of the process. There are three common design strategies that I believe worth discussing.

1. Linear Search

Only one parameter is adjusted at a time while all others are kept fixed.
Drawback is that when parameters interact with each other (for example, the combination "more filters + higher dropout" performs better), linear search is unlikely to discover such interactions.

2. Grid Search (L × M × N)

Choose 3 dimensions, for example:
Number of convolution filters: [10, 20, 40]
Fully connected hidden units: [50, 100, 200]
Dropout: [0.0, 0.25, 0.5]
Run all combinations: 3 × 3 × 3 = 27 networks.
Characteristics: Systematically covers your predefined parameter grid. But the search space grows exponentially with each added dimension. A practical approach is to grid-search only 2–3 key hyperparameters (e.g., conv filters × hidden units), while fixing the others to reasonable default values.

3. Orthogonal Experimental Design

The idea is similar to factorial design, but instead of testing all combinations, you carefully select a smaller set of combinations such that each parameter level appears in a balanced way, and the effect of different factors can be estimated cleanly.
Example :
Parameter A (filters): [16, 32]
Parameter B (hidden units): [100, 200]
Parameter C (dropout): [0.25, 0.5]
Full combinations would be 2 × 2 × 2 = 8.
An orthogonal table may select only 4 combinations while ensuring balanced coverage, such as:
A=16, B=100, C=0.25
A=16, B=200, C=0.5
A=32, B=100, C=0.5
A=32, B=200, C=0.25
Advantage is that we can have balanced coverage using fewer experiments. Drawback is that designing orthogonal tables manually is difficult.

My experimental design uses MNIST Fashion dataset **and linear search**, where for each dimension I iterate through all possible values while keeping the other parameters fixed. The parameters are as follows:

```
 conv_filters_options = [
    [8], [12], [16], [20], [32],                    # 1
    [6,12], [8,16], [10,20], [12,24], [16,32],     # 2
    [24,48], [32,64],
    [8,16,32], [10,20,40], [20,40,80]               # 3
]
conv_kernel_options = [
    [3], [5], [7],                                  # 1
    [1,1], [3,3], [3,5], [3,7], [5,3], [5,5],      # 2
```

```
        [5,7], [7,3], [7,5], [7,7],
        [3,3,3], [5,5,5], [7,7,7],                    # 3
        [7,5,3], [5,5,3], [7,7,5]
    ]
        'num_conv_layers': [1, 2, 3],
        'conv_filters': conv_filters_options,
        'conv_kernel_size': conv_kernel_options,
        'fc_hidden_nodes': [16, 25, 50, 64, 100, 128],
        'conv_dropout_rate': [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6],
        'fc_dropout_rate':     [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6],
        'dropout_after_fc1': [False, True],
        'activation': ['relu', 'tanh', 'leaky_relu'],
        'optimizer': ['adam', 'sgd', 'rmsprop'],
        'learning_rate': [0.2, 0.1, 0.05, 0.01, 0.001]
```

Assumption

Since the Fashion-MNIST dataset is more challenging, I assumed that the following configuration would yield the best accuracy: 3 convolutional layers, Dropout rate = 0.1 (because a higher rate may cause excessive information loss and prevent the model from finding good parameters, while a lower rate increases the risk of overfitting), ReLU activation, Adam optimizer, Learning rate = 0.01 (so the model does not miss the optimal parameter region for convergence), this combination is expected to produce the best performance.

Results:

```
(mnist_env)>P5-experiment-v2.py
Using device: cpu

Generating experiment plan: linear_search
Target number of experiments: 50
Epochs per experiment: 4
Generated 50 unique configurations
Loading Fashion MNIST dataset...
Training samples: 60000
Test samples: 10000

================================================================================
RUNNING 50 EXPERIMENTS
================================================================================

Experiment 1/50
Config: {
  "num_conv_layers": 2,
  "conv_filters": [
    10,
    20
  ],
  "conv_kernel_size": [
    5,
    5
  ],
  "pool_positions": [
    true,
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
```

```
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 82.23%, Test Acc: 83.96%
    Epoch 4/4: Train Acc: 83.40%, Test Acc: 84.33%
    ✓ Test Acc: 84.33%, Params: 21,840, Time: 115.5s

Experiment 2/50
Config: {
    "num_conv_layers": 1,
    "conv_filters": [
        8
    ],
    "conv_kernel_size": [
        3
    ],
    "pool_positions": [
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 88.73%, Test Acc: 88.15%
    Epoch 4/4: Train Acc: 90.26%, Test Acc: 88.77%
    ✓ Test Acc: 88.77%, Params: 68,240, Time: 95.6s

Experiment 3/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        6,
        12
    ],
    "conv_kernel_size": [
        1,
        1
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 75.58%, Test Acc: 81.14%
    Epoch 4/4: Train Acc: 77.22%, Test Acc: 79.97%
    ✓ Test Acc: 79.97%, Params: 30,056, Time: 103.7s

Experiment 4/50
Config: {
    "num_conv_layers": 3,
    "conv_filters": [
        8,
        16,
        32
    ],
    "conv_kernel_size": [
        3,
        3,
        3
    ],
    "pool_positions": [
        false,
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
```

```
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 80.62%, Test Acc: 83.48%
  Epoch 4/4: Train Acc: 81.73%, Test Acc: 83.16%
  ✓ Test Acc: 83.16%, Params: 46,448, Time: 142.6s

Experiment 5/50
Config: {
  "num_conv_layers": 1,
  "conv_filters": [
    12
  ],
  "conv_kernel_size": [
    3
  ],
  "pool_positions": [
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 88.78%, Test Acc: 88.30%
  Epoch 4/4: Train Acc: 90.63%, Test Acc: 88.97%
  ✓ Test Acc: 88.97%, Params: 102,080, Time: 107.5s

Experiment 6/50
Config: {
  "num_conv_layers": 1,
  "conv_filters": [
    16
  ],
  "conv_kernel_size": [
    3
  ],
  "pool_positions": [
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 89.19%, Test Acc: 87.40%
  Epoch 4/4: Train Acc: 90.88%, Test Acc: 88.96%
  ✓ Test Acc: 88.96%, Params: 135,920, Time: 112.8s

Experiment 7/50
Config: {
  "num_conv_layers": 1,
  "conv_filters": [
    20
  ],
  "conv_kernel_size": [
    3
  ],
  "pool_positions": [
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 88.97%, Test Acc: 87.50%
  Epoch 4/4: Train Acc: 90.44%, Test Acc: 88.13%
  ✓ Test Acc: 88.13%, Params: 169,760, Time: 124.2s

Experiment 8/50
Config: {
```

```
  "num_conv_layers": 1,
  "conv_filters": [
    32
  ],
  "conv_kernel_size": [
    3
  ],
  "pool_positions": [
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 88.66%, Test Acc: 84.78%
  Epoch 4/4: Train Acc: 90.36%, Test Acc: 88.09%
  ✓ Test Acc: 88.09%, Params: 271,280, Time: 149.9s

Experiment 9/50
Config: {
  "num_conv_layers": 2,
  "conv_filters": [
    8,
    16
  ],
  "conv_kernel_size": [
    1,
    1
  ],
  "pool_positions": [
    true,
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 81.71%, Test Acc: 80.85%
  Epoch 4/4: Train Acc: 82.64%, Test Acc: 82.34%
  ✓ Test Acc: 82.34%, Params: 39,920, Time: 109.9s

Experiment 10/50
Config: {
  "num_conv_layers": 2,
  "conv_filters": [
    10,
    20
  ],
  "conv_kernel_size": [
    1,
    1
  ],
  "pool_positions": [
    true,
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 79.14%, Test Acc: 82.95%
  Epoch 4/4: Train Acc: 80.59%, Test Acc: 82.87%
  ✓ Test Acc: 82.87%, Params: 49,800, Time: 116.7s

Experiment 11/50
Config: {
  "num_conv_layers": 2,
  "conv_filters": [
    12,
```

```
      24
    ],
    "conv_kernel_size": [
      1,
      1
    ],
    "pool_positions": [
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 81.86%, Test Acc: 82.44%
    Epoch 4/4: Train Acc: 83.19%, Test Acc: 83.84%
    ✓ Test Acc: 83.84%, Params: 59,696, Time: 124.1s

Experiment 12/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
      16,
      32
    ],
    "conv_kernel_size": [
      1,
      1
    ],
    "pool_positions": [
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 82.62%, Test Acc: 83.82%
    Epoch 4/4: Train Acc: 83.72%, Test Acc: 84.10%
    ✓ Test Acc: 84.10%, Params: 79,536, Time: 137.5s

Experiment 13/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
      24,
      48
    ],
    "conv_kernel_size": [
      1,
      1
    ],
    "pool_positions": [
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 83.44%, Test Acc: 83.91%
    Epoch 4/4: Train Acc: 84.82%, Test Acc: 85.14%
    ✓ Test Acc: 85.14%, Params: 119,408, Time: 164.3s

Experiment 14/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
      32,
```

```
      64
    ],
    "conv_kernel_size": [
      1,
      1
    ],
    "pool_positions": [
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 83.66%, Test Acc: 84.13%
    Epoch 4/4: Train Acc: 84.74%, Test Acc: 85.30%
    √  Test Acc: 85.30%, Params: 159,536, Time: 196.8s

Experiment 15/50
Config: {
    "num_conv_layers": 3,
    "conv_filters": [
      10,
      20,
      40
    ],
    "conv_kernel_size": [
      3,
      3,
      3
    ],
    "pool_positions": [
      false,
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 80.40%, Test Acc: 83.33%
    Epoch 4/4: Train Acc: 80.84%, Test Acc: 82.42%
    √  Test Acc: 82.42%, Params: 59,720, Time: 178.9s

Experiment 16/50
Config: {
    "num_conv_layers": 3,
    "conv_filters": [
      20,
      40,
      80
    ],
    "conv_kernel_size": [
      3,
      3,
      3
    ],
    "pool_positions": [
      false,
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 80.24%, Test Acc: 81.86%
    Epoch 4/4: Train Acc: 81.73%, Test Acc: 82.76%
    √  Test Acc: 82.76%, Params: 136,880, Time: 293.1s
```

Experiment 17/50
Config: {

```
  "num_conv_layers": 1,
  "conv_filters": [
    8
  ],
  "conv_kernel_size": [
    5
  ],
  "pool_positions": [
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
```

}
  Epoch 2/4: Train Acc: 87.93%, Test Acc: 87.02%
  Epoch 4/4: Train Acc: 89.26%, Test Acc: 87.81%
  √ Test Acc: 87.81%, Params: 58,368, Time: 99.6s

Experiment 18/50
Config: {

```
  "num_conv_layers": 1,
  "conv_filters": [
    8
  ],
  "conv_kernel_size": [
    7
  ],
  "pool_positions": [
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
```

}
  Epoch 2/4: Train Acc: 86.78%, Test Acc: 86.36%
  Epoch 4/4: Train Acc: 87.91%, Test Acc: 86.29%
  √ Test Acc: 86.29%, Params: 49,360, Time: 99.4s

Experiment 19/50
Config: {

```
  "num_conv_layers": 2,
  "conv_filters": [
    6,
    12
  ],
  "conv_kernel_size": [
    3,
    3
  ],
  "pool_positions": [
    true,
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
```

}
  Epoch 2/4: Train Acc: 81.75%, Test Acc: 84.63%
  Epoch 4/4: Train Acc: 82.84%, Test Acc: 85.68%
  √ Test Acc: 85.68%, Params: 16,280, Time: 98.9s
Experiment 20/50
Config: {

```
  "num_conv_layers": 2,
  "conv_filters": [
    6,
    12
```

```
    ],
    "conv_kernel_size": [
        3,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 78.70%, Test Acc: 82.22%
    Epoch 4/4: Train Acc: 80.25%, Test Acc: 82.17%
    √  Test Acc: 82.17%, Params: 12,032, Time: 100.2s

Experiment 21/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        6,
        12
    ],
    "conv_kernel_size": [
        3,
        7
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 79.84%, Test Acc: 82.22%
    Epoch 4/4: Train Acc: 80.52%, Test Acc: 83.39%
    √  Test Acc: 83.39%, Params: 9,560, Time: 99.9s

Experiment 22/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        6,
        12
    ],
    "conv_kernel_size": [
        5,
        3
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 80.80%, Test Acc: 84.02%
    Epoch 4/4: Train Acc: 81.80%, Test Acc: 83.42%
    √  Test Acc: 83.42%, Params: 16,376, Time: 99.2s

Experiment 23/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        6,
        12
```

```
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 79.62%, Test Acc: 82.63%
    Epoch 4/4: Train Acc: 80.70%, Test Acc: 82.34%
    √  Test Acc: 82.34%, Params: 12,128, Time: 100.0s

Experiment 24/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        6,
        12
    ],
    "conv_kernel_size": [
        5,
        7
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 78.86%, Test Acc: 82.18%
    Epoch 4/4: Train Acc: 79.61%, Test Acc: 83.17%
    √  Test Acc: 83.17%, Params: 9,656, Time: 100.1s

Experiment 25/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        6,
        12
    ],
    "conv_kernel_size": [
        7,
        3
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 78.01%, Test Acc: 82.03%
    Epoch 4/4: Train Acc: 78.98%, Test Acc: 81.64%
    √  Test Acc: 81.64%, Params: 11,120, Time: 100.0s

Experiment 26/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        6,
        12
```

```
    ],
    "conv_kernel_size": [
      7,
      5
    ],
    "pool_positions": [
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 76.53%, Test Acc: 79.97%
    Epoch 4/4: Train Acc: 77.75%, Test Acc: 80.93%
    √  Test Acc: 80.93%, Params: 8,072, Time: 99.6s

Experiment 27/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
      6,
      12
    ],
    "conv_kernel_size": [
      7,
      7
    ],
    "pool_positions": [
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 75.39%, Test Acc: 79.23%
    Epoch 4/4: Train Acc: 76.62%, Test Acc: 80.07%
    √  Test Acc: 80.07%, Params: 6,800, Time: 101.3s

Experiment 28/50
Config: {
    "num_conv_layers": 3,
    "conv_filters": [
      8,
      16,
      32
    ],
    "conv_kernel_size": [
      5,
      5,
      5
    ],
    "pool_positions": [
      false,
      true,
      true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 74.44%, Test Acc: 78.02%
    Epoch 4/4: Train Acc: 76.34%, Test Acc: 79.53%
    √  Test Acc: 79.53%, Params: 31,216, Time: 134.8s

Experiment 29/50
Config: {
    "num_conv_layers": 3,
```

```
  "conv_filters": [
    8,
    16,
    32
  ],
  "conv_kernel_size": [
    7,
    7,
    7
  ],
  "pool_positions": [
    false,
    true,
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 71.42%, Test Acc: 76.43%
  Epoch 4/4: Train Acc: 73.07%, Test Acc: 77.05%
  √ Test Acc: 77.05%, Params: 33,968, Time: 129.9s
Experiment 30/50
Config: {
  "num_conv_layers": 3,
  "conv_filters": [
    8,
    16,
    32
  ],
  "conv_kernel_size": [
    7,
    5,
    3
  ],
  "pool_positions": [
    false,
    true,
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
}
  Epoch 2/4: Train Acc: 72.21%, Test Acc: 76.65%
  Epoch 4/4: Train Acc: 72.86%, Test Acc: 76.37%
  √ Test Acc: 76.37%, Params: 23,216, Time: 127.6s

Experiment 31/50
Config: {
  "num_conv_layers": 3,
  "conv_filters": [
    8,
    16,
    32
  ],
  "conv_kernel_size": [
    5,
    5,
    3
  ],
  "pool_positions": [
    false,
    true,
    true
  ],
  "pool_size": 2,
  "fc_hidden_nodes": 50,
  "dropout_after_conv": true,
  "conv_dropout_rate": 0.5,
  "dropout_after_fc1": false,
  "fc_dropout_rate": 0.5,
  "activation": "relu",
  "optimizer": "adam",
  "learning_rate": 0.01
```

```
}
   Epoch 2/4: Train Acc: 77.23%, Test Acc: 80.83%
   Epoch 4/4: Train Acc: 78.44%, Test Acc: 80.35%
   √  Test Acc: 80.35%, Params: 34,224, Time: 135.2s

Experiment 32/50
Config: {
   "num_conv_layers": 3,
   "conv_filters": [
      8,
      16,
      32
   ],
   "conv_kernel_size": [
      7,
      7,
      5
   ],
   "pool_positions": [
      false,
      true,
      true
   ],
   "pool_size": 2,
   "fc_hidden_nodes": 50,
   "dropout_after_conv": true,
   "conv_dropout_rate": 0.5,
   "dropout_after_fc1": false,
   "fc_dropout_rate": 0.5,
   "activation": "relu",
   "optimizer": "adam",
   "learning_rate": 0.01
}
   Epoch 2/4: Train Acc: 70.92%, Test Acc: 74.50%
   Epoch 4/4: Train Acc: 72.43%, Test Acc: 74.76%
   √  Test Acc: 74.76%, Params: 26,480, Time: 131.9s

Experiment 33/50
Config: {
   "num_conv_layers": 2,
   "conv_filters": [
      10,
      20
   ],
   "conv_kernel_size": [
      5,
      5
   ],
   "pool_positions": [
      true,
      true
   ],
   "pool_size": 2,
   "fc_hidden_nodes": 16,
   "dropout_after_conv": true,
   "conv_dropout_rate": 0.5,
   "dropout_after_fc1": false,
   "fc_dropout_rate": 0.5,
   "activation": "relu",
   "optimizer": "adam",
   "learning_rate": 0.01
}
   Epoch 2/4: Train Acc: 80.83%, Test Acc: 83.64%
   Epoch 4/4: Train Acc: 82.22%, Test Acc: 84.21%
   √  Test Acc: 84.21%, Params: 10,586, Time: 115.2s

Experiment 34/50
Config: {
   "num_conv_layers": 2,
   "conv_filters": [
      10,
      20
   ],
   "conv_kernel_size": [
      5,
      5
   ],
   "pool_positions": [
      true,
      true
   ],
   "pool_size": 2,
   "fc_hidden_nodes": 25,
   "dropout_after_conv": true,
   "conv_dropout_rate": 0.5,
   "dropout_after_fc1": false,
   "fc_dropout_rate": 0.5,
```

```
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
```

    Epoch 2/4: Train Acc: 81.39%, Test Acc: 84.06%
    Epoch 4/4: Train Acc: 82.58%, Test Acc: 85.13%
    √ Test Acc: 85.13%, Params: 13,565, Time: 114.9s

Experiment 35/50
Config: {

```
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 64,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
```

    Epoch 2/4: Train Acc: 82.22%, Test Acc: 83.98%
    Epoch 4/4: Train Acc: 82.70%, Test Acc: 84.01%
    √ Test Acc: 84.01%, Params: 26,474, Time: 115.3s

Experiment 36/50
Config: {

```
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 100,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
```

    Epoch 2/4: Train Acc: 80.71%, Test Acc: 83.08%
    Epoch 4/4: Train Acc: 82.12%, Test Acc: 83.53%
    √ Test Acc: 83.53%, Params: 38,390, Time: 117.0s

Experiment 37/50
Config: {

```
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 128,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
```

```
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 81.87%, Test Acc: 83.95%
    Epoch 4/4: Train Acc: 82.86%, Test Acc: 84.30%
    √  Test Acc: 84.30%, Params: 47,658, Time: 116.1s

Experiment 38/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.0,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 85.62%, Test Acc: 85.43%
    Epoch 4/4: Train Acc: 86.67%, Test Acc: 84.71%
    √  Test Acc: 84.71%, Params: 21,840, Time: 114.7s

Experiment 39/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.1,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 85.40%, Test Acc: 85.01%
    Epoch 4/4: Train Acc: 86.60%, Test Acc: 86.66%
    √  Test Acc: 86.66%, Params: 21,840, Time: 115.6s
Experiment 40/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.2,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
```

```
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 85.06%, Test Acc: 85.51%
    Epoch 4/4: Train Acc: 85.92%, Test Acc: 86.50%
    ✓ Test Acc: 86.50%, Params: 21,840, Time: 116.7s

Experiment 41/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.3,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 84.03%, Test Acc: 84.83%
    Epoch 4/4: Train Acc: 84.81%, Test Acc: 85.96%
    ✓ Test Acc: 85.96%, Params: 21,840, Time: 116.7s

Experiment 42/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.4,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 83.06%, Test Acc: 85.15%
    Epoch 4/4: Train Acc: 84.09%, Test Acc: 84.16%
    ✓ Test Acc: 84.16%, Params: 21,840, Time: 115.8s

Experiment 43/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.6,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
```

```
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 80.08%, Test Acc: 83.70%
    Epoch 4/4: Train Acc: 81.25%, Test Acc: 83.84%
    ✓ Test Acc: 83.84%, Params: 21,840, Time: 116.1s

Experiment 44/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.0,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 82.02%, Test Acc: 83.66%
    Epoch 4/4: Train Acc: 82.50%, Test Acc: 84.29%
    ✓ Test Acc: 84.29%, Params: 21,840, Time: 115.8s

Experiment 45/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.1,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 81.58%, Test Acc: 82.37%
    Epoch 4/4: Train Acc: 82.59%, Test Acc: 84.10%
    ✓ Test Acc: 84.10%, Params: 21,840, Time: 115.5s

Experiment 46/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.2,
    "activation": "relu",
```

```
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 82.18%, Test Acc: 84.65%
    Epoch 4/4: Train Acc: 82.84%, Test Acc: 83.96%
    ✓ Test Acc: 83.96%, Params: 21,840, Time: 115.5s

Experiment 47/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.3,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 81.79%, Test Acc: 84.28%
    Epoch 4/4: Train Acc: 82.94%, Test Acc: 83.74%
    ✓ Test Acc: 83.74%, Params: 21,840, Time: 114.8s

Experiment 48/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.4,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 81.49%, Test Acc: 82.89%
    Epoch 4/4: Train Acc: 82.26%, Test Acc: 82.61%
    ✓ Test Acc: 82.61%, Params: 21,840, Time: 115.0s

Experiment 49/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": false,
    "fc_dropout_rate": 0.6,
    "activation": "relu",
```

```
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 81.80%, Test Acc: 83.12%
    Epoch 4/4: Train Acc: 82.32%, Test Acc: 83.52%
    ✓  Test Acc: 83.52%, Params: 21,840, Time: 115.3s
Experiment 50/50
Config: {
    "num_conv_layers": 2,
    "conv_filters": [
        10,
        20
    ],
    "conv_kernel_size": [
        5,
        5
    ],
    "pool_positions": [
        true,
        true
    ],
    "pool_size": 2,
    "fc_hidden_nodes": 50,
    "dropout_after_conv": true,
    "conv_dropout_rate": 0.5,
    "dropout_after_fc1": true,
    "fc_dropout_rate": 0.5,
    "activation": "relu",
    "optimizer": "adam",
    "learning_rate": 0.01
}
    Epoch 2/4: Train Acc: 72.60%, Test Acc: 80.34%
    Epoch 4/4: Train Acc: 74.11%, Test Acc: 80.52%
    ✓  Test Acc: 80.52%, Params: 21,840, Time: 116.2s

Results saved to experiment_results.json

================================================================================
EXPERIMENT ANALYSIS
================================================================================

Total experiments: 50
Successful: 50
Failed: 0

--------------------------------------------------------------------------------
TOP 5 MODELS BY ACCURACY
--------------------------------------------------------------------------------

1. Test Accuracy: 88.97%
     Parameters: 102,080
     Training time: 107.5s
     Config: {
          "num_conv_layers": 1,
          "conv_filters": [
                12
          ],
          "conv_kernel_size": [
                3
          ],
          "pool_positions": [
                true
          ],
          "pool_size": 2,
          "fc_hidden_nodes": 50,
          "dropout_after_conv": true,
          "conv_dropout_rate": 0.5,
          "dropout_after_fc1": false,
          "fc_dropout_rate": 0.5,
          "activation": "relu",
          "optimizer": "adam",
          "learning_rate": 0.01
}

2. Test Accuracy: 88.96%
     Parameters: 135,920
     Training time: 112.8s
     Config: {
          "num_conv_layers": 1,
          "conv_filters": [
                16
          ],
          "conv_kernel_size": [
                3
          ],
          "pool_positions": [
                true
```

```
        ],
        "pool_size": 2,
        "fc_hidden_nodes": 50,
        "dropout_after_conv": true,
        "conv_dropout_rate": 0.5,
        "dropout_after_fc1": false,
        "fc_dropout_rate": 0.5,
        "activation": "relu",
        "optimizer": "adam",
        "learning_rate": 0.01
}

3. Test Accuracy: 88.77%
    Parameters: 68,240
    Training time: 95.6s
    Config: {
        "num_conv_layers": 1,
        "conv_filters": [
                8
        ],
        "conv_kernel_size": [
                3
        ],
        "pool_positions": [
                true
        ],
        "pool_size": 2,
        "fc_hidden_nodes": 50,
        "dropout_after_conv": true,
        "conv_dropout_rate": 0.5,
        "dropout_after_fc1": false,
        "fc_dropout_rate": 0.5,
        "activation": "relu",
        "optimizer": "adam",
        "learning_rate": 0.01
}

4. Test Accuracy: 88.53%
    Parameters: 169,760
    Training time: 124.2s
    Config: {
        "num_conv_layers": 1,
        "conv_filters": [
                20
        ],
        "conv_kernel_size": [
                3
        ],
        "pool_positions": [
                true
        ],
        "pool_size": 2,
        "fc_hidden_nodes": 50,
        "dropout_after_conv": true,
        "conv_dropout_rate": 0.5,
        "dropout_after_fc1": false,
        "fc_dropout_rate": 0.5,
        "activation": "relu",
        "optimizer": "adam",
        "learning_rate": 0.01
}

5. Test Accuracy: 88.09%
    Parameters: 271,280
    Training time: 149.9s
    Config: {
        "num_conv_layers": 1,
        "conv_filters": [
                32
        ],
        "conv_kernel_size": [
                3
        ],
        "pool_positions": [
                true
        ],
        "pool_size": 2,
        "fc_hidden_nodes": 50,
        "dropout_after_conv": true,
        "conv_dropout_rate": 0.5,
        "dropout_after_fc1": false,
        "fc_dropout_rate": 0.5,
        "activation": "relu",
        "optimizer": "adam",
        "learning_rate": 0.01
}
```

```
--------------------------------------------------------------------------------
TOP 5 FASTEST MODELS
--------------------------------------------------------------------------------

1. Training time: 95.6s
     Test Accuracy: 88.77%
     Parameters: 68,240

2. Training time: 98.9s
     Test Accuracy: 85.68%
     Parameters: 16,280

3. Training time: 99.2s
     Test Accuracy: 85.26%
     Parameters: 16,376

4. Training time: 99.4s
     Test Accuracy: 86.96%
     Parameters: 49,360

5. Training time: 99.6s
     Test Accuracy: 81.16%
     Parameters: 8,072

--------------------------------------------------------------------------------
TOP 5 MOST EFFICIENT MODELS (Accuracy/1K Parameters)
--------------------------------------------------------------------------------

1. Efficiency: 11.77
     Test Accuracy: 80.07%
     Parameters: 6,800

2. Efficiency: 10.05
     Test Accuracy: 81.16%
     Parameters: 8,072

3. Efficiency: 8.72
     Test Accuracy: 83.39%
     Parameters: 9,560

4. Efficiency: 8.61
     Test Accuracy: 83.17%
     Parameters: 9,656

5. Efficiency: 7.95
     Test Accuracy: 84.21%
     Parameters: 10,586

================================================================================
EXPERIMENTS COMPLETE!
================================================================================

Results saved to: experiment_results.json
Plots saved to: experiment_plots/
```

Surprisingly, the best performance was achieved with only one convolutional layer, which was unexpected. In addition, dropout turned out to be very important. This best model also suggests that the classic baseline configuration—activation: relu, optimizer: adam, and learning rate: 0.01—remains a strong default choice for experiments.
However, it is worth noting that my experiment did not exhaustively explore all possible parameter combinations in the linear search. I only tested the first 50 configurations. The entire run took about 2 hours to complete.

Extention

Replace the first layer of the MNIST network with a filter (tried sobel x, sobel y, laplacian, gaussian, 0° 45° 90° Gabor) and retrain the rest of the network, holding the first layer constant.

```
Project1>P5-use-filter-replace-v4.py
Creating filter visualizations...
Filter visualizations saved as filter_banks.png
```

```
============================================================
Running experiment with sobel_x filter
============================================================

Loaded pretrained model successfully

Train Epoch: 1 [0/60000 (0%)]	Loss: 17.247017

Test set: Average loss: 0.1736, Accuracy: 9430/10000 (94.30%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.329241

Test set: Average loss: 0.1347, Accuracy: 9581/10000 (95.81%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.365130

Test set: Average loss: 0.1240, Accuracy: 9632/10000 (96.32%)

Train Epoch: 4 [0/60000 (0%)]	Loss: 0.437689

Test set: Average loss: 0.1111, Accuracy: 9656/10000 (96.56%)

Model saved as mnist_model_sobel_x.pth

============================================================
Running experiment with sobel_y filter
============================================================

Loaded pretrained model successfully

Train Epoch: 1 [0/60000 (0%)]	Loss: 15.524207

Test set: Average loss: 0.1724, Accuracy: 9487/10000 (94.87%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.383744

Test set: Average loss: 0.1468, Accuracy: 9556/10000 (95.56%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.295650

Test set: Average loss: 0.1290, Accuracy: 9608/10000 (96.08%)

Train Epoch: 4 [0/60000 (0%)]	Loss: 0.365607

Test set: Average loss: 0.1231, Accuracy: 9615/10000 (96.15%)

Model saved as mnist_model_sobel_y.pth

============================================================
Running experiment with laplacian filter
============================================================

Loaded pretrained model successfully
Train Epoch: 1 [0/60000 (0%)]	Loss: 5.352652

Test set: Average loss: 0.1793, Accuracy: 9469/10000 (94.69%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.540626

Test set: Average loss: 0.1437, Accuracy: 9568/10000 (95.68%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.310482

Test set: Average loss: 0.1167, Accuracy: 9631/10000 (96.31%)

Train Epoch: 4 [0/60000 (0%)]	Loss: 0.273984

Test set: Average loss: 0.1026, Accuracy: 9669/10000 (96.69%)

Model saved as mnist_model_laplacian.pth

============================================================
Running experiment with gaussian filter
============================================================

Loaded pretrained model successfully

Train Epoch: 1 [0/60000 (0%)]	Loss: 4.095048

Test set: Average loss: 0.0971, Accuracy: 9707/10000 (97.07%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.232049

Test set: Average loss: 0.0762, Accuracy: 9757/10000 (97.57%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.286132
```

Test set: Average loss: 0.0668, Accuracy: 9787/10000 (97.87%)

Train Epoch: 4 [0/60000 (0%)] Loss: 0.201384

Test set: Average loss: 0.0693, Accuracy: 9775/10000 (97.75%)

Model saved as mnist_model_gaussian.pth

==
Running experiment with gabor_0 filter
==

Loaded pretrained model successfully
Train Epoch: 1 [0/60000 (0%)] Loss: 5.735612

Test set: Average loss: 0.1095, Accuracy: 9651/10000 (96.51%)

Train Epoch: 2 [0/60000 (0%)] Loss: 0.354052

Test set: Average loss: 0.0917, Accuracy: 9721/10000 (97.21%)

Train Epoch: 3 [0/60000 (0%)] Loss: 0.128770

Test set: Average loss: 0.0739, Accuracy: 9761/10000 (97.61%)

Train Epoch: 4 [0/60000 (0%)] Loss: 0.198270

Test set: Average loss: 0.0734, Accuracy: 9766/10000 (97.66%)

Model saved as mnist_model_gabor_0.pth

==
Running experiment with gabor_45 filter
==

Loaded pretrained model successfully

Train Epoch: 1 [0/60000 (0%)] Loss: 13.211176

Test set: Average loss: 0.1351, Accuracy: 9567/10000 (95.67%)

Train Epoch: 2 [0/60000 (0%)] Loss: 0.392681

Test set: Average loss: 0.1106, Accuracy: 9639/10000 (96.39%)

Train Epoch: 3 [0/60000 (0%)] Loss: 0.350604

Test set: Average loss: 0.0966, Accuracy: 9666/10000 (96.66%)

Train Epoch: 4 [0/60000 (0%)] Loss: 0.371549

Test set: Average loss: 0.0906, Accuracy: 9690/10000 (96.90%)

Model saved as mnist_model_gabor_45.pth

==
Running experiment with gabor_90 filter
==

Loaded pretrained model successfully

Train Epoch: 1 [0/60000 (0%)] Loss: 7.831794

Test set: Average loss: 0.1280, Accuracy: 9593/10000 (95.93%)

Train Epoch: 2 [0/60000 (0%)] Loss: 0.450442

Test set: Average loss: 0.1071, Accuracy: 9658/10000 (96.58%)

Train Epoch: 3 [0/60000 (0%)] Loss: 0.226562

Test set: Average loss: 0.0952, Accuracy: 9697/10000 (96.97%)

Train Epoch: 4 [0/60000 (0%)] Loss: 0.176759

Test set: Average loss: 0.0904, Accuracy: 9697/10000 (96.97%)

Model saved as mnist_model_gabor_90.pth

==
EXPERIMENT SUMMARY
==
sobel_x - Final Test Accuracy: 96.56%
sobel_y - Final Test Accuracy: 96.15%
laplacian - Final Test Accuracy: 96.69%
gaussian - Final Test Accuracy: 97.75%

```
gabor_0          - Final Test Accuracy: 97.66%
gabor_45         - Final Test Accuracy: 96.90%
gabor_90         - Final Test Accuracy: 96.97%

Comparison plot saved as experiments_comparison.png
```

Sobel X detects vertical edges, Sobel Y detects horizontal edges, Laplacian detects strong edge variations in all directions (second-order edges), Gaussian filter blurs/smooths the image and reduces noise, Gabor filters are designed to mimic the receptive fields of visual cortex cells.

Among all the filters, the Gaussian filter performed the best, followed by the 0° Gabor filter (which is sensitive to vertical textures). My hypothesis is that the Gaussian filter does not remove any important information — instead, it preserves the full signal and passes it to the next layer for learning.

The 0° Gabor filter outperforms Sobel X by a large margin, likely because Sobel X can only detect vertical edges. In contrast, the 0° Gabor filter can detect not only vertical edges but also vertical textures, stripes, and frequency patterns, making it a much more complex and informative visual feature detector.

3. Acknowledgements

Thanks to Professor Maxwell and the TA Sihe for your support. I also referred to the official OpenCV documentation, Stack Overflow discussions, and various online tutorials throughout the project.