Akshaj Raut 002337519

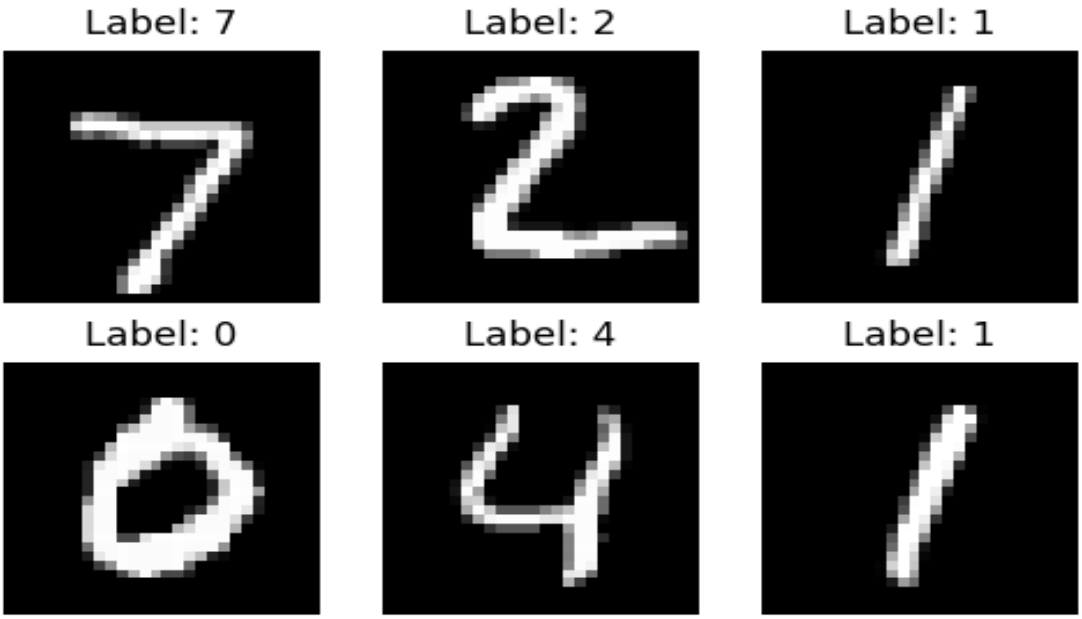
Project 5 Report: Recognition using Deep Networks

This project provides a comprehensive, hands-on exploration of deep learning by guiding the development and analysis of a convolutional neural network (CNN) for handwritten digit recognition. Using the MNIST dataset, the project begins with designing a network that incorporates convolutional layers, pooling, dropout, and fully connected layers to classify digits accurately. Once the basic model is built and trained, the project extends its scope by employing transfer learning to adapt the pre-trained network for classifying Greek letters. This adaptation involves freezing the original network weights and modifying the final layer to accommodate the new classes. Additionally, the project includes an experimental phase where various network architectures and hyperparameters, such as the number of filters, dropout rates, and dense layer sizes, are systematically varied to assess their impact on performance and training efficiency. Overall, this project not only deepens the understanding of how CNNs learn and extract features but also demonstrates the practical application of transfer learning and the importance of hyperparameter tuning in achieving optimal performance.

Task 1: Building and Training the Network

A. Data Acquisition and Visualization

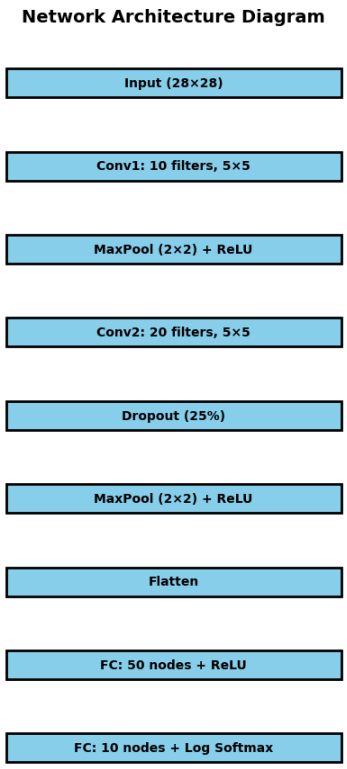
The MNIST dataset, containing 60,000 training images and 10,000 test images (each 28×28 pixels), was loaded directly from torchvision. To ensure consistency, the test set was preserved in its original order, and the first six images were visualized in a 2×3 grid using matplotlib. This step not only confirmed that the data was correctly loaded but also provided an initial glimpse into the digit samples.



B. Network Construction

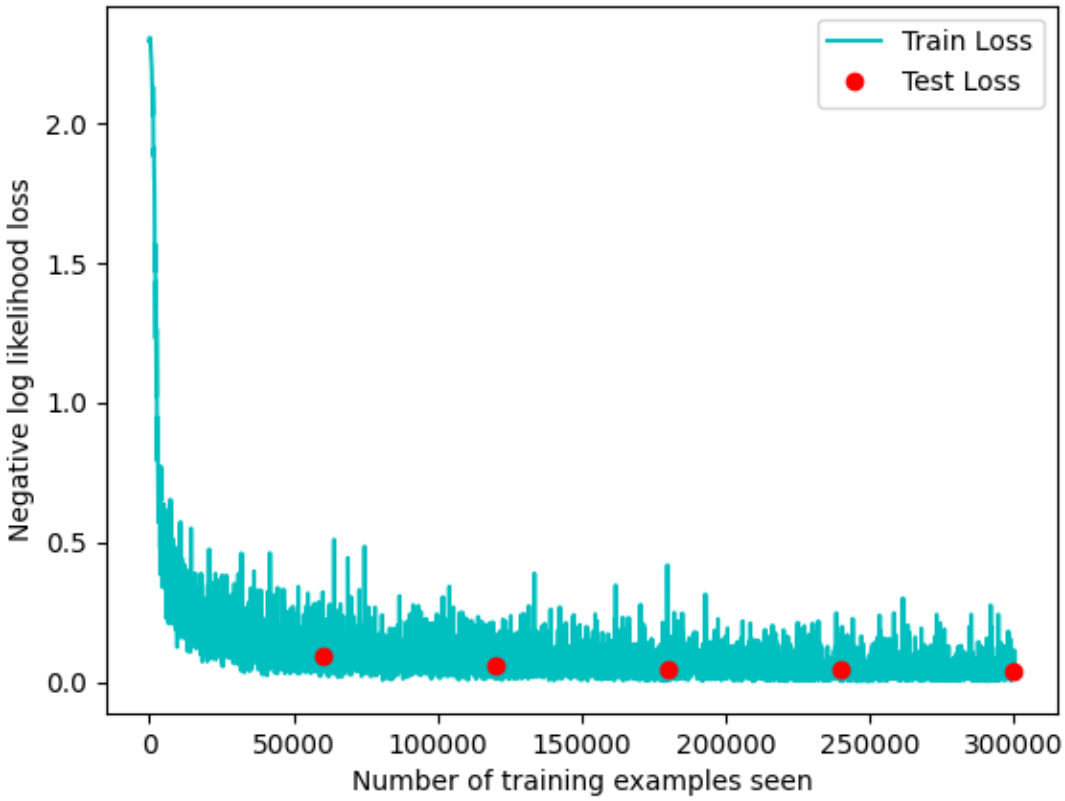
A convolutional network was designed with the following architecture:

- Layer 1: A convolutional layer with 10 filters (5×5) to capture basic image features.
- Pooling and Activation: A 2×2 max pooling layer followed by a ReLU activation to emphasize the most significant features.
- Layer 2: A second convolutional layer with 20 filters, followed by a dropout layer (with a dropout rate between 5% and 50%) to mitigate overfitting.
- Further Pooling: Another max pooling layer with ReLU activation.
- Flattening and Dense Layers: The output was flattened and fed into a fully connected layer with 50 neurons (using ReLU), followed by a final dense layer with 10 neurons and a log_softmax activation to yield log-probabilities for each digit class.



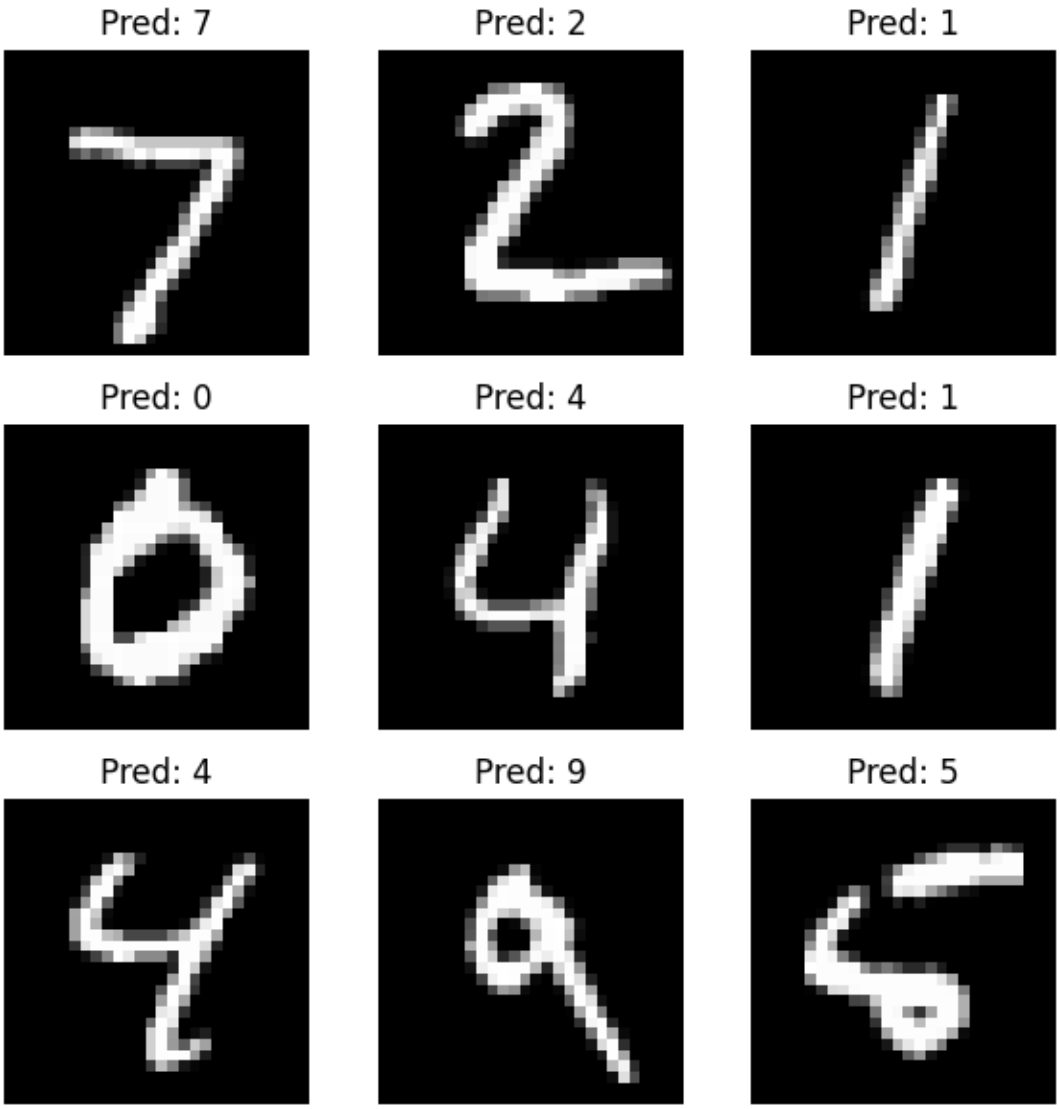
C. Training Process

The network was trained for 5 complete epochs using a batch size of 64. Throughout training, the negative log likelihood loss was computed, and the Adam optimizer was used to update the network's weights. Both training and test losses were recorded and plotted over time, providing valuable insights into the network's learning progress. The trained model was then saved for subsequent tasks.



D. Evaluation on the Test Set

A separate script was employed to load the trained network and evaluate it on the first 10 images from the test set. Prior to evaluation, the network was set to evaluation mode (using model.eval()) to ensure consistent dropout behavior. For each of these images, the script printed the 10 output probabilities (rounded to two decimal places), the predicted digit (the index with the highest probability), and the actual label. Additionally, a 3×3 grid displaying the first nine images with their predicted labels was generated, visually demonstrating the network's performance.

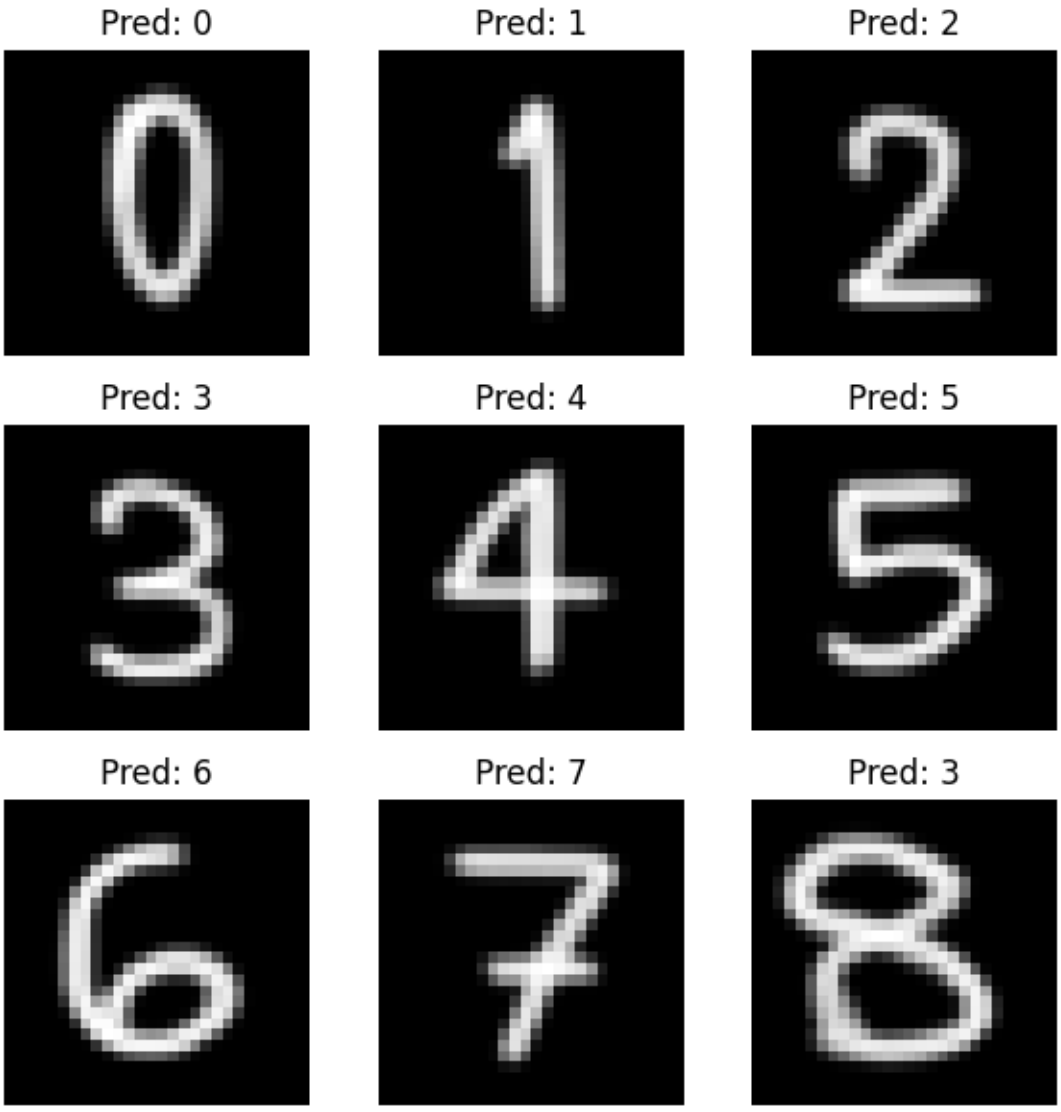



```
Running on device: mps
Image 0: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00']  Pred=7  True=7
Image 1: ['0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=2  True=2
Image 2: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=1  True=1
Image 3: ['1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=0  True=0
Image 4: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=4  True=4
Image 5: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=1  True=1
Image 6: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=4  True=4
Image 7: ['0.00', '0.00', '0.00', '0.00', '0.02', '0.00', '0.00', '0.00', '0.00', '0.98']  Pred=9  True=9
Image 8: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.99', '0.01', '0.00', '0.00', '0.00']  Pred=5  True=5
Image 9: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.99']  Pred=9  True=9
```

E. Testing with Custom Images

The network was further validated on a set of custom handwritten digits. These digits were created using thick strokes on a black background, then individually cropped, converted to grayscale, and resized to 28×28 pixels. The images were preprocessed to match the MNIST format (without color inversion since they were

already in the correct format) and then evaluated with the model. Both numerical predictions and a visual grid of the results were produced, confirming the network's ability to generalize to new inputs.

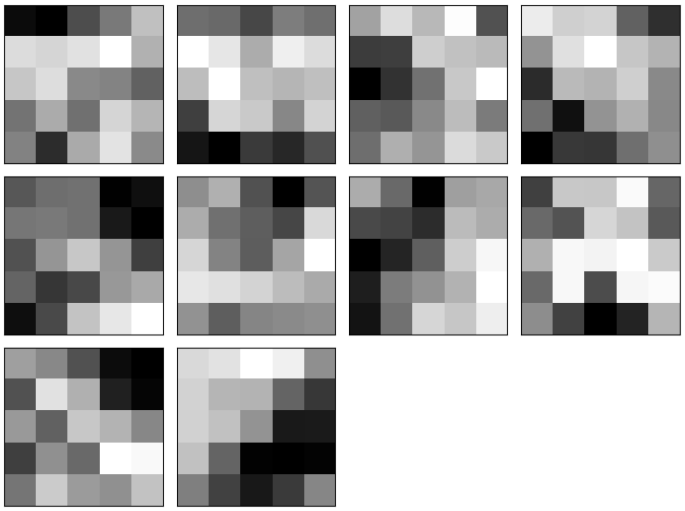



```
Running on device: mps
Image 0: [1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]  Predicted=0  True=0
Image 1: [0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]  Predicted=1  True=1
Image 2: [0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]  Predicted=2  True=2
Image 3: [0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00]  Predicted=3  True=3
Image 4: [0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00]  Predicted=4  True=4
Image 5: [0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00]  Predicted=5  True=5
Image 6: [0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00]  Predicted=6  True=6
Image 7: [0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00]  Predicted=7  True=7
Image 8: [0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00]  Predicted=3  True=8
Image 9: [0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00]  Predicted=9  True=9
```

Task 2: Analyzing the Network

A. Examining the First Convolutional Layer

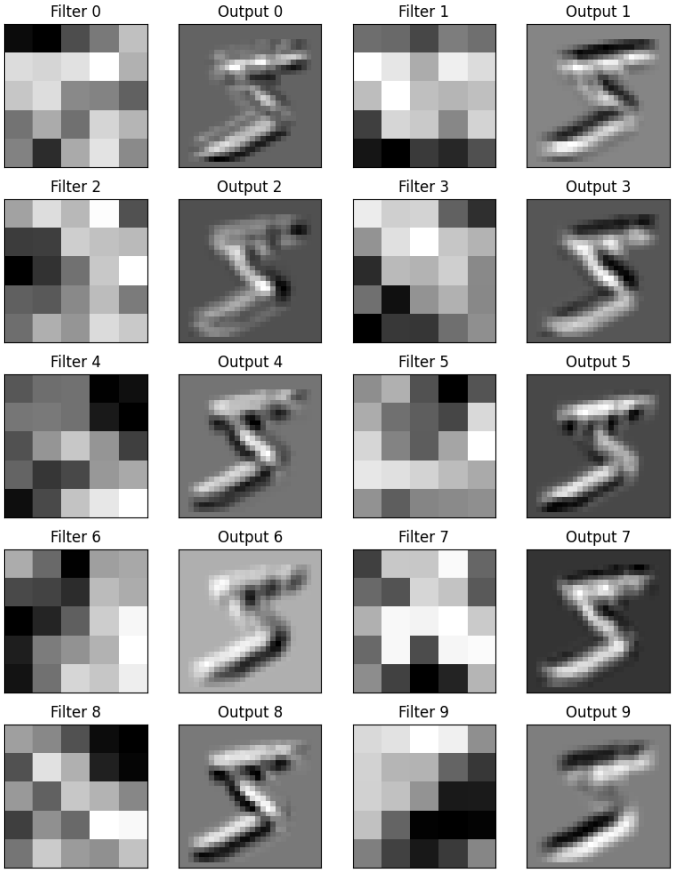
The focus of this task was to inspect the internal workings of the network, specifically the first convolutional layer (conv1). This layer consists of 10 filters of size 5×5, each designed to extract low-level features from the input images. The weights of these filters were extracted and printed, and their tensor shape ([10, 1, 5, 5]) was confirmed. These filters were visualized using matplotlib by arranging them in a 3×4 grid (with unused subplots disabled for clarity). Upon inspection, each filter exhibits a unique pattern—some appear to emphasize vertical or horizontal transitions akin to Sobel filters, while others seem to capture diagonal lines or corner-like features. This diversity suggests that the network is effectively learning to extract essential low-level features such as edges, textures, and strokes from the handwritten digit images.



B. Show the Effect of the Filters

To further understand how these filters process the data, we applied them to a sample MNIST digit image using OpenCV's filter2D function within a torch.no_grad() context, ensuring that no gradients were computed during

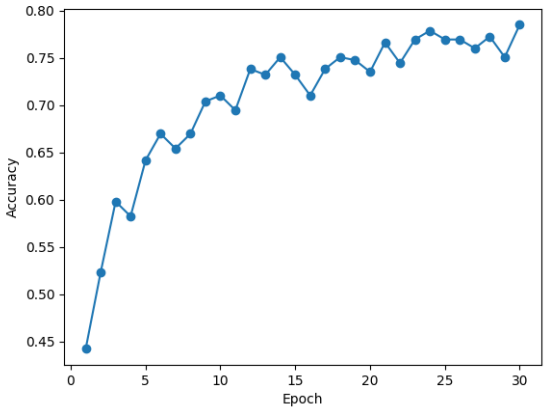
evaluation. Each filter produced a transformed output that accentuates specific characteristics of the digit. For example, filters resembling edge detectors highlighted the boundaries and sharp transitions in the image, while others emphasized more subtle features such as corners and internal textures. The filtered outputs were arranged in a grid to visually compare how each filter responds to the same input. The observed variations in the outputs validate our interpretation of the filters: different filters focus on different parts of the digit, collectively contributing to the network's ability to learn complex representations from simple, low-level features. Overall, these results confirm that the first convolutional layer plays a crucial role in extracting foundational features that enable the network to perform accurate digit classification.

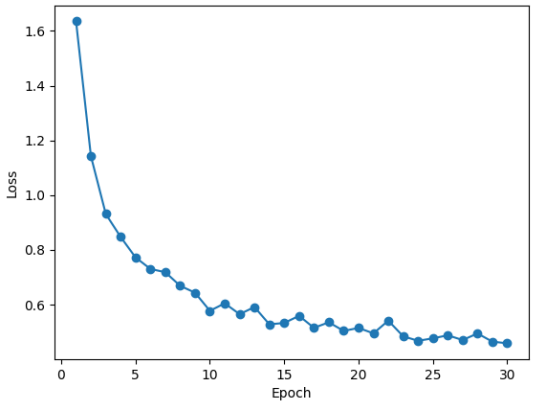


Task 3: Transfer Learning on Greek Letters

In this part of the project, the pre-trained MNIST digit classifier was adapted to recognize Greek letters (alpha, beta, and gamma). The network's pre-trained

weights were loaded and most layers were frozen, except for the final fully connected layer which was replaced to accommodate three output classes. The Greek letter images were preprocessed to match MNIST's format (grayscale, 28×28 pixels, and appropriate intensity inversion), and the model was fine-tuned on this new dataset. Training loss and accuracy were monitored and plotted, demonstrating that the adapted model gradually learned to distinguish between the Greek letters, despite the limited data.





Task 4: Design your own experiment

1. **Objective:**

- Investigate how varying key network hyperparameters affects both the performance (accuracy, loss) and training efficiency (time per epoch) of a convolutional neural network on the FashionMNIST dataset.

2. **Dimensions to Explore:**

- **Number of Filters in the First Convolutional Layer:**

Options: 10, 20, 40

Rationale: More filters may capture richer features, potentially improving accuracy, but they also increase computational cost.

- **Dropout Rate:**

Options: 0.1, 0.25, 0.5

Rationale: Dropout is used to prevent overfitting. Lower dropout might lead to overfitting, while too high a dropout might underfit.

- **Size of the Dense (Fully Connected) Layer:**

Options: 50, 100

Rationale: A larger dense layer could improve the network's ability to combine features, yet it may introduce more parameters without substantial benefits.

3. **Metrics to Record:**

- **Training Loss:** Average loss per epoch during training.
- **Test Loss:** Loss evaluated on the test set after each epoch.
- **Test Accuracy:** Overall accuracy on the test set.
- **Training Time:** Total training time per epoch or for the full training run.

1. **Number of Filters in the First Convolutional Layer:**

Hypothesis: Increasing the number of filters will enhance the network's ability to extract relevant features, resulting in improved test accuracy. However, a higher number of filters is also expected to increase training time due to the greater number of parameters.

2. **Dropout Rate:**

Hypothesis: A moderate dropout rate (around 0.25) will strike an optimal balance between regularization and learning capacity. A too-low dropout may not sufficiently prevent overfitting, whereas a too-high dropout might remove too much information, leading to underfitting and a drop in accuracy.

3. **Dense Layer Size:**

Hypothesis: Expanding the dense layer from 50 to 100 units should allow the network to capture more complex feature interactions, thereby slightly improving accuracy. However, the improvement is expected to be marginal relative to the increase in computational cost.

Experimental Execution and Results

The experiments were executed on the FashionMNIST dataset by systematically varying the three parameters. A grid search strategy was employed, exploring 18 different configurations as summarized below:

Filters=10:

With 0.1 dropout and 50 dense units: 88.55% accuracy (53.14 sec)

With 0.1 dropout and 100 dense units: 88.95% accuracy (52.92 sec)

With 0.25 dropout and 50 dense units: 88.62% accuracy (52.45 sec)

With 0.25 dropout and 100 dense units: 89.05% accuracy (52.64 sec)

With 0.5 dropout and 50 dense units: 88.22% accuracy (52.47 sec)

With 0.5 dropout and 100 dense units: 88.76% accuracy (52.28 sec)

Filters=20:

With 0.1 dropout and 50 dense units: 89.41% accuracy (57.11 sec)

With 0.1 dropout and 100 dense units: 89.98% accuracy (56.57 sec)

With 0.25 dropout and 50 dense units: 90.11% accuracy (56.83 sec)

With 0.25 dropout and 100 dense units: 90.53% accuracy (55.94 sec)

With 0.5 dropout and 50 dense units: 89.87% accuracy (55.39 sec)

With 0.5 dropout and 100 dense units: 90.01% accuracy (55.37 sec)

Filters=40:

With 0.1 dropout and 50 dense units: 90.61% accuracy (69.18 sec)

With 0.1 dropout and 100 dense units: 90.62% accuracy (67.02 sec)

With 0.25 dropout and 50 dense units: 90.86% accuracy (67.23 sec)

With 0.25 dropout and 100 dense units: 90.99% accuracy (67.67 sec)

With 0.5 dropout and 50 dense units: 90.41% accuracy (66.92 sec)

With 0.5 dropout and 100 dense units: 91.19% accuracy (66.55 sec)

Discussion and Analysis

- Impact of Increasing Filters:

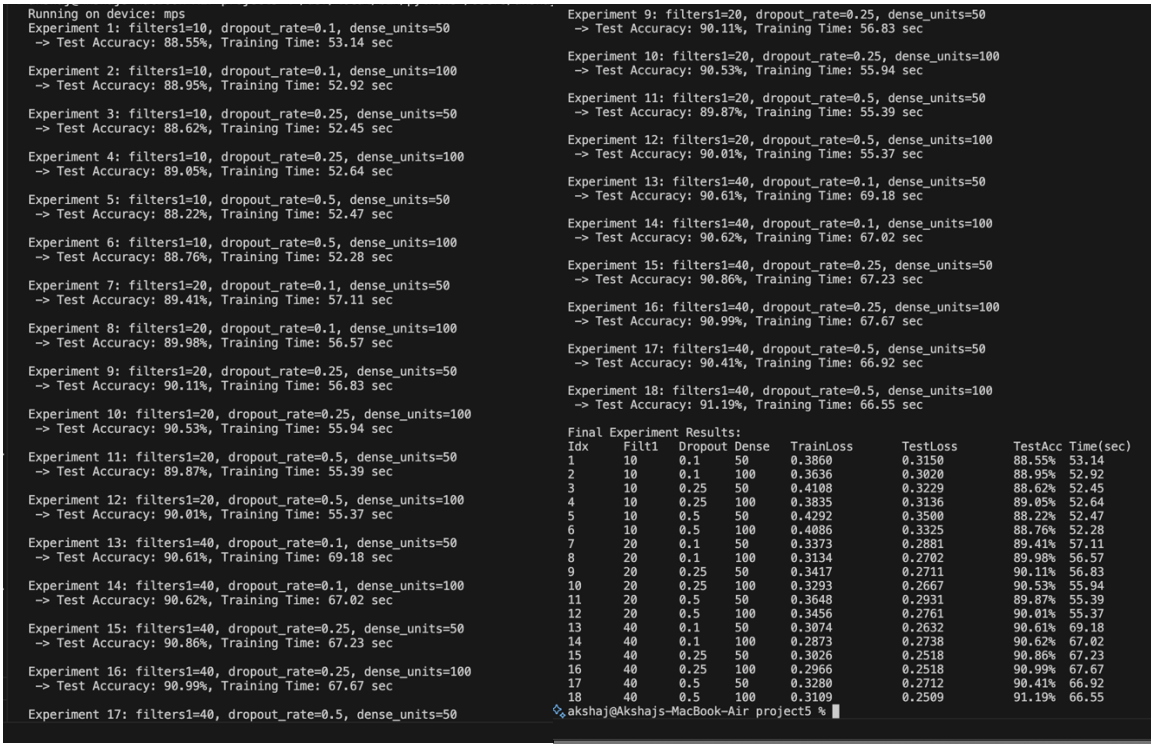
The results confirmed that increasing the number of filters improved accuracy. Models with 40 filters achieved the highest performance (up to 91.19%), supporting the hypothesis that more filters provide richer feature extraction. However, as expected, training time increased noticeably when moving from 10 or 20 filters to 40 filters.

- Optimal Dropout Rate:

For configurations with 10 and 20 filters, a dropout rate of 0.25 yielded slightly better performance compared to 0.1 or 0.5. Interestingly, for the largest model (40 filters), the highest accuracy (91.19%) was achieved with a dropout rate of 0.5 and 100 dense units. This suggests that the optimal dropout rate may depend on the overall capacity of the network: in larger models, a higher dropout rate might be necessary to counterbalance the increased risk of overfitting.

- Dense Layer Size:

Increasing the dense layer size from 50 to 100 units provided marginal improvements in accuracy across most configurations. This supports the hypothesis that a larger dense layer can capture more complex feature interactions, but the benefits are incremental and must be weighed against the additional computational cost.



```
Running on device: mps
Experiment 1: filters1=10, dropout_rate=0.1, dense_units=50
 -> Test Accuracy: 88.55%, Training Time: 53.14 sec

Experiment 2: filters1=10, dropout_rate=0.1, dense_units=100
 -> Test Accuracy: 88.95%, Training Time: 52.92 sec

Experiment 3: filters1=10, dropout_rate=0.25, dense_units=50
 -> Test Accuracy: 88.62%, Training Time: 52.45 sec

Experiment 4: filters1=10, dropout_rate=0.25, dense_units=100
 -> Test Accuracy: 89.05%, Training Time: 52.64 sec

Experiment 5: filters1=10, dropout_rate=0.5, dense_units=50
 -> Test Accuracy: 88.22%, Training Time: 52.47 sec

Experiment 6: filters1=10, dropout_rate=0.5, dense_units=100
 -> Test Accuracy: 88.76%, Training Time: 52.28 sec

Experiment 7: filters1=20, dropout_rate=0.1, dense_units=50
 -> Test Accuracy: 89.41%, Training Time: 57.11 sec

Experiment 8: filters1=20, dropout_rate=0.1, dense_units=100
 -> Test Accuracy: 89.98%, Training Time: 56.57 sec

Experiment 9: filters1=20, dropout_rate=0.25, dense_units=50
 -> Test Accuracy: 90.11%, Training Time: 56.83 sec

Experiment 10: filters1=20, dropout_rate=0.25, dense_units=100
 -> Test Accuracy: 90.53%, Training Time: 55.94 sec

Experiment 11: filters1=20, dropout_rate=0.5, dense_units=50
 -> Test Accuracy: 89.87%, Training Time: 55.39 sec

Experiment 12: filters1=20, dropout_rate=0.5, dense_units=100
 -> Test Accuracy: 90.01%, Training Time: 55.37 sec

Experiment 13: filters1=40, dropout_rate=0.1, dense_units=50
 -> Test Accuracy: 90.61%, Training Time: 69.18 sec

Experiment 14: filters1=40, dropout_rate=0.1, dense_units=100
 -> Test Accuracy: 90.62%, Training Time: 67.02 sec

Experiment 15: filters1=40, dropout_rate=0.25, dense_units=50
 -> Test Accuracy: 90.86%, Training Time: 67.23 sec

Experiment 16: filters1=40, dropout_rate=0.25, dense_units=100
 -> Test Accuracy: 90.99%, Training Time: 67.67 sec

Experiment 17: filters1=40, dropout_rate=0.5, dense_units=50
```

```
Experiment 9: filters1=20, dropout_rate=0.25, dense_units=50
 -> Test Accuracy: 90.11%, Training Time: 56.83 sec

Experiment 10: filters1=20, dropout_rate=0.25, dense_units=100
 -> Test Accuracy: 90.53%, Training Time: 55.94 sec

Experiment 11: filters1=20, dropout_rate=0.5, dense_units=50
 -> Test Accuracy: 89.87%, Training Time: 55.39 sec

Experiment 12: filters1=20, dropout_rate=0.5, dense_units=100
 -> Test Accuracy: 90.01%, Training Time: 55.37 sec

Experiment 13: filters1=40, dropout_rate=0.1, dense_units=50
 -> Test Accuracy: 90.61%, Training Time: 69.18 sec

Experiment 14: filters1=40, dropout_rate=0.1, dense_units=100
 -> Test Accuracy: 90.62%, Training Time: 67.02 sec

Experiment 15: filters1=40, dropout_rate=0.25, dense_units=50
 -> Test Accuracy: 90.86%, Training Time: 67.23 sec

Experiment 16: filters1=40, dropout_rate=0.25, dense_units=100
 -> Test Accuracy: 90.99%, Training Time: 67.67 sec

Experiment 17: filters1=40, dropout_rate=0.5, dense_units=50
 -> Test Accuracy: 90.41%, Training Time: 66.92 sec

Experiment 18: filters1=40, dropout_rate=0.5, dense_units=100
 -> Test Accuracy: 91.19%, Training Time: 66.55 sec

Final Experiment Results:
Idx     Filt1   Dropout Dense   TrainLoss       TestLoss        TestAcc Time(sec)
1       10      0.1     50      0.3860          0.3150          88.55%  53.14
2       10      0.1     100     0.3636          0.3020          88.95%  52.92
3       10      0.25    50      0.4108          0.3229          88.62%  52.45
4       10      0.25    100     0.3835          0.3136          89.05%  52.64
5       10      0.5     50      0.4292          0.3500          88.22%  52.47
6       10      0.5     100     0.4086          0.3325          88.76%  52.28
7       20      0.1     50      0.3373          0.2881          89.41%  57.11
8       20      0.1     100     0.3134          0.2702          89.98%  56.57
9       20      0.25    50      0.3417          0.2711          90.11%  56.83
10      20      0.25    100     0.3293          0.2667          90.53%  55.94
11      20      0.5     50      0.3648          0.2931          89.87%  55.39
12      20      0.5     100     0.3456          0.2761          90.01%  55.37
13      40      0.1     50      0.3074          0.2632          90.61%  69.18
14      40      0.1     100     0.2873          0.2738          90.62%  67.02
15      40      0.25    50      0.3026          0.2518          90.86%  67.23
16      40      0.25    100     0.2966          0.2518          90.99%  67.67
17      40      0.5     50      0.3280          0.2712          90.41%  66.92
18      40      0.5     100     0.3109          0.2509          91.19%  66.55
akshaj@Akshajs-MacBook-Air project5 %
```

Conclusion:

Overall, the experiments largely supported the initial hypotheses:

- Increasing the number of filters improved test accuracy but also increased training time.
- A moderate dropout rate was effective for smaller networks, while a higher dropout rate proved beneficial in the context of a larger model.
- Enlarging the dense layer led to slight performance gains, though these improvements were modest.

The best-performing configuration was achieved with 40 filters, a dropout rate of 0.5, and 100 dense units, resulting in a test accuracy of 91.19% with a training time of approximately 66.55 seconds. This outcome validates the idea that model capacity and careful regularization can enhance performance, though they do incur a computational cost.

Reflection

This project offered a practical, hands-on exploration of deep learning, beginning with constructing and training a convolutional neural network for digit recognition and extending it through transfer learning and various experiments. It provided valuable insights into how each layer contributes to feature extraction and decision-making, and demonstrated that even minor adjustments in network architecture or training parameters can have a significant impact on performance. The transfer learning component was particularly enlightening, as it underscored the advantages of using pre-trained models for tackling new tasks with limited data. Overall, this project not only enhanced my proficiency with PyTorch and CNNs but also sparked a deeper curiosity about advanced applications in computer vision.

Acknowledgements

I would like to extend my sincere thanks to my instructor for their unwavering guidance and support throughout this project. I also appreciate the wealth of resources and tutorials provided, which were instrumental in deepening my understanding of deep learning and PyTorch. Additionally, I am especially grateful to Atharva Nayak for his invaluable assistance and insights that helped me tremendously in successfully completing this project.