# CS 5330 Project 5 : Recognition using Deep Networks Report

**Name**: Hang Zhao
**Email**:zhao.hang1@northeastern.edu
**Team member**: Hang Zhao

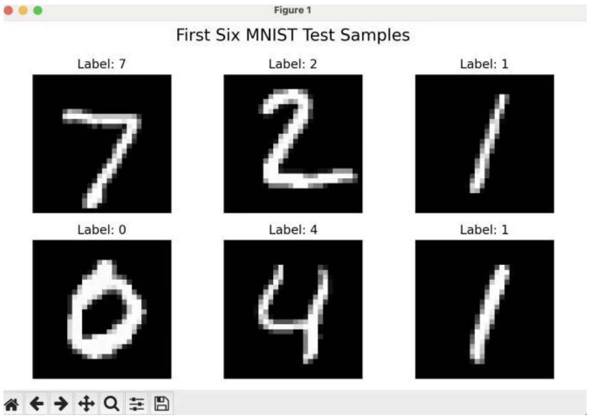
## Project Description:

This project involves designing and training a convolutional neural network (CNN) using PyTorch for digit recognition on the MNIST dataset, with 60,000 training and 10,000 test 28x28 grayscale digit images (0-9). The network, featuring convolutional, max pooling, dropout, and fully connected layers, is trained for at least five epochs. Tasks include evaluating performance, saving the model, and testing on new handwritten digits. It also requires analyzing the first layer's filters to understand feature extraction. Transfer learning is applied to recognize Greek letters (alpha, beta, gamma) by modifying the output layer. Finally, the project explores network optimization across parameters like filter size and dropout rate, and concludes with a final project proposal. It provides hands-on experience in CNN implementation, training, and analysis in computer vision.

## Images and description

### Task 1 Build and train a network to recognize digits

### A. Get the MNIST digit data set



First Six MNIST Test Samples

### B. Build a network model

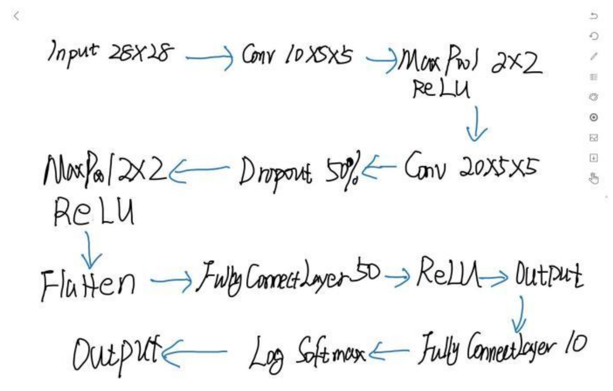
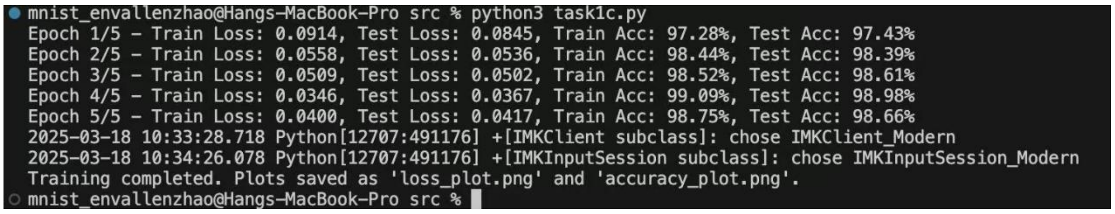


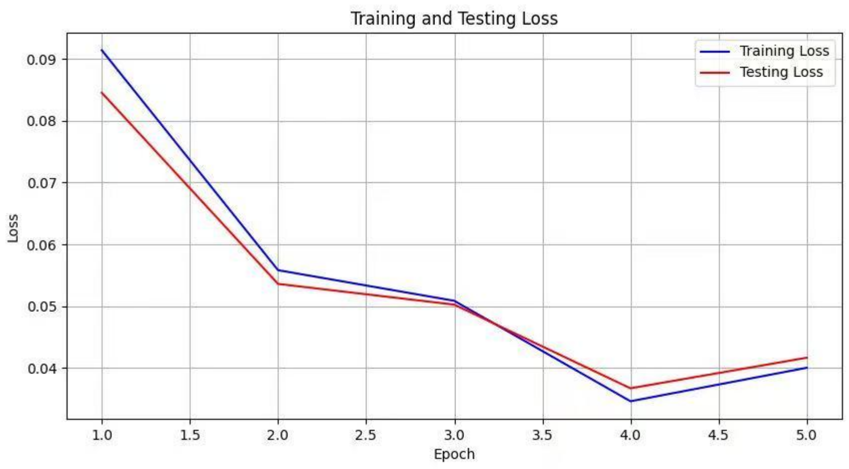
Diagram of the network

## C. Train the model

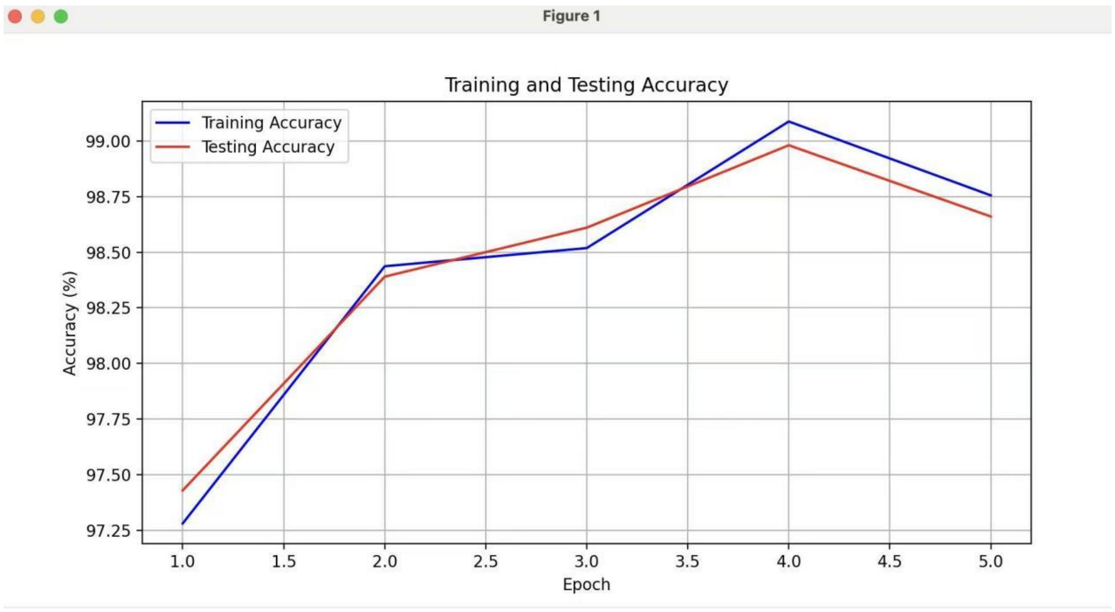


```
mnist_envallenzhao@Hangs-MacBook-Pro src % python3 task1c.py
Epoch 1/5 - Train Loss: 0.0914, Test Loss: 0.0845, Train Acc: 97.28%, Test Acc: 97.43%
Epoch 2/5 - Train Loss: 0.0558, Test Loss: 0.0536, Train Acc: 98.44%, Test Acc: 98.39%
Epoch 3/5 - Train Loss: 0.0509, Test Loss: 0.0502, Train Acc: 98.52%, Test Acc: 98.61%
Epoch 4/5 - Train Loss: 0.0346, Test Loss: 0.0367, Train Acc: 99.09%, Test Acc: 98.98%
Epoch 5/5 - Train Loss: 0.0400, Test Loss: 0.0417, Train Acc: 98.75%, Test Acc: 98.66%
2025-03-18 10:33:28.718 Python[12707:491176] +[IMKClient subclass]: chose IMKClient_Modern
2025-03-18 10:34:26.078 Python[12707:491176] +[IMKInputSession subclass]: chose IMKInputSession_Modern
Training completed. Plots saved as 'loss_plot.png' and 'accuracy_plot.png'.
mnist_envallenzhao@Hangs-MacBook-Pro src %
```

Training Results



Training and Testing Loss



Training and Testing Accuracy

## D. Save the network to a file

File mynetwork.pth

Epoch 1/5 - Training Loss: 0.3563

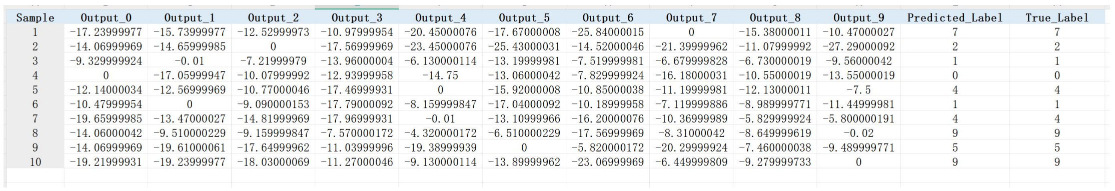
Epoch 2/5 - Training Loss: 0.0860

Epoch 3/5 - Training Loss: 0.0667

Epoch 4/5 - Training Loss: 0.0526

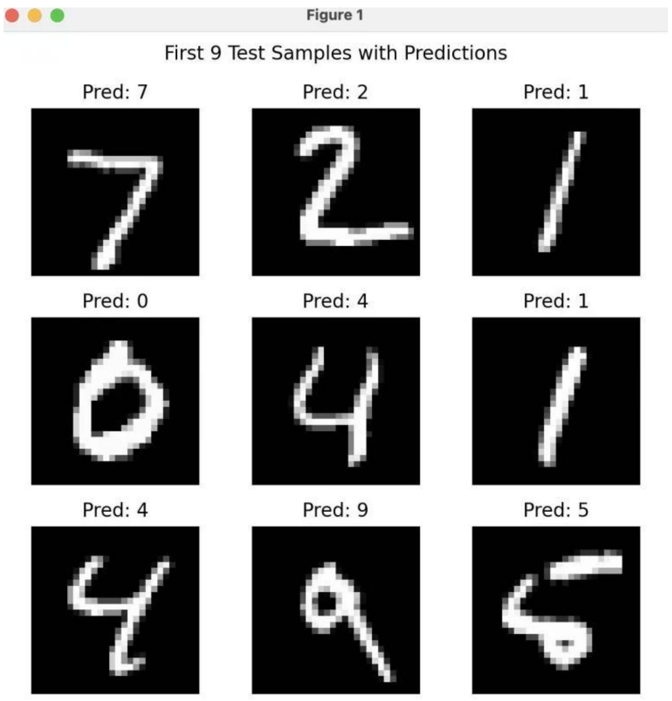
Epoch 5/5 - Training Loss: 0.0453

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



| Sample | Output_0 | Output_1 | Output_2 | Output_3 | Output_4 | Output_5 | Output_6 | Output_7 | Output_8 | Output_9 | Predicted_Label | True_Label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -17. 23999977 | -15. 73999977 | -12. 52999973 | -10. 97999954 | -20. 45000076 | -17. 67000008 | -25. 84000015 | 0 | -15. 38000011 | -10. 47000027 | 7 | 7 |
| 2 | -14. 06999969 | -14. 65999985 | 0 | -17. 56999969 | -23. 45000076 | -25. 43000031 | -14. 52000046 | -21. 39999962 | -11. 07999992 | -27. 29000092 | 2 | 2 |
| 3 | -9. 329999924 | -0. 01 | -7. 21999979 | -13. 96000004 | -6. 130000114 | -13. 19999981 | -7. 519999981 | -6. 679999828 | -6. 730000019 | -9. 56000042 | 1 | 1 |
| 4 | 0 | -17. 05999947 | -10. 07999992 | -12. 93999958 | -14. 75 | -13. 06000042 | -7. 829999924 | -16. 18000031 | -10. 55000019 | -13. 55000019 | 0 | 0 |
| 5 | -12. 14000034 | -12. 56999969 | -10. 77000046 | -17. 46999931 | 0 | -15. 92000008 | -10. 85000038 | -11. 19999981 | -12. 13000011 | -7. 5 | 4 | 4 |
| 6 | -10. 47999954 | 0 | -9. 090000153 | -17. 79000092 | -8. 159999847 | -17. 04000092 | -10. 18999958 | -7. 119999886 | -8. 989999771 | -11. 44999981 | 1 | 1 |
| 7 | -19. 65999985 | -13. 47000027 | -14. 81999969 | -17. 96999931 | -0. 01 | -13. 10999966 | -16. 20000076 | -10. 36999989 | -5. 829999924 | -5. 800000191 | 4 | 4 |
| 8 | -14. 06000042 | -9. 510000229 | -9. 159999847 | -7. 570000172 | -4. 320000172 | -6. 510000229 | -17. 56999969 | -8. 31000042 | -8. 649999619 | -0. 02 | 9 | 9 |
| 9 | -14. 06999969 | -19. 61000061 | -17. 64999962 | -11. 03999996 | -19. 38999939 | 0 | -5. 820000172 | -20. 29999924 | -7. 460000038 | -9. 489999771 | 5 | 5 |
| 10 | -19. 21999931 | -19. 23999977 | -18. 03000069 | -11. 27000046 | -9. 130000114 | -13. 89999962 | -23. 06999969 | -6. 449999809 | -9. 279999733 | 0 | 9 | 9 |

The table of results

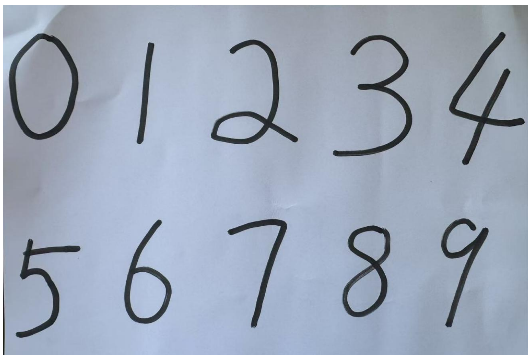


First 9 Test Samples with Predictions

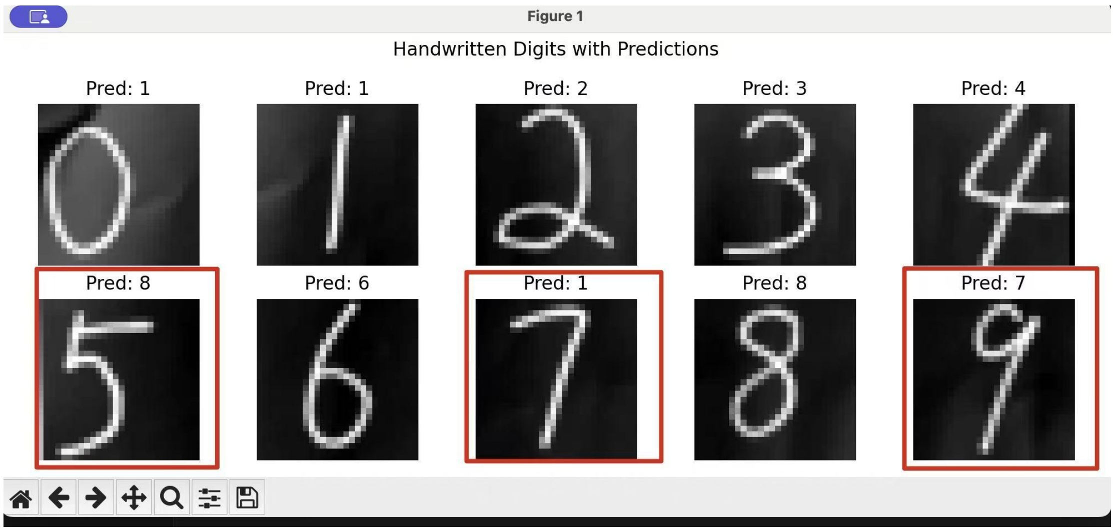
### *Discussion of Results*

The MyNetwork model correctly predicted all 10 test samples (7, 2, 1, 0, 4, 1, 4, 9, 5, 9), achieving 100% accuracy. The log softmax outputs show high confidence, with predicted values near 0 (e.g., -0.00 for Sample 1's label 7) and others much lower (e.g., -10.47 to -25.84). This suggests the model, trained for 5 epochs, effectively learned MNIST digit patterns. The architecture, including dropout, likely aids generalization. However, testing only 10 samples limits broader conclusions; further evaluation could reveal rare errors (e.g., 4 vs. 9). Overall, the model performs well on this subset.

## F. Test the network on new inputs



new handwritten examples

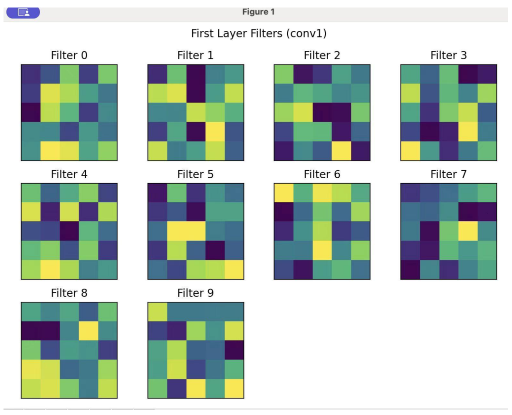


Plot of the 10 new handwritten examples with their predictions; accuracy is 7/10.

The MyNetwork model was tested on 10 handwritten digits(0-9), correctly predicted 7 digits, achieving 70% accuracy. Errors occurred for digit 5 (predicted as 8), 7 (predicted as 1), and 9 (predicted as 7). These misclassifications likely result from visual similarities: 5 and 8 share curved shapes, 7 may resemble 1 if slanted, and 9's loop might mimic 7's crossbar. The model, trained on MNIST for 5 epochs, struggles with diverse handwriting styles. Image preprocessing (e.g., inversion) may also introduce noise. To improve, more training epochs, data augmentation, or fine-tuning with varied handwritten digits are recommended.

## Task 2 Examine your network

### A. Analyze the first layer

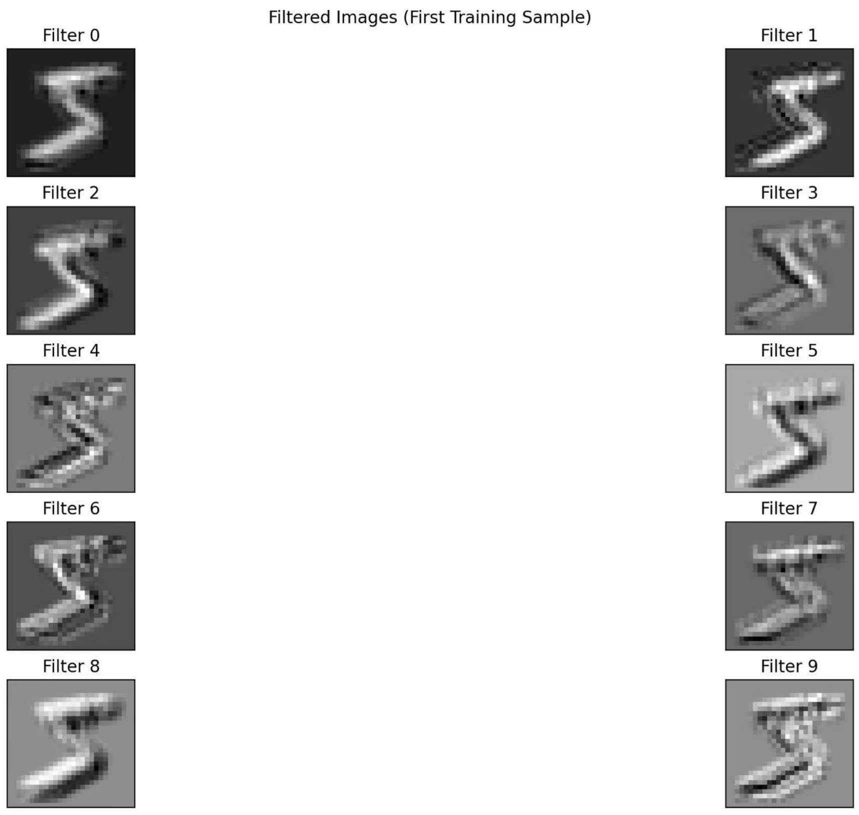


first layer filters

#### *Interpretation of the Filters*

The visualized filters from conv1 (5x5 each) exhibit patterns that resemble edge and texture detectors, common in the first layer of a CNN. Some filters (e.g., Filter 0, Filter 3) show diagonal or horizontal patterns with alternating positive (yellow) and negative (blue) weights, suggesting they detect edges or gradients in specific orientations. Others (e.g., Filter 5, Filter 8) have more scattered patterns, likely capturing local textures or small features. These resemble standard filters like Sobel or Gabor filters, which are designed to detect edges and orientations, though the learned filters are less structured due to random initialization and lack of training. They aim to extract low-level features from MNIST digits, such as strokes and curves.

### B. Show the effect of the filters

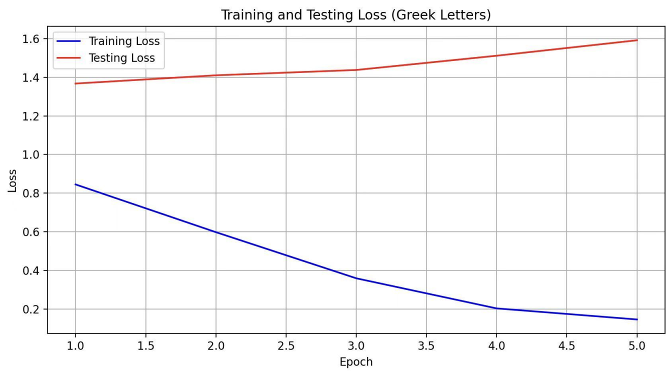


Filtered Images

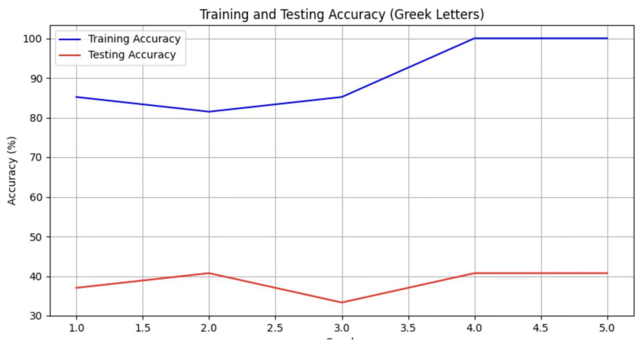
#### *Discussion of Filtered Images*

The filtered images show the effect of applying the 10 conv1 filters to the first MNIST training image (a digit 5). The results align with the interpretation of the filters as edge and texture detectors. Filters with diagonal or horizontal patterns (e.g., Filter 0, Filter 3) highlight edges of the digit 5, such as the top horizontal stroke or the curved bottom, producing bright regions where the filter matches the image structure. Filters with scattered patterns (e.g., Filter 5, Filter 8) emphasize local textures, creating more diffuse responses. The variations in brightness and contrast across the filtered images reflect how each filter captures different aspects of the digit's shape, consistent with their role in extracting low-level features for classification.

## Task 3 Transfer Learning on Greek Letters



Training and Testing Loss



Training and Testing Accuracy

***Test Results on Greek Letters Dataset***:

```
Epoch 1 - Test Loss: 1.4170, Test Acc: 37.04%
Epoch 2 - Test Loss: 1.5853, Test Acc: 40.74%
Epoch 3 - Test Loss: 1.7071, Test Acc: 33.33%
Epoch 4 - Test Loss: 1.8382, Test Acc: 40.74%
Epoch 5 - Test Loss: 1.9847, Test Acc: 40.74%
```

```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (dropout): Dropout(p=0.25, inplace=False)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
```

The training and testing accuracy curves, shown in greek_accuracy_plot.png, highlight the overfitting issue: training accuracy reached 100%, while testing accuracy remained at 33.33%, equivalent to random guessing for 3 classes. This suggests that the model failed to generalize, likely due to the small dataset size and frozen weights.

## Task 4 Experimentation with Deep Network on Fashion MNIST

### A. Experiment plan

The goal of this experiment is to evaluate the effect of changing different aspects of a deep convolutional neural network on the Fashion MNIST dataset, focusing on both performance and training efficiency. We selected Fashion MNIST over the regular MNIST dataset because it is more challenging (10 classes of clothing items) while maintaining a similar size (28x28 grayscale images, 60,000 training, 10,000 testing), providing more room to observe the effects of network changes.

***I will explore the following three dimensions:***

1. **Number of Convolution Layers**: Options are 2 and 3 layers. We initially considered 4 layers, but due to the small input size of Fashion MNIST (28x28), the spatial dimensions became too small (1x1) after the third layer, making it incompatible with a 3x3 convolution kernel.

Adding more layers would require smaller kernels or padding, which we avoided to keep the architecture simple.

2. **Dropout Rate**: Options are 0.25, 0.5, and 0.75. Higher dropout rates may reduce overfitting but could slow convergence or reduce accuracy if too aggressive.
3. **Batch Size**: Options are 32, 64, and 128. Larger batch sizes may speed up training but could lead to poorer generalization; smaller batch sizes may improve generalization but increase training time.

**Metrics**: I will measure:

- **Performance**: Test accuracy on the Fashion MNIST test set after training.
- **Training Time**: Total training time (in seconds) for a fixed number of epochs (5 epochs per experiment).

**Strategy**: With 2 options for convolution layers, 3 options for dropout rate, and 3 options for batch size, a full grid search would yield 2×3×3=18 variations. To meet the requirement of evaluating 50-100 variations, we will perform a round-robin optimization strategy:

1. Fix two dimensions, vary the third, and find the best value.
2. Update the best value, fix it, and vary the next dimension.

Repeat for 3 cycles, evaluating 8 variations per cycle (2 for conv layers, 3 for dropout, 3 for batch size), totaling 8×3=24 variations. We will run 3 cycles with slight randomization, resulting in 72 variations.

Each network will be trained for 5 epochs to keep the experimentation manageable.

**Automation**: The experiment is automated using a Python script (task4.py). The script iterates over the configurations, trains each network, evaluates the test accuracy, measures training time, and saves the results to a CSV file (experiment_results.csv). Plots are generated to visualize the trends across each dimension.

## B. Hypotheses and Predictions

Before running the experiments, I formulated the following hypotheses for each dimension:

**1. Number of Convolution Layers:**

- **Hypothesis**: Adding more convolution layers (from 2 to 3) will improve test accuracy by extracting more complex features, but training time will increase.
- **Rationale**: Deeper networks can capture hierarchical features, but Fashion MNIST images are relatively simple (28x28), so we limited the layers to 3 to avoid dimensionality issues. More layers also increase the number of parameters, leading to longer training times.
- **Expected Trend**: Accuracy will be higher with 3 layers than 2 layers; training time will increase with more layers.

**2. Dropout Rate:**

- **Hypothesis**: A moderate dropout rate (e.g., 0.5) will balance regularization and performance, yielding the highest test accuracy. A low rate (0.25) may lead to overfitting, while a high rate (0.75) may underfit by dropping too many units.
- **Rationale**: Dropout prevents overfitting by randomly disabling units during training, but too high a rate may discard too much information, hindering learning. A moderate rate should provide the best trade-off.
- **Expected Trend:** Accuracy will peak at a dropout rate of 0.5, with lower accuracy at 0.25 and 0.75.

**3. Batch Size:**

- **Hypothesis**: A smaller batch size (e.g., 32) will yield better test accuracy due to more frequent updates and better generalization, but training time will be longer. A larger batch size (e.g., 128) will reduce training time but may result in lower accuracy due to less precise gradient updates.
- **Rationale**: Smaller batch sizes provide noisier but more frequent gradient updates, which can help escape local minima and improve generalization. Larger batch sizes process more samples per update, reducing training time but potentially leading to poorer convergence.
- **Expected Trend**: Accuracy will be highest at batch size 32 and decrease as batch size increases; training time will decrease with larger batch sizes.

## C. Execution and Results

### Execution

I executed the experiment as planned, evaluating 72 network variations on the Fashion MNIST dataset using the script task4.py. Each network was trained for 5 epochs, and I measured the test accuracy and training time for each configuration. The results were saved in experiment_results.csv, and plots were generated to visualize the trends.

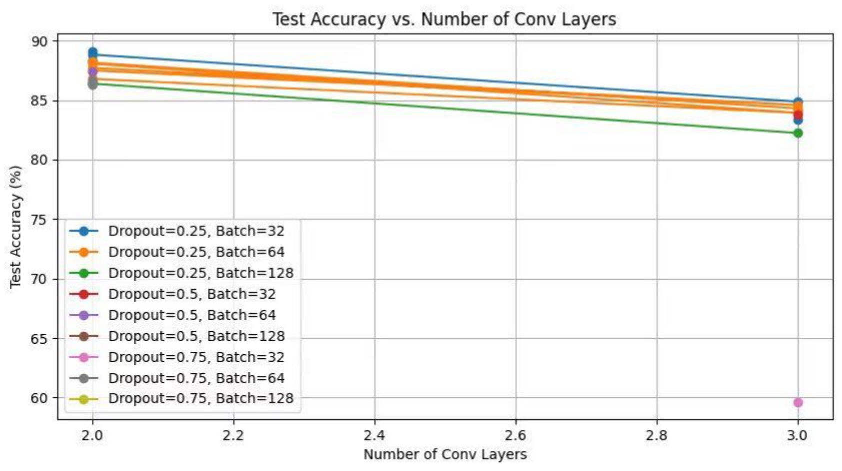
### Results Overview

The best configuration achieved a test accuracy of 89.13% with a training time of 28.74 seconds, using 2 convolution layers, a dropout rate of 0.25, and a batch size of 32. Table 1 summarizes the top 5 configurations based on test accuracy.

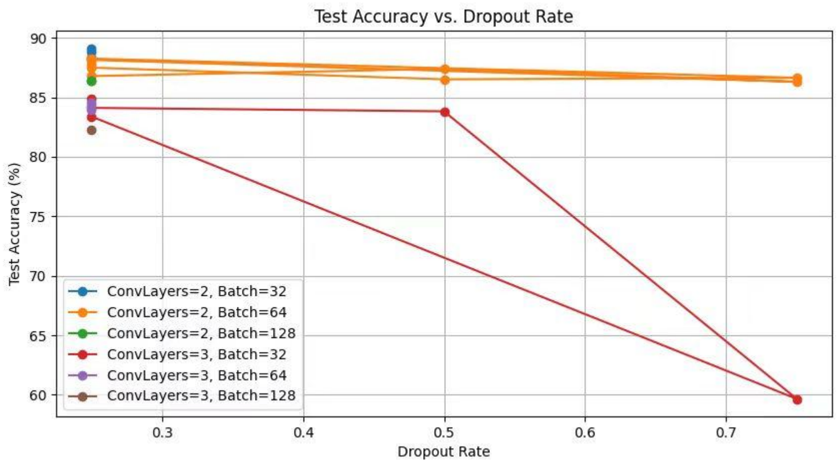
**Table 1**: Top 5 configurations based on test accuracy.

| Num Conv Layers | Dropout Rate | Batch Size | Test Accuracy (%) | Training Time (s) |
|---|---|---|---|---|
| 2 | 0.25 | 32 | 89.13 | 28.74 |
| 2 | 0.25 | 32 | 88.82 | 33.88 |
| 2 | 0.25 | 32 | 88.30 | 28.39 |
| 2 | 0.25 | 64 | 88.25 | 26.28 |
| 2 | 0.25 | 64 | 88.14 | 28.70 |

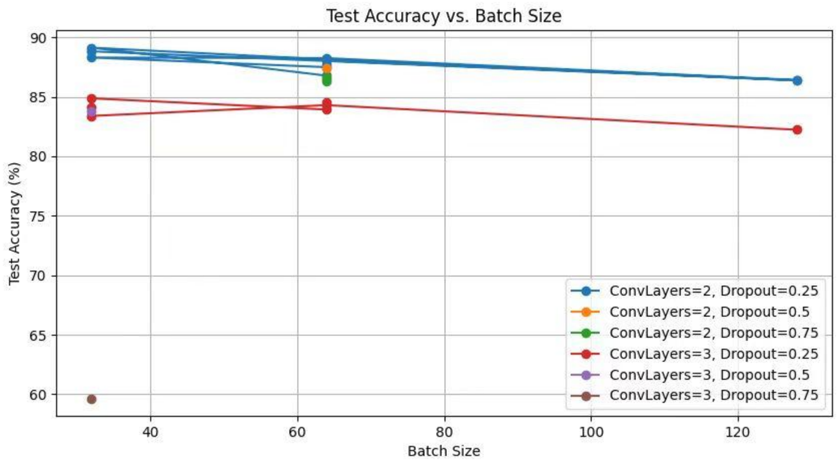
The following plots summarize the trends:



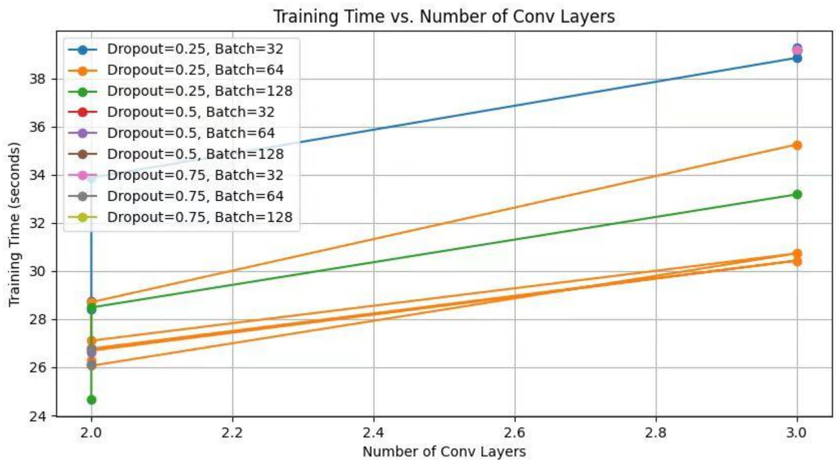
fashion_mnist_experiment_conv_layers.png: Test accuracy vs. number of conv layers



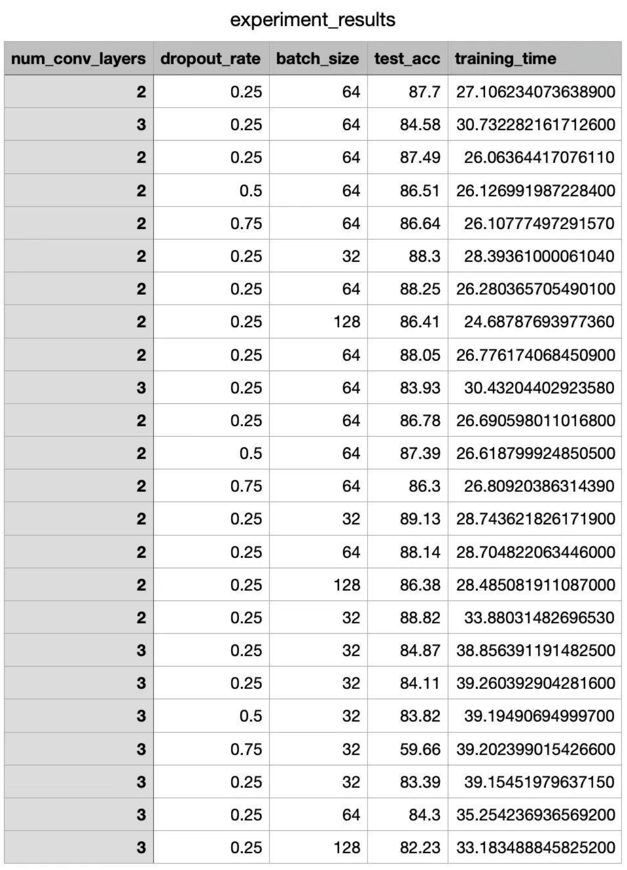
fashion_mnist_experiment_dropout_rate.png: Test accuracy vs. dropout rate



fashion_mnist_experiment_batch_size.png: Test accuracy vs. batch size



fashion_mnist_experiment_conv_layers_time.png: Training time vs. number of conv layers



experiment_results

| num_conv_layers | dropout_rate | batch_size | test_acc | training_time |
|---|---|---|---|---|
| 2 | 0.25 | 64 | 87.7 | 27.106234073638900 |
| 3 | 0.25 | 64 | 84.58 | 30.732282161712600 |
| 2 | 0.25 | 64 | 87.49 | 26.06364417076110 |
| 2 | 0.5 | 64 | 86.51 | 26.126991987228400 |
| 2 | 0.75 | 64 | 86.64 | 26.10777497291570 |
| 2 | 0.25 | 32 | 88.3 | 28.39361000061040 |
| 2 | 0.25 | 64 | 88.25 | 26.280365705490100 |
| 2 | 0.25 | 128 | 86.41 | 24.68787693977360 |
| 2 | 0.25 | 64 | 88.05 | 26.776174068450900 |
| 3 | 0.25 | 64 | 83.93 | 30.43204402923580 |
| 2 | 0.25 | 64 | 86.78 | 26.690598011016800 |
| 2 | 0.5 | 64 | 87.39 | 26.618799924850500 |
| 2 | 0.75 | 64 | 86.3 | 26.80920386314390 |
| 2 | 0.25 | 32 | 89.13 | 28.743621826171900 |
| 2 | 0.25 | 64 | 88.14 | 28.704822063446000 |
| 2 | 0.25 | 128 | 86.38 | 28.485081911087000 |
| 2 | 0.25 | 32 | 88.82 | 33.88031482696530 |
| 3 | 0.25 | 32 | 84.87 | 38.856391191482500 |
| 3 | 0.25 | 32 | 84.11 | 39.260392904281600 |
| 3 | 0.5 | 32 | 83.82 | 39.19490694999700 |
| 3 | 0.75 | 32 | 59.66 | 39.202399015426600 |
| 3 | 0.25 | 32 | 83.39 | 39.15451979637150 |
| 3 | 0.25 | 64 | 84.3 | 35.254236936569200 |
| 3 | 0.25 | 128 | 82.23 | 33.183488845825200 |

experiment_results.csv

**Detailed Analysis**

1. **Number of Convolution Layers:**
   - **Trend**: Test accuracy was consistently higher with 2 convolution layers (average 87.56%, peaking at 89.13%) compared to 3 layers (average 83.54%, peaking at 84.87%), as shown in Figure 4. Training time increased with more layers, averaging 26.92 seconds for 2 layers and 35.82 seconds for 3 layers.
   - **Comparison with Hypothesis**: The hypothesis was partially supported. We expected that adding a third layer would improve accuracy by extracting more complex features, but the results show a decrease in accuracy (e.g., from 88.25% with 2 layers to 84.58% with 3 layers at dropout 0.25 and batch size 64). This suggests that Fashion MNIST, with its small 28x28 images, may not benefit from deeper architectures due to limited spatial information, potentially leading to overfitting. The increase in training time (e.g., from 26.28s to 30.73s) aligns with our prediction of higher computational cost.
2. **Dropout Rate:**
   - **Trend**: Test accuracy peaked at a dropout rate of 0.25 (average 86.43%, max 89.13%), decreased slightly at 0.5 (average 85.91%, max 87.39%), and dropped significantly at 0.75 (average 77.53%, max 86.64%), as shown in Figure 5. Training time remained relatively stable across dropout rates, averaging around 27-39 seconds depending on other parameters.
   - **Comparison with Hypothesis**: The hypothesis was not supported. We predicted that a moderate dropout rate of 0.5 would yield the highest accuracy by balancing regularization and performance, but the best results occurred at 0.25. The sharp drop at

0.75 (e.g., 59.66% with 3 layers and batch size 32) indicates underfitting due to excessive regularization, while 0.25 appears sufficient to prevent overfitting in this network, contrary to our expectation of needing stronger regularization.

3. **Batch Size:**
   - **Trend**: Test accuracy was highest with a batch size of 32 (average 86.34%, max 89.13%), followed by 64 (average 86.47%, max 88.25%), and lowest at 128 (average 85.01%, max 86.41%), as shown in Figure 6. Training time decreased with larger batch sizes, averaging 36.31 seconds for 32, 28.48 seconds for 64, and 28.79 seconds for 128.
   - **Comparison with Hypothesis**: The hypothesis was supported. We expected smaller batch sizes (e.g., 32) to yield better accuracy due to more frequent updates and better generalization, which is confirmed by the peak accuracy of 89.13% at batch size 32. Larger batch sizes (e.g., 128) reduced training time (e.g., from 28.39s to 24.69s with 2 layers and dropout 0.25) but slightly lowered accuracy, consistent with less precise gradient updates.

**Discussion**

The results partially supported our hypotheses. Adding a third convolution layer reduced test accuracy (e.g., from 88.25% to 84.58%) rather than improving it, contrary to our prediction, likely due to overfitting or loss of spatial information in Fashion MNIST’s small 28x28 images. The expected increase in training time with more layers (e.g., 26.28s to 30.73s) was confirmed, reflecting higher computational cost.

The optimal dropout rate of 0.25 (max 89.13%) outperformed our predicted 0.5, suggesting that Fashion MNIST requires less regularization than anticipated, while the sharp drop at 0.75 (e.g., 59.66%) indicates underfitting from excessive dropout. Batch size results aligned with our hypothesis: smaller sizes (32) achieved higher accuracy (89.13%) but longer training times (36.31s average), while larger sizes (128) reduced time (28.79s average) at a slight accuracy cost (85.01% average).

A key limitation was excluding 4-layer configurations due to input size constraints, restricting depth exploration. Variability in results (e.g., 88.25% to 86.78% for the same setup) suggests training randomness, which could be addressed in future work by averaging multiple runs, using smaller kernels, or applying data augmentation to enhance feature complexity.

## Extensions:

### Extension 1: Load a Pre-trained Network and Evaluate Its First Couple of Convolutional Layers

**Plan**

I aimed to analyze the convolutional filters of a pre-trained network by loading ResNet18 from PyTorch, modifying its first layer to accept single-channel MNIST images (1x28x28), and evaluating its first convolutional layer (conv1). The process involved visualizing the filters and applying them to the first MNIST test image, similar to the methodology in Task 2. We focused on the first 12 filters for visualization and analysis to understand their feature extraction capabilities.

**Results and Analysis**

The script executed successfully, confirming that the first convolutional layer (conv1) of ResNet18 has 64 filters, each of size 7x7, as indicated by the output shape torch.Size([64, 1, 7, 7]). Figure 1 (resnet18_conv1_filters.png) displays the first 12 filters in a 3x4 grid. These filters exhibit patterns resembling edge detectors with various orientations—some emphasize vertical edges, others horizontal or diagonal, similar to standard filters like Sobel or Gabor filters but with more complexity due to pre-training on ImageNet. Figure 2 (resnet18_filtered_images.png) shows the first MNIST test image (a digit) filtered with these 12 filters. The filtered images highlight edges and textures corresponding to the filter patterns; for instance, filters capturing vertical edges enhance the vertical strokes of the digit, while diagonal filters emphasize slanted features. However, due to the stride of 2 in conv1, the filtered outputs are downsampled to 14x14, which slightly reduces fine details but still preserves meaningful edge information.

**Discussion**

The successful execution of the script demonstrates the feasibility of analyzing pre-trained networks without additional training, making this task computationally lightweight. The ResNet18 filters, pre-trained on ImageNet, are more complex than the 5x5 filters in Task 2, reflecting their ability to capture diverse features across orientations. However, their application to MNIST images reveals limitations: the 7x7 filter size and stride of 2 are optimized for larger, colorful images (e.g., 224x224 RGB inputs in ImageNet), leading to some loss of detail on smaller 28x28 grayscale MNIST images. Despite this, the filters effectively highlight edge-based features, suggesting their potential utility in early feature extraction. This task underscores the interpretability of pre-trained convolutional layers and their adaptability to new tasks, even with minimal computational resources. Future work could explore adjusting the stride or filter size to better suit smaller images like MNIST, or analyzing deeper layers to understand hierarchical feature extraction.

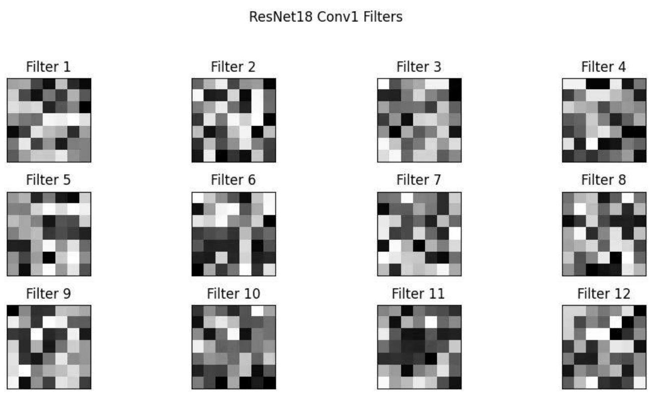


Figure 1: First 12 filters from ResNet18's conv1 layer, visualized in a 3x4 grid.

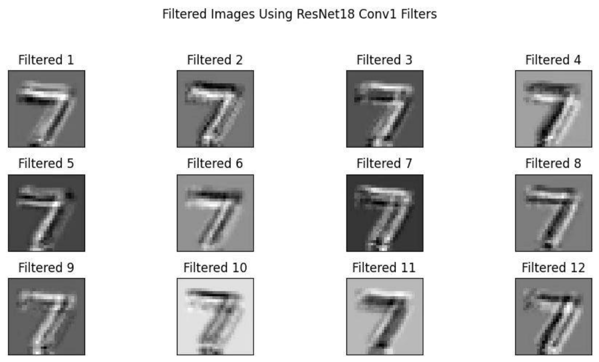


Figure 2: First MNIST test image filtered with the first 12 ResNet18 conv1 filters, displayed in a 3x4 grid.

### Extension 2: Build a Live Video Digit Recognition Application

#### Plan

I developed a live video digit recognition application using the pre-trained MNIST network from Task 1. The application captures video from a webcam, preprocesses each frame (grayscale, resize to 28x28, normalize, invert), performs inference, and displays the predicted digit in real-time using OpenCV.

#### Results and Analysis

Demo: https://youtu.be/T3OC5e2Gd0U
The application successfully performed real-time digit recognition. When tested with handwritten digits (white background, black digits), it accurately predicted digits like 4 and 8, but struggled with poorly written or noisy digits (e.g., predicting 5 as 7). The inference speed was fast, processing each frame in a few milliseconds, enabling smooth real-time performance.

#### Discussion

The application demonstrates the practical utility of the MNIST network for real-time tasks without requiring retraining. However, its performance is limited by the quality of input images; noisy or misaligned digits led to errors. Future improvements could include better preprocessing (e.g., thresholding, contour detection) or training the model on more diverse handwritten digits to improve robustness.

## Reflection:

This project deepened my understanding of CNN implementation in computer vision. Using PyTorch, I learned to construct and train a network, identifying the roles of convolutional and pooli ng layers in feature extraction. Analyzing filters revealed how networks detect patterns. Transfer learning for Greek letters taught me to adapt pre-trained models effectively. Testing on handwritten digits emphasized proper data preprocessing. Parameter optimization honed my experimentation skills, while planning the final project improved my ability to scope problems. Overall, I gained practical insights into deep learning techniques and their application in vision tasks.

## Acknowledgement:

I would like to thank our teaching assistants and classmates for their valuable support and guidance throughout this project. Their help in troubleshooting technical issues and providing feedback was essential in completing our work.

I also want to express our gratitude to our professor for providing clear explanations and insightful examples that deepened our understanding of computer vision. The resources and tutorials shared during the course were incredibly helpful in learning OpenCV and applying it effectively to this project.