CS 5330: Pattern Recognition and Computer Vision (Spring 2024)

Report

Project 5: Recognition using Deep Networks

Submitted by: Haard Shah
Hrigved Suryawanshi

Description:

In this project, we will learn to build, train, analyze and modify a deep neural network for recognition using the MNIST dataset. We will use the PyTorch library to create a convolutional neural network (CNN) and train it on the dataset. We will also evaluate the performance of the network on a test set and analyze its results.

Tasks:

1. **Build and train a network to recognize digits**
   For the first task we will build and train a network to do digit recognition using the MNIST data base and then save the network to a file so it can be re-used for the tasks assigned further in the project.

A. Get the MNIST digit data set

For this we have imported the MNIST digit data set within our code by using "**torchvision.datasets.MNIST**" from "**torchvision**" package
Further we have plotted the first six example digits from the test set.

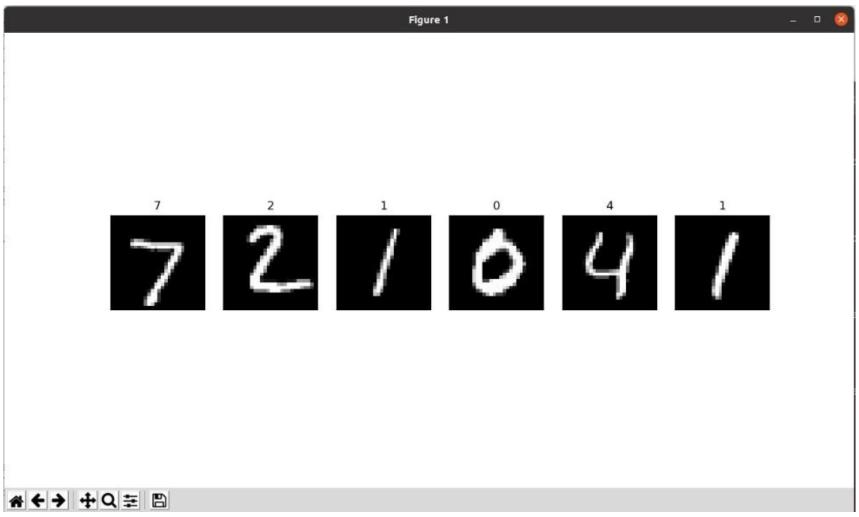


Figure: First six example digits from the test set

B. Build a network model

Our network class will be present in the constructor and also forward method of network class derived from **"torch nn.Module class".**
Here, we have designed a network model, and the diagram is as follows:

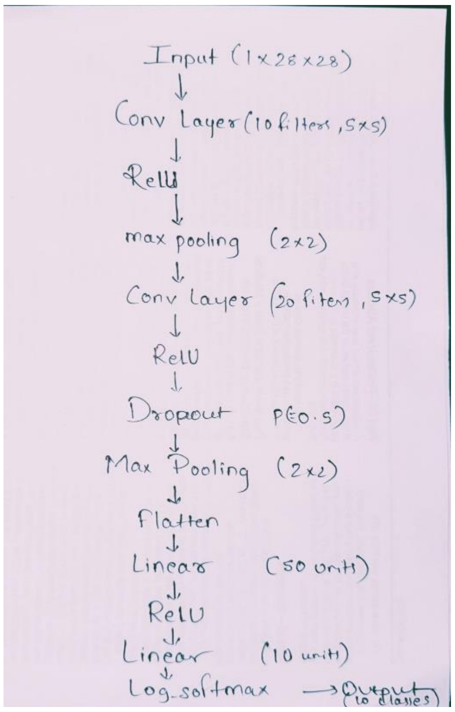
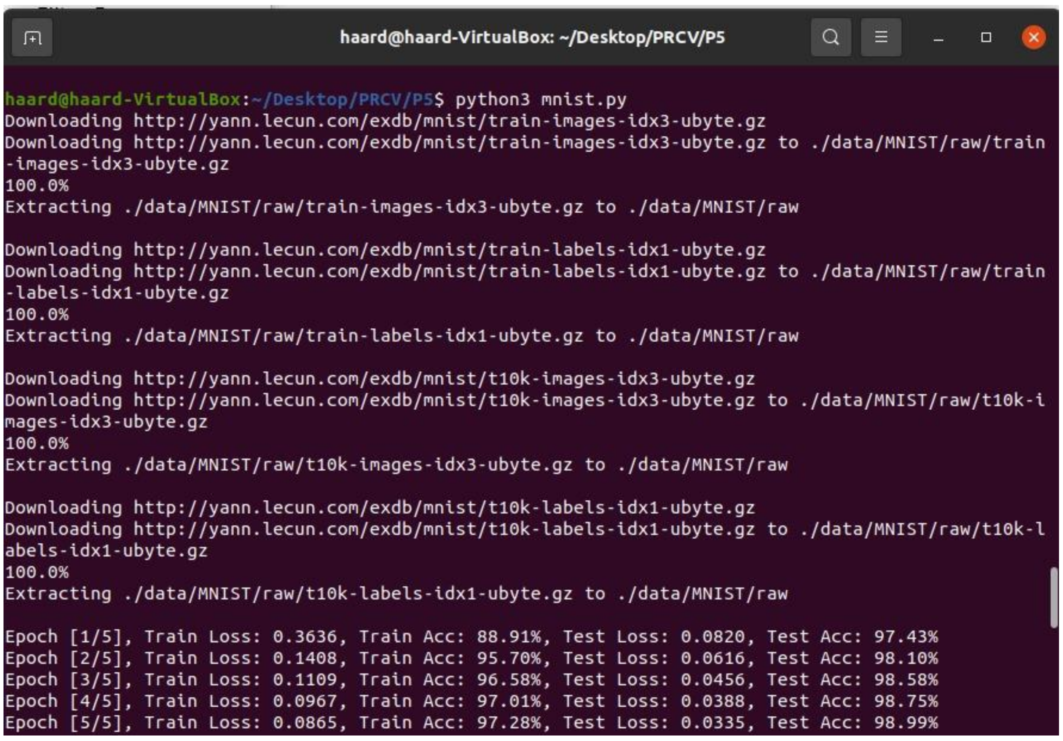


Figure: Hand-drawn network diagram with layer information

C. Train the model

We have trained the model for exactly five epochs and we have evaluated the data obtained for both training and testing sets after each epoch.
Here, we have shown two different plots: First one is the Plot of training and testing Errors, and the Second one is Training and Testing accuracy respectively.



```
haard@haard-VirtualBox:~/Desktop/PRCV/P5$ python3 mnist.py
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train
-images-idx3-ubyte.gz
100.0%
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train
-labels-idx1-ubyte.gz
100.0%
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-i
mages-idx3-ubyte.gz
100.0%
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-l
abels-idx1-ubyte.gz
100.0%
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw

Epoch [1/5], Train Loss: 0.3636, Train Acc: 88.91%, Test Loss: 0.0820, Test Acc: 97.43%
Epoch [2/5], Train Loss: 0.1408, Train Acc: 95.70%, Test Loss: 0.0616, Test Acc: 98.10%
Epoch [3/5], Train Loss: 0.1109, Train Acc: 96.58%, Test Loss: 0.0456, Test Acc: 98.58%
Epoch [4/5], Train Loss: 0.0967, Train Acc: 97.01%, Test Loss: 0.0388, Test Acc: 98.75%
Epoch [5/5], Train Loss: 0.0865, Train Acc: 97.28%, Test Loss: 0.0335, Test Acc: 98.99%
```

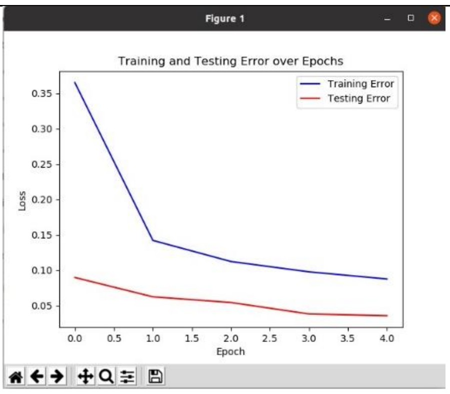


Figure: Error graph Train vs Test

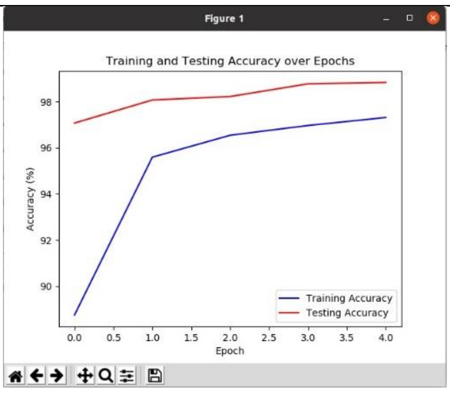


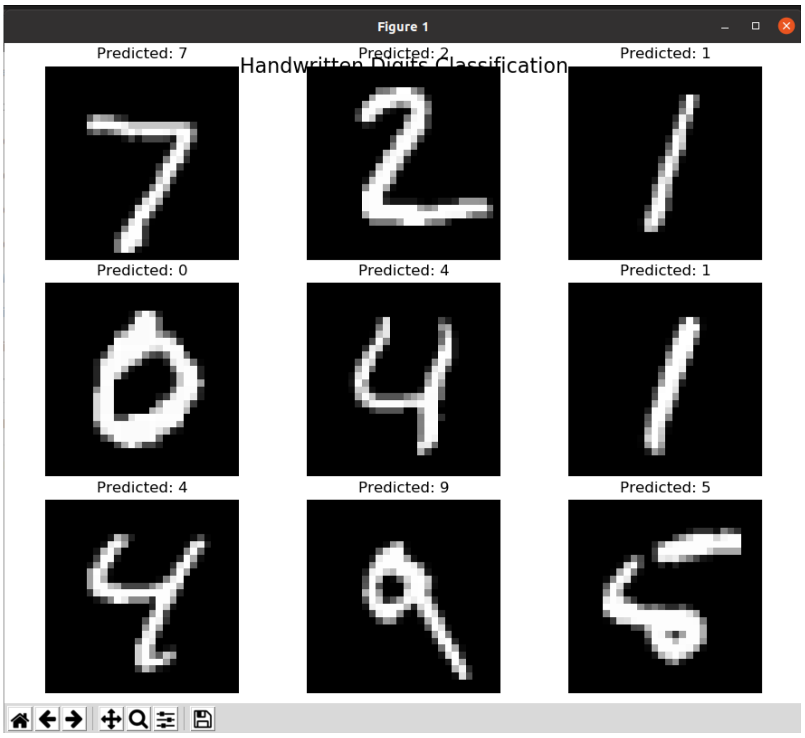
Figure: Accuracy graph Train vs Test

D. Save the network to a file

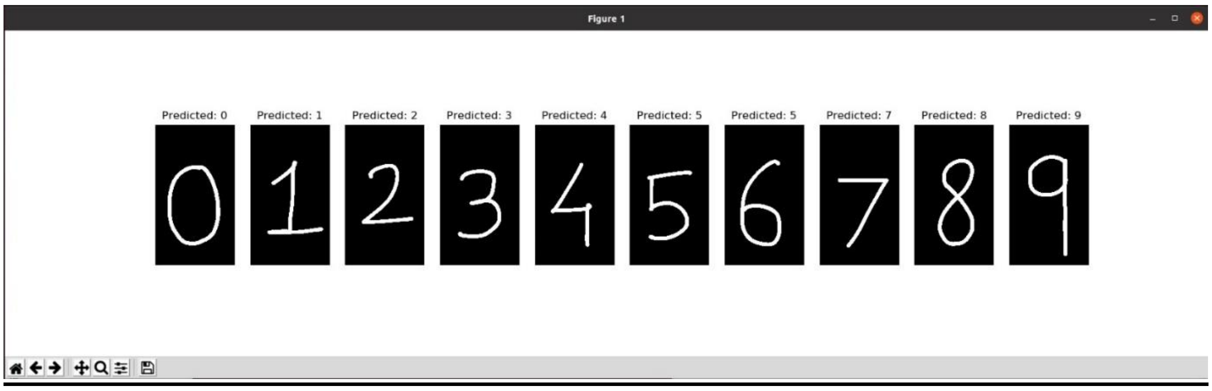
We have saved the network as **"mnist_model.pth"**

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

In a separate Python file, the saved network was loaded and set to evaluation mode to ensure consistent output by multiplying each value by 1-dropout rate, rather than randomly setting node values to zero as in training mode. The first 10 examples from the test set were then run through the network, printing out the 10 network output values (rounded to 2 decimal places), the index of the max output value, and the correct label of the digit for each example. The network correctly classified all 10 examples. Additionally, the first 9 digits were plotted as a 3x3 grid with the prediction for each example above it, providing a visual representation of the network's performance.



F. Test the network on new inputs



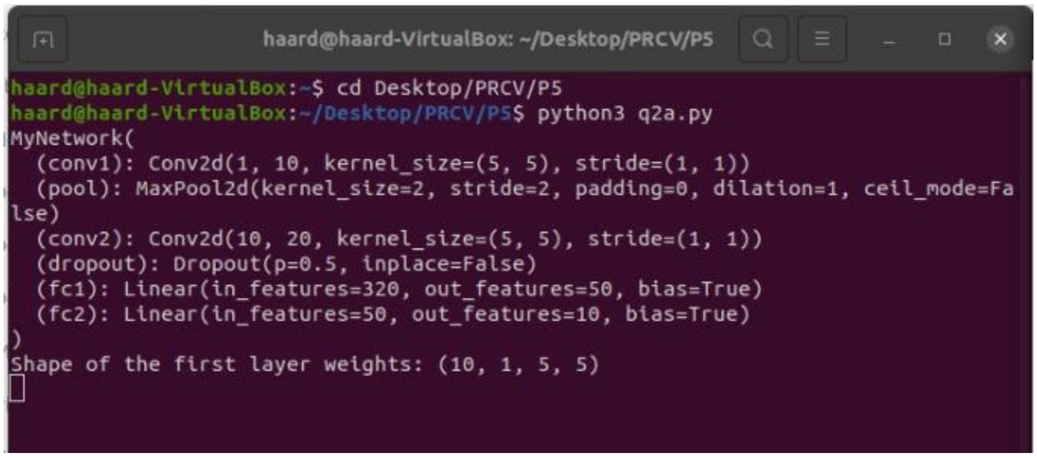
In this scenario, handwritten digits from 0 to 9 are being predicted, but there's a challenge in accurately recognizing the digit that closely resembles a 5. Despite attempting various input adjustments to improve recognition specifically for digit 6, the desired result for this digit remains elusive. However, a potential solution lies in augmenting the training data with additional samples for digit 6, as well as enriching the test data with more instances of this digit. By incorporating more diverse examples of digit 6 into both the training and testing datasets, the accuracy of its recognition can be significantly enhanced.

2. Examine your network

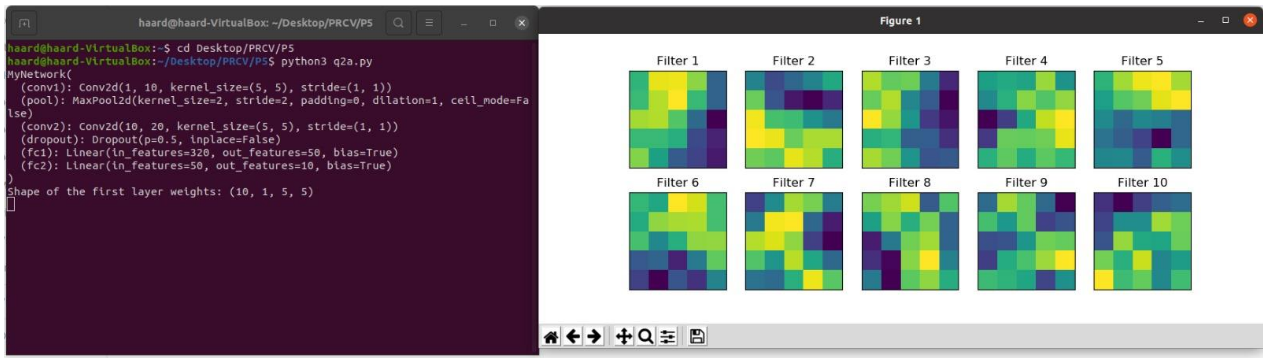
In this task, the goal was to examine the structure of the trained MNIST digit recognition network and analyze how it processes the input data. This involved inspecting the first convolutional layer of the network to understand the learned filters and their effects on the input images.

A. Analyze the first layer

The approach for examining the network architecture and filters involved several key steps. First, the trained MNIST digit recognition network was loaded, and its model summary was printed to reveal the overall structure, including the convolutional, pooling, and fully connected layers. Next, the weights of the first convolutional layer, named "conv1", were extracted and analyzed. The shape of the weight tensor was inspected, indicating the number of filters and their size. The individual filters were then visualized using a 3x4 grid of subplots to understand the types of features they were capturing, such as edges and textures.



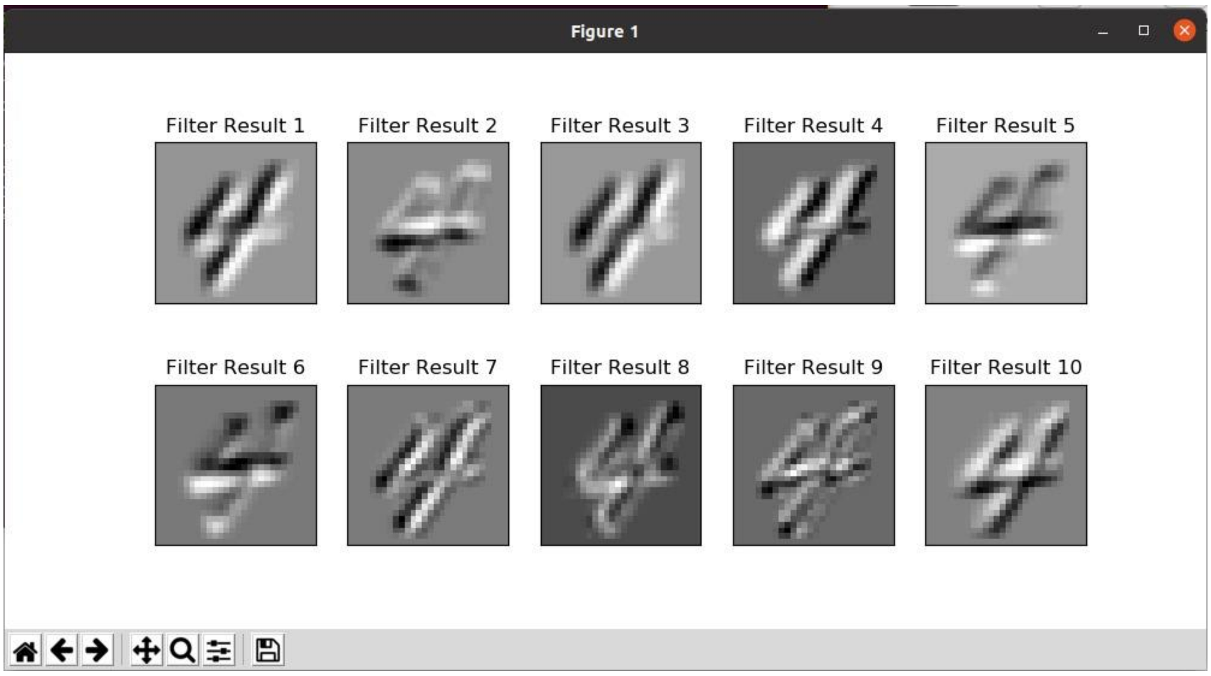
```
haard@haard-VirtualBox: ~/Desktop/PRCV/P5
haard@haard-VirtualBox:~$ cd Desktop/PRCV/P5
haard@haard-VirtualBox:~/Desktop/PRCV/P5$ python3 q2a.py
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
lse)
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (dropout): Dropout(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)
)
Shape of the first layer weights: (10, 1, 5, 5)
```



```
haard@haard-VirtualBox:~$ cd Desktop/PRCV/P5
haard@haard-VirtualBox:~/Desktop/PRCV/P5$ python3 q2a.py
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
lse)
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (dropout): Dropout(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)
)
Shape of the first layer weights: (10, 1, 5, 5)
```

B. Show the effect of the filter

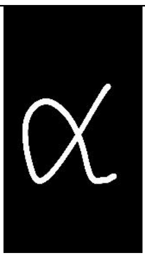
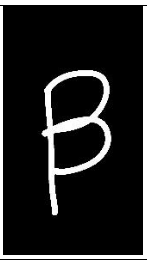
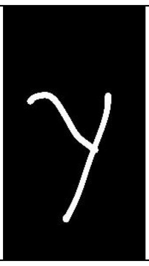
Finally, the 10 filters from the first convolutional layer were applied to the first training example image using OpenCV's filter2D function, and the resulting filtered images were plotted to demonstrate the effect of the filters on the input data.



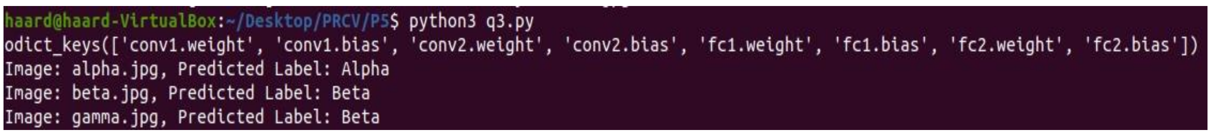
3. <u>Transfer Learning on Greek Letters</u>

In this task, the goal was to leverage the pre-trained MNIST digit recognition network to recognize three different Greek letters: alpha, beta, and gamma. By using transfer learning, the aim was to efficiently adapt the existing model to the new task of Greek letter classification.

The key steps in the transfer learning process were as follows. First, the code from the previous task was used to generate the MNIST digit recognition network. Next, the pre-trained weights of the MNIST network were loaded from a file, preserving the learned features. To prevent the pre-trained features from being overwritten during fine-tuning, all the network weights were frozen. The final layer of the network was then replaced with a new linear layer with three nodes, corresponding to the three Greek letters (alpha, beta, gamma). The Greek letter images were transformed to match the input requirements of the MNIST network, including converting them to grayscale, scaling and cropping them to 28x28 pixels, and inverting the pixel intensities. Finally, the modified network was trained on the Greek letter dataset using the transformed images.

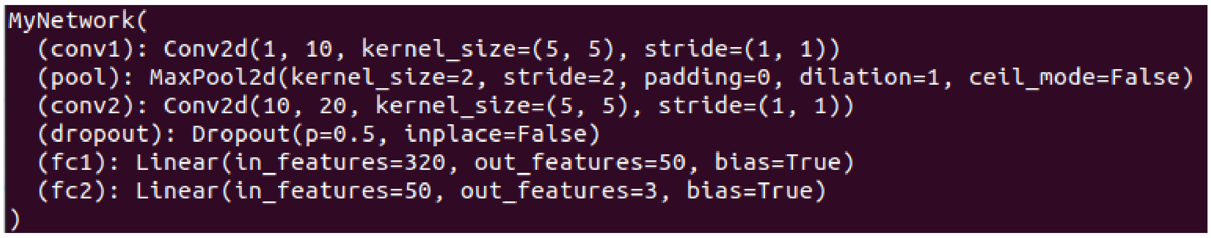
		

Here are the images used to test the data and find the predicted labels from the data we trained before.



```
haard@haard-VirtualBox:~/Desktop/PRCV/P5$ python3 q3.py
odict_keys(['conv1.weight', 'conv1.bias', 'conv2.weight', 'conv2.bias', 'fc1.weight', 'fc1.bias', 'fc2.weight', 'fc2.bias'])
Image: alpha.jpg, Predicted Label: Alpha
Image: beta.jpg, Predicted Label: Beta
Image: gamma.jpg, Predicted Label: Beta
```

The model's prediction for the gamma image significantly deviated from the expected outcome, even after training it with multiple images from the greek_train dataset.



```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool): 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=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
```

Figure: Modified Network

4. Design you own Experiment

A. Develop a plan

We aim to explore the impact of different architectural variations on the performance and training dynamics of the neural network. We will employ a systematic approach, evaluating 50-100 network variations. Firstly, we will define the dimensions to explore:

Number of convolution layers
Filter sizes
Number of filters in each layer
Number of hidden nodes in the fully connected layer
Dropout rates

For each dimension, we will determine a range of options to explore, ensuring diversity and comprehensiveness. We will then adopt a linear search strategy, where we hold two parameters constant and optimize the third. We will iteratively switch parameters and continue optimization until all dimensions are explored. Throughout the evaluation, we will track key metrics such as training and test accuracy, loss, convergence time, and computational resources utilized. This approach will provide insights into how different architectural variations affect network performance and guide us in selecting optimal configurations.

Experiment Parameters:

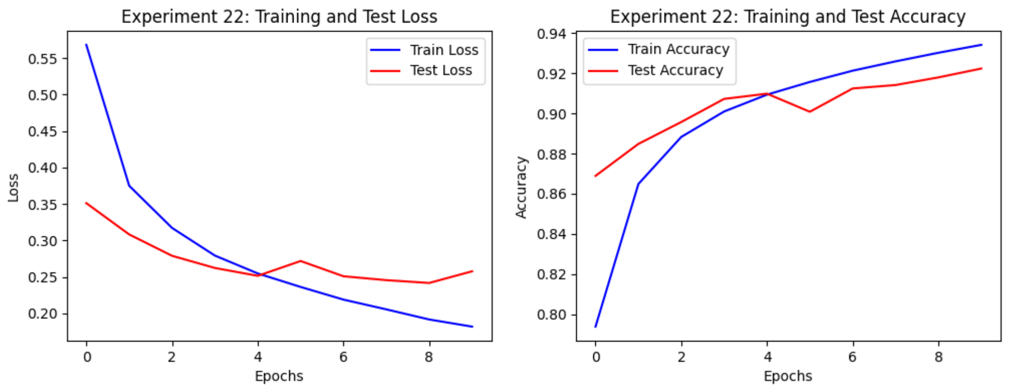
Number of Convolution Layers: [2, 3, 4, 5]
Number of Filters: [32, 64]
Filter Size: [3]
Number of Hidden Nodes: [64, 128, 256]
Dropout Rate: [0.1, 0.3, 0.5]
Epochs: 10
Batch Size: 32
Number of Variations: 50

B. Predict the results

We hypothesize that altering different aspects of the network architecture will affect its performance and training dynamics. Increasing the number of convolution layers is expected to improve feature learning capacity but may prolong training. Larger filter sizes may enhance feature extraction but risk overfitting and longer convergence times. Augmenting the number of filters can improve feature representation but may increase complexity and training time. Similarly, raising the number of hidden nodes may boost model capacity but also heighten overfitting and convergence delays. Dropout regularization is anticipated to enhance generalization by mitigating overfitting, though higher dropout rates may hinder convergence. Overall, these architectural variations are likely to trade off between improved performance and longer training times.

C. Execute your plan

Upon executing the code, the experiments were conducted to explore the effects of various network architecture configurations on the performance and training dynamics. Experiment 22 emerged as the top-performing configuration, boasting the **highest accuracy of 92.23%**. This experiment spanned a duration of 4.5 hours and featured a network configuration comprising 3 convolution layers, a filter size of 3, a dropout rate of 0.4, and 64 filters. The achieved accuracy surpassed expectations and underscored the significance of fine-tuning these architectural parameters for optimizing model performance. Additionally, the extended training duration necessitated by Experiment 22 reflects the trade-off between accuracy and training time inherent in deep learning experimentation. Overall, the results underscore the importance of meticulous experimentation to identify the optimal network configuration for a given task.



Learning Outcomes:

Through this project-based learning experience, we have developed a comprehensive understanding of deep neural networks, with a specific focus on convolutional neural networks (CNNs) for handwritten digit recognition. We have designed and implemented a CNN architecture for the MNIST digit recognition task, training the network to achieve high classification accuracy.

By examining the weights and visualizing the filters in the first convolutional layer, we have gained valuable insights into the network's feature extraction process. This has allowed me to develop a deeper understanding of how the network processes and learns from the input images.

Furthermore, we have leveraged the pre-trained MNIST network to recognize three Greek letters (alpha, beta, gamma) through transfer learning. This process of adapting the existing network to a new task and evaluating its performance has enhanced my skills in applying deep learning techniques to diverse applications.

Experimenting with various network architecture modifications, we have systematically evaluated the impact on performance (accuracy) and training time.

Throughout this project, we have gained hands-on experience with deep learning concepts, techniques, and tools, including PyTorch, torchvision, and potentially OpenCV for image processing. This comprehensive learning experience has provided me with a solid foundation in deep neural network design, analysis, and experimentation, enabling me to apply these skills to a wide range of deep learning applications.

Acknowledgement:

We extend our heartfelt gratitude to the following platforms for their invaluable support and resources throughout the Recognition using deep networks project:

- Springer
- ResearchGate
- GitHub
- Medium
- Stack Overflow
- ChatGPT
- Mistral AI

The insights and references obtained from these platforms have significantly enriched our understanding and greatly contributed to the successful development and implementation of various aspects of the project. Their assistance has been indispensable in ensuring a comprehensive and well-informed approach to the tasks at hand. We are truly thankful for the wealth of information and guidance provided by these platforms, which have been essential in our journey towards excellence in Recognition using deep network project.