Name: Carolina Li
Project: Recognition using Deep Networks
Date: Nov/18/2024

Description:

This project focuses on building, training, and analyzing deep networks for image recognition, using the MNIST digit recognition dataset as a foundation. It involves designing a convolutional neural network with layers for feature extraction, dropout for regularization, and fully connected layers for classification. Tasks include training the network, visualizing performance through training and testing accuracy plots, and saving the model for reuse. A key objective is to understand how network architecture and hyperparameters affect performance.

The project also includes transfer learning, where the trained MNIST network is adapted to classify Greek letters (alpha, beta, gamma, omega, epsilon, and rho) by freezing pre-trained layers and fine-tuning the final classification layer. Additional tasks explore visualizing convolutional filters and applying the trained model to real-world handwritten inputs, demonstrating the network's ability to generalize.

A major component is experimentation, where network dimensions (e.g., number of layers, filter sizes, dropout rates) are varied to evaluate their impact on accuracy and training time. Extensions include recognizing additional Greek letters, applying pre-trained networks to other datasets, and testing innovative ideas like live video recognition. This project provides a comprehensive introduction to deep learning, offering hands-on experience in designing, fine-tuning, and evaluating neural networks**.**

Reflection:

Working on this task gave me a hands-on understanding of building and training a convolutional neural network (CNN) using PyTorch to recognize digits from the MNIST dataset. I learned how to design a model by stacking layers like convolution, pooling, dropout, and activation functions, and how to optimize its performance through iterative training and evaluation. Visualizing the filters from the first convolutional layer and seeing how they process images helped me understand how the network extracts features from input data. I also explored the concept of transfer learning by adapting the pre-trained model to recognize Greek letters, which showed me how to reuse existing models for new tasks with minimal adjustments. Overall, this project deepened my knowledge of CNNs, their flexibility, and the importance of visualization and testing in machine learning workflows.

Acknowledgement:

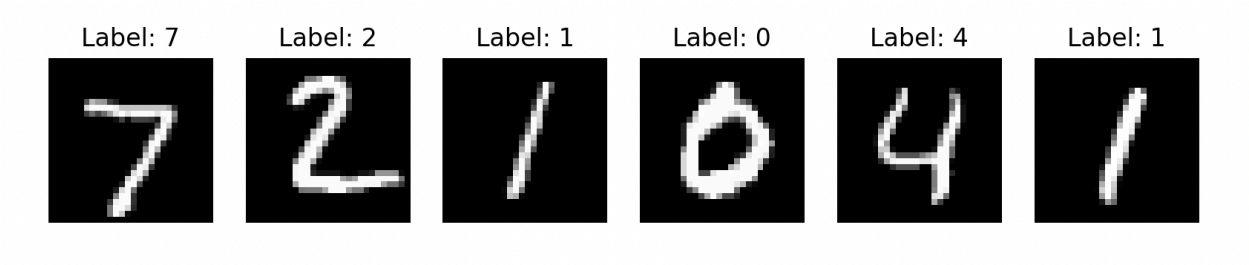
I would like to acknowledge the following resources and individuals for their contributions to this project:

1. **Pytorch and MNIST Documentation**

Task1: Build and train a network to recognize digits

A. Get the MNIST digit data set

First six digits:

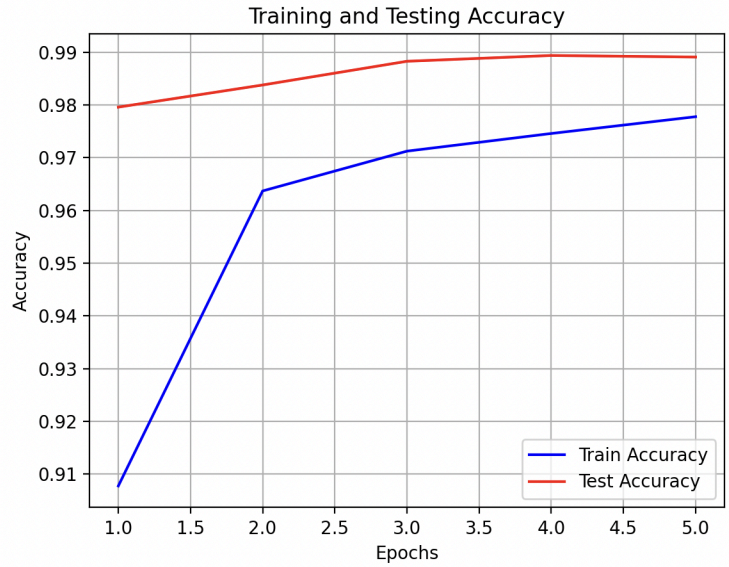


B. Build a network model:

```
(base) carolina1650@MacBookPro DeepNetwork % python -u "/Users/carolina1650/DeepNetwork/task1.py"
2024-11-16 21:08:01.839 python[12251:4126914] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-16 21:08:01.839 python[12251:4126914] +[IMKInputSession subclass]: chose IMKInputSession_Modern
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 10, 24, 24]             260
         MaxPool2d-2          [-1, 10, 12, 12]               0
            Conv2d-3            [-1, 20, 8, 8]           5,020
         MaxPool2d-4            [-1, 20, 4, 4]               0
           Dropout-5            [-1, 20, 4, 4]               0
            Linear-6                  [-1, 50]          16,050
            Linear-7                  [-1, 10]             510
================================================================
Total params: 21,840
Trainable params: 21,840
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.07
Params size (MB): 0.08
Estimated Total Size (MB): 0.16
----------------------------------------------------------------
```

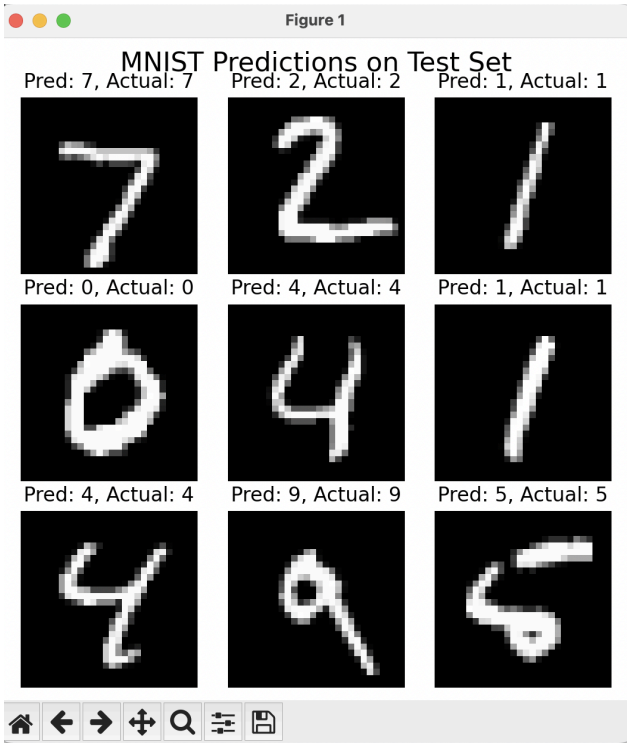
C. Train the model:

Epoch 1/5 | Train Accuracy: 0.9077 | Test Accuracy: 0.9796
Epoch 2/5 | Train Accuracy: 0.9637 | Test Accuracy: 0.9838
Epoch 3/5 | Train Accuracy: 0.9712 | Test Accuracy: 0.9883
Epoch 4/5 | Train Accuracy: 0.9746 | Test Accuracy: 0.9894
Epoch 5/5 | Train Accuracy: 0.9778 | Test Accuracy: 0.9891



The red line shows the test accuracy is higher than the train accuracy in general, but two of them stay similar to the trend from epoch 2.0 and stay stable from epoch 4.0.

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

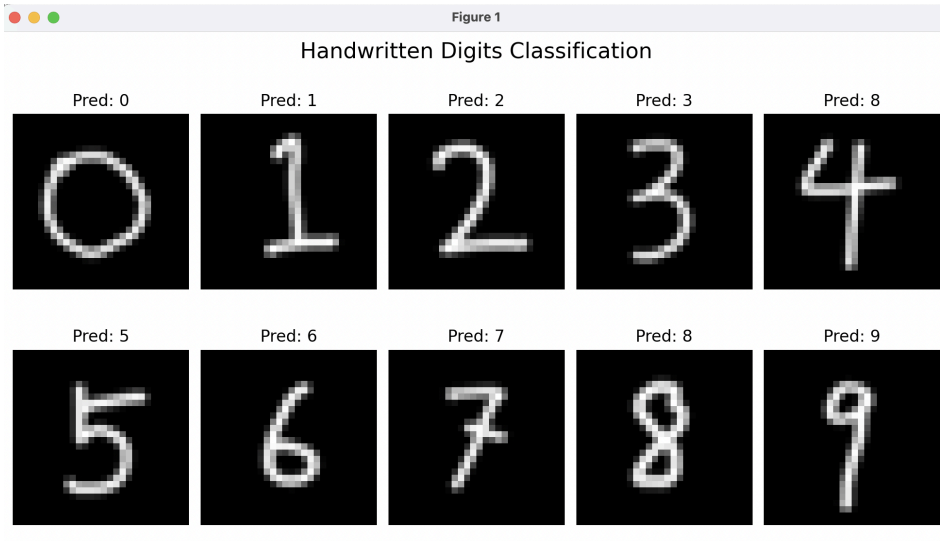


This grid shows the first 9 predictions generated from the trained model on the MNIST test set. The model accurately predicted all digits.

```
(venv) (base) carolina1650@MacBookPro DeepNetwork % python -u "/Users/carolina1650/DeepNetwork/
readNetworks.py"
/Users/carolina1650/DeepNetwork/readNetworks.py:9: FutureWarning: You are using `torch.load` wi
th `weights_only=False` (the current default value), which uses the default pickle module impli
citly. It is possible to construct malicious pickle data which will execute arbitrary code duri
ng unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models fo
r more details). In a future release, the default value for `weights_only` will be flipped to `
True`. This limits the functions that could be executed during unpickling. Arbitrary objects wi
ll no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by th
e user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only
=True` for any use case where you don't have full control of the loaded file. Please open an is
sue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load('mnist_model.pth'))
Evaluating the model on the first 10 test images:

Image 1:
Output values: [-16.70, -16.64, -12.87, -11.69, -21.69, -16.30, -30.10, -0.00, -15.62, -12.81]
Predicted: 7, Actual: 7

Image 2:
Output values: [-12.67, -13.51, -0.00, -16.62, -20.65, -20.31, -13.11, -21.49, -15.19, -22.90]
Predicted: 2, Actual: 2

Image 3:
Output values: [-12.27, -0.00, -10.76, -12.79, -9.11, -11.85, -11.91, -9.10, -10.99, -11.52]
Predicted: 1, Actual: 1

Image 4:
Output values: [-0.00, -22.90, -18.64, -20.06, -18.14, -12.44, -9.37, -18.81, -13.96, -14.62]
Predicted: 0, Actual: 0

Image 5:
Output values: [-19.57, -21.75, -16.93, -23.12, -0.00, -19.30, -16.94, -17.21, -16.81, -8.31]
Predicted: 4, Actual: 4

Image 6:
Output values: [-15.89, -0.00, -13.33, -17.53, -11.14, -16.06, -14.98, -11.21, -14.23, -14.90]
Predicted: 1, Actual: 1

Image 7:
Output values: [-23.10, -17.18, -14.37, -21.18, -0.00, -14.86, -17.73, -14.93, -8.93, -8.96]
Predicted: 4, Actual: 4

Image 8:
Output values: [-16.97, -20.01, -13.60, -16.09, -7.59, -14.02, -19.76, -13.25, -10.89, -0.00]
Predicted: 9, Actual: 9

Image 9:
Output values: [-14.88, -21.47, -17.89, -17.43, -18.46, -0.00, -8.51, -17.35, -10.82, -11.69]
Predicted: 5, Actual: 5

Image 10:
Output values: [-20.54, -24.73, -18.74, -13.78, -10.58, -15.92, -27.24, -12.75, -10.81, -0.00]
Predicted: 9, Actual: 9
```

F. **Test the network on new inputs**

I wrote the digits on the ipad initially, then resized to 28x28 using the ImageMagick package. Then it was converted to black background and white color text.




```
(venv) (base) carolina1650@MacBookPro DeepNetwork % python -u "/Users/carolina1650/DeepNetwork/Test_Han
dwritten_digits.py"
/Users/carolina1650/DeepNetwork/Test_Handwritten_digits.py:10: FutureWarning: You are using `torch.load` with `weights_only=False` (the
 current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will
execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more detail
s). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be execut
ed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by
the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don'
t have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load('mnist_model.pth'))
Digit 0: Predicted as 0
Digit 1: Predicted as 1
Digit 2: Predicted as 2
Digit 3: Predicted as 3
Digit 4: Predicted as 8
Digit 5: Predicted as 5
Digit 6: Predicted as 6
Digit 7: Predicted as 7
Digit 8: Predicted as 8
Digit 9: Predicted as 9
2024-11-17 21:00:42.095 python[81622:5694209] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-17 21:00:42.095 python[81622:5694209] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

The figure shows results on prediction of handwritten digits using the MNIST model. All images were resized to 28x28 pixels. The handwritten ‘4’ was predicted as ‘8’, and the rest of the predictions are all correct.

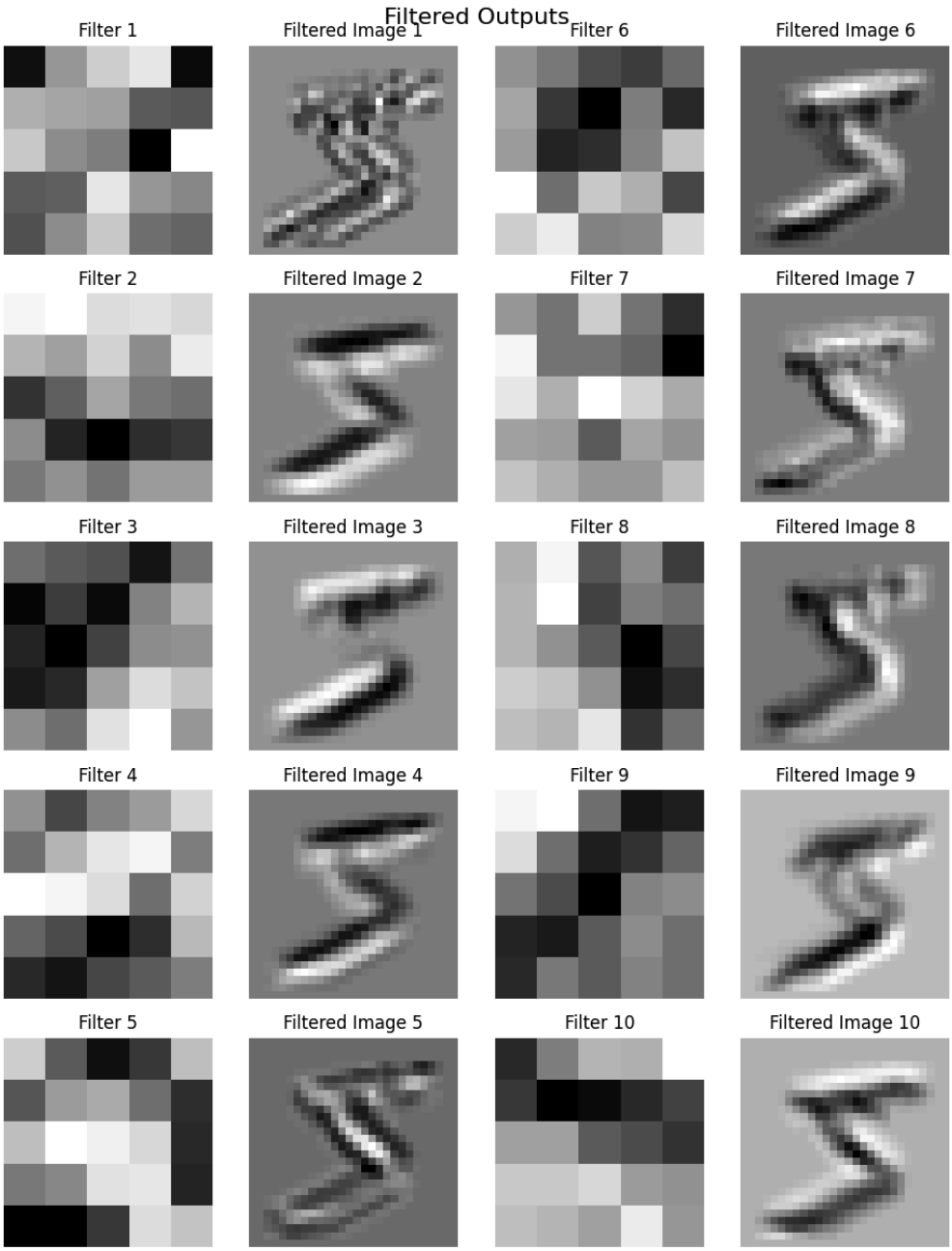
Task2: Examine your network

A. Analyze the first layer


```
(venv) (base) carolina1650@MacBookPro DeepNetwork % python -u "/Users/carolina1650/DeepNetwo
rk/task2_AnalyzeFirstLayer.py"
/Users/carolina1650/DeepNetwork/task2_AnalyzeFirstLayer.py:12: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default
 value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during un
pickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `
weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowe
d to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start sett
ing `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to
 this experimental feature.
  model.load_state_dict(torch.load('mnist_model.pth'))
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=10, bias=True)
)
2024-11-17 21:23:09.186 python[87192:5732369] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-17 21:23:09.186 python[87192:5732369] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

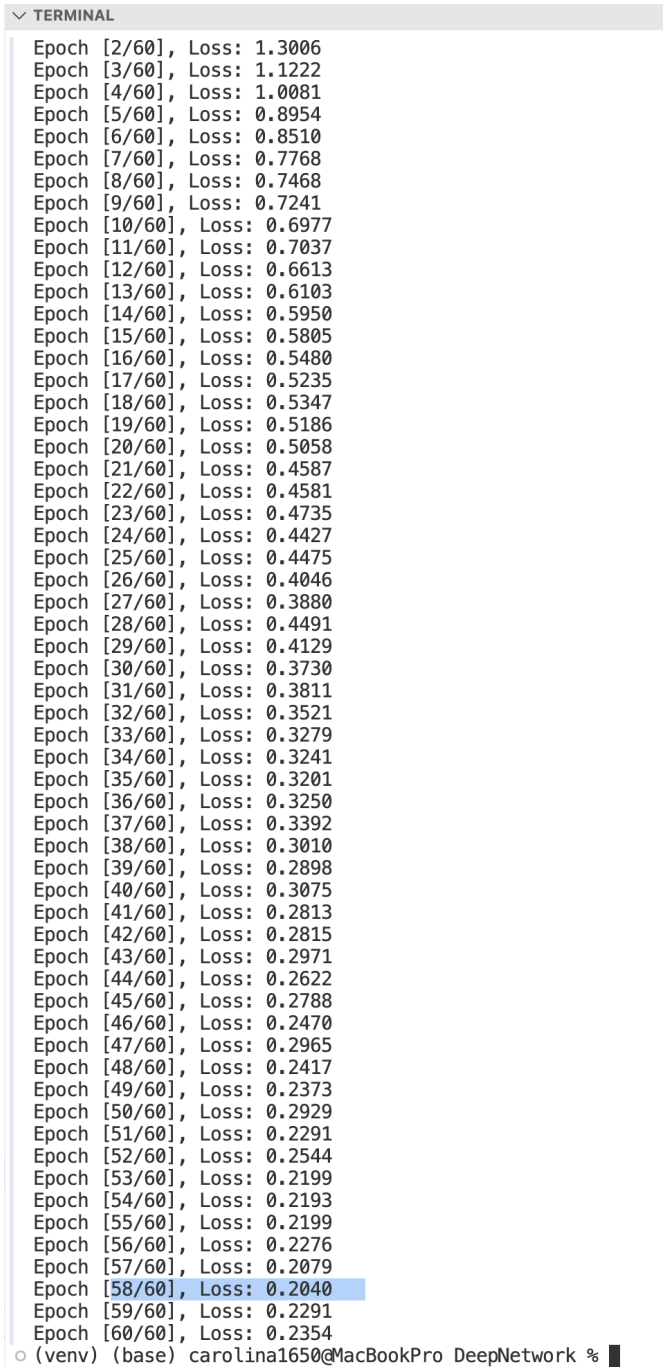
The figure shows the filter from the first convolution layer of a pre-trained neural network. The weights have a shape of [10, 1, 5, 5]. Each figure encodes an unique pattern and they are able to detect edges and main textures.

B. Show the effect of the filter



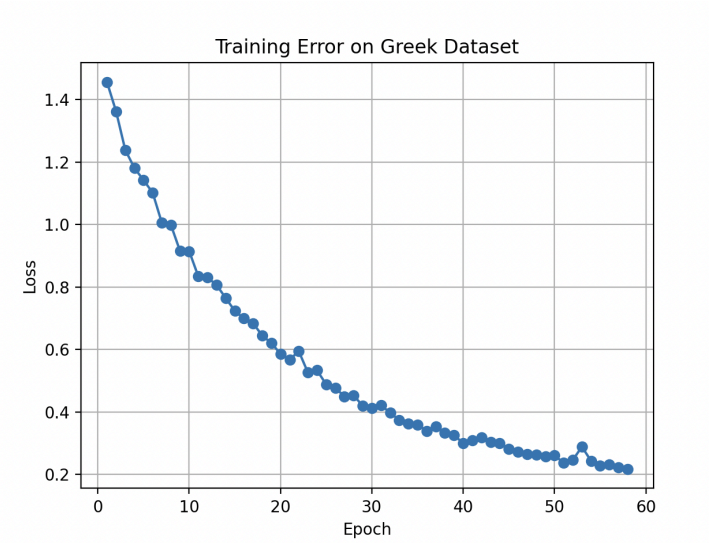
The result does make sense, the result shows what will be after the effect of the first convolution layer, including sharp edges, gradient and main textures of the original image.

Task3. Transfer Learning on Greek Letters



Up until epoch 58, the loss was decreasing, and increasing after epoch 58 the loss started increasing. So we can say the loss at epoch 58 is as minimal as possible. At … 27 examples

```
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)
)
```




```
Per-Class Accuracy:
  alpha: 100.00% (3/3)
  beta: 0.00% (0/3)
  gamma: 100.00% (3/3)

Overall Accuracy on Hand-Drawn Greek Letters: 66.67%

Detailed Predictions:
Image 1: Predicted = alpha, Actual = alpha
Image 2: Predicted = alpha, Actual = alpha
Image 3: Predicted = alpha, Actual = alpha
Image 4: Predicted = gamma, Actual = beta
Image 5: Predicted = gamma, Actual = beta
Image 6: Predicted = gamma, Actual = beta
Image 7: Predicted = gamma, Actual = gamma
Image 8: Predicted = gamma, Actual = gamma
Image 9: Predicted = gamma, Actual = gamma
(venv) (base) carolina1650@MacBookPro DeepNetwork %
```

The figure displays loss on Greek letters over 58 epochs while the model fine-tunes using the fully-connected layer. The loss starts high as 1.4 and decreases steadily over epochs, around epoch 50, the loss stabilized around 0.2. The loss curve is relatively smooth, indicating learning rates and batch size are chosen appropriately.

Task 4: Design Your Own Experiment:

Experiment Design:

The number of convolution layers: [2, 3, 4]
The size of the convolution filters: 3x3
The number of convolution filters in a layer: [(16, 32), (32, 64), (64, 128)]
The number of hidden nodes in the Dense layer: 128
The dropout rates of the Dropout layer: [0.3, 0.5, 0.7]
Whether to add another dropout layer after the fully connected layer: No, only one dropout layer is used after the Dense layer.
The size of the pooling layer filters: 2x2
The number or location of pooling layers: One pooling layer is added after each convolutional layer.
The activation function for each layer: ReLU
The number of epochs of training: 5
The batch size while training: 64

Plan:
The experiment aims to optimize the performance and training time of a convolutional neural network (CNN) on the MNIST dataset by exploring three key dimensions: the number of convolution layers, the number of filters in each layer, and the dropout rates. The metrics for evaluation include training loss and test accuracy. The parameter grid contains:

- **Convolution layers**: [2,3,4][2, 3, 4][2,3,4],
- **Filters per layer**: [(16,32),(32,64),(64,128)][(16, 32), (32, 64), (64, 128)][(16,32),(32,64),(64,128)],
- **Dropout rates**: [0.3,0.5,0.7][0.3, 0.5, 0.7][0.3,0.5,0.7].

A linear search strategy is used where two dimensions are held constant while varying the third. Approximately 3x3x3=27 combinations are evaluated.

Hypothesis:

Increasing the number of convolution layers will improve accuracy but may increase training time; larger filter sizes (e.g., (64,128)(64, 128)(64,128)) will capture more complex patterns, improving performance at the cost of computational overhead.

Moderate dropout rates (0.50.50.5) will balance overfitting and underfitting, leading to optimal performance.

Results:

The CNN is trained for 5 epochs using a batch size of 64. Each configuration uses ReLU activations, 3x3 convolution filters, and 2x2 max-pooling layers after each convolution. A single dropout layer follows the Dense layer with 128 hidden nodes. The results, including accuracy and the best-performing configuration, are saved in a text file.

Extensions

Extension 1: Evaluate more dimensions on “Transfer Learning on Greek Letters”

The figure shows losses over 10 epochs. The loss starts at 1.3 and reaches a value below 0.9 when it reaches 10 epochs, this means the model is learning and adapting its weights.

Results show the accuracy increased after increasing the dimension of the MaxPool2D layer.

```
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)
)

Layer-wise Output Dimensions:
Layer: Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1)), Output Shape: torch.Size([1, 10, 24, 24])
Layer: MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False), Output Shape: torch.Size([1, 20, 4, 4])
Layer: Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1)), Output Shape: torch.Size([1, 20, 8, 8])
Layer: Dropout(p=0.5, inplace=False), Output Shape: torch.Size([1, 20, 4, 4])
Layer: Linear(in_features=320, out_features=50, bias=True), Output Shape: torch.Size([1, 50])
Layer: Linear(in_features=50, out_features=3, bias=True), Output Shape: torch.Size([1, 3])
Epoch [1/10], Loss: 1.2991
Epoch [2/10], Loss: 1.2141
Epoch [3/10], Loss: 1.1760
Epoch [4/10], Loss: 1.1118
Epoch [5/10], Loss: 1.0623
Epoch [6/10], Loss: 1.0057
Epoch [7/10], Loss: 0.9546
Epoch [8/10], Loss: 0.9219
Epoch [9/10], Loss: 0.8925
Epoch [10/10], Loss: 0.8322
2024-11-23 16:36:53.950 python[99617:2803213] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-23 16:36:53.950 python[99617:2803213] +[IMKInputSession subclass]: chose IMKInputSession_Modern

Evaluation Accuracy on Hand-Drawn Greek Letters: 55.56%
Per-Class Accuracy:
  alpha: 66.67%
  beta: 0.00%
  gamma: 100.00%
Input Shape: torch.Size([1, 1, 28, 28])
Output Logits Shape: torch.Size([1, 3])
Predicted Class: 2, Actual Class: 0
(venv) (base) carolina1650@MacBookPro DeepNetwork %
```

This extension increased the dimension of MaxPool2D from [1, 10, 12, 12] to [1, 20, 4 , 4]

Although adding the dimension did not increase the accuracy significantly, the figure below is the result before increasing the accuracy:

```
Per-Class Accuracy:
  alpha: 100.00% (3/3)
  beta: 0.00% (0/3)
  gamma: 100.00% (3/3)

Overall Accuracy on Hand-Drawn Greek Letters: 66.67%

Detailed Predictions:
Image 1: Predicted = alpha, Actual = alpha
Image 2: Predicted = alpha, Actual = alpha
Image 3: Predicted = alpha, Actual = alpha
Image 4: Predicted = gamma, Actual = beta
Image 5: Predicted = gamma, Actual = beta
Image 6: Predicted = gamma, Actual = beta
Image 7: Predicted = gamma, Actual = gamma
Image 8: Predicted = gamma, Actual = gamma
Image 9: Predicted = gamma, Actual = gamma
(venv) (base) carolina1650@MacBookPro DeepNetwork %
```

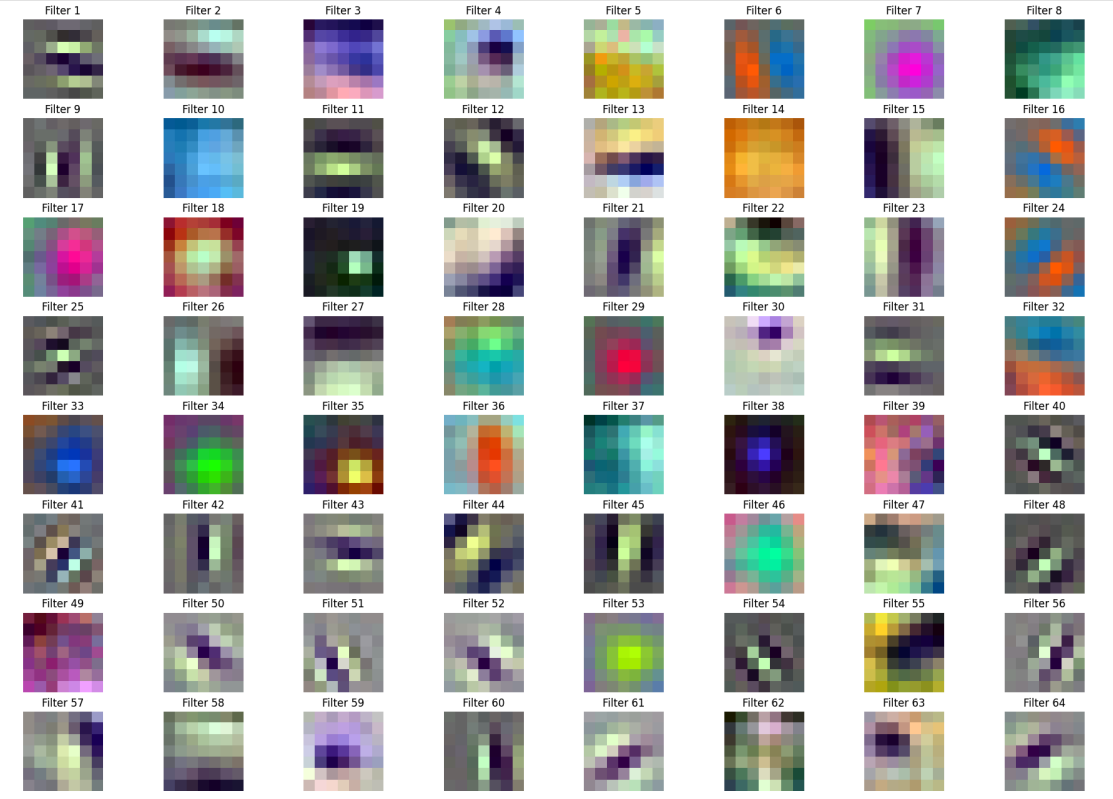
Extension 2: train on omega, epsilon and rho

The graph shows loss of fine-tuned neural network on the omega, epsilon and rho datasets. The loss starts somewhere around 1.15 and decreases gradually over 60 epochs.

```
Per-Class Accuracy:
  epsilon: 0.00% (0/3)
  omega: 100.00% (3/3)
  rho: 0.00% (0/3)

Overall Accuracy on Hand-Drawn Greek Letters: 33.33%

Detailed Predictions:
Image 1: Predicted = omega, Actual = epsilon
Image 2: Predicted = omega, Actual = epsilon
Image 3: Predicted = omega, Actual = epsilon
Image 4: Predicted = omega, Actual = omega
Image 5: Predicted = omega, Actual = omega
Image 6: Predicted = omega, Actual = omega
Image 7: Predicted = omega, Actual = rho
Image 8: Predicted = omega, Actual = rho
Image 9: Predicted = epsilon, Actual = rho
(venv) (base) carolina1650@MacBookPro DeepNetwork %
```

Extension 4: There are many pre-trained networks available in the PyTorch package. Try loading one and evaluate its first couple of convolutional layers as in task 2.

Architecture of pre-trained model:



The image shows 64 filters from the 1st convolutional layer of a pre-trained ResNet-18 model. The filter shows a variety of patterns , colors and gradients, most filters show obvious edges or color patterns.

Final Project Design:

I would like to elaborate on this project:

- Image Recognition/Classification using Deep Neural Network
- Undecided but may use the Fashion MNIST dataset
- Reference:
    - https://www.kaggle.com/code/cdeotte/25-million-images-0-99757-mnist
    - https://www.kaggle.com/code/shivamb/cnn-architectures-vgg-resnet-inception-tl
    - https://www.kaggle.com/code/arunkumarramanan/awesome-cv-with-fashion-mnist-classification
    -