CS 5330 Project 5

Kevin Heleodoro

Northeastern University

April 3, 2024

Abstract

The focus of this assignment was to train a deep network to recognize digits using the MNIST dataset. After initial training and testing, the network was expanded to recognize alpha, beta, and gamma greek letters. As a last step, another neural network was developed using the MNIST Fashion dataset which showed the effects and modifying different dimensions of the network training process.

1 Build and train a network to recognize digits

a.

After loading the MNIST training and testing datasets I was able to print out some example digits to get a better understanding of the types of images the network would be working with. [2] [16] [14]

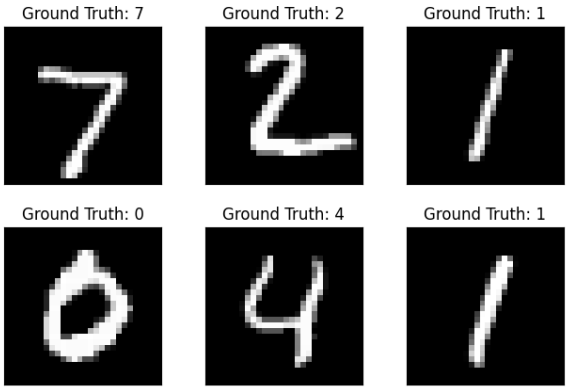


Figure 1: Example images from the test dataset

b.

The network was designed [1] [15] [4] to follow the specifications given in the assignment and consists of:

- A convolution layer with 10 5×5 filters.
- A max pooling layer with a 2×2 window and a ReLU function applied.
- A convolution layer with 20 5×5 filters.
- A dropout layer with a 0.5 dropout rate (50%).
- A max pooling layer with a 2×2 window and a ReLU function applied.

- A flattening operation followed by a fully connected Linear layer with 50 nodes and a ReLU function on the output.
- A final fully connected Linear layer with 10 nodes and the log_softmax function applied to the output.

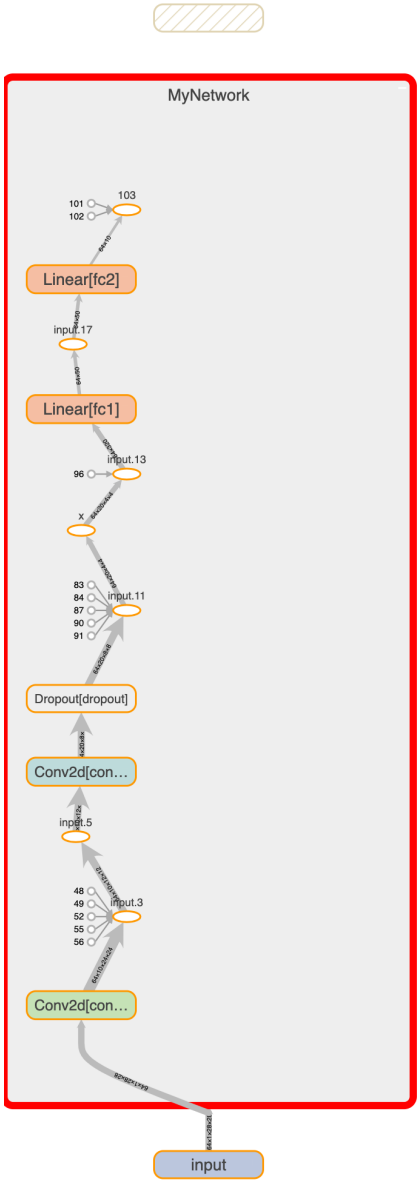


Figure 2: Neural Network diagram created using Tensorboard [18]

c.

Training the network on the MNIST digit dataset provided us with a consistent learning curve that supported a trend for increased accuracy after each epoch. After running a few examples I printed a plot of the network being trained over 8 epochs. The final epoch of this training/testing cycle resulted in an accuracy rating of 99%. [6] [7] [8]

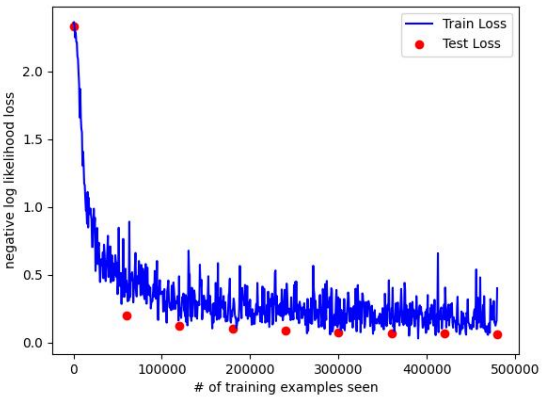
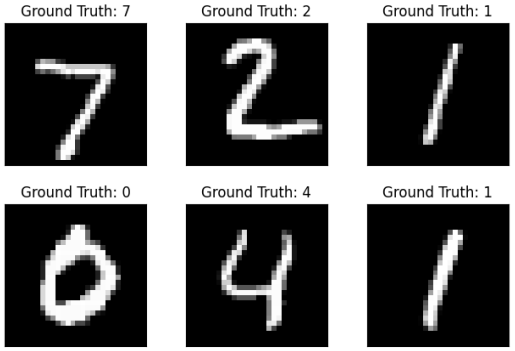


Figure 3: Learning curve of the neural network over 8 epochs.

d.

After the initial training cycle above, I tested the network in a new script by loading the state dictionary of the network. The network returned outstanding results on a small test sample of 10 images. [7]



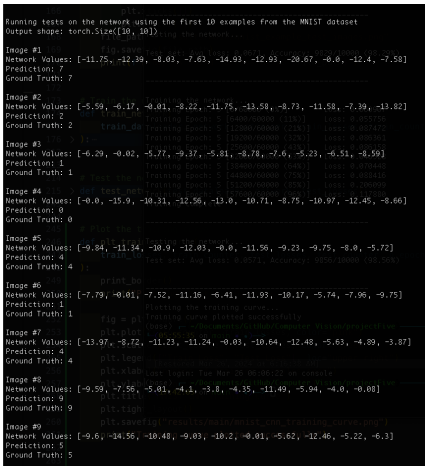
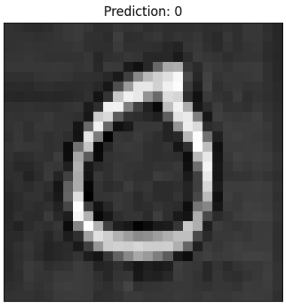


Figure 4: Results from testing trained neural network on sample images

e.

The final step of this task was to test the trained network on custom hand-written digits. I used the ImageMagick [3] tool for image formatting to follow the expected size requirements of 28x28. The first round of testing produced poor results, but after inverting the intensities of the images and increasing sharpness I began to get better results. The final modification I made was to follow PyTorch's guidance and switch to using *torchvision.transforms.v2* for the transforms applied to the dataset. [11] [12] [10]



(a) Example of custom test image

```
Running tests using custom images ...

--------------------------------------------------

Loading custom images from data/original
Number of custom images: 10
Image #1
Network Values: [-2.22, -2.47, -2.32, -2.16, -2.51, -2.34, -2.48, -2.4, -1.94, -2.33]
Prediction: 8
Ground Truth: 0

Image #2
Network Values: [-2.18, -2.23, -2.34, -2.44, -2.19, -2.49, -2.6, -2.25, -1.94, -2.54]
Prediction: 8
Ground Truth: 1

Image #3
Network Values: [-2.2, -2.32, -2.34, -2.0, -2.58, -2.26, -2.69, -2.44, -2.08, -2.3]
Prediction: 3
Ground Truth: 2

Image #4
Network Values: [-2.3, -2.5, -2.36, -2.34, -2.34, -2.3, -2.54, -2.44, -1.72, -2.44]
Prediction: 8
Ground Truth: 3

Image #5
Network Values: [-2.26, -2.53, -2.27, -2.39, -2.31, -2.56, -2.57, -2.41, -1.69, -2.36]
Prediction: 8
Ground Truth: 4

Image #6
Network Values: [-2.27, -2.46, -2.38, -2.18, -2.44, -2.36, -2.79, -2.39, -1.82, -2.23]
Prediction: 8
Ground Truth: 5

Image #7
Network Values: [-2.21, -2.39, -2.5, -2.09, -2.41, -2.29, -2.54, -2.35, -2.11, -2.24]
Prediction: 3
Ground Truth: 6

Image #8
Network Values: [-2.22, -2.31, -2.33, -2.42, -2.34, -2.56, -2.54, -2.2, -1.86, -2.45]
Prediction: 8
Ground Truth: 7

Image #9
Network Values: [-2.16, -2.46, -2.45, -2.35, -2.32, -2.41, -2.49, -2.38, -1.87, -2.29]
Prediction: 8
Ground Truth: 8

Image #10
Network Values: [-2.27, -2.37, -2.34, -2.4, -2.27, -2.56, -2.62, -2.23, -1.77, -2.46]
Prediction: 8
Ground Truth: 9
```

(b) Initial round of predictions: 1/10 correct

```
Number of custom images: 10
Image #1
Network Values: [-1.2, -3.29, -2.67, -1.76, -3.46, -2.49, -3.13, -3.38, -2.32, -2.06]
Prediction: 0
Ground Truth: 0

Image #2
Network Values: [-2.34, -1.69, -2.05, -1.98, -3.17, -2.87, -3.05, -3.02, -1.56, -3.1]
Prediction: 8
Ground Truth: 1

Image #3
Network Values: [-2.8, -2.69, -1.39, -2.34, -2.23, -3.23, -3.03, -2.6, -1.57, -3.03]
Prediction: 2
Ground Truth: 2

Image #4
Network Values: [-3.42, -3.68, -3.34, -0.74, -3.71, -1.95, -3.32, -3.62, -2.01, -2.71]
Prediction: 3
Ground Truth: 3

Image #5
Network Values: [-3.96, -2.73, -3.19, -2.11, -1.81, -2.22, -3.74, -1.83, -1.75, -2.08]
Prediction: 8
Ground Truth: 4

Image #6
Network Values: [-3.37, -3.9, -2.79, -1.99, -3.86, -1.01, -2.83, -3.83, -1.4, -3.35]
Prediction: 5
Ground Truth: 5

Image #7
Network Values: [-2.49, -3.81, -3.11, -3.08, -2.89, -2.06, -1.13, -4.26, -1.43, -3.07]
Prediction: 6
Ground Truth: 6

Image #8
Network Values: [-2.59, -2.5, -2.41, -2.04, -2.95, -2.74, -3.74, -1.76, -1.59, -2.25]
Prediction: 8
Ground Truth: 7

Image #9
Network Values: [-2.8, -3.59, -2.81, -2.12, -3.91, -2.26, -2.88, -3.77, -0.78, -2.67]
Prediction: 8
Ground Truth: 8

Image #10
Network Values: [-2.81, -1.99, -2.63, -1.97, -2.43, -2.98, -3.85, -1.89, -1.83, -2.13]
Prediction: 8
Ground Truth: 9
```

(c) Final round of predictions: 6/10 correct

2 Examine your network

a.

For this task I used the same trained network to create plots of the first convolutional layer to visualize the ten filters that comprise it.

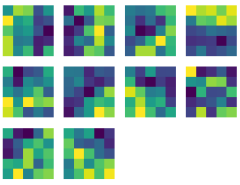


Figure 6: Filters within the first Convolutional layer of the network

Applying the filters to the sample image gives us a range of different variations of coloring and shading. The differences between the filtered images coincide with the filter being applied to them in shading pattern.

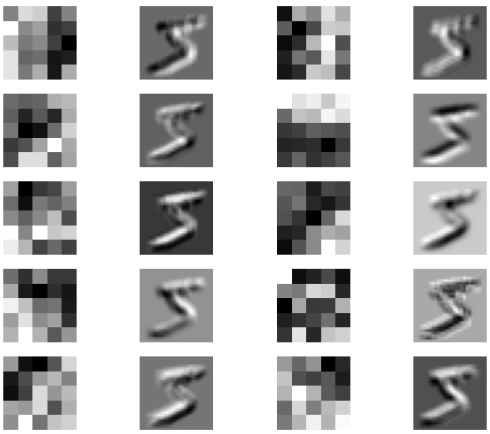


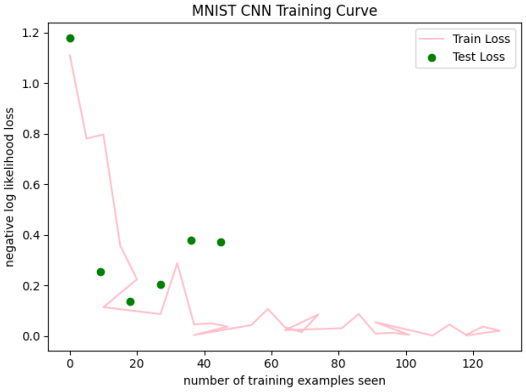
Figure 7: Filters applied to the first training sample image

3 Transfer Learning on Greek Letters

Training on the greek letters presented some difficulty at first and I needed to adjust my network parameters. I increased my learning rate to 0.1 and momentum to 0.8. After more testing I was able to achieve 100% on the test data with 7 epochs. On average the network was achieving 7 or 8 out of a possible 9 correct answers. The training curve showed no consistency or pattern when I printed the plot. A link to the training and test images can be found on my GitHub.

```
Loading the trained model...
Before: MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (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)
)

------------------------------------------------

Freezing the network weights...

------------------------------------------------

Replacing the last layer with a new layer with three nodes...
After: MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (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)
)
```

(a) Network before and after modifications



(b) Learning curve with no pattern or trend

```
Train Epoch: 7 [0/27 (0%)]      Loss: 0.008647
Train Epoch: 7 [5/27 (17%)]     Loss: 0.038618
Train Epoch: 7 [10/27 (33%)]    Loss: 0.156512
Train Epoch: 7 [15/27 (50%)]    Loss: 0.253824
Train Epoch: 7 [20/27 (67%)]    Loss: 0.027097
Train Epoch: 7 [10/27 (83%)]    Loss: 0.010680
Training complete

Test set: Average loss: 0.0885, Accuracy: 9/9 (100.00%)
```

(c) Network results on test dataset

4 Design your own experiment

For this task I decided to create another neural network, but this time I would be using the MNIST Fashion dataset. [13] [17] [5] [9] This network would be used to evaluate the effects of changing different aspects on the results produced. Before deciding on the specific dimensions to perform the evaluation on I ran a training and testing loop with the following parameters:

- Epochs: 5
- Batch Size: 4
- Mean: 0.5
- Standard Deviation: 0.5
- Learning Rate: 0.01
- Momentum: 0.5
- Optimizer: SGD

The initial results were: 'Test set (Epoch 5) - Avg loss: -11.2641, Accuracy: 8894/10000 (88.94%)'.

I did further experimentation with changing the mean, standard deviation, and optimizers. I then decided to automate the evaluation of adjusting the number of epochs, dropout rates in the Dropout layer, and the batch size for training data. My initial hypothesis was that as each of these increases the accuracy would also increase.

The full results of the experiment can be found in my GitHub repo. What I found was that increasing these dimensions one at a time did not have the effect that I thought it might. The highest accuracy seemed to range around 87-88%.

5 Reflection

This assignment has shown me the flexibility that neural networks have with how they interpret training data and how they can be evaluated. It was interesting to see how even slight variations in the input images could cause a reliable network to return sub-optimal results in recognition.

References

[1] *Build the Neural Network*. https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html. Accessed: 2024-04-02.

[2] *Data Loading and Processing Tutorial*. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html. Accessed: 2024-04-02.

[3] ImageMagick Studio LLC. *ImageMagick Download*. https://imagemagick.org/script/download.php. Accessed: 2024-03-12. 2024.

[4] G. Koehler. *PyTorch MNIST Example*. https://nextjournal.com/gkoehler/pytorch-mnist. Accessed: 2024-04-02.

[5] Pankaj Kumar. *Fashion MNIST with PyTorch - 93% Accuracy*. https://www.kaggle.com/code/pankajj/fashion-mnist-with-pytorch-93-accuracy. Accessed: 2024-03-12. 2024.

[6] PyTorch. *Optimization Tutorial*. https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html. Accessed: 2024-03-12. 2024.

[7] PyTorch. *Saving & Loading Model for Inference*. https://pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html. Accessed: 2024-03-12. 2024.

[8] PyTorch. *torch.nn*. https://pytorch.org/docs/stable/nn.html. Accessed: 2024-03-12. 2024.

[9] PyTorch. *torch.optim.Adam*. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html. Accessed: 2024-03-12. 2024.

[10] PyTorch. *torchvision Transforms*. https://pytorch.org/vision/stable/transforms.html. Accessed: 2024-03-12. 2024.

[11] PyTorch. *torchvision.datasets.ImageFolder*. https://pytorch.org/vision/main/generated/torchvision.datasets.ImageFolder.html. Accessed: 2024-03-12. 2024.

[12] PyTorch. *torchvision.transforms.functional.adjust$_s$harpness*. https://pytorch.org/vision/main/generated/torchvision.transforms.functional.adjust_sharpness.html. Accessed: 2024-03-12. 2024.

[13] PyTorch. *Training a Classifier*. https://pytorch.org/tutorials/beginner/introyt/trainingyt.html. Accessed: 2024-03-12. 2024.

[14] *PyTorch Quickstart Tutorial*. https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html. Accessed: 2024-04-02.

[15] *torch.nn — PyTorch 1.x documentation*. https://pytorch.org/docs/stable/nn.html. Accessed: 2024-04-02.

[16] *torchvision.datasets.MNIST*. https://pytorch.org/vision/stable/generated/torchvision.datasets.MNIST.html. Accessed: 2024-04-02.

[17] Ayşegül Tüzün. "Fashion MNIST Data Training Using PyTorch". In: *Medium* (2024). Accessed: 2024-03-12.

[18] *Visualizing Models, Data, and Training with TensorBoard*. https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html. Accessed: 2024-04-02.