Project 5: Recognition using Deep Networks

CS 5330: Pattern Recognition and Computer Vision

Arjun Rajeev Warrier

Abstract

In project 5, the goal is to build, train, analyze, and modify a deep network for a recognition task. This network is run on the MNIST dataset, a greek letter dataset and the MNIST fashion dataset. Through this project, transfer learning and hyper-parameter variation effects on the results have been observed.

Experiment

The functionalities that were programmed as part of this project has been described below. This is a brief overview with short explanations and images that demonstrate the effect of these codes.

Project 5: Recognition using Deep Networks CS 5330: Pattern Recognition and Computer Vision Arjun Rajeev Warrier

Abstract

In project 5, the goal is to build, train, analyze, and modify a deep network for a recognition task. This network is run on the MNIST dataset, a greek letter dataset and the MNIST fashion dataset. Through this project, transfer learning and hyper-parameter variation effects on the results have been observed.

Experiment

The functionalities that were programmed as part of this project has been described below. This is a brief overview with short explanations and images that demonstrate the effect of these codes.

1. Build and train a network to recognize digits

The MNIST digit data consists of a training set of 60k 28x28 labeled digits and a test set of 10k 28x28 labeled digits. These datasets were loaded using the dataloader function in torchvision. The first 6 example digits are shown in Fig. 1.

In order to make the code repeatable, the random seed for the torch package was set to torch.manual seed(42).A network model was built as shown in Fig.2.

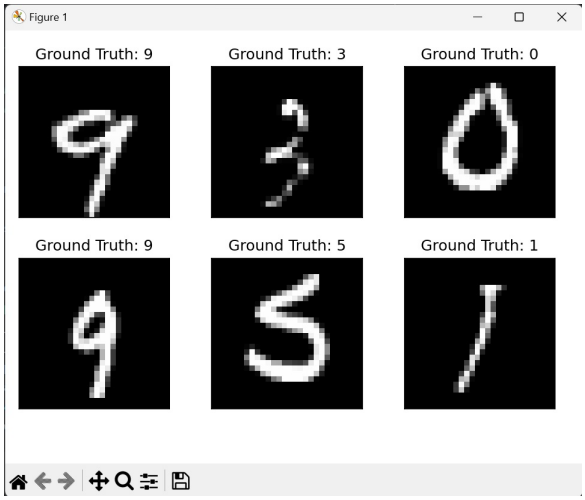

Figure 1: Task 1A : First six example digits.

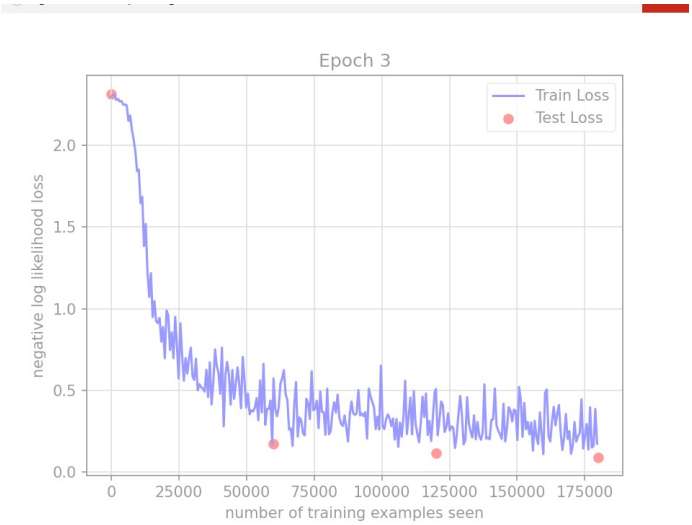

Figure 2: Task 1D : Accuracy scores for training and testing.

The model was trained for 5 epochs and the model was evaluated on both the training and test data after each epoch. Results can be seen in Fig. 3. Both accuracies improve over time as expected and can be seen from the descending plot as the number of examples seen increase.
The network was then saved with its parameters(weights and biases) to an external location /results(will be recalled). This saved network is loaded in an another program and then run on 9 examples. The predicted result has been shown against the corresponding image tested in Fig.4.

To further test the network, nine new handwritten examples were tested in the saved network. It gave near perfect results after just 5 epochs. This can be seen in Fig.6.

```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(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)
)
```

Figure 3: Task 1C : Network Built.

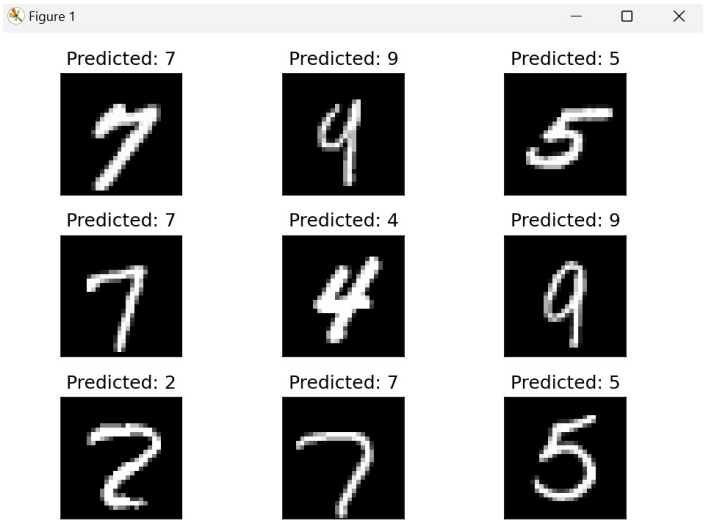

Figure 4: Task 1F : Printed values and the plot of the first 9 digits.

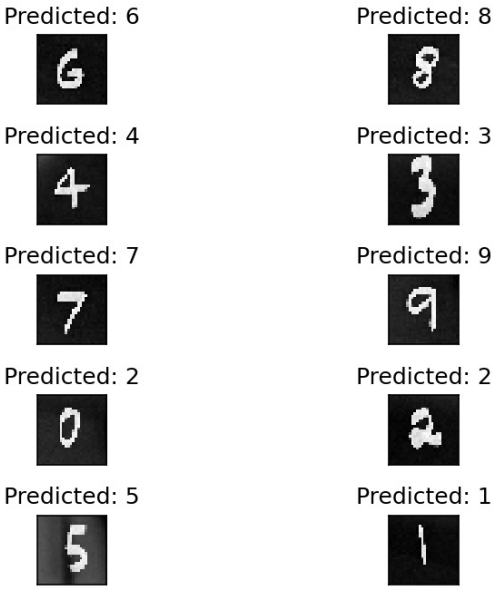

Figure 5: Task 1G : Tested on custom hand drawn digits.

2. Examine the network

The second task is to examine your network and analyze how it processes the data. The trained network was loaded into a seperate program and the first layer weights were obtained. The ten filters, each of size 5x5 were then visualized as seen in Fig.7.

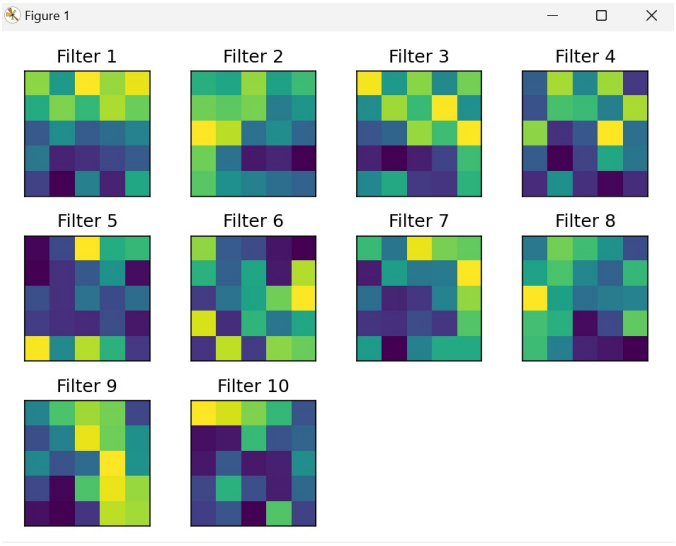

Figure 6: Task 2A: Visualize the ten filters

The effects of these filters on an input image was then visualized on a single input. This has been shown in Fig.7.

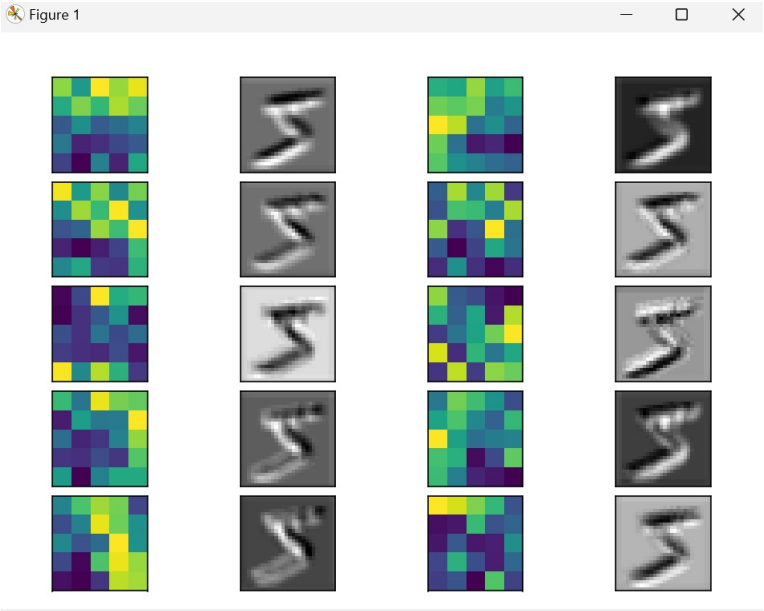

Figure 7: Task 2B: Input through 10 filters.

3. Transfer Learning on Greek Letters

The previously trained model was again loaded. The final linear layer was then adjusted for 6 possible output classes: alpha, beta, gamma, eta, sigma and chi. With just 20 epochs, the model was able to reduce training losses to a considerable amount. A plot of this training error has been shown in Fig.8 and the modified network has been printed out in Fig.9. Additional examples were also included as test inputs and the system showed 80

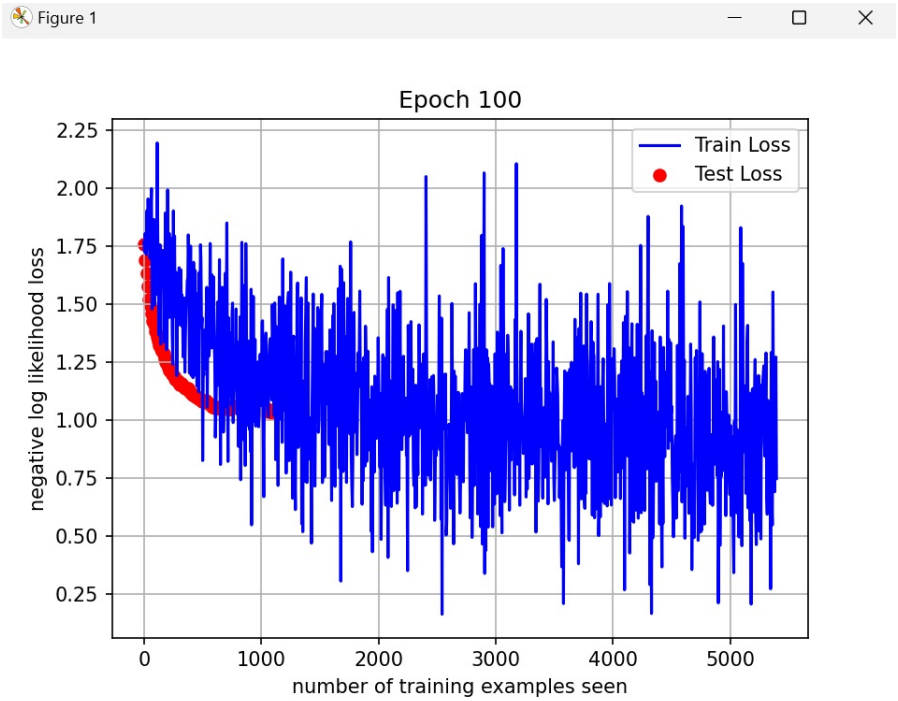

Figure 8: Training Losses after 100 epochs.

```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=6, bias=True)
)
```

Figure 9: Modified network.

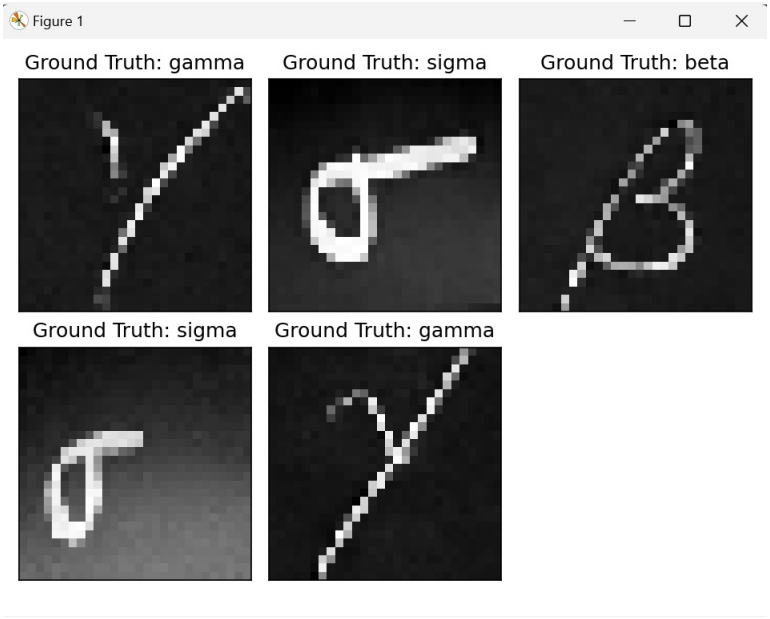

Figure 10: Examples of training inputs.

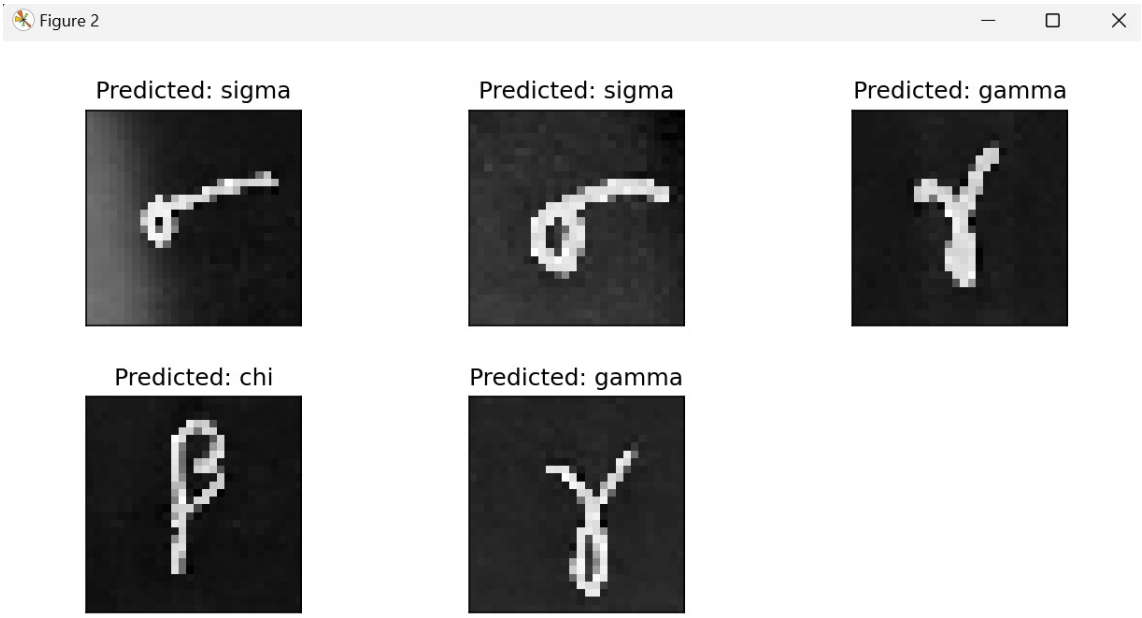

Figure 11: Test after 20 iterations.

4. Variation of hyper-parameters

A new network was trained with multiple combinations of hyper-parameters. The variation is described below:

1. Number of convolution layers: Increasing the number of convolutional layers can increase the representational power of the network, allowing it to learn more complex features. However, too many layers can lead to overfitting and slower training time.

2. Convolution filter size: Increasing the size of the convolutional filters can increase the receptive field of the network and enable it to capture larger patterns in the input. However, larger filters also require more computation, which can slow down training.

3. Number pf convolution filters: Increasing the number of convolutional filters can increase the number of learned features in each layer, making the network more expressive. However, too many filters can lead to overfitting and longer training time.

4. Number of epochs: Increasing the number of epochs can allow the network to better fit the training data and improve its performance. However, too many epochs can lead to overfitting and longer training time.

The results of the difference combinations on separate a separate test set can be seen in Fig.10.

```
Accuracy for hyperparameters (2, 1, 16): 79.96%, Time taken: 15.22 seconds
Accuracy for hyperparameters (2, 1, 32): 77.01%, Time taken: 21.89 seconds
Accuracy for hyperparameters (2, 2, 16): 78.01%, Time taken: 15.48 seconds
Accuracy for hyperparameters (2, 2, 32): 81.34%, Time taken: 25.00 seconds
Accuracy for hyperparameters (2, 3, 16): 81.33%, Time taken: 17.29 seconds
Accuracy for hyperparameters (2, 3, 32): 80.35%, Time taken: 24.88 seconds
Accuracy for hyperparameters (3, 1, 16): 65.04%, Time taken: 15.65 seconds
Accuracy for hyperparameters (3, 1, 32): 64.74%, Time taken: 21.91 seconds
Accuracy for hyperparameters (3, 2, 16): 73.97%, Time taken: 16.78 seconds
Accuracy for hyperparameters (3, 2, 32): 74.12%, Time taken: 25.99 seconds
Accuracy for hyperparameters (3, 3, 16): 75.29%, Time taken: 17.73 seconds
Accuracy for hyperparameters (3, 3, 32): 74.88%, Time taken: 25.65 seconds
Accuracy for hyperparameters (4, 1, 16): 26.10%, Time taken: 16.74 seconds
Accuracy for hyperparameters (4, 1, 32): 47.37%, Time taken: 23.88 seconds
Accuracy for hyperparameters (4, 2, 16): 67.26%, Time taken: 17.98 seconds
Accuracy for hyperparameters (4, 2, 32): 69.90%, Time taken: 27.25 seconds
Accuracy for hyperparameters (4, 3, 16): 75.14%, Time taken: 17.57 seconds
Accuracy for hyperparameters (4, 3, 32): 70.18%, Time taken: 26.81 seconds
```

Figure 12: Task 4: Accuracies and time taken for different combinations with 1 epoch each of training. hyperparams are in the frm (number of conv layers, filter size, number of conv filters) followed by accuracy and computation times.

8. Extensions

As extensions, I have added three more greek letters to my Greek Letter identification model: eta, sigma and chi. I have trained and tested the model on different input sets of sixes similar to the alpha, beta and gamma collection. The result has been shown along with the rest of the task.
I have also tested the system for four parameters instead of the three hyper-parameter variations requested.

Reflections

Some key takeaways for this project were:

- The structure and definition of a neural network was new. The effect of varying each parameter and the consequences of overfitting were also visibly observed in this project.

- Transfer learning and its applications are now more approachable. Just how much it speeds up the training is clearly visible thorugh testing on other digit datasets.

- This project gives an idea of how to implement deep learning models for identification and for tracking purposes.

References

[1] https://nextjournal.com/gkoehler/pytorch-mnist

[2] https://pytorch.org/documentation