Recognition using Deep Networks

Yuan Zhao

CS5330 40208: Pattern Recognition and Computer Vision, Project 5

April 1st, 2024

Project Overview

In this project, I aim to delve into the world of deep learning by constructing, training, and analyzing a deep neural network specifically designed for digit recognition tasks. Utilizing the widely acknowledged MNIST dataset, I have developed a neural network model that incorporates various layers, including convolutional, max pooling, dropout, and linear layers.

This endeavor is not just limited to building and training the network; it also encompasses practical applications, testing its effectiveness on both standardized test images and actual handwritten digits. Moreover, I explore the intricacies of network analysis by examining the impact of individual layers and conduct an experiment with transfer learning, this time using Greek letters.

The project reaches its climax with a series of experiments that tweak different network parameters on the MNIST Fashion dataset, with the goal of enhancing the network's performance and training efficiency. Through this comprehensive project, I aim to deepen my understanding of neural networks and explore their potential in the field of image recognition.

Experiment and data Analysis.

Task 1: Build and train a network to recognize digits

1. Get the MNIST digit data set.

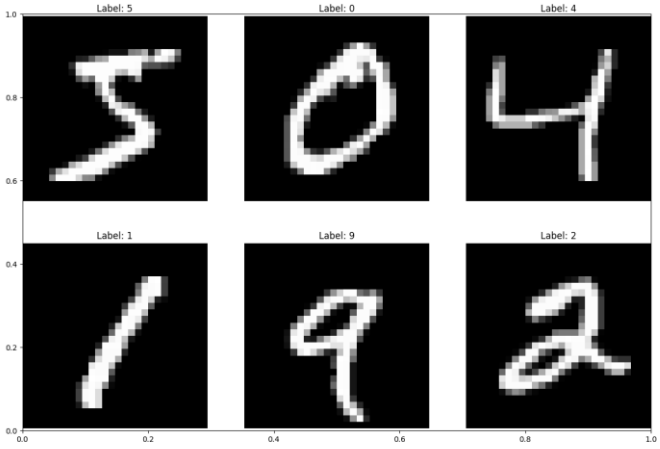

Fig 1. six images from MNIST train dataset

The following are the first six image examples from the test dataset, accompanied by their respective ground truth labels.

2. Build a network model.

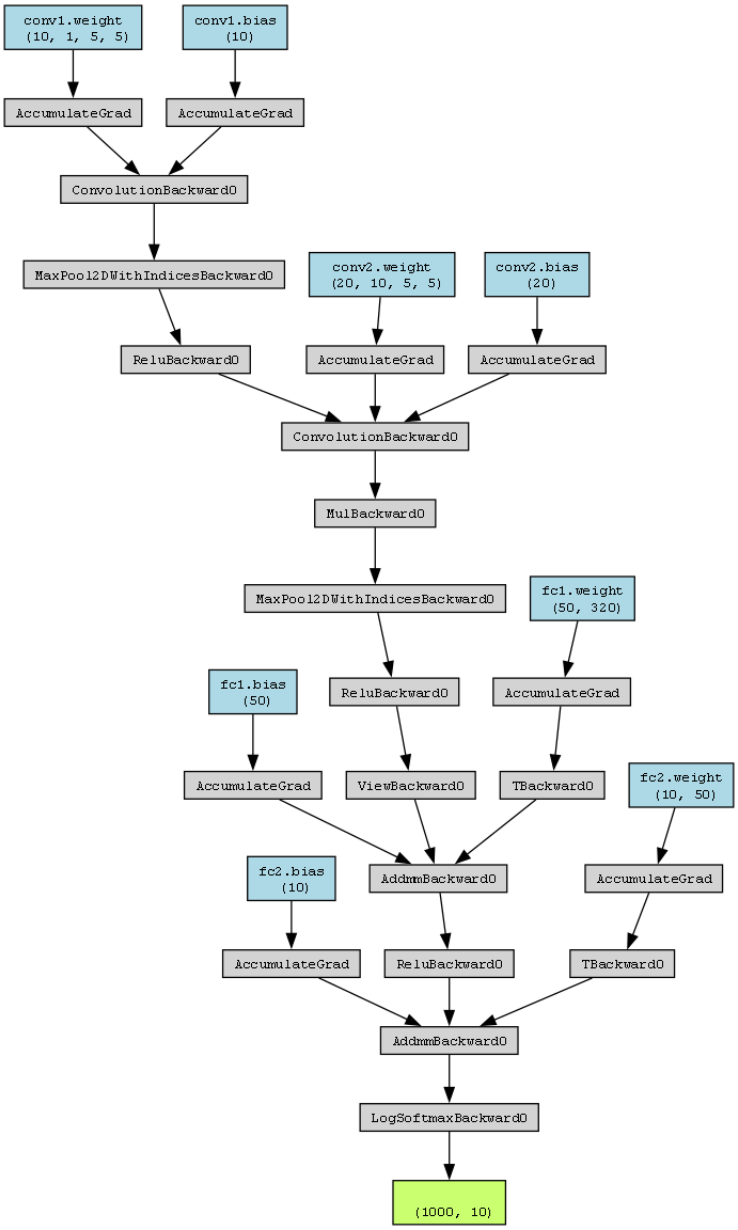

Fig 2. Diagram of the CNN(The diagram is drawn by plug-in *torchviz*)

3. Train the model.

Before the ten epochs of training, we run our test loop once before even starting the training with randomly initialized network parameters.

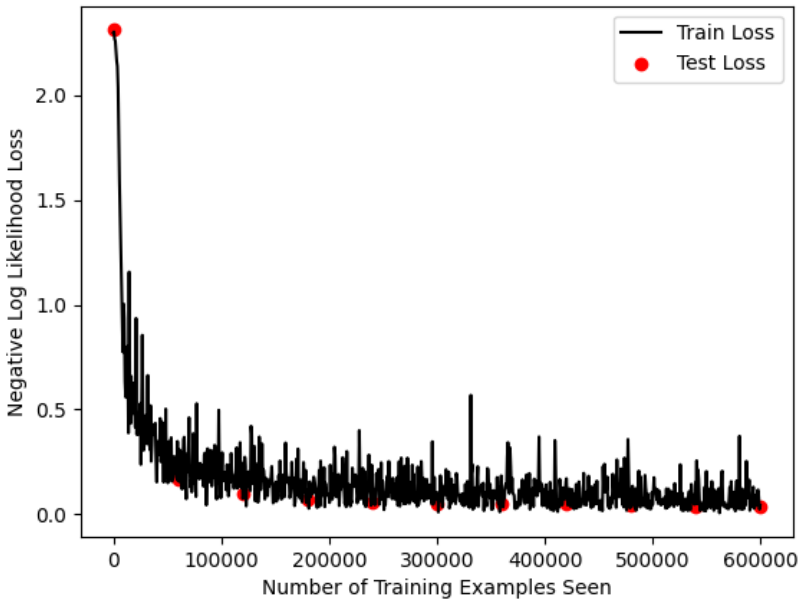

Fig 3. The train (shown in black line) and test (shown in red dots) error for 10 epochs

Test set: Avg. loss: 2.3130, Accuracy: 1319/10000 (13%)

Training Epoch: 1 Loss: 0.346922 [59520/60000 (99%)]
Test set: Avg. loss: 0.1657, Accuracy: 9488/10000 (95%)

Training Epoch: 2 Loss: 0.164133 [59520/60000 (99%)]
Test set: Avg. loss: 0.0971, Accuracy: 9690/10000 (97%)

Training Epoch: 3 Loss: 0.113983 [59520/60000 (99%)]
Test set: Avg. loss: 0.0690, Accuracy: 9785/10000 (98%)

Training Epoch: 4 Loss: 0.233948 [59520/60000 (99%)]
Test set: Avg. loss: 0.0578, Accuracy: 9815/10000 (98%)

Training Epoch: 5 Loss: 0.089778 [59520/60000 (99%)]
Test set: Avg. loss: 0.0520, Accuracy: 9830/10000 (98%)

Training Epoch: 6 Loss: 0.182434 [59520/60000 (99%)]
Test set: Avg. loss: 0.0503, Accuracy: 9833/10000 (98%)

Training Epoch: 7 Loss: 0.157230 [59520/60000 (99%)]
Test set: Avg. loss: 0.0472, Accuracy: 9847/10000 (98%)

Training Epoch: 8 Loss: 0.035644 [59520/60000 (99%)]
Test set: Avg. loss: 0.0395, Accuracy: 9865/10000 (99%)

Training Epoch: 9 Loss: 0.148860 [59520/60000 (99%)]
Test set: Avg. loss: 0.0383, Accuracy: 9877/10000 (99%)

Training Epoch: 10 Loss: 0.040427 [59520/60000 (99%)]
Test set: Avg. loss: 0.0346, Accuracy: 9893/10000 (99%)

4. Read the network and run it on a test set.

Execute the model on the initial 10 samples in the test set. Display a plot where the corresponding labels are positioned above each subplot.

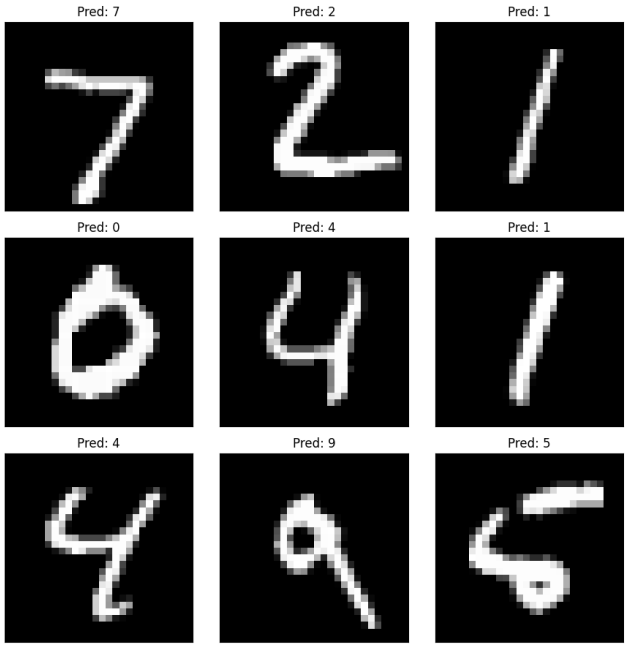

Fig 4. 9 digits from MNIST test dataset with their predictions

5. Test the network on new inputs.

The MNIST dataset comprises images of white digits against a black background. However, my new input images feature black digits on a white background. To preprocess them, I resized them to 28x28 size and inverted the intensity.

The following plot displays the outcome: each subplot showcases the digits after preprocessing alongside their predictions. While it accurately identifies most digits, it erroneously classifies the number 7 as the number 3, and number 8 as the number 5, due to their resemblance. The presence of a line through the writing style of the number 7 may lead to recognition errors.

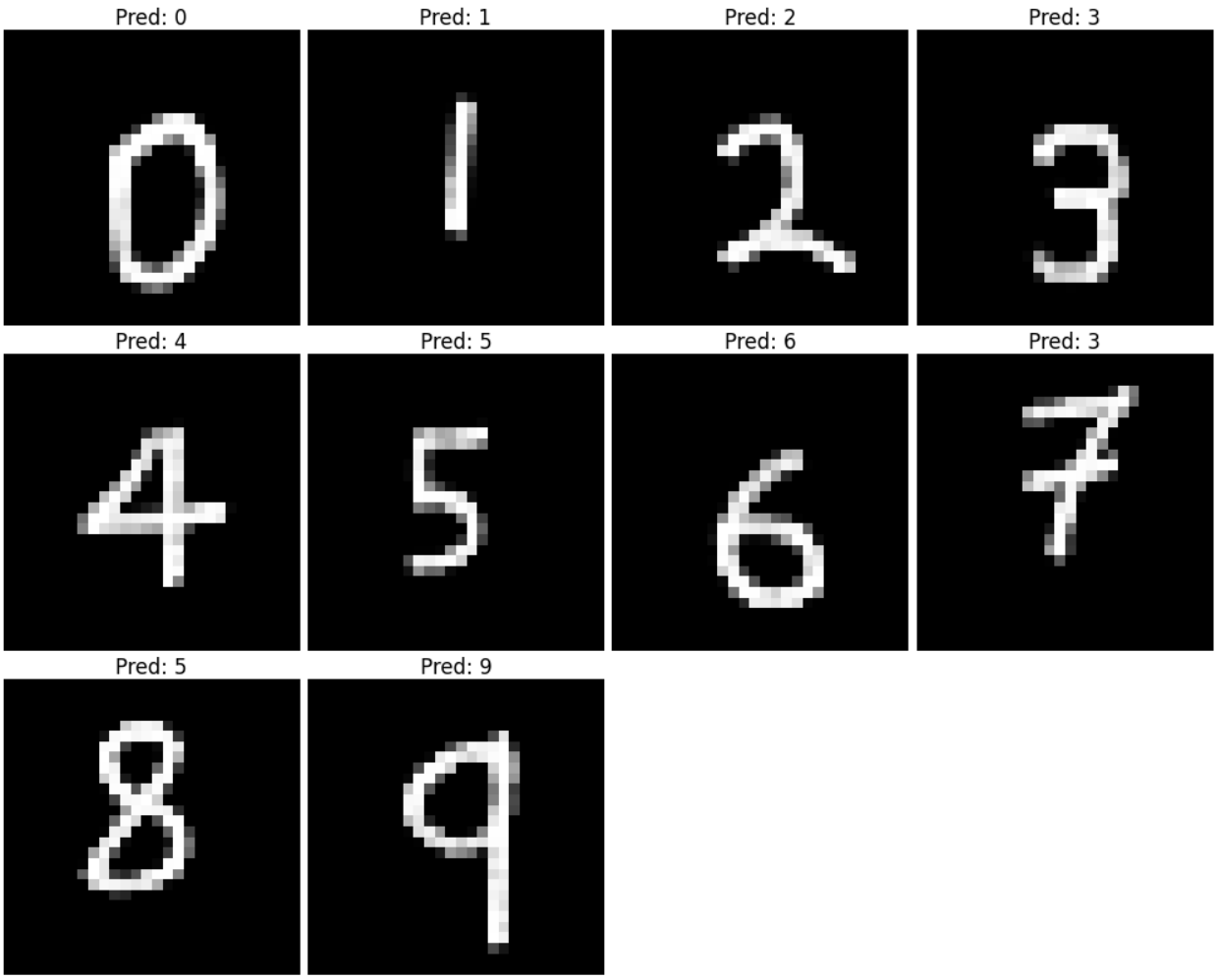

Fig 5. 10 new handwritten examples with their predictions, accuracy 8/10.

Below are the output values, where the highlighted values represent the absolute values and the minimum value is close to zero. The underlined value indicates the true label. Despite this, the error in digit recognition persists, with the second absolute value also being close to zero.

Image 1, output values: [-0.00, -16.06, -13.80, -17.39, -21.07, -8.13, -11.60, -9.95, -10.25, -13.87], prediction: 0, true label: 0

Image 2, output values: [-8.61, -0.02, -7.26, -7.85, -4.39, -6.57, -6.15, -8.84, -6.02, -7.18], prediction: 1, true label: 1

Image 3, output values: [-9.67, -3.58, -0.03, -8.38, -10.87, -10.50, -15.83, -5.92, -10.41, -9.75], prediction: 2, true label: 2

Image 4, output values: [-11.25, -14.51, -9.19, -0.08, -11.99, -2.85, -12.82, -10.46, -6.72, -4.25], prediction: 3, true label: 3

Image 5, output values: [-18.16, -10.38, -9.89, -10.62, -0.00, -11.27, -11.82, -11.07, -9.88, -9.69], prediction: 4, true label: 4

Image 6, output values: [-16.36, -13.66, -10.61, -5.70, -8.00, -0.27, -13.82, -12.01, -4.71, -1.51], prediction: 5, true label: 5

Image 7, output values: [-10.76, -13.90, -6.95, -5.75, -8.93, -7.13, -0.44, -18.03, -1.04, -13.92], prediction: 6, true label: 6

Image 8, output values: [-11.85, -2.61, -4.83, -0.63, -10.67, -6.15, -14.56, -1.78, -1.69, -3.52], prediction: 3, true label: 7

Image 9, output values: [-6.72, -8.21, -2.66, -1.94, -4.91, -0.80, -6.59, -8.49, -1.63, -2.04], prediction: 5, true label: 8

Image 10, output values: [-23.83, -23.93, -16.44, -10.04, -7.38, -12.61, -28.79, -14.05, -9.40, -0.00], prediction: 9, true label: 9

Task 2: Examine your network

1. Analyze the first layer.

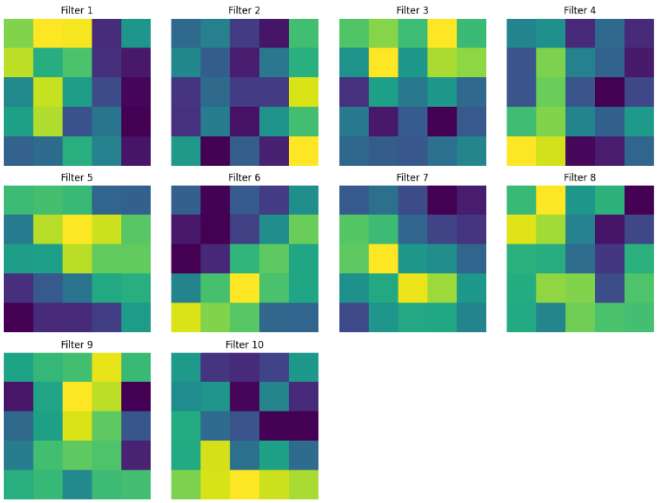

Fig 6. Filters of the first layer images

2. Show the effect of the filters.

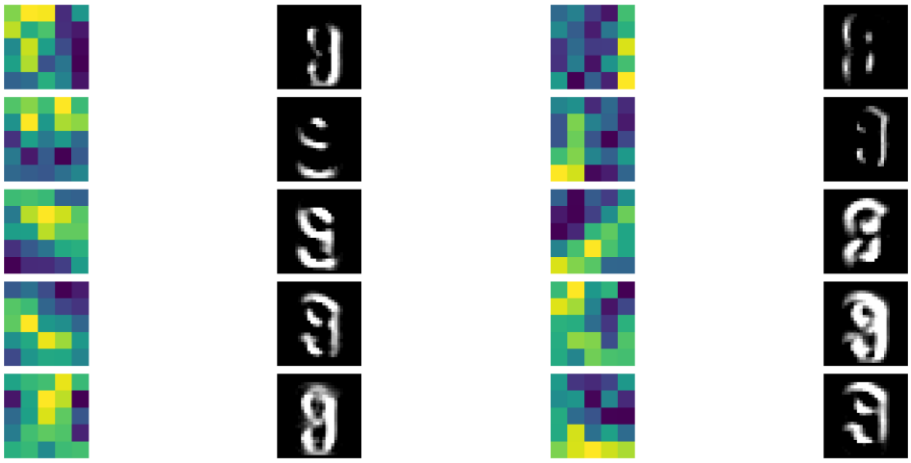

Fig 7. 10 filters from the first layer and their effect on an example image

In terms of the kernels, bright pixels represent large positive values, while dark pixels represent large negative values. Most of these filters appear to be adept at detecting edge features within the image. For instance, consider the tenth filter and its corresponding output: the upper and lower segments of the digit "9" exhibit nearly horizontal edges, emphasized as dark lines. Surprisingly, the filter displays brightness at the top and darkness at the bottom, akin to the Sobel Y filter, which typically highlights horizontal edges.

Nevertheless, absolutes aren't always applicable. Examining the result of the middle digit in the second column reveals that the digit "9" appears darker in contrast to the white background. In this scenario, there isn't a discernible angle or edge explicitly concentrated upon; instead, the feature it learns seems rather random and challenging to describe.

Task 3: Learning on Greek Letters

1. The updated network has the following structure:

```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_dropout): Dropout2d(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)
```

2. The Training loss

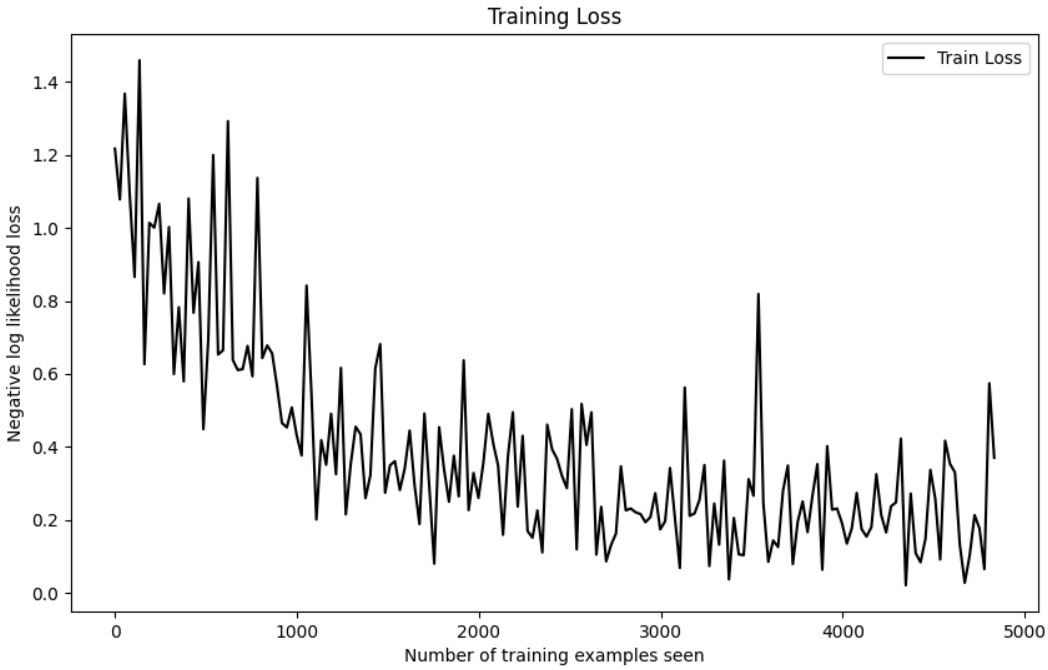

Fig 7. Training error for the Greek data over 30 epochs

When the batch size is set to 5, it typically takes around 20 to 25 epochs to nearly perfect the identification process. However, beyond 20 epochs, the learning curve tends to plateau, indicating that further epochs yield minimal improvements in performance.

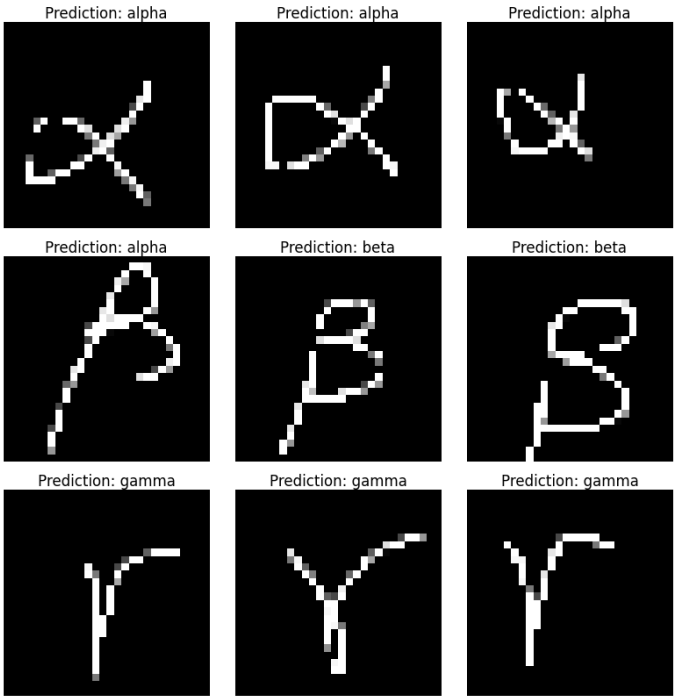

Fig 8. Testing the network on handwritten data; accuracy is 89% (8/9)

Task 4. Design your own experiment

1. Develop a plan.

During this experimental phase, the primary goal is to investigate the influence of modifying various aspects of the deep network architecture on the MNIST Fashion dataset. Given that the MNIST Fashion dataset is more intricate than the standard MNIST digits dataset, it provides an excellent opportunity to observe the repercussions of these architectural alterations. The experiment focuses on exploring the following dimensions:

- The number of convolution layers: 1, 2, and 3.
- The number of convolution filters in a layer: 10, 40, and 80.
- The dropout rates of the Dropout layer: 0.25, 0.5, and 0.75.

The experimental approach employs a linear search strategy, where two parameters remain constant while the third parameter is varied. This process is iterated in a round-robin manner, with the variable parameter changing sequentially. The intention is to automate this procedure to systematically evaluate 27 (3 x 3 x 3) network variations efficiently.

2. Predict the results.
   - **Convolution layers**: Increasing the number of convolution layers (depth) increases the model's capacity to learn complex features. More layers increase computational cost and training time. More layers can lead to a higher risk of overfitting, especially with smaller datasets, because the model has more parameters to adjust.
   - **Convolution filters in a layer**: Increasing the number of filters increases the model’s capacity and computational complexity, potentially leading to longer training times.
   - **Dropout rate**: Too high a dropout rate might lead to underfitting, where the model fails to learn significant patterns in the data. Too low a dropout rate might be insufficient to prevent overfitting.

3. Actual Results.

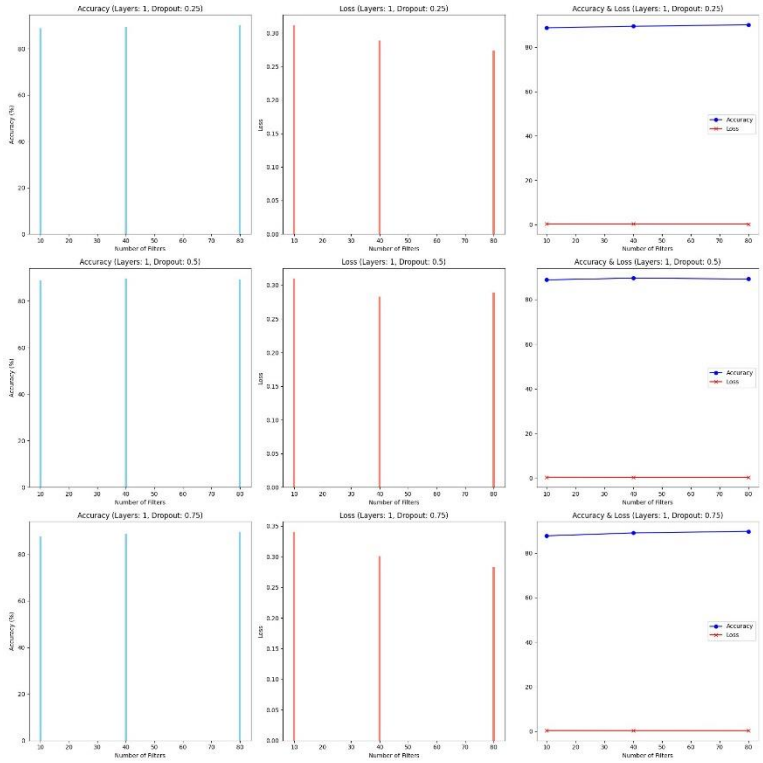

Fig. 9: For one convolution layer, with different dropout rates and filters

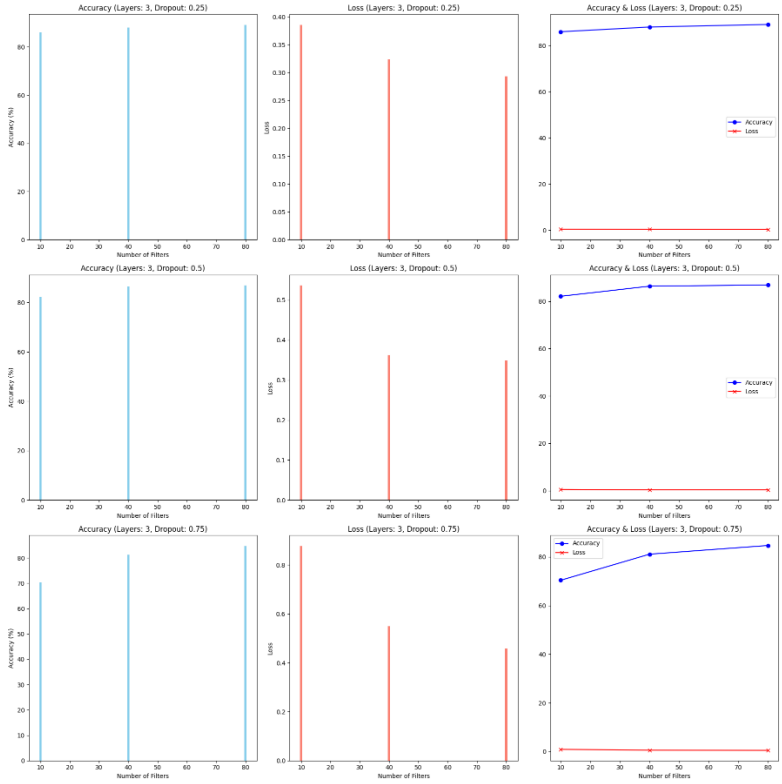

Fig. 10: For two convolution layers, accuracy and loss, with different dropout rates and filters

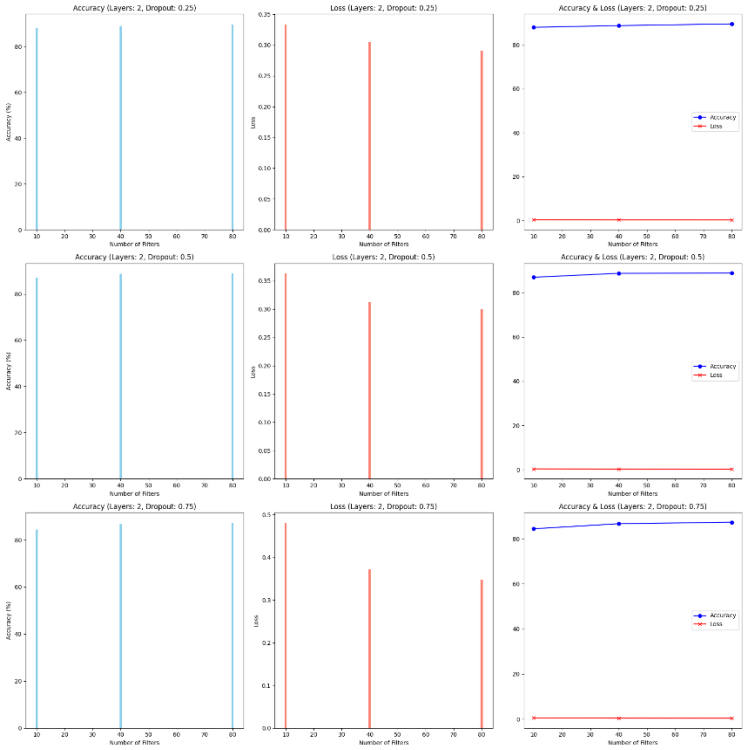

Fig. 11: For two convolution layers, accuracy and loss, with different dropout rates and filters

4. Data analysis and discussion

A. Fixing the Number of Layers

When layers = 1:

- Filters: Increasing from 10 to 80 filters consistently improves accuracy and decreases loss, showing that more filters help the model to learn better when the depth is limited.

- Dropout: A moderate dropout rate of 0.5 seems to give a good balance between accuracy and overfitting. However, a dropout rate of 0.75 appears to degrade performance, indicating potential underfitting.

When layers = 2:

- Filters: As with one layer, more filters (80 vs 10) generally improve accuracy. However, the increase in accuracy is not as significant as when going from one to two layers, which may indicate diminishing returns with added complexity.

- Dropout: Increasing the dropout rate results in a decrease in accuracy, more notably than with one layer, perhaps because the additional layer already provides some regularization effect.

When layers = 3:

- Filters: The accuracy gain from increasing filters is less consistent. The best performance is achieved with 80 filters and a dropout rate of 0.25, but the difference in accuracy compared to lower filter counts is smaller than in shallower networks.

- Dropout: High dropout rates substantially reduce accuracy, suggesting that too much regularization in a deeper network can hinder learning.

B. Fixing the Number of Filters

When filters = 10:

- Layers: Increasing the number of layers from 1 to 3 generally decreases accuracy, suggesting that with few filters, additional layers may not be beneficial and can lead to overfitting or other optimization difficulties.

- Dropout: A dropout rate of 0.5 or less seems beneficial, but 0.75 leads to a substantial decrease in accuracy across all layer configurations, indicating that too much regularization is counterproductive for such a small number of filters.

When filters = 40:

- Layers: There is an initial increase in accuracy when adding a second layer, but a third layer decreases accuracy again. This suggests there is an optimal depth for a given filter size.

- Dropout: Again, a moderate dropout rate provides the best results. Too high a dropout rate with more layers seems to lead to underfitting.

When filters = 80:

- Layers: The highest accuracy is achieved with a single layer, but adding more layers doesn't drastically reduce performance. This suggests that a higher number of filters can support deeper architectures better than a lower number of filters.

- Dropout: A moderate dropout rate of 0.5 in deeper networks (2 or 3 layers) doesn't result in as large a drop in accuracy as a 0.75 rate, which consistently lowers performance.

C. Fixing the Dropout Rate

When dropout = 0.25:

- Layers: The accuracy tends to decrease as more layers are added, with the best performance at one layer. This indicates that a low dropout rate is not enough to counteract the potential overfitting associated with increased depth.

- Filters: Increasing filters generally improves performance across all numbers of layers, suggesting that at this dropout rate, the network benefits from more complex feature extraction capabilities.

When dropout = 0.5:

- Layers: The trend is similar to a 0.25 dropout rate, but the decrease in accuracy with additional layers is less pronounced, indicating better generalization.

- Filters: More filters improve accuracy in shallow networks, but in deeper networks (3 layers), the increase in filters does not result in a significant improvement in accuracy.

When dropout = 0.75:

- Layers: This high dropout rate leads to a significant decrease in accuracy, particularly as the number of layers increases, which can be attributed to excessive regularization leading to underfitting.

- Filters: Even with a high number of filters, the high dropout rate does not compensate for the loss of information due to the high dropout, resulting in lower performance.

D. Overall Conclusion:

- A single layer with a high number of filters and a low to moderate dropout rate seems to provide the best performance.

- As the number of layers increases, the benefit of adding more filters diminishes, and the optimal dropout rate tends to be lower to prevent underfitting.

- There is a complex interaction between depth, width (number of filters), and dropout rate that needs to be carefully balanced to achieve optimal performance.

Extensions

This extension part generated ten Gabor filters spanning different angles from 0 to 180, which were then manually designated as parameters for the initial convolutional layer. The ensuing results illustrate each Gabor filter alongside its respective output. Despite the 5x5 filter size containing only 25 pixels, too few for Gabor filters, I experimented with various configurations of lambda, gamma, etc., to ensure distinct orientations. Some filters extract horizontal edges, as evident in the third one in the first column, while others capture vertical edges like the first two in the first column. Additionally, certain filters extract diagonal and anti-diagonal edges, exemplified by the bottom one in the first column and the first one in the third column. Their output effectively highlights the corresponding edges. However, in terms of performance, the model featuring Gabor filters in lieu of the initial convolutional layer filters achieved only 74% accuracy compared to the original model's 98% accuracy.

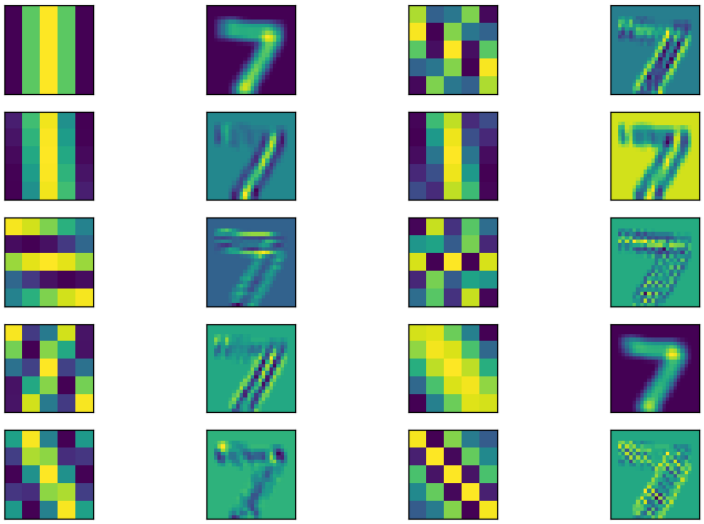

Fig 12. Shown the first layer of Gabor filters

Conclusion

Through this project, my journey into the depths of deep learning has been both challenging and enlightening. Starting with Task 2, I dove into the intricacies of visualizing filter weights, gaining insights into how each filter impacts an image. This practical application not only solidified my understanding but also set the stage for further exploration in Task 3, where I refined my skills in fine-tuning models. The hands-on experience provided by these tasks was invaluable, allowing me to navigate the complexities of training and evaluating deep learning models using PyTorch with greater confidence.

The real test of my newfound knowledge came with Task 4, a rigorous exploration into the effects of hyperparameters on network performance. By experimenting with 27 different models, I was immersed in the practicalities of deep network training, grappling with the challenges of lengthy training sessions that stretched 2-3 hours for a batch, yet yielded a maximum accuracy of just 89%. This outcome hinted at possible limitations within the "flexible" model framework. Despite these hurdles, the experiments were a revelation, offering me a clear perspective on how each network layer operates and the pivotal role hyperparameters play in model training and performance. Remarkably, the features discerned by the convolutional layers resonated with concepts we had covered in previous class sessions, bridging theory with practice. This project marked my initial foray into training a deep network from scratch, furnishing me with practical experience and a nuanced understanding of the foundational structures of deep networks, a venture that was both challenging and profoundly rewarding.

Acknowledgements

https://medium.com/fenwicks/tutorial-1-mnist-the-hello-world-of-deep-learning-abd252c47709#:~:text=,digit%20is%20written%20there

https://nextjournal.com/gkoehler/pytorch-mnist

https://pytorch.org/tutorials/beginner/basics/intro.html

https://github.com/numpy/numpy-tutorials/blob/main/content/tutorial-deep-learning-on-mnist.md