CS 5330 Project-5

Saideep Arikontham

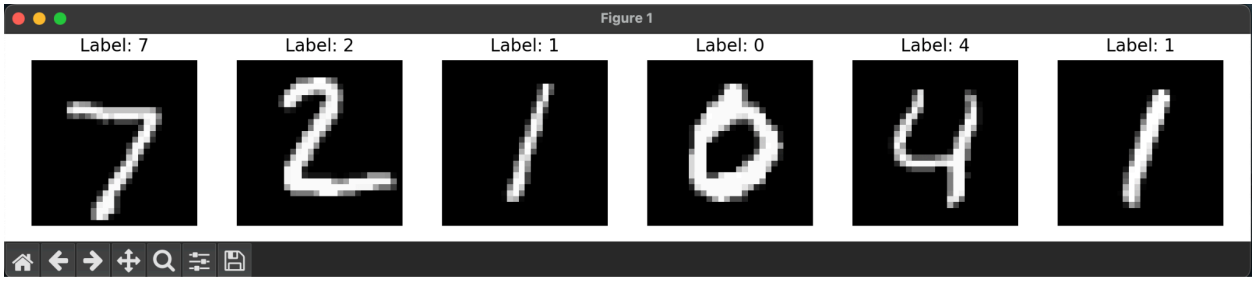
Title: Recognition using Deep Networks

Project Overview

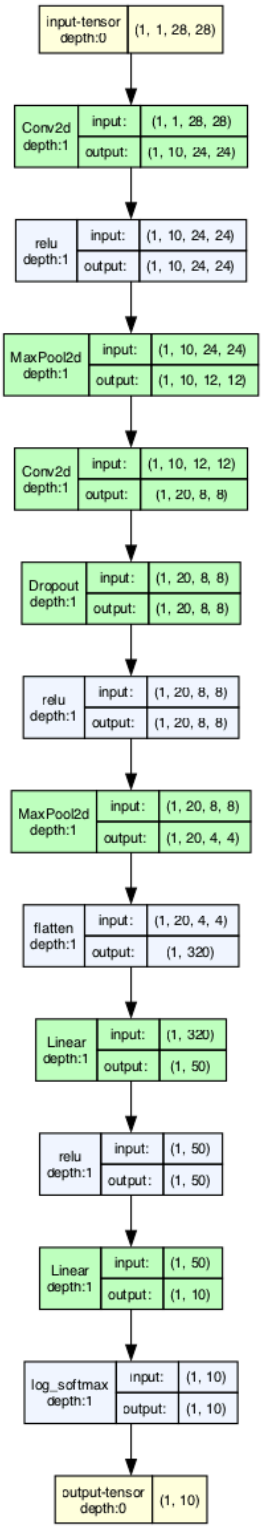
This project primarily focuses on working with Convolutional Neural networks and understanding them with the help of MNIST dataset. The network was designed according to a specified architecture involving convolutional, pooling, dropout, and dense layers. I trained the network over several epochs, saved the trained model, and evaluated it on test images, both from the original dataset and from custom handwritten digits. Furthermore, I visualized intermediate outputs and analyzed the learned filters. This MNIST trained network was then used to apply transfer learning to classify Greek letters (alpha, beta, gamma). To conduct extensive experimentation to understand the impact of architectural changes, I used a more complicated Fashion MNIST dataset. Not only architectural changes, but this experimentation also gave me an idea about how important hyperparameters are in Deep networks. This project gave me deep insights into model design, training behavior, interpretability, and optimization strategies in deep learning.

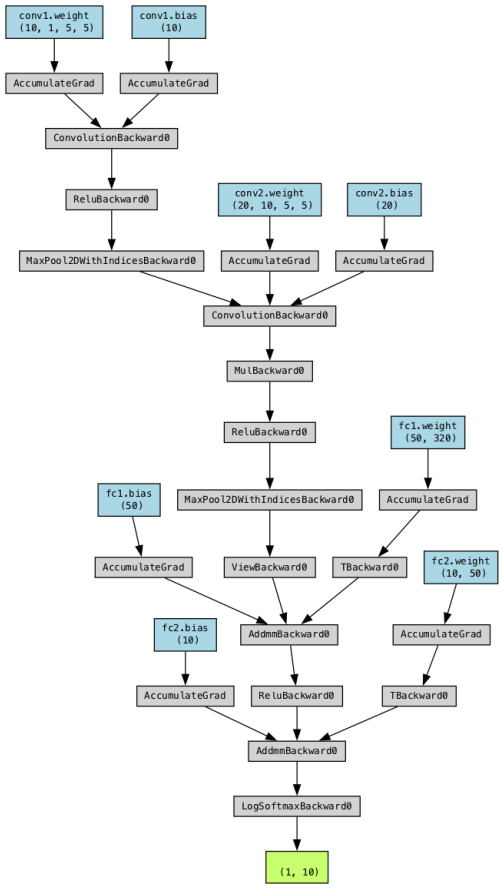
Directed Tasks and required images

1. **Build and train network to recognize digits: (A)**The MNIST digit data consists of a training set of 60k 28x28 labeled digits and a test set of 10k 28x28 labeled digits.After downloading the MNIST dataset from torch datasets, I have visualized the first 6 digits of the Test set using matplotlib. Below are the 6 test samples after plotting (we can see the style in which the digits are written):

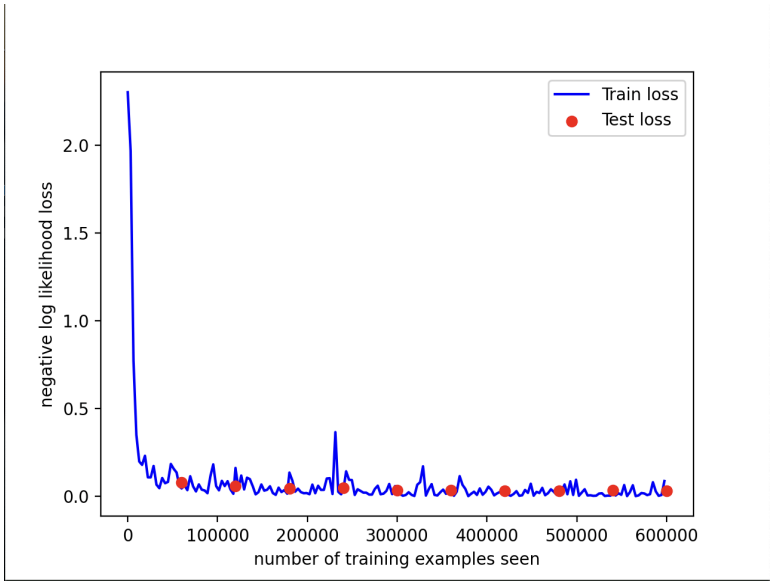


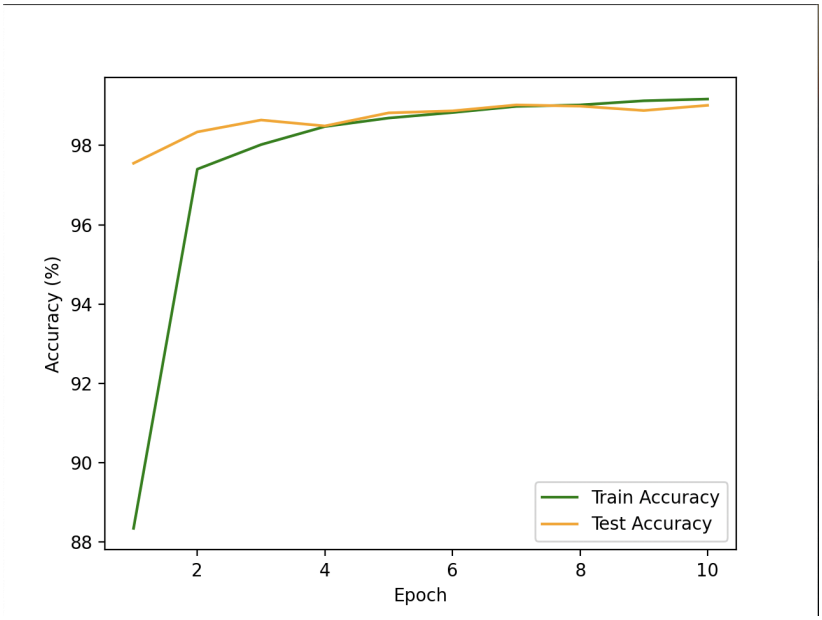
- **(B)** To print the network, I have used different visualization libraries like torchview and torchviz in a separate environment. Below are the network images from torchview and torchviz respectively.





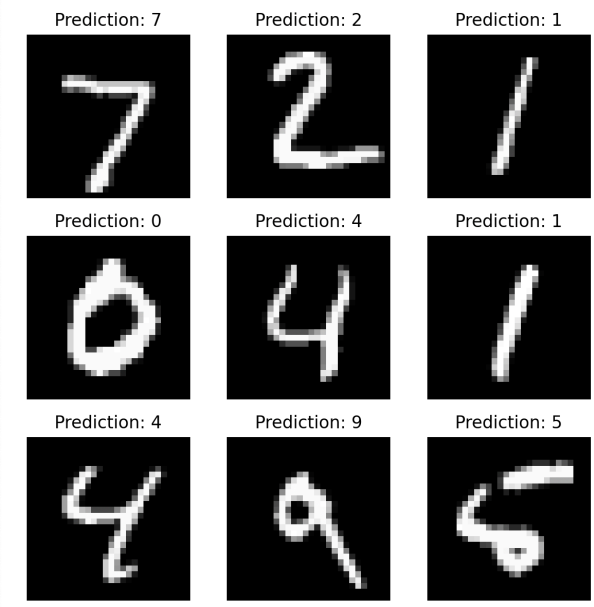
- **(C)** I have trained the above network for 10 epochs with a batch size of 64, learning rate of 0.01 and momentum of 0.9. Below is the plot of train and test losses and the later plot is for train and test accuracies. Stochastic gradient descent is used as an optimizer.





- We can see that the train loss improves a lot just after one pass through the entire training examples (1 epoch) which is reflected in the accuracy plot. The training loss stabilizes after one epoch and the decrease is very less.
- We can see that the train accuracy is improving slightly every epoch.
- In the plot for train and test losses, we can see that the train loss is very close to the test loss for each epoch indicating that the model is generalizing well on unseen data.

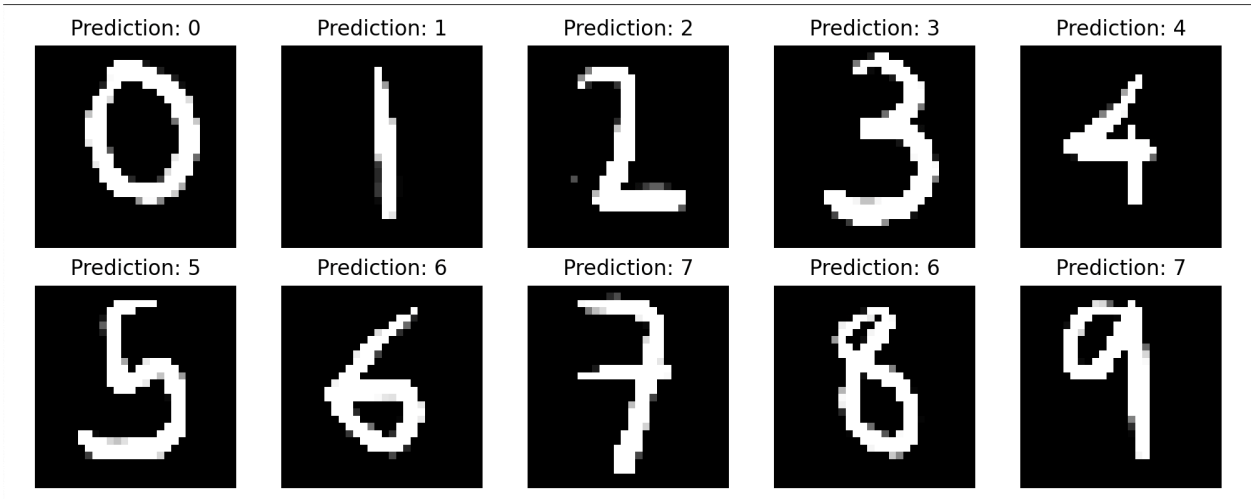
- **(D)** The above network's weights are saved to be used for any further analysis.
- **(E)** After loading the above network and inferring the test data, below are the prediction results on first 9 images.



- We can see that the model successfully predicts all of the test samples. We can see in the below prediction probability outputs that each classification also has a high confidence.

```
Image 1:
  Output values: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00']
  Predicted: 7, True Label: 7

Image 2:
  Output values: ['0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted: 2, True Label: 2

Image 3:
  Output values: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted: 1, True Label: 1

Image 4:
  Output values: ['1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted: 0, True Label: 0

Image 5:
  Output values: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted: 4, True Label: 4

Image 6:
  Output values: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted: 1, True Label: 1

Image 7:
  Output values: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']
  Predicted: 4, True Label: 4

Image 8:
  Output values: ['0.00', '0.00', '0.00', '0.00', '0.01', '0.00', '0.00', '0.00', '0.00', '0.99']
  Predicted: 9, True Label: 9

Image 9:
  Output values: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.99', '0.00', '0.00', '0.00', '0.00']
  Predicted: 5, True Label: 5
```

- **(F)** To see how well the model training has worked, we can test its generalization on my hand written images. Below is the output for that:



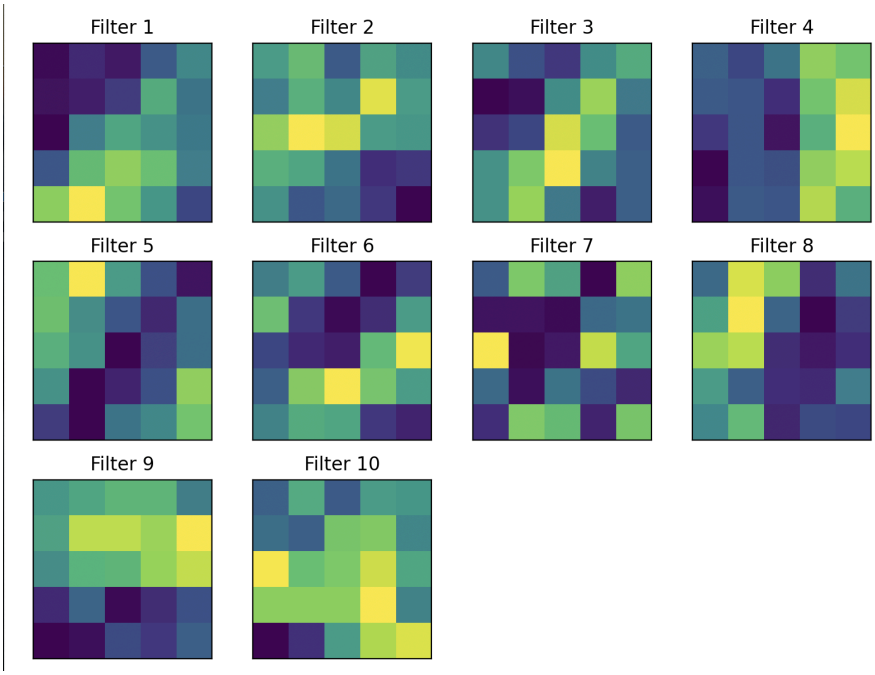
- My writing is different from the style of the MNIST images. However, the model was able to classify difficult cases like 4 and 7 perfectly. However, the model fails to classify 8 (likely due to smaller upper circle, hence classified as 6) and 9 (just an incorrect prediction case, however, prediction is not random and can be considered its closest match i.e., 7).

2. **Examine the network: (A)** In a separate notebook, I have examined the kernel filters from the first layer (conv1) whose tensor size is [10, 1, 5, 5]. Below are the weights of each filter printed as output:

```
conv1.weight shape: torch.Size([10, 1, 5, 5])
Filter 1 with shape (5, 5):
[[-0.42201328 -0.31806725 -0.37726805 -0.13588181  0.05368593]
 [-0.40683863 -0.35716206 -0.2622627   0.22208007 -0.03917788]
 [-0.4494954   0.01010386  0.19515796  0.1094411  -0.01802939]
 [-0.14987127  0.2878786   0.40662348  0.30420548  0.01157686]
 [ 0.38934806  0.6045715   0.32535967  0.12561232 -0.21290116]]

Filter 2 with shape (5, 5):
[[ 0.09713861  0.22621226 -0.16677812  0.12027553  0.02354352]
 [-0.02775162  0.1828075   0.02167204  0.46237928  0.09706802]
 [ 0.3296584   0.50595623  0.4448343   0.08738009  0.07761721]
 [ 0.18210219  0.13429037 -0.07236187 -0.31416693 -0.2920258 ]
 [ 0.05026319 -0.17667848 -0.09943163 -0.28406173 -0.44423023]]

Filter 3 with shape (5, 5):
[[ 0.03169272 -0.18235996 -0.26281586  0.03816959  0.16019253]
 [-0.41007325 -0.38376942  0.03740939  0.3365561  -0.0293291 ]
 [-0.2774735  -0.21709552  0.42811516  0.23121275 -0.14356005]
 [ 0.05735103  0.2756652   0.4875389  -0.00202324 -0.13553475]
 [ 0.05892952  0.3295091  -0.02874276 -0.33295    -0.13221557]]

Filter 4 with shape (5, 5):
[[-0.09777969 -0.20518906 -0.02945557  0.39764947  0.32567656]
 [-0.12066058 -0.1448919  -0.28755653  0.3265858   0.5269876 ]
 [-0.25708035 -0.15794739 -0.37123364  0.23141026  0.5936305 ]
 [-0.4311084  -0.14143942 -0.1794566   0.39544377  0.47395718]
 [-0.39808193 -0.12355663 -0.1533386   0.46794218  0.2329692 ]]

Filter 5 with shape (5, 5):
[[ 1.60469294e-01  4.06390369e-01  3.52100097e-02 -2.22423017e-01
  -3.94563526e-01]
 [ 1.75312057e-01 -9.39623732e-03 -2.04279572e-01 -3.40686202e-01
  -1.23846285e-01]
 [ 1.09374262e-01  4.21008299e-04 -4.29276735e-01 -2.72571623e-01
  -1.26816139e-01]
 [ 1.11224484e-02 -4.32231218e-01 -3.48030210e-01 -2.13086709e-01
   2.47629583e-01]
 [-2.77811348e-01 -4.33572263e-01 -1.00489788e-01 -3.30957472e-02
   1.86026037e-01]]
```

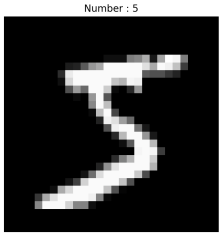
```
Filter 6 with shape (5, 5):
[[ 0.02321287  0.12565356 -0.10560409 -0.36200514 -0.20301215]
 [ 0.25644657 -0.215413   -0.3443197  -0.24877304  0.12455041]
 [-0.1769412  -0.26289922 -0.28910425  0.23124148  0.48596504]
 [-0.09796771  0.31096002  0.4981412   0.28171983  0.13315214]
 [ 0.02975887  0.18506113  0.1561344  -0.21529357 -0.27802867]]

Filter 7 with shape (5, 5):
[[-0.08226613  0.09188057  0.03003467 -0.19709422  0.10745347]
 [-0.18001953 -0.17716429 -0.18785898 -0.06599024 -0.0454412 ]
 [ 0.18168986 -0.19294898 -0.17433873  0.14645785  0.03545216]
 [-0.06491228 -0.18652707 -0.04818251 -0.1098891  -0.15483738]
 [-0.14475943  0.09452663  0.06836487 -0.15983525  0.08710346]]

Filter 8 with shape (5, 5):
[[-0.0457854   0.41795036  0.30729553 -0.22024615 -0.00333338]
 [ 0.14157617  0.4662024  -0.05834192 -0.32438117 -0.18379894]
 [ 0.32813475  0.37367433 -0.2218431  -0.27800077 -0.21375218]
 [ 0.12345772 -0.08003287 -0.21537398 -0.22789407  0.02535698]
 [ 0.06129854  0.22548255 -0.24045865 -0.13982137 -0.15172324]]

Filter 9 with shape (5, 5):
[[ 0.08442619  0.14814961  0.22980031  0.22962956 -0.02555929]
 [ 0.13411896  0.46505928  0.4657635   0.39277363  0.57826734]
 [ 0.04111239  0.20952226  0.22953913  0.3854157   0.47093013]
 [-0.3777155  -0.14502959 -0.49646717 -0.37510207 -0.23841462]
 [-0.51369435 -0.47730136 -0.26262003 -0.32338282 -0.1658231 ]]

Filter 10 with shape (5, 5):
[[-0.20420213  0.07129654 -0.23707172  0.01469035 -0.01240586]
 [-0.15947746 -0.21244732  0.18617345  0.20733851 -0.06738564]
 [ 0.40379733  0.15167347  0.19654462  0.3435268   0.06512299]
 [ 0.22926933  0.22730015  0.22477952  0.41413605 -0.07653196]
 [-0.50247663 -0.37224823  0.00902474  0.29360035  0.3619977 ]]
```

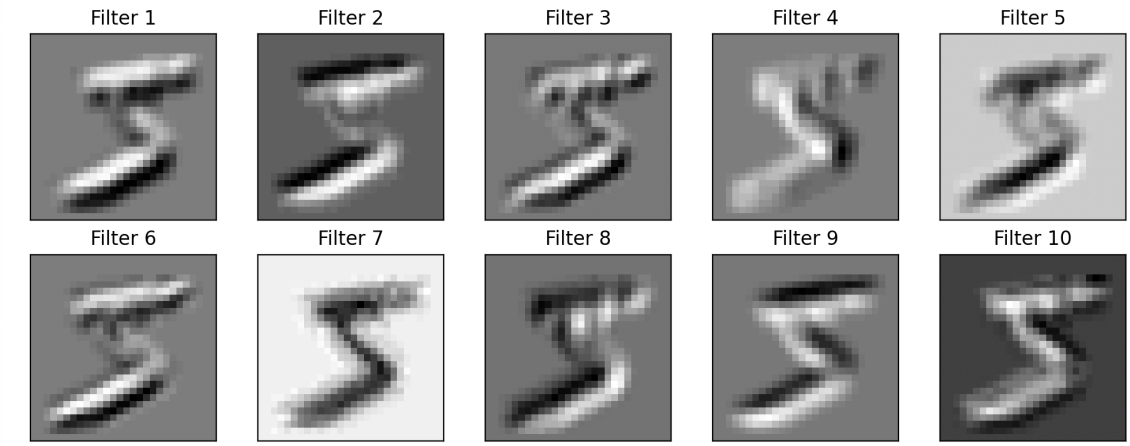
- These filters are visualized using matplotlib's pyplot subplots (3x4) and below are the results:



- Higher values in the filter are denoted by lighter color and lower values with dark colors. These weights might help in determining the edges, strokes, patterns, separate background from the actual information and more. Some filters with high positive values right next to high negative values to detect edges. It is highly difficult to pinpoint what these filters are doing exactly. These filters do not look like any standard filters as these weights are learned and updated through each batch and it does not make sense that they look like standard filters. Even if they do, it has to be absolutely by chance because they will be influenced by the training task, training hyperparameters and more.

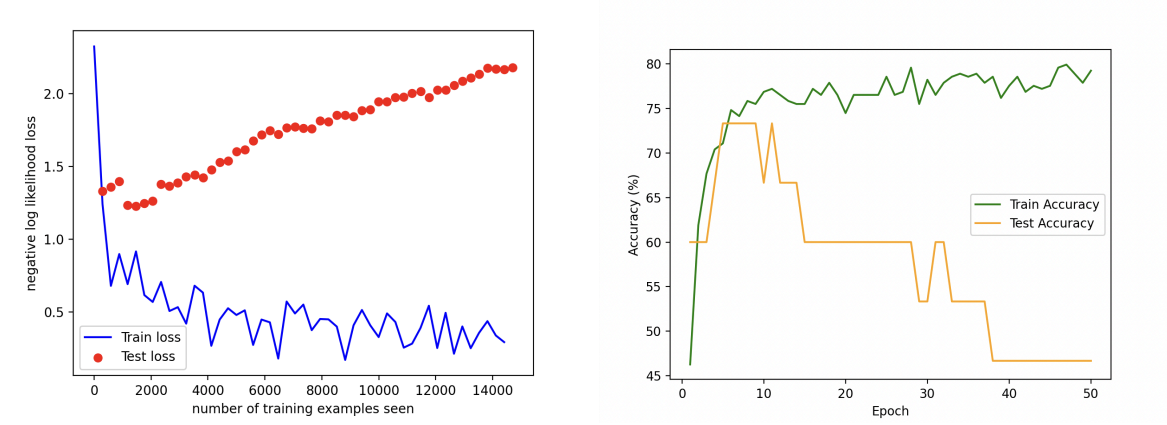
- When these filters are applied to an image, below would be the outputs respectively.





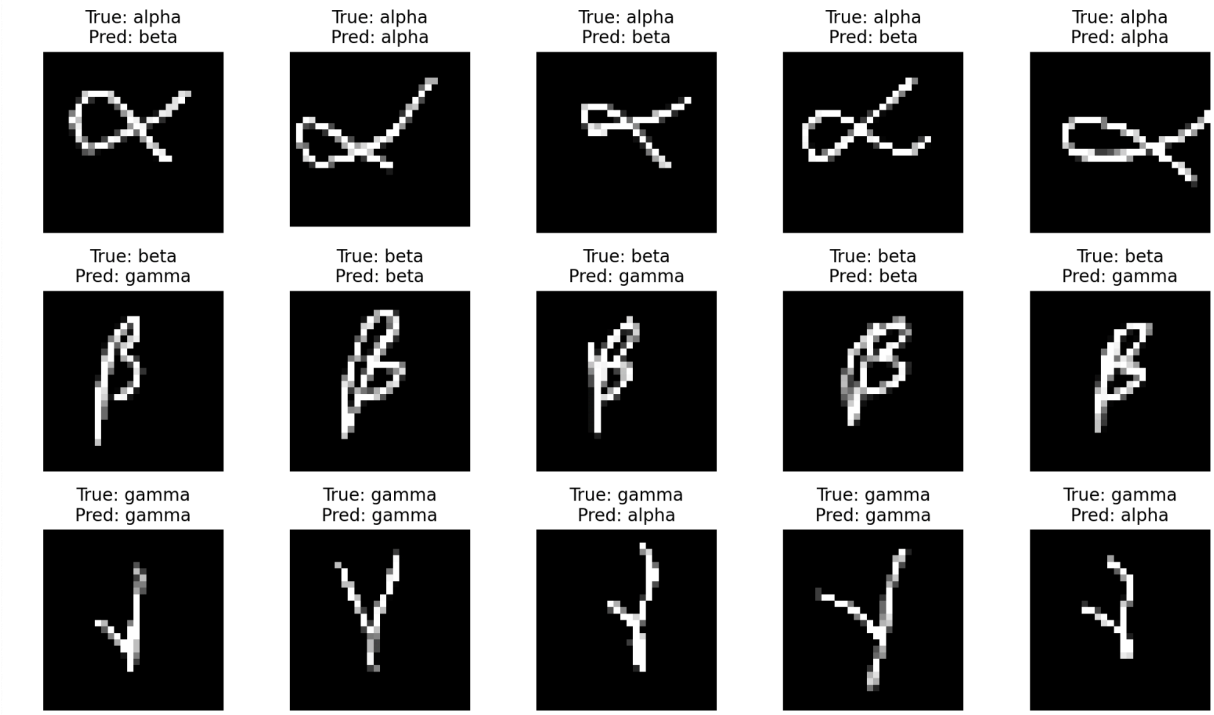
- We can see that when the filters are applied, each filter produces a different result and thus passes on different information to the next layers. We can see that even though filter 2 and filter 8 look different, their output looks quite similar (internal pattern seems different). But as we can see, the filters perfectly captured the edges, the pattern of the digit though each filter differently.

3. **Transfer learning on Greek letters:** Using the trained MNIST network, we can perform transfer learning to re-train selected Network layer's weights to make it a Greek letter predicting network. We freeze the entire network and replace the final layer with only 3 output hidden units to predict for each of the greek letters. SGD is used as an optimizer. I have used all the additional letters as the training set and my own hand drawn images as the test set.

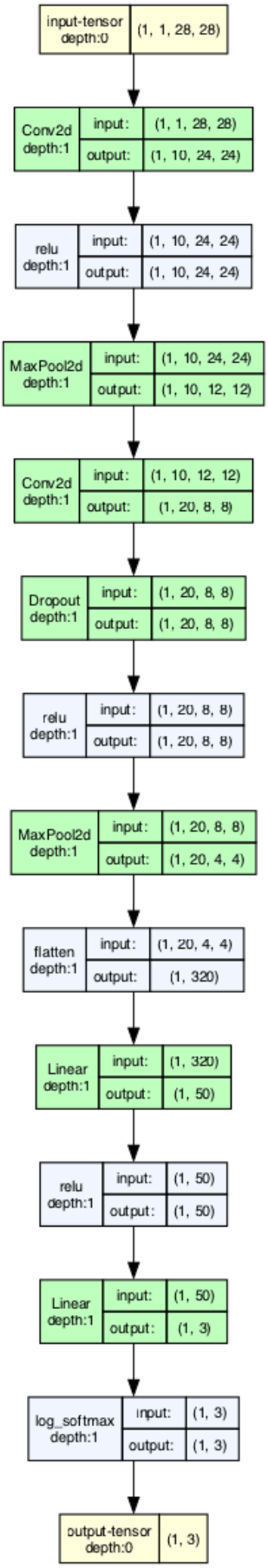


- We can see that after 5 epochs, the test loss consistently increases and the model is failing to generalize on the test data. The train accuracy is improving while the test accuracy is declining indicating a possible underfit and the model failing to learn patterns.

- I have trained the model for 50 epochs with a learning rate of 0.001 and a momentum of 0.7 and a batch size of 16.
- However, I am using the test loss to save the best model weights after training the model so that all the bad generalization can be negated. Below is how this trained network works on greek letters (hand written test images):



- We can see that each class is being correctly predicted only twice (thrice in case of gamma) which is not really great. This means that accuracy for alpha is 40%, accuracy for beta is 40% and accuracy for gamma is 60%.
- Below is the image of modified network


input-tensor
depth:0
(1, 1, 28, 28)
Conv2d
depth:1
input:
(1, 1, 28, 28)
output:
(1, 10, 24, 24)
relu
depth:1
input:
(1, 10, 24, 24)
output:
(1, 10, 24, 24)
MaxPool2d
depth:1
input:
(1, 10, 24, 24)
output:
(1, 10, 12, 12)
Conv2d
depth:1
input:
(1, 10, 12, 12)
output:
(1, 20, 8, 8)
Dropout
depth:1
input:
(1, 20, 8, 8)
output:
(1, 20, 8, 8)
relu
depth:1
input:
(1, 20, 8, 8)
output:
(1, 20, 8, 8)
MaxPool2d
depth:1
input:
(1, 20, 8, 8)
output:
(1, 20, 4, 4)
flatten
depth:1
input:
(1, 20, 4, 4)
output:
(1, 320)
Linear
depth:1
input:
(1, 320)
output:
(1, 50)
relu
depth:1
input:
(1, 50)
output:
(1, 50)
Linear
depth:1
input:
(1, 50)
output:
(1, 3)
log_softmax
depth:1
input:
(1, 3)
output:
(1, 3)
output-tensor
depth:0
(1, 3)

4. **Design of my own experimentation on Fashion MNIST: (A)** To understand more about the impact of different configurations and hyperparameters, I have designed an experiment to tune the training process on Fashion MNIST. The dimensions that I have chosen are
    a. Number of Convolution layers
    b. Number of Linear layers
    c. Learning rate (fixed list of values that will be used)
    d. Batch size
    e. Dropout
    f. Number of hidden units

- The design is set up as a simple Grid Search process for hyperparameter tuning. With the maximum values of each of the above dimensions passed as the command line arguments, a parameter grid is created and the network is trained for at least 50 different configurations.
- After the training is done, I print out the results as a dataframe with test accuracy sorted in descending order. Therefore, we have a dataframe of best trained networks at top.
- Adam optimizer is used to train all of the Fashion MNIST networks.

- **(B)** Number of convolution layers, Number of linear layers and number of hidden units (neurons) define the complexity of the model. The more of them they are, the more complex the model is. I do not have experience working on the Fashion MNIST dataset and therefore I do not know if it would require a more complex network to learn patterns required for prediction.
- The more complex the problem is, the better it is to have a more complex model. A prediction would be that since the fashion MNIST is more complex than traditional MNIST, it should require a more complex model compared to the one we build for task 1.
- Coming to Learning rate, batch size, drop out and epochs, these values help in the hyperparameter tuning. Larger Batch size allows in gradient updates after seeing a larger batch of training data. Drop out helps with prevention of overfitting and learning rate would be crucial in identifying the global minima of loss. These are impossible to predict unless we have seen at least one previous model training.
- However, Too high or too low values for these hyperparameters result in training of a poor model.

- **(C)** The experimentation is run with the below parameters:

```
Parameter grid:
Conv layers: [1, 2, 3]
Linear layers: [1, 2, 3]
Dropouts: [0.1, 0.2, 0.30000000000000004]
Hidden units: [64, 128]
Batch sizes: [32, 64]
Learning rates: [0.01, 0.001, 0.0001]
Epochs: [5, 10, 15, 20]
Total combinations: 1296
```

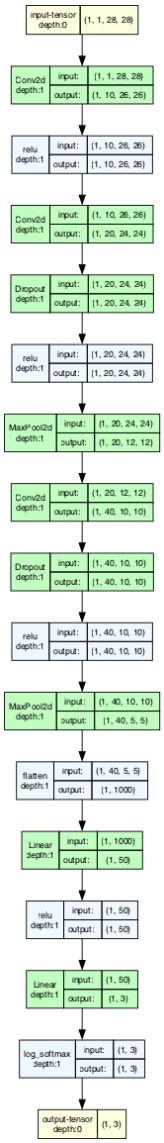
- Only a random 75 combinations of the total combinations will be used to train a model and store their respective results.
- We can see the top 5 results out of 75 combinations that were tried.

	conv_layers	linear_layers	dropout_rate	hidden_units	batch_size	learning_rate	epochs	test_acc	time_sec
73	2	1	0.3	128	32	0.001	15	92.49	152.93
35	2	3	0.3	64	64	0.001	15	92.20	120.65
47	2	3	0.1	128	32	0.001	15	92.19	189.40
23	3	2	0.2	128	64	0.001	15	92.09	120.33
55	3	3	0.3	64	64	0.001	20	92.08	177.71
..	...	...	...	...	...	...	...	...	...
10	3	1	0.3	128	32	0.010	5	83.42	59.58
42	3	2	0.3	64	32	0.010	20	83.21	257.02
34	3	3	0.3	64	32	0.010	10	80.09	135.46
69	3	3	0.1	128	32	0.010	5	10.00	73.37
48	2	1	0.3	64	32	0.010	5	10.00	54.92

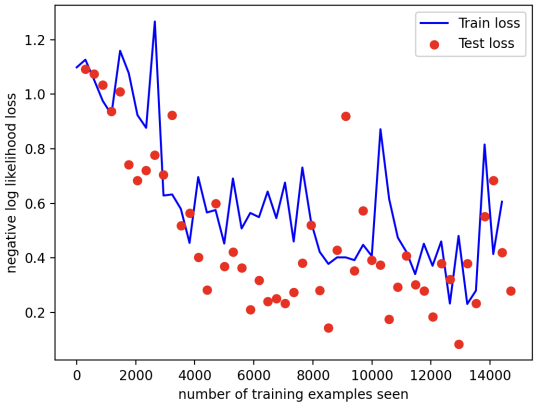
- We can see that complex networks are performing consistently on test data with high number of epochs.
- However, the best network is a simple network with 2 convolution layers and 1 linear layer which is surprising and different from what I have expected.
- But I believe using more epochs for complex networks with 2 or more convolution and linear layers might give better results.
- The performance is very bad for higher learning rates therefore 0.001 might be a very good learning rate and can be fixed to explore other dimensions.
- I still believe further exploration of complex models with more epochs will yield better results than above.
- Using hyper parameter optimization libraries like Optuna might help in achieving at the best architecture for obtaining a very high performance.

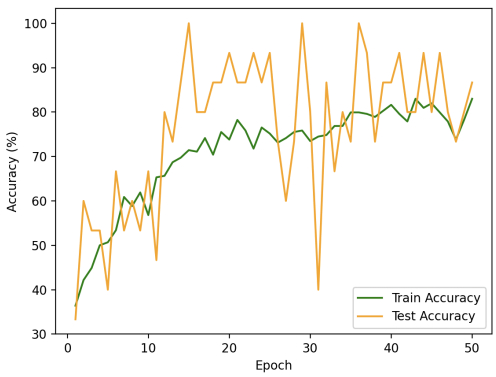
Extensions

1. **Greek letter recognition using deep networks:** Instead of freezing all the layers and updating only the weights of the final linear layer, I trained a network from scratch to see if better results can be obtained compared to the transfer learning. I have used 3 convolution layers and two linear layers for the network configuration. I also have used data augmentation to rotate and translate since there is very little data for training. I have trained for 50 epochs with a learning rate of 0.01 and momentum of 0.9.
- Below is the network used to train the model:

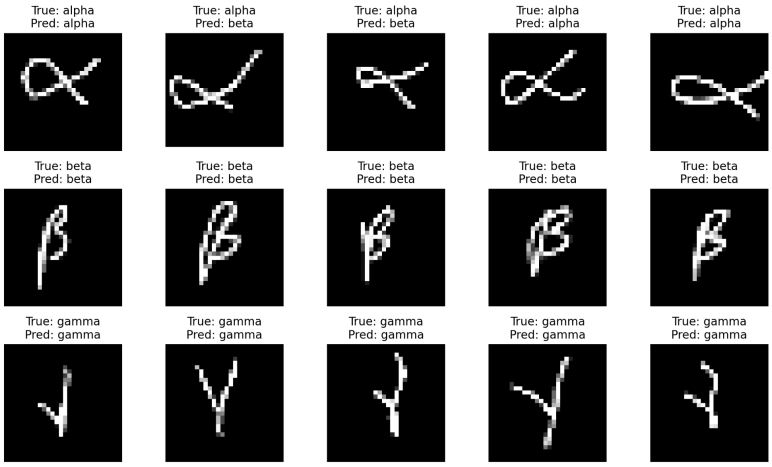


- Below are the results for the full training on the Greek letters:





- We can see that the model is learning better patterns as the train loss and test loss decrease with consistent training while the train accuracy and test accuracy improve.
- Below are the prediction results on the my hand written greek letters:



- We can see that apart from alpha other classes are predicted perfectly without any error which shows significant improvement from the transfer learning results.

2. **Live video detection:** I also built a live detection system to identify either digits or greek letters (can be switched with key press). A center bounding box is fixed and any digit/letter inside that region will be classified and labelled in the video stream. Only those predictions of high confidence (above 90%) will be labelled and others will direct a message saying the digit/letter is unclear. Below is a link of small live demonstration of this:

https://drive.google.com/file/d/1chQ-1dJ1XscJLVVgansy_SQ3sqYKW-eV/view?usp=drive_link

---

Reflection

Through this project, I gained hands-on experience working with CNNs. I already had a pretty good idea about how Neural networks function, what different activation functions do, what different optimizers. However, I had very less experience working with CNNs and this project is a good starting exercise to familiarize myself more with them. I also had a good experience working with Pytorch which I had not done a lot previously. It would be really great to work on a more complex classification task using these Deep networks which I plan to do with the final project. This project has definitely refreshed and strengthened my understanding of the deep

networks and I look forward to carrying these to work on a more complicated singular classification task to address certain classification task.

Acknowledgements

The best resources I found for this assignment were ChatGPT and cloud recordings of classes. I used ChatGPT to understand errors that I encountered while working on this project. ChatGPT also helped me understand python implementation OpenCV functions. ChatGPT also helped in understanding different Pytorch configurations. Since Python is my go to programming language, I did not have a lot of trouble in writing any classes or functions. However, ChatGPT helped in writing clear doc strings for each of the functions and classes. Throughout all the projects, ChatGPT has been a great tool to understand multiple stages and gain deeper understanding of different concepts.