CS 5330 PATTERN RECOGNITION AND COMPUTER VISION
Project 5: Recognition using Deep Networks

Ruchik Jani 002825482
Anuj Patel 002874710

1) Project description:

The goal of the project is to develop and analyze a deep neural network for digit recognition using the MNIST dataset. The tasks include building and training a convolutional neural network (CNN) using PyTorch, visualizing the dataset, designing and training the network architecture with specified layers, training the model and evaluating its performance on both training and test sets, saving the trained network, testing the network on new inputs, examining the network's structure and filters, transferring learning to recognize Greek letters, and designing an experiment to optimize the network's architecture. Each task involves structured Python coding, visualization, analysis, and experimentation to understand and improve the network's performance.

2) Description of Project Tasks:

Task 1 - Build and train a network to recognize digits:

A. Get the MNIST digit data set

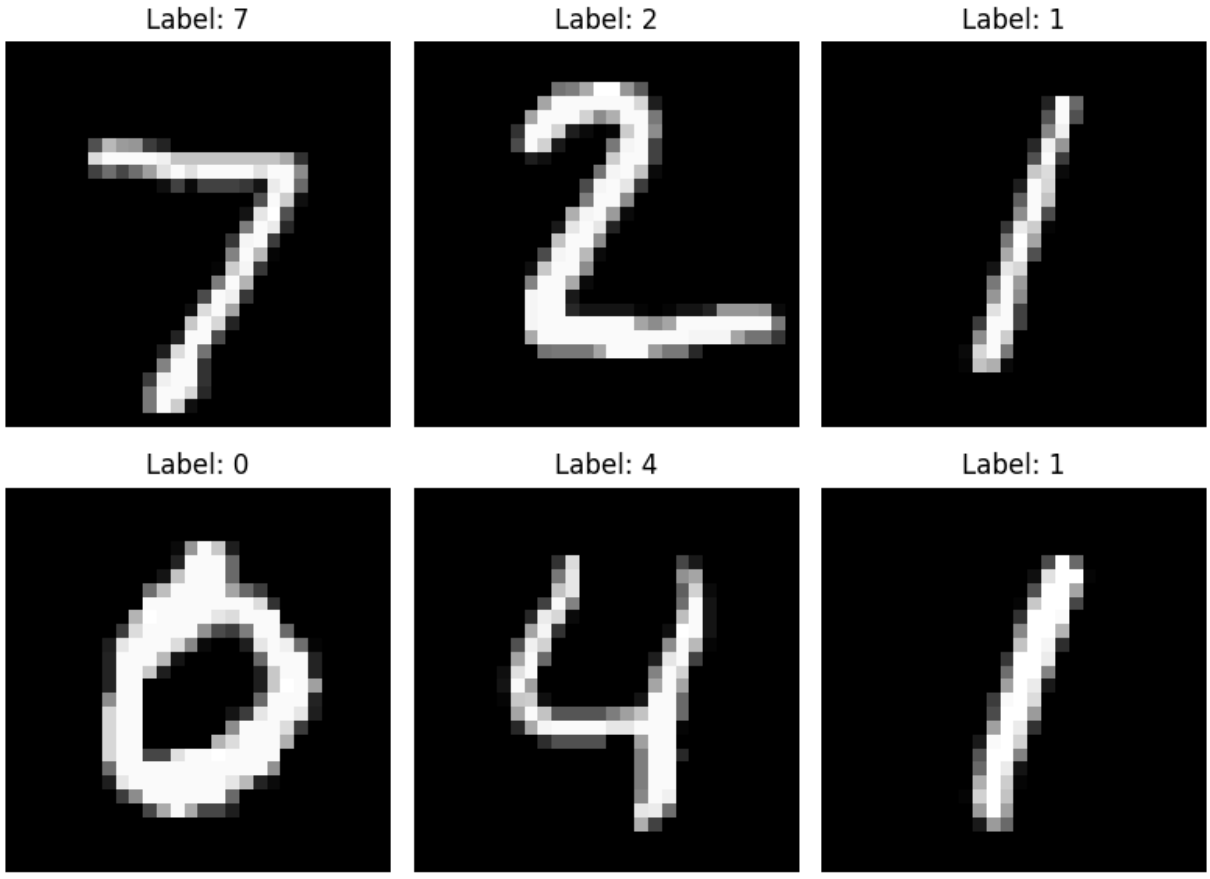


Image showing a grid of plots of the first six example digits from the test set

B. Build a network model

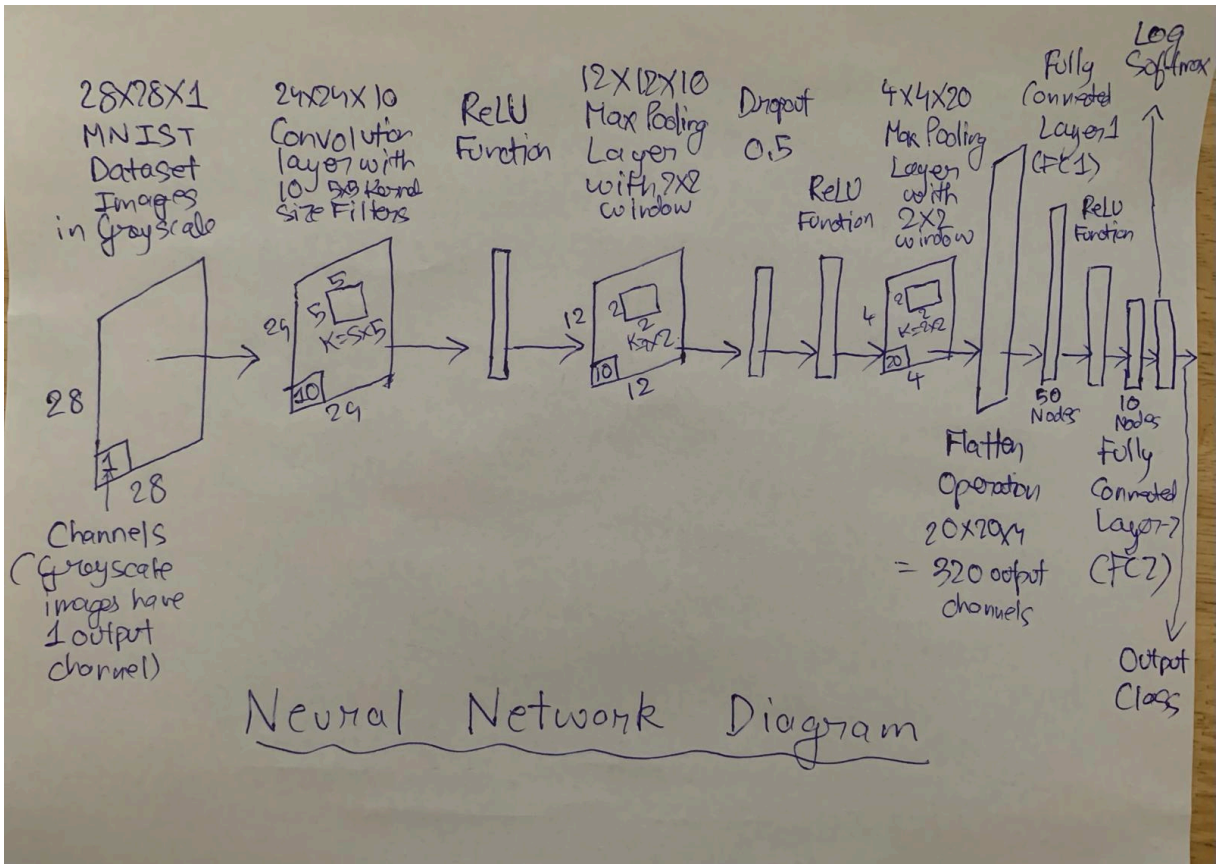


Image showing a diagram of the network

C. Train the model

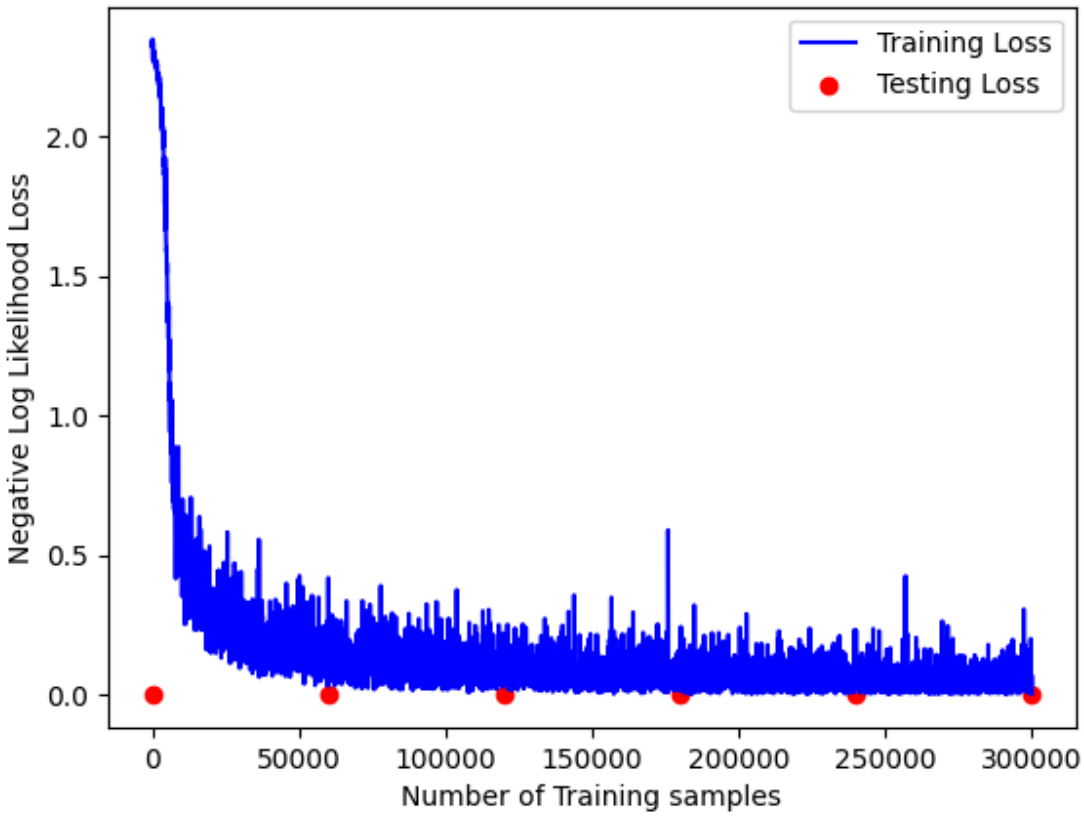


Image showing a plot of the testing and training losses against the number of training samples

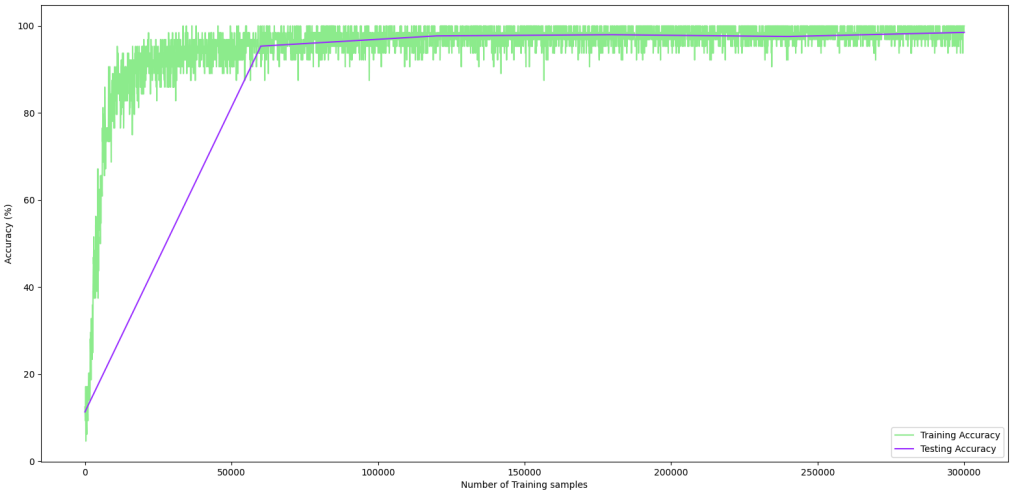


Image showing a plot of the testing and training accuracies against the number of training samples

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

```
rj@Radio-Jockey:~/Project-5$ python3 Task1E.py
Image 1
Output values: ['-14.19', '-15.42', '-8.60', '-9.86', '-20.15', '-20.93', '-28.03', '-0.00', '-14.10', '-9.24']
Index of max output value: 7. This value is located at postion 8 in the list of all output values for this image.
Predicted Label: 7, True Label: 7

Image 2
Output values: ['-9.93', '-9.11', '-0.00', '-16.32', '-23.60', '-19.86', '-12.97', '-21.33', '-11.17', '-25.44']
Index of max output value: 2. This value is located at postion 3 in the list of all output values for this image.
Predicted Label: 2, True Label: 2

Image 3
Output values: ['-9.69', '-0.00', '-9.87', '-13.23', '-7.94', '-13.00', '-9.81', '-7.84', '-8.56', '-11.94']
Index of max output value: 1. This value is located at postion 2 in the list of all output values for this image.
Predicted Label: 1, True Label: 1

Image 4
Output values: ['-0.00', '-20.42', '-12.75', '-15.57', '-21.76', '-17.30', '-9.93', '-13.03', '-13.09', '-14.68']
Index of max output value: 0. This value is located at postion 1 in the list of all output values for this image.
Predicted Label: 0, True Label: 0

Image 5
Output values: ['-19.89', '-15.13', '-18.26', '-21.26', '-0.00', '-22.46', '-15.15', '-13.31', '-17.77', '-11.28']
Index of max output value: 4. This value is located at postion 5 in the list of all output values for this image.
Predicted Label: 4, True Label: 4

Image 6
Output values: ['-12.31', '-0.00', '-13.48', '-15.76', '-8.55', '-17.96', '-14.29', '-8.00', '-11.17', '-13.22']
Index of max output value: 1. This value is located at postion 2 in the list of all output values for this image.
Predicted Label: 1, True Label: 1

Image 7
Output values: ['-20.48', '-7.89', '-14.80', '-18.05', '-0.00', '-18.17', '-17.48', '-8.75', '-6.10', '-11.19']
Index of max output value: 4. This value is located at postion 5 in the list of all output values for this image.
Predicted Label: 4, True Label: 4

Image 8
Output values: ['-14.68', '-11.23', '-10.50', '-4.12', '-5.58', '-10.65', '-17.08', '-11.37', '-5.67', '-0.02']
Index of max output value: 9. This value is located at postion 10 in the list of all output values for this image.
Predicted Label: 9, True Label: 9

Image 9
Output values: ['-18.91', '-22.44', '-13.90', '-17.86', '-18.97', '-0.00', '-8.14', '-20.21', '-12.40', '-17.09']
Index of max output value: 5. This value is located at postion 6 in the list of all output values for this image.
Predicted Label: 5, True Label: 5
```

Image showing the output values of the network, index of max output value, predicted label, and true label for the first 9 examples in the test set

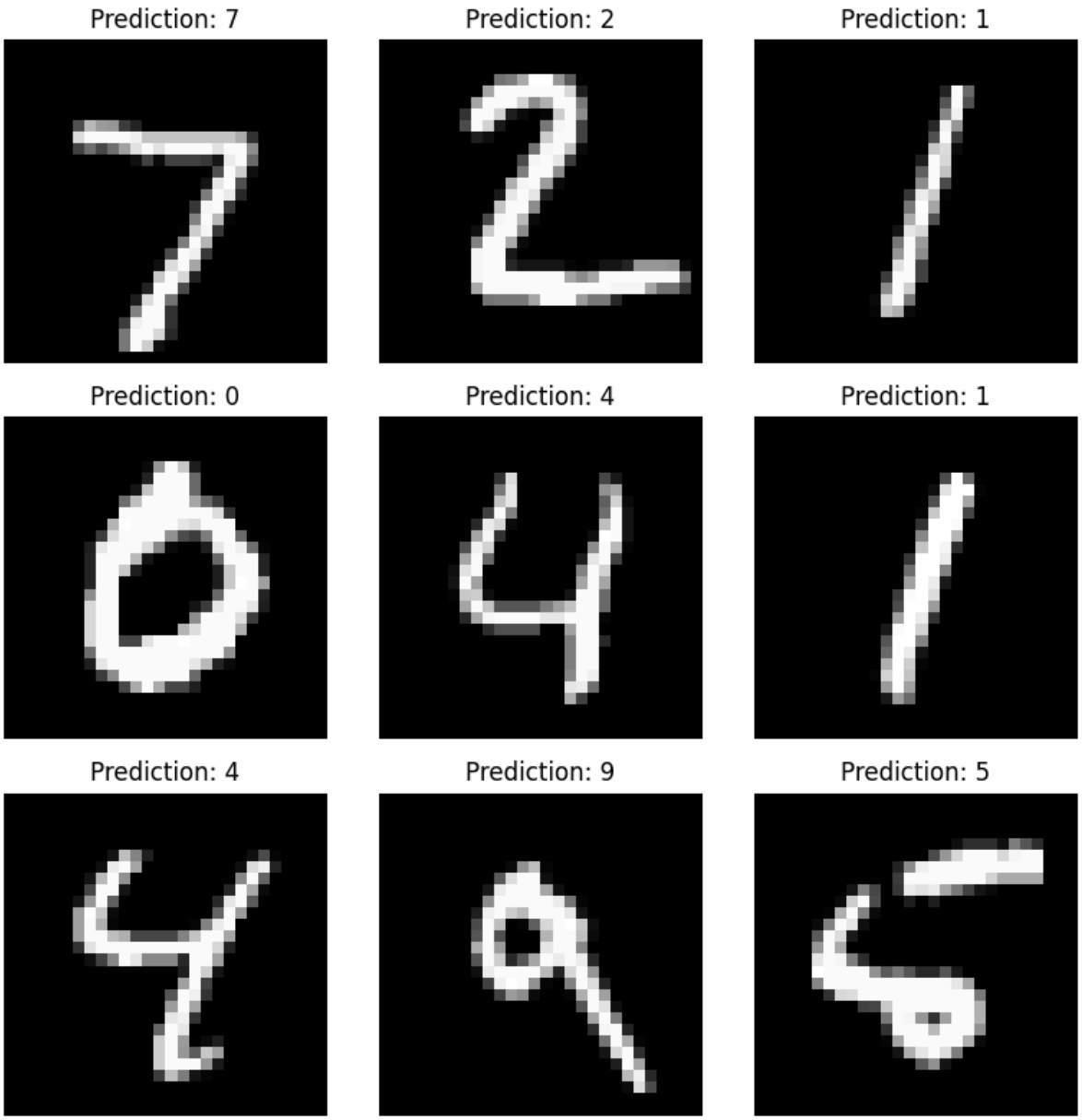


Image showing a plot of the 9 digits of the test set recognized by the trained network

The model accuracy on the test dataset is 100% as seen from the above image. This accuracy is only for the first 9 digits of the test set which were passed and correctly recognized by the network.

F. Test the network on new inputs

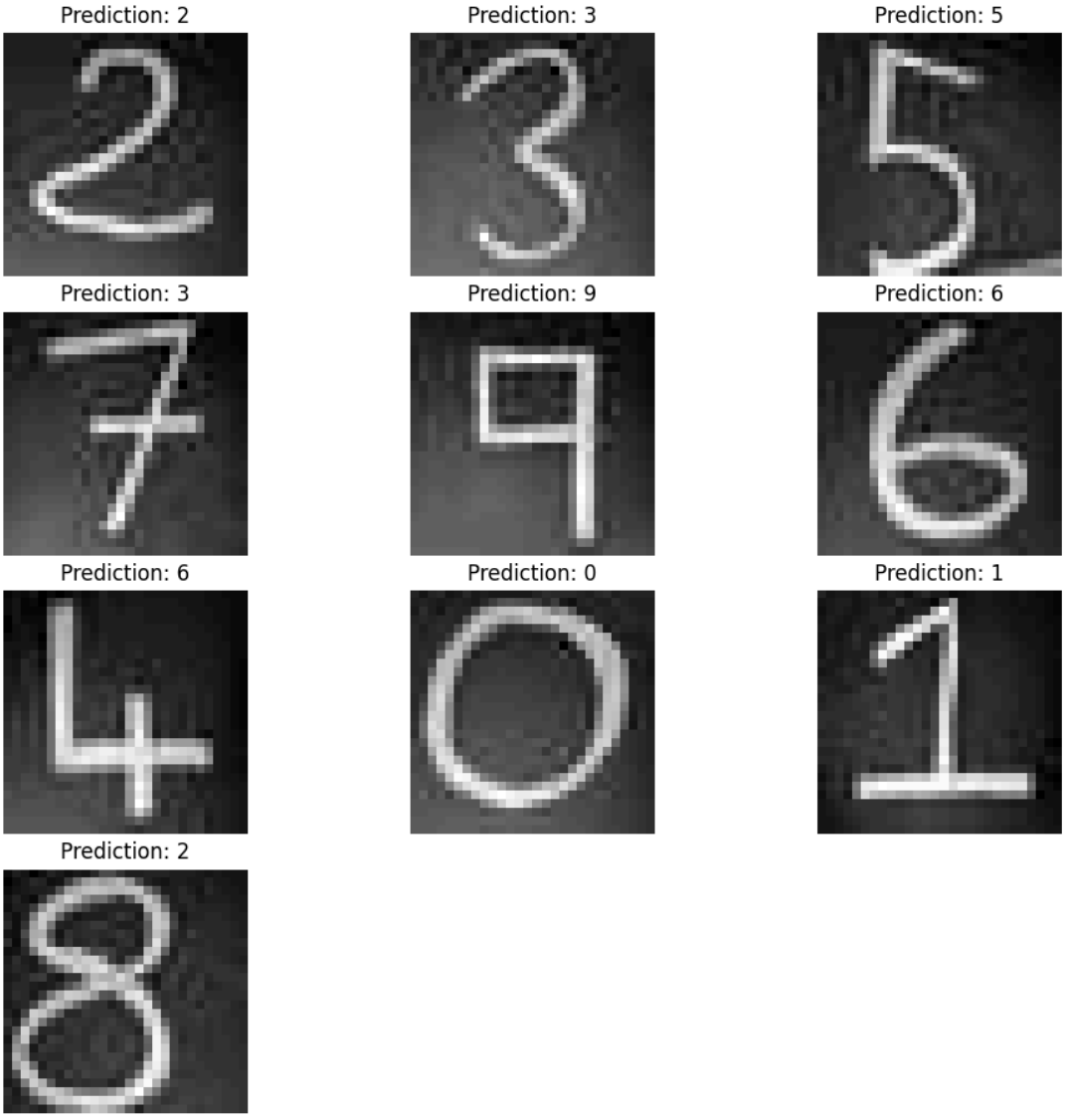


Image showing a plot of all the digits from 0 to 9 from a custom dataset recognized by the trained network

To prepare the dataset, we drew all the digits with a pencil on white paper. Then we captured the images with our phone camera. We cropped the images to square size using the camera settings. The code for this task then resizes the images to 28x28 and converts them to greyscale. The MNIST dataset images are white on a black background. So, in the code, we inverted the intensities of the converted greyscale images to match this.

The accuracy of the model on this new test dataset is 70% since three digits out of ten were incorrectly predicted: 7, 4 & 8 were predicted as 3, 6 & 2 respectively.

Task 2 - Examine your network:

A. Analyze the first layer

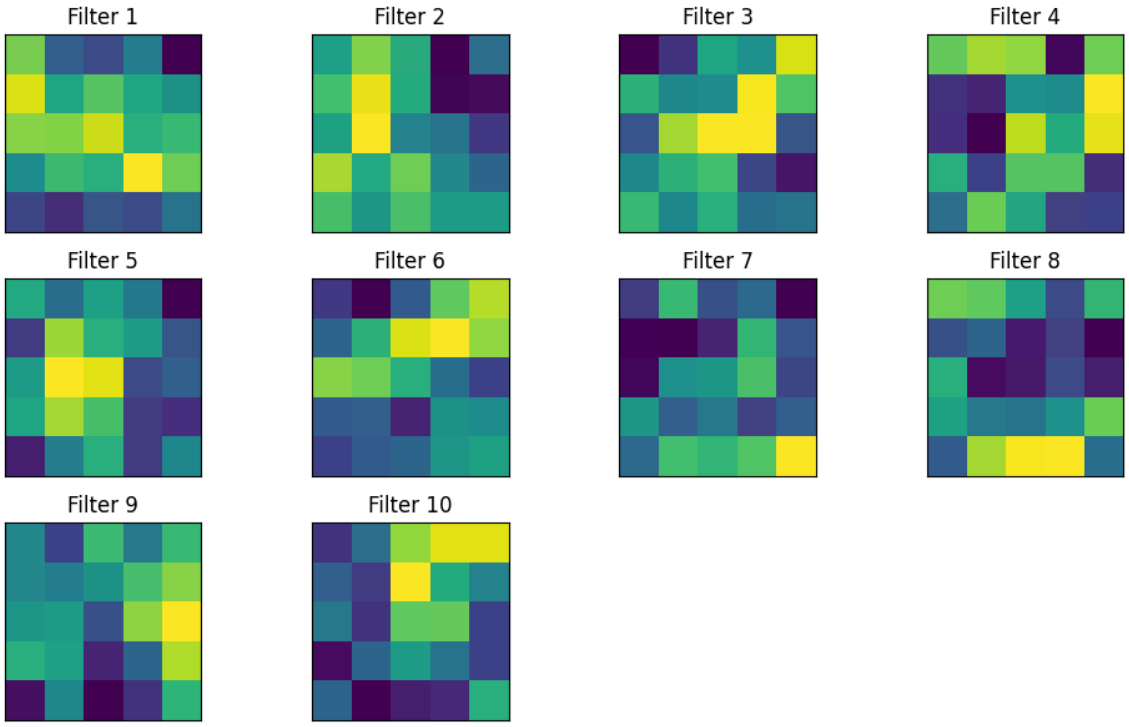


Image showing a 3 x 4 grid of plots for the filters to visualize the weights of the first layer of the neural network

When we printed the model in the code, we got the following output showing the structure of the neural network and the name of each layer. This matches with the definition of the network from Task 1B:

MyNetwork(
(conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
(pool1): 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)
(pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(fc1): Linear(in_features=320, out_features=50, bias=True)
(fc2): Linear(in_features=50, out_features=10, bias=True)
)

When we printed the shape of the weights, we got this:

Shape of weights: torch.Size([10, 1, 5, 5])

This is the description of the above values:
10 - Number of output channels (no. of filters)
1 - Number of input channels (For grayscale images, it's 1)
5,5 - Kernel size of the filter (5x5)

B. Show the effect of the filters

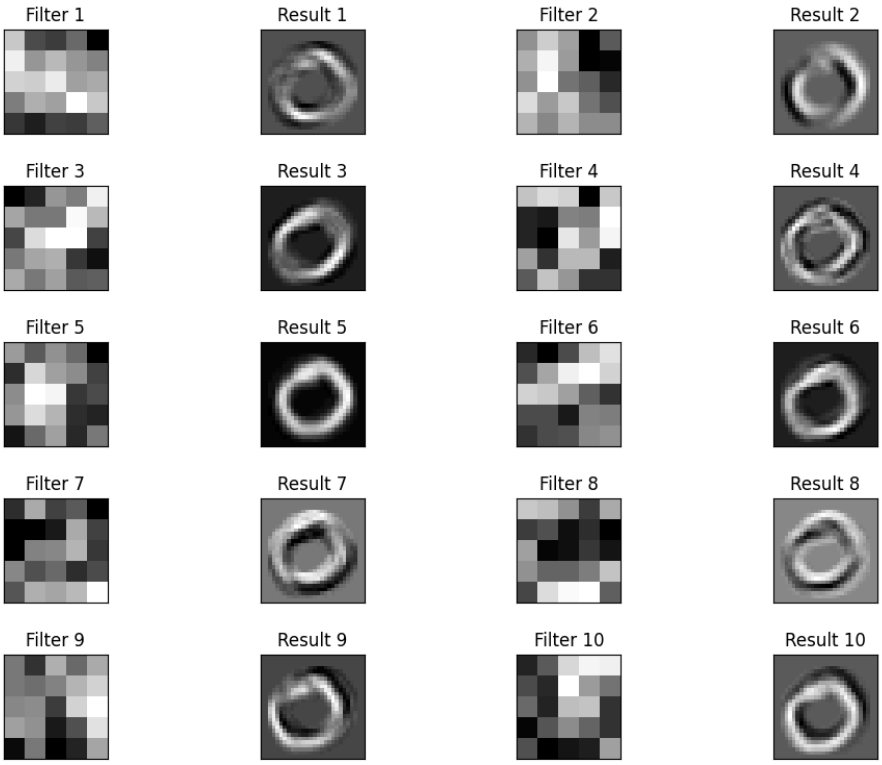


Image showing a plot of the 10 filtered images along with their corresponding filters

Upon examining the results, they make sense given the nature of the filters. Each filter appears to highlight or emphasize certain features or patterns in the input image. Filters 1, 2, 6 and 10 seem to be detecting horizontal and vertical edges, while filters 3, 4, 7, and 8 appear to be detecting diagonal or curved patterns. Filter 5 segments the image into a black background and a white digit. Filter 9 seems to be detecting an irregular pattern which could be useful in detecting texture features. The results clearly show how each filter can extract and emphasize specific characteristics or features from the input image. These filtered outputs can then be used as input to subsequent neural network layers for further processing and analysis.

Task 3 - Transfer Learning on Greek Letters:

```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool1): 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)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
```

Image showing a print of the modified network

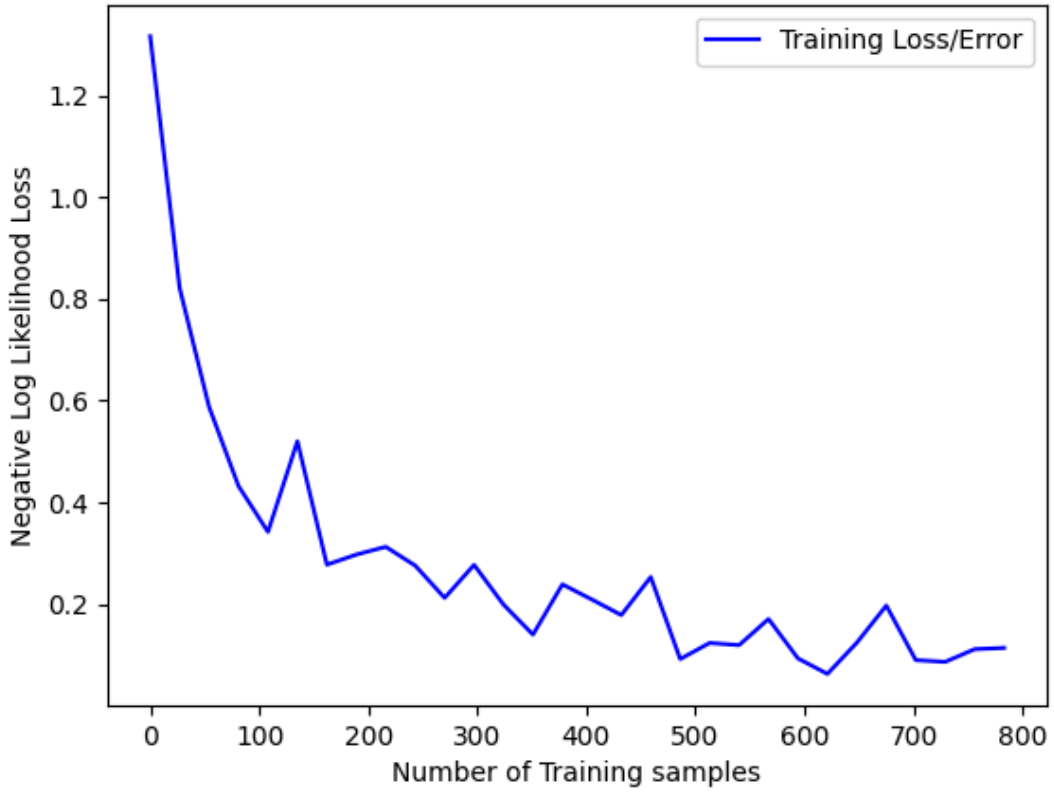


Image showing the training error for the Greek letters dataset plotted against the number of training samples

The above graph is obtained when the number of epochs set for training was 30. As per the logs generated in the code, the model requires 7 epochs to achieve 100% accuracy on the training set. **Therefore, it takes 7 epochs to perfectly identify the Greek letters using the 27 examples in the training dataset folder.**

Factoring this, we decided to use 15 epochs for model training and then evaluate it on a new dataset of Greek letters for testing.

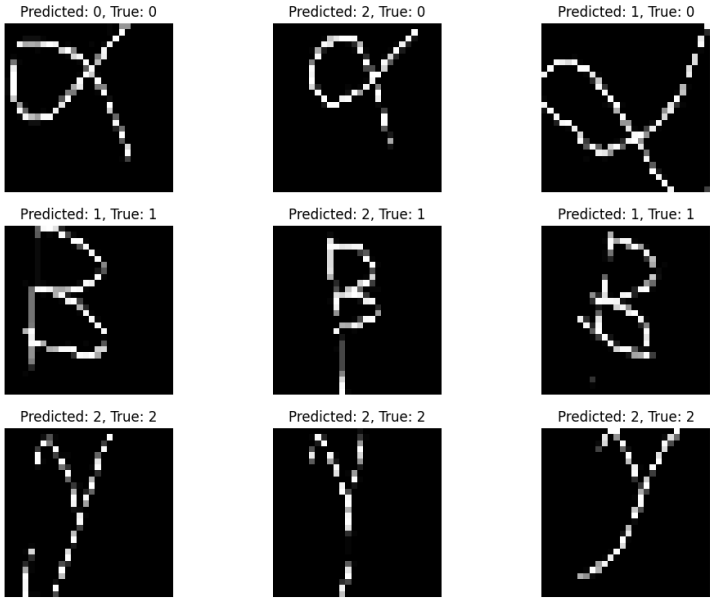


Image showing the prediction results of our newly trained model on the additional data

Meaning of Labels: 0 (alpha), 1 (beta) and 2 (gamma).

We created the test dataset by resizing the images to 128x128 before loading them to the dataloader. The above image shows that the model has correctly classified 1 out of 3 alpha, 2 out of 3 beta, and all 3 gamma. **The accuracy of the newly trained model on the test dataset is 66.67%.**

The accuracy is quite low and can be improved by increasing the size of the training dataset.

Task 4/Extension-1 - Design your own experiment:

We have selected the MNIST Fashion Dataset for this task. The MNIST Fashion dataset, with its increased complexity compared to the standard MNIST digits dataset, presents an excellent opportunity to examine the impact of architectural modifications on the model's performance.

A. Develop a plan

The goal of this task is to analyze the effect of changing the following four dimensions on the neural network. **Since we are evaluating 4 dimensions instead of 3, we will be submitting this as Extension-1 as well**:

1) The number of convolution filters in a layer
2) The size of the convolution filters
3) Whether to add another dropout layer after the fully connected layer
4) The dropout rates of the Dropout layer

To carry out this task, we have divided the analysis into 2 experiments:

Experiment-1: The no. of filters and the size of the filters

Combinations to explore:

We have determined 3 options to evaluate for the size of filters in the first layer: 3, 5 & 7.

We have determined 3 options to evaluate for the size of filters in the second layer: 3, 5 & 7.

For the number of filters in the first layer, we have determined these 4 options: 5, 10, 15 & 20.

The number of filters in the second layer is double that of the first. So, the options are: 10, 20, 30 & 40.

For this experiment, there will be a total of 3*3*4 = 36 combinations, resulting from the different options available for the size of filters in the first and second layers, as well as the number of filters in each layer.

Evaluation metrics:

We will evaluate the performance of the network on these metrics: Average Loss and Accuracy on the Test Dataset.

Experiment-2: The addition of dropout layer and the dropout rates of dropout layer

Combinations to explore:

We have determined 10 options to evaluate for the dropout rates of the dropout layer: 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 & 0.9.

We have determined 2 options to evaluate for the addition of dropout layer after the fully connected layer: add with 0.5 dropout rate, or not add, i.e., 0 dropout rate.

For this experiment, there are a total of 10*2 = 20 combinations resulting from these options.

Evaluation metrics:

Same as in experiment-1, we will evaluate the performance of the network on these metrics: Average Loss and Accuracy on the Test Dataset.

B. Predict the results

Hypothesis for Experiment-1:

Size of filters: Smaller filter sizes may help the network generalize better to new, unseen data, as they capture more localized patterns like edges and textures that are likely to be common across different parts of the input space. So, it's better to have smaller-size filters.

Number of filters: Increasing the number of filters allows the network to capture more diverse and complex features from the input data. So, it's favorable to have more filters.

Based on this, the best combinations amongst all the variations will be the first layer with 20 filters of 3x3 kernel size, and the second layer with 40 filters of 3x3 kernel size.

Hypothesis for Experiment-2:

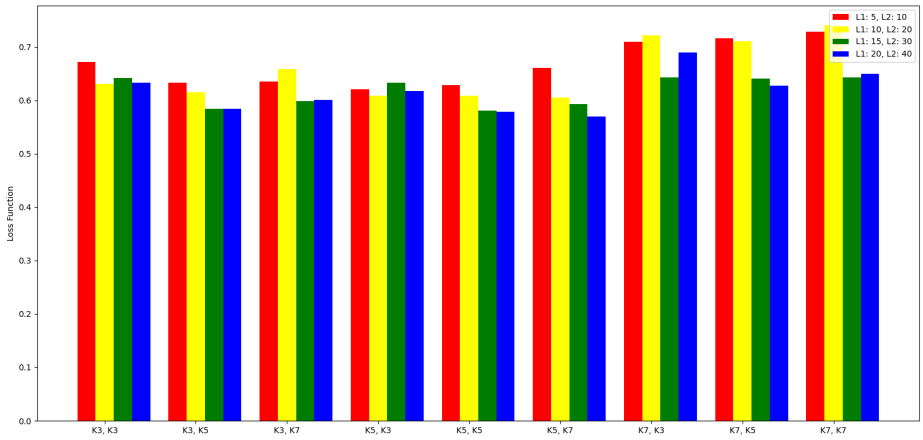
Dropout Rate: The range of the optimal dropout rate is 0.4 - 0.6. This is because too low a dropout rate can cause overfitting, and too high a dropout rate limits the ability of the network to learn from training data.

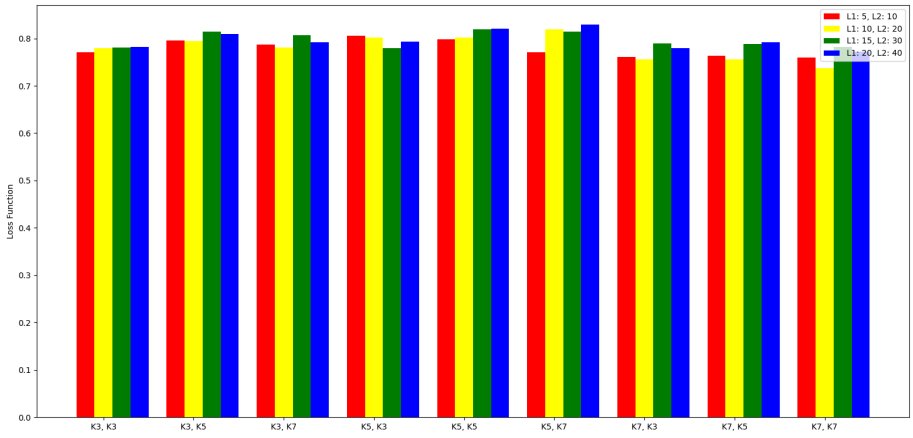
Addition of a dropout layer: Having no dropout layer will lead to overfitting as there is no regularization. Adding a dropout layer will lead to regularization and improved performance. By doing this, we expect the average loss to decrease and accuracy to increase.

Based on this, the best combination amongst all the variations will be the case when we have added a dropout layer after the fully connected layer with a dropout rate of 0.5.

C. Execute your plan

Experiment-1:





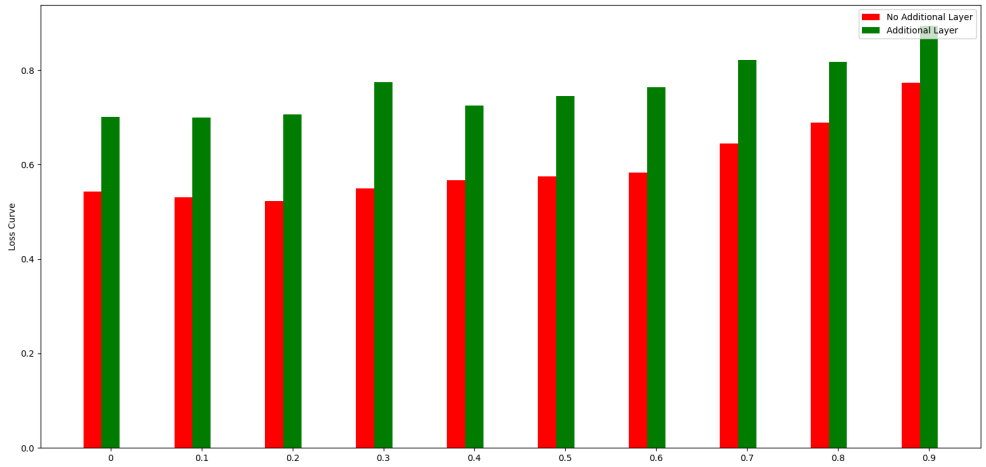
Images showing the results of experiment-1 - the comparison of average test loss under different combinations for Layer-1 & Layer-2 respectively

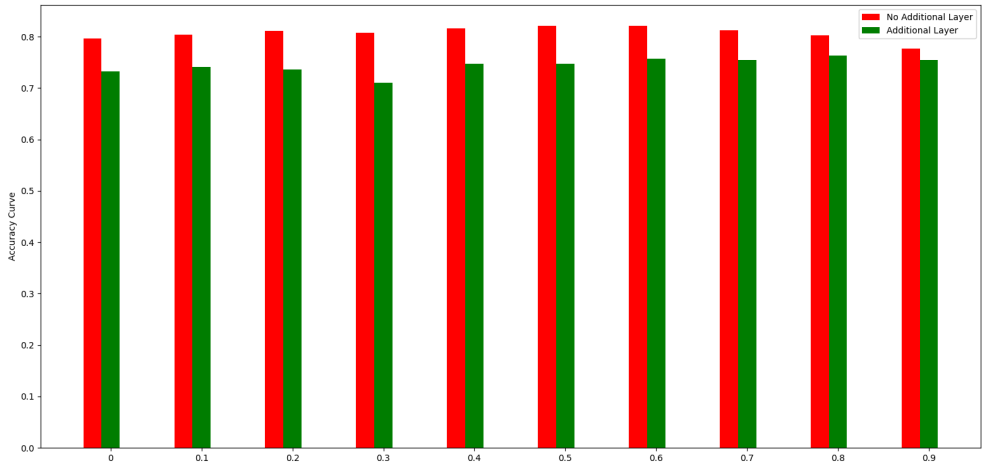
From the above results, we can see that the average loss on the test set generally increases with the increase in the size of the filters. On average, the combinations which contain 7 have a high loss value as compared to 3 & 5. **Therefore, the results support the hypothesis that increasing the size of the filters is not a good choice.**

On the other hand, increasing the number of filters causes the loss function to decrease in most of the cases. **Therefore, the results support the hypothesis that increasing the number of filters is a good choice.**

From the results obtained by executing this experiment, we found that the accuracy for the combinations proposed in the hypothesis (L1:20, K3x3, L2:40, K3x3), is 0.8088 and the average loss is 0.5847. The optimal combinations which were calculated by the code turn out to be [L1:20, K5x7, L2:40, K7x7] with an accuracy of 0.8288 and an average loss of 0.5702. **Therefore, in terms of the best combinations amongst all the variations, the results reject the hypothesis in this case. This could be happening because when the size of the filters increases, the capacity of the model to learn and represent complex relationships within the data also increases.**

Experiment-2:





Images showing the results of experiment-2 - the plots of average test loss and average test accuracy respectively against the dropout rate under different combinations

From the above results, we can see that the average loss on the test set increases initially from 0.1 to 0.3 dropout rate, then decreases in the middle range value of 0.4, and then increases to 0.9 dropout rate. The accuracy also increases from 0 to middle range values with it's peak at 0.5, and then decreases from 0.5 to 0.9. **Therefore, the results support the hypothesis that the optimal range of dropout rate is 0.4 - 0.6.**

On the other hand, we can see from the graph that adding an additional dropout layer after the fully connected layer causes the average test accuracy to decrease, and the average test loss to increase. **Therefore, the results reject the hypothesis that adding a dropout layer will lead to regularization and improved performance. This could be happening as the additional dropout layer may excessively regularize the model, hindering its capacity to learn and generalize effectively.**

From the results obtained by executing this experiment, we found that the best configuration is when the dropout rate is 0.2 with no additional dropout layer after the fully connected layer with an average loss of 0.5229 and average test accuracy of 0.8205. The average test loss and accuracy of the combination proposed in the hypothesis, i.e., dropout rate of 0.5 and an additional dropout layer is 0.5746 and 0.8205 respectively. **Therefore, in terms of the best combinations amongst all the variations, the results reject the hypothesis in this case. The lower dropout rate of 0.2 may provide sufficient regularization to prevent overfitting without excessively hindering model learning, leading to better performance. Additionally, the inclusion of an extra dropout layer with a higher dropout rate of 0.5 may introduce excessive noise during training, impairing the model's ability to learn effectively.**

3) Extensions:

Extension-1/Task-4 - Design your own experiment:

We evaluated 4 dimensions along which to change the network architecture for performance optimization instead of 3 which were mentioned in the task description. This is our extension-1.

Extension-2 - Replace the first layer of the MNIST network with a Gabor Filter bank:

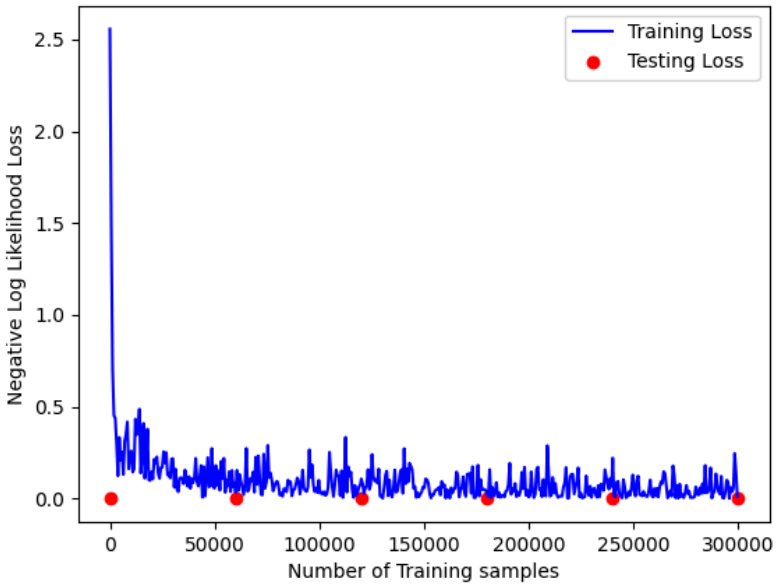


Image showing a plot of the testing and training losses against the number of training samples when using Gabor filters

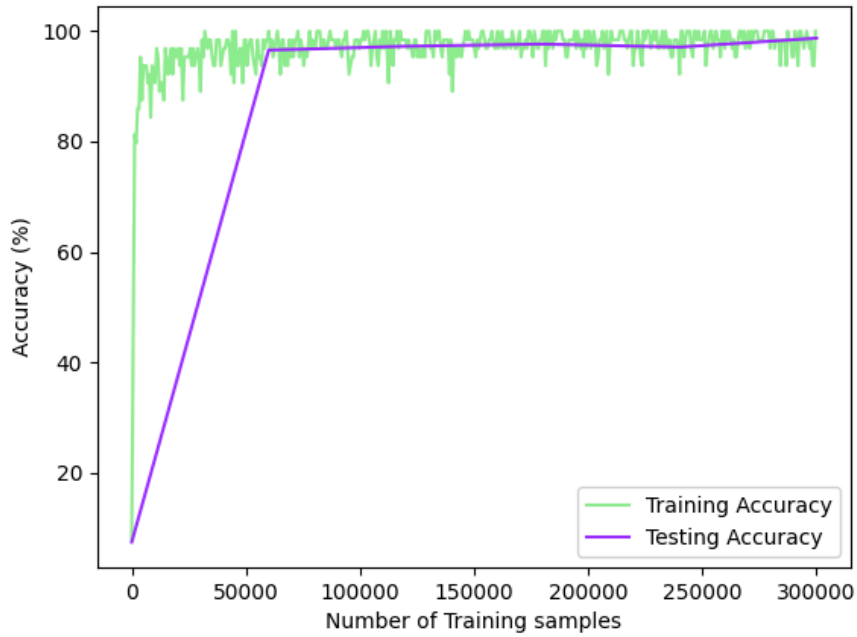


Image showing a plot of the testing and training accuracies against the number of training samples when using Gabor filters

From the above plots, we can conclude the following about loss function and accuracy:

1. **Training and Testing Loss:** The initial decrease in training loss suggests a strong transfer learning effect, where knowledge gained from pre-training the first layer with Gabor filters significantly accelerates the learning process of the subsequent layers. The stability of the testing loss throughout training suggests that the model generalizes well to unseen data. This indicates that the features learned by the Gabor filters are not only effective for the training data but also generalize well to the testing data, resulting in consistent performance across both datasets.
2. **Training and Testing Accuracy:** Both the training and testing accuracies reach 100% within the first epoch. This suggests that the model capacity may be sufficient to capture the complexity of the MNIST dataset relatively quickly. Once the model has learned the discriminative features from the Gabor filters, further training epochs may not significantly improve performance, leading to stabilized training and testing accuracies.

Therefore, the use of Gabor filters helps the model learn quickly and generalize well to similar tasks but also causes the model's performance to plateau or saturate at an early stage due to the strong discriminative power of the extracted features.

Extension-3 - Examine a different pre-trained network:

A. Analyze the first layer

For this extension, we have decided to use the pre-trained ResNet-18 model from PyTorch's model zoo collection of pre-trained models and model architectures.

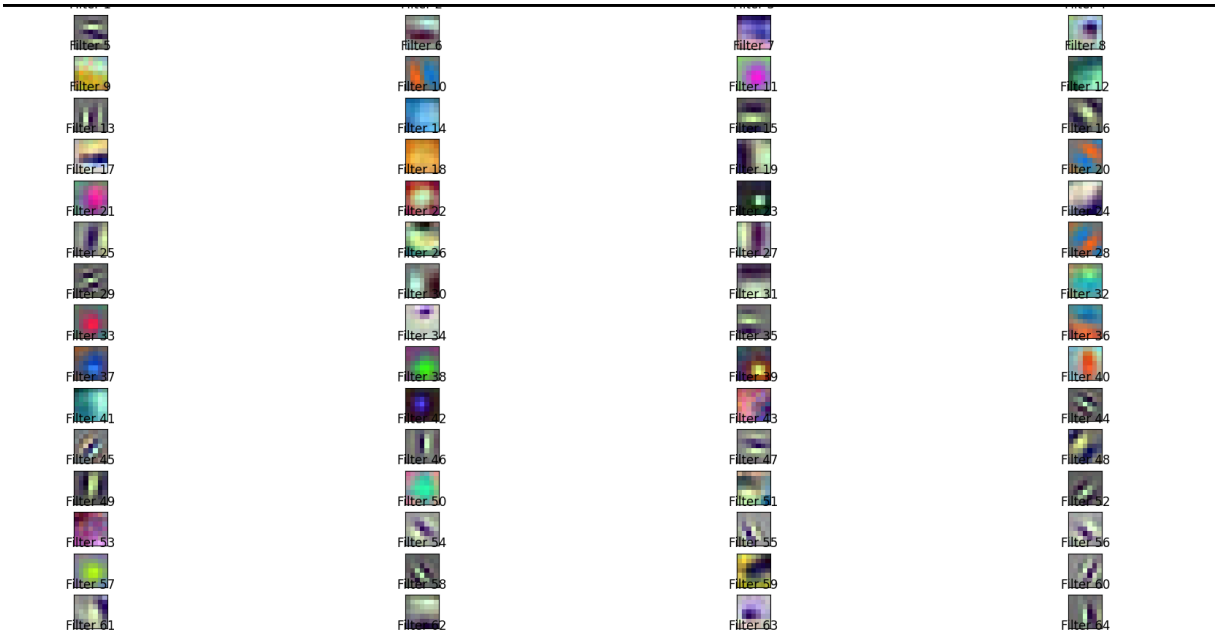


Image showing a 16 x 4 grid of plots for the filters to visualize the weights of the first layer of the ResNet-18 network

When we printed the model in the code, we got the following output showing the structure of the ResNet-18 network and the name of each layer:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Images showing the output of printing the ResNet-18 model

When we printed the shape of the weights, we got this:

```
Shape of weights: torch.Size([64, 3, 7, 7])
```

This is the description of the above values:

64 - Number of output channels (no. of filters)

3 - Number of input channels (For RGB images, it's 3)

7,7 - Kernel size of the filter (7x7)

B. Show the effect of the filters

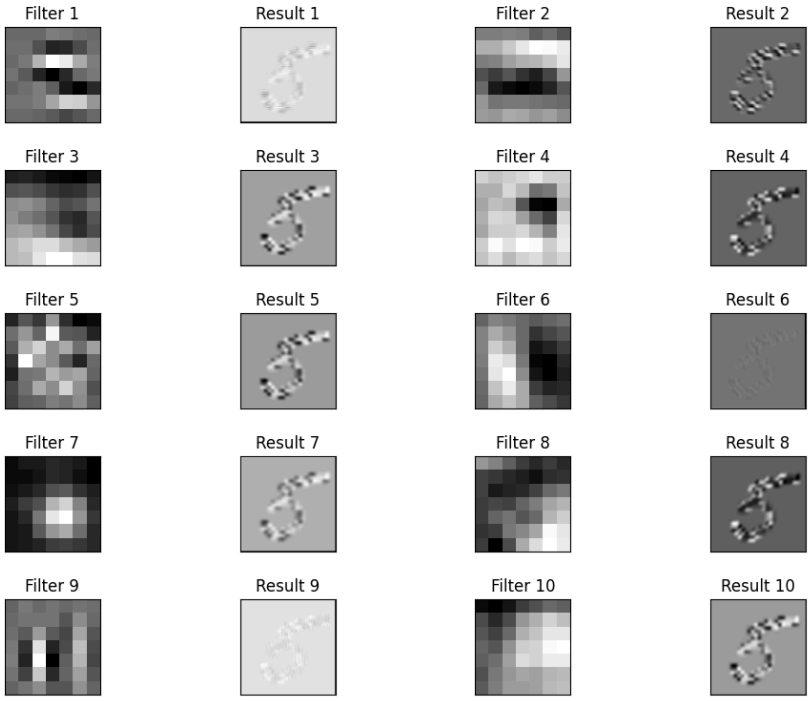


Image showing a plot of 10 filtered images along with their corresponding filters

We have applied the first 10 filters of the first layer of ResNet-18 on an image from the MNIST digits training dataset to examine their effects.

Each filter appears to highlight or emphasize certain features or patterns in the input image. Filters 4 and 10 are detecting horizontal and vertical edges. Filters 1, 6 & 9 are blur filters which are capturing the most basic & global features of the images. Filters 2 & 3 extract more fine-grained details and texture features. Filters 5 & 8 are smoothing filters causing highly pixelated images which capture the low-frequency components and features. Filter-7 is an averaging filter, capable of extracting only the most fundamental shapes and textures from the image. Filtered outputs from each layer aid subsequent neural network processing, showcasing hierarchical feature extraction in deep learning. This automated learning of relevant representations enhances performance without manual feature engineering, culminating in improved image recognition and classification accuracy.

4) Short reflection on learnings:

Through this project, we gained practical experience in building and training convolutional neural networks for digit recognition using the MNIST dataset. Working with PyTorch, we developed skills in neural network architecture design, data visualization, and model evaluation. Additionally, exploring transfer learning expanded my understanding of adapting pre-trained models to new tasks. Overall, this project deepened our knowledge of deep learning concepts and provided valuable hands-on experience in the development of deep learning based computer vision applications.

5) Acknowledgement:

We have referred to the following sources to complete this project:

1. https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html
2. https://pytorch.org/tutorials/beginner/basics/intro.html
3. Computer Vision: Algorithms and Applications by Richard Szeliski
4. https://nextjournal.com/gkoehler/pytorch-mnist
5. www.google.com
6. Prof. Bruce Maxwell - Code snippets provided in the Canvas project description, and responses provided to students in Piazza