CS 5330 – PROJECT 5
Recognition using Deep Networks

Authors:

Name: Ravi Shankar Sankara Narayanan

NUID: 001568628

Short description:

This project has multiple files that works on using deep neural networks for digit recognition and Greek letter recognitions. The tasks involve creating neural network model, training the model, plotting the accuracies and scores, extracting the layer and filter information, predicting on hand-written digits and letters and finally hyper parameter tuning for the model.

Files and descriptions:

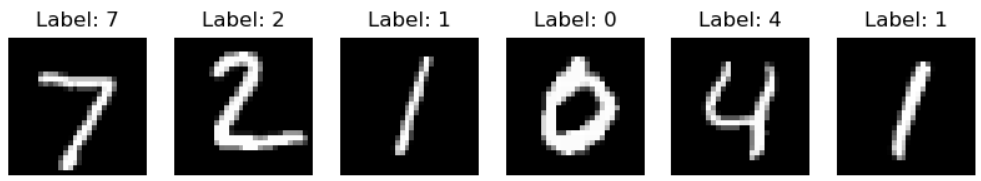
- myNetwork.py: contains the neural network architecture used in Task 1,2,3.
- task1_main.py: contains the solutions for a, b, c, and d subtasks under Task 1.
- task1e_main.py: contains the solution for Task 1, e subtask.
- task1f_main.py: contains the solution for Task 1, f subtask.
- task2_main.py: contains the solution for Task 2.
- task3_main.py: contains the solution for Task 3.
- task4_main.py: contains the solutions for Task 4 and extension 2.
- ext1.py : contains the code for extension 1

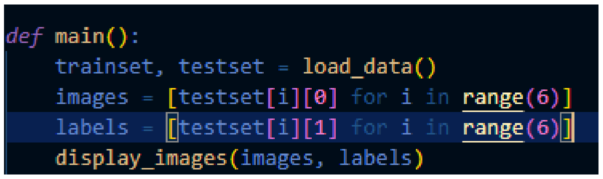
Tasks and Outputs:

Task 1: Build and train a network to recognize digits

A. Get the MNIST digit data set.

The program successfully reads the MNIST dataset and the image containing the first six digits is attached below:



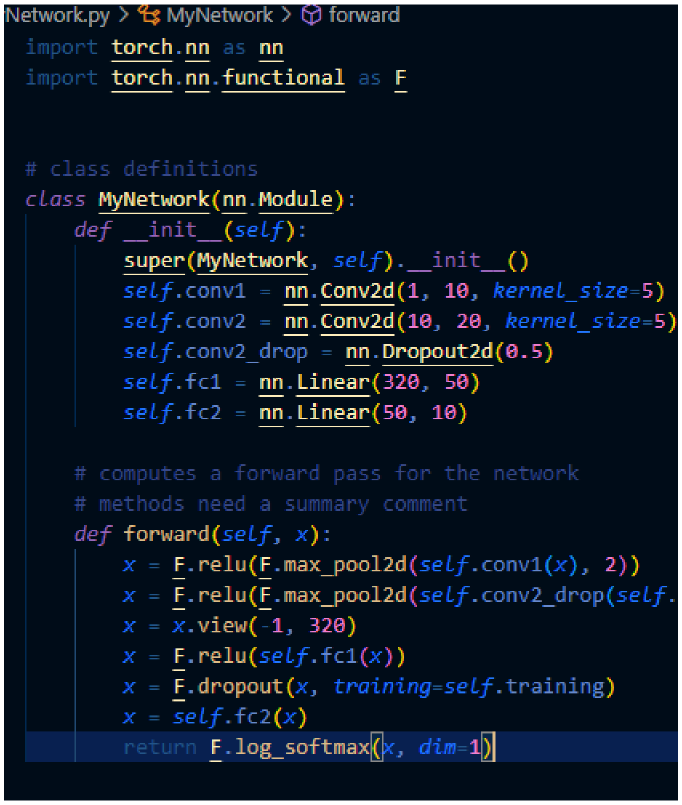


```python
def main():
    trainset, testset = load_data()
    images = [testset[i][0] for i in range(6)]
    labels = [testset[i][1] for i in range(6)]
    display_images(images, labels)
```

In my program I have used two functions to load and display the results. The load_data() function will fetch the train and test data and the display_images function will display the images and their labels.

B. Build a network model:

Since I'll be implementing this model in multiple places, I have created a separate python file for the architecture so that I can import it in other tasks. The code for the neural network is attached below:



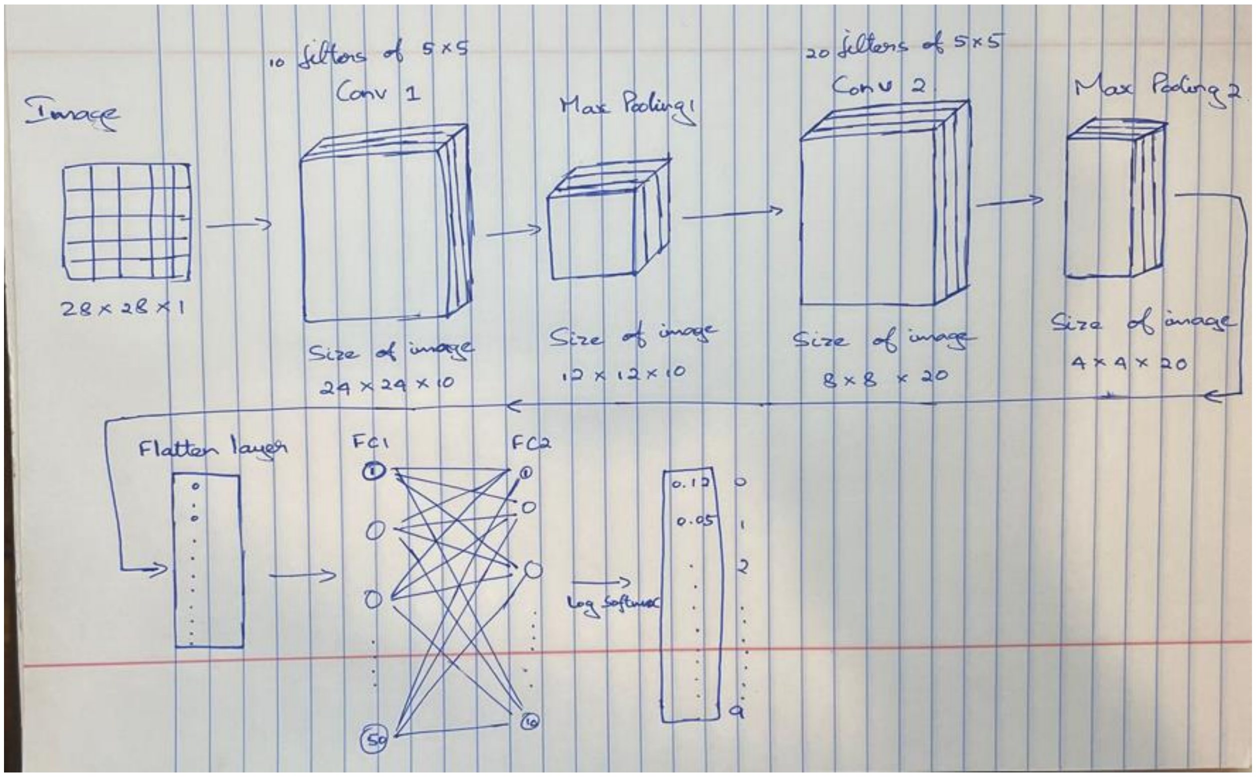
Network.py > MyNetwork > forward

```python
import torch.nn as nn
import torch.nn.functional as F


# class definitions
class MyNetwork(nn.Module):
    def __init__(self):
        super(MyNetwork, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    # computes a forward pass for the network
    # methods need a summary comment
    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

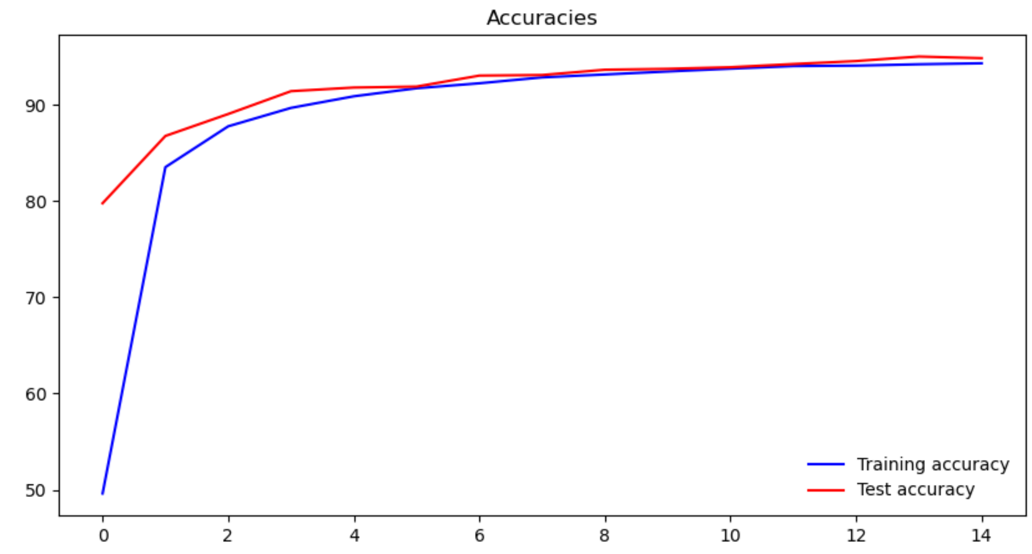
A simple and abstract hand drawn architecture of the above model is provided below:



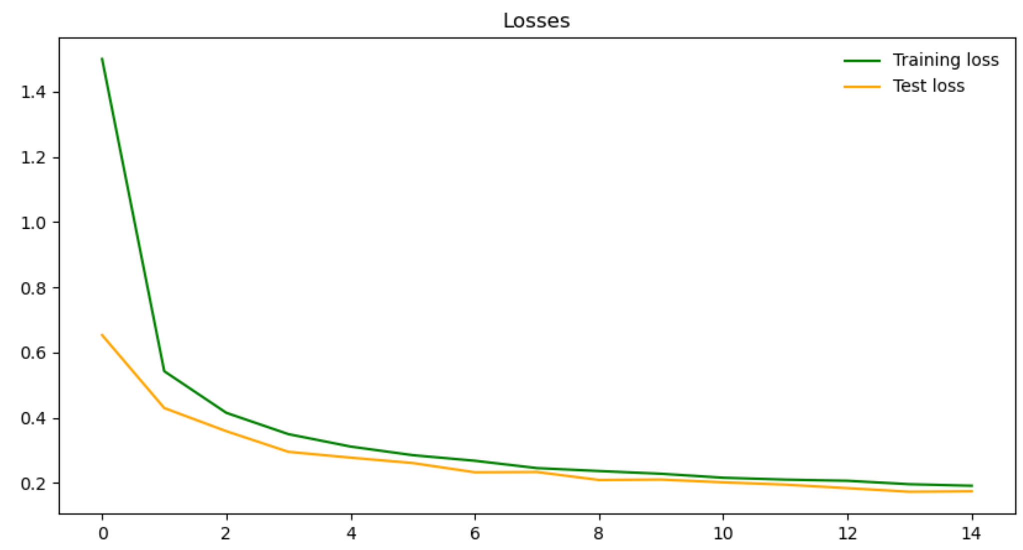
At the end after applying the log softmax we have vector of size ten that contains the probabilities of each class.

C. Train the model:

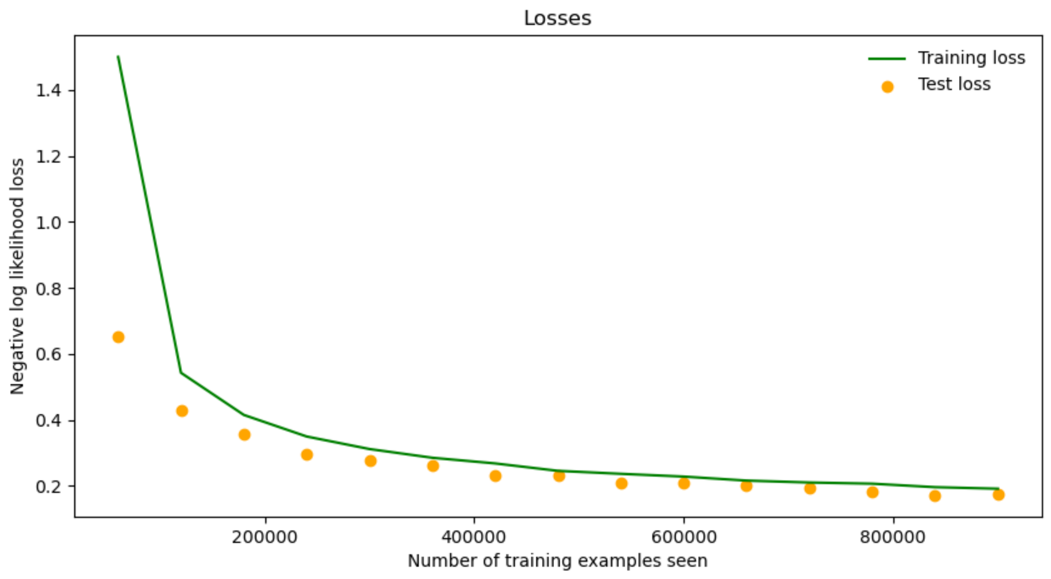
The model was trained for 15 epochs, and the resulting graphs are attached below:



In this graph I am plotting accuracy of the model against each epoch. From the above graph we can observe that the accuracy increases with each epoch and the test accuracy is also not far behind. Based on the accuracy graph it is safe to assume that the model is neither overfitting nor underfitting.

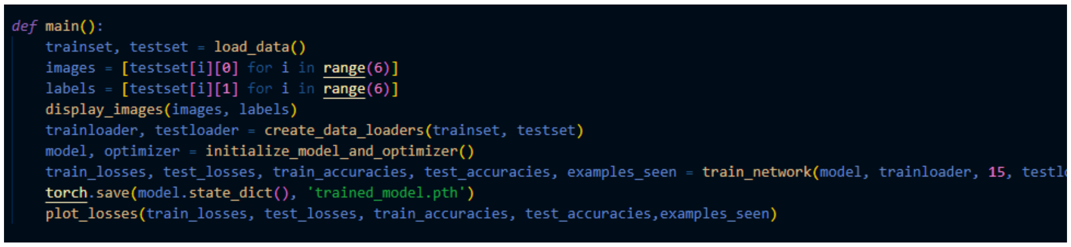


In the above graph, I have plotted the loss in each epoch, and we can observe that the loss decreases with each epoch which is also the intended behavior. The test loss also decreases with each epoch.



In the above graph I am plotting the negative log likelihood loss against the number of training examples seen. The loss decreases with the higher the number of training examples the model has seen.

D. Save the network to a file:

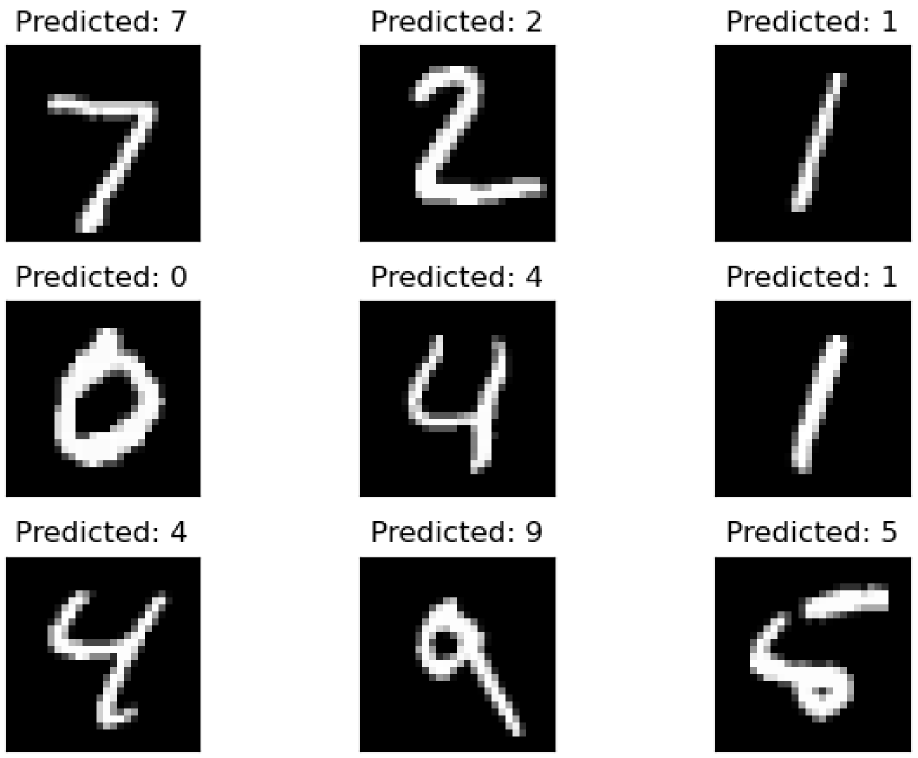


```
def main():
    trainset, testset = load_data()
    images = [testset[i][0] for i in range(6)]
    labels = [testset[i][1] for i in range(6)]
    display_images(images, labels)
    trainloader, testloader = create_data_loaders(trainset, testset)
    model, optimizer = initialize_model_and_optimizer()
    train_losses, test_losses, train_accuracies, test_accuracies, examples_seen = train_network(model, trainloader, 15, testl
    torch.save(model.state_dict(), 'trained_model.pth')
    plot_losses(train_losses, test_losses, train_accuracies, test_accuracies,examples_seen)
```

I have saved the model after training in a .pth file.

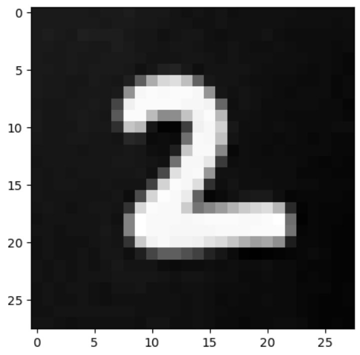
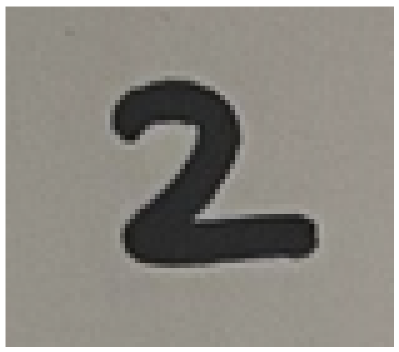
E. Read the network and run it on the test set.

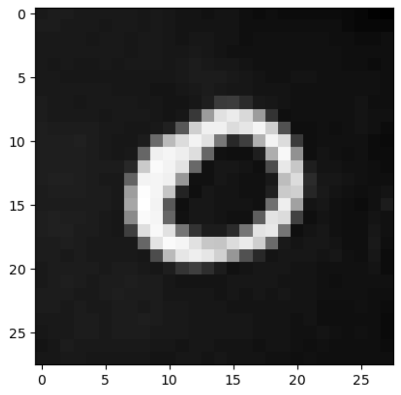
The trained network was loaded into another program and was used to predict the first 10 examples of the test set. The results are correct and is attached below:

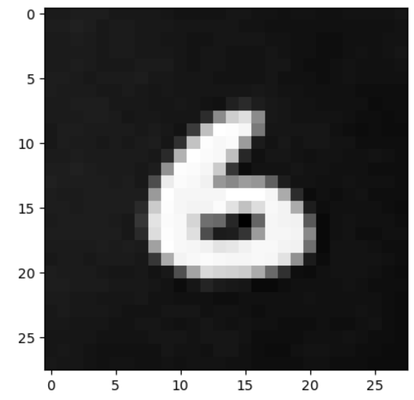


F. Test the network on new inputs.

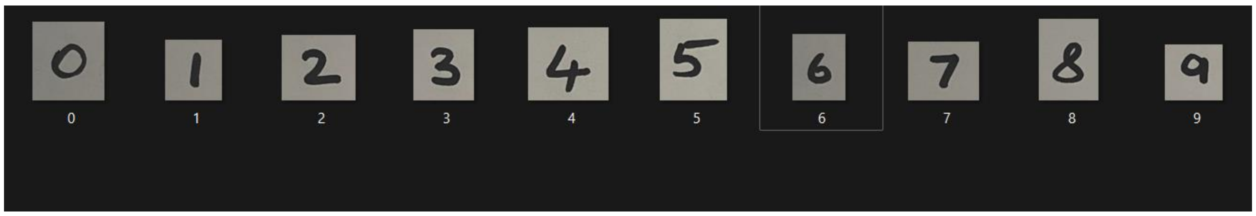
I wrote the numbers on a blank paper and provided the images to the program. The program performed the necessary preprocessing steps so that the input will look like the images from the MNIST dataset. The original images and the processed images are attached below:

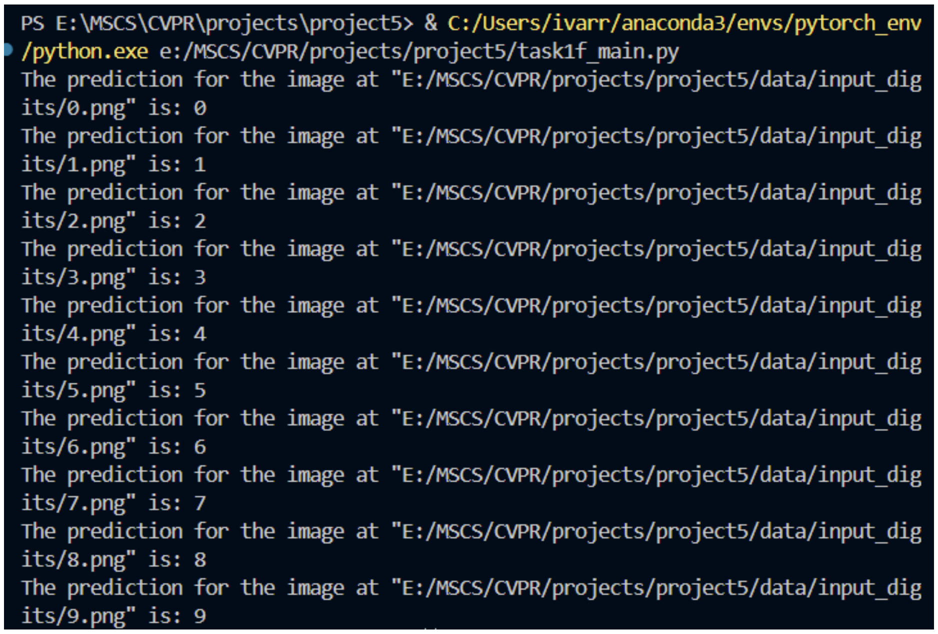







After the preprocessing the saved network from the previous task was used to predict on these new images and all the results were correct. The output from the terminal and file names are attached below:





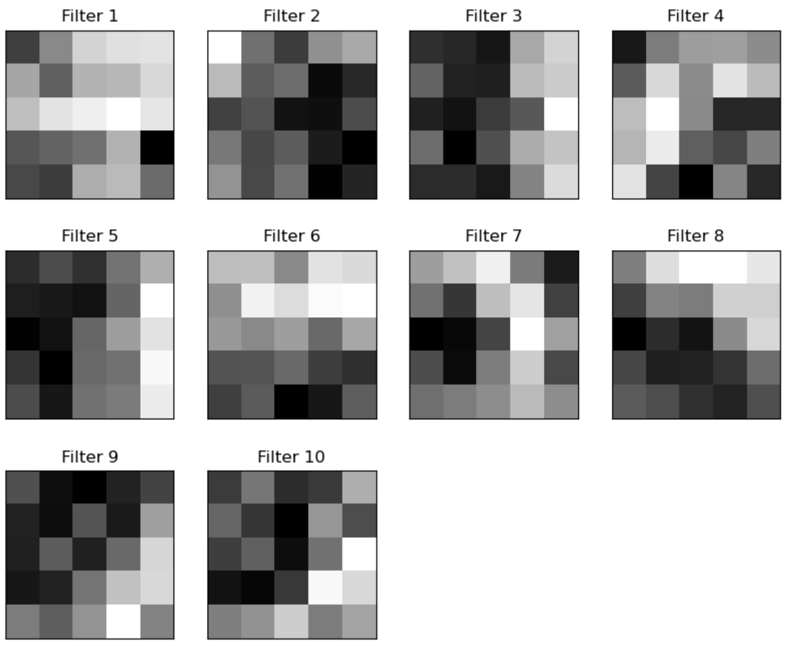
```
PS E:\MSCS\CVPR\projects\project5> & C:/Users/ivarr/anaconda3/envs/pytorch_env
/python.exe e:/MSCS/CVPR/projects/project5/task1f_main.py
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/0.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/1.png" is: 1
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/2.png" is: 2
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/3.png" is: 3
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/4.png" is: 4
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/5.png" is: 5
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/6.png" is: 6
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/7.png" is: 7
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/8.png" is: 8
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_dig
its/9.png" is: 9
```

On comparing the file names mentioned in the terminal and the actual file names we can observe that the model has predicted the images correctly.

Task 2: Examine your network

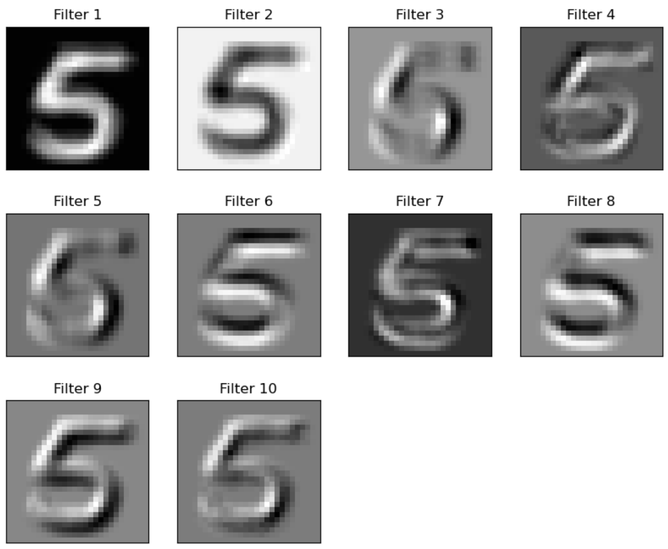
a. Analyze the first layer:

The first layer that has 10 5x5 filters are visualized in the plot below:



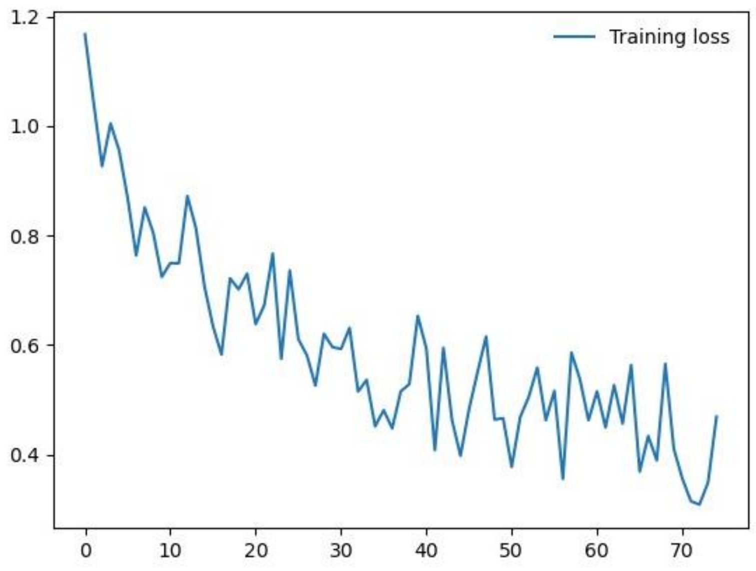
b. Show the effect of the filters.

The effects of the filters on an image are attached below:

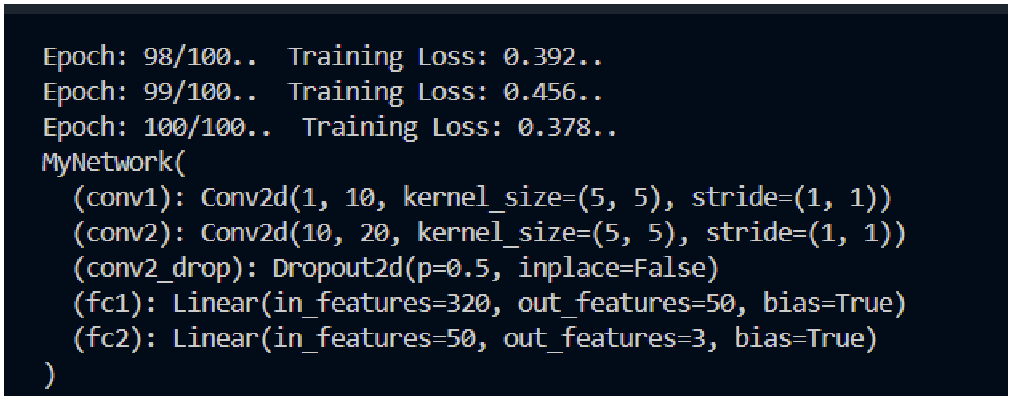


Task 3: Transfer Learning on Greek Letters

For this task, the model saved in task 1 was used. It was trained on the provided data for 75 epochs. Since the provided input data was small, I added data augmentations on the fly to improve the accuracy. The loss was plotted against the epochs and the graph is attached below:

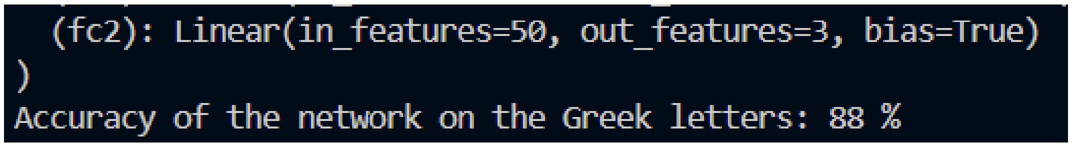


The training loss decreases but still there is some fluctuation present when it decreases. The model summary is present below:



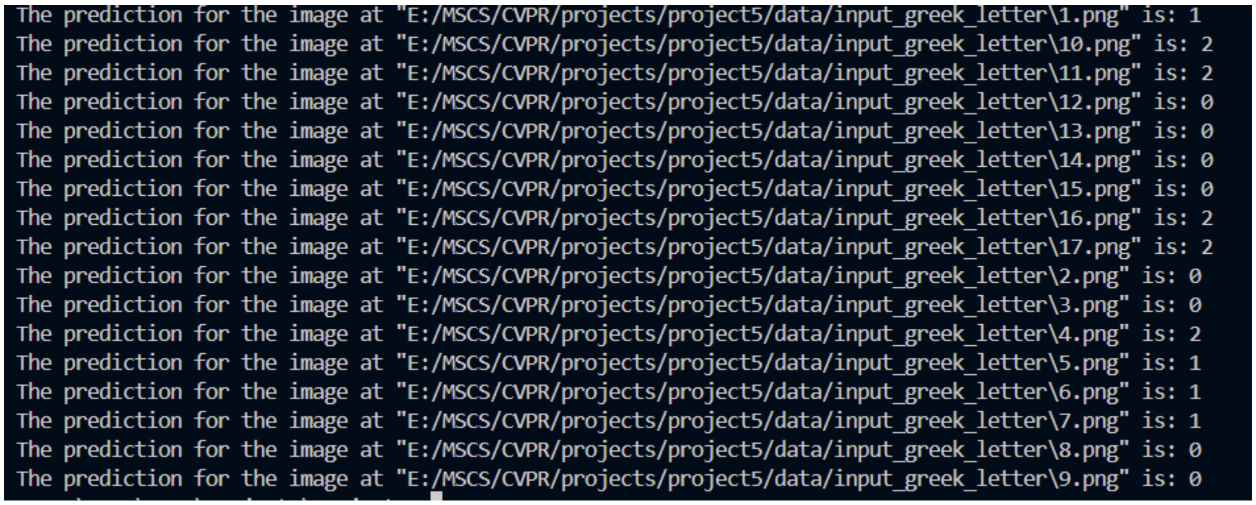
```
Epoch: 98/100..  Training Loss: 0.392..
Epoch: 99/100..  Training Loss: 0.456..
Epoch: 100/100..  Training Loss: 0.378..
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
```

And the initial accuracy after training on the Greek letters is over **85%.**

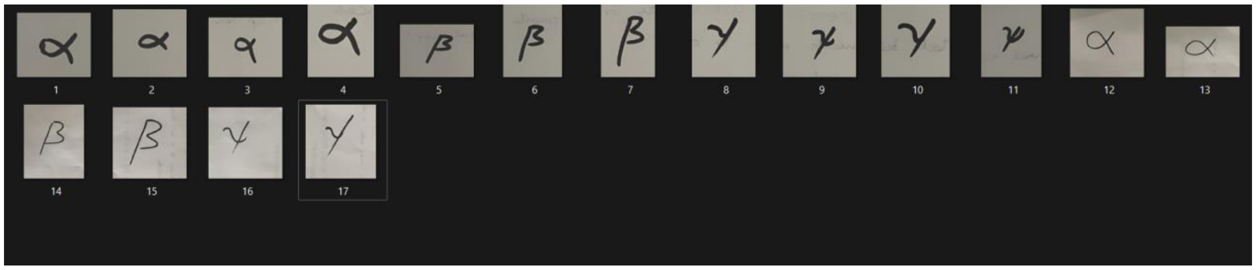


```
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
Accuracy of the network on the Greek letters: 88 %
```

I also tested the model with images of Greek letters that I wrote. The results are attached below:



```
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\1.png" is: 1
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\10.png" is: 2
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\11.png" is: 2
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\12.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\13.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\14.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\15.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\16.png" is: 2
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\17.png" is: 2
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\2.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\3.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\4.png" is: 2
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\5.png" is: 1
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\6.png" is: 1
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\7.png" is: 1
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\8.png" is: 0
The prediction for the image at "E:/MSCS/CVPR/projects/project5/data/input_greek_letter\9.png" is: 0
```



On comparing the result from the terminal with the file name in my input folder, we can observe that the model got more one instance of each class correctly. Here the three letters are represented as 0,1 and 2. 0 means alpha, 1 means beta and 2 stands for gamma.

Task 4: Design your own experiment

The goal of this task is to explore the impact of different hyperparameters on the performance of a convolutional neural network. The hyperparameters I decided to explore were the number of convolution filters, the size of these filters, and the dropout rate. I decided to explore two options for each of these parameters:

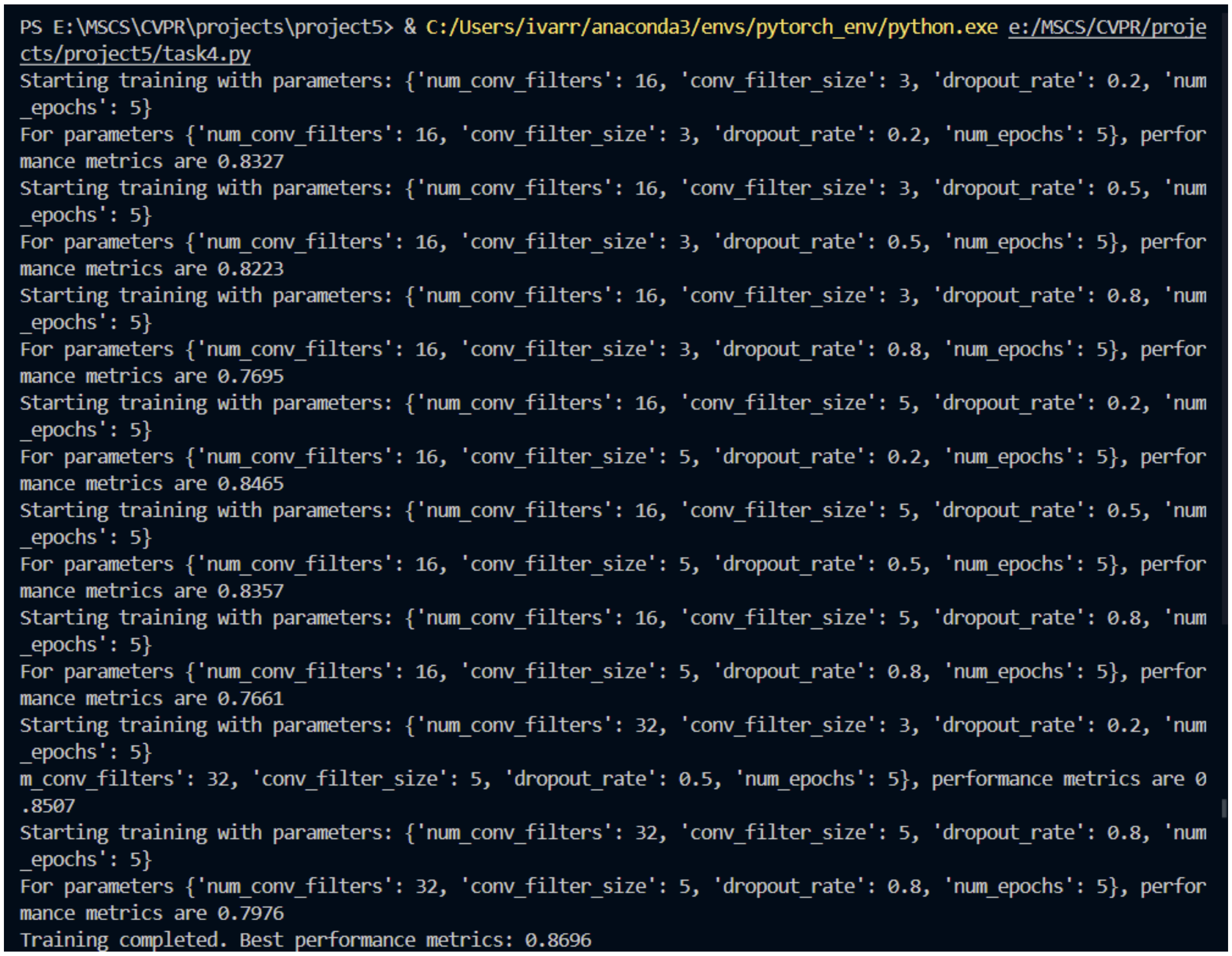
- Number of convolution filters: 16, 32
- Convolution filter size: 3, 5
- Dropout rate: 0.2, 0.5, 0.8

As instructed, I used a linear search strategy, holding two parameters constant while optimizing the third.

Before starting the evaluation, I hypothesized that:

- Increasing the number of convolution filters would improve the performance of the network, as it would be able to learn more complex features from the input data.
- Increasing the size of the convolution filters would also improve performance, as larger filters can capture larger-scale features.
- Increasing the dropout rate would initially improve performance by reducing overfitting, but beyond a certain point it would harm performance by causing underfitting.

For each combination, I printed the score:



```
PS E:\MSCS\CVPR\projects\project5> & C:/Users/ivarr/anaconda3/envs/pytorch_env/python.exe e:/MSCS/CVPR/proje
cts/project5/task4.py
Starting training with parameters: {'num_conv_filters': 16, 'conv_filter_size': 3, 'dropout_rate': 0.2, 'num
_epochs': 5}
For parameters {'num_conv_filters': 16, 'conv_filter_size': 3, 'dropout_rate': 0.2, 'num_epochs': 5}, perfor
mance metrics are 0.8327
Starting training with parameters: {'num_conv_filters': 16, 'conv_filter_size': 3, 'dropout_rate': 0.5, 'num
_epochs': 5}
For parameters {'num_conv_filters': 16, 'conv_filter_size': 3, 'dropout_rate': 0.5, 'num_epochs': 5}, perfor
mance metrics are 0.8223
Starting training with parameters: {'num_conv_filters': 16, 'conv_filter_size': 3, 'dropout_rate': 0.8, 'num
_epochs': 5}
For parameters {'num_conv_filters': 16, 'conv_filter_size': 3, 'dropout_rate': 0.8, 'num_epochs': 5}, perfor
mance metrics are 0.7695
Starting training with parameters: {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.2, 'num
_epochs': 5}
For parameters {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.2, 'num_epochs': 5}, perfor
mance metrics are 0.8465
Starting training with parameters: {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.5, 'num
_epochs': 5}
For parameters {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.5, 'num_epochs': 5}, perfor
mance metrics are 0.8357
Starting training with parameters: {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.8, 'num
_epochs': 5}
For parameters {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.8, 'num_epochs': 5}, perfor
mance metrics are 0.7661
Starting training with parameters: {'num_conv_filters': 32, 'conv_filter_size': 3, 'dropout_rate': 0.2, 'num
_epochs': 5}
m_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.5, 'num_epochs': 5}, performance metrics are 0
.8507
Starting training with parameters: {'num_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.8, 'num
_epochs': 5}
For parameters {'num_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.8, 'num_epochs': 5}, perfor
mance metrics are 0.7976
Training completed. Best performance metrics: 0.8696
```

The best performance was achieved with 32 convolution filters, a filter size of 5, and a dropout rate of 0.2, achieving a performance metric of 0.8696.

Increasing the number of convolution filters from 16 to 32 generally improved performance, supporting my first hypothesis.

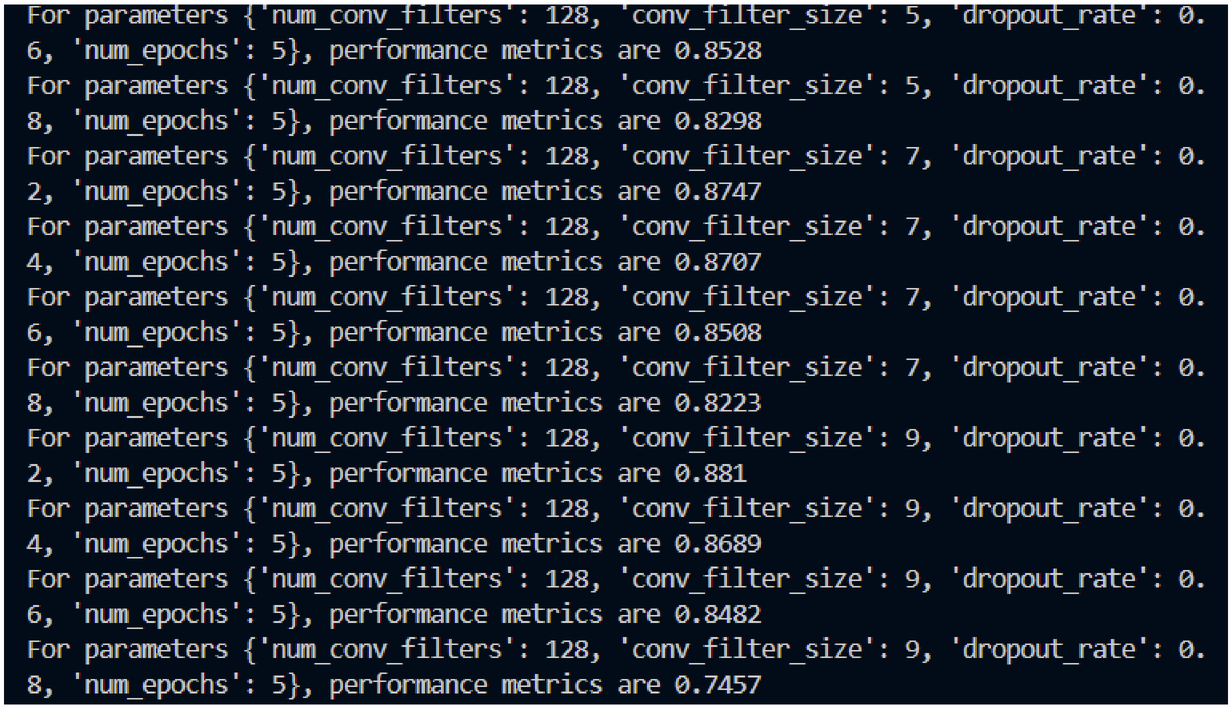
Increasing the convolution filter size from 3 to 5 also generally improved performance, supporting my second hypothesis.

The dropout rate of 0.2 generally outperformed 0.5 and 0.8, suggesting that my third hypothesis was partially correct: while some dropouts improved performance, too much dropout harmed performance.

Now I will increase the number of values for each of the three parameters so that we will have greater than 50 variations. The new parameter grid is:

- Number of convolution filters: 16, 32, 64,128.
- Convolution filter size: 3, 5, 7, 9.
- Dropout rate: 0.2, 0.4, 0.6, 0.8.

This will give me greater than 50 different network variations.



```
For parameters {'num_conv_filters': 128, 'conv_filter_size': 5, 'dropout_rate': 0.
6, 'num_epochs': 5}, performance metrics are 0.8528
For parameters {'num_conv_filters': 128, 'conv_filter_size': 5, 'dropout_rate': 0.
8, 'num_epochs': 5}, performance metrics are 0.8298
For parameters {'num_conv_filters': 128, 'conv_filter_size': 7, 'dropout_rate': 0.
2, 'num_epochs': 5}, performance metrics are 0.8747
For parameters {'num_conv_filters': 128, 'conv_filter_size': 7, 'dropout_rate': 0.
4, 'num_epochs': 5}, performance metrics are 0.8707
For parameters {'num_conv_filters': 128, 'conv_filter_size': 7, 'dropout_rate': 0.
6, 'num_epochs': 5}, performance metrics are 0.8508
For parameters {'num_conv_filters': 128, 'conv_filter_size': 7, 'dropout_rate': 0.
8, 'num_epochs': 5}, performance metrics are 0.8223
For parameters {'num_conv_filters': 128, 'conv_filter_size': 9, 'dropout_rate': 0.
2, 'num_epochs': 5}, performance metrics are 0.881
For parameters {'num_conv_filters': 128, 'conv_filter_size': 9, 'dropout_rate': 0.
4, 'num_epochs': 5}, performance metrics are 0.8689
For parameters {'num_conv_filters': 128, 'conv_filter_size': 9, 'dropout_rate': 0.
6, 'num_epochs': 5}, performance metrics are 0.8482
For parameters {'num_conv_filters': 128, 'conv_filter_size': 9, 'dropout_rate': 0.
8, 'num_epochs': 5}, performance metrics are 0.7457
```

After training the program for over two hours, the best parameters are :

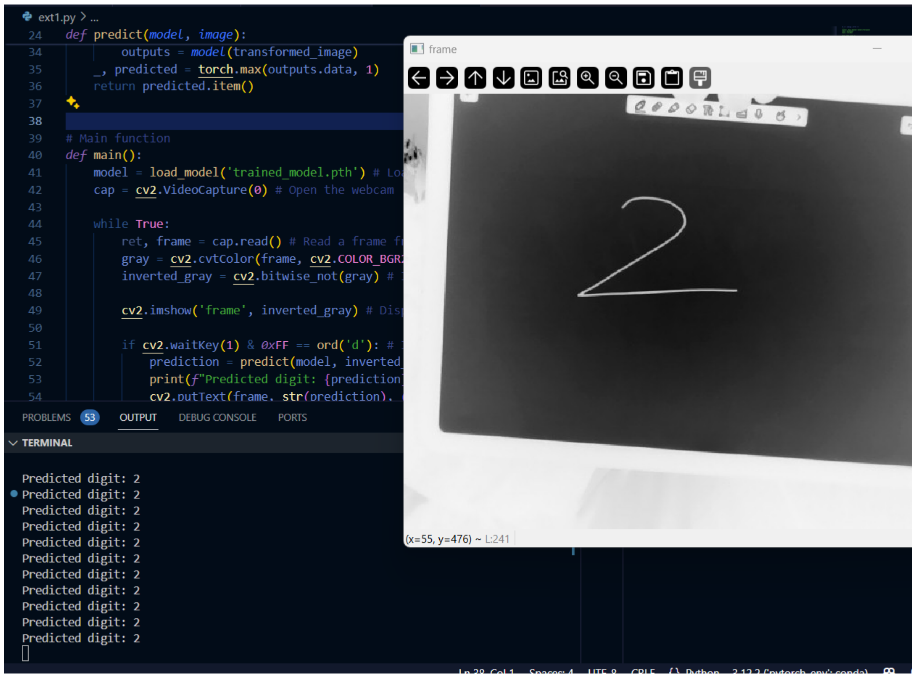
- Number of convolution filters:128.
- Convolution filter size: 9.
- Dropout rate: 0.2.

This combination has a high score of 0.881

Extensions:

Extension 1: Real-time digit recognition:

The program will read the input frame from the webcam and display it. When the user presses the ‘d’ key, if there is a digit in the camera frame, it will be detected and displayed. The camera grayscale needs to be inverted so that the input image is similar to the MNIST data. The results are attached below:



```
def predict(model, image):
        outputs = model(transformed_image)
    _, predicted = torch.max(outputs.data, 1)
    return predicted.item()


# Main function
def main():
    model = load_model('trained_model.pth') # Lo
    cap = cv2.VideoCapture(0) # Open the webcam

    while True:
        ret, frame = cap.read() # Read a frame fr
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR
        inverted_gray = cv2.bitwise_not(gray) #

        cv2.imshow('frame', inverted_gray) # Dis

        if cv2.waitKey(1) & 0xFF == ord('d'): #
            prediction = predict(model, inverted
            print(f"Predicted digit: {prediction
            cv2.putText(frame, str(prediction),
```

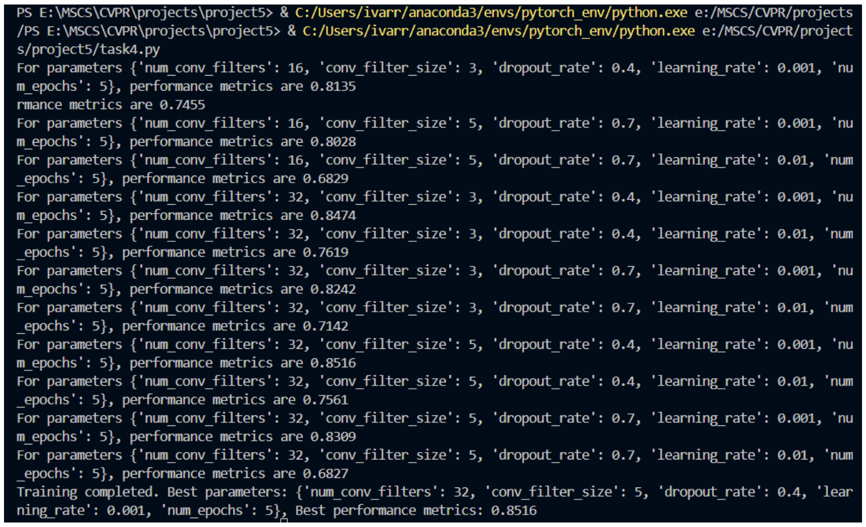
```
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
Predicted digit: 2
```

Extension 2: Additional Hyper parameters:

In task 4 , I had initially focused on only 3 hyperparameters, now I'll include learning rate and check for the optimal combination. The parameter grid is provided below:

- Number of convolution filters: 16, 32.
- Convolution filter size: 3, 5.
- Dropout rate: 0.4, 0.7.
- Learning rate: 0.01, 0.001

The best metric is provided by the program, it is attached below:



```
PS E:\MSCS\CVPR\projects\project5> & C:/Users/ivarr/anaconda3/envs/pytorch_env/python.exe e:/MSCS/CVPR/projects
/PS E:\MSCS\CVPR\projects\project5> & C:/Users/ivarr/anaconda3/envs/pytorch_env/python.exe e:/MSCS/CVPR/project
s/project5/task4.py
For parameters {'num_conv_filters': 16, 'conv_filter_size': 3, 'dropout_rate': 0.4, 'learning_rate': 0.001, 'nu
m_epochs': 5}, performance metrics are 0.8135
rmance metrics are 0.7455
For parameters {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.7, 'learning_rate': 0.001, 'nu
m_epochs': 5}, performance metrics are 0.8028
For parameters {'num_conv_filters': 16, 'conv_filter_size': 5, 'dropout_rate': 0.7, 'learning_rate': 0.01, 'num
_epochs': 5}, performance metrics are 0.6829
For parameters {'num_conv_filters': 32, 'conv_filter_size': 3, 'dropout_rate': 0.4, 'learning_rate': 0.001, 'nu
m_epochs': 5}, performance metrics are 0.8474
For parameters {'num_conv_filters': 32, 'conv_filter_size': 3, 'dropout_rate': 0.4, 'learning_rate': 0.01, 'num
_epochs': 5}, performance metrics are 0.7619
For parameters {'num_conv_filters': 32, 'conv_filter_size': 3, 'dropout_rate': 0.7, 'learning_rate': 0.001, 'nu
m_epochs': 5}, performance metrics are 0.8242
For parameters {'num_conv_filters': 32, 'conv_filter_size': 3, 'dropout_rate': 0.7, 'learning_rate': 0.01, 'num
_epochs': 5}, performance metrics are 0.7142
For parameters {'num_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.4, 'learning_rate': 0.001, 'nu
m_epochs': 5}, performance metrics are 0.8516
For parameters {'num_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.4, 'learning_rate': 0.01, 'num
_epochs': 5}, performance metrics are 0.7561
For parameters {'num_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.7, 'learning_rate': 0.001, 'nu
m_epochs': 5}, performance metrics are 0.8309
For parameters {'num_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.7, 'learning_rate': 0.01, 'num
_epochs': 5}, performance metrics are 0.6827
Training completed. Best parameters: {'num_conv_filters': 32, 'conv_filter_size': 5, 'dropout_rate': 0.4, 'lear
ning_rate': 0.001, 'num_epochs': 5}, Best performance metrics: 0.8516
```

Reflections:

This project was helpful in understanding how to build a neural network for basic deep learning projects. I learnt the need for each kind of layer and how the convolution layer applied filters over the image to learn its features. This project also taught me how a basic CNN works. And additionally, working on tuning the hyperparameters and researching more about it provided me with a deeper understanding of how each hyperparameter affects the performance of the model.

Acknowledgements:

- https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/
- https://medium.com/@sengupta.joy4u/how-to-decide-the-hyperparameters-in-cnn-bfa37b608046