CS5330 Project 5: Recognition using Deep Networks

Description:

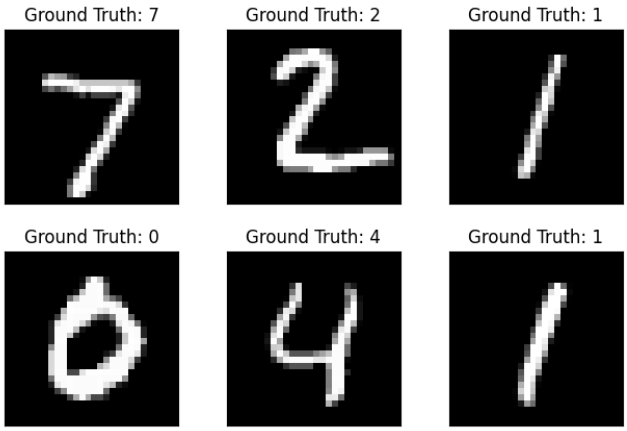
This project is the fifth project in CS5330 class. This project mainly uses the MNIST data set to train a simple convolutional neural network. After the training was completed, the working principle and effect of the convolution kernel were explored by visualizing the convolution layer and the features after the convolution kernel was applied to the image. And used the same method to explore the working principle and effect of the first two convolutional layers of the famous ResNet18 model. In addition, in this project, we also try to replace the first convolution layer with 10 defined convolution kernels and train the model on this basis. Subsequently, try to use the transfer learning method to change the already trained model into a new model that can recognize the Greek letter by changing the output hyperparameters of the last fully connected layer and retraining the parameters of the last layer. Finally, based on this model, multiple dimensions of hyperparameter selection were provided, and a total of 71 models were trained to find the best hyperparameter combination.

Task1: Build and train a network to recognize digits
Task2: Examine your network
Task3: Transfer Learning on Greek Letters
Task4: Design your own experiment
Extension1: Evaluate more dimensions on task 4
Extension2: Loading a pre-trained networks available in PyTorch package and evaluate its first couple of convolutional layers
Extension3: Replace the first layer of the MNIST network with a filter bank of your choosing and retrain the rest of the network.

1. **Task 1: Build and train a network to recognize digits**

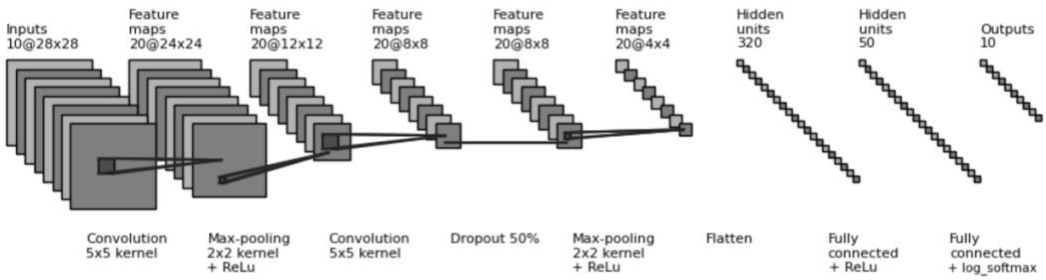
   A. Get the MNIST digit data set

   Use the torchvision.datasets.MNIST function in the torchvision package to download the MNIST dataset and use the transform function to convert them into tensor form for storage. Use the torch.utils.data.DataLoader function in the torch package to load data. Draw the first six data in the test set.



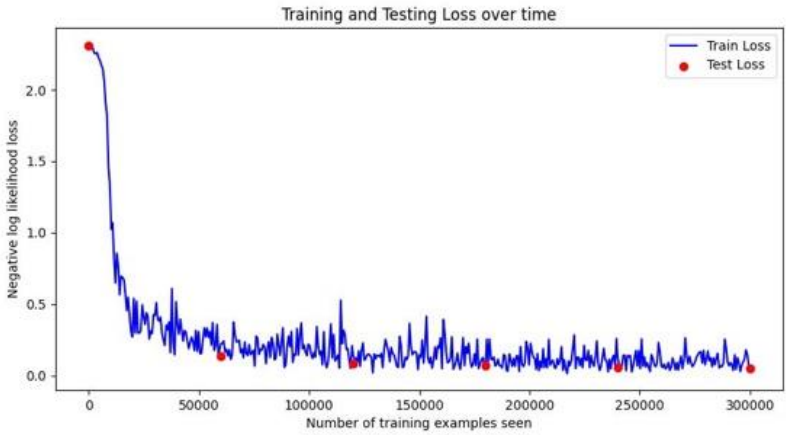
   B. Build a network model

   A neural network model was created as described in the requirements. It includes two convolutional layers, two pooling layers, a drop out layer and two fully connected layers. The input data is a 28x28 single-channel image. The output layer includes ten nodes, and the index with the largest value is the prediction result.



C. Train the model

The above model has undergone a total of five epochs of training. During the training process, the loss of the model on the training set and test set was recorded, and the data was visualized after the training.



After each epoch, the model was evaluated on the training set and test.

```
(deeplearning) yuqipeng@YuqideMacBook-Pro src % python BasicNet.py r t
torch.Size([1000, 1, 28, 28])
Epoch 0/5: Avg. Loss: 2.3119, Test Accuracy: 8.92%.
Epoch 1/5: 100%|                                   | 938/938 [00:21<00:00, 43.96batch/s]
Epoch 1/5: Avg. Loss: 0.6621, Train Accuracy: 95.53%.
Epoch 1/5: Avg. Loss: 0.1345, Test Accuracy: 95.80%.
Epoch 2/5: 100%|                                   | 938/938 [00:21<00:00, 42.99batch/s]
Epoch 2/5: Avg. Loss: 0.1874, Train Accuracy: 97.15%.
Epoch 2/5: Avg. Loss: 0.0855, Test Accuracy: 97.37%.
Epoch 3/5: 100%|                                   | 938/938 [00:20<00:00, 45.10batch/s]
Epoch 3/5: Avg. Loss: 0.1436, Train Accuracy: 97.76%.
Epoch 3/5: Avg. Loss: 0.0687, Test Accuracy: 97.82%.
Epoch 4/5: 100%|                                   | 938/938 [00:23<00:00, 40.67batch/s]
Epoch 4/5: Avg. Loss: 0.1209, Train Accuracy: 98.13%.
Epoch 4/5: Avg. Loss: 0.0610, Test Accuracy: 98.01%.
Epoch 5/5: 100%|                                   | 938/938 [00:21<00:00, 44.44batch/s]
Epoch 5/5: Avg. Loss: 0.1090, Train Accuracy: 98.21%.
Epoch 5/5: Avg. Loss: 0.0532, Test Accuracy: 98.28%.
```

D. Save the network to a file

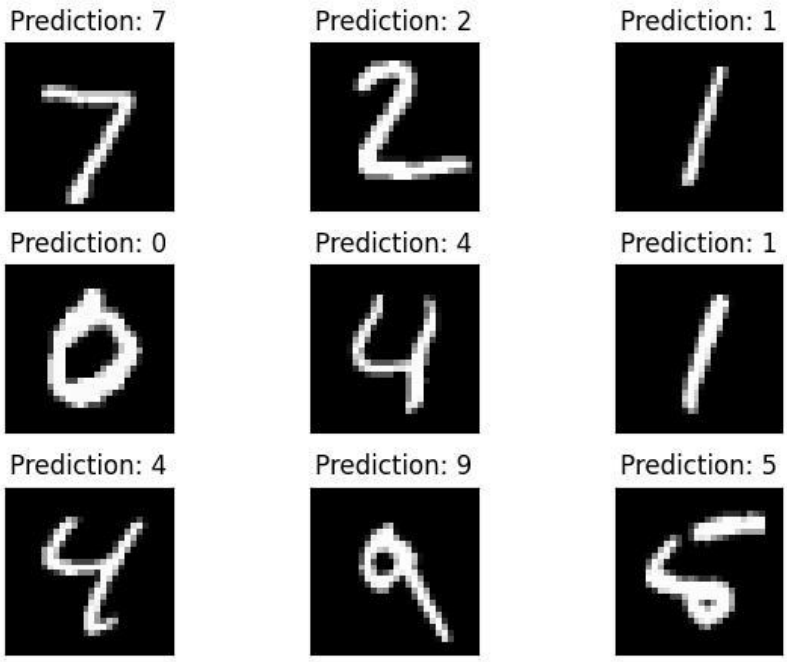
Save the trained model parameters in a file so that when using the model in the future, the parameter data can be directly read.

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

Read the parameters from the file that saves the model parameter data and use this model to predict the first ten pictures in the test set. Their prediction results are as follows:

```
----Result for the first 10 examples in the test set----
Output:
tensor([[-1.7990e+01, -1.6870e+01, -1.2340e+01, -1.0340e+01, -2.4630e+01,
         -2.0080e+01, -3.3580e+01, -0.0000e+00, -1.6780e+01, -1.5560e+01],
        [-6.7300e+00, -9.7800e+00, -0.0000e+00, -1.6730e+01, -1.6030e+01,
         -1.3580e+01, -1.1390e+01, -2.0560e+01, -1.1630e+01, -2.2200e+01],
        [-1.3740e+01, -0.0000e+00, -1.0870e+01, -1.2960e+01, -1.0490e+01,
         -1.1520e+01, -1.2750e+01, -9.5100e+00, -9.4500e+00, -1.3790e+01],
        [-0.0000e+00, -2.9340e+01, -1.6850e+01, -2.1150e+01, -2.2090e+01,
         -1.5060e+01, -1.0940e+01, -1.8540e+01, -1.8370e+01, -1.3860e+01],
        [-1.9970e+01, -2.5880e+01, -1.9210e+01, -2.1230e+01, -0.0000e+00,
         -2.2580e+01, -1.9630e+01, -1.5590e+01, -1.9040e+01, -1.0240e+01],
        [-1.6620e+01, -0.0000e+00, -1.3590e+01, -1.4840e+01, -1.1720e+01,
         -1.4370e+01, -1.6430e+01, -1.0340e+01, -1.0840e+01, -1.5070e+01],
        [-2.4980e+01, -1.2720e+01, -1.6470e+01, -1.5230e+01, -0.0000e+00,
         -1.2840e+01, -2.0270e+01, -7.9200e+00, -7.0200e+00, -7.1700e+00],
        [-1.6900e+01, -1.7060e+01, -1.1810e+01, -8.3000e+00, -5.0700e+00,
         -8.7400e+00, -1.9170e+01, -9.0100e+00, -7.9400e+00, -1.0000e-02],
        [-1.1520e+01, -1.9120e+01, -1.4620e+01, -1.6860e+01, -1.1880e+01,
         -2.0000e-02, -4.0600e+00, -1.7230e+01, -9.2800e+00, -9.4700e+00],
        [-1.8080e+01, -2.0270e+01, -1.8950e+01, -1.2440e+01, -8.2900e+00,
         -1.5880e+01, -2.5890e+01, -5.6400e+00, -6.6300e+00, -1.0000e-02]])
Prediction:
tensor([[7, 2, 1, 0, 4, 1, 4, 9, 5, 9]])
Label:
tensor([[7, 2, 1, 0, 4, 1, 4, 9, 5, 9]])
```

Display the first 9 images in the test set and their prediction results:

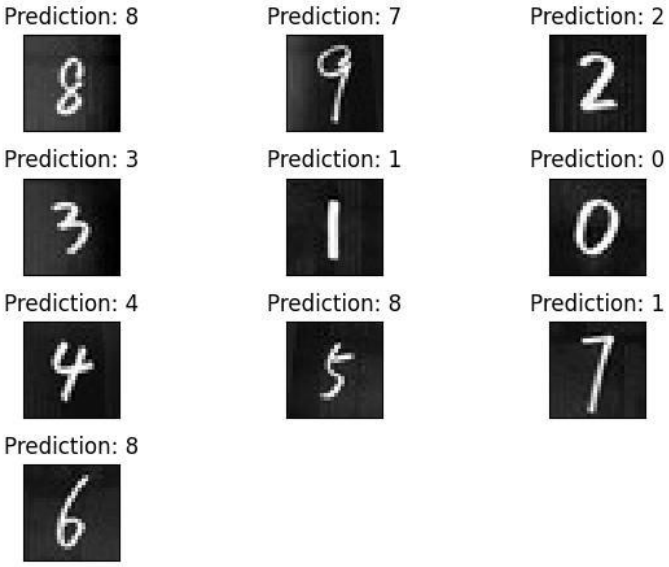


F. Test the network on new input

To better test the model's ability to recognize numbers, we applied the model to new data. We hand-written the numbers 0-9, cropped them into 28x28 grayscale images, and inverted the colors so they were white on a black background. The significance of this is to ensure that the handwritten digit images have the same size and pattern as

the digit images in the training set. The model was then used to recognize handwritten digits, and the results are as follows:

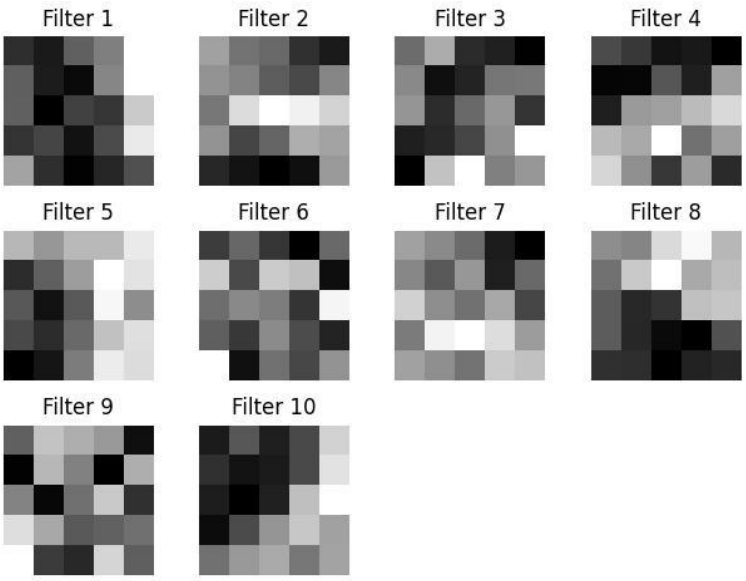
```
----Result for the my own handwriting examples----
Output:
tensor([[-2.5100, -2.0700, -2.4000, -2.6900, -2.5900, -2.3300, -2.6200, -2.8300,
         -1.4200, -2.4400],
        [-3.4300, -1.7800, -2.6900, -2.3700, -2.0400, -3.3600, -4.6900, -1.3900,
         -2.3400, -2.1200],
        [-2.5400, -1.5400, -1.1500, -2.2500, -3.8400, -3.4300, -3.6700, -2.7100,
         -2.2300, -3.4400],
        [-3.0300, -2.1700, -2.5800, -1.6300, -2.4100, -2.4100, -3.6300, -1.9100,
         -2.4500, -2.0700],
        [-3.0000, -1.1600, -2.6700, -3.2800, -2.4400, -2.3500, -2.7000, -2.9600,
         -1.7900, -2.8200],
        [-0.9800, -3.8000, -3.1100, -3.8800, -2.3400, -2.8300, -2.4900, -2.2900,
         -2.4800, -2.1700],
        [-4.4200, -2.6700, -3.8500, -4.2200, -0.8400, -3.5200, -4.1500, -2.6900,
         -1.6100, -2.0000],
        [-2.5000, -2.2200, -2.9200, -2.0800, -2.7900, -1.8900, -2.5400, -2.6300,
         -1.8500, -2.2100],
        [-2.8000, -1.5900, -2.4300, -2.5500, -2.4200, -2.7500, -3.4700, -2.0800,
         -1.9200, -2.1700],
        [-2.6000, -2.3100, -2.9300, -3.0400, -2.0900, -2.1400, -1.7300, -3.4700,
         -1.5600, -2.7300]])
Prediction:
tensor([[8, 7, 2, 3, 1, 0, 4, 8, 1, 8]])
Label:
tensor([[8, 9, 2, 3, 1, 0, 4, 5, 7, 6]])
```



2. Task 2: Examine your network

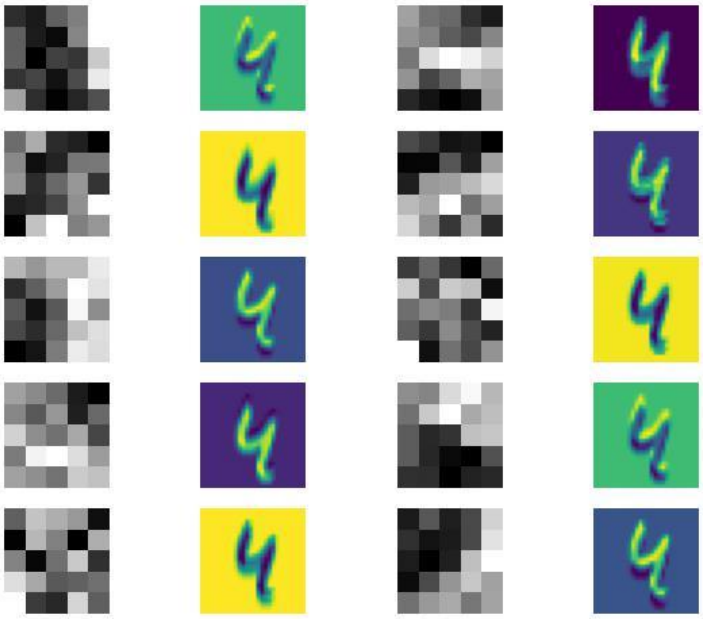
A. Analysis the first layer

To study the principle and effect of the first convolutional layer, we load the parameter data of the first convolutional layer and plot them using grayscale images.



B. Show the effect of the filters

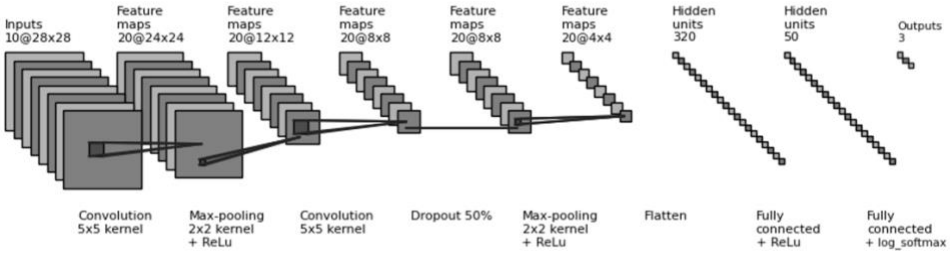
Subsequently, using the filter2D function in OpenCV, the first convolutional layer is applied to the first image in the training set, and the result is visualized to obtain the following result:



3. Task 3: Transfer Learning on Greek Letters

Use the transfer learning method, using the model designed for MNIST data and the parameter data obtained from training. Freezes all layer parameters except the last

convolutional layer so that they will not be updated during subsequent training. Then replace the last fully connected layer with a new fully connected layer with an output number of 3 and train the parameters of the last layer on the new Greek Letters training set. The new model is as follows:



The losses during training are as follows:

```
(deeplearning) yuqipeng@YuqideMacBook-Pro src % python transferLearning.py
Epoch 1/10: 100%|| 6/6 [00:00<00:00, 52.53batch/s]
Epoch 1/10: Avg. Loss: 1.0888, Train Accuracy: 66.67%.
Epoch 2/10: 100%|| 6/6 [00:00<00:00, 86.41batch/s]
Epoch 2/10: Avg. Loss: 0.7911, Train Accuracy: 88.89%.
Epoch 3/10: 100%|| 6/6 [00:00<00:00, 91.65batch/s]
Epoch 3/10: Avg. Loss: 0.6875, Train Accuracy: 96.30%.
Epoch 4/10: 100%|| 6/6 [00:00<00:00, 77.88batch/s]
Epoch 4/10: Avg. Loss: 0.5679, Train Accuracy: 96.30%.
Epoch 5/10: 100%|| 6/6 [00:00<00:00, 91.06batch/s]
Epoch 5/10: Avg. Loss: 0.5442, Train Accuracy: 96.30%.
Epoch 6/10: 100%|| 6/6 [00:00<00:00, 83.46batch/s]
Epoch 6/10: Avg. Loss: 0.4786, Train Accuracy: 100.00%.
Epoch 7/10: 100%|| 6/6 [00:00<00:00, 84.16batch/s]
Epoch 7/10: Avg. Loss: 0.5142, Train Accuracy: 100.00%.
Epoch 8/10: 100%|| 6/6 [00:00<00:00, 88.16batch/s]
Epoch 8/10: Avg. Loss: 0.3528, Train Accuracy: 100.00%.
Epoch 9/10: 100%|| 6/6 [00:00<00:00, 84.52batch/s]
Epoch 9/10: Avg. Loss: 0.3450, Train Accuracy: 100.00%.
Epoch 10/10: 100%|| 6/6 [00:00<00:00, 84.34batch/s]
Epoch 10/10: Avg. Loss: 0.3874, Train Accuracy: 100.00%.
```

Use the changed model to test the new handwritten Greek Letters test set and get the results:

```
----Result for the my own handwriting examples----
Output:
tensor([[-1.0200, -1.1800, -1.1000],
        [-1.3000, -0.7400, -1.3800],
        [-1.1700, -0.9800, -1.1700],
        [-0.9800, -0.9800, -1.3800],
        [-1.0900, -0.9700, -1.2700],
        [-1.1400, -1.0800, -1.0800],
        [-1.0700, -1.3200, -0.9400],
        [-1.1600, -1.2900, -0.8800],
        [-1.0900, -1.1800, -1.0400],
        [-1.2200, -0.9600, -1.1300],
        [-1.1900, -1.3900, -0.8100],
        [-1.1000, -1.2500, -0.9700],
        [-1.2300, -1.0200, -1.0600],
        [-1.1900, -1.2500, -0.9000],
        [-1.1800, -1.4700, -0.7700]], grad_fn=<RoundBackward1>)
Prediction:
tensor([[0, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2]])
Label:
tensor([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2]])
```

4. Task 4 & Extension 1: Design your own experiment

A. Develop a plan

To explore the best hyperparameter selection, we conducted experiments on the values of multiple hyperparameters, including the batch size of each training, the number of convolution kernels in the first convolution layer, and the number of convolution kernels in the second convolution layer. the size of the convolution kernels, the dropout rate of the dropout layer, the number of nodes in the first fully connected layer, and the total number of training epochs.

There are 5 choices for batch size, which are 16, 32, 64, 128, and 256.

There are 5 choices for the number of convolution kernels in the first convolution layer, which are 4, 8, 16, 32, and 64.

There are 5 choices for the number of convolution kernels in the second convolution layer, which are 8, 16, 32, 64, and 128.

There are 3 choices for the convolution kernel size, respectively 3, 5, and 7.

The dropout rate has 26 options from 0.20 to 0.70.

The number of nodes in the first fully connected layer has 11 options from 50 to 200.

There are 11 options for the total number of training epochs ranging from 2 to 12.

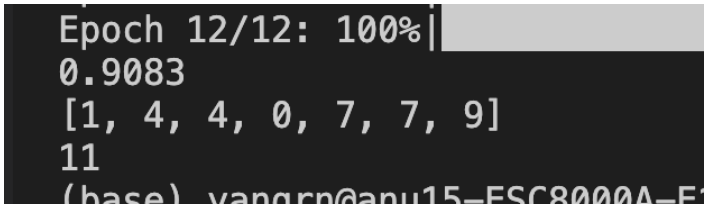
This experiment verified a total of 7 dimensions of hyperparameter selection, and a total of 71 models were trained.

B. Predict the results

Regarding the results, we believe that a smaller training batch size can give the model better training results, but at the same time it will lose a certain degree of robustness. Therefore, we believe that a batch size of 32 will have better training results. More complex models will have better training results, but they will also lose a certain degree of robustness. So, the guesses for the number of convolution kernels in the two convolutional layers are 32 and 64. Since the size of the image is small, smaller convolution kernels can extract more detailed information. Therefore, we guess that the model with a convolution kernel size of 3 is better. For the original model, the dropout rate of 0.5 is a bit too high. Here we guess 0.3 is a better choice. For a fully connected layer, more nodes mean a more complex model, but it will cause overfitting problems. So, we guess 150 is a good choice for the number of nodes. More training rounds can help the model find the optimal solution better, so for the number of training rounds, we guess 12 is the best choice.

C. Execute your plan

After experiments, the results are as follows: (The number is the index of the hyperparameter)

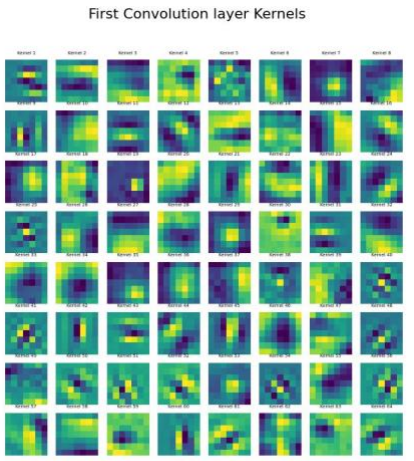


Hyperparameter	Best option
train_batch_size	32
conv1_out_channels	64
conv2_out_channels	128
kernel_size	3
dropout_rate	0.34
fc1_out_features	120
epoch	11

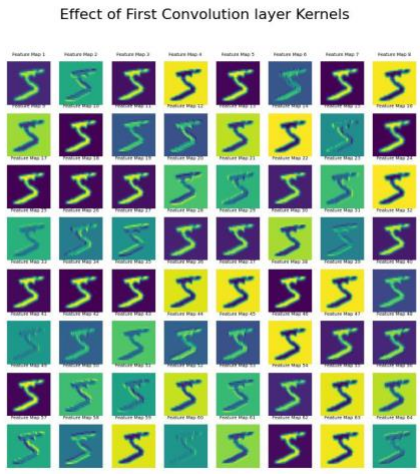
5. Extension 2: Examine ResNet18

First, get the resnet18 network and its weights ResNet18_Weights from the torchvision.models package. Since the convolution kernel in this network is a three-channel convolution kernel, we also convert the images into three-channel images and adjust the size to 224x224 so that they have the same format as the images processed by the resnet18 network. The first two convolutional layers in this network were then visualized in the same way as task2.

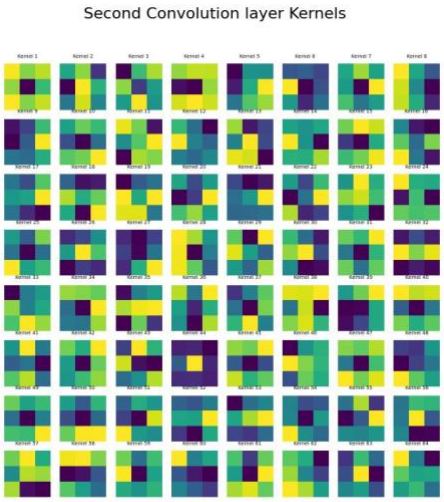
First layer kernels:



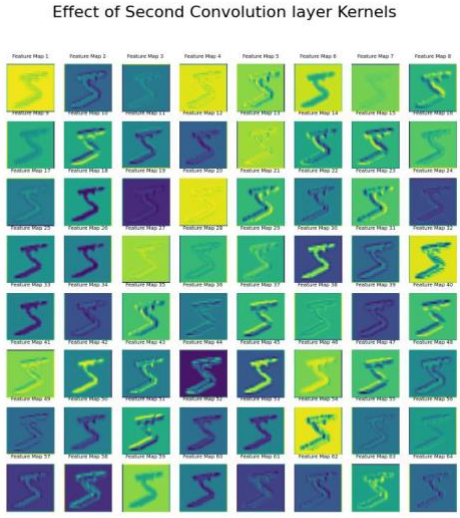
First layer kernel effect:



Second layer kernels:

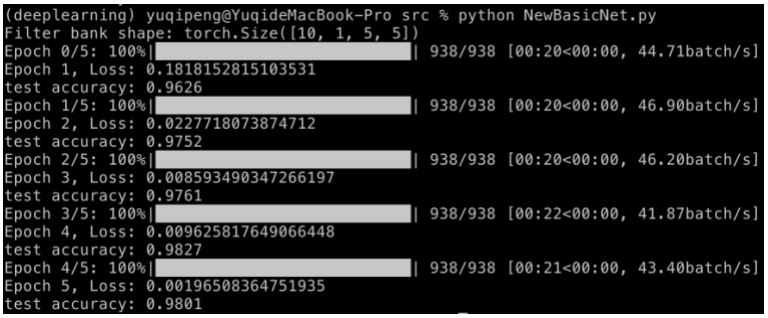


Second layer effect:



6. **Extension 3: Replace with Filter Bank**

   Change the model trained on the MNIST data and replace the first convolution layer with a self-defined filter bank, which includes 8 5x5 Gabor filters and 2 5x5 Sobel filters. Freeze this layer and retrain the remaining layers to get a new model.



Reflection:

In this project, we learned how to build a convolutional neural network under the PyTorch framework, and how to train and test the network. And conducted a more in-depth study of the convolutional layers, including how they act on image extraction features, whether they can be replaced by well-defined classic convolution kernels, etc. In addition, we also learned the concept of transfer learning, how to make small modifications to an already trained network so that it can be used for similar but completely new tasks. In addition, in this project, we also studied parameter selection and learned how to adjust hyperparameters to make the model have better training effects.

Acknowledgement:

Thanks for TAs for help debug my coding. Thanks a lot.

Travel Days:

1 Day