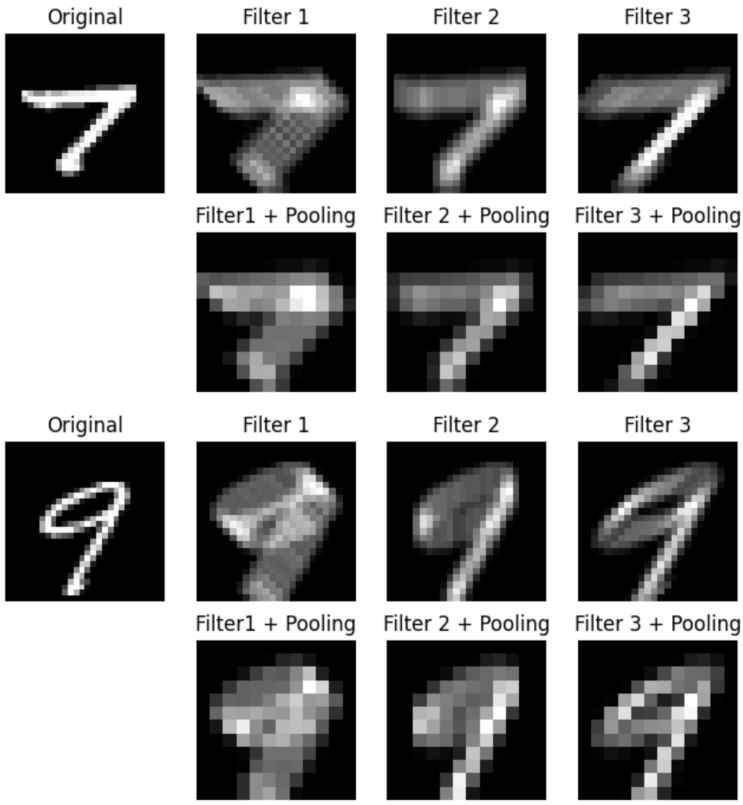
CS470 Assignment 2

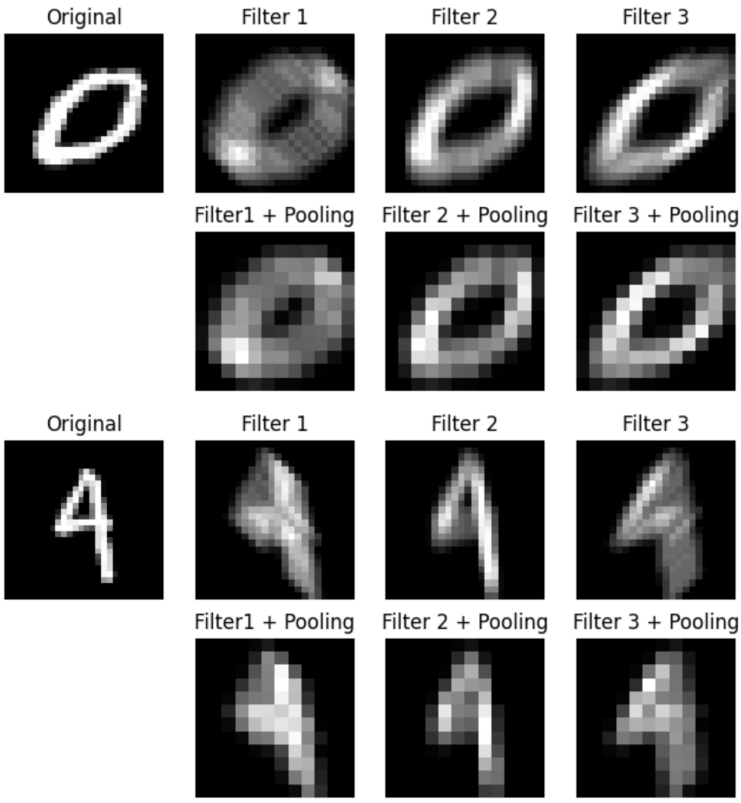
20180185 김해찬

1. Shallow Convolution Neural Networks [20pts]

1.1 Convolution and Average Pooling using NumPy [10pts]

- attach the visualization results and





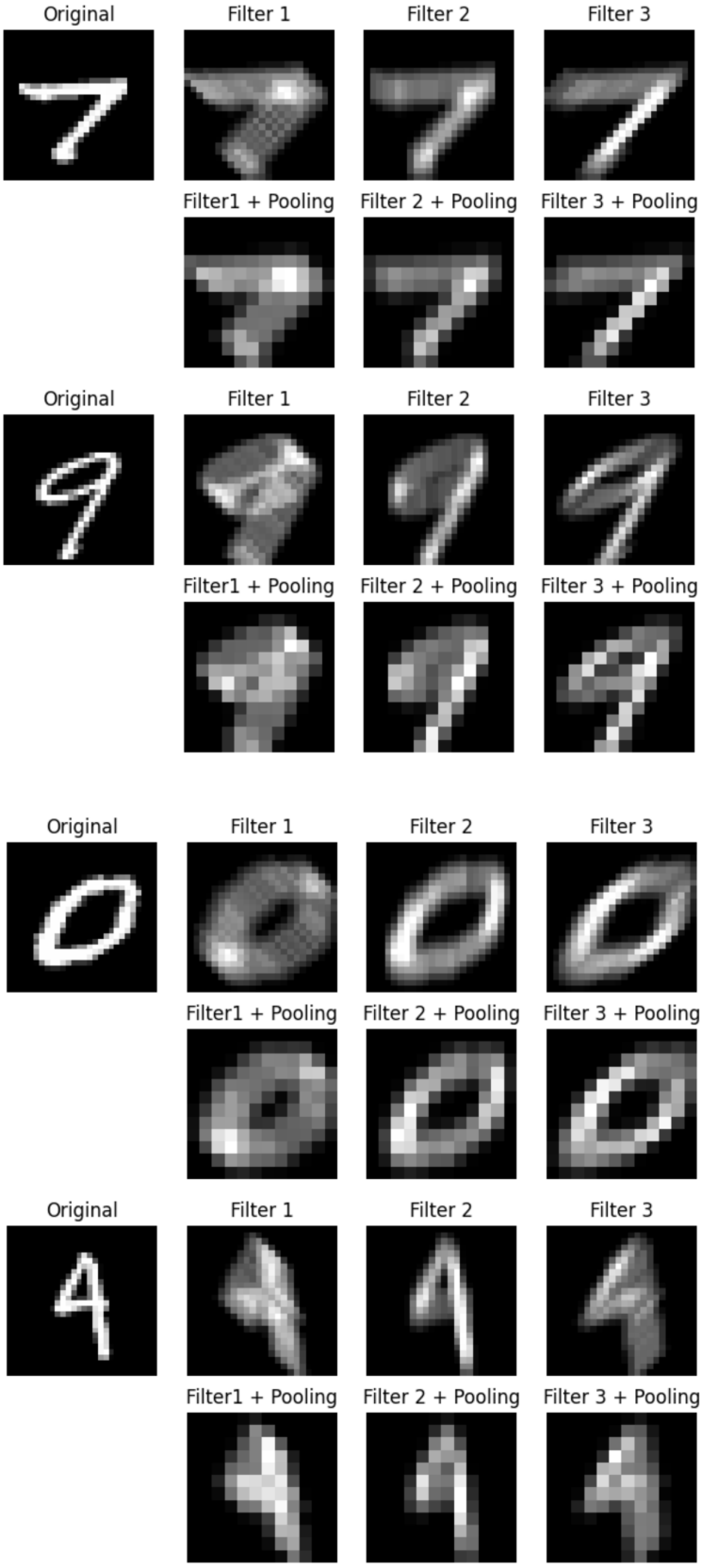
- write down your analysis stating the difference between results (Max. 600 characters).

  As same reason, results of Filter 2 (takes other values on vertical line) and Filter 3(takes other values on other diagonal) look spread in vertical and reverse diagonal direction respectively.

  In case of Average Pooling, it makes 4 parameters to 1 parameter by averaging them. That means the image information is compressed and also lost. That is reason why the results look more blurry.

1.2 Convolution and Average Pooling using PyTorch [10pts]

- attach the visualization results and



- provide whether your implementation using NumPy is the same as that using PyTorch by cal- culating any errors.

I define the error between two kinds of methods as euclidean distance between two results. The euclidean distance between numpy output and pytorch output by each example and each filter can be shown below.

```
euclidean distance of out1(Filter only)
========================================
example 0 || filter 0 : 1.0431478044892447e-06
example 0 || filter 1 : 1.2549584651970851e-06
example 0 || filter 2 : 1.074292599654985e-06
example 1 || filter 0 : 8.593887239881657e-07
example 1 || filter 1 : 9.30685797942879e-07
example 1 || filter 2 : 1.1044657250819606e-06
example 2 || filter 0 : 1.335878271048302e-06
example 2 || filter 1 : 1.4917235300867373e-06
example 2 || filter 2 : 1.6027125288846927e-06
example 3 || filter 0 : 9.51491304342438e-07
example 3 || filter 1 : 1.1427358351455717e-06
example 3 || filter 2 : 9.071124904903077e-07
```

```
euclidean distance of out2(Filter + Pooling
========================================
example 0 || filter 0 : 6.562287805738392e-07
example 0 || filter 1 : 7.463225667038346e-07
example 0 || filter 2 : 6.041041307070473e-07
example 1 || filter 0 : 4.826125465800486e-07
example 1 || filter 1 : 5.232699205414944e-07
example 1 || filter 2 : 6.352485760161269e-07
example 2 || filter 0 : 8.413884506004705e-07
example 2 || filter 1 : 9.106394020026595e-07
example 2 || filter 2 : 1.0196910211056406e-06
example 3 || filter 0 : 6.256028550934652e-07
example 3 || filter 1 : 5.817338280625892e-07
example 3 || filter 2 : 6.66254437156603e-07
```

we can know that both are almost same.

2. Convolutional Neural Networks (CNN) [60pts]

2.1 A CNN with MaxPooling layers [20pts]

- analyze the number of parameters used in each layer and the total number of parameters over the entire model considering the input image size,

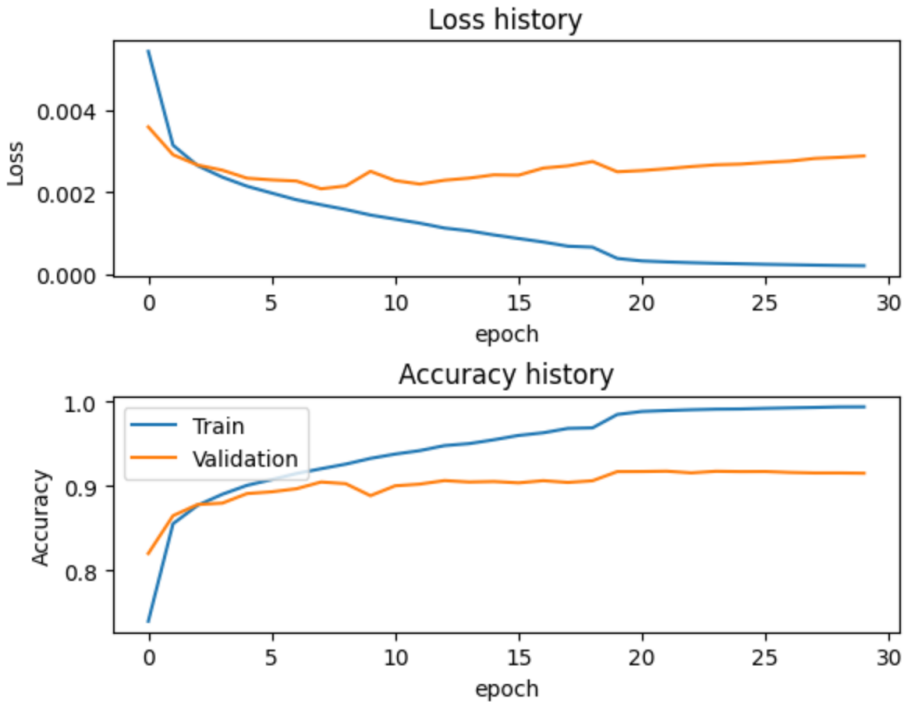
  Let the input image size as (1, 28, 28).

  - (Layer 1) Conv2d-1 (32, 3, 3) → 32 * 3 * 3 + 32 = 320 parameters
  - (Layer 3) Conv2d-2 (64, 3, 3) → 64 * 32 * 3 * 3 + 64 = 18,496 parameters

- (Layer 6) Linear-1 (64, 30976) → 64 * 30976 + 64 = 1,982,528 parameters
- (Layer 7) Linear-2 (32, 64) → 32 * 64 + 32 = 2,080 parameters
- (Layer 8) Linear-3 (10, 32) → 10 * 32 + 10 = 330 parameters

Number of total parameters is 2,003,754.

- attach the graph of training and validation accuracies over 30 epochs,



- attach the capture of the test accuracy on the 10, 000 test images from the ipynb screen.

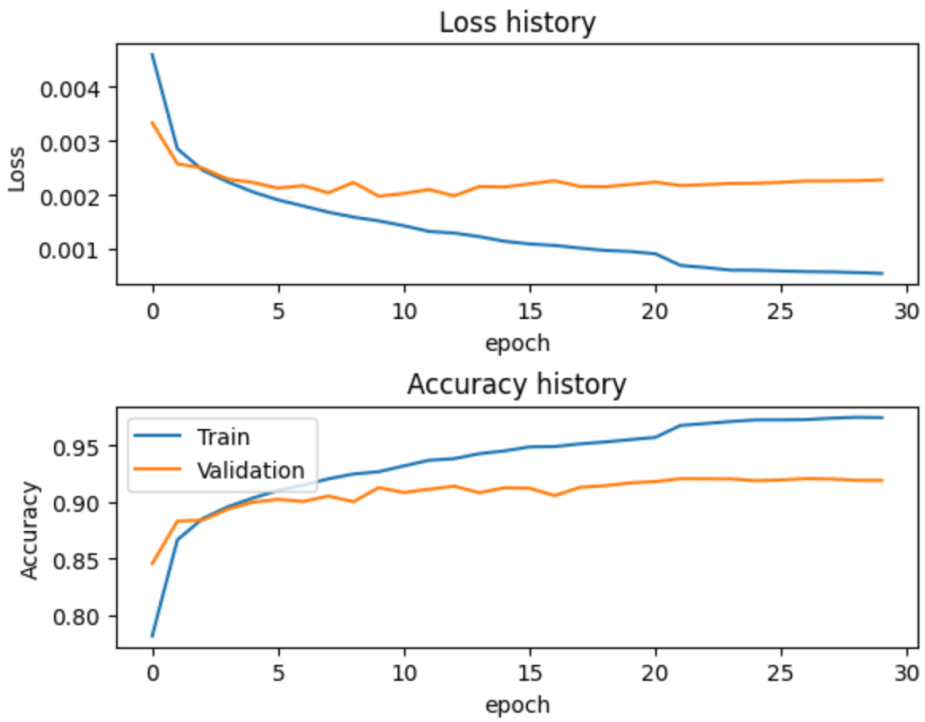
```
[8] pred_vec = []
    correct = 0
    net.eval()
    with torch.no_grad():
        for data in test_loader:
            batch, labels = data
            batch, labels = batch.to(device), labels.to(device)
            outputs = net(batch)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            pred_vec.append(predicted)
        pred_vec = torch.cat(pred_vec)

    print('Accuracy on the 10000 test images: %.2f %%' % (100 * correct / len(test_set)))
```

```
Accuracy on the 10000 test images: 91.59 %
```

2.2 Prevention of Overfitting [40pts]

- attach a graph of training-and-validation accuracies over 30 epochs with a selected technique that handles overfitting,



- report the test accuracy on the 10, 000 test images,

```
[8]  pred_vec = []
     correct = 0
     net.eval()
     with torch.no_grad():
         for data in test_loader:
             batch, labels = data
             batch, labels = batch.to(device), labels.to(device)
             outputs = net(batch)
             _, predicted = torch.max(outputs, 1)
             correct += (predicted == labels).sum().item()
             pred_vec.append(predicted)
         pred_vec = torch.cat(pred_vec)

     print('Accuracy on the 10000 test images: %.2f %%' % (100 * correct / len(test_set)))

     Accuracy on the 10000 test images: 91.91 %
```

- compare the results of the CNN model without regularization methods (analyze and explain why the selected technique works for preventing overfitting).

  I used two methods(weight decay and dropout) to prevent overfitting. As we can see from training-validation accuracies and loss graph, the gap of accuracies and loss between training and validation with two methods is less than the gap without them. Also, the accuracy on the test images is quite increased.

  The below explains why each two methods can prevent overfitting.

  - Weight decay(1e-5)

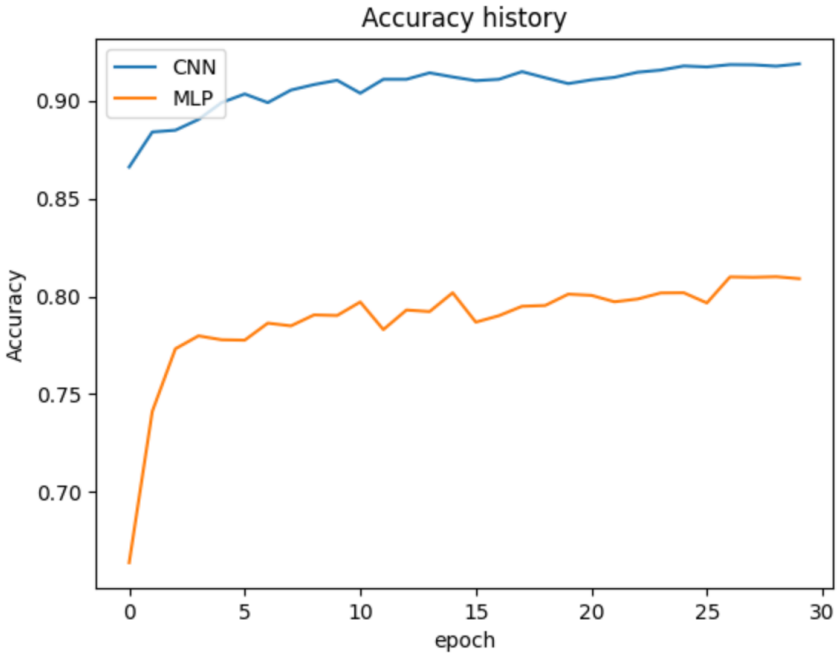
    By using Weight decay to set the lower bound of loss value, parameters can be prevented from being excessively updated (i.e., overfitting).

  - Drop out(p = 0.7)

    Dropout prevents the corresponding parameter from being updated at that point by making certain parameters zero with the probability of p. That is, the number of times parameters are updated is reduced in probability, preventing overfitting.

3. Comparison of MLP and CNN [20pts]

- attach a plot of validation accuracy curves from the two models where the x axis and y axis are the number of training epochs (which is 30) and accuracy, respectively,



- analyze the results (e.g., why does one model perform better than the other?) within one
  paragraph.

  For reasonable comparison, I tried to make numbers of total trainable parameters of both model same. The CNN model has 2,003,754 parameters and the MLP model has 2,005,610 parameters. Although the number of parameters are similar, the accuracy on test images of the CNN and MLP are 91.48% and 80.54%, respectively. I think the reason why the accuracy gap exists is affect of locality. The CNN model can consider locality information of input images. In the other hands, the MLP model can't refer their locality information because they make input images flatten immediately when they come in the model. That is reason why CNN is powerful than MLP in image task.