CS 454

Introduction to Machine Learning and Artificial Neural Networks

Prof. Dr. Ethem Alpaydın

Fall 2021

Homework #4

Neural Networks (PyTorch)

by

Abdullah Saydemir

S014646

December 22, 2021

Since I do not own the code, I will push it to the branch and later delete it after grades are announced. Code and log files are available via this link after the deadline.

Part 1 - Defining Networks

I defined three networks as follows. Variables in the convolutional layers will be changed.

```
class Network1(nn.Module):
    def __init__(self):
        super(Network1, self).__init__()
        self.conv1 = nn.Conv2d(
                                in_channels = 1,
                                out_channels= 16,
                                kernel_size = 3,
                                stride = 2,
                                padding = 1
                            )
       # [( 10 - 3 + 2*1 ) / 2] + 1 = 5
        self.conv2 = nn.Conv2d(
                            in_channels = 16,
                            out_channels= 32,
                            kernel_size = 3,
                            stride = 2,
                            padding = 1
                        )
        # [( 5 - 3 + 2 * 1 ) / 2] + 1 = 3
        self.fcl     = nn.Linear(32*3*3, 10)
        self.model = nn.Sequential(
            self.conv1,
            self.conv2,
            self.fcl
        )

    def forward(self,x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = torch.flatten(x, 1)
        return torch.log_softmax(x, dim=1)
```

Network1 has two fully connected layers followed by a fully connected layer. Activation function is ReLU instead of sigmoid.

```
class Network2(nn.Module):
    def __init__(self):
        super(Network2, self).__init__()
        # [( W - kernel_size + 2*padding ) / stride] + 1
        self.conv1 = nn.Conv2d(
                                in_channels = 1,
                                out_channels= 16,
                                kernel_size = 3,
                                stride = 2,
                                padding = 1
                            )
       # [( 10 - 3 + 2*1 ) / 2] + 1 = 5
        self.fcl    = nn.Linear(16*5*5, 10)
        self.model = nn.Sequential(
            self.conv1,
            self.fcl
        )

    def forward(self,x):
        x = torch.relu(self.conv1(x))
        x = torch.flatten(x, 1)
        return torch.log_softmax(x, dim=1)
```

Network2 has 1 convolutional layer and 1 fully connected layer. Activation function is also ReLU.

```
class Network3(nn.Module):
    # get input and output dimensions as input
    def __init__(self):
        # all derived classes must call __init__ method of super class
        super(Network3, self).__init__()

        self.model    = nn.Linear(100, 10)

    # forward method should get the input and return the output
    def forward(self,x):
        batch_size = x.shape[0]
        # flatten the image from BxCxHXW to Bx100
        x = x.view(batch_size, -1)
        x = self.model(x)
        # softmax is internally done inside cross entropy loss
        return torch.log_softmax(x, dim=1)
```

Network3 is simply a single layer perceptron.

Before doing anything, I expect the Network3 (SLP) or Network2 to provide the best results. Because the dataset size is small and there are fewer pixels in each image. I also use ReLU as the activation function which is not advised in smaller networks. So, using a complex network, I think, will not enhance learning.

Furthermore, by using these architectures, Network1 and Network2, I can also compare the results and see the advantages / disadvantages of using a second layer.

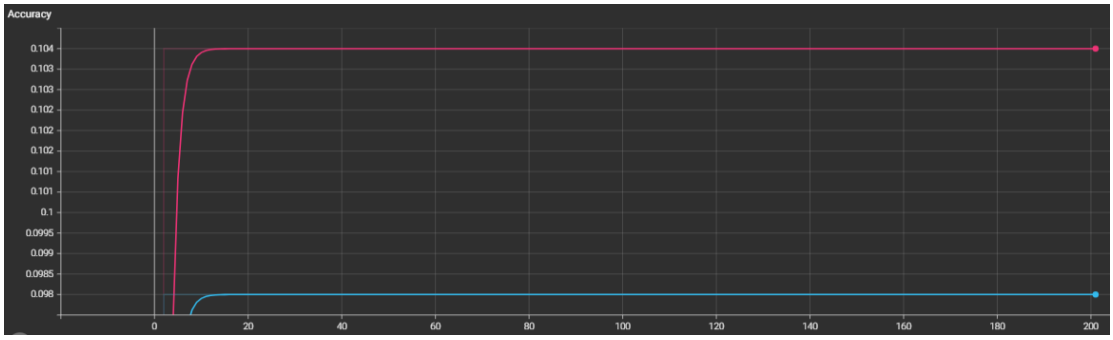
Part 2 - Trying Different Parameters & Plotting Accuracy

There needs to be 3 graphs for each network. I would like to experiment more to compare each variable's effect to the accuracy. I will provide 2 graphs for each section (for Network1 and Network2) and compare their best results with SLP (Network3). I will try two extremes and one middle point for variables. SLP will be used as the basis, since I expect the best results from it. I could not find how to show labels of the curves in tensorboard, but the one with the higher accuracy shows the training accuracy. Graphs have 0.6 smoothing by default.

a) Kernel Size

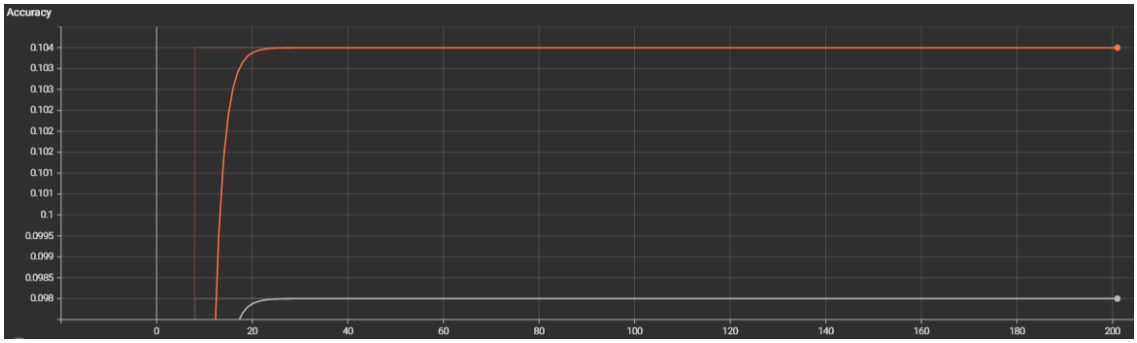
kernel_size = 1 (no padding, stride = 1) Expectation : No learning since a pixel is not a good representation of the whole image.

Network1 :



As expected no learning…

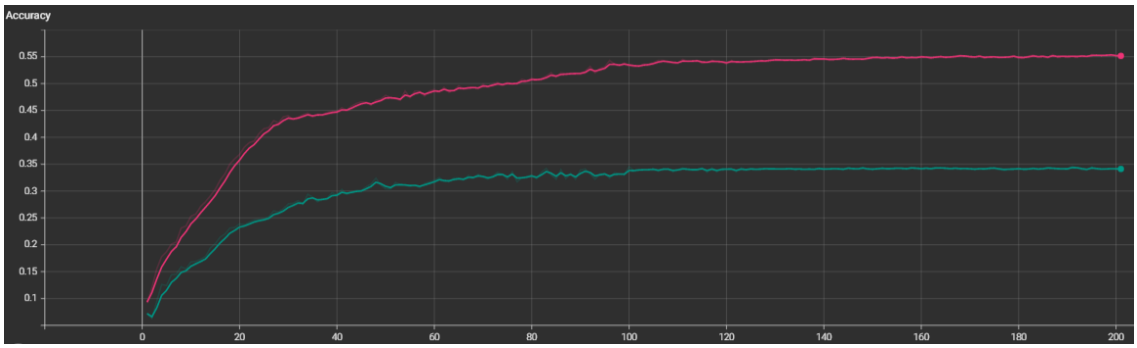
Network2 :



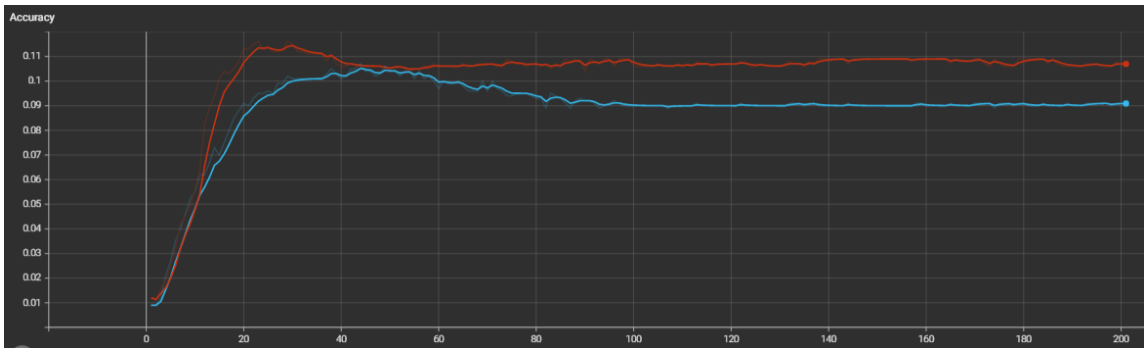
Again, no learning…

kernel_size = 4 (no padding, stride = 1) Expectation : Learning will happen, but I don't know how good it will be. I think accuracy comparison will be Network2 > Network1

Network1 :



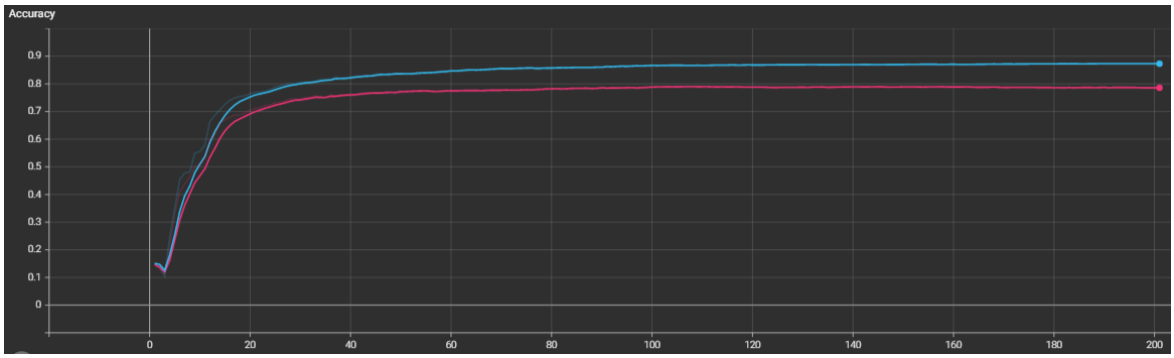
Network2 :



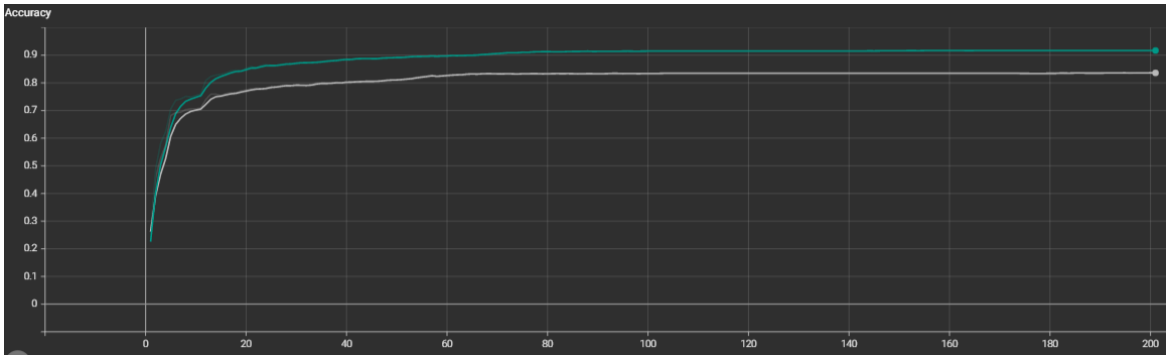
I was wrong. One layer was not good enough to represent the images. I think the resulting kernel size (7x7) was not really dense enough.

kernel_size = 10 (1 for second layer) (no padding, stride = 1) Expectation : Learning will happen, but I don't know how good it will be. I think accuracy comparison will be Network2 = Network1, because I consume everything in the first layer.

Network1 :



Network2 :

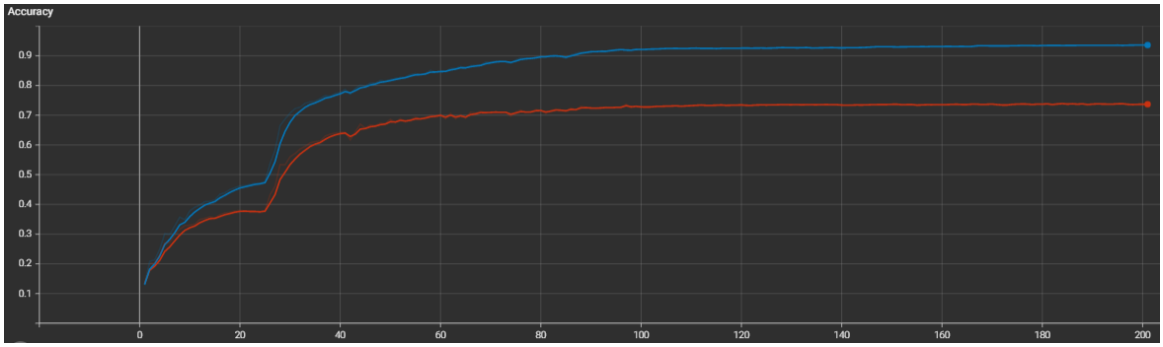


I was kind of close this time. One layer worked pretty decent. Seems like creating 32 channels from 16 channels in the second layer slightly decreases the accuracy. Consuming all of the pixels is not a bad idea.

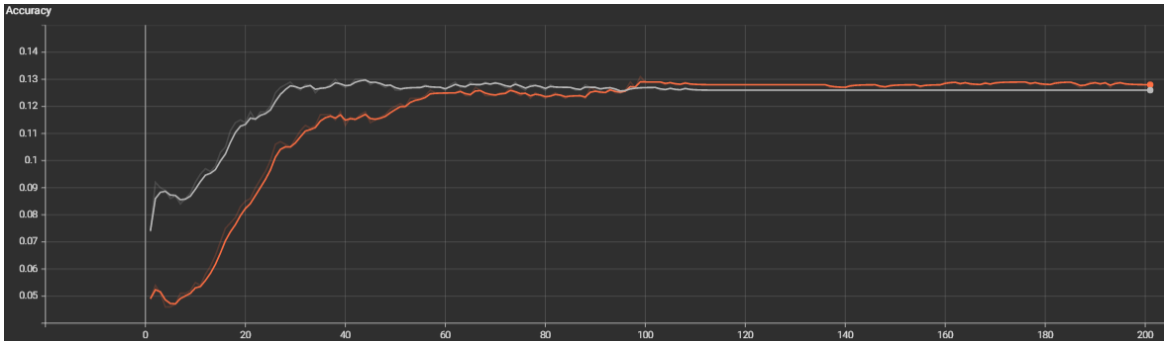
b) Stride

stride = 1 (no padding, kernel_size = 5) Expectation : Average accuracy..

Network1 :



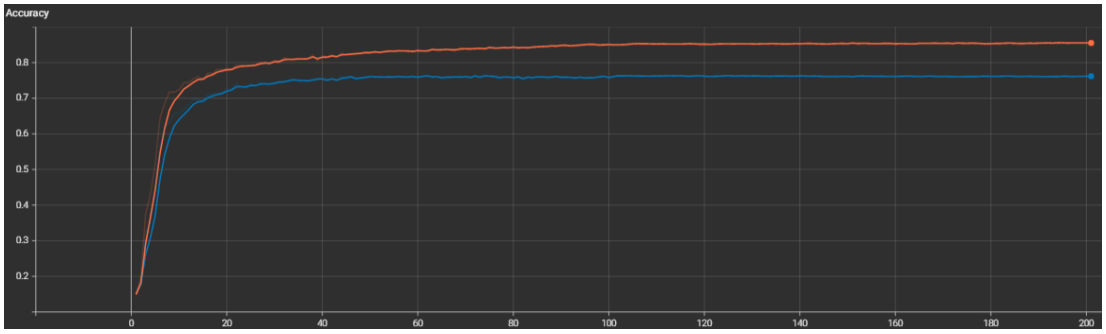
Network2 :



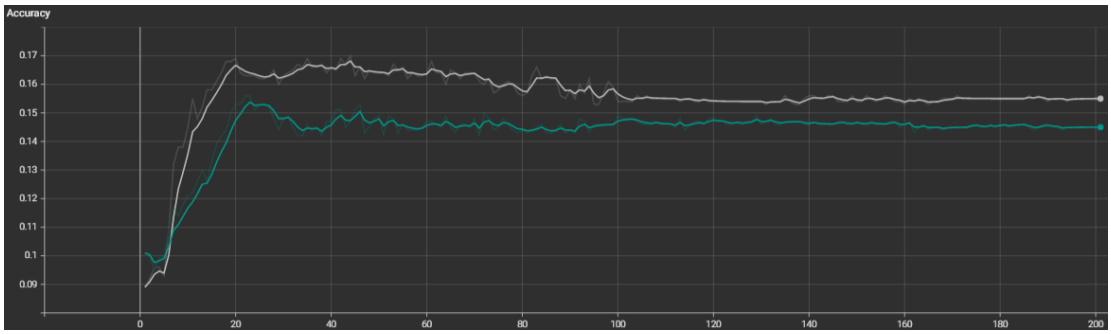
Network2 does not convolute which affects the learning rate drastically. I am not sure but kernel size might be affecting the accuracy as well. Because when we use two layer convolution, we extract more information.

stride = 3 (no padding, kernel_size = 3) Expectation : A little lower accuracy compared to previous case.

Network1 :



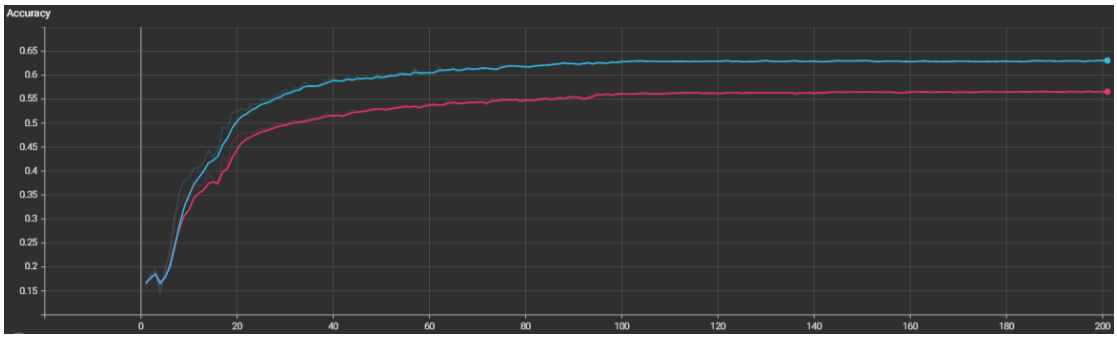
Network2 :



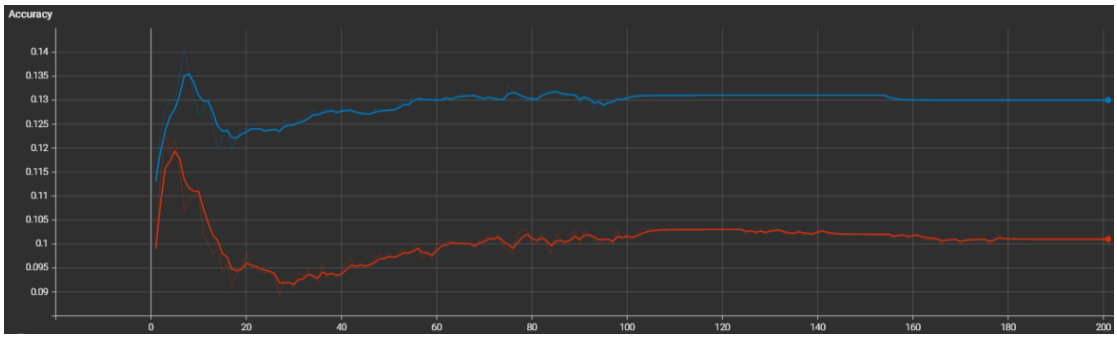
Yes. If stride = kernel_size, then we do not look at the same pixel again. Therefore, we get less number of samples which decreases the accuracy.

stride = 4 (no padding, kernel_size = 2) Expectation : Very low accuracy compared to previous ones, since we do not take some pixels into account.

Network1 :



Network2 :



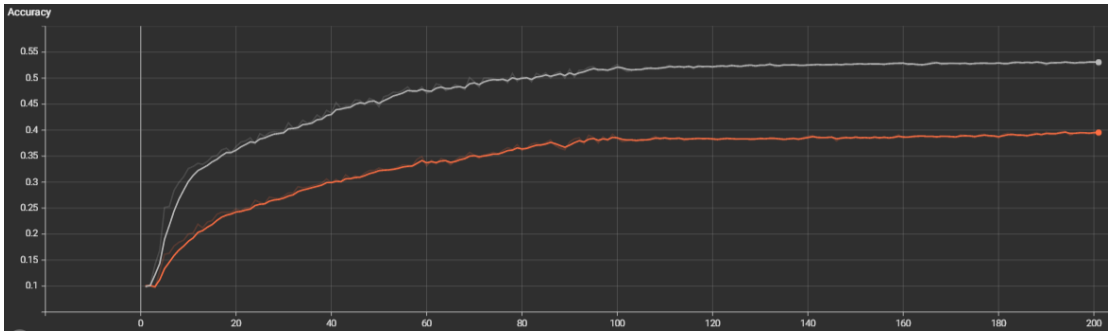
Accuracy decreased as expected. We have some pixels never used in the learning process.

c) Padding

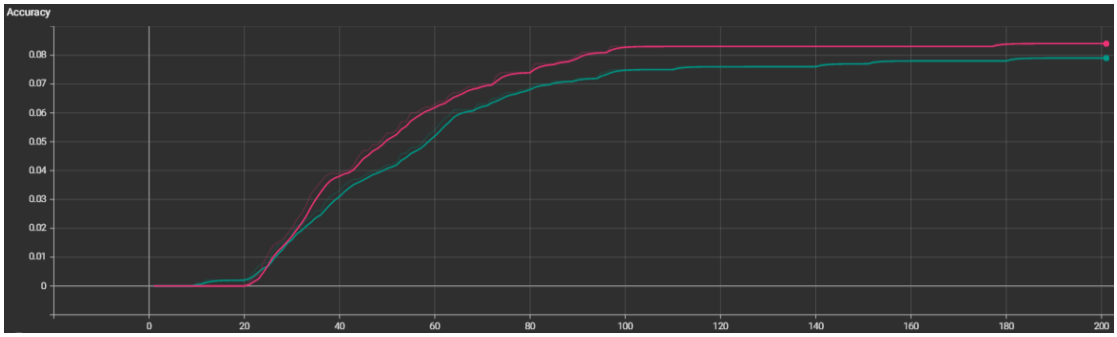
I will not try 0 padding since all of the experiments above have no padding.

padding = 1 (stride = 2, kernel_size = 3) Expectation : I think padding does not have too much effect on accuracy. I am expecting a high accuracy since other variables are normal.

Network1 :



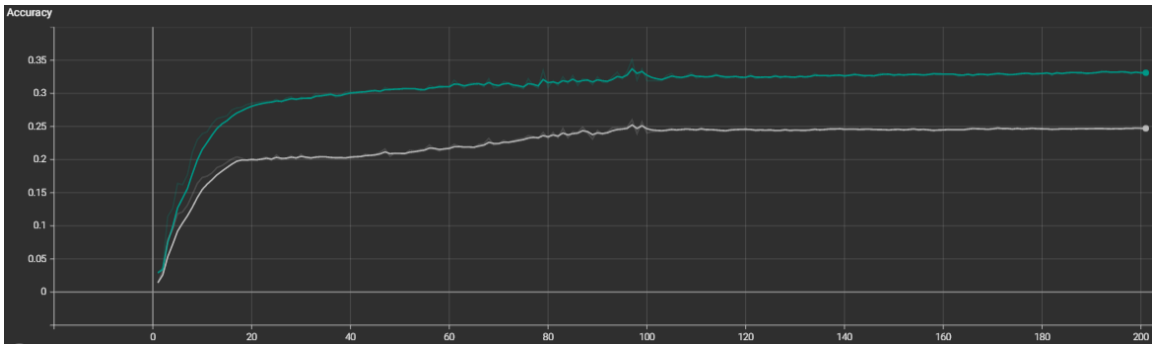
Network2 :



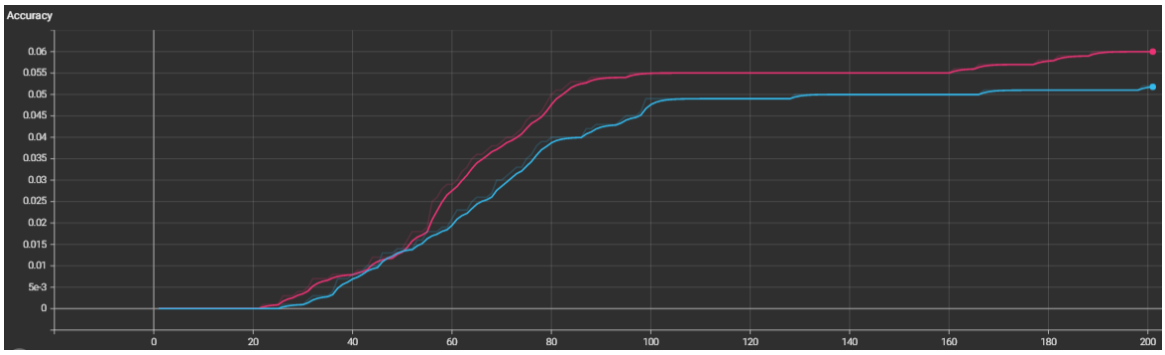
Seems like the networks are not optimized. Network1 results are not bad at least. Network2 performed poorly. Additional layer contributes to this experiment.

padding = 2 (stride = 2, kernel_size = 3) Expectation : I think padding will affect accuracy. I am expecting lower accuracy compared to the previous experiment.

Network1 :



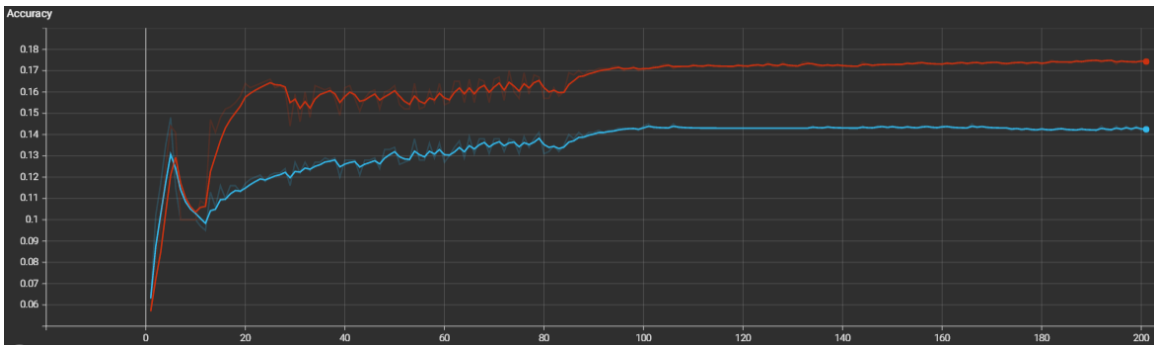
Network2 :



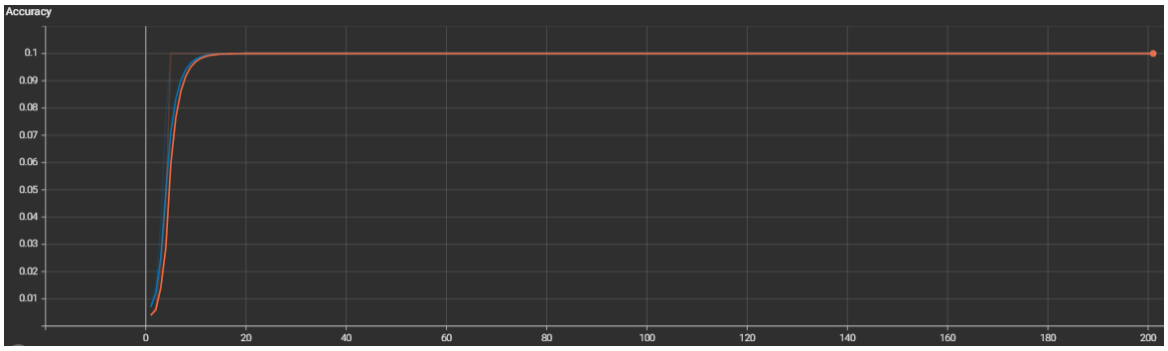
Yes. Accuracy is lower than before. Network2 converges very slowly while Network1 hits the ceiling after 30-40 epochs.

padding = 3 (stride = 2, kernel_size = 3) Expectation : I think padding will affect accuracy significantly. I am expecting low accuracy because I have multiple empty kernels that bias the prediction.

Network1 :



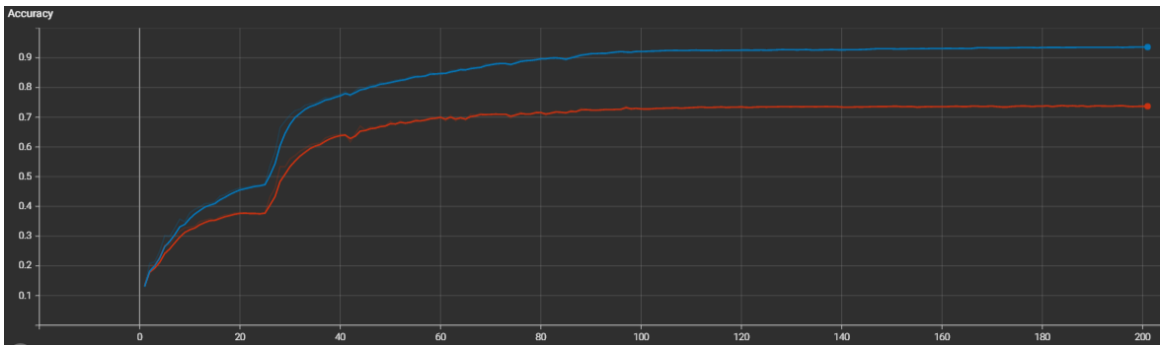
Network2 :



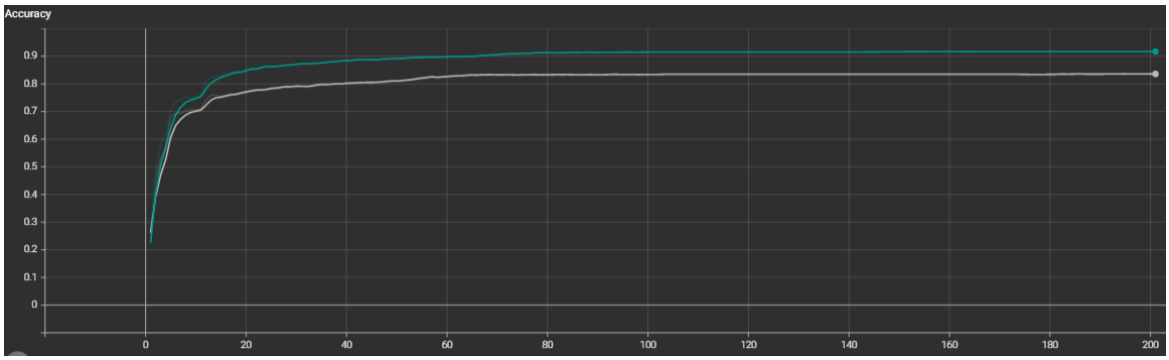
This is interesting. Network2 did not learn anything from the images, but the accuracy was higher than the previous experiment. I believe since there are 10 categories 0.1 is normal. That means in the previous experiment, paddings were giving false information to the system and actively sabotaged its functioning.

Comparison of Best Accuracies of Three Networks

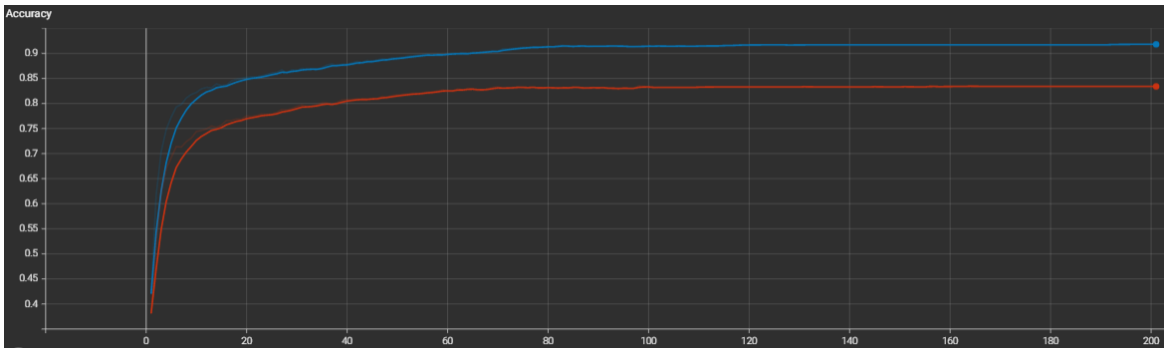
For Network1, the best result was observed when there was no padding, stride was 1 and kernel_size was 5.



For Network2, the best result was observed when there was no padding, stride was 1 and kernel_size was 10. Accuracy is very close to the best of Network1.



For Network3, there were no parameters provided since it is SLP. Accuracy is very close to the best of Network1 and Network2.



Observations

Most probably, given parameters were not perfect for the first two layers. By optimizing those, higher accuracy can be achieved. However, to my observations and according to my experiments, using complex network structures does not really affect the accuracy. Again, most probably this is a false claim, since we did not even touch some of the network variables (batch size, learning rate, gamma etc.)

Ceteris paribus, changing one variable is not a good idea for experimenting. Rather, changing the ratios among the variables would be better. For example, padding may significantly affect the accuracy if kernel size is similar to padding size (kernel size 3, padding 3) but it may help achieve better results if kernel size is higher (e.g. kernel size 10, padding 2).

Providing unnecessary data reduces accuracy. If the network has too many of them, this reduces the contribution of the real image and decreases the accuracy to a point even lower than the average. If I had time, I would like to try leaky ReLU to see if that network performs better or worse in this situation. I could not find how to use leaky ReLU in the code.

At the end of the notebook, there was a weights variable that I assume shows a similar image to digits that we obtained in HW3. I could not find a way to show or print it.

What I learned

- I learned how to define a network in the PyTorch library. I am not really comfortable with changing the system but I can at least adjust the parameters.
- I learned what is a stride, padding, kernel_size and how changing those parameters affect the accuracy.
- I am familiar with jupyter notebooks and I can do basic operations on it. Still a lot to learn about the extensions…

Questions that appeared in my mind

- Is there any formula or algorithm that gives the best variable size if other variables are provided. For example, what is the best kernel size if the image is 28x28, padding is 2 and stride is 2? Is there any method other than trial and error?
- We did not transform the data too much while importing it to the torch. If we had groups of different features in the data, how could I get specific features from the tensors? Is it available as it is in pandas like df[‘label’]?