Coding Assignment

1

20214418 Jihyeon Seong

AI604

Apr.4, 2022

1. Playing around with Neural Network

Problem 1

Building classifier for MNIST dataset

To make a suitable model for your problem, you need to understand the roles of different activation functions, regularization methods, and optimizers. Build your model to train the classifier for MNIST dataset. You can freely design the network architecture and training options.

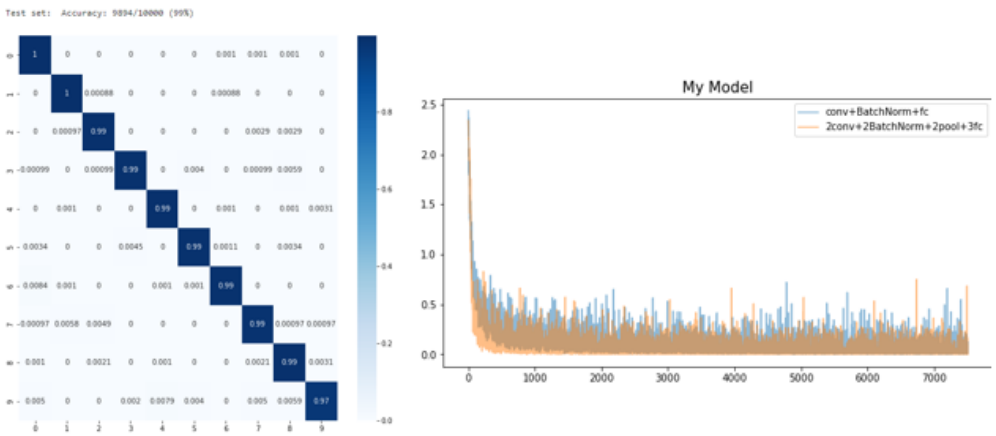
Your report should include the following to get full credit:

- Summary of your model architecture and motivation for your design choices
- Classification result including the confusion matrix for each class
- Analysis on the most confusing case (which ground-truth class was wrongly predicted as another class) and the possible reasons.

Solution.

With the experiment below (Problem2), I got hint to make best model with parameter tuned.

```
MNIST_Net(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv1_bn): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv2_bn): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=256, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```



I used 2conv layers with 2pooling and BatchNorm layers and activation function ReLU. Also added 3fc layers to classify 10 labels of MNIST datasets. With Adam Optimizer, I could train model stably.

So the result is 99% accuracy!
However, you can see that label 9 is the most difficult problem to solve for CNN, because it is quite similar to 0 or 8 with bad handwriting. I think we need more bad handwriting example of dataset to handle this problem.

Problem 2

Written Questions

a) What if we didn't clear up the gradients?

b) Why should we change the network into eval-mode?

c) Is there any difference in performance according to the activation function?

d) Is training gets done easily in experiment (3),(4) compared to experiment (1),(2)? If it doesn't, why not?

e) What would happen if there is no activation function?

f) Is there any performance difference before/after applying the batch-norm?

g) What may be the potential problems when training the neural network with a large number of parameters?

h) Given input image with shape:(H, W, C1), what would be the shape of output image after applying two convolutional filters with stride S and size F * F?

i) How did the performance and the number of parameters change after using the Convolution operation? Why did these results come out?

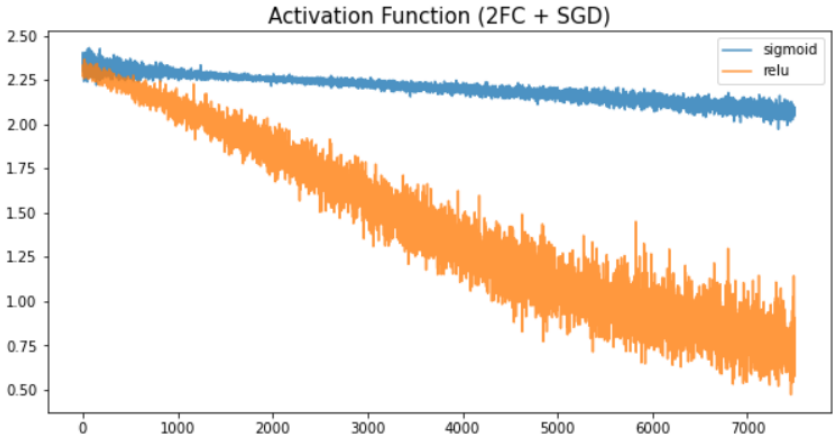
j) How did the performance change after using the Pooling operation? Why did these results come out?

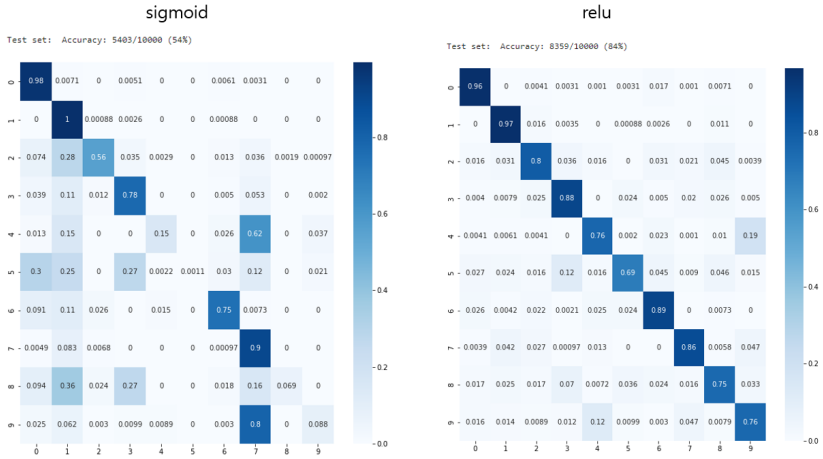
Solution.

a) Optimizer has to be initialize as zero. With gradient back propagation, the gradient is accumulated. Thus, to get appropriate result of gradient vectors' scale and orientation, it should be clear up commonly with zero.

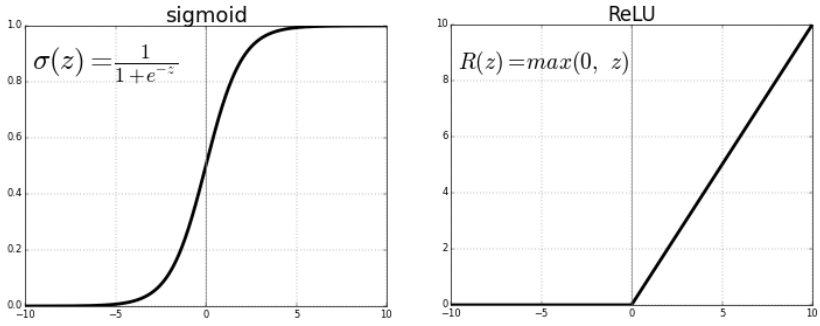
b) Evaluation mode makes model not to backward gradient with torch.no_grade and switch off the DropOut or BatchNorm layers. So we can get appropriate result for test dataset.

c) Activation function is important when training neural network. Here is training loss graph for how model is trained with different activation function with Sigmoid and ReLU, and also confusion matrix plot to see the performance comparison.



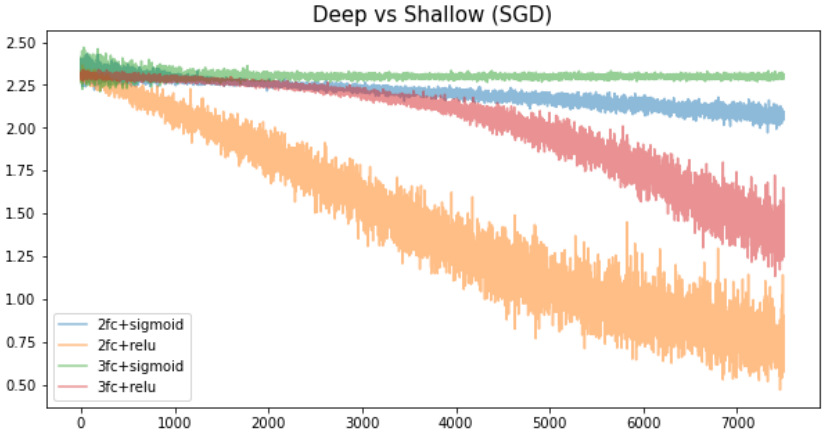


You can see that with ReLU activation function, the model is well trained and perform than Sigmoid.

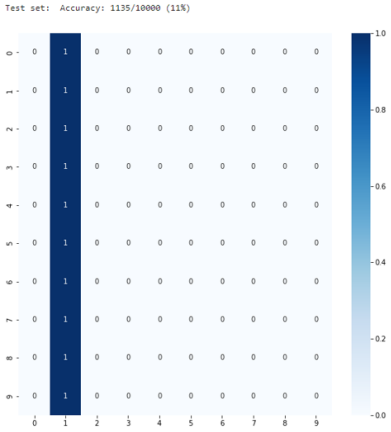


ReLU activation reduces likelihood of vanishing gradient. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases. So with ReLU function, we can train model more stable.

d) Intuitively, Most of us think that add one layer will make model performance better. However, it is not when the model's activation function and optimizer are not appropriate. Here is loss graph and confusion matrix for performance comparison.



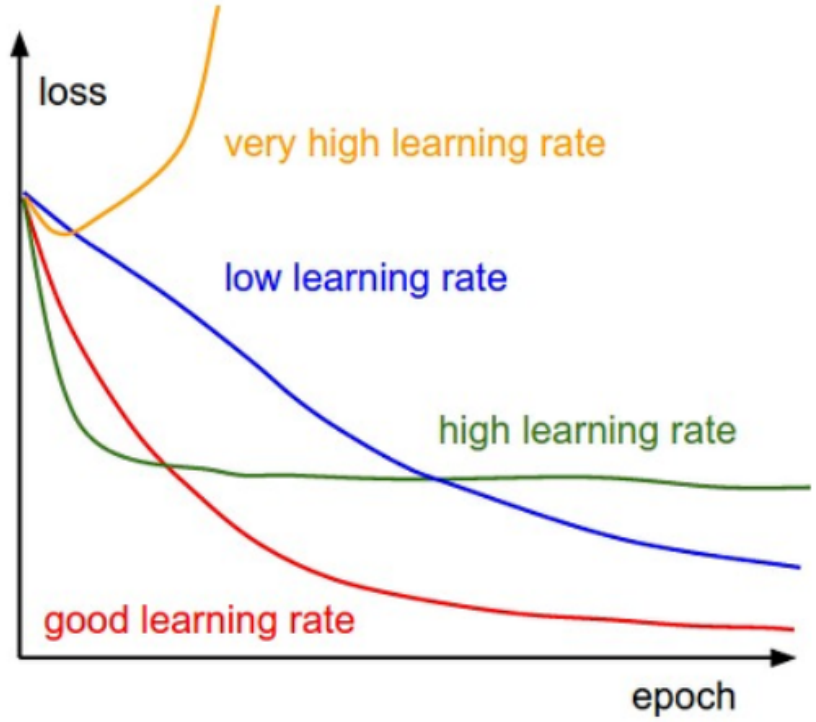
sigmoid



relu

With above result images, you can see that 3 fc layer model performs worse than 2 fc layer model, and it is especially severe in Sigmoid activation function. So we can think that first, Sigmoid activation function is not appropriate for deep network, and SGD is also not appropriate for optimizing deep network when we compare 2 and 3 fc layer with ReLU.

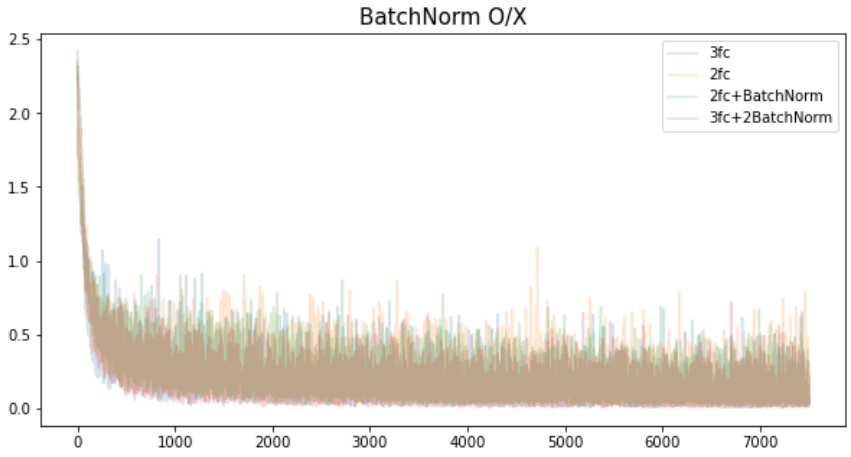
We noticed the problem of vanishing gradient with sigmoid function, so let's see SGD optimizer problem.

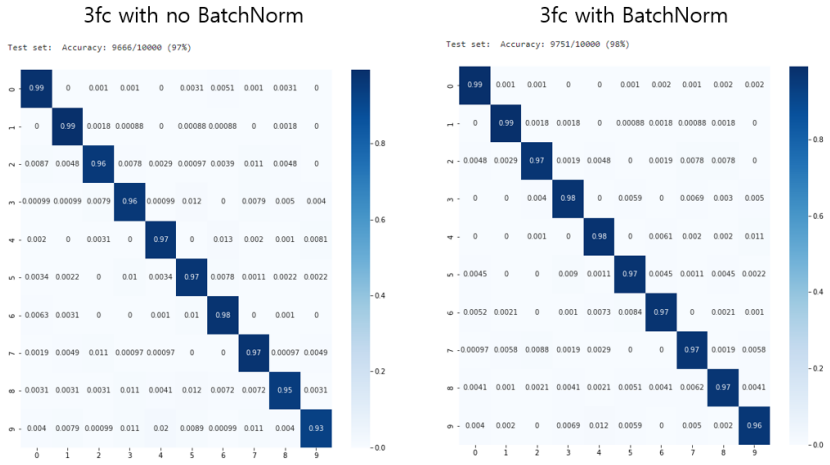


Here, You can see that SGD is quite difficult to get well tuned hyper-parameter, especially with learning rate.

e) With above question experiments, we see that activation function influence hardly on training deep neural network. And what would happen if there is no activation function?
Activation function is non-linear function that can make neural netowork's layer goes deeper and deeper, and this nonlinearlity makes model to solve non-linear problem. XOR problem is the most basic and standard example that we need activation function.

f) And now, we may can add Batch Normalization layer. What would happen? Here are loss graph and confusion matrix to compare training process and performance.





You can see that training process is quite similar but performance is little high than without BatchNorm.

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. We refer to the change in the distributions of internal nodes of a deep network, in the course of training, as Internal Covariate Shift.

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.

g) With above experiments, best result is below model architecture with 41250 parameters.

```
MNIST_Net(
  (fc0): Linear(in_features=784, out_features=50, bias=True)
  (bn0): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=50, out_features=30, bias=True)
  (bn1): BatchNorm1d(30, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=30, out_features=10, bias=True)
)
```

This is really huge model parameter number, and can bring some problems. The most important and significant problem is overfitting of model. Overfitting problem is model is only well performed with training dataset, and loose generalization. Also training efficiency is bad and we couldn't save time and computer resources.

h) Output image shape is,

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

i) With convolution layer instead fc layer can make huge difference in model architecture and performance.

```
MNIST_Net(
  (fc0): Linear(in_features=784, out_features=50, bias=True)
  (bn0): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=50, out_features=30, bias=True)
  (bn1): BatchNorm1d(30, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=30, out_features=10, bias=True)
)
```

```
MNIST_Net(
  (conv0): Conv2d(1, 8, kernel_size=(6, 6), stride=(2, 2))
  (conv0_bn): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc): Linear(in_features=1152, out_features=10, bias=True)
)
```

Fc layers

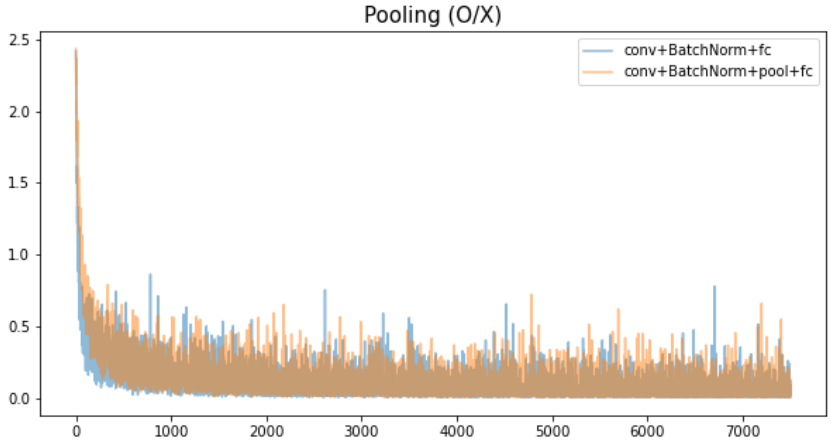
```
Test set:  Accuracy: 9751/10000 (98%)
```


Conv layers

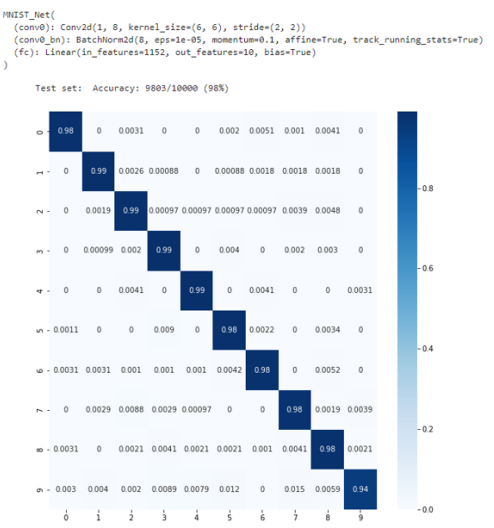
```
Test set:  Accuracy: 9803/10000 (98%)
```


Above architectures both have around 98% accuracy, but model parameter number is quite different between 41250 (fc layers) and 11842 (conv layers). Of course with similar performance makes you not care about what architecture to choose, but model efficiency is also important problem.

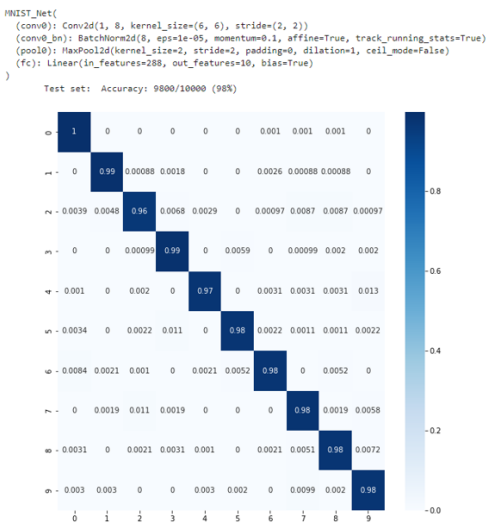
j) Pooling operation makes model more efficient.



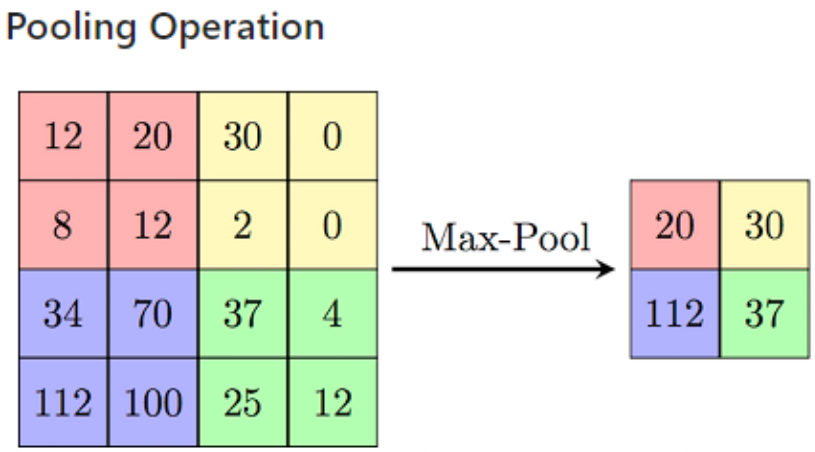
Without pooling layer



With pooling layer



With pooling layer, we can get same performance only with 3202 model parameters (compared with non-pooling model needs 11842).



Especially max pool layer, model can get locally and highly activated value, and send it to above layer. This makes translation invariance learning and makes CNN more efficient to image classification, which makes no matter where the object is, it can classify the object.

2. Playing around with Convolutional Neural Network

Problem 3

Implementation: VGG-19

By referring to the lecture slides, implement the VGG-19 model by yourself. Your report should include the following to get full credit:

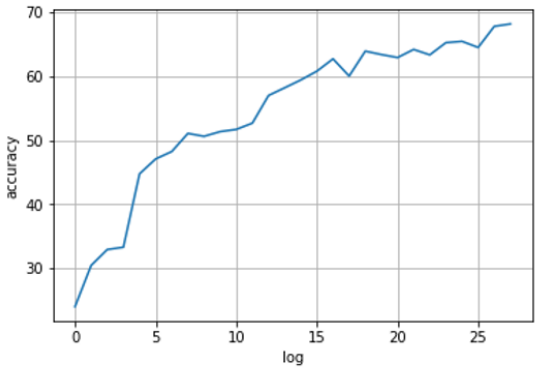
- Summary of the implemented model
- Result of the implementation test

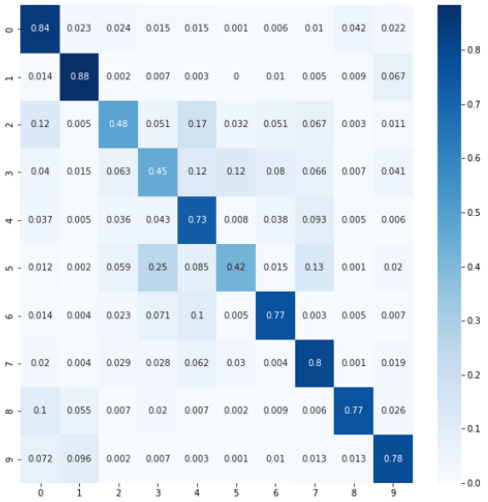
Solution.

I implemented VGG19 as followed architecture.

```
VGG19(
  (convlayer1): ConvBlock1(
    (main): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LeakyReLU(negative_slope=0.2)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): LeakyReLU(negative_slope=0.2)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
  (convlayer2): ConvBlock1(
    (main): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LeakyReLU(negative_slope=0.2)
      (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): LeakyReLU(negative_slope=0.2)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
  (convlayer3): ConvBlock2(
    (main): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LeakyReLU(negative_slope=0.2)
      (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): LeakyReLU(negative_slope=0.2)
      (4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): LeakyReLU(negative_slope=0.2)
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
```

```
  (convlayer4): ConvBlock2(
    (main): Sequential(
      (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LeakyReLU(negative_slope=0.2)
      (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): LeakyReLU(negative_slope=0.2)
      (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): LeakyReLU(negative_slope=0.2)
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
  (convlayer5): ConvBlock2(
    (main): Sequential(
      (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LeakyReLU(negative_slope=0.2)
      (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): LeakyReLU(negative_slope=0.2)
      (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): LeakyReLU(negative_slope=0.2)
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
  (linear): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=10, bias=True)
  )
)
```





With Cifar10 dataset, VGG-19 get 69% accuracy for test dataset.

Problem 4

Building classifier for CIFAR-10 dataset

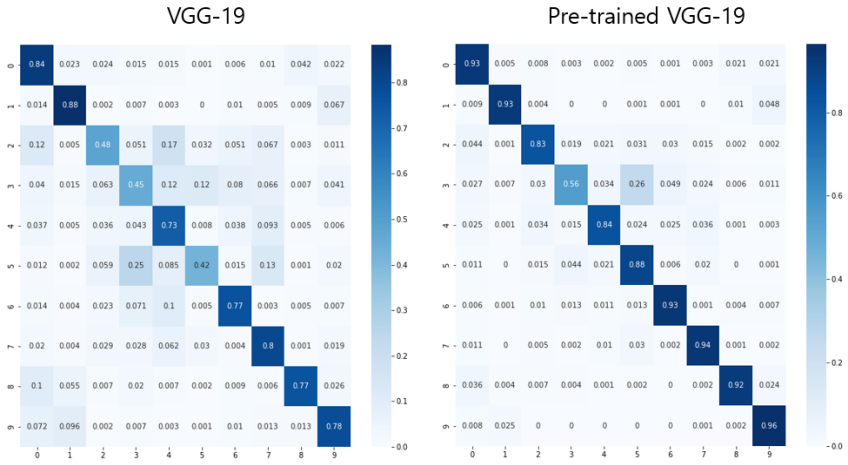
In fact, it is difficult to train a huge model from scratch. To see how difficult it is to re-train such large-scale models, you will train your VGG-19 model from scratch and compare its performance with the pre-trained VGG-19 model in CIFAR-10 dataset.

Your report should include the following to get full credit:

- Classification result for your model and the pre-trained model
- Confusion matrix normalized 0 to 1 and its analysis for each model

Solution.

With pre-trained VGG-19, model's performance gets really high.



Transfer learning can make difficult problem solve with huge dataset.