1 True/False Questions (30 pts)

1.1

True. The goal of Batch normalization is : Make the activations in each layer to have zero mean and unit variance.

1.2

True. We can fetch the contents of tensors at any time of the computation. Pytorch is very powerful.

1.3

False. Not all data augmentation technigues are always beneficial for any kinds of deep neural networks. For instance, it is unreasonable to use random vertical flip during MNIST training as digit 6 and digit 9 may look similar. We need make sure that the images after data augmentation are reasonable.

1.4

True. There are a lot of examples stating that larger epsilon works well in stackoverflow.For instance, the default value of 1e-8 for epsilon might not be a good default in general. For example, when training an Inception network on ImageNet a current good choice is 1.0 or 0.1.

1.5

False. Sometimes L-norm cannot work with dropuout together very well. So they can not guarantee a higher validation accuracy every time.

1.6

True. It is the inconsistent slope makes the training process of some neural network architectures unstable which is due to negative side of function.

1.7

True. From Lab2, I compare the weights distribution between L1 and L2. We can notice that L1 regularization encourages weights to 0 and weights distribution are sparse while weights distribution are less sparse with L2 regularization. The shape of L1 is a diamond, it leads to sparse weights distribution.

1.8

True. When we use ResNet with a lot of layers like ResNet101 or ResNet152, it is likely for us observe approximately 9x speedup for these layers as they are large M and N values.

1.9

False. It is squeeze and expand layers that lead to parameter and MAC saving and down-sample late in the network to spend more computation budgets (MACs) on larger activation maps.

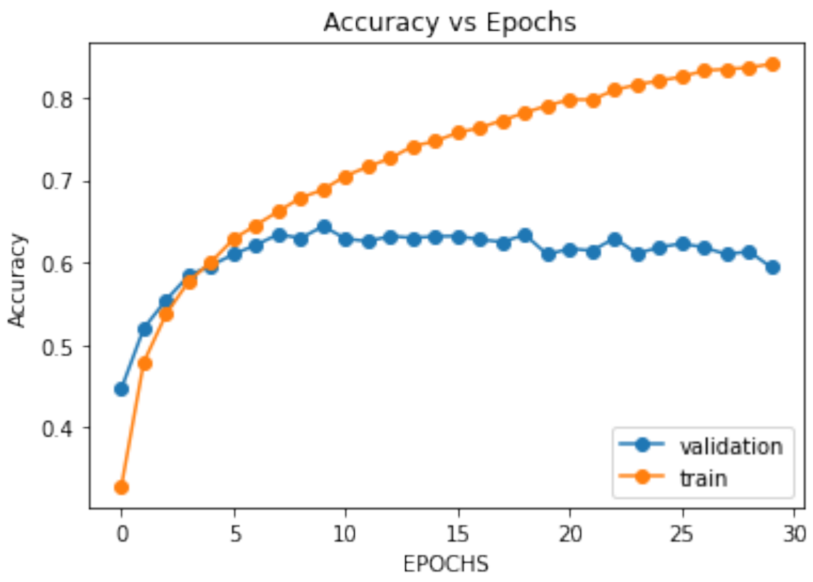
1.10

True. Shortcut connections make the loss surface smoother and easier to optimize with identity mapping and residual mapping.

2 Lab: Setting up the training pipeline (15 pts) + bonus(4 pts)

Summary:

This figure below shows the acuuracy of final model for 30 epochs and we can see that the largest of validation accuracy is larger than 63%



2 (a)

we can check the size of the output from the model. I used the code below to check the implementation of LeNet5.

```
[4]: #Verify the implementation of LeNet-5 Model

     # GPU check
     device = 'cuda' if torch.cuda.is_available() else 'cpu'
     if device =='cuda':
         print("Run on GPU...")
     else:
         print("Run on CPU...")

     # Model Definition
     net = LeNet5()
     net = net.to(device)

     # Test forward pass
     data = torch.randn(5,3,32,32)
     data = data.to(device)
     # Forward pass "data" through "net" to get output "out"
     out =  net(data) #Your code here
     # Check output shape
     assert(out.detach().cpu().numpy().shape == (5,10))
     print("Forward pass successful")
```

```
     Run on GPU...
     Forward pass successful
```

2 (b)

I used the following code to get zero mean and unit variance with the function named Transform.nomalize()

```
[3]: # Specify preprocessing function.
     # Reference mean/std value for
     transform_train  = transforms.Compose([
             transforms.ToTensor(), # convert input into tensor
             transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
         ])

     transform_val = transforms.Compose([
             transforms.ToTensor(),
             transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
         ])
```

2 (c)

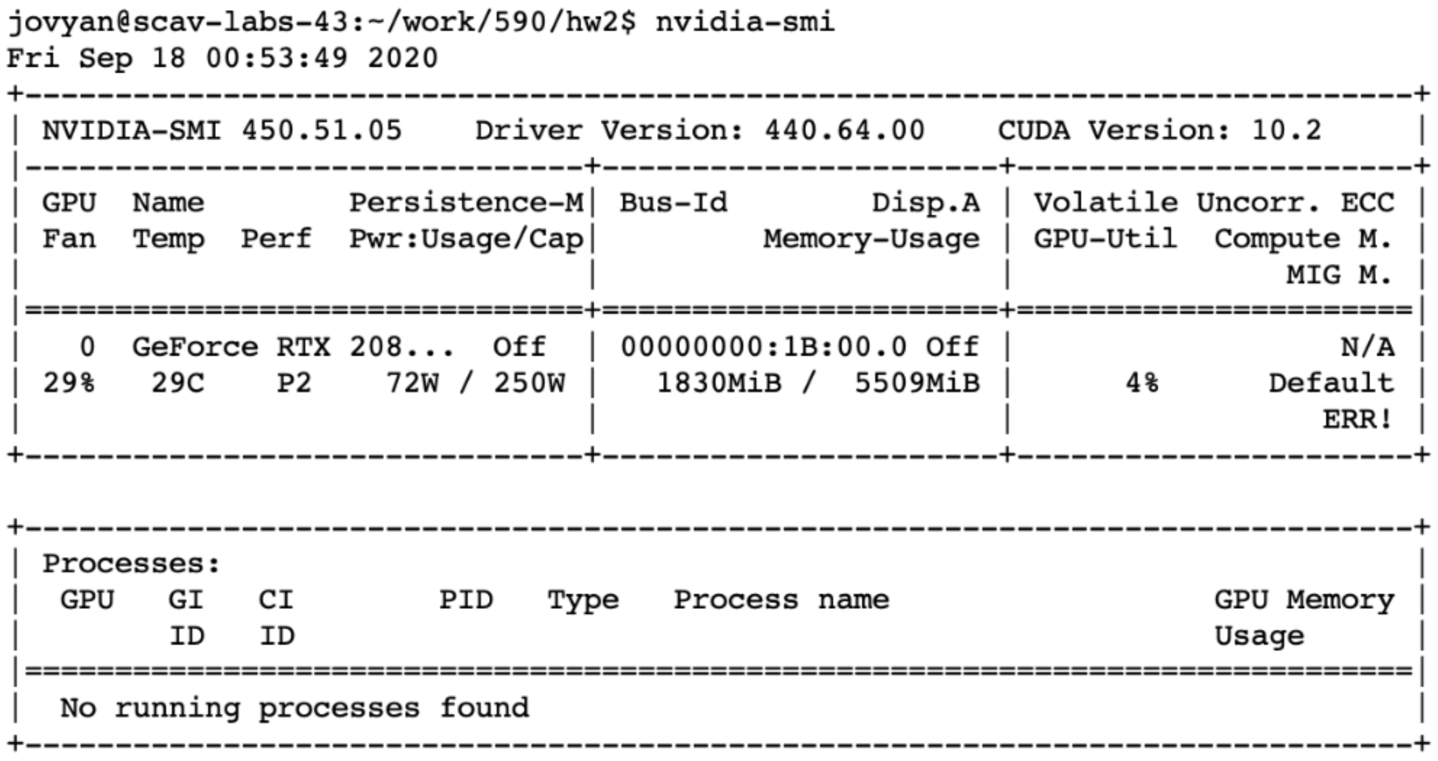
My code is below.

```
[5]: # You cannot change this line.
     from tools.dataloader import CIFAR10
     # Call the dataset Loader
     DATAROOT = "./data"
     TRAIN_BATCH_SIZE = 128
     VAL_BATCH_SIZE = 100
     trainset = CIFAR10(root=DATAROOT, train=True, download=True, transform=transform_train)
     trainloader = torch.utils.data.DataLoader(trainset, batch_size=TRAIN_BATCH_SIZE, shuffle=True, num_workers=4)
     valset = CIFAR10(root=DATAROOT, train=False, download=True, transform=transform_val)
     valloader = torch.utils.data.DataLoader(valset, batch_size=VAL_BATCH_SIZE, shuffle=False, num_workers=4)
```

```
Using downloaded and verified file: ./data/cifar10_trainval_F20.zip
Extracting ./data/cifar10_trainval_F20.zip to ./data
Files already downloaded and verified
Training dataset has 45000 examples!
Using downloaded and verified file: ./data/cifar10_trainval_F20.zip
Extracting ./data/cifar10_trainval_F20.zip to ./data
Files already downloaded and verified
Validation dataset has 5000 examples!
```

2 (d)

I use nvidia-smi to check if the code is running on GPU. We can see GPU memory is occupied and GPU is runing.



```
jovyan@scav-labs-43:~/work/590/hw2$ nvidia-smi
Fri Sep 18 00:53:49 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.05    Driver Version: 440.64.00    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce RTX 208...  Off  | 00000000:1B:00.0 Off |                  N/A |
| 29%   29C    P2    72W / 250W |   1830MiB /  5509MiB |      4%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

2 (e)

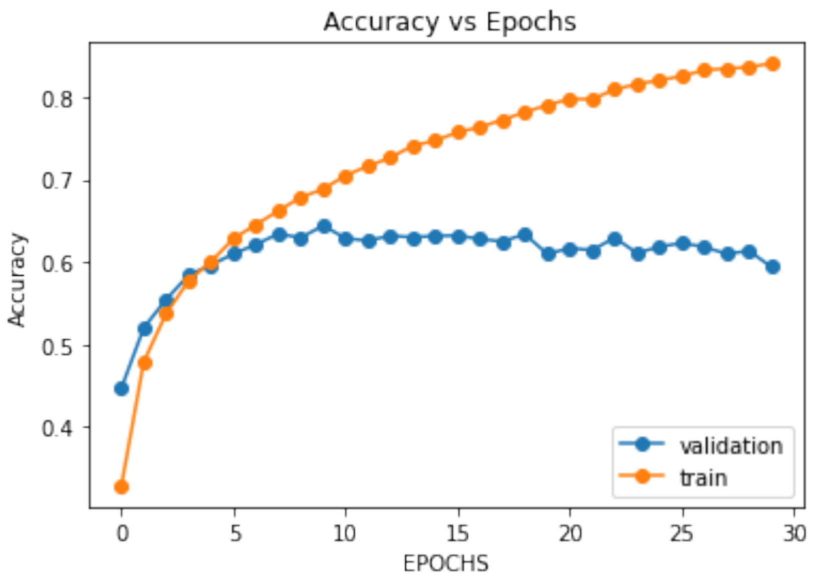
I followed the detailed instructions to implement the training process for LeNet model. Please see my code file.

2 (f)

Please see my code file.

2 (g)

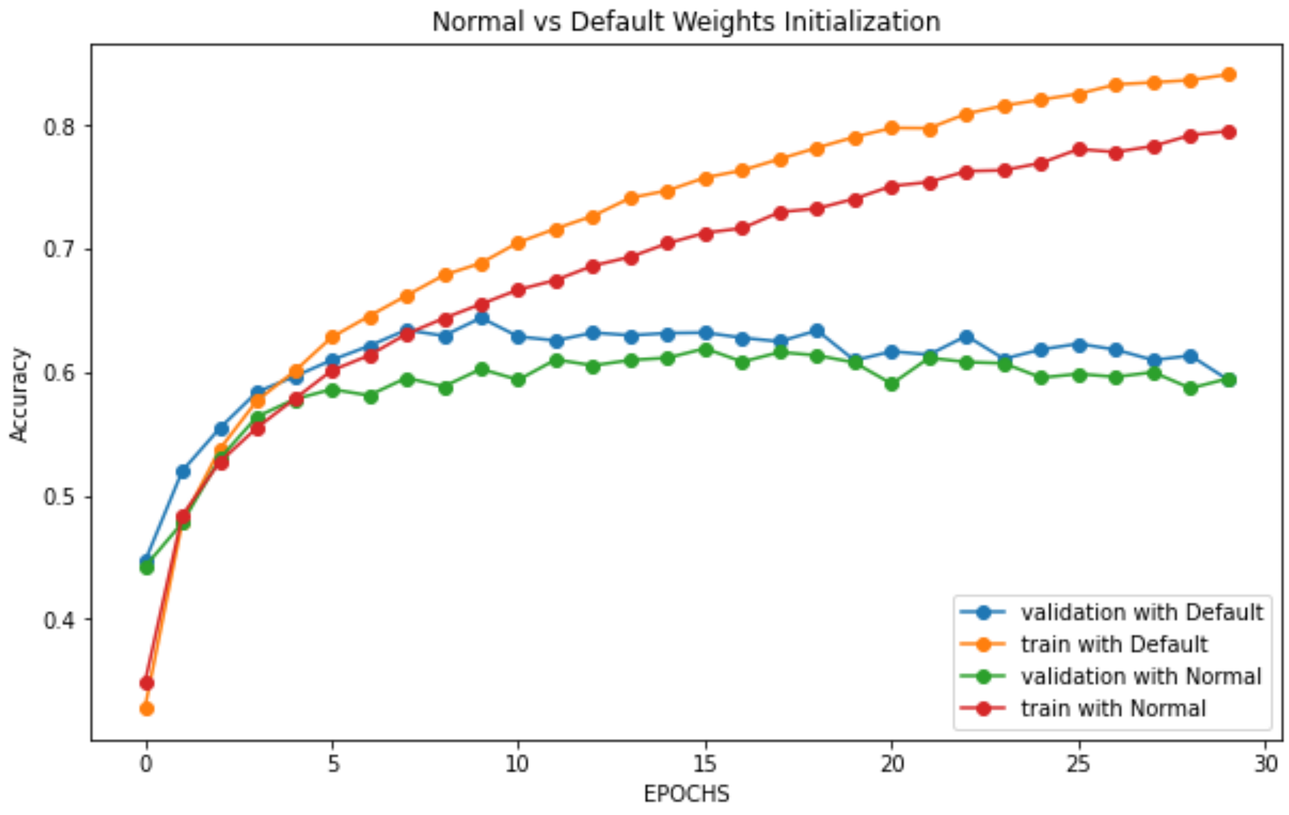
INITIAL LOSS before traning: 2.3024. I think the larger number of classes in the dataset, the larger loss value is. From the figure below, we can see that the training accuray keeps increasing but validation accuracy keeps flat after several epochs. It means that our model is overfitting.



2 Bonus

I used the code below to do Kaiming-Normal weights initialization and get the figure below. From the figure, we can see that the model with Kaiming-Normal weights initialization have a higher initial train accuracy than without Kaiming-Normal.

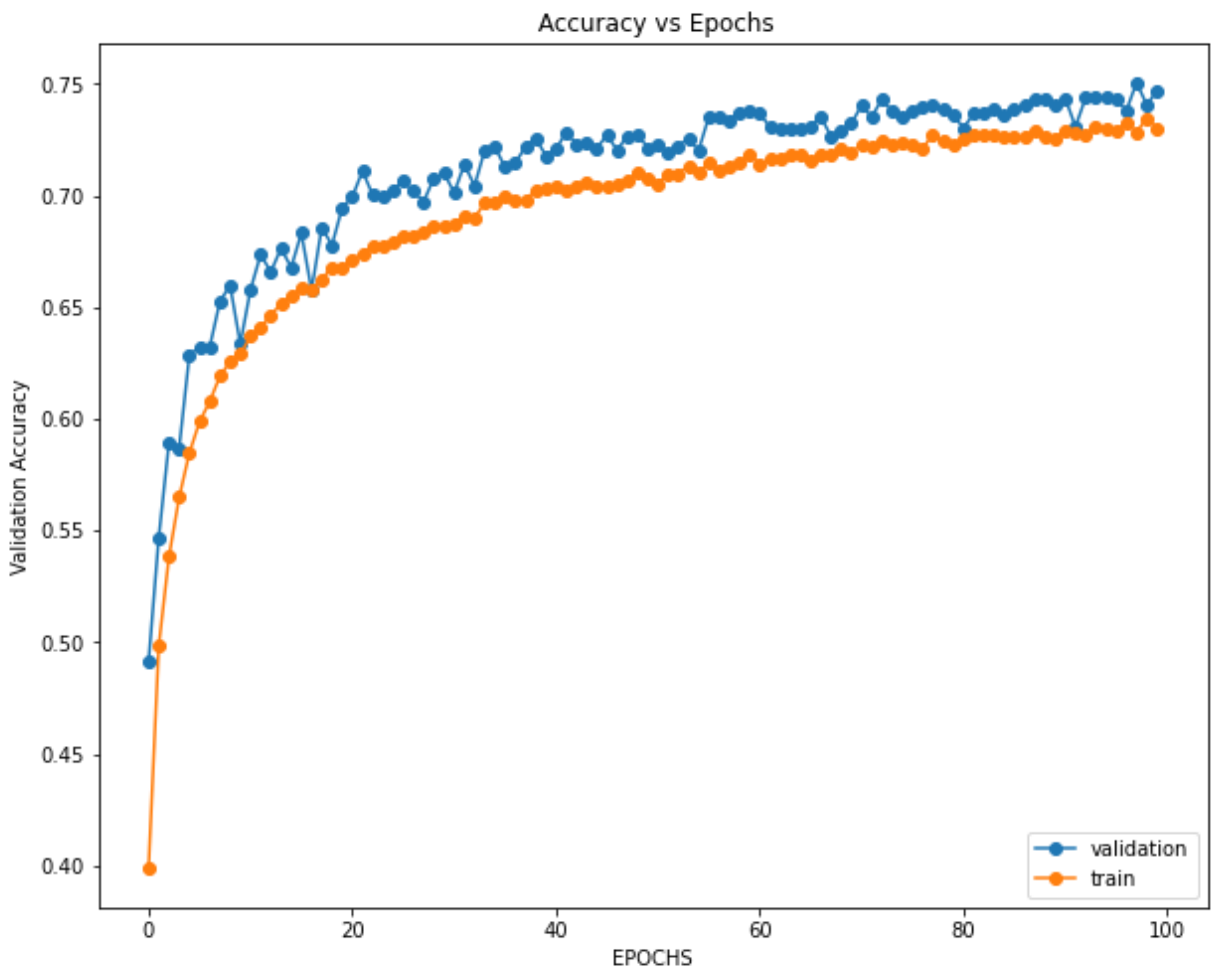
```
[12]: # Set model weights with kaiming normalization
      nn.init.kaiming_normal_(net.conv1.weight)
      nn.init.kaiming_normal_(net.conv2.weight)
      nn.init.kaiming_normal_(net.fc1.weight)
      nn.init.kaiming_normal_(net.fc2.weight)
      nn.init.kaiming_normal_(net.fc3.weight)
      avoidprint = 1
```



3 Lab: Improving the training pipeline (35 pts) + bonus (6 pts)

Summary

This figure below shows the acuuracy of final model for 100 epochs and we can see that validation accuracy is much larger than 70%



3 (a)

	Crop:padding=3	Crop:padding=6	Crop:padding=10
flip:p=0.3	0.6667	62.33	0.6284
flip:p=0.4	0.6710	62.37	0.5801
flip:p=0.6	0.6992	63.49	0.5962
flip:p=0.7	0.6720	64.56	0.5988

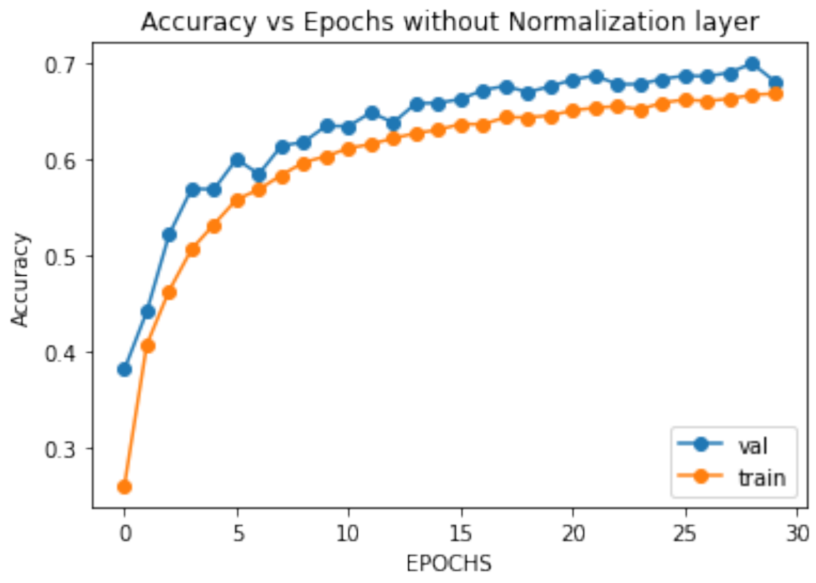
The best option is flip with p = 0.6 and crop with padding = 3.

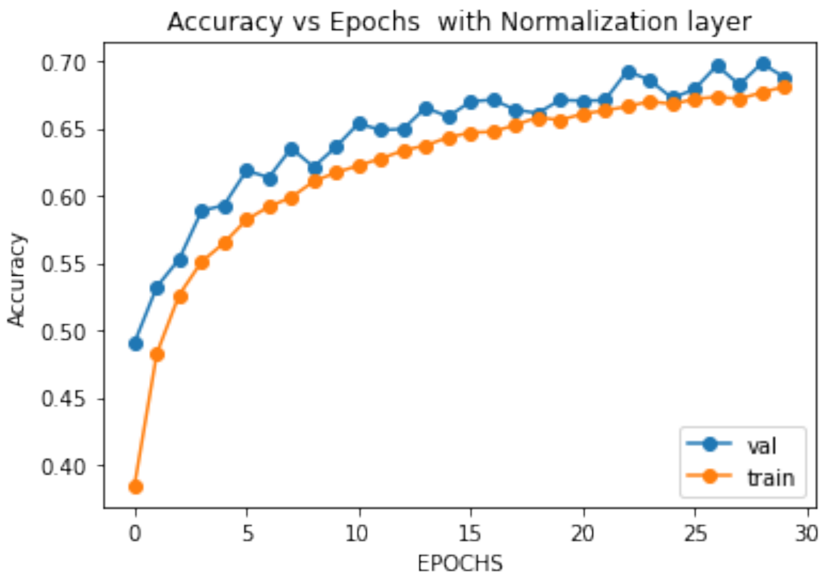
3 (b)

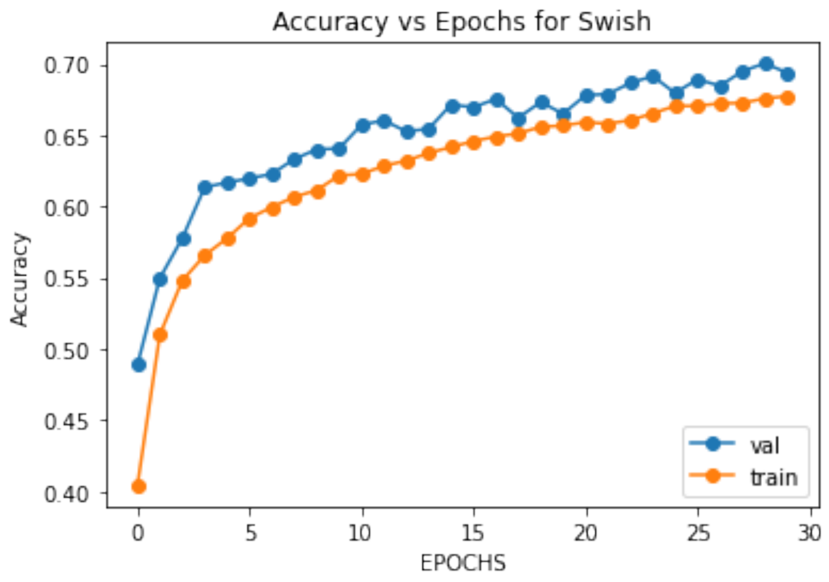
(1)After adding a batch normalization layer after each convolution/fully-connected layer except the last layer, I found out that during training process, the curve of accuracy become smoother as the figure below shows.

(2)During training process, the curve of accuracy become smoother after normalization while there has more oscillations before normalization. It means that we can make learning rate larger to improve speed after normalization. But before normalization, learning rate is large enough as there are some oscillations.

(3)Swish improves the performance. As the figure below shows that the validation accuracy imporved by a litte bit compared to Relu (the first and second figures below are generated by relu). I found out the largerst validation accuracy for Swish is 0.6913 while it is 0.6879 for Relu. This is because the non-monotonic "bump" increases the complexity of model while processing negative inputs.







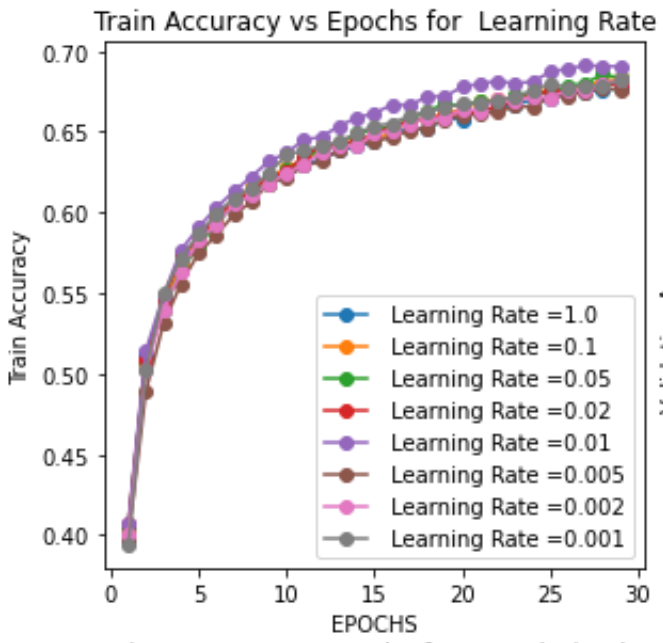
3 (c)

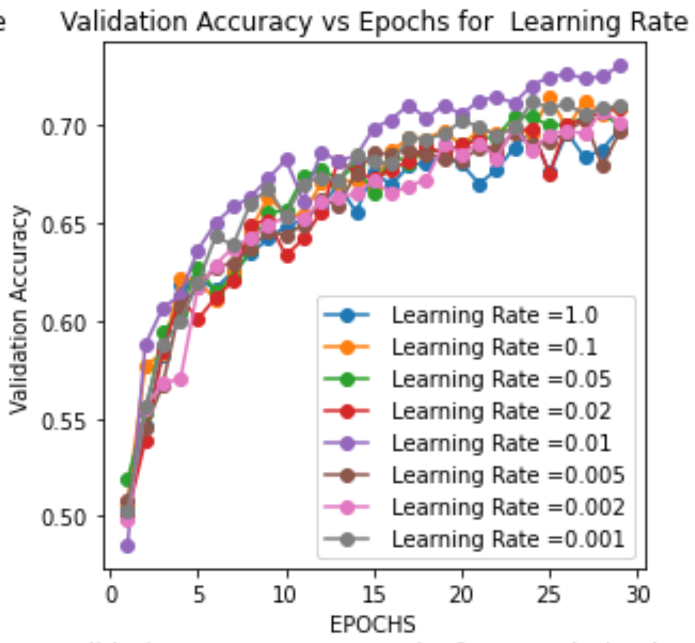
3 (c1)

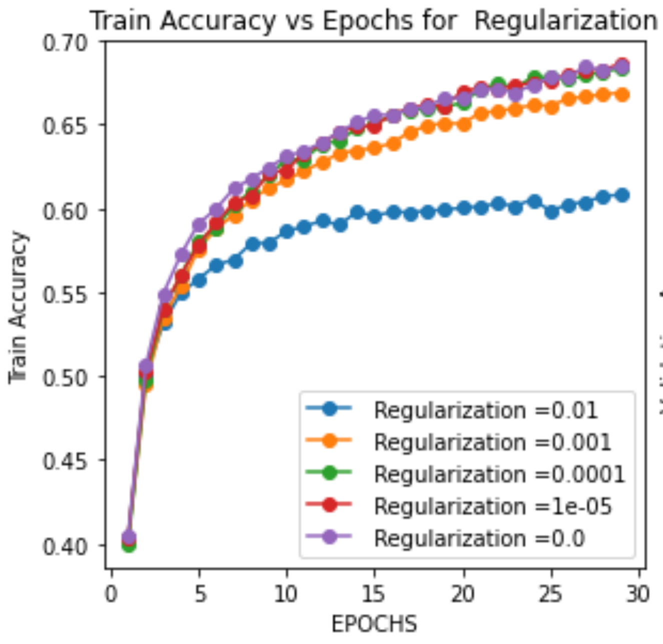
No, larger learning rates cannot guarantee better peformances. From the figure below, we can notice that there are a lot of oscillations for the curve with larger learning rate and they cannot reach an optimum. Too large learning rate will lead to oscillations and may not reach an optimum while too small learning rate cost a long time. So it is important for us to choose a suitable learning rate. The best learning rate here is 0.01

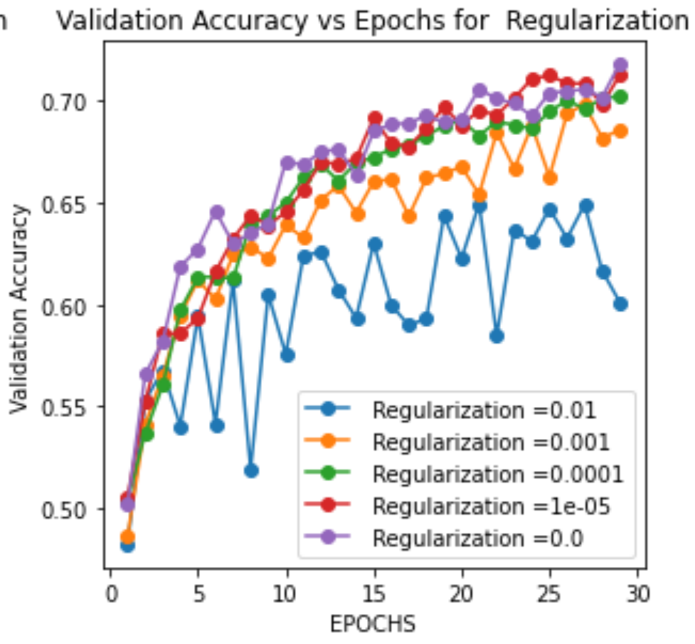
3 (c2)

Regularization is very important hyperparameter. As the figure below shows, if regularization is too small, the model will be overfitting. If regularizatoin is too large, the model will be underfitting. The best regularization here is 1e-04



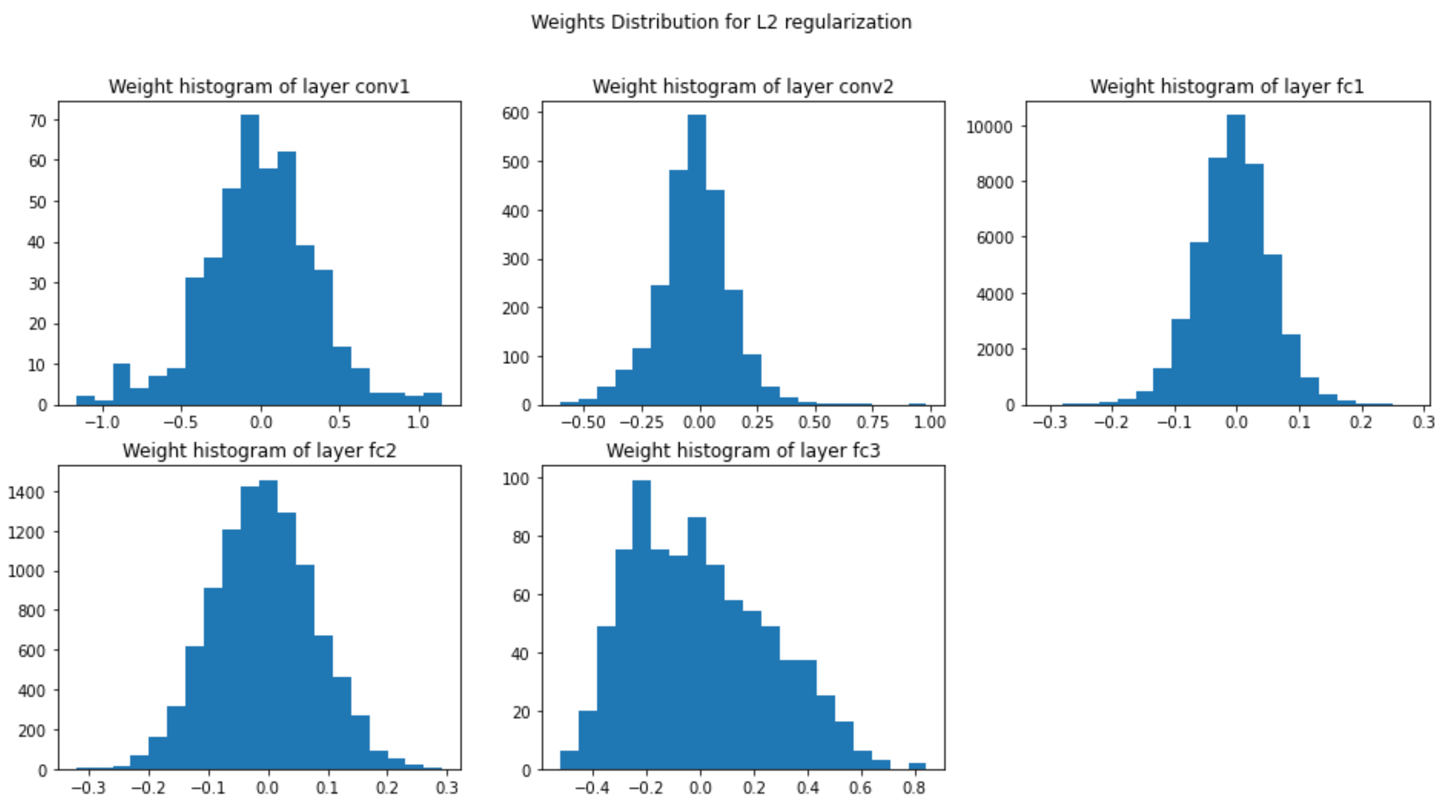


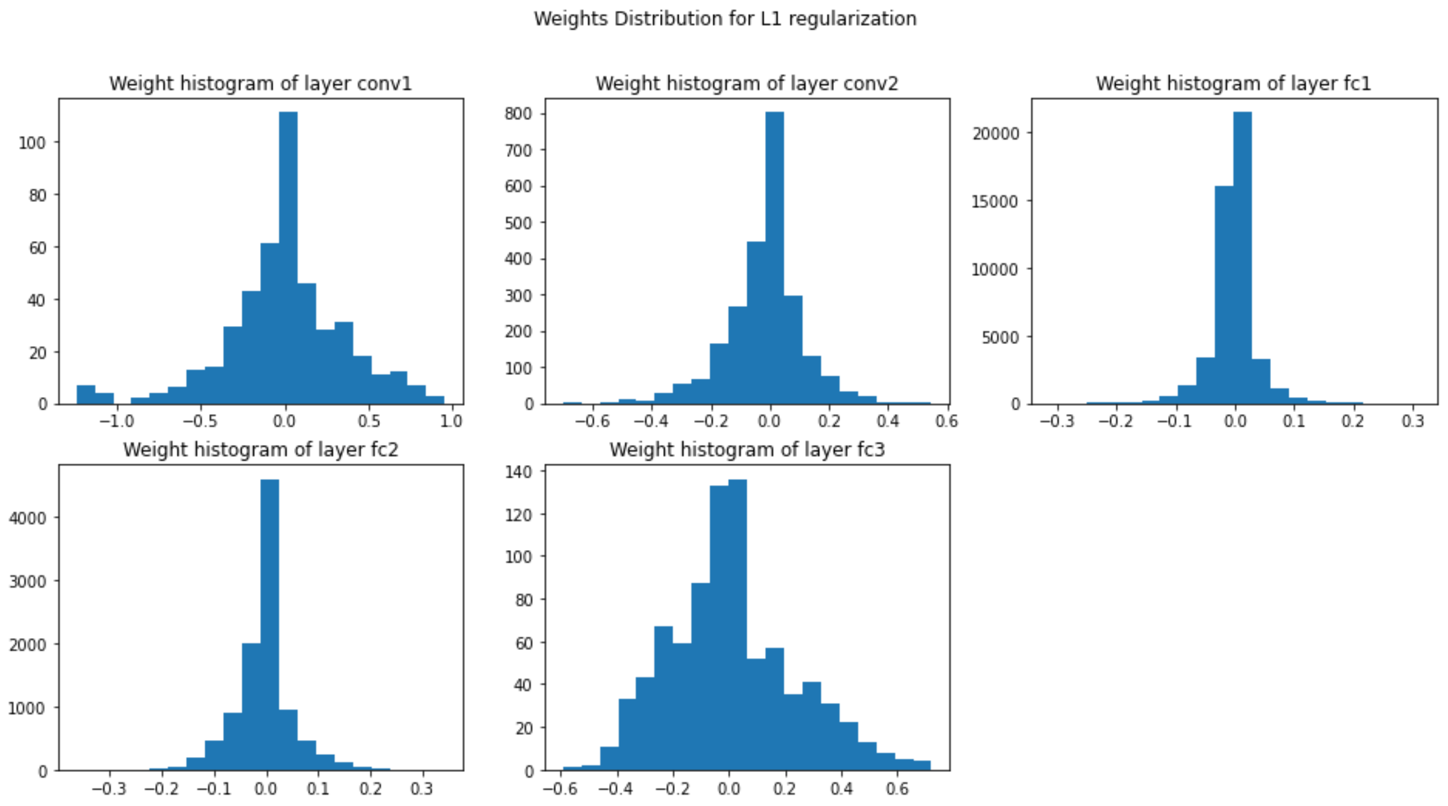




3 c Bonus

From the figures below, we can notice that L1 regularization encourages weights to 0 and weights distribution are sparse while weights distribution are less sparse with L2 regularization.


Weights Distribution for L2 regularization
Weight histogram of layer conv1
Weight histogram of layer conv2
Weight histogram of layer fc1
Weight histogram of layer fc2
Weight histogram of layer fc3


Weights Distribution for L1 regularization
Weight histogram of layer conv1
Weight histogram of layer conv2
Weight histogram of layer fc1
Weight histogram of layer fc2
Weight histogram of layer fc3

4 Lab: Advanced CNN architectures (20 pts)

(a) Implemented. Please see the code file. And the validatoin accuracy keeps around 91%.

(b) Please see predictions.csv