# Get started with Deep Neural Networks

Sun Siyi[1],

**1 Information and Computational Science Department, University of Science and Technology Beijing, Beijing 100083 CHN**

## Tasks

### 0.1 Task 1

Implement LeNet and do image classification with CIFAR10 dataset.

### 0.2 Task 2

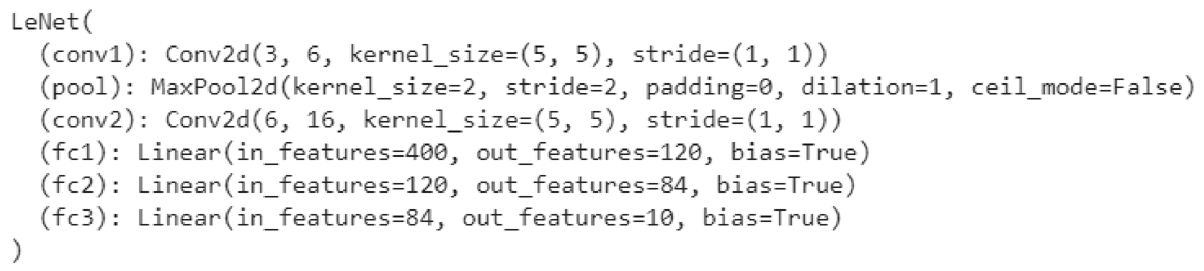
Use Tensorboard to visualize LeNet training procedure.

### 0.3 Task 3

Implement VGG-Net-16 with CIFAR10 dataset. Report the same results as you did in LeNet and compare the testing results. Analyze the results in the writeup. Code can be freely downloaded from Github.

## 1 Task 1

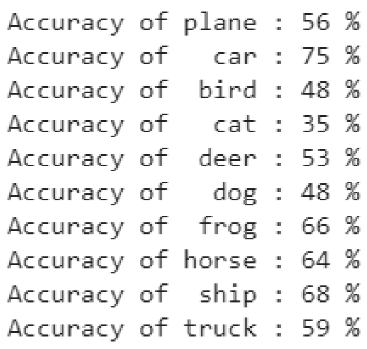
In this part, I built the LeNet network as below and do image classification with CIFAR10 dataset. I wrote my understanding of the code beside the code based on what I searched and obtained from the Internet.



```
LeNet(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (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))
  (fc1): Linear(in_features=400, 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)
)
```

**Figure 1.** LeNet network

With the hyper parameters that lr(learning rate)=0.001 and momentum=0.9, the final testing accuracy is shown below.



```
Accuracy of plane : 56 %
Accuracy of   car : 75 %
Accuracy of  bird : 48 %
Accuracy of   cat : 35 %
Accuracy of  deer : 53 %
Accuracy of   dog : 48 %
Accuracy of  frog : 66 %
Accuracy of horse : 64 %
Accuracy of  ship : 68 %
Accuracy of truck : 59 %
```

**Figure 2.** Accuracy of each categories

## 2 Task 2

### 2.1 Results

In this part, I use tensorboard to visualize my LeNet training structure shown below:

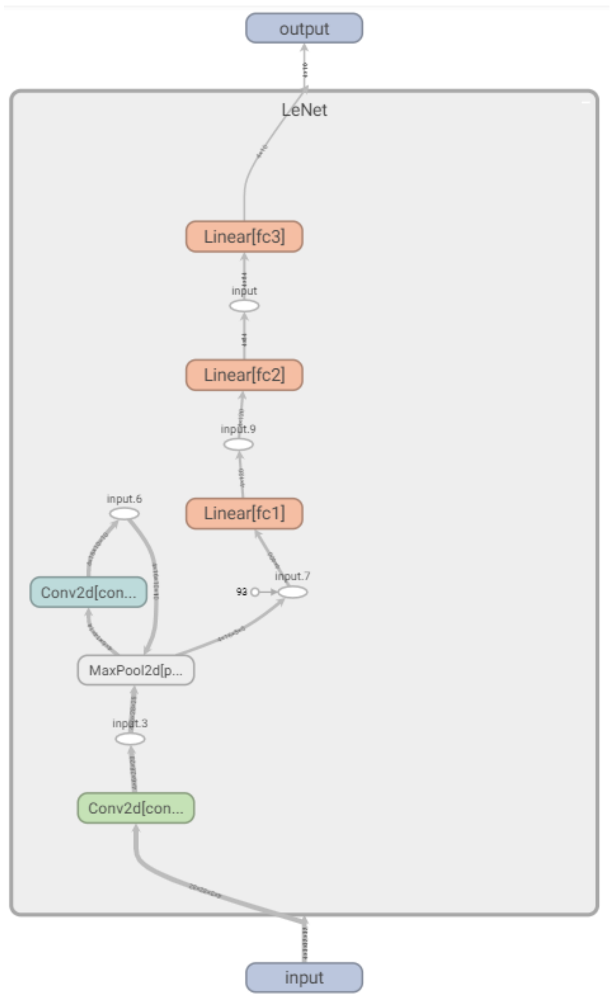


**Figure 3.** Visualization of LeNet Structure

I also obtained the training step-versus-training loss curve as follow:

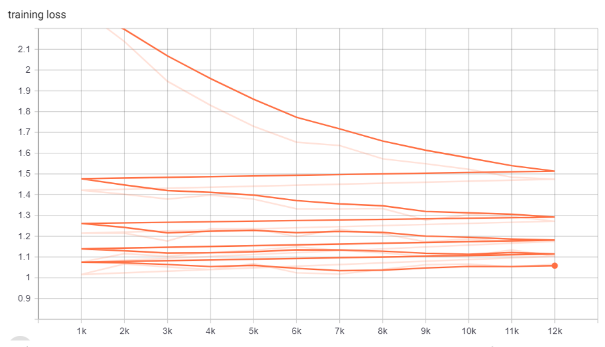


**Figure 4.** Training step-versus-Training loss curve

As is shown in the diagram, the decrease speed scarcely slow down with the increase of iteration times, which may indicates that polished model is more difficult to be refined compared with initial model.

Although we may have a general concept of the accuracy of each class, adopting the ”PR Curves” to illustrate the accuracy of specific class can be a better choice. Fig.5 and Fig.6 show high consistency with Fig.2.

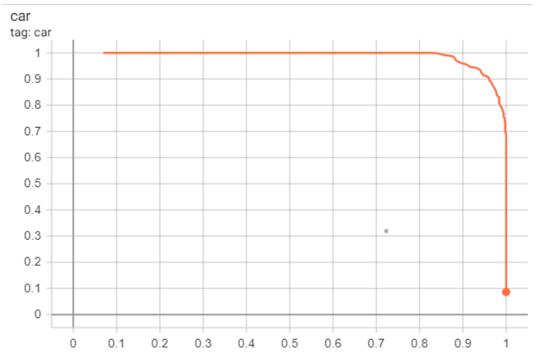


**Figure 5.** PR Curves of car

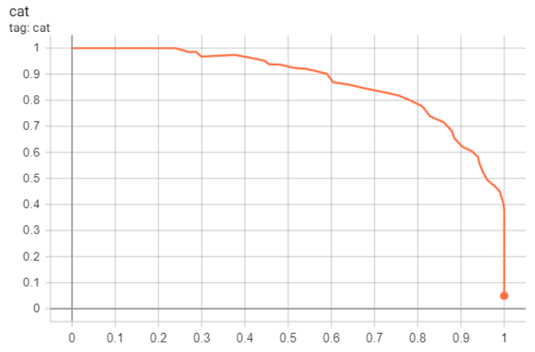
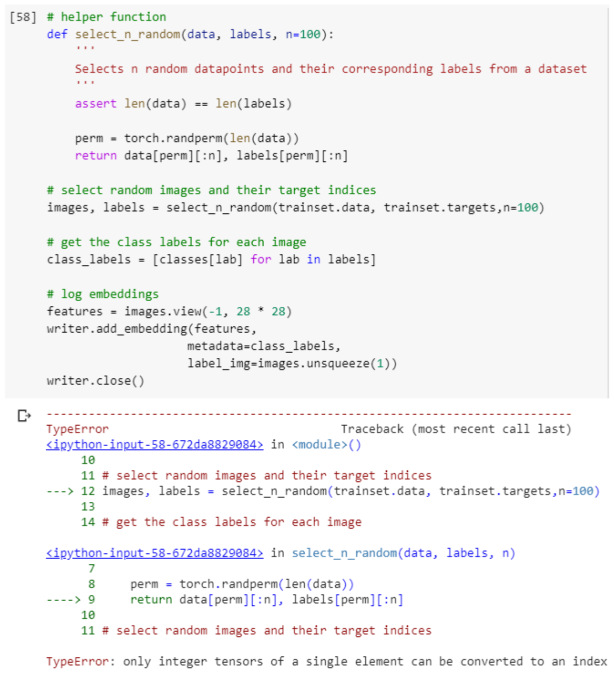


**Figure 6.** PR Curves of cat

## 2.2 Problems

I meet some problems when I tried to add a ”Projector” to TensorBoard. I searched for the mistake message, but almost no same message as what i get in the picture below:



```
[58] # helper function
     def select_n_random(data, labels, n=100):
         '''
         Selects n random datapoints and their corresponding labels from a dataset
         '''
         assert len(data) == len(labels)

         perm = torch.randperm(len(data))
         return data[perm][:n], labels[perm][:n]

     # select random images and their target indices
     images, labels = select_n_random(trainset.data, trainset.targets,n=100)

     # get the class labels for each image
     class_labels = [classes[lab] for lab in labels]

     # log embeddings
     features = images.view(-1, 28 * 28)
     writer.add_embedding(features,
                         metadata=class_labels,
                         label_img=images.unsqueeze(1))
     writer.close()
```

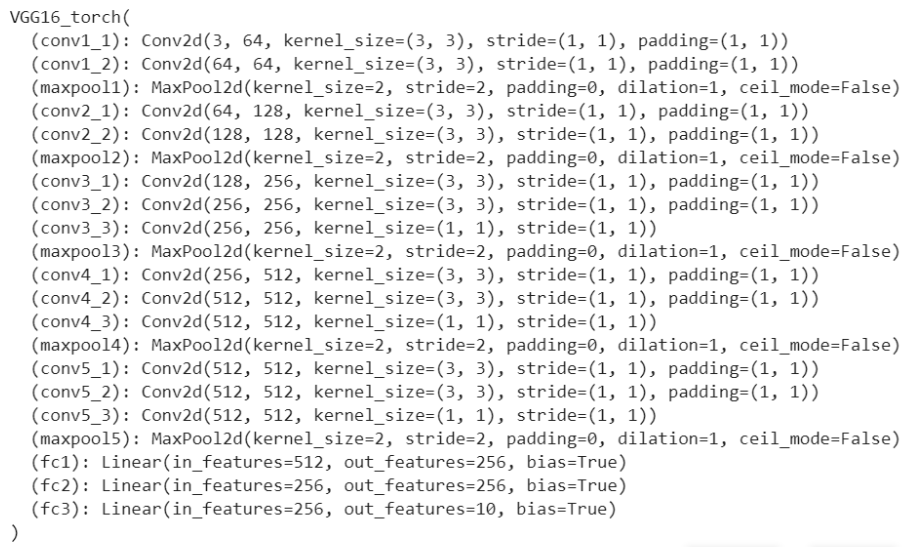
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-672da8829084> in <module>()
     10 
     11 # select random images and their target indices
---> 12 images, labels = select_n_random(trainset.data, trainset.targets,n=100)
     13 
     14 # get the class labels for each image

<ipython-input-58-672da8829084> in select_n_random(data, labels, n)
      7 
      8     perm = torch.randperm(len(data))
----> 9     return data[perm][:n], labels[perm][:n]
     10 
     11 # select random images and their target indices

TypeError: only integer tensors of a single element can be converted to an index
```

**Figure 7.** Mistake message

# 3 Task 3

## 3.1 Results

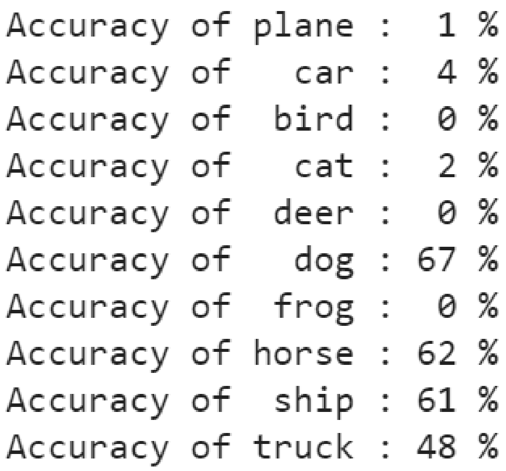
In this part, I built the VGG-Net-16 network as below and do image classification with CIFAR10 dataset.



```
VGG16_torch(
  (conv1_1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv1_2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2_1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2_2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3_1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3_2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3_3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
  (maxpool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4_1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4_2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4_3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (maxpool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv5_1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5_2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5_3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (maxpool5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=512, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=10, bias=True)
)
```

**Figure 8.** VGG-Net-16 network

With the hyper parameters that lr(learning rate)=0.0001, the final testing accuracy is shown below.



```
Accuracy of plane :  1 %
Accuracy of   car :  4 %
Accuracy of  bird :  0 %
Accuracy of   cat :  2 %
Accuracy of  deer :  0 %
Accuracy of   dog : 67 %
Accuracy of  frog :  0 %
Accuracy of horse : 62 %
Accuracy of  ship : 61 %
Accuracy of truck : 48 %
```

**Figure 9.** Accuracy of each categories

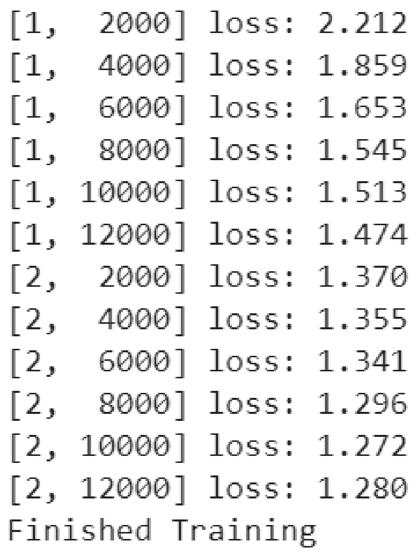
## 3.2 Comparison

Speed:

The speed of training VGG-Net-16 is much more longer(even use GPU) than LeNet. In my opinion, the complexity and multi-layer of the VGG-Net-16's structure leads to this result.

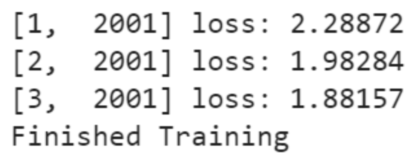
Accuracy:

It's hard to get a conclusion when we compare the accuracy. If train VGG-Net-16 long enough, the final accuracy must outperform than LeNet. The loss diagram are shown below. Because the limit of training time, I only trained 2000 minibaches for each epoch when training VGG-Net-16. In real situation, the training time is limited by many factors, which means it's impossible to train VGG-Net-16 long enough. In this situation, LeNet may achieve a comparative performance at the early period when use the same time of traning.



```
[1,  2000] loss: 2.212
[1,  4000] loss: 1.859
[1,  6000] loss: 1.653
[1,  8000] loss: 1.545
[1, 10000] loss: 1.513
[1, 12000] loss: 1.474
[2,  2000] loss: 1.370
[2,  4000] loss: 1.355
[2,  6000] loss: 1.341
[2,  8000] loss: 1.296
[2, 10000] loss: 1.272
[2, 12000] loss: 1.280
Finished Training
```

**Figure 10.** LeNet: Loss of iteration



```
[1,  2001] loss: 2.28872
[2,  2001] loss: 1.98284
[3,  2001] loss: 1.88157
Finished Training
```

**Figure 11.** VGG-Net-16: Loss of iteration