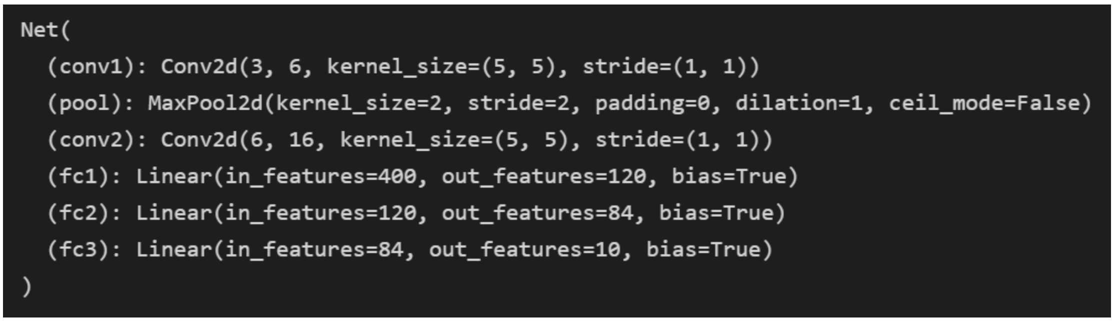
MSBD 5012 Machine Learning Homework 3 Report

Student: XXXXXXXX   ID: 12345678

In this homework, I start from the naïve CNN model in the tutorial and change test different hyperparameters. I summarize the result in the coming chapters. The first part talks about the variation of the number of hidden layers, the second part is about the variation of the number of the filters, the third part shows the variation of the learning rate, the fourth part demonstrates different optimizers, and the last part is about the batch normalization.

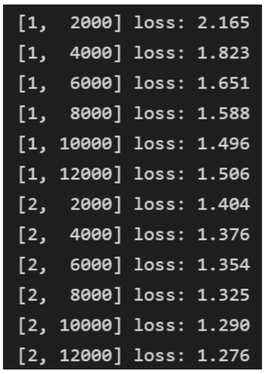
Part One: Variation of the number of hidden layers

We use the naïve model in the tutorial as the first model. We keep using the cross entropy loss, the SGD optimizer in this experiment. The structure of the model can be seen below.

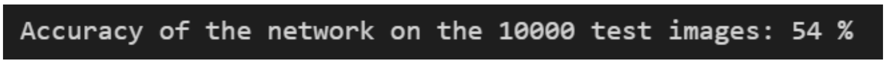


```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

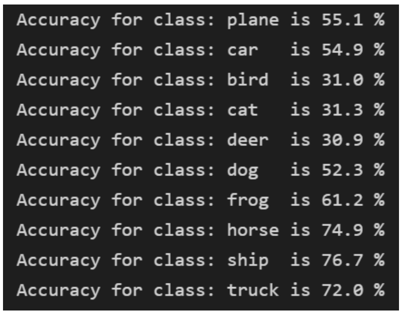
The result of this model is shown below.



```
[1,  2000] loss: 2.165
[1,  4000] loss: 1.823
[1,  6000] loss: 1.651
[1,  8000] loss: 1.588
[1, 10000] loss: 1.496
[1, 12000] loss: 1.506
[2,  2000] loss: 1.404
[2,  4000] loss: 1.376
[2,  6000] loss: 1.354
[2,  8000] loss: 1.325
[2, 10000] loss: 1.290
[2, 12000] loss: 1.276
```

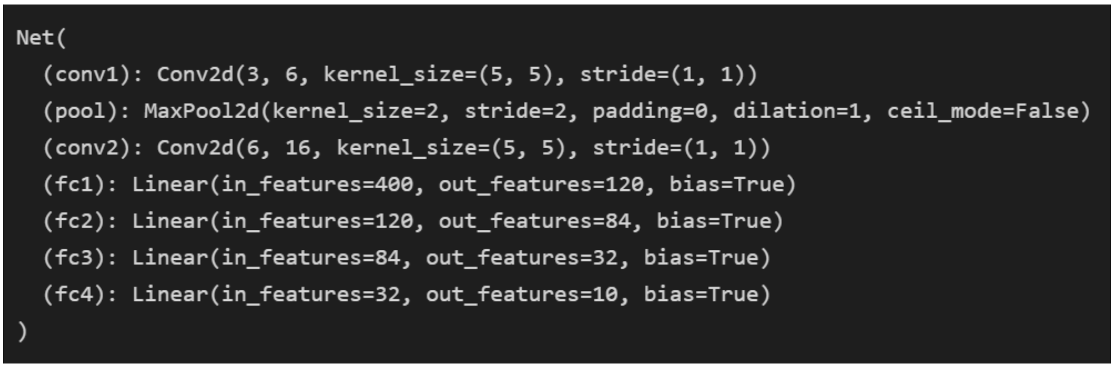


```
Accuracy of the network on the 10000 test images: 54 %
```



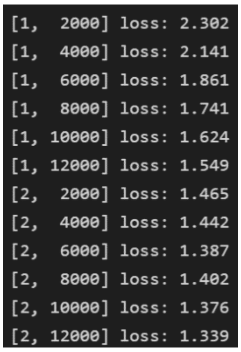
```
Accuracy for class: plane is 55.1 %
Accuracy for class: car   is 54.9 %
Accuracy for class: bird  is 31.0 %
Accuracy for class: cat   is 31.3 %
Accuracy for class: deer  is 30.9 %
Accuracy for class: dog   is 52.3 %
Accuracy for class: frog  is 61.2 %
Accuracy for class: horse is 74.9 %
Accuracy for class: ship  is 76.7 %
Accuracy for class: truck is 72.0 %
```

The second model we add another fully connected layer. The detailed description of the model can be seen below.

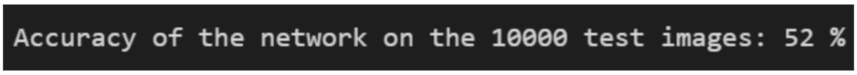


```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=32, bias=True)
  (fc4): Linear(in_features=32, out_features=10, bias=True)
)
```

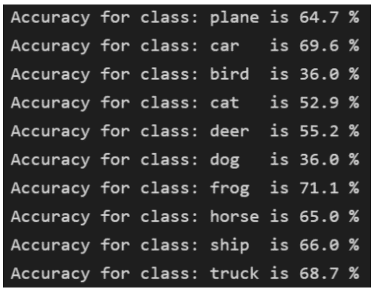
The result of the model is:



```
[1,  2000] loss: 2.302
[1,  4000] loss: 2.141
[1,  6000] loss: 1.861
[1,  8000] loss: 1.741
[1, 10000] loss: 1.624
[1, 12000] loss: 1.549
[2,  2000] loss: 1.465
[2,  4000] loss: 1.442
[2,  6000] loss: 1.387
[2,  8000] loss: 1.402
[2, 10000] loss: 1.376
[2, 12000] loss: 1.339
```



```
Accuracy of the network on the 10000 test images: 52 %
```

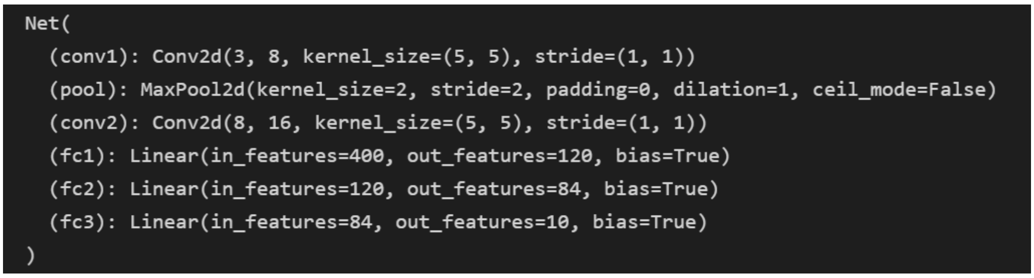


```
Accuracy for class: plane is 64.7 %
Accuracy for class: car   is 69.6 %
Accuracy for class: bird  is 36.0 %
Accuracy for class: cat   is 52.9 %
Accuracy for class: deer  is 55.2 %
Accuracy for class: dog   is 36.0 %
Accuracy for class: frog  is 71.1 %
Accuracy for class: horse is 65.0 %
Accuracy for class: ship  is 66.0 %
Accuracy for class: truck is 68.7 %
```

We can see that after adding one fully connected layer, the accuracy of the network on the 10000 test images dropped. However, the accuracy for some classes increases. Maybe **the fully connected layer increases the model performance in some detailed classes**. However, the model seems to get overfitted.

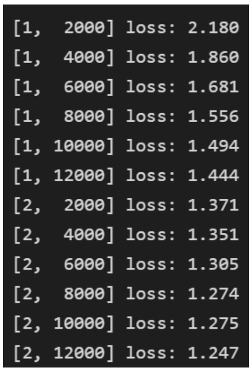
Part Two: Variation of the number of the number of the filters

The number of the filters can be changed by changing the output channels of the Convolutional layer. First, we use (32, 32, 3) → (28, 28, 8) → (14, 14, 8) → (10, 10, 16) → (5, 5, 16).

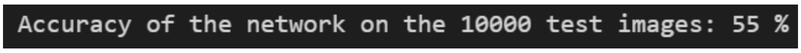


```
Net(
  (conv1): Conv2d(3, 8, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(8, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

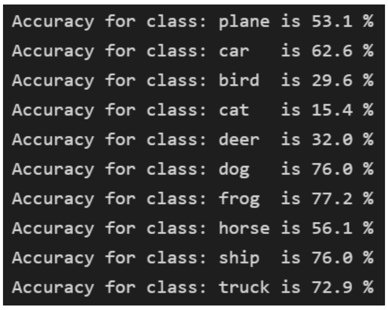
The result is:



```
[1,  2000] loss: 2.180
[1,  4000] loss: 1.860
[1,  6000] loss: 1.681
[1,  8000] loss: 1.556
[1, 10000] loss: 1.494
[1, 12000] loss: 1.444
[2,  2000] loss: 1.371
[2,  4000] loss: 1.351
[2,  6000] loss: 1.305
[2,  8000] loss: 1.274
[2, 10000] loss: 1.275
[2, 12000] loss: 1.247
```

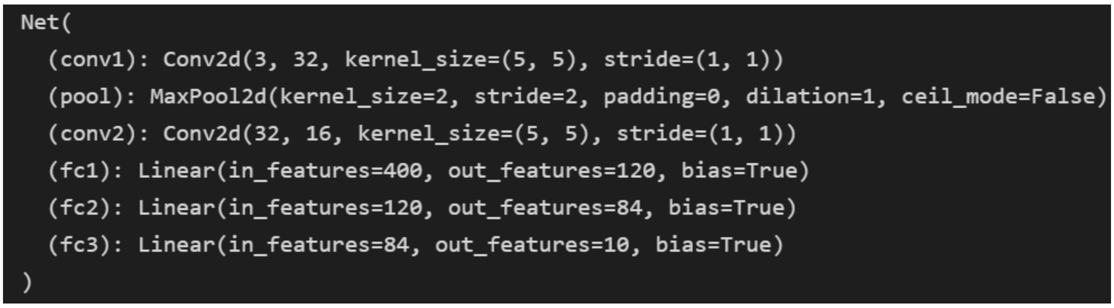


```
Accuracy of the network on the 10000 test images: 55 %
```



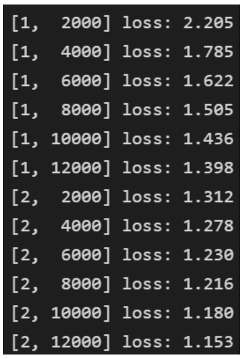
```
Accuracy for class: plane is 53.1 %
Accuracy for class: car   is 62.6 %
Accuracy for class: bird  is 29.6 %
Accuracy for class: cat   is 15.4 %
Accuracy for class: deer  is 32.0 %
Accuracy for class: dog   is 76.0 %
Accuracy for class: frog  is 77.2 %
Accuracy for class: horse is 56.1 %
Accuracy for class: ship  is 76.0 %
Accuracy for class: truck is 72.9 %
```

Then, we use (32, 32, 3) → (28, 28, 32) → (14, 14, 32) → (10, 10, 16) → (5, 5, 16).

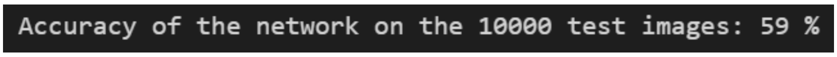


```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

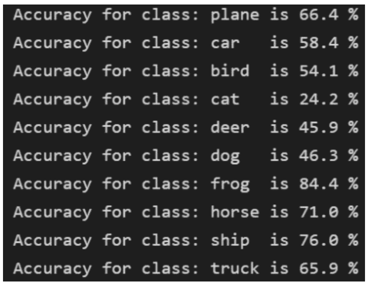
The result is:



```
[1,  2000] loss: 2.205
[1,  4000] loss: 1.785
[1,  6000] loss: 1.622
[1,  8000] loss: 1.505
[1, 10000] loss: 1.436
[1, 12000] loss: 1.398
[2,  2000] loss: 1.312
[2,  4000] loss: 1.278
[2,  6000] loss: 1.230
[2,  8000] loss: 1.216
[2, 10000] loss: 1.180
[2, 12000] loss: 1.153
```



```
Accuracy of the network on the 10000 test images: 59 %
```

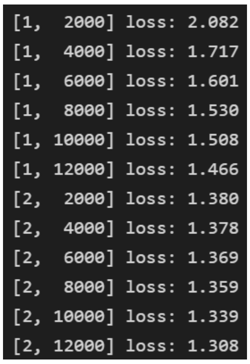


```
Accuracy for class: plane is 66.4 %
Accuracy for class: car   is 58.4 %
Accuracy for class: bird  is 54.1 %
Accuracy for class: cat   is 24.2 %
Accuracy for class: deer  is 45.9 %
Accuracy for class: dog   is 46.3 %
Accuracy for class: frog  is 84.4 %
Accuracy for class: horse is 71.0 %
Accuracy for class: ship  is 76.0 %
Accuracy for class: truck is 65.9 %
```

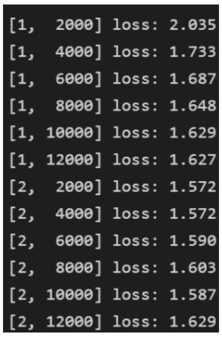
We can find that, after adding more filters, the accuracy rises, this may because **more filters can extract more features of input pictures.**

Part Three: Variation of the learning rate

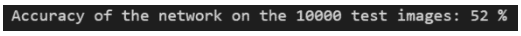
This part we use our naïve model with learning rate 0.002 and 0.005, and momenta are all set 0.9. The result can be compared below. The left three pictures is the result of learning rate 0.002, the right three are the result of learning rate 0.05.



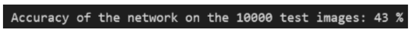
```
[1,  2000] loss: 2.082
[1,  4000] loss: 1.717
[1,  6000] loss: 1.601
[1,  8000] loss: 1.530
[1, 10000] loss: 1.508
[1, 12000] loss: 1.466
[2,  2000] loss: 1.380
[2,  4000] loss: 1.378
[2,  6000] loss: 1.369
[2,  8000] loss: 1.359
[2, 10000] loss: 1.339
[2, 12000] loss: 1.308
```



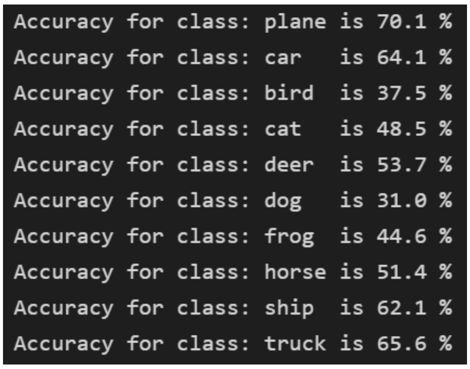
```
[1,  2000] loss: 2.035
[1,  4000] loss: 1.733
[1,  6000] loss: 1.687
[1,  8000] loss: 1.648
[1, 10000] loss: 1.629
[1, 12000] loss: 1.627
[2,  2000] loss: 1.572
[2,  4000] loss: 1.572
[2,  6000] loss: 1.590
[2,  8000] loss: 1.603
[2, 10000] loss: 1.587
[2, 12000] loss: 1.629
```



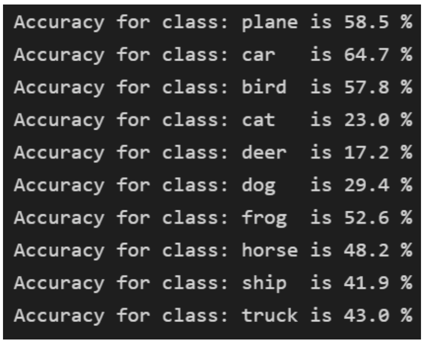
```
Accuracy of the network on the 10000 test images: 52 %
```



```
Accuracy of the network on the 10000 test images: 43 %
```



```
Accuracy for class: plane is 70.1 %
Accuracy for class: car   is 64.1 %
Accuracy for class: bird  is 37.5 %
Accuracy for class: cat   is 48.5 %
Accuracy for class: deer  is 53.7 %
Accuracy for class: dog   is 31.0 %
Accuracy for class: frog  is 44.6 %
Accuracy for class: horse is 51.4 %
Accuracy for class: ship  is 62.1 %
Accuracy for class: truck is 65.6 %
```



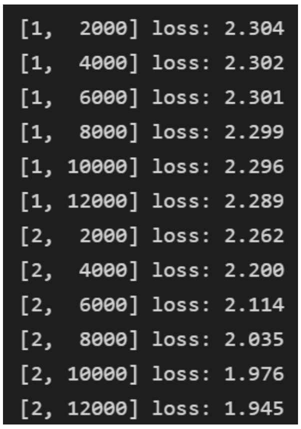
```
Accuracy for class: plane is 58.5 %
Accuracy for class: car   is 64.7 %
Accuracy for class: bird  is 57.8 %
Accuracy for class: cat   is 23.0 %
Accuracy for class: deer  is 17.2 %
Accuracy for class: dog   is 29.4 %
Accuracy for class: frog  is 52.6 %
Accuracy for class: horse is 48.2 %
Accuracy for class: ship  is 41.9 %
Accuracy for class: truck is 43.0 %
```

We can find that after the increase of the learning rate, the accuracy decreases. This is because **a larger learning rate leads to avoiding the local minimum. The gradient fluctuates around the local minimum.**

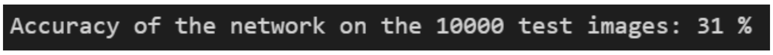
Part Four: Different Optimizers.

We make comparisons of different optimizers including SGD, Adam and RMSProp. The result are as follows:

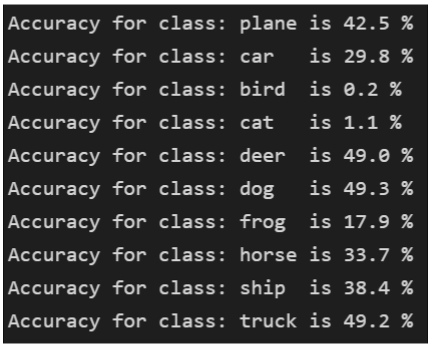
SGD: (Without momentum, learning rate = 0.001)



```
[1,  2000] loss: 2.304
[1,  4000] loss: 2.302
[1,  6000] loss: 2.301
[1,  8000] loss: 2.299
[1, 10000] loss: 2.296
[1, 12000] loss: 2.289
[2,  2000] loss: 2.262
[2,  4000] loss: 2.200
[2,  6000] loss: 2.114
[2,  8000] loss: 2.035
[2, 10000] loss: 1.976
[2, 12000] loss: 1.945
```

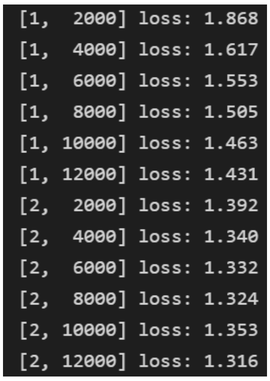


```
Accuracy of the network on the 10000 test images: 31 %
```

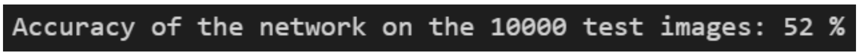


```
Accuracy for class: plane is 42.5 %
Accuracy for class: car   is 29.8 %
Accuracy for class: bird  is 0.2 %
Accuracy for class: cat   is 1.1 %
Accuracy for class: deer  is 49.0 %
Accuracy for class: dog   is 49.3 %
Accuracy for class: frog  is 17.9 %
Accuracy for class: horse is 33.7 %
Accuracy for class: ship  is 38.4 %
Accuracy for class: truck is 49.2 %
```

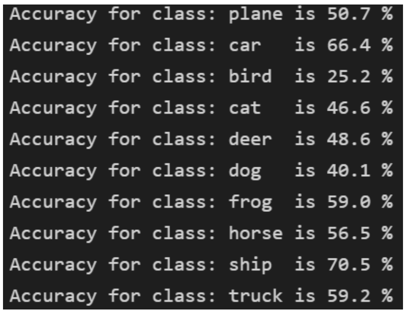
Adam: (learning rate = 0.001)



```
[1,  2000] loss: 1.868
[1,  4000] loss: 1.617
[1,  6000] loss: 1.553
[1,  8000] loss: 1.505
[1, 10000] loss: 1.463
[1, 12000] loss: 1.431
[2,  2000] loss: 1.392
[2,  4000] loss: 1.340
[2,  6000] loss: 1.332
[2,  8000] loss: 1.324
[2, 10000] loss: 1.353
[2, 12000] loss: 1.316
```

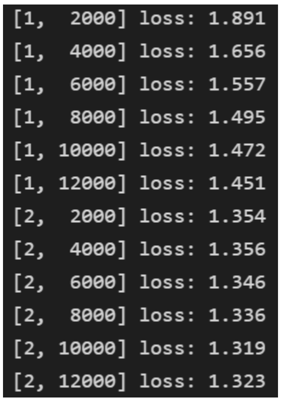


```
Accuracy of the network on the 10000 test images: 52 %
```

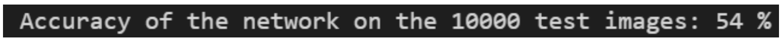


```
Accuracy for class: plane is 50.7 %
Accuracy for class: car   is 66.4 %
Accuracy for class: bird  is 25.2 %
Accuracy for class: cat   is 46.6 %
Accuracy for class: deer  is 48.6 %
Accuracy for class: dog   is 40.1 %
Accuracy for class: frog  is 59.0 %
Accuracy for class: horse is 56.5 %
Accuracy for class: ship  is 70.5 %
Accuracy for class: truck is 59.2 %
```

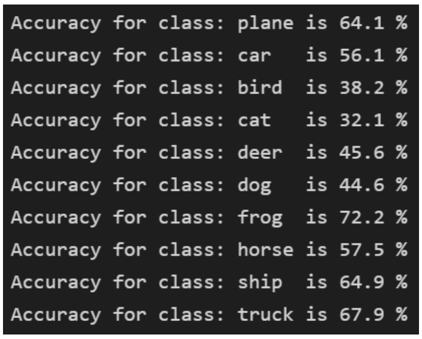
RMSProp: (learning rate = 0.001)



```
[1,  2000] loss: 1.891
[1,  4000] loss: 1.656
[1,  6000] loss: 1.557
[1,  8000] loss: 1.495
[1, 10000] loss: 1.472
[1, 12000] loss: 1.451
[2,  2000] loss: 1.354
[2,  4000] loss: 1.356
[2,  6000] loss: 1.346
[2,  8000] loss: 1.336
[2, 10000] loss: 1.319
[2, 12000] loss: 1.323
```



```
Accuracy of the network on the 10000 test images: 54 %
```

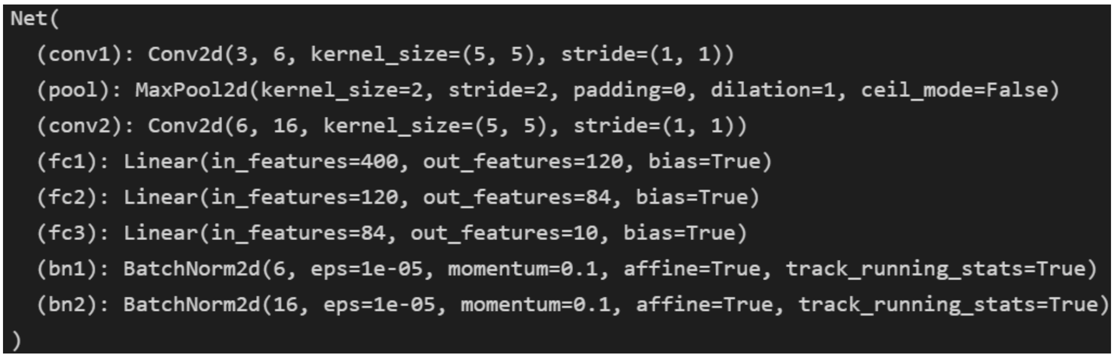


```
Accuracy for class: plane is 64.1 %
Accuracy for class: car   is 56.1 %
Accuracy for class: bird  is 38.2 %
Accuracy for class: cat   is 32.1 %
Accuracy for class: deer  is 45.6 %
Accuracy for class: dog   is 44.6 %
Accuracy for class: frog  is 72.2 %
Accuracy for class: horse is 57.5 %
Accuracy for class: ship  is 64.9 %
Accuracy for class: truck is 67.9 %
```

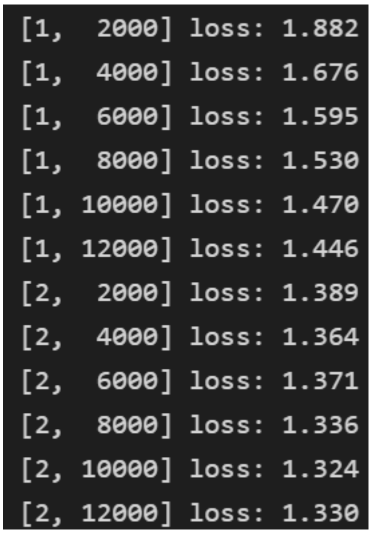
We can find that Adam and RMSProp are better optimizers. Because **both can alleviate the gradient difference. Besides, Adam optimizer uses momentum to use past gradients.** So they are better than SGD.

Part Five: Batch Normalization

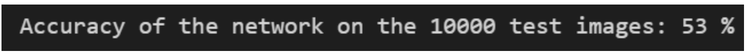
We add batch normalization into our networks and see the results (Using Adam optimizer). The BN layers are added after the convolution layers.



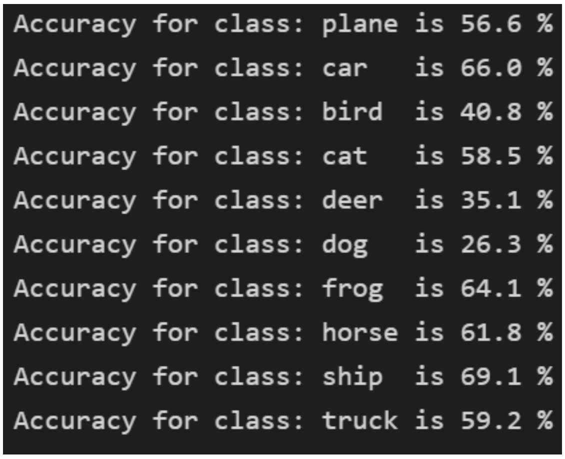
```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
  (bn1): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```



```
[1,  2000] loss: 1.882
[1,  4000] loss: 1.676
[1,  6000] loss: 1.595
[1,  8000] loss: 1.530
[1, 10000] loss: 1.470
[1, 12000] loss: 1.446
[2,  2000] loss: 1.389
[2,  4000] loss: 1.364
[2,  6000] loss: 1.371
[2,  8000] loss: 1.336
[2, 10000] loss: 1.324
[2, 12000] loss: 1.330
```



```
Accuracy of the network on the 10000 test images: 53 %
```



```
Accuracy for class: plane is 56.6 %
Accuracy for class: car   is 66.0 %
Accuracy for class: bird  is 40.8 %
Accuracy for class: cat   is 58.5 %
Accuracy for class: deer  is 35.1 %
Accuracy for class: dog   is 26.3 %
Accuracy for class: frog  is 64.1 %
Accuracy for class: horse is 61.8 %
Accuracy for class: ship  is 69.1 %
Accuracy for class: truck is 59.2 %
```

BN layer should match the dimention of the samples. And compared with the first Adam optimizer, the accuracy improve 1%. I think **this is because the BN layer normalizes the value after some layers, and after that the value will be sensitive to the activation function. It may get a better gradient and thus lead to a better result.**

All of the experiments above are proposed using a PyTorch-GPU version.