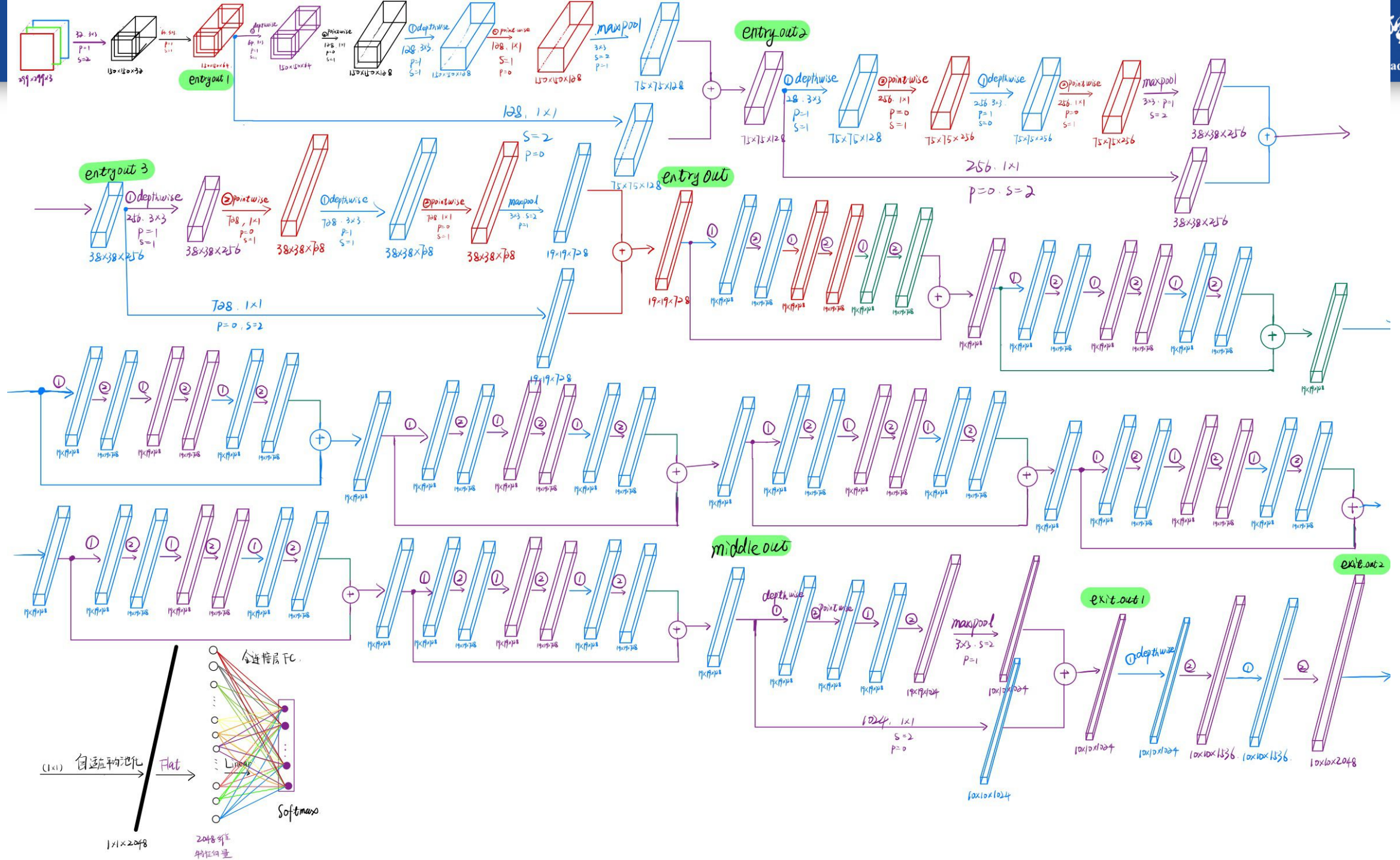
中国科学院大学
University of Chinese Academy of Sciences

# Xception: Deep Learning with Depthwise Separable Convolutions

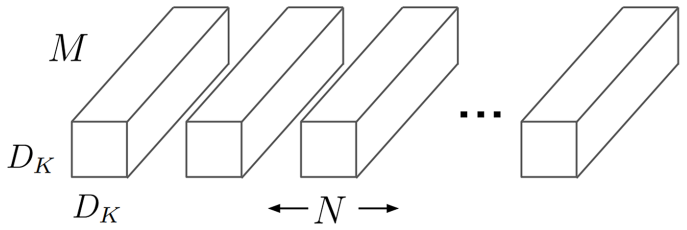
## PyTorch 复现

兰冬雷

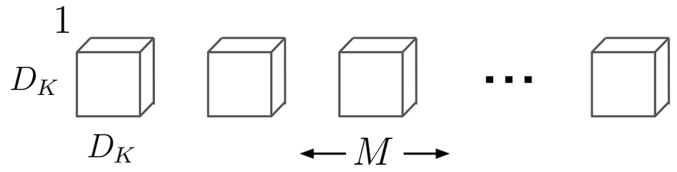
20201213


entry out 1
entry out 2
entry out 3
entry out
middle out
exit out 1
exit out 2
depthwise
pointwise
maxpool
Flat
Linear
Softmax
全连接层 FC
自适应平均池化
2048 维特征向量

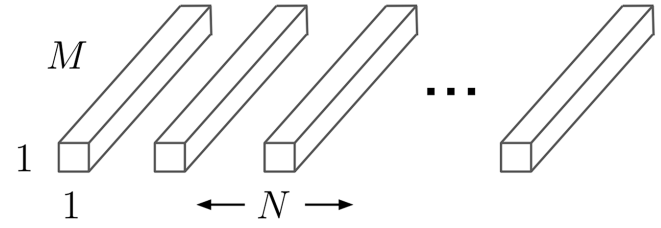
# depthwise separable convolution



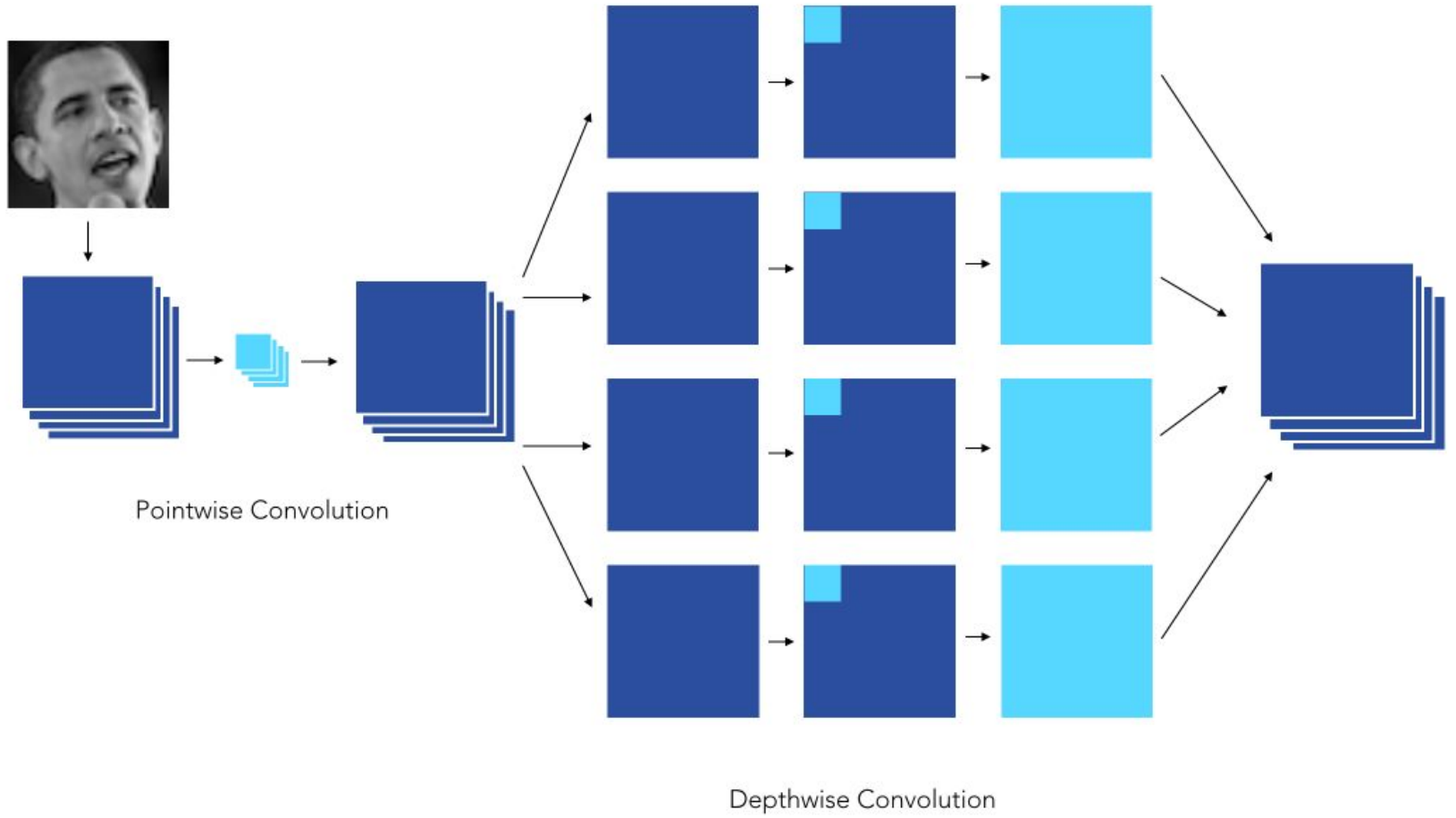

(a) Standard Convolution Filters




(b) Depthwise Convolutional Filters




(c) $1 \times 1$ Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution




depthwise separable convolutions

An artificial intelligent diagnostic system on mobile Android terminals for cholelithiasis by lightweight convolutional neural network

https://maelfabien.github.io/deeplearning/xception/#implementation-of-the-xception

# SeparableConvolution layers



1. 论文作者认为 depthwise 与 pointwise 的先后顺序是不重要的。
2. Xception 使用是先考虑空间相关（ depthwise ）、再考虑通道相关（ pointwise ）。

```
class depthwise_separable_conv(nn.Module):
    def __init__(self, nin, nout, kernel_size, padding, bias=False):
        super(depthwise_separable_conv, self).__init__()
        self.depthwise = nn.Conv2d(nin, nin, kernel_size=kernel_size, padding=padding, groups=nin, bias=bias)
        self.pointwise = nn.Conv2d(nin, nout, kernel_size=1, bias=bias)

    def forward(self, x):
        out = self.depthwise(x)
        out = self.pointwise(out)
        return out
```

# Xception Forward



```python
def forward(self, x):
    # Entry flow
    entry_out1 = self.entry_flow_1(x)
    entry_out2 = self.entry_flow_2(entry_out1) + self.entry_flow_2_residual(entry_out1)
    entry_out3 = self.entry_flow_3(entry_out2) + self.entry_flow_3_residual(entry_out2)
    entry_out = self.entry_flow_4(entry_out3) + self.entry_flow_4_residual(entry_out3)

    # Middle flow
    middle_out = self.middle_flow(entry_out) + entry_out
    for i in range(7):
      middle_out = self.middle_flow(middle_out) + middle_out

    # Exit flow
    exit_out1 = self.exit_flow_1(middle_out) + self.exit_flow_1_residual(middle_out)
    exit_out2 = self.exit_flow_2(exit_out1)

    exit_avg_pool = F.adaptive_avg_pool2d(exit_out2, (1, 1))
    exit_avg_pool_flat = exit_avg_pool.view(exit_avg_pool.size(0), -1)

    output = self.linear(exit_avg_pool_flat)

    return output
```

# 训练



- 数据集：CIFAR-10
- learning_rate = 0.045 ，每两个 EPOCH 后，乘以 0.94 进行学习率衰减
- transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))])
- 图片的 size
- EPOCH
- Batch size
- 从训练集划分出 10% 作为验证集

# 实验结果（1/4）



- 第一次训练，按照论文，输入的 size 为 299×299 ，我只能使用 BatchSize = 8，稍大一点就非常容易 CUDA out of memory。
- EPOCH = 20，跑了差不多有 10 个小时。最后得出 10 个类别的平均 ACC 为：训练（验证集）：87%，测试（测试集）：85%。

```
[20,  3750] loss: 0.2046646
[20,  4000] loss: 0.2186616
[20,  4250] loss: 0.2081674
[20,  4500] loss: 0.2155977
[20,  4750] loss: 0.2380308
[20,  5000] loss: 0.1939209
[20,  5250] loss: 0.2284337
[20,  5500] loss: 0.2230206
[19 epoch] Accuracy of the network on the validation images: 86 %
Finished Training
```

In [17]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 85 %
```

图中 LOSS 未收敛，波动

# 实验结果（2/4）



- 以 CIFAR-10 本来的大小 32×32 输入到 Xception。优化器使用 Adam，EPOCH = 100，Batch size = 128。
- 100 个 EPOCH 训练了 1 个多小时，但是只有 65% 左右的精度。

```
[92 epoch] Accuracy of the network on the validation images: 67 %
[94,   250] loss: 0.6020738
[93 epoch] Accuracy of the network on the validation images: 67 %
[95,   250] loss: 0.6013359
[94 epoch] Accuracy of the network on the validation images: 66 %
[96,   250] loss: 0.5903505
[95 epoch] Accuracy of the network on the validation images: 67 %
[97,   250] loss: 0.5993215
[96 epoch] Accuracy of the network on the validation images: 67 %
[98,   250] loss: 0.5937602
[97 epoch] Accuracy of the network on the validation images: 66 %
[99,   250] loss: 0.5958419
[98 epoch] Accuracy of the network on the validation images: 66 %
[100,   250] loss: 0.5989391
[99 epoch] Accuracy of the network on the validation images: 66 %
Finished Training
```

```
In [11]: import time
         start_time = time.asctime(time.localtime(time.time()))
         print ("Train Finish at: ", start_time)

         Train Finish at:  Thu Dec 10 19:11:32 2020
```

100 EPOCH，LOSS 还在稳定的下降！

- 修改 BatchSize = 256，增加到 200 个 EPOCH 试试。跑了有两个小时，ACC 只有 69% 左右。

# 实验结果（3/4）



- Xception 所能接受的最小输入是：？
- batch size = 256，输入为 128×128：CUDA out of memory.
- batch size = 128，输入为 128×128 ：CUDA out of memory.
- batch size = 64，输入为 128×128 ：It's Word!
- 晚上 22:40 开始跑到第二天早上的 6:40。100 个 EPOCH ACC = 84%。

```
[93,   500] loss: 0.0056548
[92 epoch] Accuracy of the network on the validation images: 84 %
[94,   250] loss: 0.0054904
[94,   500] loss: 0.0045959
[93 epoch] Accuracy of the network on the validation images: 84 %
[95,   250] loss: 0.0057868
[95,   500] loss: 0.0050828
[94 epoch] Accuracy of the network on the validation images: 84 %
[96,   250] loss: 0.0055853
[96,   500] loss: 0.0056603
[95 epoch] Accuracy of the network on the validation images: 84 %
[97,   250] loss: 0.0071294
[97,   500] loss: 0.0057695
[96 epoch] Accuracy of the network on the validation images: 84 %
[98,   250] loss: 0.0047054
[98,   500] loss: 0.0039406
[97 epoch] Accuracy of the network on the validation images: 83 %
[99,   250] loss: 0.0059585
[99,   500] loss: 0.0058971
[98 epoch] Accuracy of the network on the validation images: 84 %
[100,   250] loss: 0.0056651
[100,   500] loss: 0.0034724
[99 epoch] Accuracy of the network on the validation images: 84 %
Finished Training
```

```
[11]: import time
      start_time = time.asctime(time.localtime(time.time()))
      print ("Train Finish at: ", start_time)

      Train Finish at:  Fri Dec 11 06:39:11 2020
```

# 实验结果（4/4）



- 200 EPOCH，大约训练了 16 小时，在最后的 ACC = 87%.

```
[191 epoch] Accuracy of the network on the validation images: 86 %
[193,   250] loss: 0.0019058
[193,   500] loss: 0.0031233
[192 epoch] Accuracy of the network on the validation images: 86 %
[194,   250] loss: 0.0022766
[194,   500] loss: 0.0023194
[193 epoch] Accuracy of the network on the validation images: 86 %
[195,   250] loss: 0.0022375
[195,   500] loss: 0.0021492
[194 epoch] Accuracy of the network on the validation images: 86 %
[196,   250] loss: 0.0034841
[196,   500] loss: 0.0027367
[195 epoch] Accuracy of the network on the validation images: 86 %
[197,   250] loss: 0.0019502
[197,   500] loss: 0.0020695
[196 epoch] Accuracy of the network on the validation images: 86 %
[198,   250] loss: 0.0023790
[198,   500] loss: 0.0026829
[197 epoch] Accuracy of the network on the validation images: 86 %
[199,   250] loss: 0.0029993
[199,   500] loss: 0.0019856
[198 epoch] Accuracy of the network on the validation images: 86 %
[200,   250] loss: 0.0030558
[200,   500] loss: 0.0011285
[199 epoch] Accuracy of the network on the validation images: 87 %
Finished Training
```

```
Accuracy of plane : 89 %
Accuracy of   car : 88 %
Accuracy of  bird : 81 %
Accuracy of   cat : 74 %
Accuracy of  deer : 86 %
Accuracy of   dog : 79 %
Accuracy of  frog : 91 %
Accuracy of horse : 91 %
Accuracy of  ship : 92 %
Accuracy of truck : 88 %
```

我感觉 LOSS 还在降？？？

# 总结



- 本次实验我还没使用预训练的网络、或者在上一步训练的网络的基础之上继续开始。每次都是重新开始（如加载 ACC=84% 的网络，在此基础之上继续进行迭代优化），导致浪费了非常多的时间。

# 谢谢

兰冬雷

20201213