通过微调进行迁移学习

pytorch 一直为我们内置了前面我们讲过的那些著名网络的预训练模型，不需要我们自己去 ImageNet 上训练了，模型都在 `torchvision.models` 里面，比如我们想使用预训练的 50 层 resnet，就可以用 `torchvision.models.resnet50(pretrained=True)` 来得到

下面我们用一个例子来演示一些微调

```
import numpy as np

import torch
from torch import nn
from torch.autograd import Variable
from torch.utils.data import DataLoader

from torchvision import models
from torchvision import transforms as tfs
from torchvision.datasets import ImageFolder
```

首先我们点击下面的链接获得数据集，终端可以使用

`wget https://download.pytorch.org/tutorial/hymenoptera_data.zip`

下载完成之后，我们将其解压放在程序的目录下，这是一个二分类问题，区分蚂蚁和蜜蜂

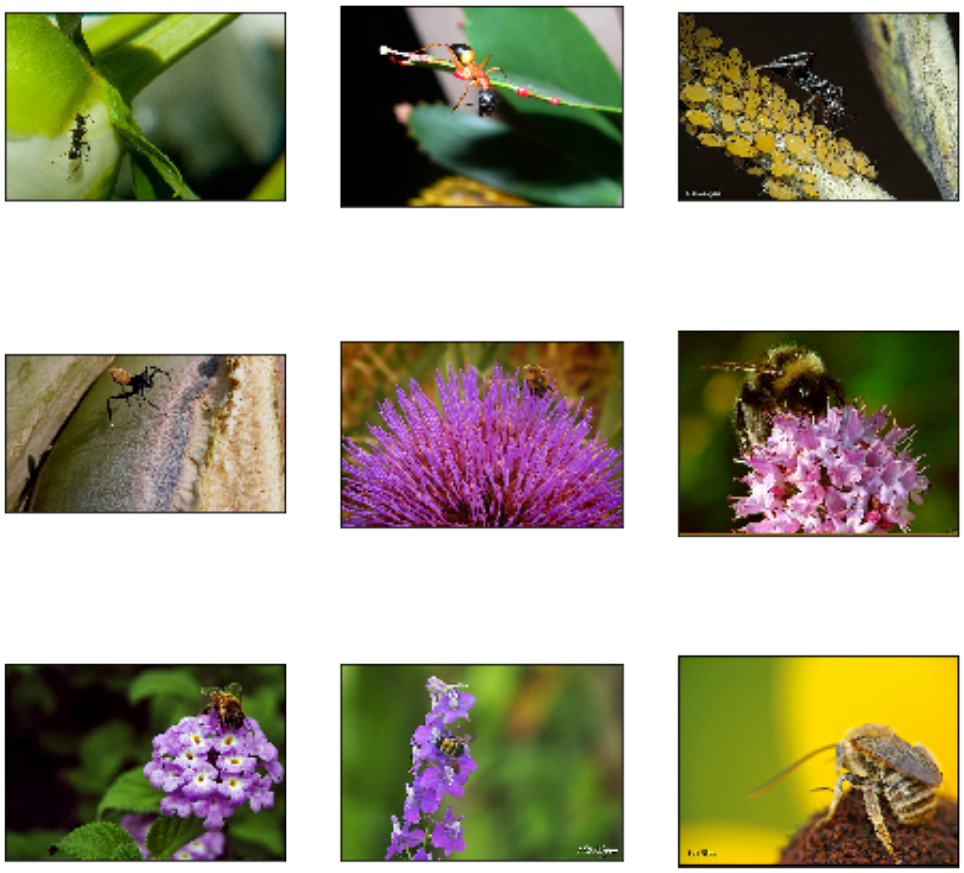
我们可以可视化一下图片，看看你能不能区分出他们来

```
import os
from PIL import Image
import matplotlib.pyplot as plt
%matplotlib inline
```

```
root_path = './hymenoptera_data/train/'
im_list = [os.path.join(root_path, 'ants', i) for i in os.listdir(root_path + 'ants')
[:4]]
im_list += [os.path.join(root_path, 'bees', i) for i in os.listdir(root_path +
'bees')[:5]]

nrows = 3
ncols = 3
figsize = (8, 8)
_, figs = plt.subplots(nrows, ncols, figsize=figsize)
```

```
for i in range(nrows):
    for j in range(ncols):
        figs[i][j].imshow(Image.open(im_list[nrows*i+j]))
        figs[i][j].axes.get_xaxis().set_visible(False)
        figs[i][j].axes.get_yaxis().set_visible(False)
plt.show()
```



```
# 定义数据预处理
train_tf = tfs.Compose([
    tfs.RandomResizedCrop(224),
    tfs.RandomHorizontalFlip(),
    tfs.ToTensor(),
    tfs.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # 使用 ImageNet 的均值和方差
])

valid_tf = tfs.Compose([
    tfs.Resize(256),
    tfs.CenterCrop(224),
    tfs.ToTensor(),
```

```
    tfs.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```
# 使用 ImageFolder 定义数据集
train_set = ImageFolder('./hymenoptera_data/train/', train_tf)
valid_set = ImageFolder('./hymenoptera_data/val/', valid_tf)
# 使用 DataLoader 定义迭代器
train_data = DataLoader(train_set, 64, True, num_workers=4)
valid_data = DataLoader(valid_set, 128, False, num_workers=4)
```

```
# 使用预训练的模型
net = models.resnet50(pretrained=True)
print(net)
```

```
ResNet(
  (conv1): Conv2d (3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), dilation=
(1, 1))
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d (64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d (64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d (256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d (256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d (256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d (256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d (512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d (512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (3): Bottleneck(
      (conv1): Conv2d (512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d (512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d (512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d (1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d (1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (3): Bottleneck(
      (conv1): Conv2d (1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (4): Bottleneck(
      (conv1): Conv2d (1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (5): Bottleneck(
      (conv1): Conv2d (1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d (1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d (1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
```

```
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d (2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d (2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (conv2): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (conv3): Conv2d (512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True)
      (relu): ReLU(inplace)
    )
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0, ceil_mode=False,
count_include_pad=True)
  (fc): Linear(in_features=2048, out_features=1000)
)
```

```
# 打出第一层的权重
print(net.conv1.weight)
```

```
Parameter containing:
(0 ,0 ,.,.) =
  1.3335e-02  1.4664e-02 -1.5351e-02  ...  -4.0896e-02 -4.3034e-02 -7.0755e-02
  4.1205e-03  5.8477e-03  1.4948e-02  ...   2.2060e-03 -2.0912e-02 -3.8517e-02
  2.2331e-02  2.3595e-02  1.6120e-02  ...   1.0281e-01  6.2641e-02  5.1977e-02
                 ...                  ⋱                    ...
 -9.0349e-04  2.7767e-02 -1.0105e-02  ...  -1.2722e-01 -7.6604e-02  7.8453e-03
  3.5894e-03  4.8006e-02  6.2051e-02  ...   2.4267e-02 -3.3662e-02 -1.5709e-02
 -8.0029e-02 -3.2238e-02 -1.7808e-02  ...   3.5359e-02  2.2439e-02  1.7077e-03

(0 ,1 ,.,.) =
 -1.8452e-02  1.1415e-02  2.3850e-02  ...   5.3736e-02  4.4022e-02 -9.4675e-03
```

```
 -7.7273e-03  1.8890e-02  6.7981e-02  ...   1.5956e-01  1.4606e-01  1.1999e-01
 -4.6013e-02 -7.6075e-02 -8.9648e-02  ...   1.2108e-01  1.6705e-01  1.7619e-01
                 ...                   ⋱                   ...
  2.8818e-02  1.3665e-02 -8.3825e-02  ...  -3.8081e-01 -3.0414e-01 -1.3966e-01
  8.2868e-02  1.3864e-01  1.5241e-01  ...  -5.1232e-03 -1.2435e-01 -1.2967e-01
 -7.2789e-03  7.7021e-02  1.3999e-01  ...   1.8427e-01  1.1144e-01  2.3438e-02

(0 ,2 ,.,.) =
 -1.8311e-02 -5.6424e-03  8.7224e-03  ...   2.5775e-02  2.6431e-02 -3.9914e-03
 -1.0098e-02  4.1615e-03  4.9730e-02  ...   1.2447e-01  1.1950e-01  1.1198e-01
 -6.3478e-02 -1.0146e-01 -9.8343e-02  ...   1.0630e-01  1.3982e-01  1.4942e-01
                 ...                   ⋱                   ...
  2.5810e-02  1.0501e-02 -7.4578e-02  ...  -3.1385e-01 -2.5495e-01 -1.2276e-01
  7.3049e-02  1.1170e-01  1.3093e-01  ...  -6.4584e-03 -1.2548e-01 -1.2446e-01
 -6.4210e-03  6.6393e-02  1.2177e-01  ...   1.9075e-01  1.1415e-01  2.3337e-02
     ⋮

(1 ,0 ,.,.) =
  6.8609e-02  3.7955e-02  5.3564e-02  ...   2.6891e-02  4.8369e-02  6.3264e-02
  6.1844e-02  1.8407e-02  2.2672e-02  ...  -4.8800e-02 -2.2130e-02 -5.7287e-03
  5.6553e-02  1.4883e-02 -6.9185e-03  ...  -1.2919e-01 -9.5042e-02 -5.8671e-02
                 ...                   ⋱                   ...
  2.3802e-02 -5.2273e-02 -1.1277e-01  ...  -2.5591e-01 -2.4049e-01 -2.0315e-01
  5.6221e-02 -2.1824e-02 -5.9207e-02  ...  -2.3806e-01 -1.9836e-01 -1.6578e-01
  5.9635e-02  3.7172e-03 -4.8716e-02  ...  -1.6098e-01 -1.4336e-01 -1.0251e-01

(1 ,1 ,.,.) =
 -9.9041e-02 -7.2129e-02 -7.2748e-02  ...  -3.6232e-02 -8.1876e-02 -8.8119e-02
 -7.0621e-02 -3.9200e-02 -1.0514e-02  ...   5.9519e-02  2.5891e-02 -1.3287e-02
 -9.3963e-02 -2.5318e-02  3.0725e-02  ...   2.0738e-01  1.6162e-01  8.6865e-02
                 ...                   ⋱                   ...
 -3.9978e-02  6.4148e-02  1.6941e-01  ...   4.5814e-01  3.7839e-01  2.5870e-01
 -6.4985e-02  1.3637e-02  1.3008e-01  ...   3.6661e-01  3.2009e-01  2.0442e-01
 -1.0433e-01 -2.7216e-02  4.1199e-02  ...   2.2797e-01  1.8622e-01  1.1854e-01

(1 ,2 ,.,.) =
  4.2717e-02  4.7269e-02  1.7798e-02  ...   3.4693e-02  2.8595e-02  4.3173e-02
  3.6308e-02  3.3459e-02 -1.2449e-03  ...   4.2910e-03  4.6513e-03  2.4962e-02
  2.3541e-02  1.6826e-02 -1.5325e-02  ...  -7.2865e-02 -7.2030e-02 -2.3747e-02
                 ...                   ⋱                   ...
  2.9623e-02  1.8176e-03 -7.9559e-02  ...  -1.8138e-01 -1.6950e-01 -6.1192e-02
  3.5233e-02 -8.5295e-03 -5.4253e-02  ...  -1.5068e-01 -1.2747e-01 -4.5340e-02
  5.4786e-02  3.9805e-02 -1.0473e-02  ...  -4.2332e-02 -5.2220e-02  5.1277e-04
     ⋮

(2 ,0 ,.,.) =
```

```
  7.3052e-03  1.5686e-02 -2.4423e-04  ...  -3.2539e-02 -3.1297e-02 -3.0249e-02
  8.6377e-03  2.4622e-02  1.1249e-02  ...  -1.0645e-02 -1.5026e-02 -3.4091e-02
  7.7960e-03  3.1337e-02  2.5109e-02  ...   1.2514e-02 -5.7764e-04 -5.3505e-03
                 ...                  ⋱                  ...
 -1.0597e-03  1.3395e-02  1.1477e-03  ...   3.6588e-02  4.6791e-02  7.5218e-02
 -1.5273e-02  8.1007e-03 -7.3382e-04  ...   3.7322e-03  1.9803e-02  7.4966e-02
 -1.9642e-02 -1.8543e-03 -6.9084e-03  ...  -3.2996e-03  2.8972e-02  8.7559e-02

(2 ,1 ,.,.) =
  1.2775e-02  8.8500e-03  2.3261e-03  ...   1.3541e-02  3.6049e-02  4.8399e-02
  1.4153e-02  1.1684e-02 -2.9641e-03  ...   2.7169e-02  5.1851e-02  3.8690e-02
  3.1984e-03  5.0850e-03 -9.3271e-03  ...   4.1137e-02  5.4548e-02  4.5094e-02
                 ...                  ⋱                  ...
 -1.7035e-02 -4.2867e-02 -8.4639e-02  ...  -4.9248e-02 -2.3901e-02  2.6232e-03
 -2.8155e-03 -8.8370e-03 -4.8081e-02  ...  -8.6432e-02 -7.1647e-02 -2.3398e-02
  2.8788e-02  2.3000e-02 -4.6879e-03  ...  -5.0775e-02 -3.9457e-02 -7.1134e-03

(2 ,2 ,.,.) =
  9.3511e-03 -3.6039e-02 -3.1732e-02  ...   3.4844e-03  4.3197e-02  4.5083e-02
 -1.1875e-02 -5.8264e-02 -6.3004e-02  ...  -1.7674e-02  1.5884e-02 -7.6554e-03
  1.4215e-02 -3.3805e-02 -5.0096e-02  ...  -2.1366e-03  6.0525e-03 -8.0126e-03
                 ...                  ⋱                  ...
  8.0828e-03 -4.9916e-02 -9.0253e-02  ...  -8.7565e-02 -9.9468e-02 -1.0924e-01
  1.8781e-02 -1.4367e-02 -4.6708e-02  ...  -1.1187e-01 -1.4514e-01 -1.4444e-01
  5.7803e-02  1.7252e-02 -5.7767e-03  ...  -7.4414e-02 -1.1313e-01 -1.2434e-01
...
     ⋮

(61,0 ,.,.) =
  1.5639e-02  2.0411e-02  3.5717e-02  ...   7.1576e-03  4.5493e-02 -7.6534e-03
 -2.1845e-02  5.6129e-02  6.0663e-02  ...  -5.6907e-02  8.5169e-02 -2.6663e-02
 -4.7579e-02  1.0689e-01  7.5667e-02  ...  -8.8005e-02  1.5159e-01 -3.2074e-02
                 ...                  ⋱                  ...
  1.4774e-03  1.2554e-01 -4.9284e-02  ...   5.9099e-02  1.5756e-01 -4.1167e-02
  5.0183e-03  6.2567e-02 -5.8753e-02  ...   4.7302e-02  8.2366e-02 -4.6016e-02
  1.6201e-02  5.4047e-03 -6.5849e-02  ...   5.1001e-02  4.6469e-02 -1.6759e-02

(61,1 ,.,.) =
  1.0200e-02  4.4592e-02  1.5321e-03  ...  -6.6545e-02  4.5731e-02  4.8529e-02
  4.5088e-03  1.2873e-01  3.0149e-02  ...  -1.4642e-01  1.4743e-01  8.3739e-02
  5.6123e-03  2.1745e-01  3.0934e-02  ...  -1.8033e-01  2.6112e-01  9.4205e-02
                 ...                  ⋱                  ...
  4.6450e-02  1.9461e-01 -1.3893e-01  ...   4.3828e-02  2.8577e-01  5.1682e-02
  2.5445e-02  9.4402e-02 -1.3597e-01  ...   5.5534e-02  1.5707e-01 -1.3938e-02
  1.6235e-02  2.9242e-02 -1.0514e-01  ...   7.2561e-02  8.2445e-02 -1.0449e-02
```

```
(61,2 ,.,.) =
  7.6312e-04  2.2041e-02  1.2029e-02  ...  -3.9314e-02  7.8765e-03  1.1339e-02
 -2.6744e-02  7.6046e-02  6.5150e-02  ...  -5.3509e-02  7.7586e-02  1.5460e-02
 -4.3151e-02  1.2443e-01  7.8582e-02  ...  -7.2363e-02  1.3668e-01 -3.7664e-03
                 ...                  ⋱                    ...
 -4.8974e-03  1.2171e-01 -4.9993e-02  ...   4.0816e-02  1.2784e-01 -2.7264e-02
 -5.7832e-03  6.6111e-02 -4.9786e-02  ...   3.8487e-02  6.9511e-02 -4.3904e-02
  3.3817e-03  3.2986e-02 -4.2698e-02  ...   4.3439e-02  3.3467e-02 -2.9978e-02
     ⋮

(62,0 ,.,.) =
  4.5842e-02  5.2311e-02  4.4927e-02  ...  -2.9410e-02  4.5506e-03  1.4374e-02
  5.2478e-02  5.1215e-02  4.7960e-02  ...  -1.1274e-01 -8.2460e-02 -2.5520e-02
  9.0280e-02  7.7345e-02  6.7258e-02  ...  -2.1452e-01 -1.1146e-01 -1.7177e-02
                 ...                  ⋱                    ...
  3.2732e-02  7.1492e-05 -1.5341e-01  ...  -2.5537e-01 -1.1447e-01  4.2084e-02
  1.9229e-02 -2.7870e-02 -1.4789e-01  ...  -2.4871e-01 -3.0199e-02  8.2352e-02
 -6.6101e-03 -5.7227e-02 -1.5145e-01  ...  -1.8681e-01  9.7780e-03  9.8369e-02

(62,1 ,.,.) =
 -3.1084e-03  1.2002e-02  1.6918e-02  ...   4.7799e-03  4.7867e-03 -8.5661e-04
  1.9549e-02  1.5186e-02  3.4710e-02  ...  -4.2644e-03 -1.3299e-02 -5.1674e-03
  2.0221e-02  1.1181e-02  5.0552e-02  ...  -6.5437e-02 -6.4139e-03  1.8563e-02
                 ...                  ⋱                    ...
 -1.1129e-02  1.0629e-02 -4.2425e-02  ...  -4.8182e-02 -2.8246e-02  4.6424e-02
  7.5653e-03  1.1309e-02 -1.3535e-02  ...  -6.6639e-02  1.0465e-02  3.6264e-02
  9.8492e-03  6.3770e-03 -1.5771e-04  ...  -3.0868e-02  2.6524e-02  3.0090e-02

(62,2 ,.,.) =
 -4.2418e-02 -2.3199e-02 -2.2478e-02  ...   1.4661e-02  1.2153e-02 -2.1124e-02
 -2.9424e-02 -2.5660e-02  1.4858e-03  ...   4.2752e-02  2.1531e-02 -5.6081e-03
 -3.9465e-02 -4.5056e-02  1.4975e-02  ...   3.0639e-02  4.1439e-02 -7.7498e-03
                 ...                  ⋱                    ...
 -3.1252e-02  7.7716e-03  1.5953e-02  ...   6.3187e-02  4.8559e-03 -2.9812e-02
 -1.2930e-02  1.6789e-02  4.8435e-02  ...   4.6301e-02  2.6589e-02 -4.1627e-02
 -1.7138e-02  7.2573e-03  4.8229e-02  ...   3.5378e-02  1.4400e-02 -4.7261e-02
     ⋮

(63,0 ,.,.) =
  1.9836e-02 -2.8316e-02  6.1643e-02  ...  -5.2705e-02  9.8331e-03  4.9678e-03
 -4.6186e-02  5.3722e-02 -2.6572e-02  ...   1.9153e-02 -2.2031e-02 -1.3209e-02
  5.2939e-02 -4.4223e-02 -3.7457e-02  ...  -2.1368e-01 -2.1620e-03  3.4433e-02
                 ...                  ⋱                    ...
 -3.5689e-02  9.3666e-02  1.1909e-01  ...   5.2430e-01 -4.5373e-02 -5.9195e-02
 -1.1405e-02  1.0079e-01 -1.9849e-01  ...   2.3351e-01 -2.6296e-01  1.2434e-01
 -1.4071e-02  5.2204e-03 -7.5740e-02  ...  -5.6693e-02  3.3713e-02 -3.4239e-02
```

```
(63,1 ,.,.) =
  1.8626e-02 -3.9080e-02  4.1489e-02  ...  -4.2309e-02  2.7354e-02 -5.3296e-03
 -4.5311e-02  6.2526e-02 -1.8939e-02  ...   2.9117e-02  3.0331e-03 -1.8276e-03
  3.6988e-02 -6.5929e-02 -2.1645e-02  ...  -2.6028e-01  5.6034e-02  3.7797e-02
                 ...                   ⋱                   ...
 -3.4665e-02  1.2999e-01  1.2347e-01  ...   6.5182e-01 -1.6673e-02 -1.2347e-01
 -1.9596e-02  1.2236e-01 -2.4884e-01  ...   3.3043e-01 -3.2030e-01  9.4385e-02
  2.0938e-03  2.5218e-02 -8.2988e-02  ...  -6.9019e-02  2.0379e-02 -1.5437e-02

(63,2 ,.,.) =
  1.3068e-02 -3.7082e-02  1.9962e-02  ...  -4.6411e-02  2.6141e-02  2.5953e-03
 -3.3303e-02  7.5309e-02 -2.8833e-02  ...   4.7499e-02 -7.2355e-03 -1.5955e-02
  6.1195e-02 -4.4915e-02 -1.3650e-01  ...  -1.3014e-01  6.3804e-03  1.3756e-02
                 ...                   ⋱                   ...
 -1.5692e-02  8.1412e-03  1.4574e-01  ...   4.6457e-01 -1.6762e-01 -2.8489e-02
 -4.5328e-02  3.7344e-02 -1.2535e-01  ...   1.0561e-01 -2.7718e-01  1.6692e-01
  1.0456e-02  1.9020e-02 -1.5351e-02  ...  -1.1350e-01  6.7615e-02 -6.7650e-03
[torch.FloatTensor of size 64x3x7x7]
```

```
# 将最后的全连接层改成二分类
net.fc = nn.Linear(2048, 2)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=1e-2, weight_decay=1e-4)
```

```
from utils import train
train(net, train_data, valid_data, 20, optimizer, criterion)
```

```
Epoch 0. Train Loss: 0.627600, Train Acc: 0.624399, Valid Loss: 0.713161, Valid Acc:
0.483750, Time 00:00:02
Epoch 1. Train Loss: 0.534958, Train Acc: 0.663161, Valid Loss: 0.612412, Valid Acc:
0.650156, Time 00:00:03
Epoch 2. Train Loss: 0.354584, Train Acc: 0.899639, Valid Loss: 0.220221, Valid Acc:
0.940937, Time 00:00:03
Epoch 3. Train Loss: 0.296389, Train Acc: 0.927885, Valid Loss: 0.207770, Valid Acc:
0.956562, Time 00:00:03
Epoch 4. Train Loss: 0.251669, Train Acc: 0.941406, Valid Loss: 0.199639, Valid Acc:
0.960469, Time 00:00:03
```

```
Epoch 5. Train Loss: 0.200544, Train Acc: 0.963041, Valid Loss: 0.185101, Valid Acc:
0.960469, Time 00:00:03
Epoch 6. Train Loss: 0.228474, Train Acc: 0.930288, Valid Loss: 0.339473, Valid Acc:
0.892656, Time 00:00:03
Epoch 7. Train Loss: 0.207282, Train Acc: 0.921274, Valid Loss: 0.149510, Valid Acc:
0.940937, Time 00:00:03
Epoch 8. Train Loss: 0.167375, Train Acc: 0.960036, Valid Loss: 0.165911, Valid Acc:
0.956562, Time 00:00:03
Epoch 9. Train Loss: 0.139687, Train Acc: 0.980469, Valid Loss: 0.167267, Valid Acc:
0.956562, Time 00:00:03
Epoch 10. Train Loss: 0.148986, Train Acc: 0.962139, Valid Loss: 0.135842, Valid Acc:
0.956562, Time 00:00:03
Epoch 11. Train Loss: 0.134802, Train Acc: 0.954327, Valid Loss: 0.153255, Valid Acc:
0.956562, Time 00:00:03
Epoch 12. Train Loss: 0.160324, Train Acc: 0.947416, Valid Loss: 0.135328, Valid Acc:
0.956562, Time 00:00:03
Epoch 13. Train Loss: 0.115013, Train Acc: 0.964844, Valid Loss: 0.171028, Valid Acc:
0.956562, Time 00:00:03
Epoch 14. Train Loss: 0.096694, Train Acc: 0.979567, Valid Loss: 0.153986, Valid Acc:
0.952656, Time 00:00:03
Epoch 15. Train Loss: 0.070175, Train Acc: 0.992188, Valid Loss: 0.146385, Valid Acc:
0.956562, Time 00:00:03
Epoch 16. Train Loss: 0.096758, Train Acc: 0.975661, Valid Loss: 0.153641, Valid Acc:
0.956562, Time 00:00:03
Epoch 17. Train Loss: 0.080756, Train Acc: 0.988281, Valid Loss: 0.145079, Valid Acc:
0.952656, Time 00:00:03
Epoch 18. Train Loss: 0.095881, Train Acc: 0.975661, Valid Loss: 0.149842, Valid Acc:
0.956562, Time 00:00:03
Epoch 19. Train Loss: 0.087257, Train Acc: 0.979567, Valid Loss: 0.157058, Valid Acc:
0.956562, Time 00:00:03
```

下面我们来可视化预测的结果

```
net = net.eval() # 将网络改为预测模式
```

读一张蚂蚁的图片

```
im1 = Image.open('./hymenoptera_data/train/ants/0013035.jpg')
im1
```


```
im = valid_tf(im1) # 做数据预处理
out = net(Variable(im.unsqueeze(0)).cuda())
pred_label = out.max(1)[1].data[0]
print('predict label: {}'.format(train_set.classes[pred_label]))
```

```
predict label: ants
```

可以看到预测的结果是对的

小练习：看看上面的网络预测过程，多尝试几张图片进行预测

```
# 保持前面的卷积层参数不变
net = models.resnet50(pretrained=True)
for param in net.parameters():
    param.requires_grad = False # 将模型的参数设置为不求梯度
net.fc = nn.Linear(2048, 2)

optimizer = torch.optim.SGD(net.fc.parameters(), lr=1e-2, weight_decay=1e-4)
```

```
train(net, train_data, valid_data, 20, optimizer, criterion)
```

```
Epoch 0. Train Loss: 0.619283, Train Acc: 0.677584, Valid Loss: 0.365831, Valid Acc:
0.808594, Time 00:00:02
Epoch 1. Train Loss: 0.476328, Train Acc: 0.835337, Valid Loss: 0.454169, Valid Acc:
0.808750, Time 00:00:02
Epoch 2. Train Loss: 0.423460, Train Acc: 0.863582, Valid Loss: 0.755112, Valid Acc:
0.538906, Time 00:00:02
Epoch 3. Train Loss: 0.382857, Train Acc: 0.856671, Valid Loss: 0.409471, Valid Acc:
0.788750, Time 00:00:02
Epoch 4. Train Loss: 0.337946, Train Acc: 0.913462, Valid Loss: 0.449115, Valid Acc:
0.784844, Time 00:00:02
Epoch 5. Train Loss: 0.341769, Train Acc: 0.880409, Valid Loss: 0.559394, Valid Acc:
0.705313, Time 00:00:02
Epoch 6. Train Loss: 0.326585, Train Acc: 0.851562, Valid Loss: 0.222567, Valid Acc:
0.933125, Time 00:00:02
Epoch 7. Train Loss: 0.263430, Train Acc: 0.924880, Valid Loss: 0.346770, Valid Acc:
0.808750, Time 00:00:02
Epoch 8. Train Loss: 0.308510, Train Acc: 0.876202, Valid Loss: 0.256893, Valid Acc:
0.940469, Time 00:00:02
Epoch 9. Train Loss: 0.238210, Train Acc: 0.924880, Valid Loss: 0.228409, Valid Acc:
0.960469, Time 00:00:02
Epoch 10. Train Loss: 0.237410, Train Acc: 0.944411, Valid Loss: 0.382277, Valid Acc:
0.784844, Time 00:00:02
Epoch 11. Train Loss: 0.200215, Train Acc: 0.956130, Valid Loss: 0.201788, Valid Acc:
0.952656, Time 00:00:02
Epoch 12. Train Loss: 0.205542, Train Acc: 0.947416, Valid Loss: 0.215446, Valid Acc:
0.960469, Time 00:00:02
Epoch 13. Train Loss: 0.223020, Train Acc: 0.930288, Valid Loss: 0.299366, Valid Acc:
0.876563, Time 00:00:02
Epoch 14. Train Loss: 0.188020, Train Acc: 0.944411, Valid Loss: 0.228825, Valid Acc:
0.940469, Time 00:00:02
Epoch 15. Train Loss: 0.182369, Train Acc: 0.963041, Valid Loss: 0.177187, Valid Acc:
0.948750, Time 00:00:02
Epoch 16. Train Loss: 0.180508, Train Acc: 0.953425, Valid Loss: 0.233851, Valid Acc:
0.936562, Time 00:00:02
Epoch 17. Train Loss: 0.165518, Train Acc: 0.963942, Valid Loss: 0.186768, Valid Acc:
0.956562, Time 00:00:02
Epoch 18. Train Loss: 0.214546, Train Acc: 0.938702, Valid Loss: 0.229794, Valid Acc:
0.936562, Time 00:00:02
Epoch 19. Train Loss: 0.185106, Train Acc: 0.946815, Valid Loss: 0.295074, Valid Acc:
0.856563, Time 00:00:02
```

可以看到只训练验证集的准确率也可以达到比较高，但是 loss 跳动比较大，因为更新的参数太少了，只有全连接层的参数

```
# 不使用预训练的模型
net = models.resnet50()
net.fc = nn.Linear(2048, 2)

optimizer = torch.optim.SGD(net.parameters(), lr=1e-2, weight_decay=1e-4)
```

```
# 打出第一层的权重
print(net.conv1.weight)
```

```
Parameter containing:
(0 ,0 ,.,.) =
1.00000e-02 *
 -0.0567 -4.1588  0.0105  ...   -0.1119 -0.2745  0.5707
 -2.5873  2.2117  0.8934  ...   -0.1620 -1.5486 -1.9326
  2.9815  6.1294 -1.2279  ...   -0.3515 -4.8073 -2.1427
           ...              ⋱              ...
 -4.3577  0.1747  3.2660  ...    5.8964 -0.7102 -0.5296
  1.7968 -0.6364  1.5020  ...    2.5407 -1.6381  1.6288
  0.9442  0.8170 -1.2126  ...   -2.0201 -1.0320  2.1268

(0 ,1 ,.,.) =
1.00000e-02 *
  1.3832 -2.4412  0.9790  ...    0.3005  0.9213 -2.4297
 -0.3945 -2.3060 -0.9967  ...   -1.9593 -1.7380 -3.2350
 -1.9276 -1.3089  4.8536  ...    3.2571  4.6997  3.3420
           ...              ⋱              ...
  0.0691 -2.5891 -3.4464  ...    2.8905 -2.2376  3.5079
 -4.7411 -0.2467 -4.0959  ...    0.3010 -2.1790 -2.0739
 -4.8234 -2.4718  1.8820  ...   -2.3377 -3.3540  2.8906

(0 ,2 ,.,.) =
1.00000e-02 *
  2.6300  4.1233 -3.6107  ...    2.6179  4.1958 -1.1002
  1.1691 -2.1549 -1.4611  ...    0.4970 -2.0865  2.2103
 -2.4360 -3.1890  1.1114  ...    2.0006  1.0416 -1.5792
           ...              ⋱              ...
 -0.7292 -0.0186  1.5743  ...    1.3281  0.1535 -4.6561
  0.5399 -1.7713  1.7584  ...   -0.7518  3.1182 -5.6182
 -2.5439  0.3842  1.0441  ...    5.0149 -3.8025 -2.0510
     ⋮
```

```
(1 ,0 ,.,.) =
1.00000e-02 *
  1.2037  3.0141 -2.0972  ...   1.1680  1.6593  1.2818
 -0.2771  3.1374 -5.3266  ...   1.4195  6.0459  1.1207
  1.9673  3.3129  3.2074  ...   0.9304 -0.8842 -1.3621
           ...             ⋱             ...
  1.3010 -1.7196  3.2471  ...  -1.8984  2.9834  2.0477
  0.7719 -2.2480 -0.6071  ...   4.6178 -3.1997  0.5709
  2.6377 -5.4742 -1.2875  ...   1.6709 -0.6052 -3.7762

(1 ,1 ,.,.) =
1.00000e-02 *
 -0.7781 -0.4420 -1.4554  ...   1.5402 -1.3788  2.5803
  2.3241 -1.3687  0.8030  ...  -2.7009 -3.5372  1.9341
 -0.0503  2.4143  1.4365  ...   0.7629  5.8957  1.7942
           ...             ⋱             ...
 -6.8150 -2.3861  2.3645  ...  -2.0620 -0.0627  4.3849
 -0.7310 -7.1793  0.2986  ...  -0.3329 -3.3218 -4.0661
 -0.9377  0.1143 -1.3899  ...   4.3625 -2.3391 -0.2196

(1 ,2 ,.,.) =
1.00000e-02 *
  0.2875 -1.2542  6.2255  ...  -0.4892  1.0364  2.3138
 -0.9701  3.3734 -1.2590  ...   0.5698  1.8889  1.9568
  1.2484  4.0164 -1.4857  ...   2.9830  1.0203  2.5866
           ...             ⋱             ...
  0.2566  3.9015 -0.9904  ...   0.4520 -6.9477 -0.9145
 -3.4336 -3.4242  2.2993  ...   1.1728  1.1233  1.2550
 -0.9215 -0.0847  1.5080  ...   2.0479 -2.3217 -0.0313
     ⋮

(2 ,0 ,.,.) =
1.00000e-02 *
  0.3407  4.4103  1.0592  ...  -0.0456 -0.5867  0.7464
 -0.4382 -1.4446  1.4835  ...  -0.9019  0.7855 -0.1822
  3.6491 -4.5887 -0.4664  ...  -0.7682 -1.0206  2.9243
           ...             ⋱             ...
 -2.3662 -1.0675 -0.7690  ...   0.8398  0.4604  1.6966
  5.0490  0.8002 -0.6414  ...   1.9684  1.7014 -3.4272
 -0.4687  1.6719  0.4215  ...  -2.6047 -2.5274  0.8941

(2 ,1 ,.,.) =
1.00000e-02 *
 -1.5808 -1.4652  2.6526  ...   1.5501  2.2062 -1.1649
  1.8200 -1.9857  0.2430  ...  -0.6430  0.8094 -2.0726
```

```
  1.1859 -2.1579 -1.1094  ...  -0.5029  3.6159 -3.6920
           ...             ⋱            ...
 -2.6073 -0.8166 -0.2750  ...  -6.0130 -0.5468  1.4847
  4.0052 -0.8093 -2.8038  ...   1.4979  2.1255 -1.2737
 -3.6938 -0.5962  1.5727  ...  -1.1885  0.3003 -2.0959

(2 ,2 ,.,.) =
1.00000e-02 *
  1.1110 -0.7731 -4.8425  ...  -1.9362 -2.7463  4.3953
 -0.2578 -1.1544 -1.5189  ...  -1.1802 -1.3668  0.1457
  0.9995  1.5784 -6.1460  ...  -1.9347  4.4065 -0.8176
           ...             ⋱            ...
 -1.3352 -1.7754  0.8768  ...  -0.0243  0.0313  2.8065
  3.7937  1.1224  1.8704  ...  -1.7029 -2.0356 -0.3463
  1.8513 -1.3500  2.8523  ...   0.2503  0.2158  2.4576
...
     ⋮

(61,0 ,.,.) =
1.00000e-02 *
  2.8970 -5.6028 -3.5206  ...   0.4689  0.6646 -3.3684
 -4.6882 -1.7180 -1.3746  ...   2.0732 -2.4011  0.7847
 -1.1916 -0.7925  5.2070  ...  -0.1810 -0.3390 -2.5826
           ...             ⋱            ...
  0.3124  2.8100 -3.1849  ...   4.5011  0.9226  2.4896
 -1.3838  2.3848  1.8429  ...  -1.2402 -1.5649  3.9464
 -1.2520 -3.7040 -0.9414  ...   3.1602  1.1630 -2.3451

(61,1 ,.,.) =
1.00000e-02 *
  1.4376 -3.0994  5.0351  ...   1.6115 -0.8978 -0.1897
  1.9688 -6.3252  1.9896  ...  -1.4432  3.4118 -1.4325
 -4.3798 -2.2465  2.0578  ...   1.2641 -4.4128 -2.4449
           ...             ⋱            ...
 -0.9304  2.7215  0.8343  ...   1.4009 -4.3836 -1.3073
  1.4389  3.9609  1.5280  ...  -0.2879 -0.5510  2.2877
  1.4933  0.7655  0.6487  ...   1.8393  1.5835  0.6096

(61,2 ,.,.) =
1.00000e-02 *
 -3.2170 -1.4534 -1.2253  ...  -0.2634  1.7823  1.8937
  0.4291  2.6203  3.0673  ...   2.2584 -5.2594 -1.1889
 -2.6205 -1.9329  0.4050  ...   2.1702 -2.3393 -2.1675
           ...             ⋱            ...
 -4.1491  2.5399  0.3679  ...   3.8015  5.4742  0.5811
 -1.5494  1.7888 -2.3090  ...  -2.0745 -2.8779  5.0300
```

```
 -0.5829 -3.6522 -1.8430  ...   3.8235  1.4160  1.8975
     ⋮

(62,0 ,.,.) =
1.00000e-02 *
  0.6749  1.9942 -1.0533  ...   0.1311  4.0191 -1.3084
 -0.2273 -5.1297  1.4480  ...   0.4108  1.2235  0.8971
 -0.0731 -2.5022  0.8157  ...   0.2392 -0.1335  0.5399
           ...             ⋱             ...
  0.8957 -0.8075  0.8147  ...  -2.4049  2.8092  3.1275
  0.2161 -1.6033  1.8246  ...   0.5912  3.7720 -3.0592
 -1.2401  2.4554  1.7200  ...   0.9085  2.1319 -4.1605

(62,1 ,.,.) =
1.00000e-02 *
  1.6733 -4.7003 -1.0108  ...   1.5136 -0.2312 -2.9744
  4.3313  0.4270 -4.4646  ...   0.4609 -0.8794 -3.5395
  1.0714 -1.9545 -0.2131  ...  -4.6601  4.3179 -1.4707
           ...             ⋱             ...
 -0.9831  0.7268  2.2284  ...  -3.1483 -0.3843 -0.0530
 -3.8084 -0.9926  2.2342  ...   1.6763  1.3814 -1.7054
 -0.6535  0.4554 -1.8025  ...   1.4265  4.4104  0.9796

(62,2 ,.,.) =
1.00000e-02 *
  1.1205 -2.0540  5.3732  ...  -3.3141  4.3511 -1.1712
 -1.1058 -3.4854  1.6923  ...   0.6975 -0.6917 -0.8297
 -1.4260 -6.3759 -6.1516  ...   2.3643 -0.5420 -0.8919
           ...             ⋱             ...
 -1.3532  0.1896  2.2477  ...  -0.0012 -1.6725  1.3585
  3.4569 -6.6761  0.1119  ...  -2.3943  4.4839  1.2020
 -2.9624  0.7192 -1.9399  ...  -1.6306 -0.3034  1.3437
     ⋮

(63,0 ,.,.) =
1.00000e-02 *
 -2.8883  0.9223  0.4727  ...   2.5254  0.6002  1.3747
 -0.6953 -0.3414 -1.8400  ...   2.5456 -0.1516 -2.1682
 -0.5400 -2.8877  1.7134  ...  -0.0886  0.7505  2.6778
           ...             ⋱             ...
 -0.5361  3.3156 -7.6066  ...   2.3346  0.8174  0.0303
 -3.4057  2.6619 -0.3110  ...  -0.3617  0.3958  0.0972
 -4.5076 -2.4737 -0.4838  ...  -0.7773 -0.8255 -2.9103

(63,1 ,.,.) =
1.00000e-02 *
```

```
 -1.3164  1.1769 -0.4530  ...  -1.9931  2.9545  7.1766
 -1.3244 -2.5797 -3.7692  ...  -0.3148  4.8628 -4.6815
 -0.1349 -2.6735  0.1890  ...   1.0249  4.3442  1.4393
           ...             ⋱             ...
  0.3001 -2.7956 -0.0273  ...  -0.2061 -4.5356  1.0858
 -2.6624 -3.0735  3.5968  ...  -1.6982  1.3687  0.6566
 -4.6600  2.9132 -1.8234  ...  -0.9616 -0.7530 -0.0238

(63,2 ,.,.) =
1.00000e-02 *
 -0.2576 -2.7887  2.5190  ...   3.4712 -3.0500  1.6395
  1.1221 -1.5014 -1.5318  ...   3.0269 -1.2203  0.1829
 -1.8898 -1.4792  0.2215  ...   1.1256 -4.2082  0.9612
           ...             ⋱             ...
 -3.3994 -2.8482  1.2415  ...  -2.4822  3.1443 -1.3401
  3.0496  0.8509 -2.7656  ...  -0.4840  2.4440 -2.3337
  0.3858  0.3989  2.2471  ...  -1.3018 -1.0115 -2.3956
[torch.FloatTensor of size 64x3x7x7]
```

```
train(net, train_data, valid_data, 20, optimizer, criterion)
```

```
Epoch 0. Train Loss: 2.607935, Train Acc: 0.485577, Valid Loss: 1.355199, Valid Acc:
0.726562, Time 00:00:01
Epoch 1. Train Loss: 2.598156, Train Acc: 0.560397, Valid Loss: 1.730986, Valid Acc:
0.273438, Time 00:00:03
Epoch 2. Train Loss: 2.768743, Train Acc: 0.520433, Valid Loss: 1.205057, Valid Acc:
0.273438, Time 00:00:03
Epoch 3. Train Loss: 2.541335, Train Acc: 0.489483, Valid Loss: 2.040526, Valid Acc:
0.273438, Time 00:00:03
Epoch 4. Train Loss: 1.286088, Train Acc: 0.576322, Valid Loss: 1.137075, Valid Acc:
0.309063, Time 00:00:03
Epoch 5. Train Loss: 1.081273, Train Acc: 0.528245, Valid Loss: 2.186767, Valid Acc:
0.277344, Time 00:00:03
Epoch 6. Train Loss: 1.001506, Train Acc: 0.529147, Valid Loss: 1.467141, Valid Acc:
0.380781, Time 00:00:03
Epoch 7. Train Loss: 0.850950, Train Acc: 0.618990, Valid Loss: 2.721868, Valid Acc:
0.273438, Time 00:00:03
Epoch 8. Train Loss: 1.439979, Train Acc: 0.511719, Valid Loss: 1.184357, Valid Acc:
0.448125, Time 00:00:03
Epoch 9. Train Loss: 1.033413, Train Acc: 0.558894, Valid Loss: 0.775361, Valid Acc:
0.468125, Time 00:00:03
```

```
Epoch 10. Train Loss: 1.197576, Train Acc: 0.595553, Valid Loss: 0.748739, Valid Acc:
0.688594, Time 00:00:03
Epoch 11. Train Loss: 0.816506, Train Acc: 0.655649, Valid Loss: 1.524445, Valid Acc:
0.301250, Time 00:00:03
Epoch 12. Train Loss: 1.000863, Train Acc: 0.525240, Valid Loss: 0.766739, Valid Acc:
0.551562, Time 00:00:03
Epoch 13. Train Loss: 0.705289, Train Acc: 0.705228, Valid Loss: 0.542888, Valid Acc:
0.726562, Time 00:00:03
Epoch 14. Train Loss: 0.676270, Train Acc: 0.608774, Valid Loss: 0.608839, Valid Acc:
0.647188, Time 00:00:03
Epoch 15. Train Loss: 0.617810, Train Acc: 0.679087, Valid Loss: 0.521360, Valid Acc:
0.726562, Time 00:00:03
Epoch 16. Train Loss: 0.760871, Train Acc: 0.594351, Valid Loss: 0.636761, Valid Acc:
0.639375, Time 00:00:03
Epoch 17. Train Loss: 1.137678, Train Acc: 0.544471, Valid Loss: 0.598063, Valid Acc:
0.715938, Time 00:00:03
Epoch 18. Train Loss: 0.719091, Train Acc: 0.685096, Valid Loss: 0.534916, Valid Acc:
0.755937, Time 00:00:03
Epoch 19. Train Loss: 0.647208, Train Acc: 0.665264, Valid Loss: 0.598654, Valid Acc:
0.695000, Time 00:00:03
```

通过上面的结果可以看到，使用预训练的模型能够非常快的达到 95% 左右的验证集准确率，而不使用预训练模型只能到 70% 左右的验证集准确率，所以使用一个预训练的模型能够在较小的数据集上也取得一个非常好的效果，因为对于图片识别分类任务，最底层的卷积层识别的都是一些通用的特征，比如形状、纹理等等，所以对于很多图像分类、识别任务，都可以使用预训练的网络得到更好的结果。