```
import os
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import torch

from torch import nn
import torch.optim as optim
import torchvision
#pip install torchvision
from torchvision import transforms, models, datasets
#https://pytorch.org/docs/stable/torchvision/index.html
import imageio
import time
import warnings
import random
import sys
import copy
import json
from PIL import Image
```

```
path ='/content/drive/My Drive/dataset/kaggle/chest/chest-xray-pneumonia.zip'
```

```
!unzip '/content/drive/My Drive/dataset/kaggle/chest-xray-pneumonia.zip'
```

```
data_transforms = {
    'train': transforms.Compose([transforms.Resize(256),
        transforms.CenterCrop(224),#从中心开始裁剪
        transforms.RandomRotation(45),#随机旋转，-45到45度之间随机选
        transforms.RandomHorizontalFlip(p=0.5),#随机水平翻转 选择一个概率概率
        transforms.RandomVerticalFlip(p=0.5),#随机垂直翻转
        transforms.ColorJitter(brightness=0.2, contrast=0.1, saturation=0.1, hue=0.1),#参数1
        transforms.RandomGrayscale(p=0.025),#概率转换成灰度率，3通道就是R=G=B
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])#均值，标准差
    ]),
    'test': transforms.Compose([transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

```
batch_size = 32
dir_path = '/content/chest_xray/'
image_datasets = {x: datasets.ImageFolder(os.path.join(dir_path, x), data_transforms[x]) for
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=batch_size, shuf
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val', 'test']}
class_names = image_datasets['train'].classes
```

```
dataloaders
```

```
{'test': <torch.utils.data.dataloader.DataLoader at 0x7fe1bd21e630>,
 'train': <torch.utils.data.dataloader.DataLoader at 0x7fe1bd21e198>,
 'val': <torch.utils.data.dataloader.DataLoader at 0x7fe1bd21e518>}
```

```
class_names
```

```
['NORMAL', 'PNEUMONIA']
```

```
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1,2,0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std*inp + mean
    inp = np.clip(inp,0,1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)

inputs,classes = next(iter(dataloaders["val"]))
```

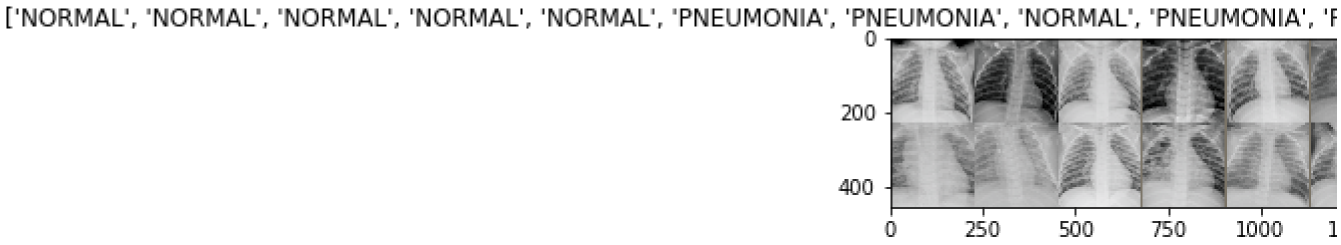
```
inputs
```

```
tensor([[[[-1.7240, -1.7583, -1.7754,  ..., -1.8268, -1.8097, -1.7925],
           [-1.6898, -1.7240, -1.7412,  ..., -1.7925, -1.7754, -1.7583],
           [-1.6384, -1.6898, -1.7069,  ..., -1.7412, -1.7240, -1.6898],
           ...,
           [ 1.1872,  1.2214,  1.2043,  ...,  0.2453, -0.0801, -0.1314],
           [ 1.2043,  1.2214,  1.2043,  ...,  0.2111, -0.0629, -0.1143],
           [ 1.1872,  1.2214,  1.2214,  ...,  0.2111, -0.0629, -0.1486]],

          [[-1.6331, -1.6681, -1.6856,  ..., -1.7381, -1.7206, -1.7031],
           [-1.5980, -1.6331, -1.6506,  ..., -1.7031, -1.6856, -1.6681],
           [-1.5455, -1.5980, -1.6155,  ..., -1.6506, -1.6331, -1.5980],
           ...,
           [ 1.3431,  1.3782,  1.3606,  ...,  0.3803,  0.0476, -0.0049],
           [ 1.3606,  1.3782,  1.3606,  ...,  0.3452,  0.0651,  0.0126],
           [ 1.3431,  1.3782,  1.3782,  ...,  0.3452,  0.0651, -0.0224]],

          [[-1.4036, -1.4384, -1.4559,  ..., -1.5081, -1.4907, -1.4733],
           [-1.3687, -1.4036, -1.4210,  ..., -1.4733, -1.4559, -1.4384],
           [-1.3164, -1.3687, -1.3861,  ..., -1.4210, -1.4036, -1.3687],
           ...,
           [ 1.5594,  1.5942,  1.5768,  ...,  0.6008,  0.2696,  0.2173],
           [ 1.5768,  1.5942,  1.5768,  ...,  0.5659,  0.2871,  0.2348],
           [ 1.5594,  1.5942,  1.5942,  ...,  0.5659,  0.2871,  0.1999]]],


         [[[-0.5082, -0.4568, -0.4397,  ..., -0.2513, -0.2171, -0.2342],
           [-0.2513, -0.1143,  0.0056,  ..., -0.2684, -0.3198, -0.3883],
           [ 0.0398,  0.0569,  0.0227,  ..., -0.3027, -0.3027, -0.3027],
           ...,
           [ 0.0569,  0.0569,  0.0741,  ..., -1.6042, -1.7925, -1.9638],
           [ 0.0227,  0.0398,  0.0741,  ..., -1.6555, -1.8097, -1.9638],
           [ 0.0398,  0.0569,  0.0741,  ..., -1.6898, -1.8268, -1.9809]],

          [[-0.3901, -0.3375, -0.3200,  ..., -0.1275, -0.0924, -0.1099],
           [-0.1275,  0.0126,  0.1352,  ..., -0.1450, -0.1975, -0.2675],
           [ 0.1702,  0.1877,  0.1527,  ..., -0.1800, -0.1800, -0.1800],
           ...,
           [ 0.1877,  0.1877,  0.2052,  ..., -1.5105, -1.7031, -1.8782],
           [ 0.1527,  0.1702,  0.2052,  ..., -1.5630, -1.7206, -1.8782],
           [ 0.1702,  0.1877,  0.2052,  ..., -1.5980, -1.7381, -1.8957]],

          [[-0.1661, -0.1138, -0.0964,  ...,  0.0953,  0.1302,  0.1128],
           [ 0.0953,  0.2348,  0.3568,  ...,  0.0779,  0.0256, -0.0441],
           [ 0.3916,  0.4091,  0.3742,  ...,  0.0431,  0.0431,  0.0431],
           ...,
           [ 0.4091,  0.4091,  0.4265,  ..., -1.2816, -1.4733, -1.6476],
           [ 0.3742,  0.3916,  0.4265,  ..., -1.3339, -1.4907, -1.6476],
           [ 0.3916,  0.4091,  0.4265,  ..., -1.3687, -1.5081, -1.6650]]],


         [[[ 0.6563,  0.6563,  0.6734,  ...,  0.5364,  0.5193,  0.5022],
           [ 0.6734,  0.6734,  0.6906,  ...,  0.5707,  0.5707,  0.5193],
           [ 0.7077,  0.7077,  0.6906,  ...,  0.6563,  0.6221,  0.5707],
           ...,
           [ 0.5536,  0.6221,  0.6906,  ...,  0.8789,  0.9132,  0.9303],
           [ 0.5707,  0.6221,  0.7077,  ...,  0.8618,  0.9132,  0.9303],
           [ 0.6221,  0.6392,  0.7248,  ...,  0.8276,  0.8961,  0.9132]],
          [[ 0.8004,  0.8004,  0.8179,  ...,  0.6779,  0.6604,  0.6429],
           [ 0.8179,  0.8179,  0.8354,  ...,  0.7129,  0.7129,  0.6604],
           [ 0.8529,  0.8529,  0.8354,  ...,  0.8004,  0.7654,  0.7129],
           ...,
           [ 0.6954,  0.7654,  0.8354,  ...,  1.0280,  1.0630,  1.0805],
           [ 0.7129,  0.7654,  0.8529,  ...,  1.0105,  1.0630,  1.0805],
           [ 0.7654,  0.7829,  0.8704,  ...,  0.9755,  1.0455,  1.0630]],

          [[ 1.0191,  1.0191,  1.0365,  ...,  0.8971,  0.8797,  0.8622],
           [ 1.0365,  1.0365,  1.0539,  ...,  0.9319,  0.9319,  0.8797],
           [ 1.0714,  1.0714,  1.0539,  ...,  1.0191,  0.9842,  0.9319],
           ...,
           [ 0.9145,  0.9842,  1.0539,  ...,  1.2457,  1.2805,  1.2980],
           [ 0.9319,  0.9842,  1.0714,  ...,  1.2282,  1.2805,  1.2980],
           [ 0.9842,  1.0017,  1.0888,  ...,  1.1934,  1.2631,  1.2805]]],


         ...,


         [[[-0.9877, -0.9705, -0.9534,  ..., -0.9020, -0.9363, -0.9192],
           [-0.9877, -0.9705, -0.9534,  ..., -0.8849, -0.9192, -0.9363],
           [-0.9705, -0.9877, -0.9705,  ..., -0.8849, -0.9020, -0.9192],
           ...,
           [ 0.6392,  0.6906,  0.7762,  ..., -2.0152, -1.9467, -1.8953],
           [ 0.6049,  0.7077,  0.7933,  ..., -1.9980, -1.9467, -1.8953],
           [ 0.6221,  0.7077,  0.7933,  ..., -1.9809, -1.9295, -1.8953]],

          [[-0.8803, -0.8627, -0.8452,  ..., -0.7927, -0.8277, -0.8102],
           [-0.8803, -0.8627, -0.8452,  ..., -0.7752, -0.8102, -0.8277],
           [-0.8627, -0.8803, -0.8627,  ..., -0.7752, -0.7927, -0.8102],
           ...,
           [ 0.7829,  0.8354,  0.9230,  ..., -1.9307, -1.8606, -1.8081],
           [ 0.7479,  0.8529,  0.9405,  ..., -1.9132, -1.8606, -1.8081],
           [ 0.7654,  0.8529,  0.9405,  ..., -1.8957, -1.8431, -1.8081]],

          [[-0.6541, -0.6367, -0.6193,  ..., -0.5670, -0.6018, -0.5844],
           [-0.6541, -0.6367, -0.6193,  ..., -0.5495, -0.5844, -0.6018],
           [-0.6367, -0.6541, -0.6367,  ..., -0.5495, -0.5670, -0.5844],
           ...,
           [ 1.0017,  1.0539,  1.1411,  ..., -1.6999, -1.6302, -1.5779],
           [ 0.9668,  1.0714,  1.1585,  ..., -1.6824, -1.6302, -1.5779],
           [ 0.9842,  1.0714,  1.1585,  ..., -1.6650, -1.6127, -1.5779]]],


         [[[-0.6965, -0.6794, -0.6281,  ..., -0.3369, -0.4226, -0.4397],
           [-0.6281, -0.6109, -0.5938,  ..., -0.3198, -0.2171, -0.1828],
           [-0.6281, -0.6281, -0.6109,  ..., -0.3369, -0.1999, -0.1657],
           ...,
           [ 0.9474,  0.9132,  0.9303,  ...,  0.7077,  0.6221,  0.6221],
           [ 0.9303,  0.9474,  0.9474,  ...,  0.6906,  0.6563,  0.6392],
           [ 0.9474,  0.9474,  0.9132,  ...,  0.7248,  0.7419,  0.6734]],

          [[-0.5826, -0.5651, -0.5126,  ..., -0.2150, -0.3025, -0.3200],
           [-0.5126, -0.4951, -0.4776,  ..., -0.1975, -0.0924, -0.0574],
           [-0.5126, -0.5126, -0.4951,  ..., -0.2150, -0.0749, -0.0399],
           ...,
           [ 1.0980,  1.0630,  1.0805,  ...,  0.8529,  0.7654,  0.7654],
           [ 1.0805,  1.0980,  1.0980,  ...,  0.8354,  0.8004,  0.7829],
           [ 1.0980,  1.0980,  1.0630,  ...,  0.8704,  0.8880,  0.8179]],

          [[-0.3578, -0.3404, -0.2881,  ...,  0.0082, -0.0790, -0.0964],
           [-0.2881, -0.2707, -0.2532,  ...,  0.0256,  0.1302,  0.1651],
           [-0.2881, -0.2881, -0.2707,  ...,  0.0082,  0.1476,  0.1825],
           ...,
           [ 1.3154,  1.2805,  1.2980,  ...,  1.0714,  0.9842,  0.9842],
           [ 1.2980,  1.3154,  1.3154,  ...,  1.0539,  1.0191,  1.0017],
           [ 1.3154,  1.3154,  1.2805,  ...,  1.0888,  1.1062,  1.0365]]],


         [[[ 0.2967,  0.2624,  0.2967,  ...,  0.3823,  0.3138,  0.2967],
           [ 0.3309,  0.2624,  0.2967,  ...,  0.4337,  0.3481,  0.3138],
           [ 0.2967,  0.2967,  0.2967,  ...,  0.4337,  0.3994,  0.3309],
           ...,
           [ 1.3242,  1.3413,  1.3755,  ...,  0.7933,  0.8789,  0.9646],
           [ 1.3584,  1.3413,  1.3584,  ...,  0.8618,  0.9132,  0.9646],
           [ 1.3584,  1.3242,  1.3413,  ...,  0.8961,  0.9132,  0.9474]],

          [[ 0.4328,  0.3978,  0.4328,  ...,  0.5203,  0.4503,  0.4328],
           [ 0.4678,  0.3978,  0.4328,  ...,  0.5728,  0.4853,  0.4503],
           [ 0.4328,  0.4328,  0.4328,  ...,  0.5728,  0.5378,  0.4678],
           ...,
           [ 1.4832,  1.5007,  1.5357,  ...,  0.9405,  1.0280,  1.1155],
           [ 1.5182,  1.5007,  1.5182,  ...,  1.0105,  1.0630,  1.1155],
           [ 1.5182,  1.4832,  1.5007,  ...,  1.0455,  1.0630,  1.0980]],

          [[ 0.6531,  0.6182,  0.6531,  ...,  0.7402,  0.6705,  0.6531],
           [ 0.6879,  0.6182,  0.6531,  ...,  0.7925,  0.7054,  0.6705],
           [ 0.6531,  0.6531,  0.6531,  ...,  0.7925,  0.7576,  0.6879],
           ...,
           [ 1.6988,  1.7163,  1.7511,  ...,  1.1585,  1.2457,  1.3328],
           [ 1.7337,  1.7163,  1.7337,  ...,  1.2282,  1.2805,  1.3328],
           [ 1.7337,  1.6988,  1.7163,  ...,  1.2631,  1.2805,  1.3154]]]])
```

```
classes
```

```
tensor([0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1])
```

```
out = torchvision.utils.make_grid(inputs)
class_names = image_datasets["val"].classes
imshow(out, title = [class_names[x] for x in classes])

# {'NORMAL': 0 , 'PNEUMONIA': 1}
```


['NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'F
0
200
400
0
250
500
750
1000
1

```
class_names
```

```
['NORMAL', 'PNEUMONIA']
```

```
model_name = 'resnet'  #可选的比较多 ['resnet', 'alexnet', 'vgg', 'squeezenet', 'densenet', 'i
#是否用人家训练好的特征来做
feature_extract = True


# 是否用GPU训练
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
CUDA is available!  Training on GPU ...
```

```
device
```

```
device(type='cuda', index=0)
```

```
def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

```
model_ft = models.resnet101()
model_ft
```

```
def initialize_model(model_name, num_classes, feature_extract, use_pretrained=True):
    # 选择合适的模型，不同模型的初始化方法稍微有点区别
    model_ft = None
    input_size = 0

    if model_name == "resnet":
        """ Resnet152
        """
        model_ft = models.resnet152(pretrained=use_pretrained)
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Sequential(nn.Linear(num_ftrs, 102),
                                   nn.LogSoftmax(dim=1))
        input_size = 224

    else:
        print("Invalid model name, exiting...")
        exit()

    return model_ft, input_size


model_ft, input_size = initialize_model(model_name, 2, feature_extract, use_pretrained=True)

#GPU计算
model_ft = model_ft.to(device)

# 模型保存
filename='checkpoint.pth'

# 是否训练所有层
params_to_update = model_ft.parameters()
print("Params to learn:")
if feature_extract:
    params_to_update = []
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            params_to_update.append(param)
            print("\t",name)
else:
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            print("\t",name)
```

```
Downloading: "https://download.pytorch.org/models/resnet152-b121ed2d.pth" to /root/.cach
```

100% 230M/230M [00:08<00:00, 28.7MB/s]

```
Params to learn:
         fc.0.weight
         fc.0.bias
```

```
# 优化器设置
optimizer_ft = optim.Adam(params_to_update, lr=1e-2)
scheduler = optim.lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)#学习率每7个epoch衰
#最后一层已经LogSoftmax()了，所以不能nn.CrossEntropyLoss()来计算了，nn.CrossEntropyLoss()相当于l
```

```
criterion = nn.NLLLoss()
```

```
optimizer_ft.param_groups[0]
```

```
{'amsgrad': False,
 'betas': (0.9, 0.999),
 'eps': 1e-08,
 'initial_lr': 0.01,
 'lr': 0.01,
 'params': [Parameter containing:
  tensor([[ 0.0158,  0.0114,  0.0178,  ...,  0.0056, -0.0073,  0.0070],
          [-0.0027, -0.0035, -0.0131,  ..., -0.0218,  0.0193,  0.0196],
          [ 0.0121,  0.0140,  0.0053,  ...,  0.0122, -0.0108, -0.0001],
          ...,
          [-0.0109,  0.0093,  0.0008,  ...,  0.0125, -0.0051, -0.0153],
          [-0.0136, -0.0058, -0.0020,  ..., -0.0075, -0.0135,  0.0095],
          [ 0.0190, -0.0088,  0.0053,  ...,  0.0140, -0.0160, -0.0184]],
         device='cuda:0', requires_grad=True), Parameter containing:
  tensor([-0.0013, -0.0212,  0.0118,  0.0199,  0.0058, -0.0104, -0.0112,  0.0099,
           0.0022,  0.0035,  0.0115, -0.0188,  0.0030,  0.0159, -0.0193, -0.0129,
          -0.0135,  0.0008,  0.0211,  0.0168,  0.0047,  0.0184, -0.0085,  0.0023,
          -0.0041,  0.0024, -0.0144, -0.0192, -0.0047, -0.0039,  0.0154,  0.0024,
           0.0145,  0.0106, -0.0114,  0.0208, -0.0110,  0.0022, -0.0091, -0.0030,
           0.0115, -0.0175,  0.0199,  0.0168,  0.0213,  0.0029,  0.0159, -0.0067,
          -0.0178, -0.0080,  0.0010, -0.0191,  0.0102,  0.0217,  0.0084,  0.0109,
          -0.0202, -0.0205,  0.0050, -0.0211, -0.0197, -0.0017,  0.0032, -0.0032,
           0.0203,  0.0087, -0.0151, -0.0098,  0.0200,  0.0034,  0.0031,  0.0171,
          -0.0122, -0.0090, -0.0123,  0.0139, -0.0065, -0.0175, -0.0140,  0.0050,
           0.0162, -0.0075,  0.0066,  0.0173, -0.0003,  0.0159,  0.0200, -0.0156,
          -0.0070, -0.0177,  0.0211,  0.0097,  0.0003,  0.0023, -0.0086,  0.0185,
          -0.0172, -0.0161, -0.0055,  0.0134,  0.0036,  0.0214], device='cuda:0',
         requires_grad=True)],
 'weight_decay': 0}
```

```
optimizer_ft.param_groups[0]['lr']
```

```
0.01
```

```
def train_model(model, dataloaders, criterion, optimizer, num_epochs=25, is_inception=False,
    since = time.time()
    best_acc = 0
    """
    checkpoint = torch.load(filename)
    best_acc = checkpoint['best_acc']
    model.load_state_dict(checkpoint['state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer'])
    model.class_to_idx = checkpoint['mapping']
    """
    model.to(device)

    val_acc_history = []
    train_acc_history = []
    train_losses = []
```

```
        valid_losses = []
        LRs = [optimizer.param_groups[0]['lr']]

        best_model_wts = copy.deepcopy(model.state_dict())

        for epoch in range(num_epochs):
            print('Epoch {}/{}'.format(epoch, num_epochs - 1))
            print('-' * 10)

            # 训练和验证
            for phase in ['train', 'test']:
                if phase == 'train':
                    model.train()  # 训练
                else:
                    model.eval()   # 验证

                running_loss = 0.0
                running_corrects = 0

                # 把数据都取个遍
                for inputs, labels in dataloaders[phase]:
                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    # 清零
                    optimizer.zero_grad()
                    # 只有训练的时候计算和更新梯度
                    with torch.set_grad_enabled(phase == 'train'):
                        if is_inception and phase == 'train':
                            outputs, aux_outputs = model(inputs)
                            loss1 = criterion(outputs, labels)
                            loss2 = criterion(aux_outputs, labels)
                            loss = loss1 + 0.4*loss2
                        else:#resnet执行的是这里
                            outputs = model(inputs)
                            loss = criterion(outputs, labels)

                        _, preds = torch.max(outputs, 1)

                        # 训练阶段更新权重
                        if phase == 'train':
                            loss.backward()
                            optimizer.step()

                    # 计算损失
                    running_loss += loss.item() * inputs.size(0)
                    running_corrects += torch.sum(preds == labels.data)

                epoch_loss = running_loss / len(dataloaders[phase].dataset)
                epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)
                time_elapsed = time.time() - since
                print('Time elapsed {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 6
                print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))


                # 得到最好那次的模型
                if phase == 'test' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())
                    state = {
                      'state_dict': model.state_dict(),
                      'best_acc': best_acc,
                      'optimizer' : optimizer.state_dict(),
                    }
                    torch.save(state, filename)
                if phase == 'test':
                    val_acc_history.append(epoch_acc)
                    valid_losses.append(epoch_loss)
                    scheduler.step(epoch_loss)
                if phase == 'train':
                    train_acc_history.append(epoch_acc)
                    train_losses.append(epoch_loss)

            print('Optimizer learning rate : {:.7f}'.format(optimizer.param_groups[0]['lr']))
            LRs.append(optimizer.param_groups[0]['lr'])
            print()

        time_elapsed = time.time() - since
        print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 6
        print('Best val Acc: {:4f}'.format(best_acc))

        # 训练完后用最好的一次当做模型最终的结果
        model.load_state_dict(best_model_wts)
        return model, val_acc_history, train_acc_history, valid_losses, train_losses, LRs


    d_losses, train_losses, LRs  = train_model(model_ft, dataloaders, criterion, optimizer_ft, nu


    for param in model_ft.parameters():
        param.requires_grad = True

    # 再继续训练所有的参数，学习率调小一点
    optimizer = optim.Adam(params_to_update, lr=1e-4)
    scheduler = optim.lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

    # 损失函数
    criterion = nn.NLLLoss()


    #----------------# Load the checkpoint

    checkpoint = torch.load(filename)
```

```
    best_acc = checkpoint['best_acc']
    model_ft.load_state_dict(checkpoint['state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer'])
    #model_ft.class_to_idx = checkpoint['mapping']


    model_ft, val_acc_history, train_acc_history, valid_losses, train_losses, LRs  = train_model(
         Epoch 0/32
         ----------
         Time elapsed 2m 43s
         train Loss: 0.5569 Acc: 0.9168
         Time elapsed 2m 55s
         test Loss: 1.4224 Acc: 0.8365
         /usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:122: UserWarning: Det
           "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarning)
         Optimizer learning rate : 0.0100000

         Epoch 1/32
         ----------
         Time elapsed 5m 38s
         train Loss: 0.4364 Acc: 0.9279
         Time elapsed 5m 50s
         test Loss: 1.2248 Acc: 0.8494
         Optimizer learning rate : 0.0100000

         Epoch 2/32
         ----------
         Time elapsed 8m 33s
         train Loss: 0.2826 Acc: 0.9448
         Time elapsed 8m 46s
         test Loss: 1.0012 Acc: 0.8638
         Optimizer learning rate : 0.0100000

         Epoch 3/32
         ----------
         Time elapsed 11m 29s
         train Loss: 0.3544 Acc: 0.9344
         Time elapsed 11m 41s
         test Loss: 1.4896 Acc: 0.8173
         Optimizer learning rate : 0.0100000

         Epoch 4/32
         ----------
         Time elapsed 14m 24s
         train Loss: 0.2890 Acc: 0.9429
         Time elapsed 14m 36s
         test Loss: 0.7926 Acc: 0.8782
         Optimizer learning rate : 0.0100000

         Epoch 5/32
         ----------
         Time elapsed 17m 19s
         train Loss: 0.4393 Acc: 0.9229
         Time elapsed 17m 32s
         test Loss: 0.8510 Acc: 0.8878
         Optimizer learning rate : 0.0100000

         Epoch 6/32
         ----------
         Time elapsed 20m 14s
         train Loss: 0.3312 Acc: 0.9363
         Time elapsed 20m 27s
         test Loss: 0.8966 Acc: 0.8622
         Optimizer learning rate : 0.0100000
```

```
Epoch 7/32
----------
Time elapsed 23m 9s
train Loss: 0.3816 Acc: 0.9335
Time elapsed 23m 21s
test Loss: 0.8659 Acc: 0.8846
Optimizer learning rate : 0.0100000

Epoch 8/32
----------
Time elapsed 26m 4s
train Loss: 0.3161 Acc: 0.9356
Time elapsed 26m 16s
test Loss: 1.0671 Acc: 0.8686
Optimizer learning rate : 0.0100000

Epoch 9/32
----------
Time elapsed 28m 60s
train Loss: 0.5085 Acc: 0.9178
Time elapsed 29m 12s
test Loss: 0.9528 Acc: 0.8926
Optimizer learning rate : 0.0100000

Epoch 10/32
----------
Time elapsed 31m 55s
train Loss: 0.3657 Acc: 0.9423
Time elapsed 32m 7s
test Loss: 0.8324 Acc: 0.8862
Optimizer learning rate : 0.0100000

Epoch 11/32
----------
Time elapsed 34m 49s
train Loss: 0.3610 Acc: 0.9365
Time elapsed 35m 2s
test Loss: 0.9890 Acc: 0.8782
Optimizer learning rate : 0.0100000

Epoch 12/32
----------
Time elapsed 37m 44s
train Loss: 0.3123 Acc: 0.9413
Time elapsed 37m 57s
test Loss: 0.7084 Acc: 0.8782
Optimizer learning rate : 0.0100000

Epoch 13/32
----------
Time elapsed 40m 39s
train Loss: 0.6543 Acc: 0.9147
Time elapsed 40m 52s
test Loss: 0.8984 Acc: 0.8878
Optimizer learning rate : 0.0100000

Epoch 14/32
```

```
----------
Time elapsed 43m 34s
train Loss: 0.3789 Acc: 0.9415
Time elapsed 43m 47s
test Loss: 0.7452 Acc: 0.9006
Optimizer learning rate : 0.0100000

Epoch 15/32
----------
Time elapsed 46m 29s
train Loss: 0.5715 Acc: 0.9214
Time elapsed 46m 42s
test Loss: 0.8080 Acc: 0.8862
Optimizer learning rate : 0.0100000

Epoch 16/32
----------
Time elapsed 49m 24s
train Loss: 0.3131 Acc: 0.9419
Time elapsed 49m 37s
test Loss: 1.4141 Acc: 0.8429
Optimizer learning rate : 0.0100000

Epoch 17/32
----------
Time elapsed 52m 20s
train Loss: 0.3541 Acc: 0.9381
Time elapsed 52m 32s
test Loss: 1.2417 Acc: 0.8590
Optimizer learning rate : 0.0100000

Epoch 18/32
----------
Time elapsed 55m 15s
train Loss: 0.4051 Acc: 0.9356
Time elapsed 55m 27s
test Loss: 1.8392 Acc: 0.8093
Optimizer learning rate : 0.0100000

Epoch 19/32
----------
Time elapsed 58m 10s
train Loss: 0.4696 Acc: 0.9252
Time elapsed 58m 22s
test Loss: 0.9729 Acc: 0.8814
Optimizer learning rate : 0.0100000

Epoch 20/32
----------
Time elapsed 61m 4s
train Loss: 0.3701 Acc: 0.9402
Time elapsed 61m 17s
test Loss: 0.9427 Acc: 0.8862
Optimizer learning rate : 0.0100000

Epoch 21/32
----------
Time elapsed 63m 59s
Time elapsed 63m 59s
train Loss: 0.4010 Acc: 0.9371
Time elapsed 64m 12s
test Loss: 1.5368 Acc: 0.8381
Optimizer learning rate : 0.0100000

Epoch 22/32
----------
Time elapsed 66m 54s
train Loss: 0.3097 Acc: 0.9433
Time elapsed 67m 7s
test Loss: 1.3240 Acc: 0.8429
Optimizer learning rate : 0.0100000

Epoch 23/32
----------
Time elapsed 69m 49s
train Loss: 0.3419 Acc: 0.9404
Time elapsed 70m 1s
test Loss: 1.4100 Acc: 0.8317
Optimizer learning rate : 0.0100000

Epoch 24/32
----------
Time elapsed 72m 43s
train Loss: 0.3370 Acc: 0.9400
Time elapsed 72m 55s
test Loss: 3.5217 Acc: 0.6923
Optimizer learning rate : 0.0100000

Epoch 25/32
----------
Time elapsed 75m 38s
train Loss: 0.4341 Acc: 0.9247
Time elapsed 75m 50s
test Loss: 1.4699 Acc: 0.8301
Optimizer learning rate : 0.0100000

Epoch 26/32
----------
Time elapsed 78m 32s
train Loss: 0.5305 Acc: 0.9252
Time elapsed 78m 45s
test Loss: 2.6856 Acc: 0.7131
Optimizer learning rate : 0.0100000

Epoch 27/32
----------
Time elapsed 81m 27s
train Loss: 0.4043 Acc: 0.9373
Time elapsed 81m 39s
test Loss: 0.8255 Acc: 0.8798
Optimizer learning rate : 0.0100000

Epoch 28/32
----------
Time elapsed 84m 22s
train Loss: 0.3580 Acc: 0.9387
```

```
Time elapsed 84m 35s
test Loss: 0.6987 Acc: 0.8990
Optimizer learning rate : 0.0100000

Epoch 29/32
----------
Time elapsed 87m 17s
train Loss: 0.3286 Acc: 0.9392
Time elapsed 87m 29s
test Loss: 2.6941 Acc: 0.7436
Optimizer learning rate : 0.0100000

Epoch 30/32
----------
Time elapsed 90m 11s
train Loss: 0.3171 Acc: 0.9406
Time elapsed 90m 24s
test Loss: 1.7229 Acc: 0.8093
Optimizer learning rate : 0.0100000

Epoch 31/32
----------
Time elapsed 93m 6s
train Loss: 0.3346 Acc: 0.9406
Time elapsed 93m 18s
test Loss: 2.3639 Acc: 0.7532
Optimizer learning rate : 0.0100000

Epoch 32/32
----------
Time elapsed 96m 0s
train Loss: 0.5060 Acc: 0.9279
Time elapsed 96m 13s
test Loss: 1.0063 Acc: 0.8782
Optimizer learning rate : 0.0100000

Training complete in 96m 13s
Best val Acc: 0.900641
```

```
"""
probs, classes = predict(image_path, model)
print(probs)
print(classes)
"""
```

```
'\nprobs, classes = predict(image_path, model)\nprint(probs)\nprint(classes)\n'
```

```
model_ft, input_size = initialize_model(model_name, 102, feature_extract, use_pretrained=True
# GPU模式
model_ft = model_ft.to(device)

# 保存文件的名字
filename='checkpoint.pth'

# 加载模型
checkpoint = torch.load(filename)
best_acc = checkpoint['best_acc']
model_ft.load_state_dict(checkpoint['state_dict'])
```

```
<All keys matched successfully>
```

```
def imshow(image, ax=None, title=None):
    """展示数据"""
    if ax is None:
        fig, ax = plt.subplots()

    # 颜色通道还原
    image = np.array(image).transpose((1, 2, 0))

    # 预处理还原
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    image = std * image + mean
    image = np.clip(image, 0, 1)

    ax.imshow(image)
    ax.set_title(title)

    return ax
```

```
# 得到一个batch的测试数据
dataiter = iter(dataloaders['val'])
images, labels = dataiter.next()

model_ft.eval()

if train_on_gpu:
    output = model_ft(images.cuda())
else:
    output = model_ft(images)
```

```
output.shape
```

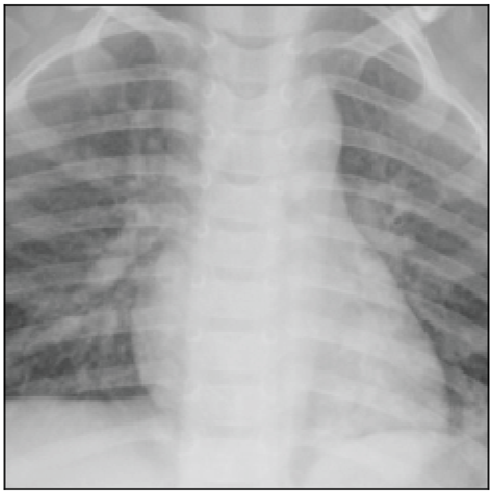
```
torch.Size([16, 102])
```

```
def im_convert(tensor):
    """ 展示数据"""
    image = tensor.to("cpu").clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1,2,0)
    image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456, 0.406))
    image = image.clip(0, 1)

    return image


_, preds_tensor = torch.max(output, 1)

preds = np.squeeze(preds_tensor.numpy()) if not train_on_gpu else np.squeeze(preds_tensor.cpu
preds
```

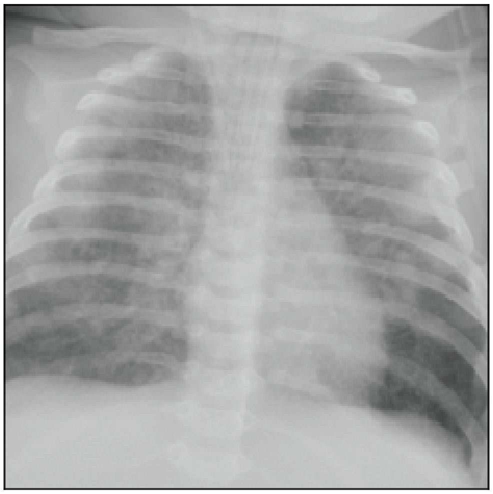
```
 array([1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1])
```

```
cat_to_name={'0':'NORMAL',
             '1': 'PNEUMONIA'}
             # # {'NORMAL': 0 , 'PNEUMONIA': 1}


fig=plt.figure(figsize=(20, 20))
columns =4
rows = 2

for idx in range (columns*rows):
    ax = fig.add_subplot(rows, columns, idx+1, xticks=[], yticks=[])
    plt.imshow(im_convert(images[idx]))
    ax.set_title("{} ({})".format(cat_to_name[str(preds[idx])], cat_to_name[str(labels[idx].i
                 color=("green" if cat_to_name[str(preds[idx])]==cat_to_name[str(labels[idx].
plt.show()
```

PNEUMONIA (PNEUMONIA)

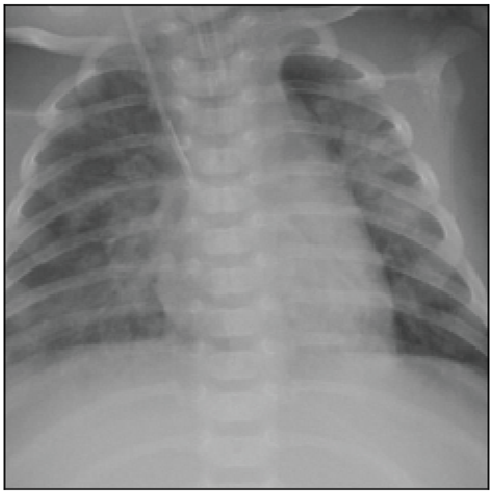


NORMAL (PNEUMONIA)

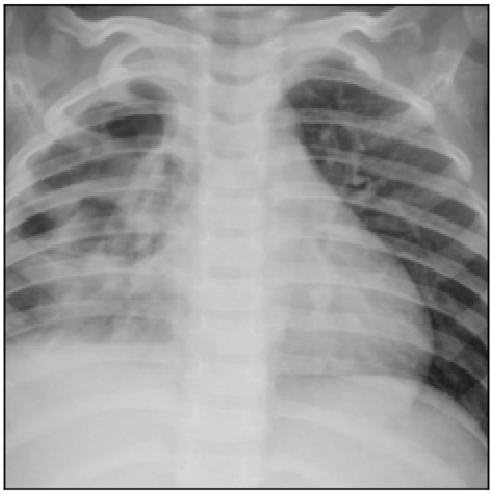


NO

NORMAL (PNEUMONIA)



PNEUMONIA (PNEUMONIA)



NO