卷积神经网络

项目：为小狗识别应用编写算法

在此 notebook 中，我们已经为你提供一些模板代码，要成功完成此项目，你需要实现其他功能。除此之外，不需要修改所提供的代码。标题中以（**实现**）开头的部分表明你必须在下面的代码块中提供其他功能。我们会在每个部分提供说明，并在以“TODO”开头的代码块中提供实现细节。请仔细阅读说明。

> **注意**：完成所有代码实现后，最后需要将 iPython Notebook 导出为 HTML 文档。在将 notebook 导出为 HTML 前，请运行所有代码单元格，使审阅者能够查看最终实现和输出结果。然后导出 notebook，方法是：使用顶部的菜单并依次转到**文件 -> 下载为 -> HTML (.html)**。提交内容应该同时包含此 notebook 和完成的文档。

除了实现代码之外，还需要回答与项目和代码实现相关的问题。请仔细阅读每个问题，并在**答案：**下方的文本框中填写答案。我们将根据每个问题的答案以及实现代码评估你提交的项目。

> **注意：**可以通过 **Shift + Enter** 键盘快捷键执行代码和标记单元格，并且可以通过双击单元格进入编辑模式，编辑标记单元格。

审阅标准还包含可选的“锦上添花”建议，可以指导你在满足最低要求的基础上改进项目。如果你打算采纳这些建议，则应该在此 Jupyter notebook 中添加代码。

为何要完成这道练习

在此 notebook 中，你将开发一种可用于移动应用或网络应用的算法。最终你的代码将能够将任何用户提供的图像作为输入。如果从图像中检测出小狗，该算法将大致识别出小狗品种。如果检测出人脸，该算法将大致识别出最相似的小狗品种。下图显示了最终项目的潜在示例输出（但是我们希望每个学员的算法行为都不一样。）。

Sample Dog Output

在此实际应用中，你需要将一系列模型整合到一起并执行不同的任务；例如，检测图中人脸的算法与推理小狗品种的 CNN 将不一样。有很多地方都可能会出错，没有什么完美的算法。即使你的答案不完美，也可以创造有趣的用户体验。

项目规划

我们将此 notebook 分成了几个独立的步骤。你可以通过以下链接浏览此 notebook。

- 第 0 步：导入数据集
- 第 1 步：检测人脸
- 第 2 步：检测小狗
- 第 3 步：（从头开始）创建分类小狗品种的 CNN
- 第 4 步：（使用迁移学习）创建分类小狗品种的 CNN
- 第 5 步：编写算法
- 第 6 步：测试算法

第 0 步：导入数据集

首先下载人脸和小狗数据集：

注意：如果你使用的是 Udacity 工作区，你*不需要重新下载它们 - 它们可以在 `/ data` 文件夹中找到，如下面的单元格所示。

- 下载小狗数据集 (https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip)。解压文件并将其放入此项目的主目录中，位置为 `/dog_images` 。

In [1]:
```
import numpy as np
from glob import glob

# load filenames for human and dog images
human_files = np.array(glob("/data/lfw/*/*"))
dog_files = np.array(glob("/data/dog_images/*/*/*"))

# print number of images in each dataset
print('There are %d total human images.' % len(human_files))
print('There are %d total dog images.' % len(dog_files))
```

```
There are 13233 total human images.
There are 8351 total dog images.
```

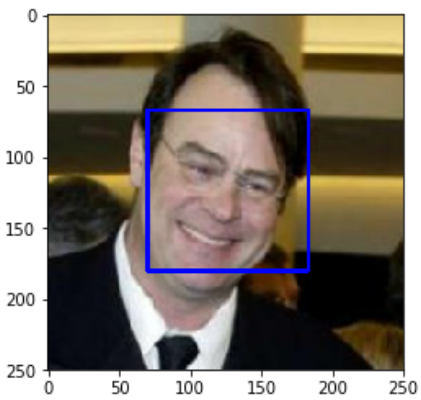
第 1 步：检测人脸

在此部分，我们使用 OpenCV 的哈儿特征级联分类器 (http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html) 检测图像中的人脸。

OpenCV 提供了很多预训练的人脸检测器，它们以 XML 文件的形式存储在 github (https://github.com/opencv/opencv/tree/master/data/haarcascades) 上。我们下载了其中一个检测器并存储在 `haarcascades` 目录中。在下个代码单元格中，我们将演示如何使用此检测器从样本图像中检测人脸。

In [2]:
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

# load color (BGR) image
img = cv2.imread(human_files[0])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

```
Number of faces detected: 1
```



在使用任何人脸检测器之前，标准做法是将图像转换为灰阶图像。 `detectMultiScale` 函数会执行存储在 `face_cascade` 中的分类器并将灰阶图像当做参数。

在上述代码中， `faces` 是一个包含检测到的人脸的 numpy 数组，其中每行对应一张检测到的人脸。检测到的每张人脸都是一个一维数组，其中有四个条目，分别指定了检测到的人脸的边界框。数组中的前两个条目（在上述代码中提取为 `x` 和 `y` ）指定了左上角边界框的水平和垂直位置。数组中的后两个条目（提取为 `w` 和 `h` ）指定了边界框的宽和高。

编写人脸检测器

我们可以编写一个函数，如果在图像中检测到人脸，该函数将返回 `True` ，否则返回 `False` 。此函数称为 `face_detector` ，参数为图像的字符串文件路径，并出现在以下代码块中。

In [3]:
```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

（实现）评估人脸检测器

问题 1：使用以下代码单元格测试 `face_detector` 函数的性能。

- 对于 `human_files` 中的前100 张图像，有多少图像检测到了人脸？
- 对于 `dog_files` 中的前100 张图像，有多少图像检测到了人脸？

理想情况下，我们希望所有人脸图像都能检测到人脸，所有小狗图像都不能检测到人脸。我们的算法不能满足此目标，但是依然达到了可接受的水平。我们针对每个数据集的前 100 张图像提取出文件路径，并将它们存储在 numpy 数组 `human_files_short` 和 `dog_files_short` 中。

答案： 98% 和 17% （请在此单元格中填写结果和/或百分比）

In [4]:
```
from tqdm import tqdm

human_files_short = human_files[:100]
dog_files_short = dog_files[:100]

#-#-# Do NOT modify the code above this line. #-#-#

## TODO: Test the performance of the face_detector algorithm
## on the images in human_files_short and dog_files_short.
print(np.mean([face_detector(human) for human in human_files_short]))
print(np.mean([face_detector(dog) for dog in dog_files_short]))
```

```
0.98
0.17
```

建议在算法中使用 OpenCV 的人脸检测器来检测人脸图像，但是你也可以尝试其他方法，尤其是利用深度学习的方法:)。请在以下代码单元格中设计并测试你的人脸检测算法。如果你打算完成此_可选_任务，请报告 `human_files_short` 和 `dog_files_short` 的效果。

In [ ]:
```
### (Optional)
### TODO: Test performance of anotherface detection algorithm.
### Feel free to use as many code cells as needed.
```

第 2 步：检测小狗

在此部分，我们使用预训练的模型 (http://pytorch.org/docs/master/torchvision/models.html)检测图像中的小狗。

获取预训练的 VGG-16 模型

以下代码单元格会下载 VGG-16 模型以及在 ImageNet (http://www.image-net.org/) 上训练过的权重，ImageNet 是一个非常热门的数据集，可以用于图像分类和其他视觉任务。ImageNet 包含 1000 万以上的 URL，每个都链接到包含某个对象的图像，这些对象分成了 1000 个类别 (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a)。

In [5]:
```
import torch
import torchvision.models as models

# define VGG16 model
VGG16 = models.vgg16(pretrained=True)

# check if CUDA is available
use_cuda = torch.cuda.is_available()

# move model to GPU if CUDA is available
if use_cuda:
    VGG16 = VGG16.cuda()
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /roo
t/.torch/models/vgg16-397923af.pth
100%|██████████| 553433881/553433881 [00:25<00:00, 21324700.23it/s]
```

如果给定一张图像，此预训练的 VGG-16 模型能够针对图像中的对象返回预测结果（属于 ImageNet 中的 1000 个潜在类别之一）。

（实现）使用预训练的模型做出预测

在下个代码单元格中，你将编写一个函数，它将图像路径（例如 `'dogImages/train/001.Affenpinscher/Affenpinscher_00001.jpg'`）当做输入，并返回预训练 VGG-16 模型预测的 ImageNet 类别对应的索引。输出应该始终是在 0 - 999（含）之间的整数。

在编写该函数之前，请阅读此 PyTorch 文档 (http://pytorch.org/docs/stable/torchvision/models.html)，了解如何针对预训练的模型预处理张量。

In [6]:
```
from PIL import Image
import torchvision.transforms as transforms

def VGG16_predict(img_path):
    '''
    Use pre-trained VGG-16 model to obtain index corresponding to
    predicted ImageNet class for image at specified path

    Args:
        img_path: path to an image

    Returns:
        Index corresponding to VGG-16 model's prediction
    '''

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image
    transform = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485,0.456,0.406],
                                 std=[0.229,0.224,0.225])
                            ])
    torch.no_grad()
    img = Image.open(img_path)
    img = transform(img).unsqueeze(0)
    if use_cuda:
        img = img.cuda()

    output = VGG16(img)
    _, predicted = torch.max(output, 1)

    return predicted[0] # predicted class index
```

（实现）编写小狗检测器

查看该字典 (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a)后，你将发现：小狗对应的类别按顺序排列，对应的键是 151-268（含），包含从 `'Chihuahua'` 到 `'Mexican hairless'` 的所有类别。因此，要检查预训练的 VGG-16 模型是否预测某个图像包含小狗，我们只需检查预训练模型预测的索引是否在 151 - 268（含）之间。

请根据这些信息完成下面的 `dog_detector` 函数，如果从图像中检测出小狗，它将返回 `True` （否则返回 `False`）。

In [7]:
```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    prediction = VGG16_predict(img_path)
    return ((prediction <= 268) & (prediction >= 151))
```

（实现）评估小狗检测器

问题 2：在以下代码单元格中测试 `dog_detector` 的效果。

- 对于 `human_files_short` 中的图像，有多少图像检测到了小狗？
- 对于 `dog_files_short` 中的图像，有多少图像检测到了小狗？

答案：2% 和 99%

In [8]:
```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.
print(np.mean([dog_detector(human) for human in human_files_short]))
print(np.mean([dog_detector(dog) for dog in dog_files_short]))
```

```
0.02
0.99
```

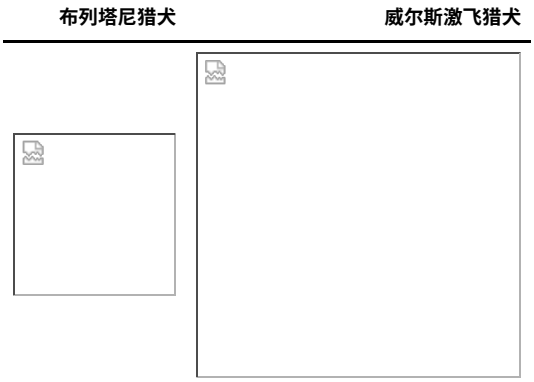
建议在算法中使用 VGG-16 检测小狗图像，但是你也可以尝试其他预训练的网络（例如 Inception-v3 (http://pytorch.org/docs/master/torchvision/models.html#inception-v3)、ResNet-50 (http://pytorch.org/docs/master/torchvision/models.html#id3) 等）。请在以下代码单元格中测试其他预训练的 PyTorch 模型。如果你打算完成此_可选_任务，请报告 `human_files_short` 和 `dog_files_short` 的效果。

In [10]:
```
### (Optional)
### TODO: Report the performance of another pre-trained network.
### Feel free to use as many code cells as needed.
```

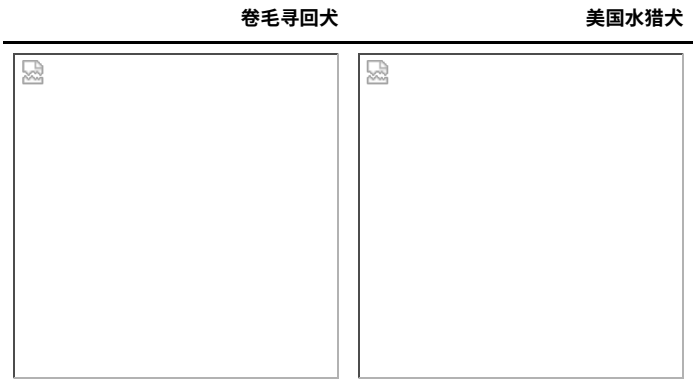
第 3 步：（从头开始）创建分类小狗品种的 CNN

创建好从图像中检测人脸和小狗的函数后，我们需要预测图像中的小狗品种。在这一步，你需要创建一个分类小狗品种的 CNN。你必须从头创建一个 CNN（因此暂时不能使用迁移学习。），并且测试准确率必须至少达到 10%。在此 notebook 的第 4 步，你将使用迁移学习创建 CNN，并且能够获得很高的准确率。

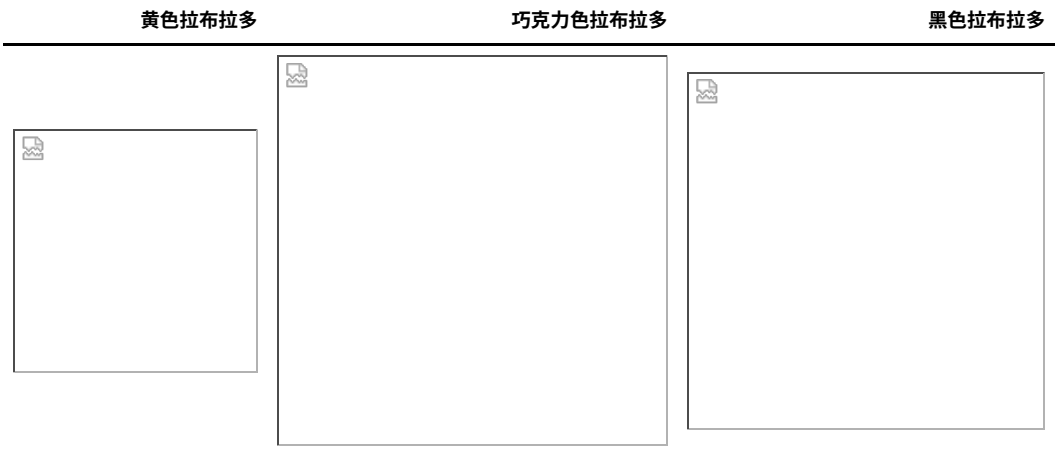
预测图中小狗的品种是一项非常难的挑战。说实话，即使是我们人类，也很难区分布列塔尼猎犬和威尔斯激飞猎犬。



还有很多其他相似的狗品种（例如卷毛寻回犬和美国水猎犬）。



同理，拉布拉多有黄色、巧克力色和黑色品种。基于视觉的算法需要克服这种同一类别差异很大的问题，并决定如何将所有这些不同肤色的小狗分类为相同的品种。



随机猜测的效果很差：除了类别数量不太平衡之外，随机猜测的正确概率约为 1/133，准确率不到 1%。

在深度学习领域，实践比理论知识靠谱得到。请尝试多种不同的架构，并相信你的直觉。希望你可以从学习中获得乐趣！

In [9]:
```
import torchvision
import os
from torchvision import datasets

from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

### TODO: Write data loaders for training, validation, and test sets
## Specify appropriate transforms, and batch_sizes

transform = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomRotation(10),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485,0.456,0.406],
                                 std=[0.229,0.224,0.225])
                               ])

data_dir = '/data/dog_images'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          transform=transform)
                  for x in ['train', 'valid', 'test']}
loaders_scratch = {x: torch.utils.data.DataLoader(image_datasets[x], batch_s
ize=4,
                                                  shuffle=True, num_workers=4)
              for x in ['train', 'valid', 'test']}

## Plot images after loading
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'valid', 'test
']}
class_names = image_datasets['train'].classes

def imshow(inp, title=None):
    """Imshow for Tensor."""
    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)  # pause a bit so that plots are updated


# Get a batch of training data
#inputs, classes = next(iter(loaders_scratch['train']))

# Make a grid from batch
#out = torchvision.utils.make_grid(inputs)

#imshow(out, title=[class_names[x] for x in classes])
```

问题 3：描述你所选的数据预处理流程。

- 你是如何调整图像大小的（裁剪、拉伸等）？你选择的输入张量大小是多少，为何？
- 你是否决定增强数据集？如果是，如何增强（平移、翻转、旋转等）？如果否，理由是？

答案：对图像做了剪裁，旋转和水平翻转，增强数据集。输入张量的大小是224×224×3，适合VGG16等深度神经网络。

（实现）模型架构

创建分类小狗品种的 CNN。使用以下代码单元格中的模板。

In [10]:
```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN
        self.conv1 = nn.Conv2d(3, 16, 2)
        self.maxpool = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(16, 32, 2)
        self.conv3 = nn.Conv2d(32, 64, 2)
        self.avgpool = nn.AvgPool2d(2)
        self.fc1 = nn.Linear(13*13*64, 133)


    def forward(self, x):
        ## Define forward behavior
        x = self.maxpool(F.relu(self.conv1(x)))
        x = self.maxpool(F.relu(self.conv2(x)))
        x = self.maxpool(F.relu(self.conv3(x)))
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)

        return x

#-#-# You so NOT have to modify the code below this line. #-#-#

# instantiate the CNN
model_scratch = Net()

# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

问题 4：列出获得最终 CNN 结构的步骤以及每步的推理过程。

答案： 采用卷积-池化-卷积-池化-卷积-池化-平均池化-全连接的步骤。采用了三层卷积神经网络，并用了平均池化和全连接层。

（实现）指定损失函数和优化器

在下个代码单元格中指定损失函数 (http://pytorch.org/docs/stable/nn.html#loss-functions)和优化器 (http://pytorch.org/docs/stable/optim.html)。在下面将所选的损失函数另存为 `criterion_scratch` ，并将优化器另存为 `optimizer_scratch` 。

In [11]:
```
import torch.optim as optim

### TODO: select loss function
criterion_scratch = nn.CrossEntropyLoss()

### TODO: select optimizer
optimizer_scratch = optim.SGD(model_scratch.parameters(), lr=0.01)
```

（实现）训练和验证模型

在以下代码单元格中训练和验证模型。将最终模型参数 (http://pytorch.org/docs/master/notes/serialization.html)保存到以下文件路径： `'model_scratch.pt'` 。

In [12]:
```
def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_pat
h):
    """returns trained model"""
    # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf

    for epoch in range(1, n_epochs+1):
        # initialize variables to monitor training and validation loss
        train_loss = 0.0
        valid_loss = 0.0

        ###################
        # train the model #
        ###################
        model.train()
        for batch_idx, (data, target) in enumerate(loaders['train']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            ## find the loss and update the model parameters accordingly
            ## record the average training loss, using something like
            ## train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data
- train_loss))
            optimizer.zero_grad()
            output = model(data)
            _, preds = torch.max(output, 1)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data -
train_loss))


        ######################
        # validate the model #
        ######################
        model.eval()
        for batch_idx, (data, target) in enumerate(loaders['valid']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            ## update the average validation loss
            output = model(data)
            _, preds = torch.max(output, 1)
            loss = criterion(output, target)

            valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data -
valid_loss))

        # print training/validation statistics
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.
format(
            epoch,
            train_loss,
            valid_loss
            ))

        ## TODO: save the model if validation loss has decreased
        if valid_loss <= valid_loss_min:
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving mo
del ...'.format(
                valid_loss_min,
                valid_loss))
            torch.save(model.state_dict(), save_path)
            valid_loss_min = valid_loss
```

```
Epoch: 1        Training Loss: 4.819863         Validation Loss: 4.681741
Validation loss decreased (inf --> 4.681741).  Saving model ...
Epoch: 2        Training Loss: 4.621337         Validation Loss: 4.621829
Validation loss decreased (4.681741 --> 4.621829).  Saving model ...
Epoch: 3        Training Loss: 4.507945         Validation Loss: 4.547992
Validation loss decreased (4.621829 --> 4.547992).  Saving model ...
Epoch: 4        Training Loss: 4.439319         Validation Loss: 4.477179
Validation loss decreased (4.547992 --> 4.477179).  Saving model ...
Epoch: 5        Training Loss: 4.391300         Validation Loss: 4.417311
Validation loss decreased (4.477179 --> 4.417311).  Saving model ...
Epoch: 6        Training Loss: 4.339286         Validation Loss: 4.545910
Epoch: 7        Training Loss: 4.296169         Validation Loss: 4.425743
Epoch: 8        Training Loss: 4.231334         Validation Loss: 4.434345
Epoch: 9        Training Loss: 4.203217         Validation Loss: 4.329884
Validation loss decreased (4.417311 --> 4.329884).  Saving model ...
Epoch: 10       Training Loss: 4.152175         Validation Loss: 4.354720
Epoch: 11       Training Loss: 4.104116         Validation Loss: 4.395571
Epoch: 12       Training Loss: 4.047976         Validation Loss: 4.321709
Validation loss decreased (4.329884 --> 4.321709).  Saving model ...
Epoch: 13       Training Loss: 4.009334         Validation Loss: 4.318789
Validation loss decreased (4.321709 --> 4.318789).  Saving model ...
Epoch: 14       Training Loss: 3.961818         Validation Loss: 4.320961
Epoch: 15       Training Loss: 3.910287         Validation Loss: 4.333254
Epoch: 16       Training Loss: 3.865785         Validation Loss: 4.287566
Validation loss decreased (4.318789 --> 4.287566).  Saving model ...
Epoch: 17       Training Loss: 3.842635         Validation Loss: 4.227672
Validation loss decreased (4.287566 --> 4.227672).  Saving model ...
Epoch: 18       Training Loss: 3.806714         Validation Loss: 4.233178
Epoch: 19       Training Loss: 3.768717         Validation Loss: 4.250889
Epoch: 20       Training Loss: 3.711874         Validation Loss: 4.206095
Validation loss decreased (4.227672 --> 4.206095).  Saving model ...
Epoch: 21       Training Loss: 3.692058         Validation Loss: 4.233789
Epoch: 22       Training Loss: 3.650938         Validation Loss: 4.187511
Validation loss decreased (4.206095 --> 4.187511).  Saving model ...
Epoch: 23       Training Loss: 3.598037         Validation Loss: 4.172476
Validation loss decreased (4.187511 --> 4.172476).  Saving model ...
Epoch: 24       Training Loss: 3.589736         Validation Loss: 4.256717
Epoch: 25       Training Loss: 3.536966         Validation Loss: 4.237239
```

（实现）测试模型

在小狗图像测试数据集上尝试模型。在以下代码单元格中计算并输出测试损失和准确率。确保测试准确率高于 10%。

In [14]:
```
def test(loaders, model, criterion, use_cuda):

    # monitor test loss and accuracy
    test_loss = 0.
    correct = 0.
    total = 0.

    model.eval()
    for batch_idx, (data, target) in enumerate(loaders['test']):
        # move to GPU
        if use_cuda:
            data, target = data.cuda(), target.cuda()
        # forward pass: compute predicted outputs by passing inputs to the m
odel
        output = model(data)
        # calculate the loss
        loss = criterion(output, target)
        # update average test loss
        test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_l
oss))
        # convert output probabilities to predicted class
        pred = output.data.max(1, keepdim=True)[1]
        # compare predictions to true label
        correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu
().numpy())
        total += data.size(0)

    print('Test Loss: {:.6f}\n'.format(test_loss))

    print('\nTest Accuracy: %2d%% (%2d/%2d)' % (
        100. * correct / total, correct, total))

# call test function
test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 4.063982


Test Accuracy: 10% (86/836)
```

第 4 步：（使用迁移学习）创建分类小狗品种的 CNN

现在你将使用迁移学习创建能够识别图中小狗品种的 CNN。你的 CNN 必须在测试集上至少达到 60% 的准确率。

（实现）为小狗数据集指定数据加载器

在以下代码单元格中编写三个独立的数据加载器 (http://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader)，用于训练、验证和测试小狗图像数据集（分别位于 `dogImages/train`、`dogImages/valid` 和 `dogImages/test` 下）。

你也可以使用在从头开始创建 CNN 这一步时创建的同一数据加载器。

In [15]:
```
## TODO: Specify data loaders
loaders_transfer = loaders_scratch
```

（实现）模型架构

使用迁移学习创建分类小狗品种的 CNN。在以下代码单元格中填写代码并将初始化的模型另存为变量 `model_transfer` 。

In [17]:
```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
model_pretrained = models.vgg16(pretrained=True)

# print out the model structure
#print(model_pretrained)

# Freeze training for all "features" layers
for param in model_pretrained.features.parameters():
    param.requires_grad = False

# add last linear layer (n_inputs -> 133 dog classes)
n_inputs = model_pretrained.classifier[6].in_features
last_layer = nn.Linear(n_inputs, 133)
model_pretrained.classifier[6] = last_layer

model_transfer = model_pretrained

# print out the model structure
print(model_transfer)

if use_cuda:
    model_transfer = model_transfer.cuda()
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
e=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
e=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
e=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=4096, out_features=133, bias=True)
  )
)
```

问题 5：列出获得最终 CNN 结构的步骤以及每步的推理过程。解释为何该结构适合解决手头的问题。

答案： 根据迁移学习，将VGG16的最后一层卷积层作为输入，应用全连接层做输出层。选用VGG16是因为该模型在分类问题中准确率比较高，利用迁移学习可以获得了良好的模型和参数，大大减少训练时间。VGG16是深层网络，包含多个卷积层，池化层和ReLU，能够准确提取图片中的特征，并减少过拟合。早期尝试不成功的原因：（1）模型结构相对简单，没有迁移学习中的预训练模型好；（2）应该使用更大的数据集来训练模型参数。

（实现）指定损失函数和优化器

在下个代码单元格中指定损失函数 (http://pytorch.org/docs/master/nn.html#loss-functions)和优化器 (http://pytorch.org/docs/master/optim.html)。在下面将所选的损失函数另存为 `criterion_transfer` ，并将优化器另存为 `optimizer_transfer` 。

In [18]:
```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer.classifier.parameters(), lr=0.
001)
```

（实现）训练和验证模型。

在以下代码单元格中训练和验证模型。将最终模型参数 (http://pytorch.org/docs/master/notes/serialization.html)保存到以下文件路径： 'model_transfer.pt' 。

In [19]:
```
# train the model
n_epochs = 5
model_transfer = train(n_epochs, loaders_transfer, model_transfer, optimizer
_transfer, criterion_transfer, use_cuda, 'model_transfer.pt')

# load the model that got the best validation accuracy (uncomment the line b
elow)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

```
Epoch: 1        Training Loss: 2.602047         Validation Loss: 1.478154
Validation loss decreased (inf --> 1.478154).  Saving model ...
Epoch: 2        Training Loss: 1.425949         Validation Loss: 1.187029
Validation loss decreased (1.478154 --> 1.187029).  Saving model ...
Epoch: 3        Training Loss: 1.226151         Validation Loss: 1.090535
Validation loss decreased (1.187029 --> 1.090535).  Saving model ...
Epoch: 4        Training Loss: 1.177853         Validation Loss: 1.070166
Validation loss decreased (1.090535 --> 1.070166).  Saving model ...
Epoch: 5        Training Loss: 1.081485         Validation Loss: 0.889808
Validation loss decreased (1.070166 --> 0.889808).  Saving model ...
```

（实现）测试模型

在小狗图像测试数据集上尝试模型。在以下代码单元格中计算并输出测试损失和准确率。确保测试准确率高于 60%。

In [20]:
```
test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 1.024484


Test Accuracy: 70% (590/836)
```

（实现）使用模型预测小狗品种

编写一个函数，它会将图像路径作为输入，并返回模型预测的小狗品种（ Affenpinscher 、 Afghan hound 等）。

In [21]:
```
### TODO: Write a function that takes a path to an image as input
### and returns the dog breed that is predicted by the model.

data_transfer = image_datasets
# list of class names by index, i.e. a name can be accessed like class_names
[0]
class_names = [item[4:].replace("_", " ") for item in data_transfer['train
'].classes]

def predict_breed_transfer(img_path):
    # load the image and return the predicted breed
    transform = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485,0.456,0.406],
                                 std=[0.229,0.224,0.225])
                               ])
    torch.no_grad()
    img = Image.open(img_path)
    img = transform(img).unsqueeze(0)
    if use_cuda:
        img = img.cuda()

    output = model_transfer(img)
    _, predicted = torch.max(output, 1)

    return class_names[predicted[0]]
```

第 5 步：编写算法

编写一个算法，它会将图像的文件路径作为输入，并首先判断图像中是否包含人脸、小狗，或二者都不含。然后，

- 如果在图像中检测到了**小狗**，则返回预测的品种。
- 如果在图像中检测到了**人脸**，则返回相似的小狗品种。
- 如果二者都没检测到，则输出错误消息。

你可以自己编写从图像中检测人脸和小狗的函数，当然也可以使用上面开发的 `face_detector` 和 `human_detector` 函数。你必须使用在第 4 步创建的 CNN 预测小狗品种。

下面提供了一些示例算法输出，但是你也可以自己设计用户体验。

Sample Human Output

（实现）编写算法

In [22]:
```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.

def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    if dog_detector(img_path):
        print('It is a dog. It looks like ')
        print(predict_breed_transfer(img_path))
    elif face_detector(img_path):
        print('It is human. It looks like ')
        print(predict_breed_transfer(img_path))
    else:
        print('Error! It cannot be identified! ')
```

第 6 步：测试算法

在此部分测试新算法啦。算法认为看起来像哪种小狗？如果你有一只狗，算法能准确预测出小狗的品种吗？如果你有一只猫，算法会错误地认为这只猫是小狗吗？

（实现）在样本图像上测试算法。

至少在计算机上用 6 张图像测试你的算法。你可以使用任何图像。至少测试两张人脸图像和两张小狗图像。

问题 6：结果比你预期的要好吗 :)?还是更糟糕 :(？请对你的算法提出至少三个值得改进的地方。

答案：（三个值得改进的地方）1. 增加模型训练次数，比如20次; 2. 使用交叉验证，进行模型训练; 3. 仔细地调节超参数，比如学习率。

```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.

## suggested code, below
for file in np.hstack((human_files[:3], dog_files[:3])):
    run_app(file)
```

In [23]:
```
for file in np.hstack((human_files[:3], dog_files[:3])):
    run_app(file)
```

```
It is human. It looks like
Beagle
It is human. It looks like
Dachshund
It is human. It looks like
Australian terrier
It is a dog. It looks like
Bullmastiff
It is a dog. It looks like
Mastiff
It is a dog. It looks like
Bullmastiff
```

In [ ]:
```

```