Convolutional Neural Networks

Project: Write an Algorithm for a Dog Identification App

In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'(IMPLEMENTATION)'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section, and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the Jupyter Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

The rubric contains *optional* "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. If you decide to pursue the "Stand Out Suggestions", you should include the code in this Jupyter notebook.

Why We're Here

In this notebook, you will make the first steps towards developing an algorithm that could be used as part of a mobile or web app. At the end of this project, your code will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling. The image below displays potential sample output of your finished project (... but we expect that each student's algorithm will behave differently!).

In this real-world setting, you will need to piece together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed. There are many points of possible failure, and no perfect algorithm exists. Your imperfect solution will nonetheless create a fun user experience!

The Road Ahead

We break the notebook into separate steps. Feel free to use the links below to navigate the notebook.

- Step 0: Import Datasets
- Step 1: Detect Humans
- Step 2: Detect Dogs
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
- Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 5: Write your Algorithm
- Step 6: Test Your Algorithm

Step 0: Import Datasets

Make sure that you've downloaded the required human and dog datasets:

Note: if you are using the Udacity workspace, you *DO NOT* need to re-download these - they can be found in the `/data` folder as noted in the cell below.

- Download the dog dataset (https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip). Unzip the folder and place it in this project's home directory, at the location `/dog_images` .
- Download the human dataset (https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip). Unzip the folder and place it in the home directory, at location `/lfw` .

Note: If you are using a Windows machine, you are encouraged to use 7zip (http://www.7-zip.org/) to extract the folder.

In the code cell below, we save the file paths for both the human (LFW) dataset and dog dataset in the numpy arrays `human_files` and `dog_files` .

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.
```

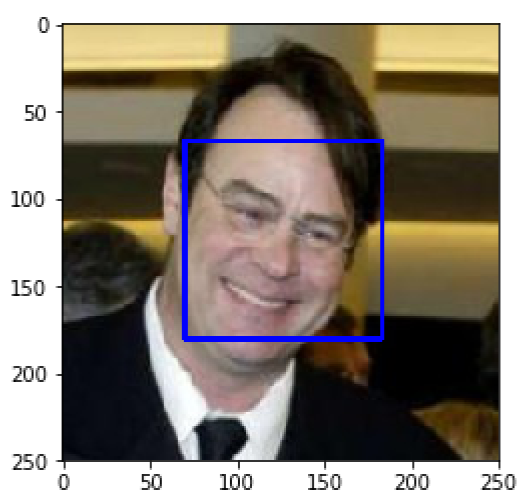
Step 1: Detect Humans

In this section, we use OpenCV's implementation of [Haar feature-based cascade classifiers (http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html)](http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html) to detect human faces in images.

OpenCV provides many pre-trained face detectors, stored as XML files on [github (https://github.com/opencv/opencv/tree/master/data/haarcascades)](https://github.com/opencv/opencv/tree/master/data/haarcascades). We have downloaded one of these detectors and stored it in the `haarcascades` directory. In the next code cell, we demonstrate how to use this detector to find human faces in a sample image.

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

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

# 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
```



Before using any of the face detectors, it is standard procedure to convert the images to grayscale. The `detectMultiScale` function executes the classifier stored in `face_cascade` and takes the grayscale image as a parameter.

In the above code, `faces` is a numpy array of detected faces, where each row corresponds to a detected face. Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. The first two entries in the array (extracted in the above code as `x` and `y`) specify the horizontal and vertical positions of the top left corner of the bounding box. The last two entries in the array (extracted here as `w` and `h`) specify the width and height of the box.

Write a Human Face Detector

We can use this procedure to write a function that returns `True` if a human face is detected in an image and `False` otherwise. This function, aptly named `face_detector`, takes a string-valued file path to an image as input and appears in the code block below.

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
```

(IMPLEMENTATION) Assess the Human Face Detector

Question 1: Use the code cell below to test the performance of the `face_detector` function.

- What percentage of the first 100 images in `human_files` have a detected human face?
- What percentage of the first 100 images in `dog_files` have a detected human face?

Ideally, we would like 100% of human images with a detected face and 0% of dog images with a detected face. You will see that our algorithm falls short of this goal, but still gives acceptable performance. We extract the file paths for the first 100 images from each of the datasets and store them in the numpy arrays `human_files_short` and `dog_files_short`.

Answer: (You can print out your results and/or write your percentages in this cell)

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.
human_face_count = 0
dog_face_count = 0

for file in human_files_short:
    if face_detector(file):
        human_face_count += 1
for file in dog_files_short:
    if face_detector(file):
        dog_face_count += 1

human_percent = human_face_count / len(human_files_short)
dog_percent = dog_face_count / len(dog_files_short)

print(f'Found human faces in about {int(human_percent * 100)}% of human images.')
print(f'Found man-dog faces in about {int(dog_percent * 100)}% of dog images.')
```

```
Found human faces in about 98% of human images.
Found man-dog faces in about 17% of dog images.
```

We suggest the face detector from OpenCV as a potential way to detect human images in your algorithm, but you are free to explore other approaches, especially approaches that make use of deep learning :). Please use the code cell below to design and test your own face detection algorithm. If you decide to pursue this *optional* task, report performance on `human_files_short` and `dog_files_short`.

In [5]:
```
### (Optional)
### TODO: Test performance of another face detection algorithm.
### Feel free to use as many code cells as needed.
"""import dlib

detector = dlib.get_frontal_face_detector()
img = dlib.load_rgb_image(human_files[0])
dets = detector(img, 1)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
#This works on my home machine which has dlib installed"""
```

Out[5]:
```
'import dlib\n\ndetector = dlib.get_frontal_face_detector()\nimg = dlib.load_rg
b_image(human_files[0])\ndets = detector(img, 1)\n\n# display the image, along
with bounding box\nplt.imshow(cv_rgb)\nplt.show()\n#This works on my home machi
ne which has dlib installed'
```

Step 2: Detect Dogs

In this section, we use a [pre-trained model (http://pytorch.org/docs/master/torchvision/models.html)](http://pytorch.org/docs/master/torchvision/models.html) to detect dogs in images.

Obtain Pre-trained VGG-16 Model

The code cell below downloads the VGG-16 model, along with weights that have been trained on [ImageNet (http://www.image-net.org/)](http://www.image-net.org/), a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of [1000 categories (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a)](https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a).

In [6]:

```python
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 [00:26<00:00, 20660685.91it/s]
```

In [7]:
```
print(VGG16.modules)
```

```
<bound method Module.modules of 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=Fa
lse)
    (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=Fa
lse)
    (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_mode=F
alse)
    (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_mode=F
alse)
    (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_mode=F
alse)
  )
  (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=1000, bias=True)
  )
)>
```

Given an image, this pre-trained VGG-16 model returns a prediction (derived from the 1000 possible categories in ImageNet) for the object that is contained in the image.

(IMPLEMENTATION) Making Predictions with a Pre-trained Model

In the next code cell, you will write a function that accepts a path to an image (such as `'dogImages/train/001.Affenpinscher/Affenpinscher_00001.jpg'`) as input and returns the index corresponding to the ImageNet class that is predicted by the pre-trained VGG-16 model. The output should always be an integer between 0 and 999, inclusive.

Before writing the function, make sure that you take the time to learn how to appropriately pre-process tensors for pre-trained models in the [PyTorch documentation (http://pytorch.org/docs/stable/torchvision/models.html)](http://pytorch.org/docs/stable/torchvision/models.html).

In [8]:
```
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

    image = Image.open(img_path).convert('RGB')



    in_transform = transforms.Compose([
                        transforms.Resize((224, 224)),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),
                                             (0.229, 0.224, 0.225))])

    image = in_transform(image)[:3,:,:].unsqueeze(0)
    if use_cuda:
        image = image.to('cuda')
    output = VGG16(image)

    _, index = output.max(-1)

    return int(index)# predicted class index
```

In [9]:
```
print(VGG16_predict('images/American_water_spaniel_00648.jpg'))
```

```
221
```

(IMPLEMENTATION) Write a Dog Detector

While looking at the [dictionary (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a)](https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a), you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from `'Chihuahua'` to `'Mexican hairless'`. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG-16 model, we need only check if the pre-trained model predicts an index between 151 and 268 (inclusive).

Use these ideas to complete the `dog_detector` function below, which returns `True` if a dog is detected in an image (and `False` if not).

In [10]:
```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.

    x = VGG16_predict(img_path)

    return (151 <= x <= 268) # true/false
```

(IMPLEMENTATION) Assess the Dog Detector

Question 2: Use the code cell below to test the performance of your `dog_detector` function.

- What percentage of the images in `human_files_short` have a detected dog?
- What percentage of the images in `dog_files_short` have a detected dog?

Answer:

In [11]:
```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.
human_dog_count = 0
dog_count = 0

for file in human_files_short:
    if dog_detector(file):
        human_dog_count += 1
for file in dog_files_short:
    if dog_detector(file):
        dog_count += 1

human_percent = human_dog_count / len(human_files_short)
dog_percent = dog_count / len(dog_files_short)

print(f'Found dog-men in about {int(human_percent * 100)}% of human images.')
print(f'Found dogs in about {int(dog_percent * 100)}% of dog images.')
```

```
Found dog-men in about 1% of human images.
Found dogs in about 100% of dog images.
```

We suggest VGG-16 as a potential network to detect dog images in your algorithm, but you are free to explore other pre-trained networks (such as [Inception-v3 (http://pytorch.org/docs/master/torchvision/models.html#inception-v3)](http://pytorch.org/docs/master/torchvision/models.html#inception-v3), ResNet-50 (http://pytorch.org/docs/master/torchvision/models.html#id3), etc). Please use the code cell below to test other pre-trained PyTorch models. If you decide to pursue this *optional* task, report performance on `human_files_short` and `dog_files_short`.

In [12]:
```
### (Optional)
### TODO: Report the performance of another pre-trained network.
### Feel free to use as many code cells as needed.
```

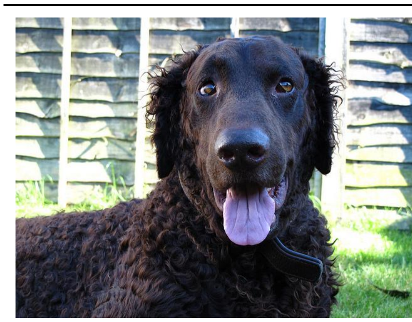
Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. In this step, you will create a CNN that classifies dog breeds. You must create your CNN *from scratch* (so, you can't use transfer learning _yet_!), and you must attain a test accuracy of at least 10%. In Step 4 of this notebook, you will have the opportunity to use transfer learning to create a CNN that attains greatly improved accuracy.

We mention that the task of assigning breed to dogs from images is considered exceptionally challenging. To see why, consider that *even a human* would have trouble distinguishing between a Brittany and a Welsh Springer Spaniel.

Brittany	Welsh Springer Spaniel
	

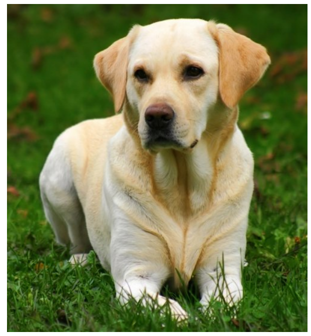
It is not difficult to find other dog breed pairs with minimal inter-class variation (for instance, Curly-Coated Retrievers and American Water Spaniels).

Curly-Coated Retriever	American Water Spaniel
	

Likewise, recall that labradors come in yellow, chocolate, and black. Your vision-based algorithm will have to conquer this high intra-class variation to determine how to classify all of these different shades as the same breed.

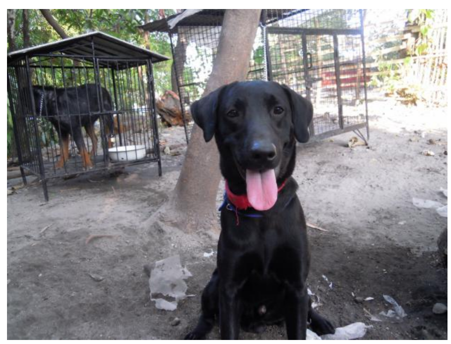
Yellow Labrador | Chocolate Labrador | Black Labrador

- | -



|

|



We also mention that random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imabalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Remember that the practice is far ahead of the theory in deep learning. Experiment with many different architectures, and trust your intuition. And, of course, have fun!

(IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate [data loaders (http://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader)](http://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader) for the training, validation, and test datasets of dog images (located at `dog_images/train` , `dog_images/valid` , and `dog_images/test` , respectively). You may find [this documentation on custom datasets (http://pytorch.org/docs/stable/torchvision/datasets.html)](http://pytorch.org/docs/stable/torchvision/datasets.html) to be a useful resource. If you are interested in augmenting your training and/or validation data, check out the wide variety of [transforms (http://pytorch.org/docs/stable/torchvision/transforms.html?highlight=transform)](http://pytorch.org/docs/stable/torchvision/transforms.html?highlight=transform)!

In [1]:
```
import os
from torchvision import datasets
from torch.utils.data.sampler import SubsetRandomSampler
import numpy as np
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

num_workers = 0
batch_size = 20

# convert data to a normalized torch.FloatTensor
transform = transforms.Compose([
    transforms.Resize(56),
    transforms.CenterCrop(56),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

transform_vat = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_data = datasets.ImageFolder('/data/dog_images/train', transform=transform)
valid_data = datasets.ImageFolder('/data/dog_images/valid', transform=transform_v
test_data = datasets.ImageFolder('/data/dog_images/test', transform=transform_vat

# prepare data loaders
train_loader = torch.utils.data.DataLoader(train_data,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=num_workers)
valid_loader = torch.utils.data.DataLoader(valid_data,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=num_workers)
test_loader = torch.utils.data.DataLoader(test_data,
                                          batch_size=batch_size,
                                          shuffle=False,
                                          num_workers=num_workers)

loaders_scratch = {'train':train_loader, 'valid':valid_loader, 'test':test_loader
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-0b74c0651b6d> in <module>()
     13 
     14 # convert data to a normalized torch.FloatTensor
---> 15 transform = transforms.Compose([
     16     transforms.Resize(56),
     17     transforms.CenterCrop(56),
```

```
NameError: name 'transforms' is not defined
```

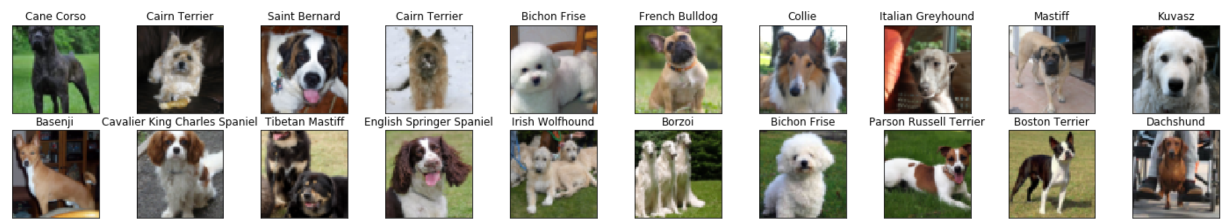
In [14]:

```python
classes = {}
for dog_type in train_data.classes:
    a, b = dog_type.split('.')
    b = b.replace('_', ' ')
    b = b.title()
    classes[int(a)] = b

print(classes)
```

```
{1: 'Affenpinscher', 2: 'Afghan Hound', 3: 'Airedale Terrier', 4: 'Akita', 5:
'Alaskan Malamute', 6: 'American Eskimo Dog', 7: 'American Foxhound', 8: 'Ameri
can Staffordshire Terrier', 9: 'American Water Spaniel', 10: 'Anatolian Shepher
d Dog', 11: 'Australian Cattle Dog', 12: 'Australian Shepherd', 13: 'Australian
Terrier', 14: 'Basenji', 15: 'Basset Hound', 16: 'Beagle', 17: 'Bearded Colli
e', 18: 'Beauceron', 19: 'Bedlington Terrier', 20: 'Belgian Malinois', 21: 'Bel
gian Sheepdog', 22: 'Belgian Tervuren', 23: 'Bernese Mountain Dog', 24: 'Bichon
Frise', 25: 'Black And Tan Coonhound', 26: 'Black Russian Terrier', 27: 'Bloodh
ound', 28: 'Bluetick Coonhound', 29: 'Border Collie', 30: 'Border Terrier', 31:
'Borzoi', 32: 'Boston Terrier', 33: 'Bouvier Des Flandres', 34: 'Boxer', 35: 'B
oykin Spaniel', 36: 'Briard', 37: 'Brittany', 38: 'Brussels Griffon', 39: 'Bull
Terrier', 40: 'Bulldog', 41: 'Bullmastiff', 42: 'Cairn Terrier', 43: 'Canaan Do
g', 44: 'Cane Corso', 45: 'Cardigan Welsh Corgi', 46: 'Cavalier King Charles Sp
aniel', 47: 'Chesapeake Bay Retriever', 48: 'Chihuahua', 49: 'Chinese Crested',
50: 'Chinese Shar-Pei', 51: 'Chow Chow', 52: 'Clumber Spaniel', 53: 'Cocker Spa
niel', 54: 'Collie', 55: 'Curly-Coated Retriever', 56: 'Dachshund', 57: 'Dalmat
ian', 58: 'Dandie Dinmont Terrier', 59: 'Doberman Pinscher', 60: 'Dogue De Bord
eaux', 61: 'English Cocker Spaniel', 62: 'English Setter', 63: 'English Springe
r Spaniel', 64: 'English Toy Spaniel', 65: 'Entlebucher Mountain Dog', 66: 'Fie
ld Spaniel', 67: 'Finnish Spitz', 68: 'Flat-Coated Retriever', 69: 'French Bull
dog', 70: 'German Pinscher', 71: 'German Shepherd Dog', 72: 'German Shorthaired
Pointer', 73: 'German Wirehaired Pointer', 74: 'Giant Schnauzer', 75: 'Glen Of
Imaal Terrier', 76: 'Golden Retriever', 77: 'Gordon Setter', 78: 'Great Dane',
79: 'Great Pyrenees', 80: 'Greater Swiss Mountain Dog', 81: 'Greyhound', 82: 'H
avanese', 83: 'Ibizan Hound', 84: 'Icelandic Sheepdog', 85: 'Irish Red And Whit
e Setter', 86: 'Irish Setter', 87: 'Irish Terrier', 88: 'Irish Water Spaniel',
89: 'Irish Wolfhound', 90: 'Italian Greyhound', 91: 'Japanese Chin', 92: 'Keesh
ond', 93: 'Kerry Blue Terrier', 94: 'Komondor', 95: 'Kuvasz', 96: 'Labrador Ret
riever', 97: 'Lakeland Terrier', 98: 'Leonberger', 99: 'Lhasa Apso', 100: 'Lowc
hen', 101: 'Maltese', 102: 'Manchester Terrier', 103: 'Mastiff', 104: 'Miniatur
e Schnauzer', 105: 'Neapolitan Mastiff', 106: 'Newfoundland', 107: 'Norfolk Ter
rier', 108: 'Norwegian Buhund', 109: 'Norwegian Elkhound', 110: 'Norwegian Lund
ehund', 111: 'Norwich Terrier', 112: 'Nova Scotia Duck Tolling Retriever', 113:
'Old English Sheepdog', 114: 'Otterhound', 115: 'Papillon', 116: 'Parson Russel
l Terrier', 117: 'Pekingese', 118: 'Pembroke Welsh Corgi', 119: 'Petit Basset G
riffon Vendeen', 120: 'Pharaoh Hound', 121: 'Plott', 122: 'Pointer', 123: 'Pome
ranian', 124: 'Poodle', 125: 'Portuguese Water Dog', 126: 'Saint Bernard', 127:
'Silky Terrier', 128: 'Smooth Fox Terrier', 129: 'Tibetan Mastiff', 130: 'Welsh
Springer Spaniel', 131: 'Wirehaired Pointing Griffon', 132: 'Xoloitzcuintli', 1
33: 'Yorkshire Terrier'}
```

In [15]:
```python
import matplotlib.pyplot as plt
%matplotlib inline

# helper function to un-normalize and display an image
def imshow(img):
    img = img / 2 + 0.5  # unnormalize
    plt.imshow(np.transpose(img, (1, 2, 0)))  # convert from Tensor image
```

In [16]:
```python
# obtain one batch of training images
dataiter = iter(train_loader)
images, labels = dataiter.next()
images = images.numpy() # convert images to numpy for display

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
# display 20 images
for idx in np.arange(20):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    label = str(labels[idx]).replace("tensor(", "")
    label = label.replace(")", "")
    ax.set_title(classes[int(label) + 1])
```



Question 3: Describe your chosen procedure for preprocessing the data.

- How does your code resize the images (by cropping, stretching, etc)? What size did you pick for the input tensor, and why?
- Did you decide to augment the dataset? If so, how (through translations, flips, rotations, etc)? If not, why not?

Answer: I picked 56 for the size mainly because that seems to be half of the standard for a lot of CNNs, which is 112. Since the end goal is only 10%, the model needed to favor speed over accuracy. Smaller files made it faster to train so I could run more architectures and see what worked best. I resized and then cropped the images to make sure the dogs' faces were visible. I didn't want to do translations because I want the network to train using the dogs' faces. I didn't do rotations because all of the test images are upright as far as I can tell. I did include flips because the dogs are sometimes oriented right-to-left and sometimes left-to-right.

(IMPLEMENTATION) Model Architecture

Create a CNN to classify dog breed. Use the template in the code cell below.

In [17]:
```
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, 64, 3)
        self.conv2 = nn.Conv2d(64, 64, 4)
        self.conv2_bn = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, 3)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(5 * 5 * 128, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 133)

        self.softmax = nn.Softmax(dim=1)

        self.LR = nn.LeakyReLU(0.2, inplace=True)

        self.drop1 = nn.Dropout(0.35)
        self.drop2 = nn.Dropout(0.2)
        self.drop3 = nn.Dropout(0.2)

    def forward(self, x):
        ## Define forward behavior

        x = self.LR(self.pool(self.conv1(x)))
        x = self.LR(self.pool(self.conv2_bn(self.conv2(x))))
        x = self.LR(self.pool(self.conv3(x)))

        x = x.view(-1, 5 * 5 * 128)
        x = self.drop1(x)
        x = self.LR(self.fc1(x))
        x = self.drop2(x)
        x = self.LR(self.fc2(x))
        x = self.drop3(x)
        x = self.softmax(self.fc3(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()
```

Question 4: Outline the steps you took to get to your final CNN architecture and your reasoning at each step.

Answer: I ended up using about 20 hours of my GPU time just training various models and seeing what worked. I settled on three convolution and pooling layers because it seemed to be enough for the network to start making sense of the images while also quickly getting the dimensions down to size. I then added three fully connected layers and played around with the dimensions of that a little. At first, I had too many of each and it ended up taking too much time to play around with anything. One factor that probably ate up the most time was the optimizer. I finally ended up abandoning SGD because I couldn't find enough consistency to know what I was doing with either the learn rate or the momentum. I was often orders of magnitude off, I think. Adam was a much easier optimizer to work with, and after a lot of diverged models I ended up with a tiny 0.0001 learn rate. This consistently got results, so I stuck with it. And lastly, what burned up hours and hours was figuring out how to manage the overfitting. They training loss would quickly begin to drop much more than the validation loss, but when I added too much dropout the results were sporadic. I decided to add more dropout to the earlier layers because they had many more weights and it would probably speed up training to turn more of them off. I wasn't able to get across the finish line until I used batch normalization, though. Because of the greatly increased training time I only used one. It still took just over 100 epochs, but I managed to get 11% accuracy.

(IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a [loss function (http://pytorch.org/docs/stable/nn.html#loss-functions)](http://pytorch.org/docs/stable/nn.html#loss-functions) and [optimizer (http://pytorch.org/docs/stable/optim.html)](http://pytorch.org/docs/stable/optim.html). Save the chosen loss function as `criterion_scratch`, and the optimizer as `optimizer_scratch` below.

In [18]:
```
import torch.optim as optim

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

### TODO: select optimizer
optimizer_scratch = optim.Adam(model_scratch.parameters(), lr=0.0001)
```

(IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. [Save the final model parameters (http://pytorch.org/docs/master/notes/serialization.html)](http://pytorch.org/docs/master/notes/serialization.html) at filepath `'model_scratch.pt'`.

In [19]:
```
model_scratch.load_state_dict(torch.load('model_scratch.pt'))
```

In [20]:

```
def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path):
    """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 - tr

            optimizer_scratch.zero_grad()

            output = model_scratch(data)
            loss = criterion_scratch(output, target)
            loss.backward()
            optimizer_scratch.step()

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

        ######################
        # 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_scratch(data)
            loss = criterion_scratch(output, target)
            valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valic

        # print training/validation statistics
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.forma
            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 model .
            torch.save(model.state_dict(), 'model_scratch.pt')
            valid_loss_min = valid_loss
```

```python
    # return trained model
    return model


# train the model
model_scratch = train(100, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')

# load the model that got the best validation accuracy
model_scratch.load_state_dict(torch.load('model_scratch.pt'))
```

```
Epoch: 1 	Training Loss: 4.640157 	Validation Loss: 4.805298
Validation loss decreased (inf --> 4.805298).  Saving model ...
Epoch: 2 	Training Loss: 4.640893 	Validation Loss: 4.808669
Epoch: 3 	Training Loss: 4.635621 	Validation Loss: 4.809540
Epoch: 4 	Training Loss: 4.635805 	Validation Loss: 4.805233
Validation loss decreased (4.805298 --> 4.805233).  Saving model ...
Epoch: 5 	Training Loss: 4.635630 	Validation Loss: 4.809662
Epoch: 6 	Training Loss: 4.634561 	Validation Loss: 4.804613
Validation loss decreased (4.805233 --> 4.804613).  Saving model ...
Epoch: 7 	Training Loss: 4.630158 	Validation Loss: 4.806309
Epoch: 8 	Training Loss: 4.631162 	Validation Loss: 4.801905
Validation loss decreased (4.804613 --> 4.801905).  Saving model ...
Epoch: 9 	Training Loss: 4.631605 	Validation Loss: 4.806793


---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-20-e2890529ea4a> in <module>()
     61 # train the model
     62 model_scratch = train(100, loaders_scratch, model_scratch, optimizer_sc
ratch,
---> 63                       criterion_scratch, use_cuda, 'model_scratch.pt')
     64 
     65 # load the model that got the best validation accuracy

<ipython-input-20-e2890529ea4a> in train(n_epochs, loaders, model, optimizer, c
riterion, use_cuda, save_path)
     13         ###################
     14         model.train()
---> 15         for batch_idx, (data, target) in enumerate(loaders['train']):
     16             # move to GPU
     17             if use_cuda:

/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py in __next
__(self)
    262         if self.num_workers == 0:  # same-process loading
    263             indices = next(self.sample_iter)  # may raise StopIteration
--> 264             batch = self.collate_fn([self.dataset[i] for i in indices])
    265             if self.pin_memory:
    266                 batch = pin_memory_batch(batch)

/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py in <listc
omp>(.0)
    262         if self.num_workers == 0:  # same-process loading
    263             indices = next(self.sample_iter)  # may raise StopIteration
--> 264             batch = self.collate_fn([self.dataset[i] for i in indices])
    265             if self.pin_memory:
```

```
    266                     batch = pin_memory_batch(batch)

/opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/
datasets/folder.py in __getitem__(self, index)
     99         """
    100         path, target = self.samples[index]
--> 101         sample = self.loader(path)
    102         if self.transform is not None:
    103             sample = self.transform(sample)

/opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/
datasets/folder.py in default_loader(path)
    145         return accimage_loader(path)
    146     else:
--> 147         return pil_loader(path)
    148 
    149 

/opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/
datasets/folder.py in pil_loader(path)
    128     with open(path, 'rb') as f:
    129         img = Image.open(f)
--> 130         return img.convert('RGB')
    131 
    132 

/opt/conda/lib/python3.6/site-packages/PIL/Image.py in convert(self, mode, matr
ix, dither, palette, colors)
    890         """
    891 
--> 892         self.load()
    893 
    894         if not mode and self.mode == "P":

/opt/conda/lib/python3.6/site-packages/PIL/ImageFile.py in load(self)
    233 
    234                             b = b + s
--> 235                             n, err_code = decoder.decode(b)
    236                             if n < 0:
    237                                 break

KeyboardInterrupt: 
```

(IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 10%.

In [21]:
```
torch.save(model_scratch.state_dict(), 'model_scratch.pt')
```

In [2]:
```python
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 model
        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_loss))
        # 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().nu
        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)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-529b60cf43ff> in <module>()
     29
     30 # call test function
---> 31 test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)

NameError: name 'loaders_scratch' is not defined
```

Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)

You will now use transfer learning to create a CNN that can identify dog breed from images. Your CNN must attain at least 60% accuracy on the test set.

(IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate [data loaders (http://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader)](http://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader) for the training, validation, and test datasets of dog images (located at `dogImages/train` , `dogImages/valid` , and `dogImages/test` , respectively).

If you like, **you are welcome to use the same data loaders from the previous step**, when you created a CNN from scratch.

In [23]:

```python
## TODO: Specify data loaders
import os
import torch
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
import numpy as np
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

num_workers = 0
batch_size = 20

# convert data to a normalized torch.FloatTensor
transform = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

transform_vat = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_data = datasets.ImageFolder('/data/dog_images/train', transform=transform)
valid_data = datasets.ImageFolder('/data/dog_images/valid', transform=transform_v
test_data = datasets.ImageFolder('/data/dog_images/test', transform=transform_vat

# prepare data loaders
train_loader = torch.utils.data.DataLoader(train_data,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=num_workers)
valid_loader = torch.utils.data.DataLoader(valid_data,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=num_workers)
test_loader = torch.utils.data.DataLoader(test_data,
                                          batch_size=batch_size,
                                          shuffle=False,
                                          num_workers=num_workers)

loaders_transfer = {'train':train_loader, 'valid':valid_loader, 'test':test_loade
```

(IMPLEMENTATION) Model Architecture

Use transfer learning to create a CNN to classify dog breed. Use the code cell below, and save your initialized model as the variable `model_transfer` .

In [24]:
```
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim

use_cuda = torch.cuda.is_available()

model_transfer = models.vgg16(pretrained=True)

for param in model_transfer.features.parameters():
    param.requires_grad = False

if use_cuda:
    model_transfer = model_transfer.cuda()

model_transfer.classifier[6] = nn.Linear(in_features=4096, out_features=133, bias

if use_cuda:
    model_transfer.cuda()
```

Question 5: Outline the steps you took to get to your final CNN architecture and your reasoning at each step. Describe why you think the architecture is suitable for the current problem.

Answer: I tried to add unecessary layers to the VGG16 model at first, but it didn't work right. Then I ended up just replacing every full connected layer, but it took forever to train. Finally, I only replaced the last layer and made it output to 133 nodes instead of 1000. I wanted to stick with VGG16 because it already included all the dog classifications and I wanted to preserver that training.

(IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a [loss function (http://pytorch.org/docs/master/nn.html#loss-functions)](http://pytorch.org/docs/master/nn.html#loss-functions) and [optimizer (http://pytorch.org/docs/master/optim.html)](http://pytorch.org/docs/master/optim.html). Save the chosen loss function as `criterion_transfer` , and the optimizer as `optimizer_transfer` below.

In [25]:
```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.Adam(model_transfer.classifier.parameters(), lr=0.000
```

(IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. [Save the final model parameters (http://pytorch.org/docs/master/notes/serialization.html)](http://pytorch.org/docs/master/notes/serialization.html) at filepath `'model_transfer.pt'` .

In [26]:
```python
print(model_transfer)
```

```
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=Fa
lse)
    (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=Fa
lse)
    (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_mode=F
alse)
    (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_mode=F
alse)
    (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_mode=F
alse)
  )
  (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)
  )
)
```

In [27]:
```
# train the model
model_transfer.classifier.train()


def train_transfer(n_epochs, loaders, model, optimizer, criterion, use_cuda, save
    """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.classifier.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 - tr

            optimizer_transfer.zero_grad()

            output = model_transfer(data)
            loss = criterion_transfer(output, target)
            loss.backward()
            optimizer_transfer.step()

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

        ######################
        # validate the model #
        ######################
        model.classifier.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_transfer(data)
            loss = criterion_transfer(output, target)
            valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid

        # print training/validation statistics
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.forma
            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 model .
            torch.save(model.state_dict(), 'model_transfer.pt')
            valid_loss_min = valid_loss
    # return trained model
    return model


n_epochs = 5
model_transfer = train_transfer(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 below)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

```
Epoch: 1        Training Loss: 1.810401         Validation Loss: 0.804112
Validation loss decreased (inf --> 0.804112).  Saving model ...
Epoch: 2        Training Loss: 0.708185         Validation Loss: 0.703448
Validation loss decreased (0.804112 --> 0.703448).  Saving model ...
Epoch: 3        Training Loss: 0.473151         Validation Loss: 0.667949
Validation loss decreased (0.703448 --> 0.667949).  Saving model ...
Epoch: 4        Training Loss: 0.342131         Validation Loss: 0.625627
Validation loss decreased (0.667949 --> 0.625627).  Saving model ...
Epoch: 5        Training Loss: 0.282588         Validation Loss: 0.690844
```

(IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 60%.

In [29]:
```
torch.save(model_transfer.state_dict(), 'model_transfer_backup.pt')
```

In [30]:
```
test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.704237


Test Accuracy: 78% (658/836)
```

(IMPLEMENTATION) Predict Dog Breed with the Model

Write a function that takes an image path as input and returns the dog breed (`Affenpinscher` , `Afghan hound` , etc) that is predicted by your model.

In [31]:
```
### TODO: Write a function that takes a path to an image as input
### and returns the dog breed that is predicted by the model.

# list of class names by index, i.e. a name can be accessed like class_names[0]
def predict_breed_transfer(img_path):
    # load the image and return the predicted breed

    orig_image = Image.open(img_path).convert('RGB')



    in_transform = transforms.Compose([
                        transforms.Resize((224, 224)),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224

    image = in_transform(orig_image)[:3,:,:].unsqueeze(0)
    if use_cuda:
        image = image.to('cuda')
    output = model_transfer(image)

    _, index = output.max(-1)

    #img = cv2.imread(img_path)
    #image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #pixels = np.array(image)
    plt.imshow(orig_image)
    plt.show()
    return print(f"You look like a ...\n{classes[int(index)]}\n\n")
```

Step 5: Write your Algorithm

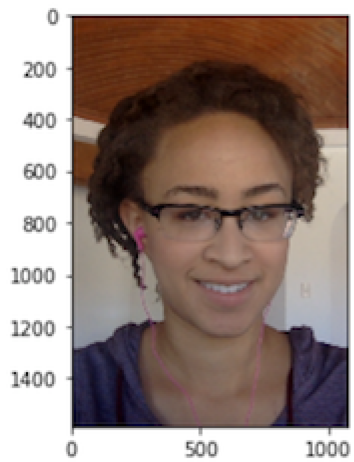
Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide output that indicates an error.

You are welcome to write your own functions for detecting humans and dogs in images, but feel free to use the `face_detector` and `human_detector` functions developed above. You are **required** to use your CNN from Step 4 to predict dog breed.

Some sample output for our algorithm is provided below, but feel free to design your own user experience!

```
hello, human!
```




```
You look like a ...
Chinese_shar-pei
```

(IMPLEMENTATION) Write your Algorithm

In [32]:

```
### 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
    img = cv2.imread(img_path)
    if dog_detector(img_path):
        print('Hai pupper!')
        predict_breed_transfer(img_path)
    elif face_detector(img_path):
        print('Hello hooman!')
        predict_breed_transfer(img_path)
    else:
        print('Oh no!  Not hoomans or puppers ¯\_(ツ)_/¯')
        orig_image = Image.open(img_path).convert('RGB')
        plt.imshow(orig_image)
        plt.show()
```

Step 6: Test Your Algorithm

In this section, you will take your new algorithm for a spin! What kind of dog does the algorithm think that *you* look like? If you have a dog, does it predict your dog's breed accurately? If you have a cat, does it mistakenly think that your cat is a dog?

(IMPLEMENTATION) Test Your Algorithm on Sample Images!

Test your algorithm on at least six images on your computer. Feel free to use any images you like. Use at least two human and two dog images.

Question 6: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of improvement for your algorithm.

Answer: The possible points of improvement would be:

1. Train the model_transfer further to get the classification accuracy up.
2. There's a small chance that a human will get recognized as a dog, but I'm not sure how to make the dog_detector work better.
3. I would be helpful to somehow display a typ image of the predicted dog breed for comparison to the original image.

In [33]:
```python
## 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.
path = 'images/My_Images/*'
my_files = np.array(glob(path))
## suggested code, below
for file in np.hstack(my_files):
    run_app(file)
```

```
Hello hooman!
```


```
You look like a ...
Petit Basset Griffon Vendeen
```