dog_app

April 10, 2020

1 Convolutional Neural Networks

1.1 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.
## 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. Unzip the folder and place it in this project's home directory, at the location `/dog_images`.
- Download the human dataset. 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 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 [10]: 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

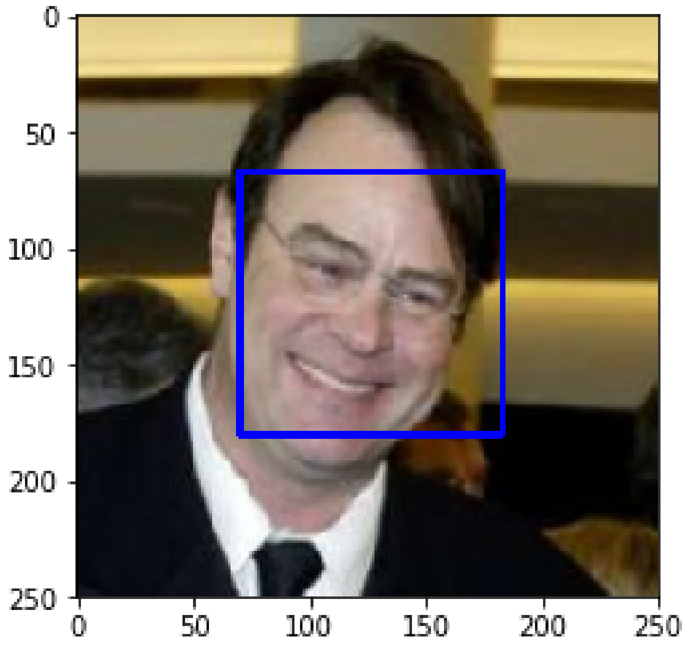
In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

OpenCV provides many pre-trained face detectors, stored as XML files on github. 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 [11]: 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
```



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.

1.1.1 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 [12]: # 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
```

1.1.2 (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 [15]: 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
         def Face_Detection(Files):
             Detected_Faces=0
             No_Faces_Detected=0
             for File in Files:
                 Detection=face_detector(File)
                 if Detection:
                     Detected_Faces=Detected_Faces+1
                 else:
                     No_Faces_Detected=No_Faces_Detected+1


             return print('Number of Faces Detected are {} and The Number of Faces Not Detected

         ## on the images in human_files_short and dog_files_short.

         #HumanFiles_Short
         print('In Humans_short_file:')
         Face_Detection(human_files_short) # This Shows 98 Faces Were Detected In the Humans Dat
```

```
        #DogFiles_short
        print('In Dogs_short_file:')
        Face_Detection(dog_files_short)
```

```
In Humans_short_file:
Number of Faces Detected are 98 and The Number of Faces Not Detected are 2
In Dogs_short_file:
Number of Faces Detected are 17 and The Number of Faces Not Detected are 83
```

1.1.3 The Above Results show 98% Of Human Faces Were Detected from the Human_Files_Short and 17% of Images from dogs_short_file were detected

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

## Step 2: Detect Dogs
In this section, we use a pre-trained model to detect dogs in images.

1.1.4 Obtain Pre-trained VGG-16 Model

The code cell below downloads the VGG-16 model, along with weights that have been trained on ImageNet, 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.

```
In [16]: 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 /root/.torch/models/vgg
100%|| 553433881/553433881 [00:13<00:00, 41079242.66it/s]
```

```
In [17]: print(VGG16)
```

```
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_mode=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_mode=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_mode=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=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.

1.1.5 (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.

```
In [18]: from PIL import Image
         import torchvision.transforms as transforms
```

```
In [19]: #Practice
         IMG=Image.open(dog_files[1])
         print('Type of the File/Image:',type(IMG))
         IMG
```

```
Type of the File/Image: <class 'PIL.JpegImagePlugin.JpegImageFile'>
```

```
Out[19]:
```

```
In [20]: #Image Pre-Processing
         Preprocess=transforms.Compose([
             transforms.Resize(256),
             transforms.CenterCrop(224),
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

         ])

         Input_Tensor=Preprocess(IMG)

         Input_Tensor.shape

Out[20]: torch.Size([3, 224, 224])

In [21]: Final_Tensor=Input_Tensor.unsqueeze(0)
         Final_Tensor.shape

Out[21]: torch.Size([1, 3, 224, 224])

In [101]: 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


#Open The Image using PIL Libraries Image Module
IMAGE=Image.open(img_path)

#Image Pre-Processing Using Transforms Module of Pytorch
PREPROCESS=transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

#Preprocess the Image By Passing the to PREPROCESS PIPELINE
Tensor=PREPROCESS(IMAGE)

#Unsqueeze The Tensor To a Mini Bacth Image
Input_Image=Tensor.unsqueeze(0)

#Move the Image To GPU if it is Available
if torch.cuda.is_available():
    Input_Batch=Input_Image.to('cuda')




#Predict The Class of the input Image Using the Pre-Trained Model VGG16
#Freeze the Weights Since we are Not Training any of them (Fixed Feature extracter

VGG16.eval()

with torch.no_grad():
    output=VGG16(Input_Batch)


    #Get the Predicted Class of the Model Using torc.max(output,1)[1].item()
```

```
            Final_output=torch.max(output,1)[1].item()

        VGG16.train()

        return Final_output # predicted class index
```

```
In [25]: #Example 1
         VGG16_predict(human_files[1])

         #Example 2
         VGG16_predict(dog_files_short[2])
```

```
The Predicted Class of The Image Is: 456
The Predicted Class of The Image Is: 243
```

1.1.6 (IMPLEMENTATION) Write a Dog Detector

While looking at the dictionary, 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 [102]: ### returns "True" if a dog is detected in the image stored at img_path

          def dog_detector(img_path):
              ## TODO: Complete the function.


              #Open The Image using The Pythons PIL Libraries Image Module
              IMAGE=Image.open(img_path)

              #IMage pre-Processing on the Input IMAGE
              Preprocess=transforms.Compose([
                  transforms.Resize(256),
                  transforms.CenterCrop(224),
                  transforms.ToTensor(),
                  transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
              ])

              #Pass The Image to PReprocess Pipeline
              Tensor=Preprocess(IMAGE)

              #Squeeze The Tensor To a Mini Batch
              Tensor=Tensor.unsqueeze(0)
```

```
            if torch.cuda.is_available:
                Tensor=Tensor.to('cuda')

            with torch.no_grad():
                output=VGG16(Tensor)


            Final_Output=torch.max(output,1)[1].item()

            if (Final_Output>=151 and Final_Output<=268):
                Final_Outputs=True


            else:
                Final_Outputs=False

            return Final_Outputs
```

```
In [27]: ### TODO: Test the performance of the dog_detector function
         ### on the images in human_files_short and dog_files_short
         #Human_Files_Short
         for humans in human_files_short:
             dog_detector(humans)
```

```
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 456 and is it Dog?: False
The Predicted Class is 678 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 768 and is it Dog?: False
The Predicted Class is 683 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 432 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 819 and is it Dog?: False
The Predicted Class is 457 and is it Dog?: False
The Predicted Class is 0 and is it Dog?: False
The Predicted Class is 424 and is it Dog?: False
The Predicted Class is 678 and is it Dog?: False
The Predicted Class is 617 and is it Dog?: False
The Predicted Class is 903 and is it Dog?: False
The Predicted Class is 903 and is it Dog?: False
The Predicted Class is 722 and is it Dog?: False
The Predicted Class is 722 and is it Dog?: False
The Predicted Class is 617 and is it Dog?: False
The Predicted Class is 903 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 917 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 862 and is it Dog?: False
The Predicted Class is 400 and is it Dog?: False
The Predicted Class is 678 and is it Dog?: False
The Predicted Class is 902 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 610 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 585 and is it Dog?: False
The Predicted Class is 617 and is it Dog?: False
The Predicted Class is 457 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 610 and is it Dog?: False
The Predicted Class is 432 and is it Dog?: False
The Predicted Class is 903 and is it Dog?: False
The Predicted Class is 617 and is it Dog?: False
The Predicted Class is 903 and is it Dog?: False
The Predicted Class is 459 and is it Dog?: False
The Predicted Class is 400 and is it Dog?: False
The Predicted Class is 585 and is it Dog?: False
The Predicted Class is 917 and is it Dog?: False
The Predicted Class is 400 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 678 and is it Dog?: False
The Predicted Class is 400 and is it Dog?: False
The Predicted Class is 585 and is it Dog?: False
The Predicted Class is 678 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 678 and is it Dog?: False
The Predicted Class is 683 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 457 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 465 and is it Dog?: False
The Predicted Class is 465 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 617 and is it Dog?: False
The Predicted Class is 739 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 683 and is it Dog?: False
The Predicted Class is 620 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 0 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 875 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 610 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 903 and is it Dog?: False
The Predicted Class is 834 and is it Dog?: False
The Predicted Class is 0 and is it Dog?: False
The Predicted Class is 906 and is it Dog?: False
The Predicted Class is 585 and is it Dog?: False
The Predicted Class is 400 and is it Dog?: False
The Predicted Class is 903 and is it Dog?: False
The Predicted Class is 465 and is it Dog?: False
The Predicted Class is 683 and is it Dog?: False
The Predicted Class is 652 and is it Dog?: False
The Predicted Class is 743 and is it Dog?: False
The Predicted Class is 683 and is it Dog?: False
```

```
In [28]: #Dog_short_Files

         for dogs in dog_files_short:
             dog_detector(dogs)
```

```
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 210 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 246 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 163 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 209 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 210 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 243 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 234 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 165 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 227 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
The Predicted Class is 236 and is it Dog?: True
```

1.1.7 (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: The percentage of Images in the human_files_short which are detected as dogs is 1% and The Percentage of images in dog_files_short whcih are detected as dogs is 100%

1.2 PREDICTIONS

```
In [30]: from PIL import Image
         #Human_files_short
         IMAGE=Image.open(human_files_short[0])
         dog_detector(human_files_short[0])
         IMAGE
```

```
The Predicted Class is 906 and is it Dog?: False
```

Out[30]:

```
In [31]: #Dog_Files_short
         IMAGE=Image.open(dog_files_short[1])
         dog_detector(dog_files_short[1])
         IMAGE
```

```
The Predicted Class is 243 and is it Dog?: True
```

Out[31]:

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, ResNet-50, 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 [32]: ### (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

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!

1.2.1 (IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate data loaders 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 to be a useful resource. If you are interested in augmenting your training and/or validation data, check out the wide variety of transforms!

```
In [62]: ### TODO: Write data loaders for training, validation, and test sets
         ## Specify appropriate transforms, and batch_sizes
         import os
         import torch
         from torchvision import datasets, transforms

         data_dir='/data/dog_images/'
         train_dir=os.path.join(data_dir,'train/')
         test_dir=os.path.join(data_dir,'test/')
         valid_dir=os.path.join(data_dir,'valid/')

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


         train_transforms = transforms.Compose([transforms.Resize(size=258),
                                              transforms.RandomHorizontalFlip(),
                                              transforms.RandomRotation(10),
                                              transforms.CenterCrop(224),
                                              transforms.ToTensor(),
                                              transforms.Normalize([0.5, 0.5, 0.5],
                                                                   [0.5, 0.5, 0.5])])

        validTest_transforms = transforms.Compose([transforms.Resize(size=258),
                                                  transforms.CenterCrop(224),
                                                  transforms.ToTensor(),
                                                  transforms.Normalize([0.5, 0.5, 0.5],
                                                                       [0.5, 0.5, 0.5])])

        train_dataset = datasets.ImageFolder(train_dir, transform=train_transforms)
        valid_dataset = datasets.ImageFolder(valid_dir, transform=validTest_transforms)
        test_dataset = datasets.ImageFolder(test_dir, transform=validTest_transforms)

        trainLoader = torch.utils.data.DataLoader(train_dataset,
                                                 batch_size=128,
                                                 shuffle=True,
                                                 num_workers=0)

        validLoader = torch.utils.data.DataLoader(valid_dataset,
                                                 batch_size=128,
                                                 shuffle=True,
                                                 num_workers=0)

        testLoader = torch.utils.data.DataLoader(test_dataset,
                                                batch_size=64,
                                                shuffle=False,
                                                num_workers=0)
```

```
In [63]: print('Total Number of training images:',len(train_dataset))
         print('Total Number of validation images:',len(valid_dataset))
         print('Total Number of test images:',len(test_dataset))
```

```
Total Number of training images: 6680
Total Number of validation images: 835
Total Number of test images: 836
```

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 have Applied the transforms Module Provided by Pytorch Framework and Resized the Image to be 256*256 px and also Centercropped the Image with the size 224*224. The size of the Input Tensor is 2242243 The Images are Transformed into the tensors and also Normalized based on the

Provided Mean and Std. Data Augmentation: I used Randomhorizontal flip with probabililty of default value and Randomrotation of degree 10 .

1.2.2 (IMPLEMENTATION) Model Architecture

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

```
In [64]: import torch.nn as nn
         import torch.nn.functional as F
```

```
In [65]: # 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(3,16,3,padding=1)
                 self.Conv2=nn.Conv2d(16,32,3,padding=1)
                 self.Conv3=nn.Conv2d(32,64,3,padding=1)
                 self.Conv4=nn.Conv2d(64,128,3,padding=1)
                 self.Conv5=nn.Conv2d(128,256,3,padding=1)
                 self.pool=nn.MaxPool2d(2,2)
                 self.fc1=nn.Linear(12544,512)
                 self.fc2=nn.Linear(512,512)
                 self.fc3=nn.Linear(512,133)
                 self.dropout = nn.Dropout(0.5)

             def forward(self, x):
                 ## Define forward behavior
                 x=self.pool(F.relu(self.Conv1(x)))
                 x=self.pool(F.relu(self.Conv2(x)))
                 x=self.pool(F.relu(self.Conv3(x)))
                 x=self.pool(F.relu(self.Conv4(x)))
                 x=self.pool(F.relu(self.Conv5(x)))
                 x = x.view(-1, 12544)
                 x = F.relu(self.fc1(x))
                 x = self.dropout(x)
                 x = F.relu(self.fc2(x))
                 x = self.dropout(x)
                 x = 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()
```

```
In [66]: model_scratch
```

```
Out[66]: Net(
           (Conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (Conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (Conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (Conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (Conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
           (fc1): Linear(in_features=12544, out_features=512, bias=True)
           (fc2): Linear(in_features=512, out_features=512, bias=True)
           (fc3): Linear(in_features=512, out_features=133, bias=True)
           (dropout): Dropout(p=0.5)
         )
```

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

Answer: As with the data loaders, I decided to follow the original VGG16 paper (Simonyan K, Zisserman A 2015) for the neural network architecture. As the project requirements are to only achieve a 10% test accuracy I took the simplest model from the VGG16 paper (Table 1, Column A) and further simplified it.

I chose to have 5 convolutional layers with a kernel size of 3x3 and a padding of 1, which gradually increases the number of feature maps, but keeps the size. In between every convolutional layer, there is a maxpool layer with a 2x2 kernel and a stride of 2, that halfs the size of all featuremaps. After 5 convolutions and maxpool layers we end up with 256 7x7 feature maps.

Then the feature maps are flattened to a vector of length 12544 and fed into the fully connected (FC) layers for classification. Following the VGG16 paper I took 3 FC layers, but reduced the number of nodes per layer as we have only 133 classes, not a 1000.

A further deviation from the paper, is that the last layer in my network is a FC layer, not a softmax layer. This is because for training, I used PyTorche's CrossEntropyLoss() class, that combines a log-softmax output-layer activation and a negative log-likelihood loss-function.

When testing the neural net, the output of the network is fed into a softmax function to obtain class probabilities.

1.2.3 (IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_scratch`, and the optimizer as `optimizer_scratch` below.

```
In [67]: import torch.optim as optim

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

         ### TODO: select optimizer
         optimizer_scratch = optim.Adam(model_scratch.parameters(), lr=0.001)
```

1.2.4 (IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. Save the final model parameters at filepath `'model_scratch.pt'`.

```
In [70]: from PIL import ImageFile
         ImageFile.LOAD_TRUNCATED_IMAGES = True

         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 - train_lo

                     # clear gradients of all optimized variables
                     optimizer.zero_grad()

                     # forwarward pass
                     output = model(data)

                     # calculate batch loss
                     loss = criterion(output, target)

                     # backward pass
                     loss.backward()

                     # perform optimization step
                     optimizer.step()

                     # update training 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
                with torch.no_grad():
                    output = model(data)
                loss = criterion(output, target)
                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 model...'.form
                torch.save(model.state_dict(), save_path)
                valid_loss_min = valid_loss

        # return trained model
        return model


    # define loaders_scratch
    loaders_scratch = {'train': trainLoader,
                       'valid': validLoader,
                       'test': testLoader}

    # train the model
    model_scratch = train(25, 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: 4.884323         Validation Loss: 4.863628
Validation loss decreased (inf --> 4.863628). Saving model...
Epoch: 2         Training Loss: 4.857897         Validation Loss: 4.835469
Validation loss decreased (4.863628 --> 4.835469). Saving model...
Epoch: 3         Training Loss: 4.719491         Validation Loss: 4.619555
```

```
Validation loss decreased (4.835469 --> 4.619555). Saving model...
Epoch: 4        Training Loss: 4.570438        Validation Loss: 4.499638
Validation loss decreased (4.619555 --> 4.499638). Saving model...
Epoch: 5        Training Loss: 4.494217        Validation Loss: 4.428037
Validation loss decreased (4.499638 --> 4.428037). Saving model...
Epoch: 6        Training Loss: 4.386159        Validation Loss: 4.296769
Validation loss decreased (4.428037 --> 4.296769). Saving model...
Epoch: 7        Training Loss: 4.264459        Validation Loss: 4.192825
Validation loss decreased (4.296769 --> 4.192825). Saving model...
Epoch: 8        Training Loss: 4.170181        Validation Loss: 4.080296
Validation loss decreased (4.192825 --> 4.080296). Saving model...
Epoch: 9        Training Loss: 4.090755        Validation Loss: 4.043947
Validation loss decreased (4.080296 --> 4.043947). Saving model...
Epoch: 10        Training Loss: 3.999058        Validation Loss: 3.951761
Validation loss decreased (4.043947 --> 3.951761). Saving model...
Epoch: 11        Training Loss: 3.933572        Validation Loss: 3.940479
Validation loss decreased (3.951761 --> 3.940479). Saving model...
Epoch: 12        Training Loss: 3.886173        Validation Loss: 3.935418
Validation loss decreased (3.940479 --> 3.935418). Saving model...
Epoch: 13        Training Loss: 3.808164        Validation Loss: 3.827018
Validation loss decreased (3.935418 --> 3.827018). Saving model...
Epoch: 14        Training Loss: 3.757411        Validation Loss: 3.794480
Validation loss decreased (3.827018 --> 3.794480). Saving model...
Epoch: 15        Training Loss: 3.705557        Validation Loss: 3.758253
Validation loss decreased (3.794480 --> 3.758253). Saving model...
Epoch: 16        Training Loss: 3.614367        Validation Loss: 3.764199
Epoch: 17        Training Loss: 3.581193        Validation Loss: 3.732887
Validation loss decreased (3.758253 --> 3.732887). Saving model...
Epoch: 18        Training Loss: 3.531867        Validation Loss: 3.767703
Epoch: 19        Training Loss: 3.447338        Validation Loss: 3.615593
Validation loss decreased (3.732887 --> 3.615593). Saving model...
Epoch: 20        Training Loss: 3.427010        Validation Loss: 3.554329
Validation loss decreased (3.615593 --> 3.554329). Saving model...
Epoch: 21        Training Loss: 3.376791        Validation Loss: 3.606595
Epoch: 22        Training Loss: 3.277090        Validation Loss: 3.535795
Validation loss decreased (3.554329 --> 3.535795). Saving model...
Epoch: 23        Training Loss: 3.226920        Validation Loss: 3.512028
Validation loss decreased (3.535795 --> 3.512028). Saving model...
Epoch: 24        Training Loss: 3.201728        Validation Loss: 3.485039
Validation loss decreased (3.512028 --> 3.485039). Saving model...
Epoch: 25        Training Loss: 3.135055        Validation Loss: 3.498969
```

1.2.5 (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 [82]: 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().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: 3.540429


Test Accuracy: 16% (138/836)
```

## 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.

1.2.6 (IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate data loaders 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 [90]: ## TODO: Specify data loaders
         data_dir='/data/dog_images/'
         train_dir=os.path.join(data_dir,'train/')
         test_dir=os.path.join(data_dir,'test/')
         valid_dir=os.path.join(data_dir,'valid/')

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


         train_transforms = transforms.Compose([transforms.Resize(size=258),
                                               transforms.RandomHorizontalFlip(),
                                               transforms.RandomRotation(10),
                                               transforms.CenterCrop(224),
                                               transforms.ToTensor(),
                                               transforms.Normalize([0.485, 0.456, 0.406],
                                                                    [0.229, 0.224, 0.225])])

         validTest_transforms = transforms.Compose([transforms.Resize(size=258),
                                                   transforms.CenterCrop(224),
                                                   transforms.ToTensor(),
                                                   transforms.Normalize([0.485, 0.456, 0.406],
                                                                    [0.229, 0.224, 0.225])])


         train_dataset = datasets.ImageFolder(train_dir, transform=train_transforms)
         valid_dataset = datasets.ImageFolder(valid_dir, transform=validTest_transforms)
         test_dataset = datasets.ImageFolder(test_dir, transform=validTest_transforms)

         trainLoader = torch.utils.data.DataLoader(train_dataset,
                                                  batch_size=128,
                                                  shuffle=True,
                                                  num_workers=0)

         validLoader = torch.utils.data.DataLoader(valid_dataset,
                                                  batch_size=128,
                                                  shuffle=True,
                                                  num_workers=0)

         testLoader = torch.utils.data.DataLoader(test_dataset,
                                                  batch_size=64,
                                                  shuffle=False,
                                                  num_workers=0)
```

1.2.7 (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 [91]: import torchvision.models as models
         import torch.nn as nn

         ## TODO: Specify model architecture
         # check if CUDA is available
         use_cuda = torch.cuda.is_available()

         # download VGG16 pretrained model
         model_transfer = models.vgg16(pretrained=True)

         # Freeze parameters of the model to avoid brackpropagation
         for param in model_transfer.parameters():
             param.requires_grad = False

         # get the number of dog classes from the train_dataset
         number_of_dog_classes = len(train_dataset.classes)

         # Define dog breed classifier part of model_transfer
         classifier = nn.Sequential(nn.Linear(25088, 4096),
                                    nn.ReLU(),
                                    nn.Dropout(0.5),
                                    nn.Linear(4096, 512),
                                    nn.ReLU(),
                                    nn.Dropout(0.5),
                                    nn.Linear(512, number_of_dog_classes))

         # Rplace the original classifier with the dog breed classifier from above
         model_transfer.classifier = classifier


         if use_cuda:
             model_transfer = model_transfer.cuda()

In [92]: model_transfer

Out[92]: 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_mode=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_mode=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_mode=False)
      )
      (classifier): Sequential(
        (0): Linear(in_features=25088, out_features=4096, bias=True)
        (1): ReLU()
        (2): Dropout(p=0.5)
        (3): Linear(in_features=4096, out_features=512, bias=True)
        (4): ReLU()
        (5): Dropout(p=0.5)
        (6): Linear(in_features=512, out_features=133, bias=True)
      )
    )
```

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: As the VGG16 network was also trained on dogs, among other things, it contains useful high level feature information in the later convolutional layers.

This is why I used the entire feature extractor part from the VGG16 pretrained model keeping the weights constant, to avoid overfitting when training on a small new dataset (~7000 dog images). Then I replaced the classifier part with my own dog breed classifier.

The drog breed classifier is modeled along the original VGG16 calssifier with 3 fully connected layers 2 dropout layers to reduce the number of parameters and then 2 ReLU activations.

I changed the number of nodes per fully connected layer to match the number of dog classes we have, i.e. 133.

1.2.8 (IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_transfer`, and the optimizer as `optimizer_transfer` below.

```
In [93]: criterion_transfer =nn.CrossEntropyLoss()
         optimizer_transfer = optim.Adam(model_transfer.classifier.parameters(), lr=0.001)
```

1.2.9 (IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. Save the final model parameters at filepath `'model_transfer.pt'`.

```
In [94]: from PIL import ImageFile
         ImageFile.LOAD_TRUNCATED_IMAGES = True
```

```
In [95]: def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path):

             valid_loss_min = np.Inf

             print(f"Batch Size: {loaders['train'].batch_size}\n")

             for epoch in range(1, n_epochs+1):
                 train_loss = 0.0
                 valid_loss = 0.0

                 # train the model
                 model.train()
                 for batch_idx, (data, target) in enumerate(loaders['train']):
                     if use_cuda:
                         data, target = data.cuda(), target.cuda()

                     optimizer.zero_grad()
                     output = model(data)
                     loss = criterion(output, target)
                     loss.backward()
                     optimizer.step()
                     train_loss += ((1 / (batch_idx + 1)) * (loss.data - train_loss))

                     if (batch_idx + 1) % 5 == 0:
                         print(f'Epoch:{epoch}/{n_epochs} \tBatch:{batch_idx + 1}')
                         print(f'Train Loss: {train_loss}\n')

                 # validate the model
                 model.eval()
                 for batch_idx, (data, target) in enumerate(loaders['valid']):
                     if use_cuda:
                         data, target = data.cuda(), target.cuda()
                     with torch.no_grad():
```

```
                output = model(data)
            loss = criterion(output, target)
            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
            ))

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

    # return trained model
    return model


# define loaders_transfer
loaders_transfer = {'train': trainLoader,
                    'valid': validLoader,
                    'test': testLoader}

model_transfer = train(7, loaders_transfer, model_transfer, optimizer_transfer,
                       criterion_transfer, use_cuda, 'model_transfer.pt')

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

```
Batch Size: 128

Epoch:1/7         Batch:5
Train Loss: 6.744067192077637

Epoch:1/7         Batch:10
Train Loss: 5.830857276916504

Epoch:1/7         Batch:15
Train Loss: 5.488731861114502

Epoch:1/7         Batch:20
Train Loss: 5.310342311859131

Epoch:1/7         Batch:25
Train Loss: 5.159582138061523
```

```
Epoch:1/7         Batch:30
Train Loss: 5.043935775756836

Epoch:1/7         Batch:35
Train Loss: 4.929346561431885

Epoch:1/7         Batch:40
Train Loss: 4.809517860412598

Epoch:1/7         Batch:45
Train Loss: 4.698791027069092

Epoch:1/7         Batch:50
Train Loss: 4.595192909240723

Epoch: 1         Training Loss: 4.527544         Validation Loss: 2.790016
Validation loss decreased (inf --> 2.790016). Saving model...
Epoch:2/7         Batch:5
Train Loss: 3.3974080085754395

Epoch:2/7         Batch:10
Train Loss: 3.2088894844055176

Epoch:2/7         Batch:15
Train Loss: 3.1254780292510986

Epoch:2/7         Batch:20
Train Loss: 3.0540482997894287

Epoch:2/7         Batch:25
Train Loss: 2.980360984802246

Epoch:2/7         Batch:30
Train Loss: 2.9267208576202393

Epoch:2/7         Batch:35
Train Loss: 2.8529624938964844

Epoch:2/7         Batch:40
Train Loss: 2.799086809158325

Epoch:2/7         Batch:45
Train Loss: 2.7415084838867188

Epoch:2/7         Batch:50
Train Loss: 2.694981336593628
```

```
Epoch: 2        Training Loss: 2.661341         Validation Loss: 1.322083
Validation loss decreased (2.790016 --> 1.322083). Saving model...
Epoch:3/7         Batch:5
Train Loss: 2.0948681831359863

Epoch:3/7         Batch:10
Train Loss: 2.0389177799224854

Epoch:3/7         Batch:15
Train Loss: 2.0134973526000977

Epoch:3/7         Batch:20
Train Loss: 2.0263025760650635

Epoch:3/7         Batch:25
Train Loss: 1.9965153932571411

Epoch:3/7         Batch:30
Train Loss: 1.963607907295227

Epoch:3/7         Batch:35
Train Loss: 1.9280827045440674

Epoch:3/7         Batch:40
Train Loss: 1.9187381267547607

Epoch:3/7         Batch:45
Train Loss: 1.8962645530700684

Epoch:3/7         Batch:50
Train Loss: 1.892004132270813

Epoch: 3        Training Loss: 1.885331         Validation Loss: 1.061932
Validation loss decreased (1.322083 --> 1.061932). Saving model...
Epoch:4/7         Batch:5
Train Loss: 1.6263694763183594

Epoch:4/7         Batch:10
Train Loss: 1.5980726480484009

Epoch:4/7         Batch:15
Train Loss: 1.6040397882461548

Epoch:4/7         Batch:20
Train Loss: 1.589615821838379

Epoch:4/7         Batch:25
Train Loss: 1.5828917026519775
```

```
Epoch:4/7         Batch:30
Train Loss: 1.5739378929138184

Epoch:4/7         Batch:35
Train Loss: 1.550330638885498

Epoch:4/7         Batch:40
Train Loss: 1.5558743476867676

Epoch:4/7         Batch:45
Train Loss: 1.5535045862197876

Epoch:4/7         Batch:50
Train Loss: 1.5445736646652222

Epoch: 4         Training Loss: 1.539927         Validation Loss: 0.902427
Validation loss decreased (1.061932 --> 0.902427). Saving model...
Epoch:5/7         Batch:5
Train Loss: 1.3149114847183228

Epoch:5/7         Batch:10
Train Loss: 1.303647756576538

Epoch:5/7         Batch:15
Train Loss: 1.333670735359192

Epoch:5/7         Batch:20
Train Loss: 1.3507320880889893

Epoch:5/7         Batch:25
Train Loss: 1.3458675146102905

Epoch:5/7         Batch:30
Train Loss: 1.3403440713882446

Epoch:5/7         Batch:35
Train Loss: 1.3413878679275513

Epoch:5/7         Batch:40
Train Loss: 1.3476673364639282

Epoch:5/7         Batch:45
Train Loss: 1.3388484716415405

Epoch:5/7         Batch:50
Train Loss: 1.3455009460449219
```

```
Epoch: 5         Training Loss: 1.349855         Validation Loss: 0.859427
Validation loss decreased (0.902427 --> 0.859427). Saving model...
Epoch:6/7         Batch:5
Train Loss: 1.234503984451294

Epoch:6/7         Batch:10
Train Loss: 1.2394636869430542

Epoch:6/7         Batch:15
Train Loss: 1.2402468919754028

Epoch:6/7         Batch:20
Train Loss: 1.2269606590270996

Epoch:6/7         Batch:25
Train Loss: 1.2418113946914673

Epoch:6/7         Batch:30
Train Loss: 1.2292767763137817

Epoch:6/7         Batch:35
Train Loss: 1.2273476123809814

Epoch:6/7         Batch:40
Train Loss: 1.2270817756652832

Epoch:6/7         Batch:45
Train Loss: 1.2176815271377563

Epoch:6/7         Batch:50
Train Loss: 1.2109863758087158

Epoch: 6         Training Loss: 1.210793         Validation Loss: 0.806719
Validation loss decreased (0.859427 --> 0.806719). Saving model...
Epoch:7/7         Batch:5
Train Loss: 1.098770260810852

Epoch:7/7         Batch:10
Train Loss: 1.0816725492477417

Epoch:7/7         Batch:15
Train Loss: 1.1219053268432617

Epoch:7/7         Batch:20
Train Loss: 1.1193809509277344

Epoch:7/7         Batch:25
Train Loss: 1.1127084493637085
```

```
Epoch:7/7          Batch:30
Train Loss: 1.1053835153579712

Epoch:7/7          Batch:35
Train Loss: 1.1191606521606445

Epoch:7/7          Batch:40
Train Loss: 1.1257866621017456

Epoch:7/7          Batch:45
Train Loss: 1.1387702226638794

Epoch:7/7          Batch:50
Train Loss: 1.134273648262024

Epoch: 7         Training Loss: 1.144647         Validation Loss: 0.788688
Validation loss decreased (0.806719 --> 0.788688). Saving model...
```

1.2.10 (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 [96]: 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
                 output = F.softmax(output, dim=1)
                 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)

    test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

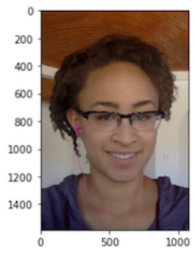
```
Test Loss: 0.771300


Test Accuracy: 75% (627/836)
```

1.2.11 (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 [97]: ### 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]
         #class_names = [item[4:].replace("_", " ") for item in data_transfer['train'].classes]
         class_names = [item[4:].replace("_", " ") for item in train_dataset.classes]

         model_transfer.load_state_dict(torch.load('model_transfer.pt'))

         def predict_breed_transfer(img_path):
             # load the image and return the predicted breed
             img = Image.open(img_path)

             # Define normalization step for image
             normalize = transforms.Normalize(mean=(0.485, 0.456, 0.406),
                                              std=(0.229, 0.224, 0.225))

             # Define transformations of image
             preprocess = transforms.Compose([transforms.Resize(258),
                                              transforms.CenterCrop(224),
                                              transforms.ToTensor(),
                                              normalize])

             # Preprocess image to 4D Tensor (.unsqueeze(0) adds a dimension)
             img_tensor = preprocess(img).unsqueeze_(0)

             # Move tensor to GPU if available
             if use_cuda:
```

hello, human!



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

Sample Human Output

```
        img_tensor = img_tensor.cuda()

    ## Inference
    # Turn on evaluation mode
    model_transfer.eval()

    # Get predicted category for image
    with torch.no_grad():
        output = model_transfer(img_tensor)
        prediction = torch.argmax(output).item()

    # Turn off evaluation mode
    model_transfer.train()

    # Use prediction to get dog breed
    breed = class_names[prediction]

    return breed
```

## 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!

1.2.12 (IMPLEMENTATION) Write your Algorithm

```
In [103]: ### 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  face_detector(img_path):
        print('Hello Human!')
        plt.imshow(Image.open(img_path))
        plt.show()
        print(f'You look like a ... {predict_breed_transfer(img_path)}')
        print('\n-----------------------------------\n')
    elif dog_detector(img_path):
        plt.imshow(Image.open(img_path))
        plt.show()
        print(f'This is a picture of a ... {predict_breed_transfer(img_path)}')
        print('\n-----------------------------------\n')
    else:
        plt.imshow(Image.open(img_path))
        plt.show()
        print('Sorry, I did not detect a human or a dog in this image.')
        print('\n-----------------------------------\n')
```

## 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?

1.2.13 (IMPLEMENTATION) Test Your Algorithm on Sample Images!

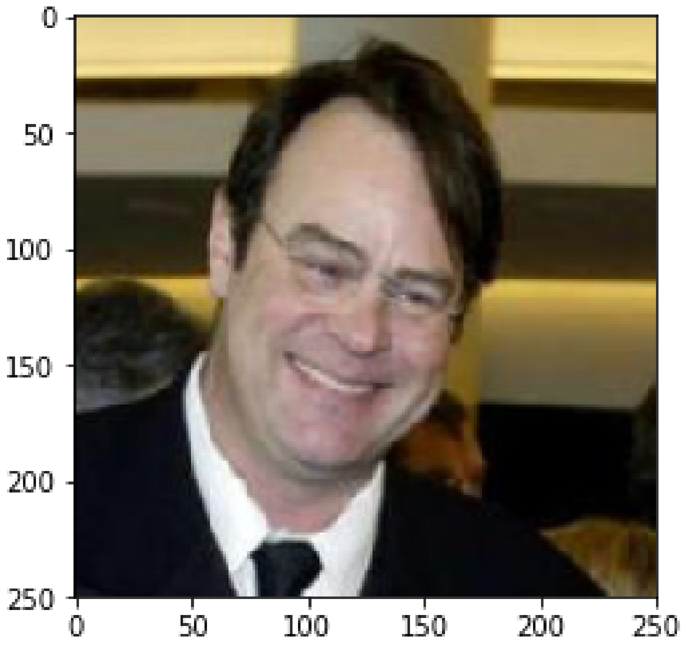
Test your algorithm at least six images on your computer. Feel free to use any images you like. Use at least two human and two dog images.

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

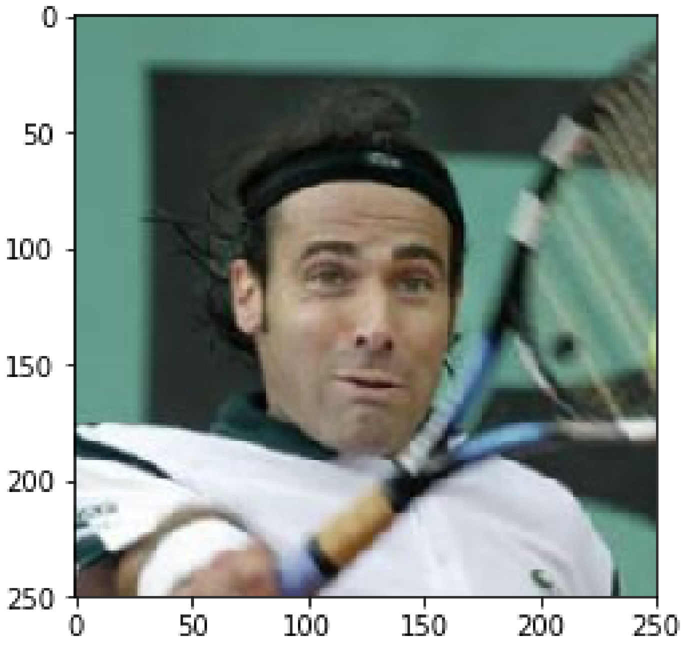
Answer: (Three possible points for improvement)

```
In [104]: ## 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[:10], dog_files[:10])):
              run_app(file)
```

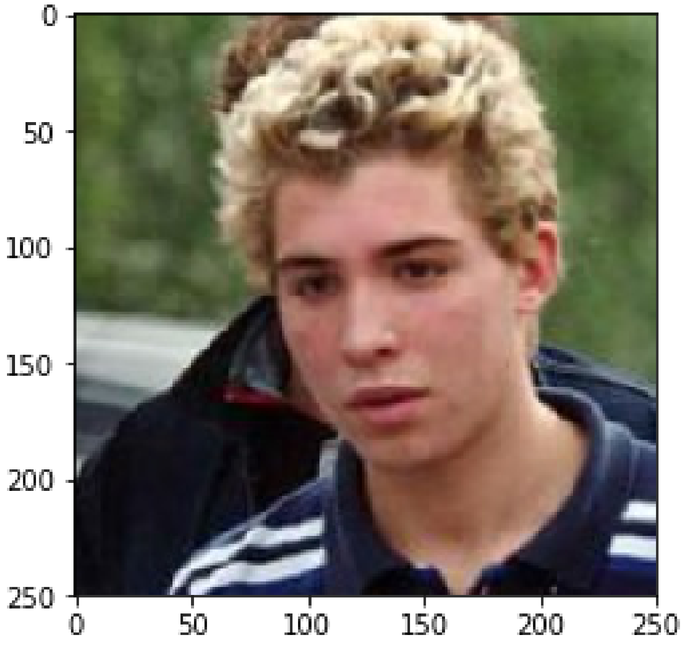
```
Hello Human!
```



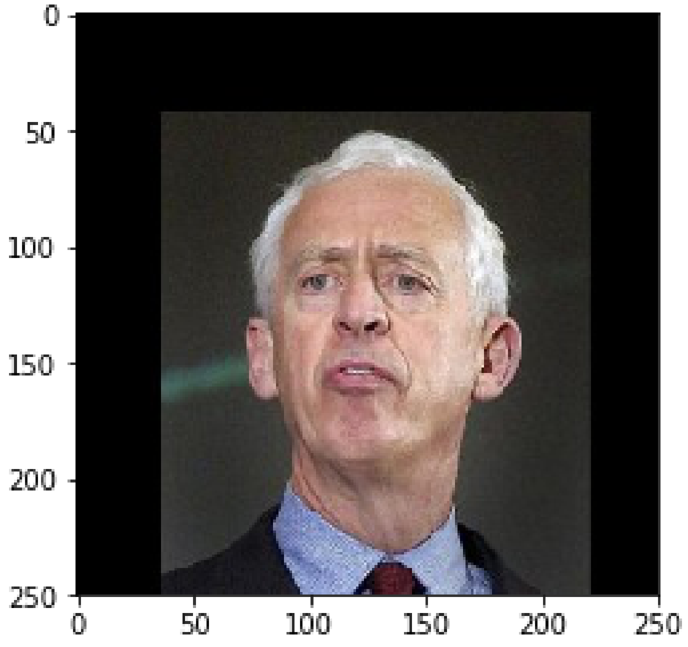
```
You look like a ... Dachshund

-----------------------------------

Hello Human!
```



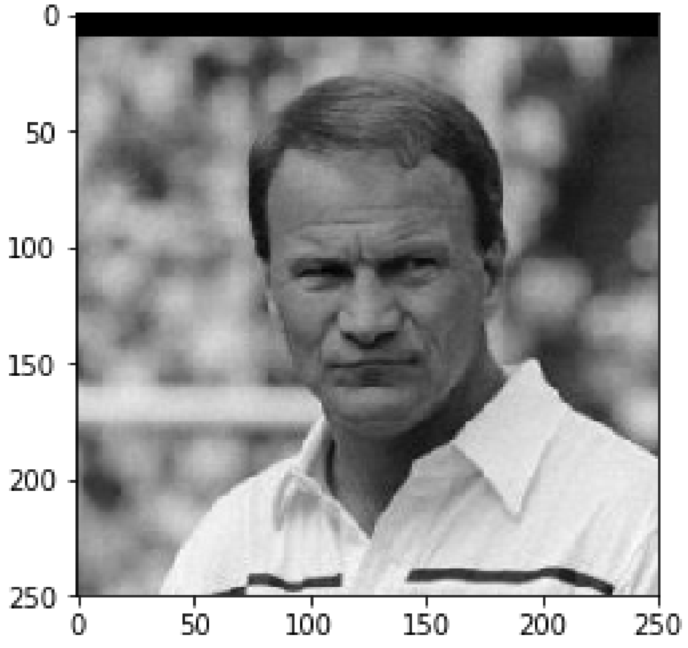
```
You look like a ... Chinese crested

-----------------------------------

Hello Human!
```



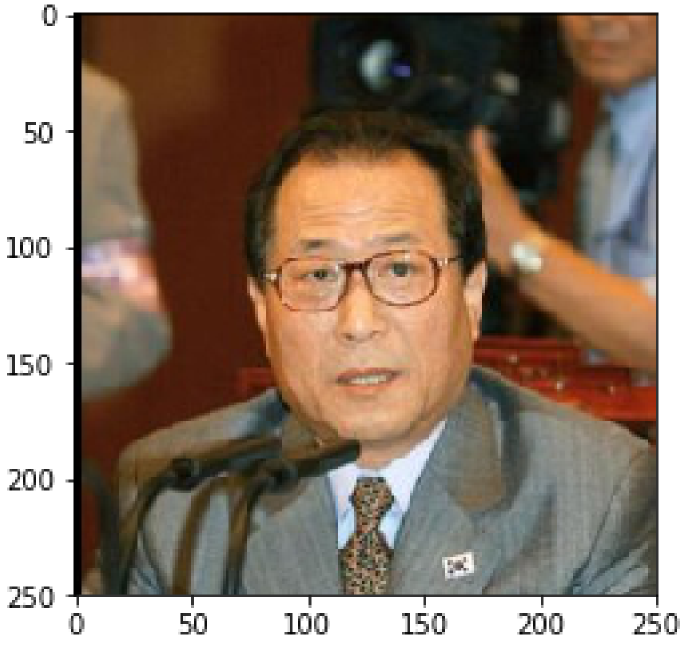
```
You look like a ... Black russian terrier

-----------------------------------

Hello Human!
```



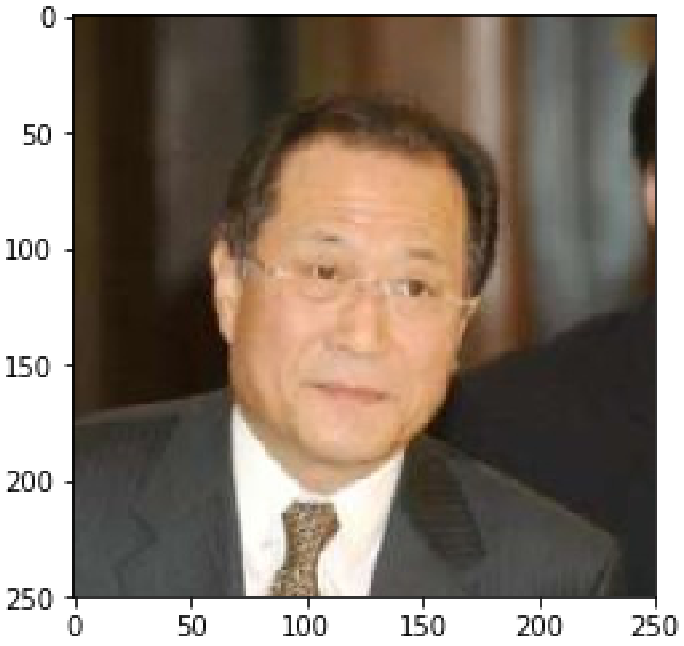
```
You look like a ... Chesapeake bay retriever

-----------------------------------

Hello Human!
```



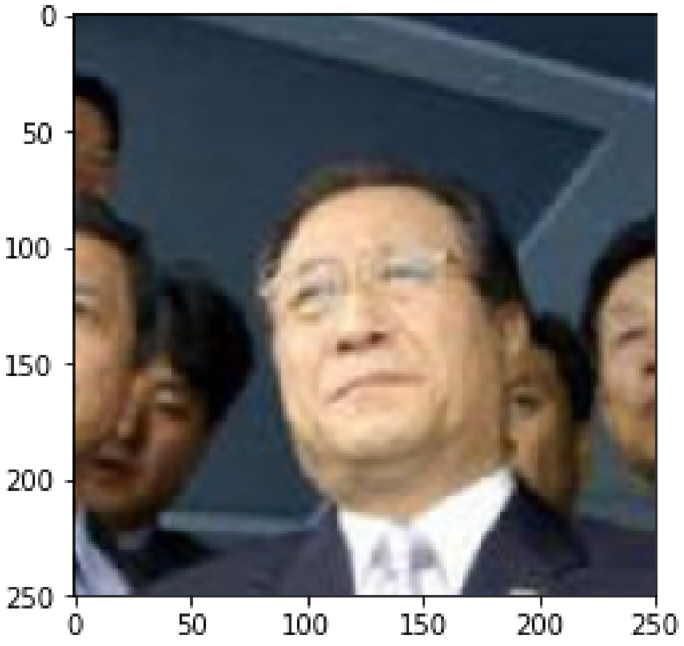
```
You look like a ... Norwegian buhund

-----------------------------------

Hello Human!
```



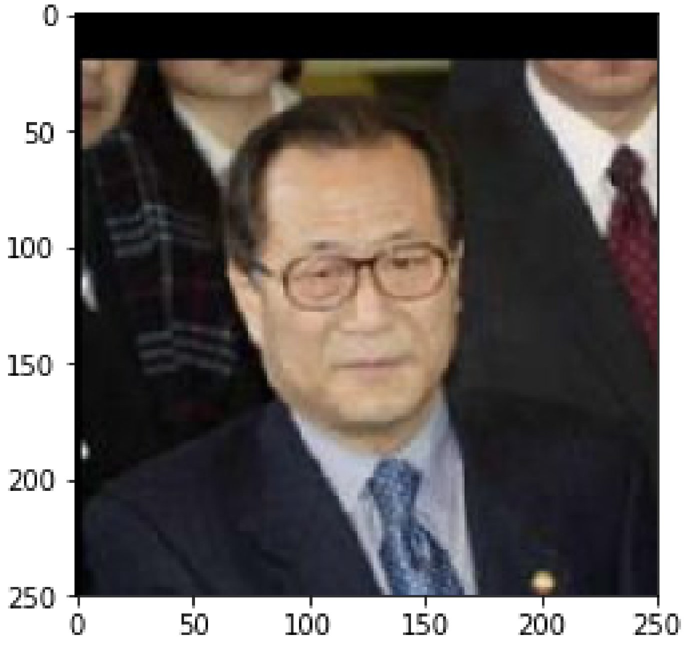
```
You look like a ... Xoloitzcuintli

-----------------------------------

Hello Human!
```



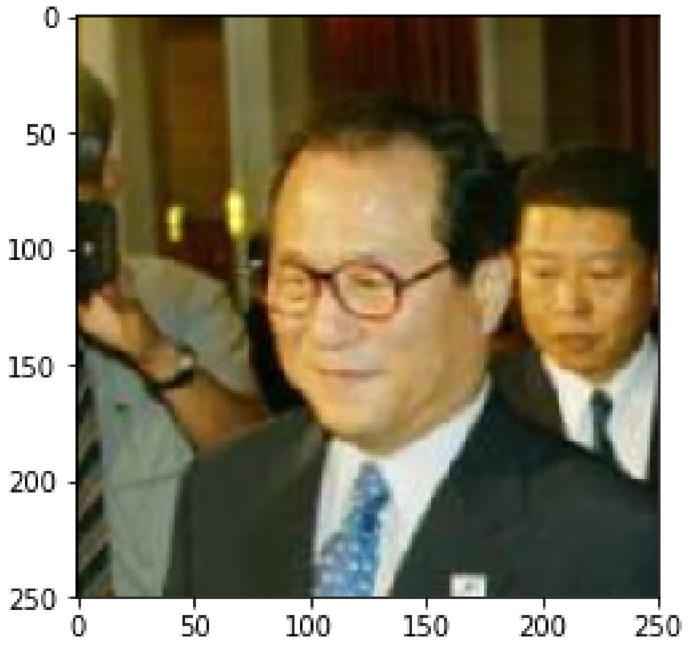
```
You look like a ... Italian greyhound

-----------------------------------

Hello Human!
```



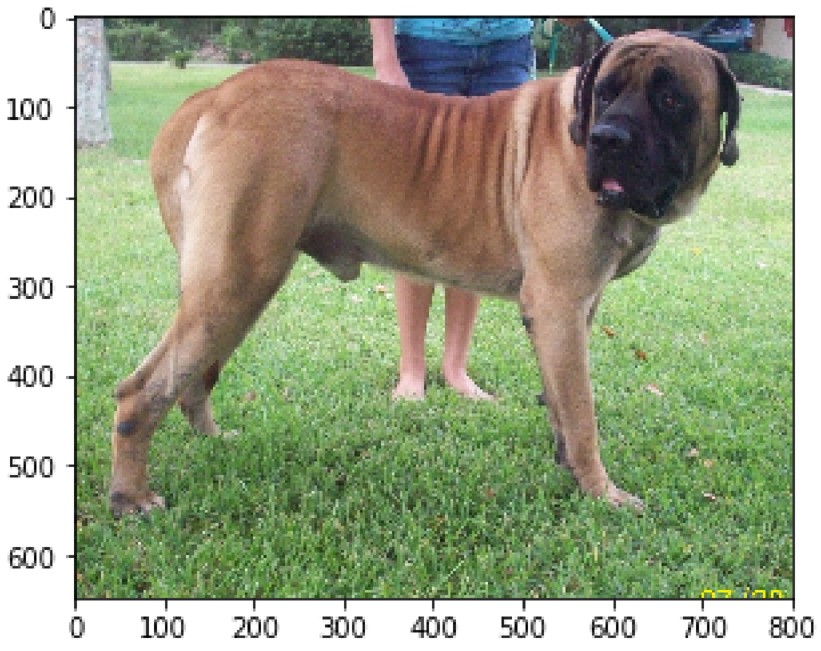
```
You look like a ... Doberman pinscher

-----------------------------------

Hello Human!
```



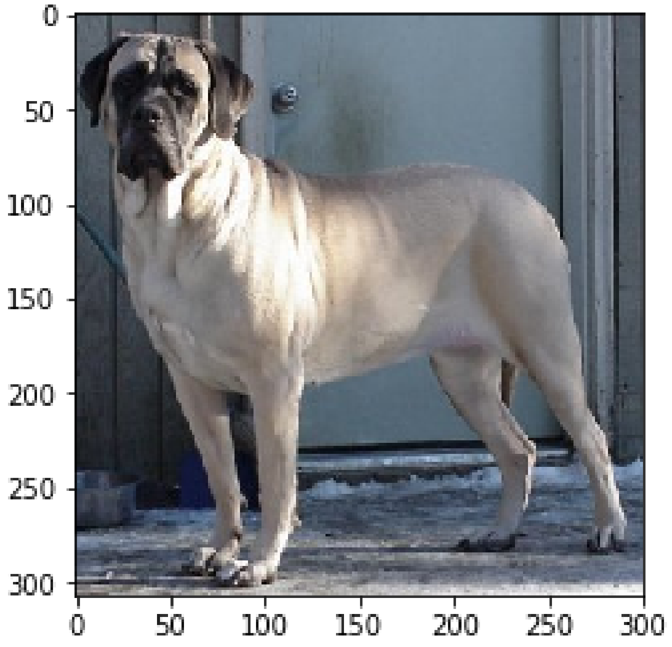
```
You look like a ... Xoloitzcuintli

-----------------------------------

Hello Human!
```



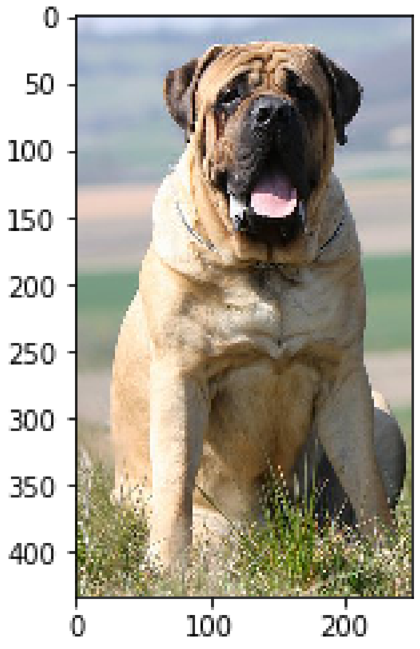
```
You look like a ... Xoloitzcuintli

-----------------------------------
```



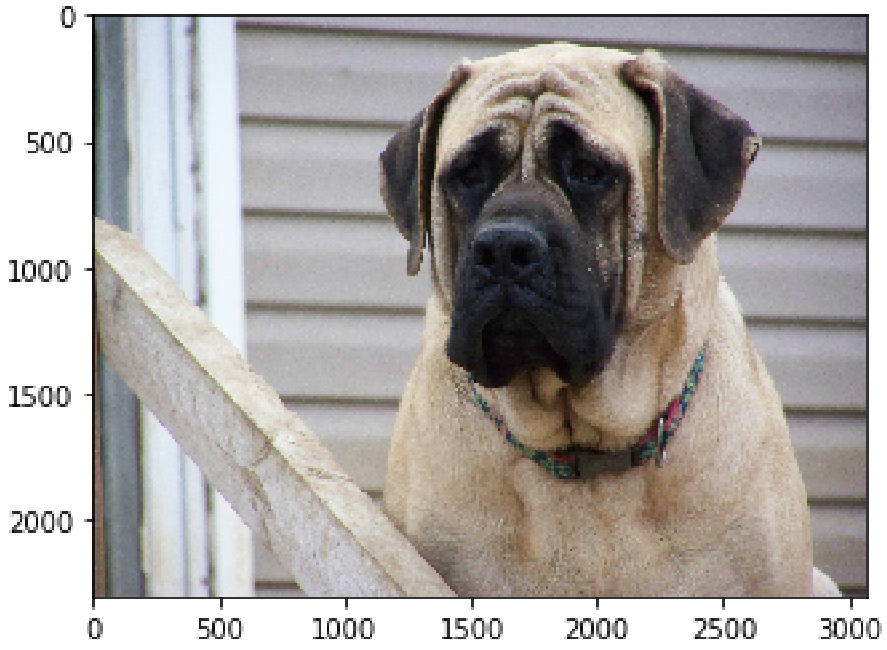
```
This is a picture of a ... Bullmastiff

-----------------------------------
```



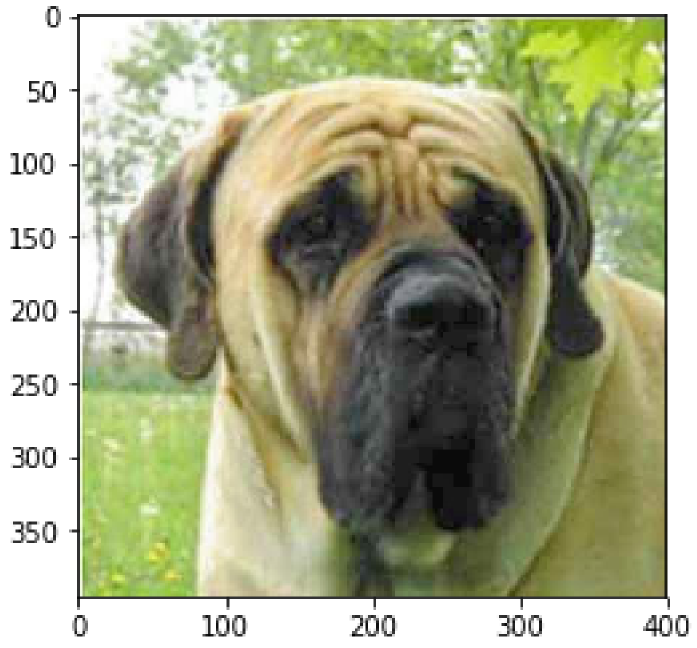
```
This is a picture of a ... Mastiff

-----------------------------------
```



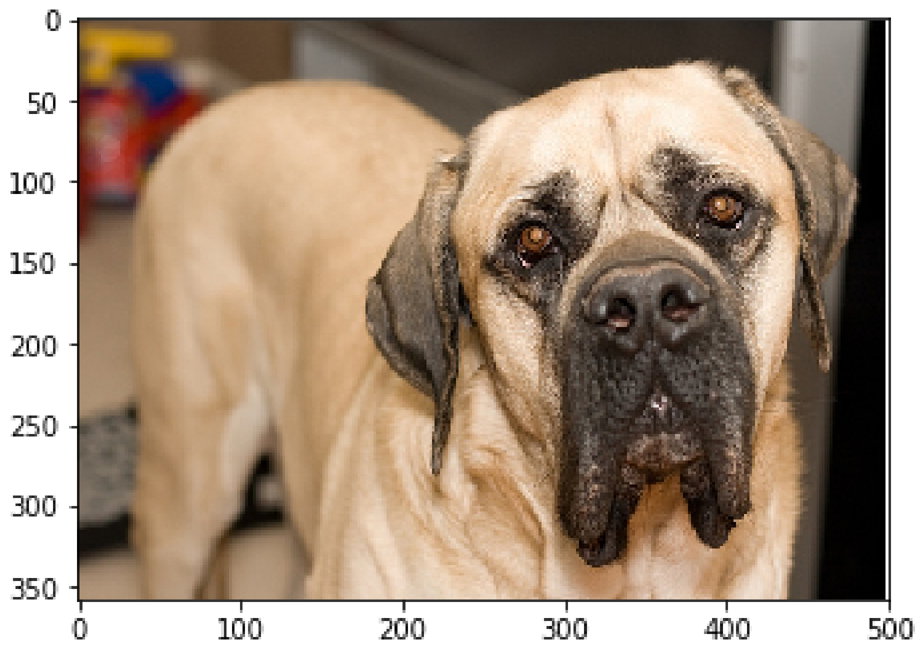
```
This is a picture of a ... Bullmastiff

-----------------------------------
```



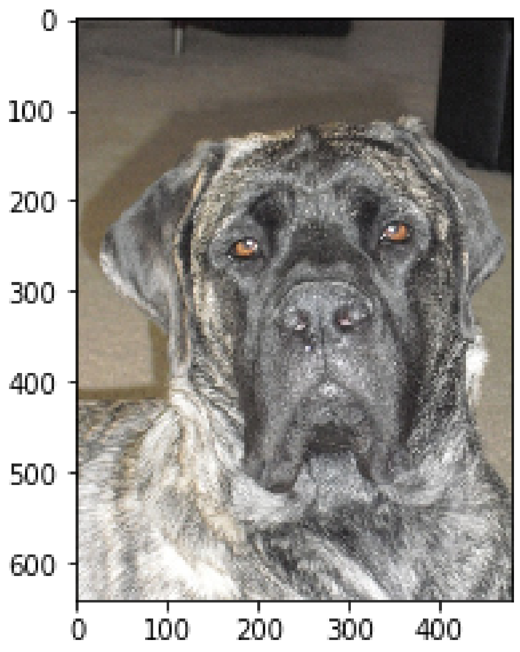
```
This is a picture of a ... Bullmastiff

-----------------------------------
```



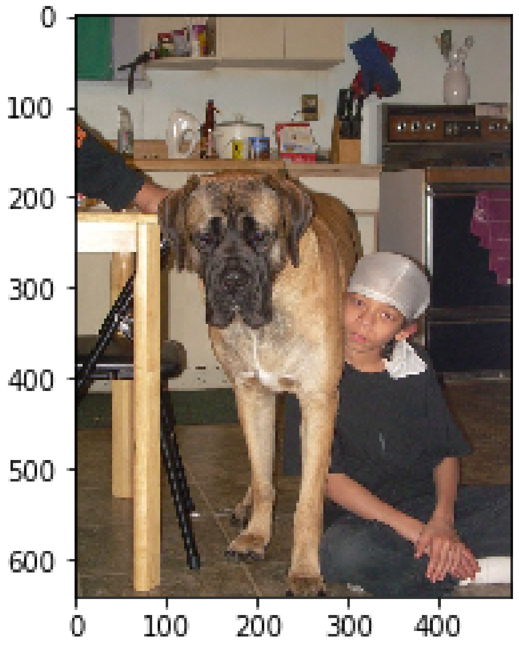
```
This is a picture of a ... Bullmastiff

-----------------------------------
```



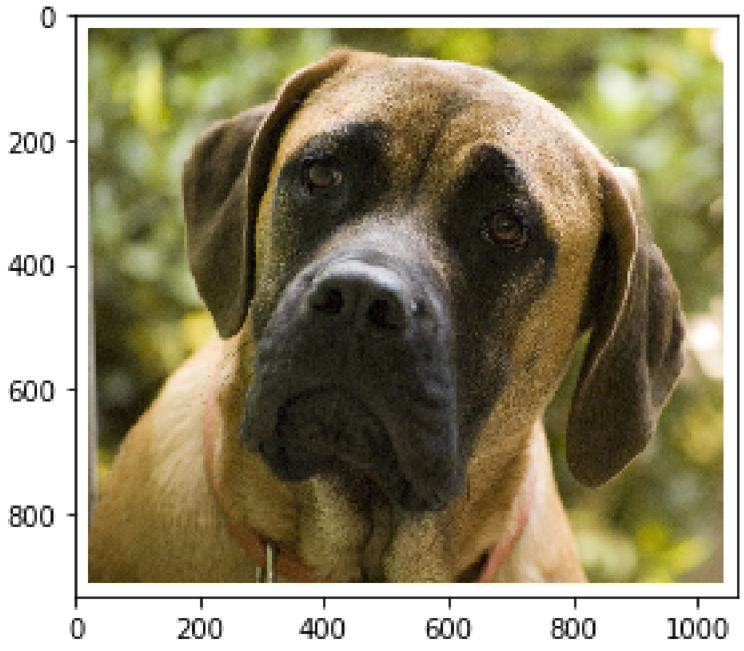
```
This is a picture of a ... Bullmastiff

-----------------------------------
```



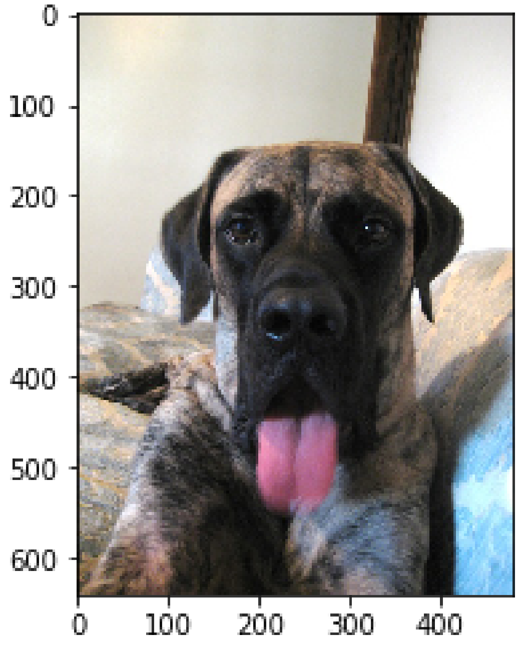
```
This is a picture of a ... Mastiff

-----------------------------------
```



```
This is a picture of a ... Mastiff

-----------------------------------
```



```
This is a picture of a ... Bullmastiff

-----------------------------------
```



```
This is a picture of a ... Bullmastiff

-----------------------------------
```

```
In [ ]:
```

```
In [ ]:
```