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!).

 Sample Dog Output

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

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

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.' % le
n(human_files))
print('There are %d total dog images.' % len(
dog_files))
```

```
There are 13233 total human imag
es.
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_det to detect human faces in images.

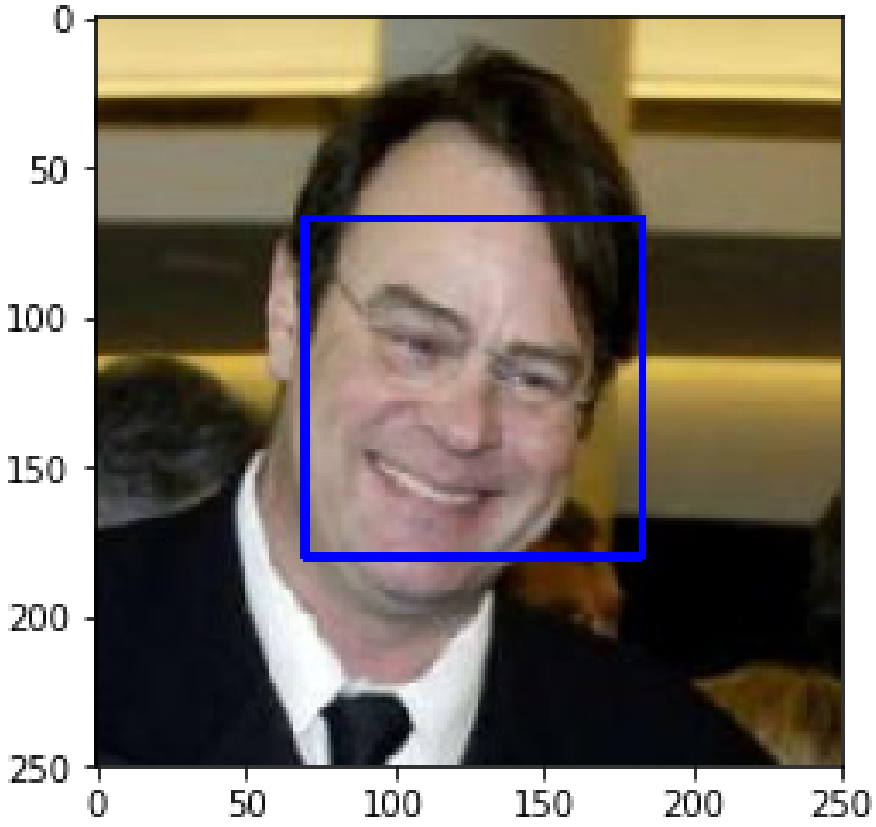
OpenCV provides many pre-trained face detectors, stored as XML files on github (https://github.com/opencv/opencv/tree/master/data/haarc We have downloaded one of these detectors and stored it in the `haarcascades` directory. In the next code cell, we demonstrate how to use this detector to find human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

In [2]:

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcas
cades/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.

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_det
ector algorithm
## on the images in human_files_short and dog
_files_short.
human_files_true, human_files_false = 0, 0
dog_files_true, dog_files_false = 0, 0

for i in range(100):
    if face_detector(human_files_short[i]):
        human_files_true += 1
    else:
        human_files_false += 1

    if face_detector(dog_files_short[i]):
        dog_files_true += 1
    else:
        dog_files_false += 1

print("Percentage of human faces in 100 human
image files is {} ".format(human_files_true))
print("Percentage of human faces in 100 dog i
mage files is {} ".format(dog_files_true))
```

```
Percentage of human faces in 100
human image files is 98
Percentage of human faces in 100
dog image files is 17
```

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 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 (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/), 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).

In [6]:

```
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()
```

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/Affenpinsch` 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).

In [7]:

```
from PIL import Image
import torchvision.transforms as transforms

def load_image(img_path):
    ''' Load in and transform an image'''
    image = Image.open(img_path).convert('RGB')


    # VGG-16 Takes 224x224 images as input, so we resize all of them and convert data to a normalized torch.FloatTensor
    in_transform = transforms.Compose([
                        transforms.Resize(224),
                        transforms.CenterCrop(224),
                        transforms.ToTensor(),
                        transforms.Normalize((0.5, 0.5, 0.5),
                                             (0.25, 0.25, 0.25))])

    # discard the transparent, alpha channel (that's the :3) and add the batch dimension
    image = in_transform(image)[:3,:,:].unsqueeze(0)

    return image
```

```
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 = load_image(img_path)
    if use_cuda:
        image = image.cuda()
    predict = VGG16(image)
    #predict_index = predict.data.argmax()
    predict_index = torch.max(predict, 1)[1].item()

    return predict_index # predicted class index
```

In [8]:

```
VGG16_predict(dog_files_short[1])
```

Out[8]:

243

(IMPLEMENTATION) Write a Dog Detector

While looking at the dictionary (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3al), 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 [9]:

```
### returns "True" if a dog is detected in th
e image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    index = VGG16_predict(img_path)
    if (index>=151 and index<=268):
        return True # return true for the abo
ve condition
    else:
        return False
```

In [10]:

```
dog_detector(dog_files_short[50])
```

Out[10]:

True

In [11]:

```
dog_detector(human_files_short[50])
```

Out[11]:

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 [12]:

```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.
from tqdm import tqdm

dog_predict_true, human_predict_true = 0, 0

for i in tqdm(range(len(human_files_short))):
    if dog_detector(human_files_short[i]):
        human_predict_true += 1
for i in tqdm(range(len(dog_files_short))):
    if dog_detector(dog_files_short[i]):
        dog_predict_true += 1

print("Percentage of human faces predicted through dog detector is {} ".format(human_predict_true))
print("Percentage of dog faces predicted through dog detector is {} ".format(dog_predict_true))
```

```
100%|██████████| 100/100 [00:03<
00:00, 29.63it/s]
100%|██████████| 100/100 [00:04<
00:00, 25.14it/s]

Percentage of human faces predic
ted through dog detector is 0
Percentage of dog faces predicte
d through dog detector is 100
```

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), ResNet-50 (http://pytorch.org/docs/master/torchvision/models.html#ic 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 [13]:

```
### (Optional)
### TODO: Report the performance of another p
re-trained network.
### Feel free to use as many code cells as ne
eded.
```

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 Wa

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

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.D](http://pytorch.org/docs/stable/data.html#torch.utils.data.D) 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 [14]:

```
import os
from torchvision import datasets
import torchvision.transforms as transforms
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
### TODO: Write data loaders for training, va
lidation, and test sets
## Specify appropriate transforms, and batch_
sizes

# no. of subprocesses used for data loading
num_workers = 0
# samples per batch to load
batch_size = 20

# Data Directory
data_dir = '/data/dog_images/'
train_data_dir = os.path.join(data_dir, 'trai
n/')
valid_data_dir = os.path.join(data_dir, 'vali
d/')
test_data_dir = os.path.join(data_dir, 'tes
t/')

# convert data into a normalised torch.FloatT
ensor
transform_train = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.RandomRotation(30),
```

```
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.
25, 0.25, 0.25))
    ])

transform_validation = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.
25, 0.25, 0.25))
    ])

# choose the training, valid and test dataset
s
train_data = datasets.ImageFolder(train_data_
dir, transform = transform_train)
valid_data = datasets.ImageFolder(valid_data_
dir, transform = transform_validation)
test_data = datasets.ImageFolder(test_data_di
r, transform = transform_validation)

# prepare data loaders
train_loader = torch.utils.data.DataLoader(tr
ain_data, batch_size = batch_size, num_worker
s = num_workers, shuffle = True)
valid_loader = torch.utils.data.DataLoader(va
lid_data, batch_size = batch_size, num_worker
s = num_workers, shuffle = False)
test_loader = torch.utils.data.DataLoader(val
id_data, batch_size = batch_size, num_workers
= num_workers, shuffle = False)
```

```
# loading from scratch
loaders_scratch = {'train': train_loader, 'va
lid': valid_loader, 'test': test_loader}
```

In [15]:

```
print("Total no. of classes in image data dir
ectory : {}" .format(len(train_data.classes
)))
print("Total no. of training data in image da
ta directory : {}" .format(len(train_data)))
print("Total no. of valid data in image data
 directory : {}" .format(len(valid_data)))
print("Total no. of test in image data direct
ory : {}" .format(len(test_data)))
```

```
Total no. of classes in image da
ta directory : 133
Total no. of training data in im
age data directory : 6680
Total no. of valid data in image
data directory : 835
Total no. of test in image data
directory : 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:

- For preprocessing the data, firstly i load the data directory, then transform the data for train, valid & test, after then data folder are loaded with transformation and finaly prepared the data loaders.
- I resized the images in 224x224 pixels, because VGG takes the same size specification as its input.
- Augmentation is done via *RandomRotation* of 30 units and *RandomHorizontalFip* and also conversion into Tensors and Normalization the images is been done.

(IMPLEMENTATION) Model Architecture

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

In [16]:

```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and com
plete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN

        # convolutional layer (sees 224*224*3
image tensor)
        self.conv1 = nn.Conv2d(3, 16, 3, padd
ing = 1)
        # convolutional layer (sees 112*112*1
6 image tensor)
        self.conv2 = nn.Conv2d(16, 32, 3, pad
ding = 1)
        # convolutional layer (sees 56*56*32
 image tensor)
        self.conv3 = nn.Conv2d(32, 64, 3, pad
ding = 1)
        # convolutional layer (sees 28*28*64
 image tensor)

        # maxpooling layer
        self.pool = nn.MaxPool2d(2, 2)

        # fully connected layers
```

```
        self.fc1 = nn.Linear(28*28*64, 500)
        self.fc2 = nn.Linear(500, 133)

        # dropout layer
        self.dropout = nn.Dropout(0.25)

    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)))

        # flatten image
        x = x.view(-1, 28*28*64)

        # dropout layer1
        x = self.dropout(x)

        # fully connected layer1
        x = F.relu(self.fc1(x))

        # dropout layer2
        x = self.dropout(x)

        # fully connected layer2
        x = self.fc2(x)

        return x

#-#-# You so NOT have to modify the code belo
w this line. #-#-#
```

```
# instantiate the CNN
model_scratch = Net()

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

In [17]:

```
print(model_scratch)
```

```
Net(
  (conv1): Conv2d(3, 16, kernel_
size=(3, 3), stride=(1, 1), padd
ing=(1, 1))
  (conv2): Conv2d(16, 32, kernel
_size=(3, 3), stride=(1, 1), pad
ding=(1, 1))
  (conv3): Conv2d(32, 64, kernel
_size=(3, 3), stride=(1, 1), pad
ding=(1, 1))
  (pool): MaxPool2d(kernel_size=
2, stride=2, padding=0, dilation
=1, ceil_mode=False)
  (fc1): Linear(in_features=5017
6, out_features=500, bias=True)
  (fc2): Linear(in_features=500,
out_features=133, bias=True)
  (dropout): Dropout(p=0.25)
)
```

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

Answer:

- The input is RGB image cropped into 224x224 pixel and the depth is 3 (3 colors), we will have our input with the shape 3x224x224.
- The desired no. of output is the same as no. of classes (here we have 133 classes in our images datasets).
- Therefore, I created 3 convolutional layers with relu activation function and one max pooling layer(2,2), the first layer takes (3,224,224) input and converted it into a depth of 16 layers, the filter used was 3x3 with stride of 1 and padding of 1.

1.first convolutional layer (sees 224 *224* 3 image tensor).

2.second convolutional layer (sees 112 *112* 16 tensor).

3.third convolutional layer (sees 56 *56* 32 tensor).

- After the 3rd convolutional layer, the shape of image tensor becomes 28 *28* 64
- one pool layer (2,2) was used in order to reduce the size of the images to half.
- Then flatten image input using view function to reshape the tensor.

- Then, two full connected Linear layers with relu activation function were added, and dropout layers for the hidden layers with a percentage of 25% to avoid the bias:
- first linear layer (64 *4* 4 -> 500).
- second linear layer (500 -> 133), as the output classes is 133.

(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) and [optimizer](http://pytorch.org/docs/stable/optim.html). Save the chosen loss function as `criterion_scratch`, and the optimizer as `optimizer_scratch` below.

In [19]:

```
import torch.optim as optim

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

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

(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)
at filepath `'model_scratch.pt'`.

In [20]:

```
def train(n_epochs, loaders, model, optimizer
, criterion, use_cuda, save_path):
    """returns trained model"""
    # initialize tracker for minimum validati
on loss
    valid_loss_min = np.Inf

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

        ####################
        # train the model #
        ####################
        model.train()
        for batch_idx, (data, target) in enum
erate(loaders['train']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), t
arget.cuda()
            ## find the loss and update the m
odel parameters accordingly
            ## record the average training lo
ss, using something like
            ## train_loss = train_loss + ((1
 / (batch_idx + 1)) * (loss.data - train_los
s))
```

```
            # clear the gradients of all opti
mized variables
            optimizer.zero_grad()
            # forward pass: compute predicted
outputs by passing inputs to the model
            output = model(data)
            # compute batch loss
            loss = criterion(output, target)
            # backpropagation
            loss.backward()
            # perform a single optimizer step
(parameter update)
            optimizer.step()
            # update training loss
            train_loss = train_loss + ((1 / (
batch_idx + 1)) * (loss.data - train_loss))

        ######################
        # validate the model #
        ######################
        model.eval()
        for batch_idx, (data, target) in enum
erate(loaders['valid']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), t
arget.cuda()
            ## update the average validation
 loss

            # forward pass
            output = model(data)
```

```
            # compute batch loss(validation l
oss)
            loss = criterion(output, target)
            # update validation loss
            valid_loss = valid_loss + ((1 / (
batch_idx + 1)) * (loss.data - valid_loss))

        # print training/validation statistic
s
        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
at(
            valid_loss_min,
            valid_loss))
            torch.save(model.state_dict(), sa
ve_path)
            valid_loss_min = valid_loss
    # return trained model
    return model


# train the model
model_scratch = train(25, loaders_scratch, mo
```

```
del_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('mod
el_scratch.pt'))
```

```
Epoch: 1        Training Loss:
4.880257        Validation Loss:
4.863909
Validation loss decreased (inf -
-> 4.863909).  Saving model ...
Epoch: 2        Training Loss:
4.819847        Validation Loss:
4.713901
Validation loss decreased (4.863
909 --> 4.713901).  Saving model
...
Epoch: 3        Training Loss:
4.598966        Validation Loss:
4.498900
Validation loss decreased (4.713
901 --> 4.498900).  Saving model
...
Epoch: 4        Training Loss:
4.415049        Validation Loss:
4.389143
Validation loss decreased (4.498
900 --> 4.389143).  Saving model
...
Epoch: 5        Training Loss:
4.310143        Validation Loss:
4.317900
Validation loss decreased (4.389
143 --> 4.317900).  Saving model
...
Epoch: 6        Training Loss:
4.232559        Validation Loss:
4.261103
```

```
Validation loss decreased (4.317
900 --> 4.261103).  Saving model
...
Epoch: 7        Training Loss:
4.163848        Validation Loss:
4.245881
Validation loss decreased (4.261
103 --> 4.245881).  Saving model
...
Epoch: 8        Training Loss:
4.088342        Validation Loss:
4.227340
Validation loss decreased (4.245
881 --> 4.227340).  Saving model
...
Epoch: 9        Training Loss:
4.014004        Validation Loss:
4.207281
Validation loss decreased (4.227
340 --> 4.207281).  Saving model
...
Epoch: 10       Training Loss:
3.949992        Validation Loss:
4.276871
Epoch: 11       Training Loss:
3.901620        Validation Loss:
4.081582
Validation loss decreased (4.207
281 --> 4.081582).  Saving model
...
Epoch: 12       Training Loss:
3.827834        Validation Loss:
4.093440
```

```
Epoch: 13       Training Loss:
3.757213        Validation Loss:
4.122375
Epoch: 14       Training Loss:
3.690010        Validation Loss:
4.096100
Epoch: 15       Training Loss:
3.610906        Validation Loss:
4.116344
Epoch: 16       Training Loss:
3.562145        Validation Loss:
4.043021
Validation loss decreased (4.081
582 --> 4.043021).  Saving model
...
Epoch: 17       Training Loss:
3.470571        Validation Loss:
4.188698
Epoch: 18       Training Loss:
3.413700        Validation Loss:
4.076882
Epoch: 19       Training Loss:
3.319431        Validation Loss:
3.980415
Validation loss decreased (4.043
021 --> 3.980415).  Saving model
...
Epoch: 20       Training Loss:
3.260282        Validation Loss:
4.025881
Epoch: 21       Training Loss:
3.185282        Validation Loss:
4.084913
```

```
Epoch: 22       Training Loss:
3.089607        Validation Loss:
4.125288
Epoch: 23       Training Loss:
3.018247        Validation Loss:
4.028902
Epoch: 24       Training Loss:
2.929642        Validation Loss:
4.109260
Epoch: 25       Training Loss:
2.858461        Validation Loss:
4.062340
```

(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]:

```
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 enumerat
e(loaders['test']):
        # move to GPU
        if use_cuda:
            data, target = data.cuda(), targe
t.cuda()
        # forward pass: compute predicted out
puts 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 pre
dicted class
        pred = output.data.max(1, keepdim=Tru
e)[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_l
oss))

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

# call test function
test(loaders_scratch, model_scratch, criterio
n_scratch, use_cuda)
```

```
Test Loss: 3.980415


Test Accuracy: 12% (108/835)
```

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.](http://pytorch.org/docs/master/data.html#torch.utils.data.) 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 [22]:

```
## TODO: Specify data loaders
import os
from torchvision import datasets
import torchvision.transforms as transforms
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
### TODO: Write data loaders for training, va
lidation, and test sets
## Specify appropriate transforms, and batch_
sizes

# no. of subprocesses used for data loading
num_workers = 0
# samples per batch to load
batch_size = 20
# percentage of training set to use as valida
tion
valid_size = 0.2

# Data Directory
data_dir = '/data/dog_images/'
train_data_dir = os.path.join(data_dir, 'trai
n/')
valid_data_dir = os.path.join(data_dir, 'vali
d/')
test_data_dir = os.path.join(data_dir, 'tes
t/')

# convert data into a normalised torch.FloatT
ensor
```

```
transform_train = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.RandomRotation(30),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.
25, 0.25, 0.25))
    ])

transform_validation = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.
25, 0.25, 0.25))
    ])

# choose the training, valid and test dataset
s
train_data = datasets.ImageFolder(train_data_
dir, transform = transform_train)
valid_data = datasets.ImageFolder(valid_data_
dir, transform = transform_validation)
test_data = datasets.ImageFolder(test_data_di
r, transform = transform_validation)

# prepare data loaders
train_loader = torch.utils.data.DataLoader(tr
ain_data, batch_size = batch_size, num_worker
s = num_workers, shuffle = True)
valid_loader = torch.utils.data.DataLoader(va
lid_data, batch_size = batch_size, num_worker
```

```
s = num_workers, shuffle = False)
test_loader = torch.utils.data.DataLoader(val
id_data, batch_size = batch_size, num_workers
= num_workers, shuffle = False)

# loading from scratch
loaders_transfer = {'train': train_loader, 'v
alid': valid_loader, 'test': test_loader}
```

In [23]:

```
# looking at the specialised convolutional an
d pooling layers of VGG16
# also its input and output features of fully
connected layers
print(VGG16)
print(VGG16.classifier[6].in_features)
print(VGG16.classifier[6].out_features)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_si
ze=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_s
ize=(3, 3), stride=(1, 1), paddi
ng=(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), padd
ing=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel
_size=(3, 3), stride=(1, 1), pad
ding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=
2, stride=2, padding=0, dilation
=1, ceil_mode=False)
    (10): Conv2d(128, 256, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (13): ReLU(inplace)
```

```
    (14): Conv2d(256, 256, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=
2, stride=2, padding=0, dilation
=1, ceil_mode=False)
    (17): Conv2d(256, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=
2, stride=2, padding=0, dilation
=1, ceil_mode=False)
    (24): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(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=2508
8, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=409
6, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=409
6, out_features=1000, bias=True)
  )
)
4096
1000
```

(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

## TODO: Specify model architecture
# loading the VGG16 model from pytorch
model_transfer = models.vgg16(pretrained = True)

# gradient update stops/Freezing pretrained parameters
for param in model_transfer.features.parameters():
    param.requires_grad = False

# producing a new linear layer to introduce our needed outputs
new_fc_layer = nn.Linear(model_transfer.classifier[6].in_features, 133)

# updating the last classifier layer
model_transfer.classifier[6] = new_fc_layer

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

In [25]:

```
print(model_transfer)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_si
ze=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_s
ize=(3, 3), stride=(1, 1), paddi
ng=(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), padd
ing=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel
_size=(3, 3), stride=(1, 1), pad
ding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=
2, stride=2, padding=0, dilation
=1, ceil_mode=False)
    (10): Conv2d(128, 256, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (13): ReLU(inplace)
```

```
    (14): Conv2d(256, 256, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=
2, stride=2, padding=0, dilation
=1, ceil_mode=False)
    (17): Conv2d(256, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=
2, stride=2, padding=0, dilation
=1, ceil_mode=False)
    (24): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kerne
l_size=(3, 3), stride=(1, 1), pa
dding=(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=2508
8, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=409
6, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=409
6, 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:

- I downloaded the VGG16 model and freezed all its parameters, as for transfer learning, I don't want to train them. They are already well trained.
- I just changed the Final layer of classifier (VGG16.classifier[6]) with my new Linear layer
- As the model has many Conv layers, which are pretrained, they are the ones perfectly suitable for producing feature maps.
- The classifier is also well trained.

Therefore,after training of the last linear layer, the model should have best accuracy.

(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) and optimizer (http://pytorch.org/docs/master/optim.html). Save the chosen loss function as `criterion_transfer`, and the optimizer as `optimizer_transfer` below.

In [26]:

```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer
.classifier.parameters(), lr = 0.01)
```

(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)
at filepath `'model_transfer.pt'`.

In [27]:

```
# train the model
n_epochs = 20
model_transfer = train(n_epochs, loaders_tran
sfer, model_transfer, optimizer_transfer, cri
terion_transfer,
                       use_cuda, 'model_trans
fer.pt')

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

```
Epoch: 1        Training Loss:
1.920205        Validation Loss:
0.742900
Validation loss decreased (inf -
-> 0.742900).  Saving model ...
Epoch: 2        Training Loss:
0.946691        Validation Loss:
0.632401
Validation loss decreased (0.742
900 --> 0.632401).  Saving model
...
Epoch: 3        Training Loss:
0.770598        Validation Loss:
0.518646
Validation loss decreased (0.632
401 --> 0.518646).  Saving model
...
Epoch: 4        Training Loss:
0.654239        Validation Loss:
0.542028
Epoch: 5        Training Loss:
0.555160        Validation Loss:
0.517031
Validation loss decreased (0.518
646 --> 0.517031).  Saving model
...
Epoch: 6        Training Loss:
0.524660        Validation Loss:
0.477228
Validation loss decreased (0.517
031 --> 0.477228).  Saving model
...
```

```
Epoch: 7        Training Loss:
0.466377        Validation Loss:
0.529039
Epoch: 8        Training Loss:
0.425194        Validation Loss:
0.479513
Epoch: 9        Training Loss:
0.391157        Validation Loss:
0.548726
Epoch: 10       Training Loss:
0.360926        Validation Loss:
0.475407
Validation loss decreased (0.477
228 --> 0.475407).  Saving model
...
Epoch: 11       Training Loss:
0.324178        Validation Loss:
0.454239
Validation loss decreased (0.475
407 --> 0.454239).  Saving model
...
Epoch: 12       Training Loss:
0.309422        Validation Loss:
0.461585
Epoch: 13       Training Loss:
0.291246        Validation Loss:
0.458360
Epoch: 14       Training Loss:
0.271114        Validation Loss:
0.482902
Epoch: 15       Training Loss:
0.247063        Validation Loss:
0.478848
```

```
Epoch: 16       Training Loss:
0.223336        Validation Loss:
0.452776
Validation loss decreased (0.454
239 --> 0.452776).  Saving model
...
Epoch: 17       Training Loss:
0.217183        Validation Loss:
0.471119
Epoch: 18       Training Loss:
0.207872        Validation Loss:
0.495870
Epoch: 19       Training Loss:
0.187619        Validation Loss:
0.545509
Epoch: 20       Training Loss:
0.174690        Validation Loss:
0.468300
```

(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 [28]:

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

```
Test Loss: 0.452776


Test Accuracy: 86% (720/835)
```

(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 [32]:

```
### TODO: Write a function that takes a path
 to an image as input
### and returns the dog breed that is predict
ed by the model.

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

def load_image(img_path):
    ''' Load in and transform an image'''
    image = Image.open(img_path).convert('RG
B')


    # The model takes 224x224 images as inpu
t, so we resize all of them and convert data
 to a normalized torch.FloatTensor
    in_transform = transforms.Compose([
                        transforms.Resize(224
),
                        transforms.CenterCrop
(224),
                        transforms.ToTensor
(),
                        transforms.Normalize
((0.5, 0.5, 0.5),

(0.25, 0.25, 0.25))])
```

```
    # discard the transparent, alpha channel
 (that's the :3) and add the batch dimension
    image = in_transform(image)[:3,:,:].unsqu
eeze(0)
    return image


def predict_breed_transfer(img_path):
    # load the image and return the predicted
breed
    image = load_image(img_path)
    if use_cuda:
        image = image.cuda()
    predict = model_transfer(image)
    model_transfer.eval()
    torch.no_grad()
    #predict_index = torch.argmax(predict)
    predict_index = torch.max(predict, 1)[1]

    return class_names[predict_index]
```

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!

 Sample Human Output

(IMPLEMENTATION) Write your Algorithm

In [33]:

```
### TODO: Write your algorithm.
### Feel free to use as many code cells as ne
eded.

def run_app(img_path):
    ## handle cases for a human face, dog, an
d neither
    image = Image.open(img_path) # loading th
e image

    if dog_detector(img_path):
        plt.imshow(image)
        plt.show()
        prediction = predict_breed_transfer(i
mg_path)
        print("Hey!!!There is a doggy.It look
s like {}".format(prediction))

    elif face_detector(img_path):
        plt.imshow(image)
        plt.show()
        prediction = predict_breed_transfer(i
mg_path)
        print("Hello Human!!!If you were a do
ggy, you would look like {}".format(predictio
n))

    else:
        plt.imshow(img)
        plt.show()
```

```
        print("Oh Sorry!!!Neither human nor d
oggy detected")
```

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 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)

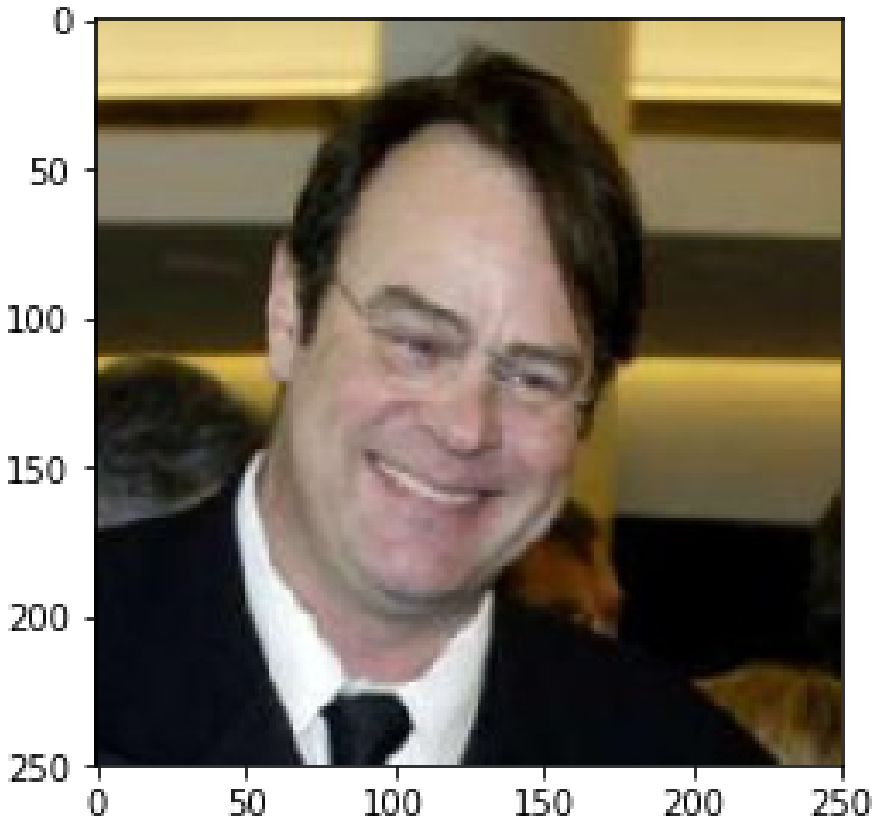
- The scratch model outputs an accuracy of 12%, which is good but needs more improvement.It looks like the model is a bit overfitting the data, so the validation loss increases even in 25 epochs.

I think we can improve the scratch model accuracy by:

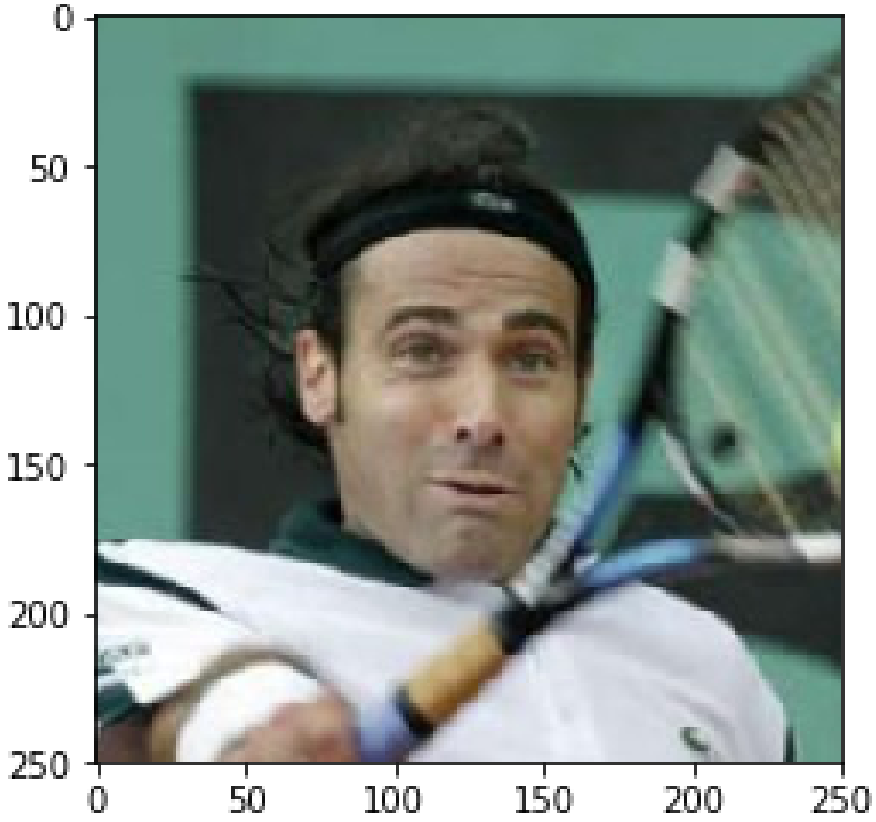
- Increasing the dropout layer by a value of 0.1 as I go by each classification layer.Constant Dropout is likely overfitting the data in the later classification layers.
- I have used 3 convolutional and 2 classification layers.Perhaps increasing them suffices my accuracy.
- We saw that even in 25 epochs, the validation loss increases.Perhaps adding a bit more training data and increasing the no.of epochs may generalise the model better.

In [34]:

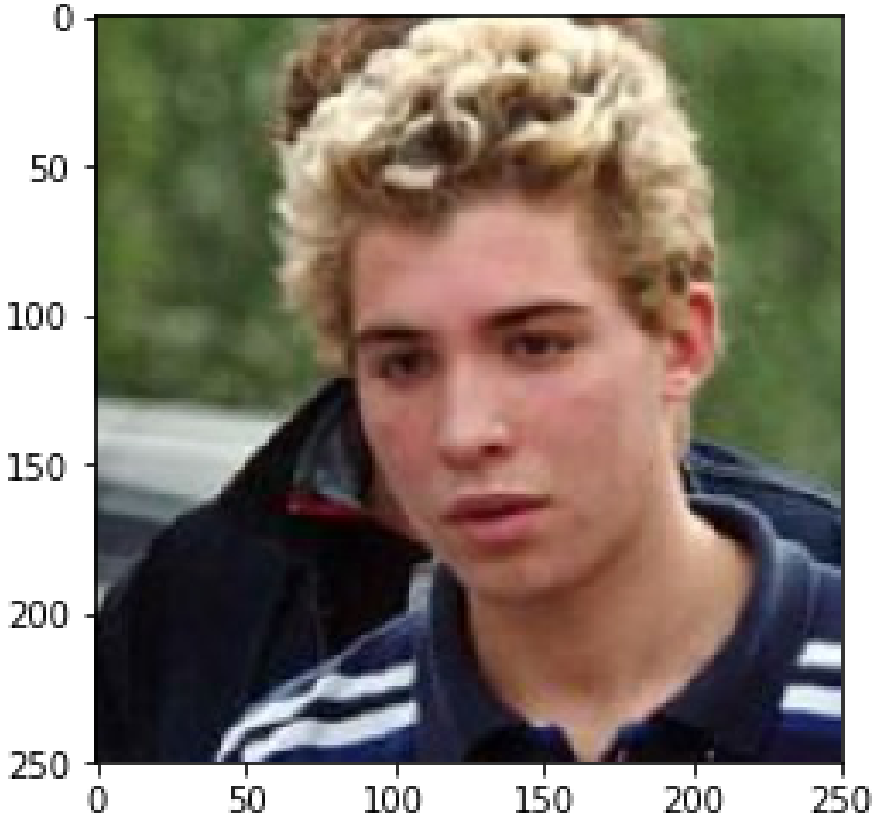
```
## TODO: Execute your algorithm from Step 6 o
n
## at least 6 images on your computer.
## Feel free to use as many code cells as nee
ded.

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



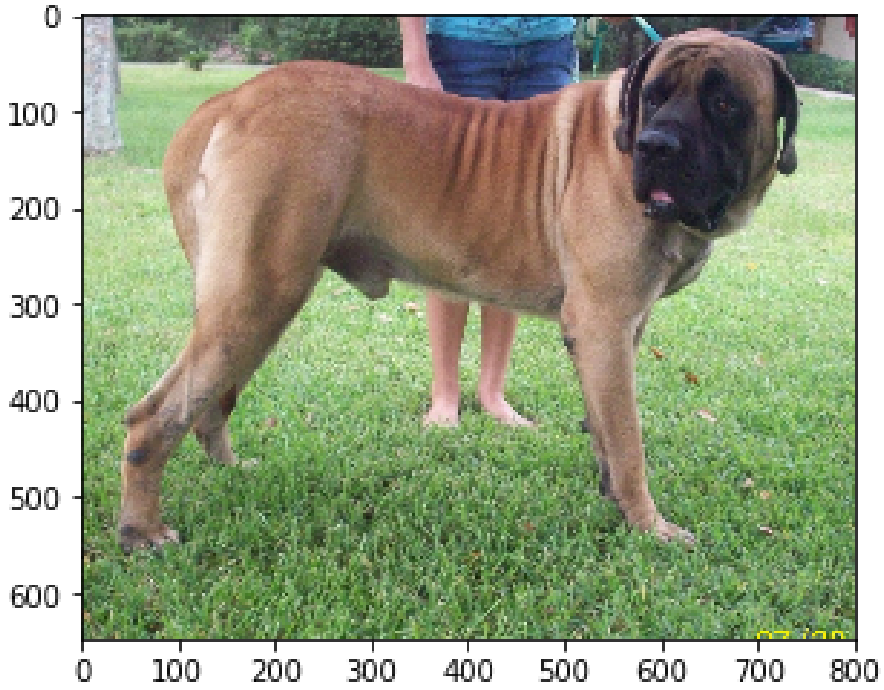

```
Hello Human!!!If you were a dogg
y, you would look like Brittany
```



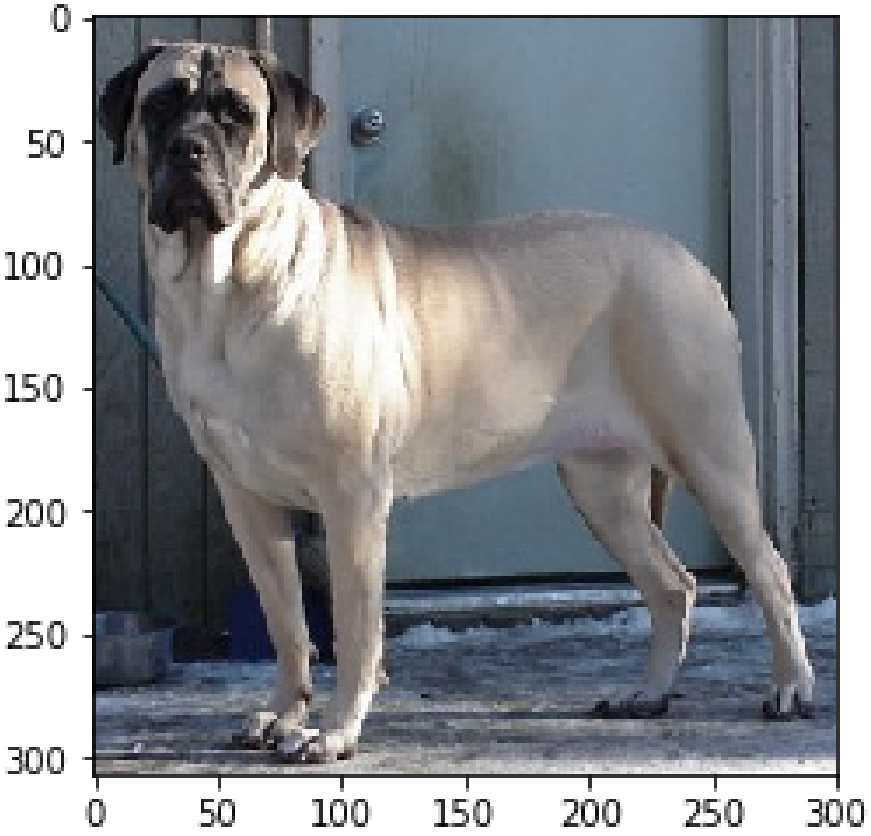
Hello Human!!!If you were a dogg
y, you would look like Silky ter
rier



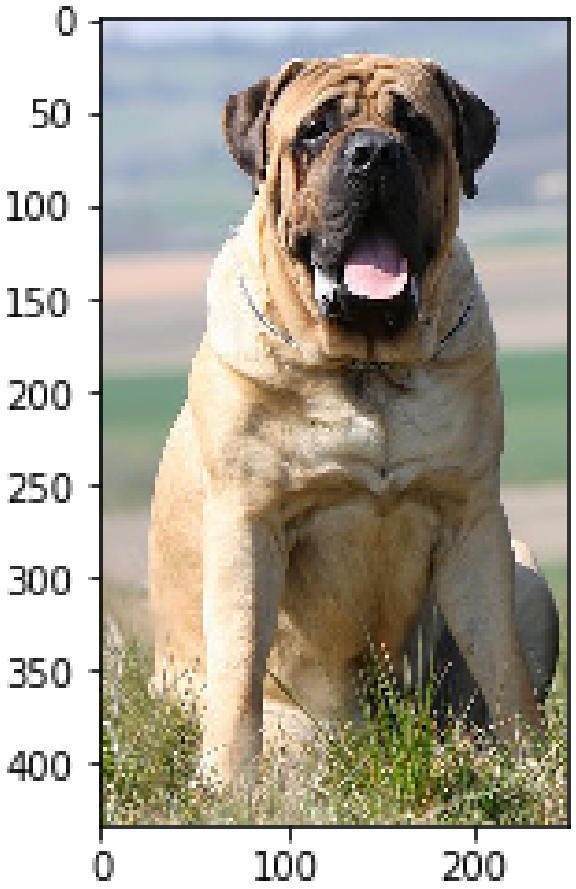

```
Hello Human!!!If you were a dogg
y, you would look like American
water spaniel
```



Hey!!!There is a doggy.It looks like Mastiff



Hey!!!There is a doggy.It looks like Mastiff

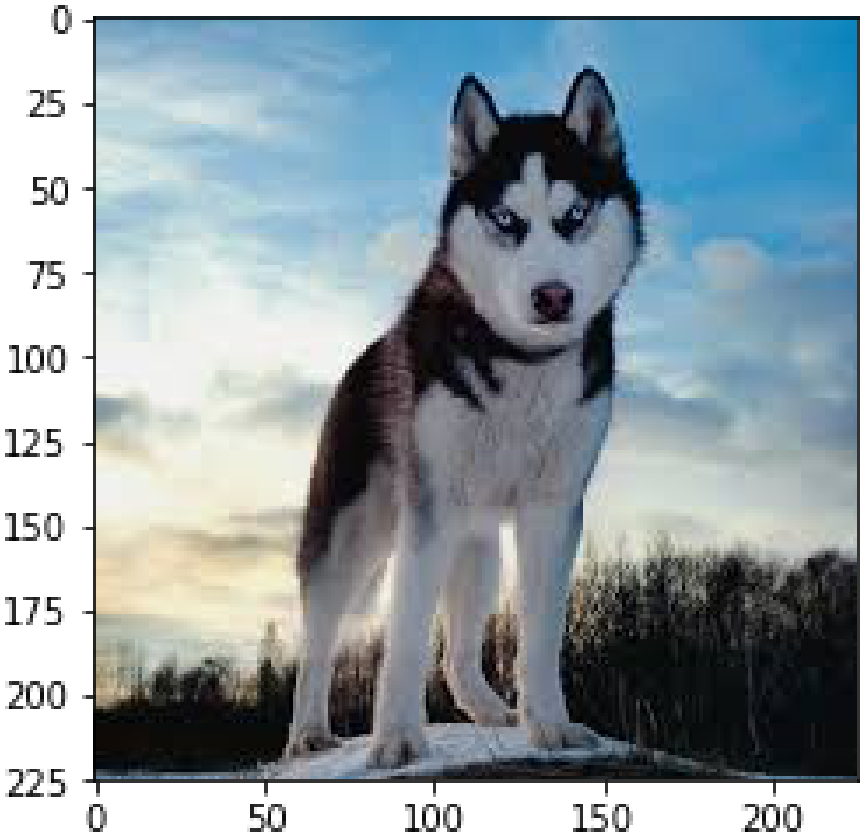


Hey!!!There is a doggy.It looks like Mastiff

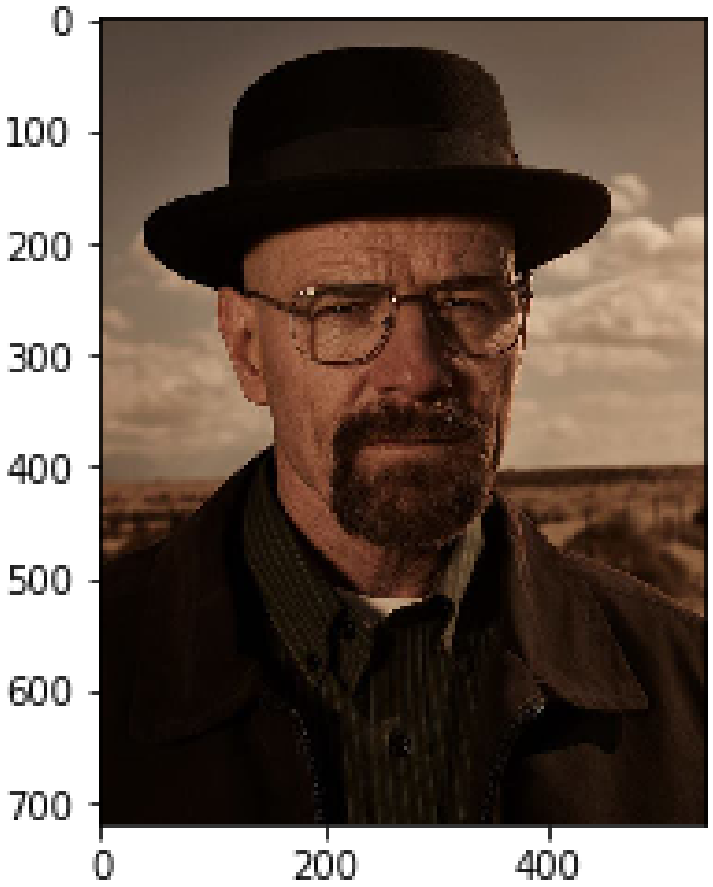
In [36]:

```
#Testing the algorithm with six new uploaded
 images
# Specimen Code
my_images = glob("./New Images/*")
for file in np.hstack((my_images)):
    run_app(file)
```

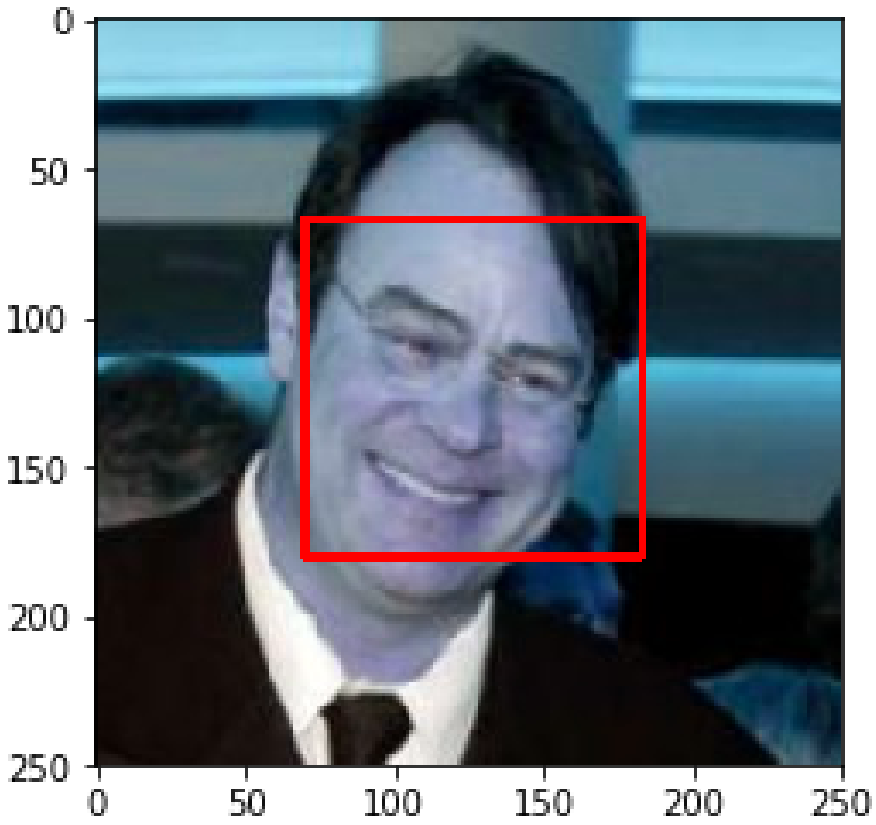
Hey!!!There is a doggy.It looks like Komondor



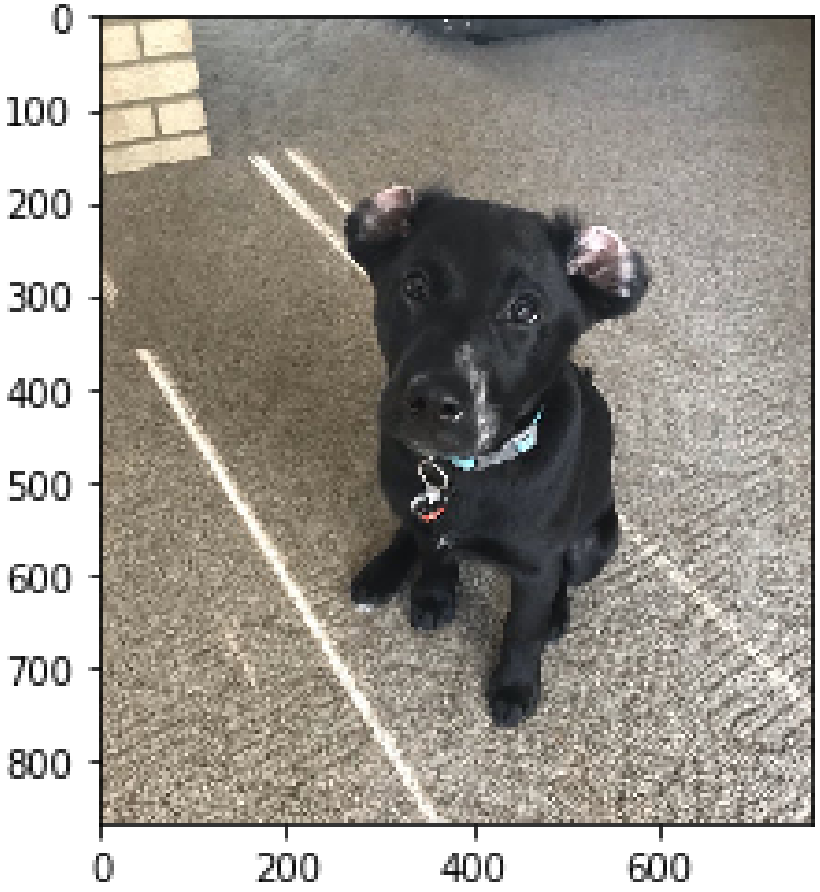
```
Hey!!!There is a doggy.It looks
like Alaskan malamute
```



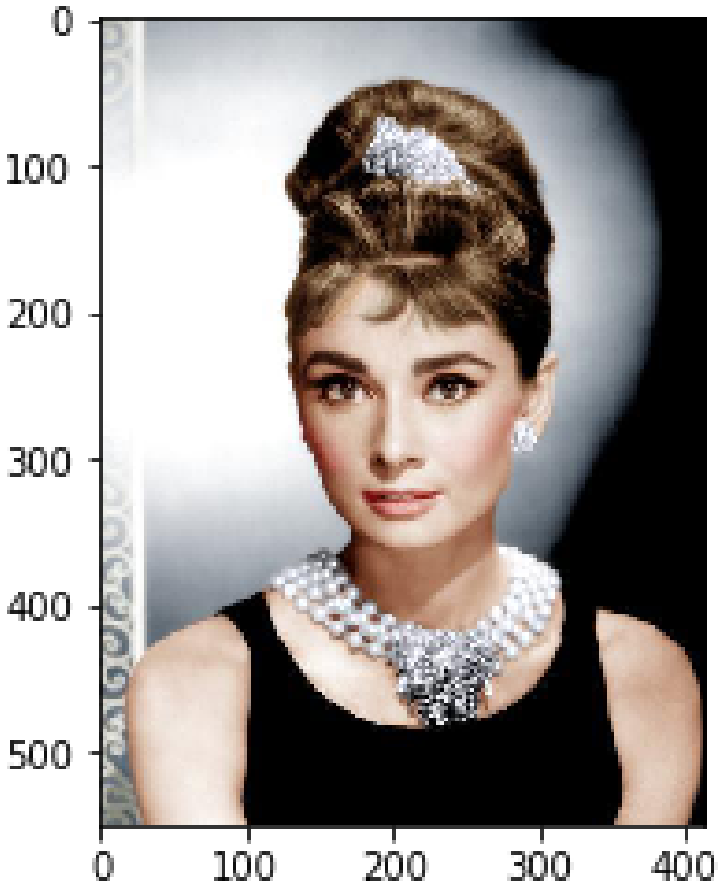

```
Hello Human!!!If you were a dogg
y, you would look like Pharaoh h
ound
```




```
Oh Sorry!!!Neither human nor dog
gy detected
```



Hey!!!There is a doggy.It looks like Manchester terrier




```
Hello Human!!!If you were a dogg
y, you would look like Cocker sp
aniel
```

In [ ]: