dog_app

March 9, 2019

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: * 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 [1]: import numpy as np
        from glob import glob

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

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

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

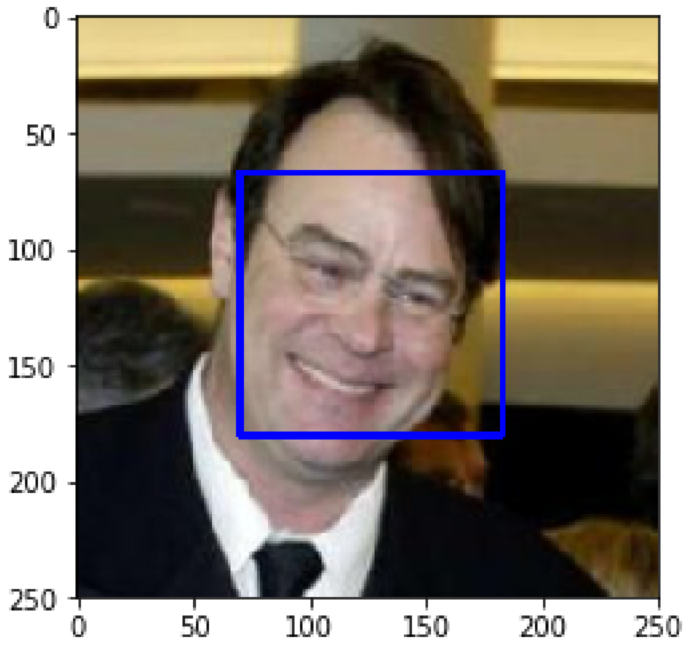
## Step 1: Detect Humans

In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers 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 [2]: import cv2
        import matplotlib.pyplot as plt
        %matplotlib inline

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

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

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

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

        # get bounding box for each detected face
        for (x,y,w,h) in faces:
            # add bounding box to color image
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        # convert BGR image to RGB for plotting
        cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

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

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



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 [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
```

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 [4]: from tqdm import tqdm

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

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

        ## TODO: Test the performance of the face_detector algorithm
        ## on the images in human_files_short and dog_files_short.
        h_count = 0
        d_count = 0

        for human in human_files_short:
            if face_detector(human):
                h_count += 1

        for human in dog_files_short:
            if face_detector(human):
                d_count += 1

        print("Humans count in human files short ", h_count, " out of 100")
        print("Humans count in dogs files short", d_count, " out of 100")
```

```
Humans count in human files short  98  out of 100
Humans count in dogs files short 17  out of 100
```

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 to detect dogs in images.

1.1.3 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 [5]: import torch
        import torchvision.models as models

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

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

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

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.torch/models/vgg
100%|| 553433881/553433881 [00:21<00:00, 26106535.51it/s]
```

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.

1.1.4 (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 [6]: from PIL import Image
        import torchvision.transforms as transforms

        def VGG16_predict(img_path):
```

```
    '''
    Use pre-trained VGG-16 model to obtain index corresponding to
    predicted ImageNet class for image at specified path

    Args:
        img_path: path to an image

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

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image
    img = Image.open(img_path).convert('RGB')
    in_transform = transforms.Compose([
                        transforms.Resize(size=(244, 244)),
                        transforms.ToTensor()
                            ])
    image = in_transform(img)[:3,:,:].unsqueeze(0)
    if use_cuda:
        image = image.cuda()
    ret = VGG16(image)
    return torch.max(ret,1)[1].item() # predicted class index
```

```
In [7]: VGG16_predict(dog_files_short[55])
```

```
Out[7]: 163
```

1.1.5 (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 [8]: ### returns "True" if a dog is detected in the 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
            else:
                return False
            # true/false
```

```
In [9]: dog_detector(dog_files_short[0])

Out[9]: True

In [10]: dog_detector(dog_files_short[80])

Out[10]: True
```

1.1.6 (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.
         h_count = 0
         d_count = 0

         for f in human_files_short:
             if dog_detector(f):
                 h_count += 1

         for f in dog_files_short:
             if dog_detector(f):
                 d_count += 1

         print("Dogs count in human files short ", h_count, " out of 100")
         print("Dogs count in dogs files short", d_count, " out of 100")
```

```
Dogs count in human files short  0  out of 100
Dogs count in dogs files short 99  out of 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, 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 [13]: ### (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!

1.1.7 (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 [11]: import os
         import torch
         from torchvision import datasets, transforms

         ### TODO: Write data loaders for training, validation, and test sets
         ## Specify appropriate transforms, and batch_sizes
         train_dir = '/data/dog_images/train'
         valid_dir = '/data/dog_images/valid'
         test_dir = '/data/dog_images/test'
```

```
train_transforms = transforms.Compose([transforms.RandomRotation(30),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406],
                                                            [0.229, 0.224, 0.225])])

valid_transforms = transforms.Compose([transforms.Resize(256),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      transforms.Normalize([0.485, 0.456, 0.406],
                                                           [0.229, 0.224, 0.225])])

test_transforms = transforms.Compose([transforms.Resize(256),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      transforms.Normalize([0.485, 0.456, 0.406],
                                                           [0.229, 0.224, 0.225])])

train_image_datasets = datasets.ImageFolder(train_dir, transform=train_transforms)
valid_image_datasets = datasets.ImageFolder(valid_dir, transform=valid_transforms)
test_image_datasets = datasets.ImageFolder(test_dir, transform=test_transforms)

train_loader = torch.utils.data.DataLoader(train_image_datasets, batch_size=32, shuffle
valid_loader = torch.utils.data.DataLoader(valid_image_datasets, batch_size=32, shuffle
test_loader = torch.utils.data.DataLoader(test_image_datasets, batch_size=32, shuffle=T

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

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 resized images of valid and test set to 256, and center cropping them to 224 x 224. For the training set I have augmented data by doing random rotation by 30 degrees,also have applied random resizing by 224 x 224 and finally done random horizontal flip. Since the dataset is very less I have augmented data to avoid overfitting and to have more generalization. And finally applied standard normalization to all 3 sets of data.

1.1.8 (IMPLEMENTATION) Model Architecture

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

```
In [12]: import numpy as np
         from glob import glob
         from PIL import ImageFile
         import matplotlib.pyplot as plt
         %matplotlib inline

         ImageFile.LOAD_TRUNCATED_IMAGES =True
         use_cuda = torch.cuda.is_available()
```

```
In [13]: num_classes = np.array(glob("/data/dog_images/test/*"))
         len(num_classes)
```

```
Out[13]: 133
```

```
In [17]: import torch.nn as nn
         import torch.nn.functional as F

         # define the CNN architecture
         class Net(nn.Module):
             ### TODO: choose an architecture, and complete the class
             def __init__(self):
                 super(Net, self).__init__()
                 ## Define layers of a CNN
                 self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
                 self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
                 self.conv3 = nn.Conv2d(32, 64, 3, padding=1)

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

                 # fully-connected
                 self.fc1 = nn.Linear(28*28*64, 2048)
                 self.fc2 = nn.Linear(2048, 133)

                 # drop-out
                 self.dropout = nn.Dropout(0.15)

             def forward(self, x):
                 ## Define forward behavior
                 x = F.relu(self.conv1(x))
                 x = self.pool(x)
                 x = F.relu(self.conv2(x))
                 x = self.pool(x)
                 x = F.relu(self.conv3(x))
                 x = self.pool(x)

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

```
            x = F.relu(self.fc1(x))
            x = self.dropout(x)
            x = self.fc2(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 [18]: model_scratch
```

```
Out[18]: 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))
           (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
           (fc1): Linear(in_features=50176, out_features=2048, bias=True)
           (fc2): Linear(in_features=2048, out_features=133, bias=True)
           (dropout): Dropout(p=0.15)
         )
```

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

Answer: My network consists of 3 convolution layers with a maxpooling layer of stride and kernel size of 2 and 3 fully connected layers with a dropout layer of 15%.

=> First convolution layer takes in the inputs with 3 layers and produces stack of 16 layers with kernel size 3, stride and padding kept to 1

=> Then I am passing it to Maxpooling layer of kernel size 2 and stride 2

=> Second convolution layer takes in the inputs with 16 layers and produces stack of 32 layers with kernel size 3, stride and padding kept to 1

=> Then I am passing it to Maxpooling layer of kernel size 2 and stride 2

=> Third convolution layer takes in the inputs with 32 layers and produces stack of 64 layers with kernel size 3, stride and padding kept to 1

=> Then I am passing it to Maxpooling layer of kernel size 2 and stride 2

=> Fourth convolution layer takes in the inputs with 64 layers and produces stack of 128 layers with kernel size 3, stride and padding kept to 1

=> Then I am passing it to Maxpooling layer of kernel size 2 and stride 2

=> Flattening the features to pass it to fully connected layers. With the previous maxpooling the shape will be of 28x28x64

=> First Fully connected layer takes in the features flattened of size 50176 and produces out features of 2048 and applying ReLU activation function to it.

=> To avoid overfitting applying a dropout of 15%

=> Second Fully connected layer takes in the features flattened of size 2048 and produces out features of 512 and applying ReLU activation function to it. Then apply a dropout of 15%.

=> Third Fully connected layer takes in the features flattened of size 512 and produces out features of 133.

=> Third fully connected layer will produce the output which predicts dog breed class.

1.1.9 (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 [19]: import torch.optim as optim
         import numpy as np

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

         ### TODO: select optimizer
         optimizer_scratch = optim.SGD(model_scratch.parameters(), lr=0.1)
```

1.1.10 (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 [20]: def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path):
             """returns trained model"""
             # initialize tracker for minimum validation loss
             valid_loss_min = np.Inf

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

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

```
                loss = criterion(output, target)
                loss.backward()
                optimizer.step()
                train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss)
            ######################
            # validate the model #
            ######################
            model.eval()
            for batch_idx, (data, target) in enumerate(loaders['valid']):
                # move to GPU
                if use_cuda:
                    data, target = data.cuda(), target.cuda()
                ## update the average validation loss
                output = model(data)
                loss = criterion(output, target)
                valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss)

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

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

```
In [21]: # train the model
         model_scratch = train(20, 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: 4.839423         Validation Loss: 4.814130
Validation loss decreased (inf --> 4.814130).  Saving model ...
Epoch: 2         Training Loss: 4.730881         Validation Loss: 4.617319
Validation loss decreased (4.814130 --> 4.617319).  Saving model ...
Epoch: 3         Training Loss: 4.639011         Validation Loss: 4.416156
Validation loss decreased (4.617319 --> 4.416156).  Saving model ...
Epoch: 4         Training Loss: 4.586841         Validation Loss: 4.418290
```

```
Epoch: 5         Training Loss: 4.565521        Validation Loss: 4.422129
Epoch: 6         Training Loss: 4.499034        Validation Loss: 4.398131
Validation loss decreased (4.416156 --> 4.398131).  Saving model ...
Epoch: 7         Training Loss: 4.435208        Validation Loss: 4.369047
Validation loss decreased (4.398131 --> 4.369047).  Saving model ...
Epoch: 8         Training Loss: 4.429318        Validation Loss: 4.284343
Validation loss decreased (4.369047 --> 4.284343).  Saving model ...
Epoch: 9         Training Loss: 4.369661        Validation Loss: 4.472616
Epoch: 10         Training Loss: 4.315464        Validation Loss: 4.197600
Validation loss decreased (4.284343 --> 4.197600).  Saving model ...
Epoch: 11         Training Loss: 4.254987        Validation Loss: 4.186837
Validation loss decreased (4.197600 --> 4.186837).  Saving model ...
Epoch: 12         Training Loss: 4.204750        Validation Loss: 4.063301
Validation loss decreased (4.186837 --> 4.063301).  Saving model ...
Epoch: 13         Training Loss: 4.142750        Validation Loss: 4.020271
Validation loss decreased (4.063301 --> 4.020271).  Saving model ...
Epoch: 14         Training Loss: 4.126878        Validation Loss: 3.965173
Validation loss decreased (4.020271 --> 3.965173).  Saving model ...
Epoch: 15         Training Loss: 4.075943        Validation Loss: 3.924674
Validation loss decreased (3.965173 --> 3.924674).  Saving model ...
Epoch: 16         Training Loss: 4.022735        Validation Loss: 3.812227
Validation loss decreased (3.924674 --> 3.812227).  Saving model ...
Epoch: 17         Training Loss: 4.012201        Validation Loss: 3.817307
Epoch: 18         Training Loss: 4.016880        Validation Loss: 3.722237
Validation loss decreased (3.812227 --> 3.722237).  Saving model ...
Epoch: 19         Training Loss: 3.922482        Validation Loss: 3.847304
Epoch: 20         Training Loss: 3.912327        Validation Loss: 3.985157
```

1.1.11 (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 [22]: 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))
```

```
In [23]: # call test function
         test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 3.785149


Test Accuracy: 12% (106/836)
```

## 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.

1.1.12 (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 [14]: ## TODO: Specify data loaders
         loaders_transfer = loaders_scratch
```

1.1.13 (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 [16]: import torchvision.models as models
         import torch.nn as nn
```

```
## TODO: Specify model architecture

model_transfer = models.vgg19(pretrained=True)

for param in model_transfer.parameters():
    param.requires_grad_ = False

classifier = nn.Sequential(
                        nn.Linear(model_transfer.classifier[0].in_features, 700),
                        nn.ReLU(),
                        nn.Linear(700, 133))

model_transfer.classifier = classifier

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

```
Downloading: "https://download.pytorch.org/models/vgg19-dcbb9e9d.pth" to /root/.torch/models/vgg
100%|| 574673361/574673361 [00:09<00:00, 62729939.75it/s]
```

In [26]:
```
model_transfer
```

Out[26]:
```
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): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Conv2d(256, 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): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(inplace)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=700, bias=True)
    (1): ReLU()
    (2): Linear(in_features=700, 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 am using vgg19 model for the transfer learning, because I feel it is a small network which is enough to classify the dog breeds. I am creating 2 fully connected layers and replacing it with the last classifier layer of the vgg19 model. My last layer produces outputs of 133. This will predict the dog breed.

1.1.14 (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 [27]: criterion_transfer = nn.CrossEntropyLoss()
         optimizer_transfer = optim.SGD(model_transfer.parameters(), lr=0.001)
```

1.1.15 (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 [28]: # train the model
         model_transfer = train(5, loaders_transfer, model_transfer, optimizer_transfer, criteri

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

```
Epoch: 1         Training Loss: 4.018939        Validation Loss: 1.809410
Validation loss decreased (inf --> 1.809410).  Saving model ...
Epoch: 2         Training Loss: 2.015372        Validation Loss: 0.968666
Validation loss decreased (1.809410 --> 0.968666).  Saving model ...
Epoch: 3         Training Loss: 1.484649        Validation Loss: 0.777202
Validation loss decreased (0.968666 --> 0.777202).  Saving model ...
Epoch: 4         Training Loss: 1.287650        Validation Loss: 0.693065
Validation loss decreased (0.777202 --> 0.693065).  Saving model ...
Epoch: 5         Training Loss: 1.179836        Validation Loss: 0.699238
```

1.1.16 (IMPLEMENTATION) Test the Model

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

```
In [29]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.712813


Test Accuracy: 79% (663/836)
```

1.1.17 (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 [17]: from PIL import Image
         model_transfer.load_state_dict(torch.load('model_transfer.pt', map_location=lambda stor
```

```
In [18]: ### 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 loaders_transfer['train'].dataset

         def predict_breed_transfer(img_path):
             # load the image and return the predicted breed
             image = Image.open(img_path).convert('RGB')
             prediction_transform = transforms.Compose([transforms.Resize(size=(224, 224)),
                                                  transforms.ToTensor(),
                                                  transforms.Normalize([0.485, 0.456, 0.406],
                                                                       [0.229, 0.224, 0.225])])
             image = prediction_transform(image)[:3,:,:].unsqueeze(0)
             model = model_transfer.cpu()
             model.eval()
             index = torch.argmax(model(image))
             return class_names[index]
```

```
In [32]: # img_path = os.path.join('./images', img_file)
         prediction = predict_breed_transfer(dog_files[1997])
         print("Predition breed:",prediction)
```

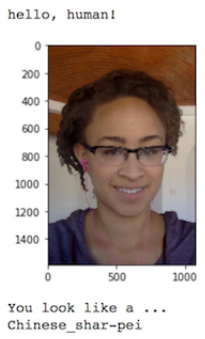
```
Predition breed: Bernese mountain dog
```

```
In [33]: dog_files[1997]
```

```
Out[33]: '/data/dog_images/train/023.Bernese_mountain_dog/Bernese_mountain_dog_01662.jpg'
```

```
In [34]: for dogs in dog_files[:100]:
             prediction = predict_breed_transfer(dogs)
             print('Filename', dogs, 'Prediction class', prediction)
```

```
Filename /data/dog_images/train/103.Mastiff/Mastiff_06833.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06826.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06871.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06812.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06831.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06867.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06845.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06865.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06866.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06862.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06853.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06877.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06839.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06824.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06814.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06842.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06813.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06851.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06820.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06852.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06846.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06870.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06854.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06817.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06819.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06850.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06844.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06841.jpg Prediction class Neapolitan mastif
Filename /data/dog_images/train/103.Mastiff/Mastiff_06858.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06861.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06818.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06864.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06863.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06876.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06811.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06848.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06832.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06849.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06843.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06860.jpg Prediction class Pointer
Filename /data/dog_images/train/103.Mastiff/Mastiff_06834.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06857.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06829.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06822.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06840.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06875.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06874.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06837.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06879.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06856.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06868.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06821.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06869.jpg Prediction class Cane corso
Filename /data/dog_images/train/103.Mastiff/Mastiff_06835.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06838.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06809.jpg Prediction class Mastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06872.jpg Prediction class Bullmastiff
Filename /data/dog_images/train/103.Mastiff/Mastiff_06828.jpg Prediction class Mastiff
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04181.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04209.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04192.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04179.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04198.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04171.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04182.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04174.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04167.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04193.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04183.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04163.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04187.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04207.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04166.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04168.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04212.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04204.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04160.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04180.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04210.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04186.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04199.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04185.jpg Prediction cla
```



Sample Human Output

```
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04176.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04159.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04191.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04201.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04200.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04170.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04189.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04214.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04157.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04188.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04196.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04195.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04211.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04162.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04206.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04161.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04205.jpg Prediction cla
Filename /data/dog_images/train/059.Doberman_pinscher/Doberman_pinscher_04173.jpg Prediction cla
```

## 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.

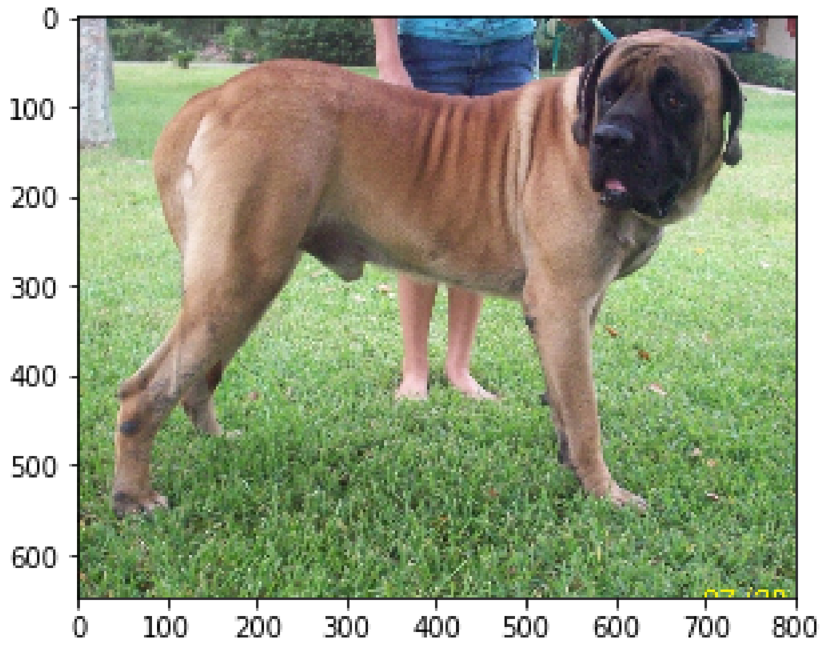
Some sample output for our algorithm is provided below, but feel free to design your own user experience!

1.1.18 (IMPLEMENTATION) Write your Algorithm

```
In [19]: ### 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
             image = Image.open(img_path)
             if dog_detector(img_path) is True:
                 prediction = predict_breed_transfer(img_path)
                 print("Dogs Detected!")
                 plt.imshow(image)
                 plt.show()
                 print("It looks like a {0}".format(prediction))
             elif face_detector(img_path) > 0:
                 prediction = predict_breed_transfer(img_path)
                 print("hello, human!")
                 plt.imshow(image)
                 plt.show()
                 print("If you were a dog..You may look like a {0}".format(prediction))
             else:
                 print("You are a alien! X..X")
```

```
In [20]: run_app(dog_files[0])
```

```
Dogs Detected!
```



```
It looks like a Bullmastiff
```

## 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?

1.1.19 (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.

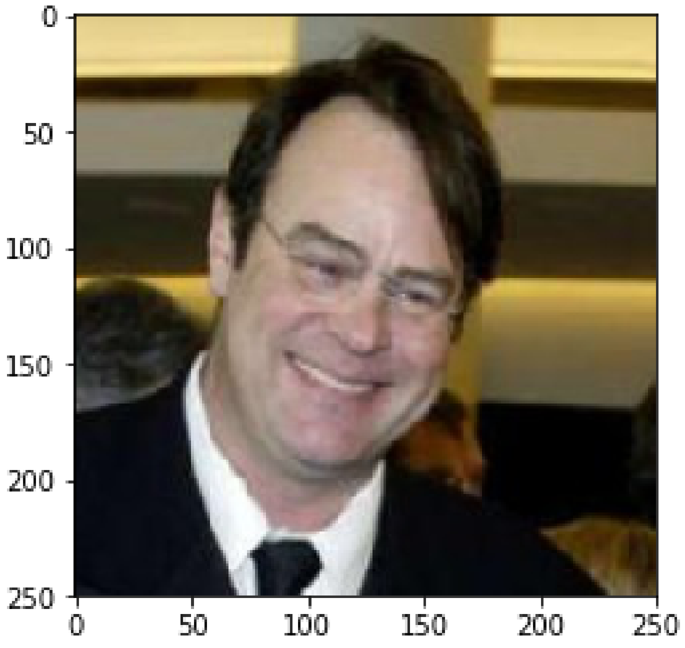
Question 6: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of improvement for your algorithm.

Answer: I am satisfied with the output for human faces. But for dogs since the accuracy is less its prediction is not that good.

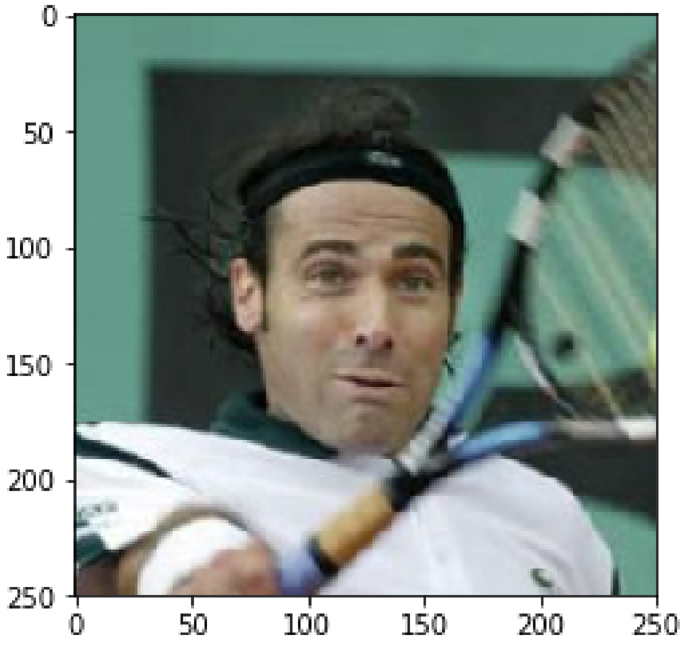
To improve my algorithm, first thing I have to improve my model accuracy may be by using other pre-trained models like resnet152 etc. Next I have to train it for more epochs and also fine tuning will help to increase the accuracy. Next thing I can use other optimizers or loss functions to achieve accuracy. ALso i think there should be a option to show unknown objects

```
In [21]: ## TODO: Execute your algorithm from Step 6 on
         ## at least 6 images on your computer.
         ## Feel free to use as many code cells as needed.

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

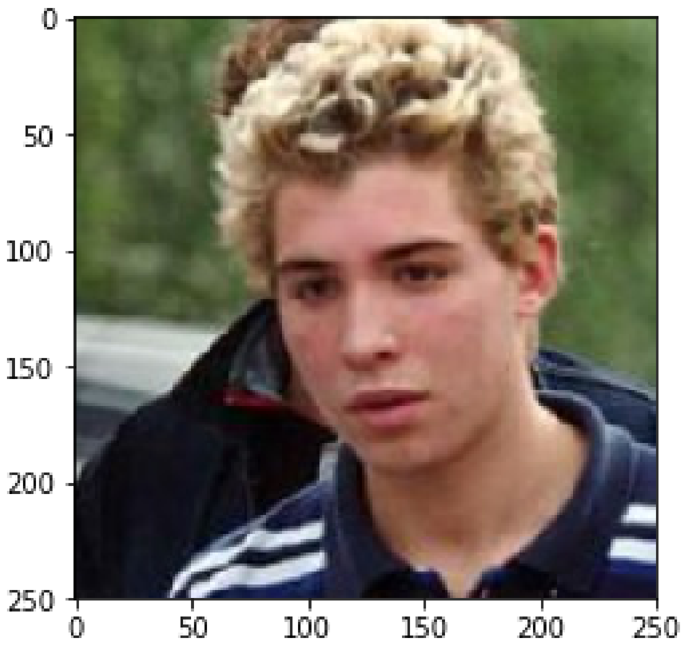
```
hello, human!
```



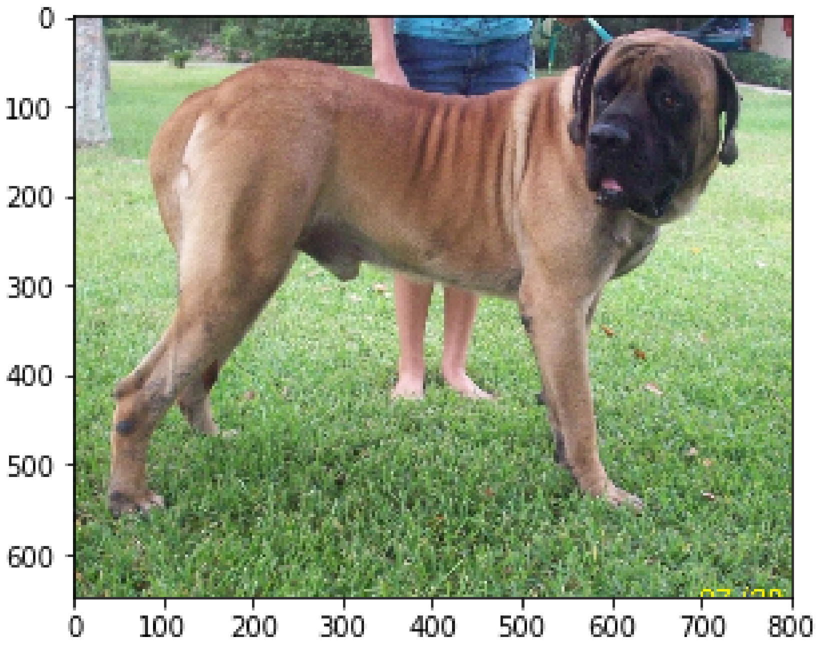
```
If you were a dog..You may look like a Labrador retriever
hello, human!
```



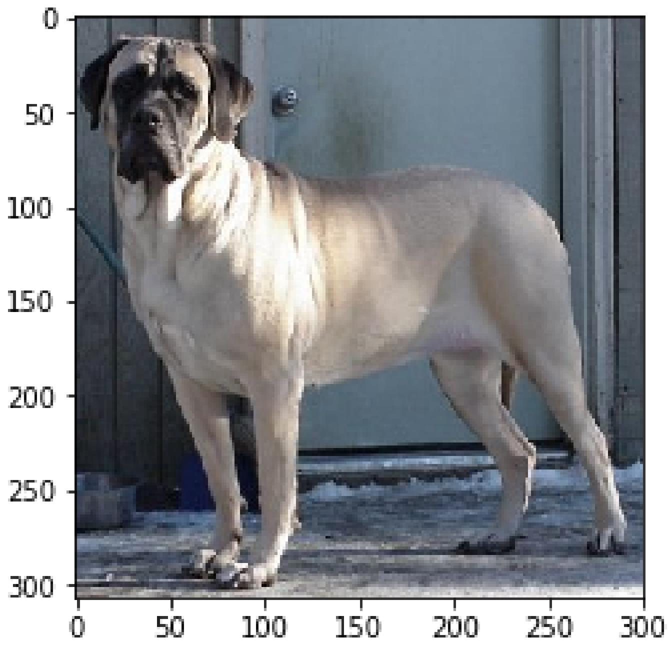
```
If you were a dog..You may look like a Pharaoh hound
hello, human!
```



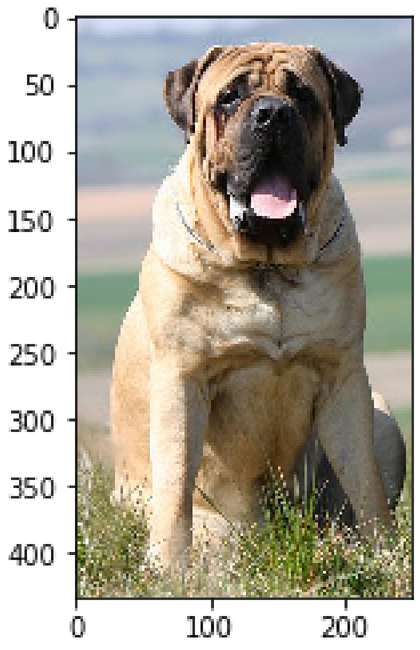
```
If you were a dog..You may look like a Irish water spaniel
Dogs Detected!
```



```
It looks like a Bullmastiff
Dogs Detected!
```



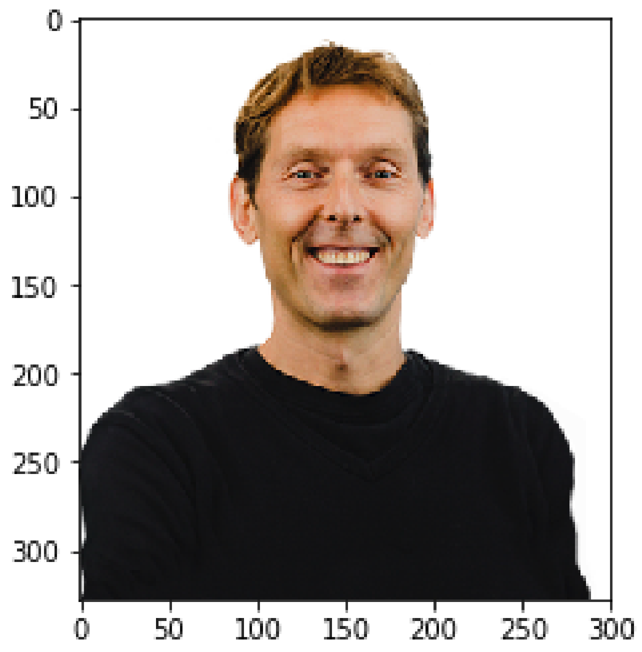
```
It looks like a Bullmastiff
Dogs Detected!
```



```
It looks like a Bullmastiff
```

```
In [22]: # custom uploads

         custom_human = ['./custom_images/custom_human1.jpg', './custom_images/custom_human2.jpg
         custom_dogs = ['./custom_images/custom_dog1.jpg', './custom_images/custom_dog2.jpg', '.
         for file in np.hstack((custom_human, custom_dogs)):
             run_app(file)
```

```
hello, human!
```

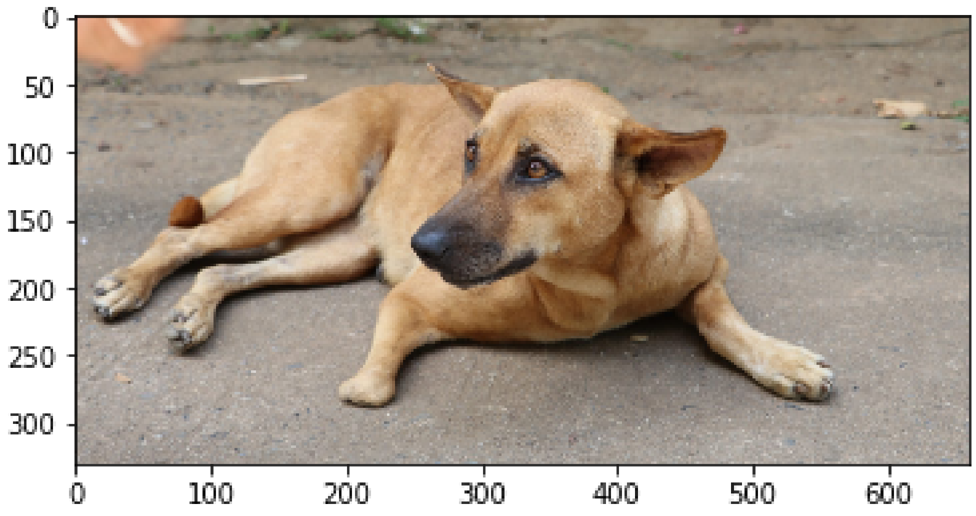


```
If you were a dog..You may look like a Afghan hound
hello, human!
```

```
If you were a dog..You may look like a Labrador retriever
hello, human!
```

```
If you were a dog..You may look like a Great dane
You are a alien! X..X
Dogs Detected!
```

```
It looks like a Afghan hound
Dogs Detected!
```



```
It looks like a Labrador retriever
```

```
In [ ]:
```