dog_app

September 2, 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.
```

```
In [2]: human_files[1]
```

```
Out[2]: '/data/lfw/Alex_Corretja/Alex_Corretja_0001.jpg'
```

Step 1: Detect Humans

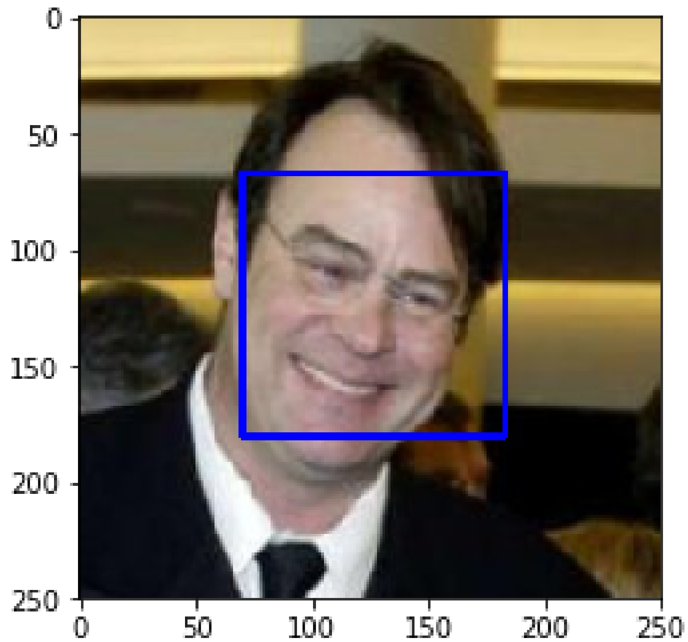
In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

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 [3]: 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 [4]: # 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 [5]: from tqdm import tqdm

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

        face_dec = np.vectorize(face_detector)
        #-#-# Do NOT modify the code above this line. #-#-#

        ## TODO: Test the performance of the face_detector algorithm
        ## on the images in human_files_short and dog_files_short.
        human_face_detected =face_dec(human_files_short)
        dog_detected = face_dec(dog_files_short)

        print(" Human Face Detected with {:.1f}% accuracy".format(sum(human_face_detected)))
        print("Dog detection is done with  {:.1f}% error".format(sum(dog_detected)))
```

```
 Human Face Detected with 98.0% accuracy
Dog detection is done with  17.0% error
```

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 [ ]: ### (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 [13]: 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.

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 [19]: from PIL import Image
         import torchvision.transforms as transforms
         from torch.autograd import Variable


         def transfrom(image_path):
             img =Image.open(image_path)



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

```
        Args:
            img_path: path to an image

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

        ## TODO: Complete the function.
        ## Load and pre-process an image from the given img_path
        ## Return the *index* of the predicted class for that image
        transform_img = transforms.Compose([
            transforms.Resize(size=(224,224)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

        ])
        img = Image.open(img_path)
        img = transform_img(img)

        # PyTorch pretrained models expect the Tensor dims to be (num input imgs, num color
        # Currently however, we have (num color channels, height, width); let's fix this by
        img = img.unsqueeze(0)  # Insert the new axis at index 0 i.e. in front of the other

        # Now that we have preprocessed our img, we need to convert it into a
        # Variable; PyTorch models expect inputs to be Variables. A PyTorch Variable is a
        # wrapper around a PyTorch Tensor.
        img = Variable(img)

        prediction = VGG16(img.cuda())  # Returns a Tensor of shape (batch, num class label
        prediction = prediction.cpu().data.numpy().argmax()  # Our prediction will be the i

        return prediction # predicted class index
```

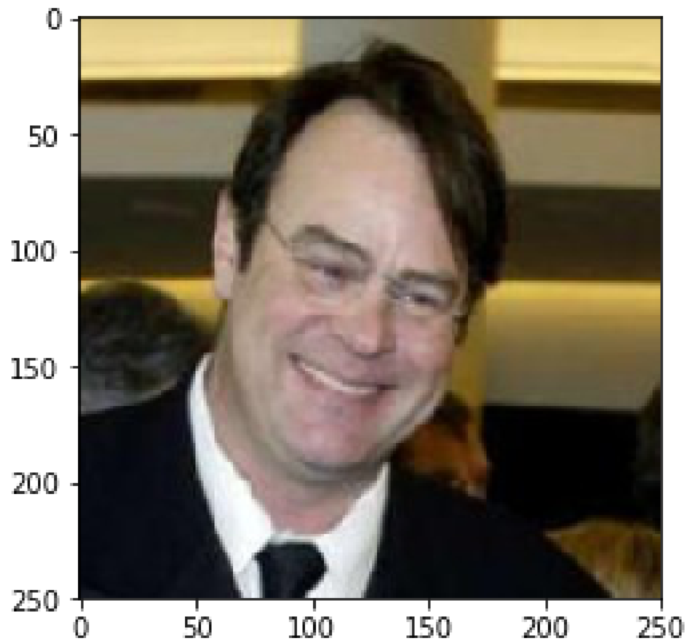
In [15]:
```
from PIL import Image
import glob
image = Image.open(human_files[0])
# summarize some details about the image
print(image.format)
print(image.mode)
print(image.size)
# show the image
image.show()
```

```
JPEG
RGB
(250, 250)
```

In [9]:
```
image.show()
```

In [16]:
```
# load and display an image with Matplotlib
from matplotlib import image
from matplotlib import pyplot
# load image as pixel array
data = image.imread(human_files[0])
# summarize shape of the pixel array
print(data.dtype)
print(data.shape)
# display the array of pixels as an image
pyplot.imshow(data)
pyplot.show()
```

```
uint8
(250, 250, 3)
```



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 [20]: ### returns "True" if a dog is detected in the image stored at img_path
         def dog_detector(img_path):
             ## TODO: Complete the function.
             pred=VGG16_predict(img_path)
             if pred >= 151 and pred <= 268:
                 return True
             else:
                 return False
```

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 [21]: from tqdm import tqdm
         ### TODO: Test the performance of the dog_detector function
         ### on the images in human_files_short and dog_files_short.
         human_files_short = human_files[:100]
         dog_files_short = dog_files[:100]

         counter_human = 0
         counter_dog   = 0

         for human_file in tqdm(human_files_short):
             if(dog_detector(human_file)):
                 counter_human +=1
         for dog_file in tqdm(dog_files_short):
             if(dog_detector(dog_file)):
                 counter_dog +=1
         print("Detected dogs in human files ",counter_human,"%")
         print("Detected dogs in dogs files ",counter_dog,"%")

100%|| 100/100 [00:03<00:00, 30.83it/s]
100%|| 100/100 [00:04<00:00, 25.87it/s]

Detected dogs in human files  1 %
Detected dogs in dogs files  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 [ ]: ### (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 [31]: import torch
         from torchvision import datasets
         import torchvision.transforms as transforms

         ### TODO: Write data loaders for training, validation, and test sets
         ## Specify appropriate transforms, and batch_sizes
         data_dir = "/data/dog_images/"
         num_workers = 0
         batch_size = 10
         data_transforms = {
             'train' : transforms.Compose([
             transforms.Resize(256),
             transforms.RandomResizedCrop(224),
             transforms.RandomHorizontalFlip(), # randomly flip and rotate
             transforms.RandomRotation(15),
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
             ]),
             # no need of image augmentation for the validation test set
             'valid' : transforms.Compose([
             transforms.Resize(256),
             transforms.CenterCrop(224),
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
             ]),
             # test dataset flips can  be found out in fast.ai
             'test' : transforms.Compose([
             transforms.Resize(256),
             transforms.CenterCrop(224),
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
             ]),
         }


         train_dir = data_dir + '/train'
         valid_dir = data_dir + '/valid'
         test_dir = data_dir + '/test'

         image_datasets = {
             'train' : datasets.ImageFolder(root=train_dir,transform=data_transforms['train']),
             'valid' : datasets.ImageFolder(root=valid_dir,transform=data_transforms['valid']),
             'test' : datasets.ImageFolder(root=test_dir,transform=data_transforms['test'])
         }

         # Loading Dataset
         loaders_scratch = {
             'train' : torch.utils.data.DataLoader(image_datasets['train'],batch_size = batch_si
```

```
            'valid' : torch.utils.data.DataLoader(image_datasets['valid'],batch_size = batch_si
            'test' : torch.utils.data.DataLoader(image_datasets['test'],batch_size = batch_size
        }
```

In [32]:
```
print(loaders_scratch['train'])
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7f9ba434f5f8>
```

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: The code resizes the image to an image of 256 x 256, then it center crops imag to a size of 224 X 224, the image is then normalized as this the requirement for f Yes I have used image augmentation I have randomly flipped the image horizontally while I have then randomly rotated the image by 15 degrees.

1.1.8 (IMPLEMENTATION) Model Architecture

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

In [33]:
```
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)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.conv3 = nn.Conv2d(32, 64, 3)
        self.conv4 = nn.Conv2d(64, 128, 3)
        self.conv5 = nn.Conv2d(128, 256, 3)

        self.fc1 = nn.Linear(256 * 6 * 6, 133)
        self.max_pool = nn.MaxPool2d(2, 2,ceil_mode=True)

        self.dropout = nn.Dropout(0.20)

        self.conv_bn1 = nn.BatchNorm2d(224,3)
        self.conv_bn2 = nn.BatchNorm2d(16)
        self.conv_bn3 = nn.BatchNorm2d(32)
        self.conv_bn4 = nn.BatchNorm2d(64)
        self.conv_bn5 = nn.BatchNorm2d(128)
        self.conv_bn6 = nn.BatchNorm2d(256)
```

```
    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.max_pool(x)
        x = self.conv_bn2(x)

        x = F.relu(self.conv2(x))
        x = self.max_pool(x)
        x = self.conv_bn3(x)

        x = F.relu(self.conv3(x))
        x = self.max_pool(x)
        x = self.conv_bn4(x)

        x = F.relu(self.conv4(x))
        x = self.max_pool(x)
        x = self.conv_bn5(x)

        x = F.relu(self.conv5(x))
        x = self.max_pool(x)
        x = self.conv_bn6(x)

        x = x.view(-1, 256 * 6 * 6)

        x = self.dropout(x)
        x = self.fc1(x)

        return x

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

# instantiate the CNN
model_scratch = Net()

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

In [34]:
```
model_scratch
```

Out[34]:
```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=9216, out_features=133, bias=True)
  (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
```

```
    (dropout): Dropout(p=0.2)
    (conv_bn1): BatchNorm2d(224, eps=3, momentum=0.1, affine=True, track_running_stats=Tr
    (conv_bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (conv_bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (conv_bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (conv_bn5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
    (conv_bn6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stat
  )
```

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

Answer:

The model architecture is postulated below: 1. I have used 5 Convolutional layers with the colvolution of kernel size = 3, stride = 1 and padding = 0, where the input channel size is 3 and output is 16 and each layer , the layer can be defined as follows layer 1 : (3,16) in_channel=3, out_channel=16 for 2 to 5 convolutional layers layer n :(out_channl{n-1}, out_channel{n-1}*2), in_channel=out_channl{n-1}, out_channel=out_channl{n-1}*2

2. I have used Relu activation function to prevent from the vanishing gradient problem.
3. I have introducd invariance using Max pooling (2 X 2) which reduces the size of the size of the input matrices reduced to half.
4. Batch Normalization 2D is a technique to provide inputs that are zero mean or variance 1.
5. I have implemented Dropout(0.2) to handle overfitting.
6. At the end I have included the classifier with the output 133, that is the number of breeds of the dogs that are to be classified.

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 [35]: import torch.optim as optim

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

         ### TODO: select optimizer
         optimizer_scratch = optim.SGD(model_scratch.parameters(), lr=0.005)
```

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 [36]: import numpy as np
         from PIL import ImageFile
         ImageFile.LOAD_TRUNCATED_IMAGES = True

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

```python
"""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()
        # forward pass
        output = model(data)
        # Loss
        loss = criterion(output, target)
        # backward pass
        loss.backward()
        # Optimization
        optimizer.step()
        # update training loss
        # train_loss += loss.item()*data.size(0)
        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()

        output = model(data)
        loss = criterion(output, target)
        # update average validation loss
        valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss)
```

```
                # calculate average losses
            train_loss = train_loss/len(loaders['train'].dataset)
            valid_loss = valid_loss/len(loaders['valid'].dataset)



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

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

        # return trained model
        return model


    # 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: 0.000703         Validation Loss: 0.005346
Validation loss decreased (inf --> 0.005346).  Saving model ...
Epoch: 2         Training Loss: 0.000669         Validation Loss: 0.005083
Validation loss decreased (0.005346 --> 0.005083).  Saving model ...
Epoch: 3         Training Loss: 0.000652         Validation Loss: 0.004939
Validation loss decreased (0.005083 --> 0.004939).  Saving model ...
Epoch: 4         Training Loss: 0.000638         Validation Loss: 0.005062
Epoch: 5         Training Loss: 0.000625         Validation Loss: 0.004834
Validation loss decreased (0.004939 --> 0.004834).  Saving model ...
Epoch: 6         Training Loss: 0.000614         Validation Loss: 0.004707
Validation loss decreased (0.004834 --> 0.004707).  Saving model ...
Epoch: 7         Training Loss: 0.000596         Validation Loss: 0.004621
Validation loss decreased (0.004707 --> 0.004621).  Saving model ...
Epoch: 8         Training Loss: 0.000587         Validation Loss: 0.004491
Validation loss decreased (0.004621 --> 0.004491).  Saving model ...
```

```
Epoch: 9        Training Loss: 0.000579        Validation Loss: 0.004621
Epoch: 10        Training Loss: 0.000567        Validation Loss: 0.004341
Validation loss decreased (0.004491 --> 0.004341).  Saving model ...
Epoch: 11        Training Loss: 0.000560        Validation Loss: 0.004234
Validation loss decreased (0.004341 --> 0.004234).  Saving model ...
Epoch: 12        Training Loss: 0.000550        Validation Loss: 0.004316
Epoch: 13        Training Loss: 0.000545        Validation Loss: 0.004362
Epoch: 14        Training Loss: 0.000536        Validation Loss: 0.004092
Validation loss decreased (0.004234 --> 0.004092).  Saving model ...
Epoch: 15        Training Loss: 0.000525        Validation Loss: 0.004052
Validation loss decreased (0.004092 --> 0.004052).  Saving model ...
Epoch: 16        Training Loss: 0.000525        Validation Loss: 0.003984
Validation loss decreased (0.004052 --> 0.003984).  Saving model ...
Epoch: 17        Training Loss: 0.000516        Validation Loss: 0.003974
Validation loss decreased (0.003984 --> 0.003974).  Saving model ...
Epoch: 18        Training Loss: 0.000511        Validation Loss: 0.003887
Validation loss decreased (0.003974 --> 0.003887).  Saving model ...
Epoch: 19        Training Loss: 0.000505        Validation Loss: 0.003971
Epoch: 20        Training Loss: 0.000501        Validation Loss: 0.003881
Validation loss decreased (0.003887 --> 0.003881).  Saving model ...
```

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 [37]: def test(loaders, model, criterion, use_cuda):

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

             model.eval()
             for batch_idx, (data, target) in enumerate(loaders['test']):
                 # move to GPU
                 if use_cuda:
                     data, target = data.cuda(), target.cuda()
                 # forward pass: compute predicted outputs by passing inputs to the model
                 output = model(data)
                 # calculate the loss
                 loss = criterion(output, target)
                 # update average test loss
                 test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))
                 # convert output probabilities to predicted class
                 pred = output.data.max(1, keepdim=True)[1]
                 # compare predictions to true label
```

```
                correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().numpy())
                total += data.size(0)

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

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

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

```
Test Loss: 3.121946


Test Accuracy: 24% (201/836)
```

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).

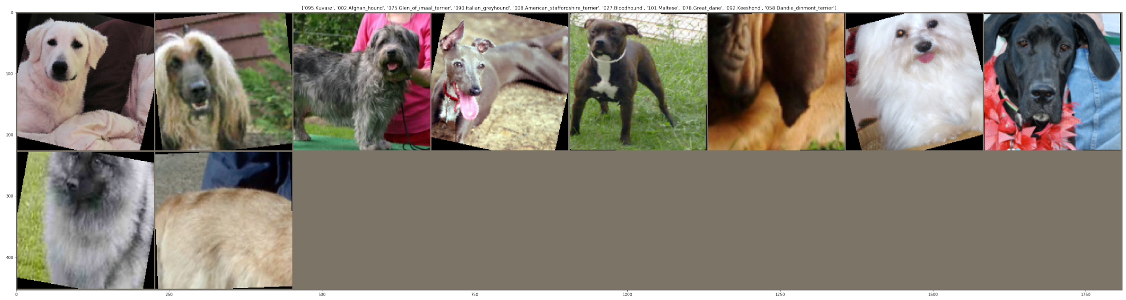
If you like, **you are welcome to use the same data loaders from the previous step**, when you created a CNN from scratch.

```
In [40]: from torchvision import datasets
         import torchvision.transforms as transforms

         data_dir = "/data/dog_images/"
         num_workers = 0
         batch_size = 10
         data_transforms = {
             'train' : transforms.Compose([
             transforms.Resize(256),
             transforms.RandomResizedCrop(224),
             transforms.RandomHorizontalFlip(), # randomly flip and rotate
             transforms.RandomRotation(15),
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
             ]),
             # no need of image augmentation for the validation test set
             'valid' : transforms.Compose([
             transforms.Resize(256),
```

```
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ]),
        # test dataset flips can  be found out in fast.ai
        'test' : transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ]),
    }


    train_dir = data_dir + '/train'
    valid_dir = data_dir + '/valid'
    test_dir = data_dir + '/test'

    image_datasets = {
        'train' : datasets.ImageFolder(root=train_dir,transform=data_transforms['train']),
        'valid' : datasets.ImageFolder(root=valid_dir,transform=data_transforms['valid']),
        'test' : datasets.ImageFolder(root=test_dir,transform=data_transforms['test'])
    }
    class_names = image_datasets['train'].classes
    # Loading Dataset
    loaders_transfer= {
        'train' : torch.utils.data.DataLoader(image_datasets['train'],batch_size = batch_si
        'valid' : torch.utils.data.DataLoader(image_datasets['valid'],batch_size = batch_si
        'test' : torch.utils.data.DataLoader(image_datasets['test'],batch_size = batch_size
    }
```

In [41]:
```
import torchvision.utils as torchutil
import matplotlib.pyplot as plt
plt.ion()
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    fig = plt.figure(figsize=(50, 25))
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated
```

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

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

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



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 [42]: import torchvision.models as models
         import torch.nn as nn

         ## TODO: Specify model architecture
         model_transfer = models.vgg16(pretrained=True)
         # Freeze the pre-trained weights
         for param in model_transfer.features.parameters():
             param.required_grad = False

         # Get the input of the last layer of VGG-16
         n_inputs = model_transfer.classifier[6].in_features

         # Create a new layer(n_inputs -> 133)
         # The new layer's requires_grad will be automatically True.
         last_layer = nn.Linear(n_inputs, 133)

         # Change the last layer to the new layer.
         model_transfer.classifier[6] = last_layer

         # Print the model.
         print(model_transfer)
```

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

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=4096, out_features=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 have used VGG16 model to run my transfer learning, I have updated the final layer of the classifier in the architecture to make it suitable with my problem statement of classifying dog breeds, as I know there are 133 classes for the datasets.

Since VGG16 is pretrained model I think it is suitable for the classification algorithm. Since there is a large dataset overfitting will not be much of an issue.

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 [43]: criterion_transfer = nn.CrossEntropyLoss()
         optimizer_transfer = optim.SGD(model_transfer.classifier.parameters(),lr=0.001,momentum
```

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 [45]: import numpy as np
         from PIL import ImageFile
         ImageFile.LOAD_TRUNCATED_IMAGES = True
         # train the model
         n_epochs = 10


         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()
        # forward pass
        output = model(data)
        # Loss
        loss = criterion(output, target)
        # backward pass
        loss.backward()
        # Optimization
        optimizer.step()
        # update training loss
        # train_loss += loss.item()*data.size(0)
        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()

        output = model(data)
        loss = criterion(output, target)
        # update average validation loss
        valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss)


        # calculate average losses
    train_loss = train_loss/len(loaders['train'].dataset)
    valid_loss = valid_loss/len(loaders['valid'].dataset)



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

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

```
                    valid_loss_min,
                    valid_loss))
                    torch.save(model.state_dict(), save_path)
                    valid_loss_min = valid_loss

        # return trained model
        return model


    model_transfer = train(n_epochs, loaders_transfer, model_transfer, optimizer_transfer,

    # 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: 0.000244        Validation Loss: 0.000850
Validation loss decreased (inf --> 0.000850).  Saving model ...
Epoch: 2         Training Loss: 0.000230        Validation Loss: 0.000849
Validation loss decreased (0.000850 --> 0.000849).  Saving model ...
Epoch: 3         Training Loss: 0.000220        Validation Loss: 0.000719
Validation loss decreased (0.000849 --> 0.000719).  Saving model ...
Epoch: 4         Training Loss: 0.000213        Validation Loss: 0.000735
Epoch: 5         Training Loss: 0.000210        Validation Loss: 0.000747
Epoch: 6         Training Loss: 0.000199        Validation Loss: 0.000793
Epoch: 7         Training Loss: 0.000198        Validation Loss: 0.000715
Validation loss decreased (0.000719 --> 0.000715).  Saving model ...
Epoch: 8         Training Loss: 0.000195        Validation Loss: 0.000666
Validation loss decreased (0.000715 --> 0.000666).  Saving model ...
Epoch: 9         Training Loss: 0.000191        Validation Loss: 0.000642
Validation loss decreased (0.000666 --> 0.000642).  Saving model ...
Epoch: 10         Training Loss: 0.000195        Validation Loss: 0.000721
```

1.1.16 (IMPLEMENTATION) Test the Model

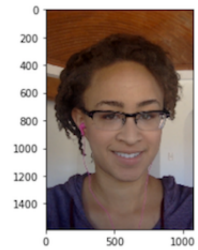
Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 60%.

```
In [46]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.645538


Test Accuracy: 81% (683/836)
```

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.



Sample Human Output

```
In [47]: ### 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 image_datasets['train'].classes]
         # Load the trained model 'model_transfer.pt'
         model_transfer.load_state_dict(torch.load('model_transfer.pt', map_location='cpu'))
         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(mean=[0.485, 0.456, 0.406], s

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


             model_transfer.eval()
             idx = torch.argmax(model_transfer(image))
             return class_names[idx]
```

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 [48]: ### TODO: Write your algorithm.
         ### Feel free to use as many code cells as needed.
         from PIL import Image
         def run_app(img_path):
             ## handle cases for a human face, dog, and neither
             img = Image.open(img_path)
             plt.imshow(img)
             plt.show()

             if dog_detector(img_path) is True:
                 prediction = predict_breed_transfer(img_path)
                 print("A dog has been detected which most likely to be {0} breed".format(predic
             elif face_detector(img_path) > 0:
                 prediction = predict_breed_transfer(img_path)
                 print("This is a Human who looks like a {0}".format(prediction))
             else:
                 print("Neither Human nor Dog")
```

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!

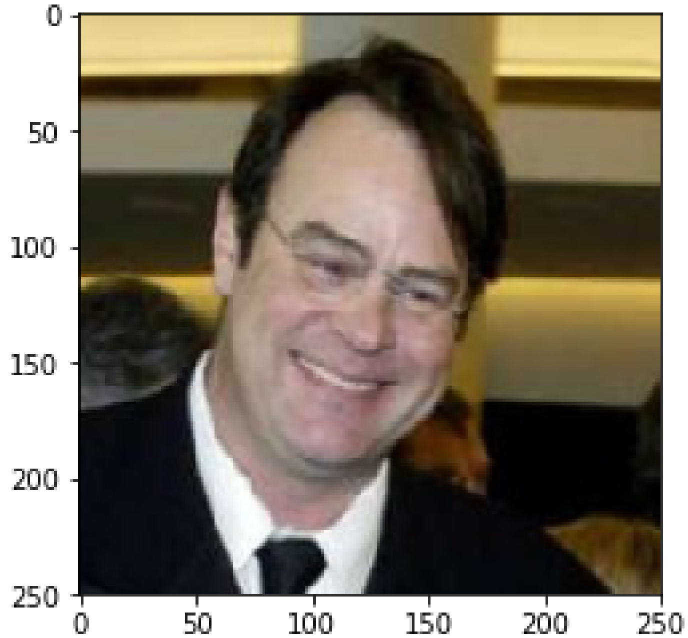
Test your algorithm at least six images on your computer. Feel free to use any images you like. Use at least two human and two dog images.

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

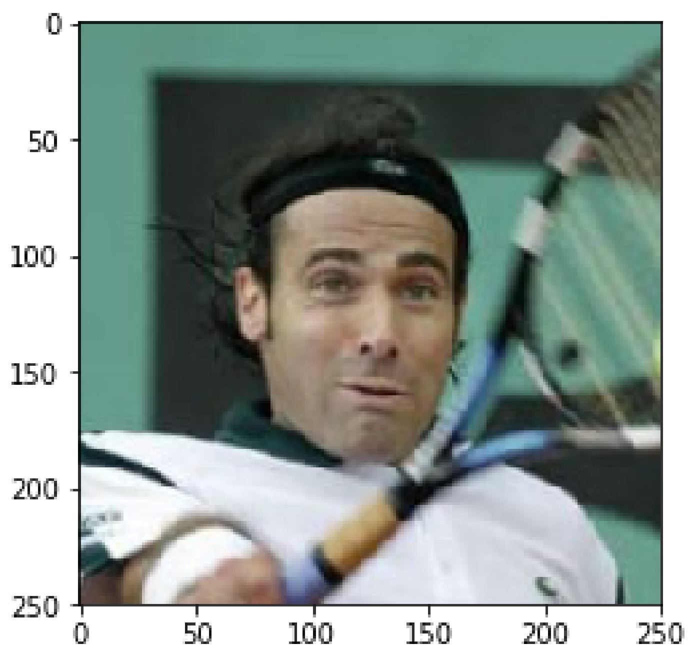
Answer: (Three possible points for improvement)

1. Hyperparameter tuning can help with the improvement of the model.
2. Modular coding can really decrease the redundancy inb the code.
3. We can compare a number of models to find the best model with better accuracy.

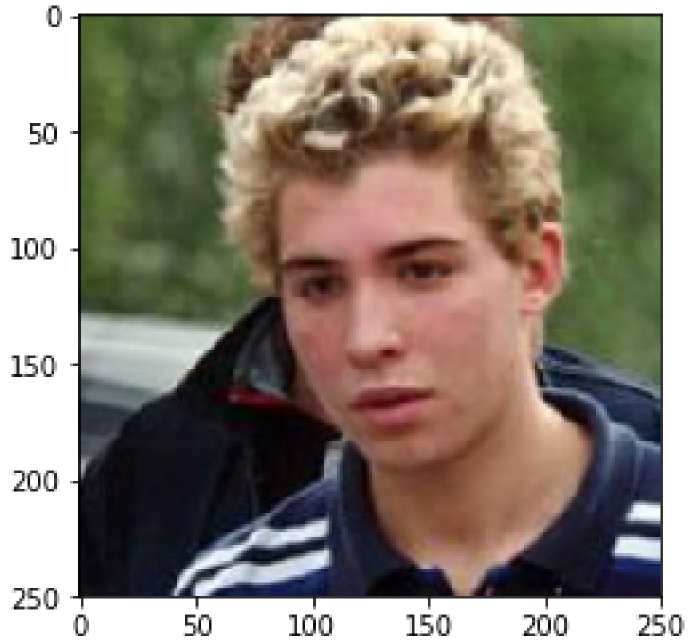
```
In [49]: ## TODO: Execute your algorithm from Step 6 on
         ## at least 6 images on your computer.
         ## Feel free to use as many code cells as needed.

         ## suggested code, below
         for file in np.hstack((human_files[:6], dog_files[:6])):
             run_app(file)
```



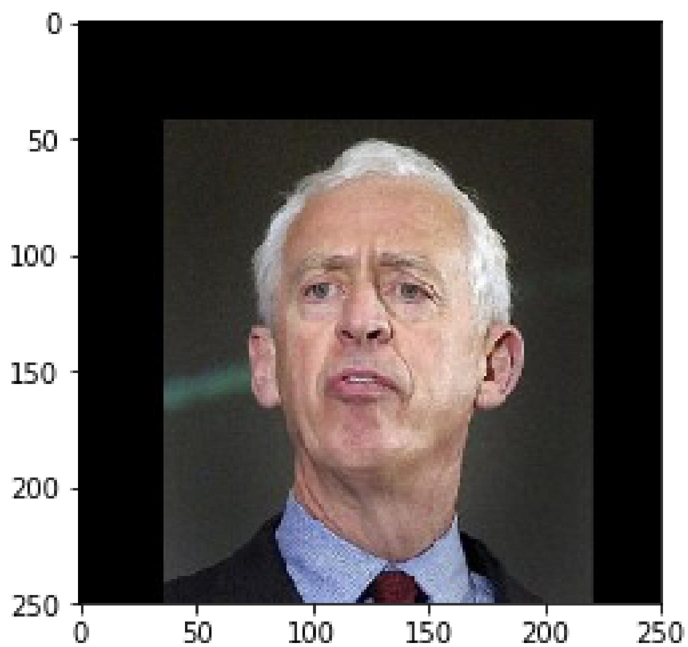
This is a Human who looks like a Beagle



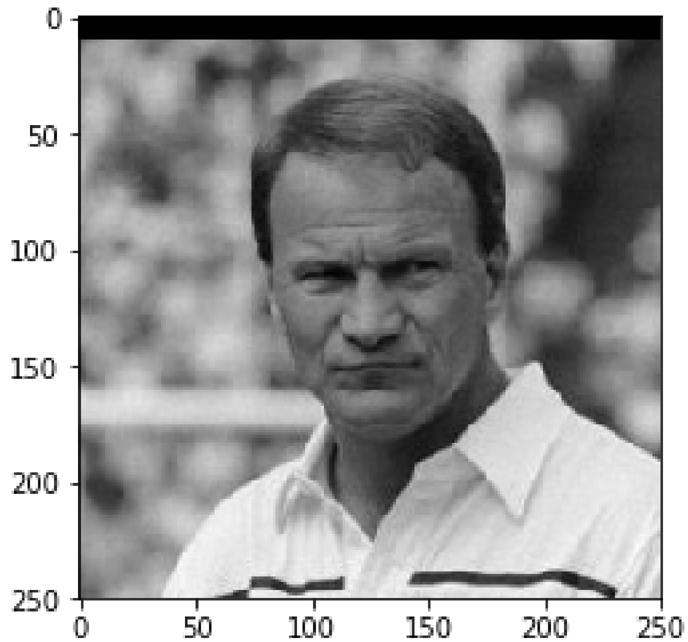
```
This is a Human who looks like a Dachshund
```



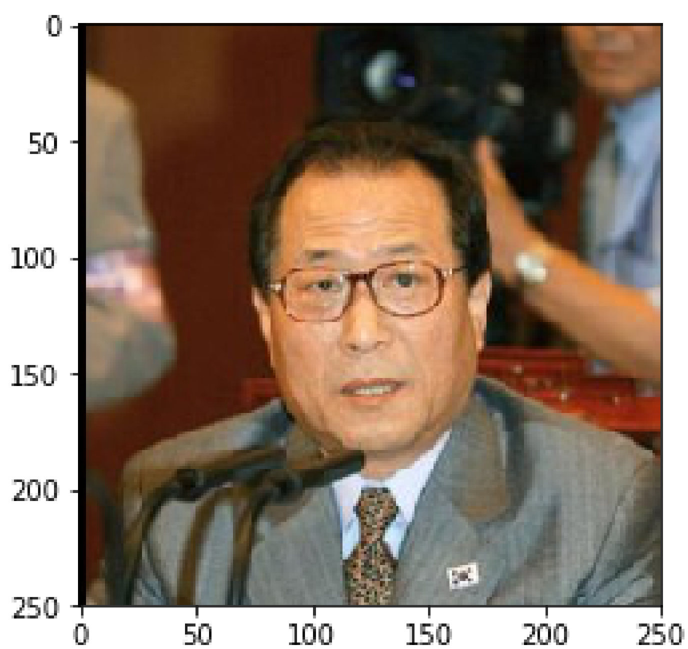
```
This is a Human who looks like a Cocker spaniel
```



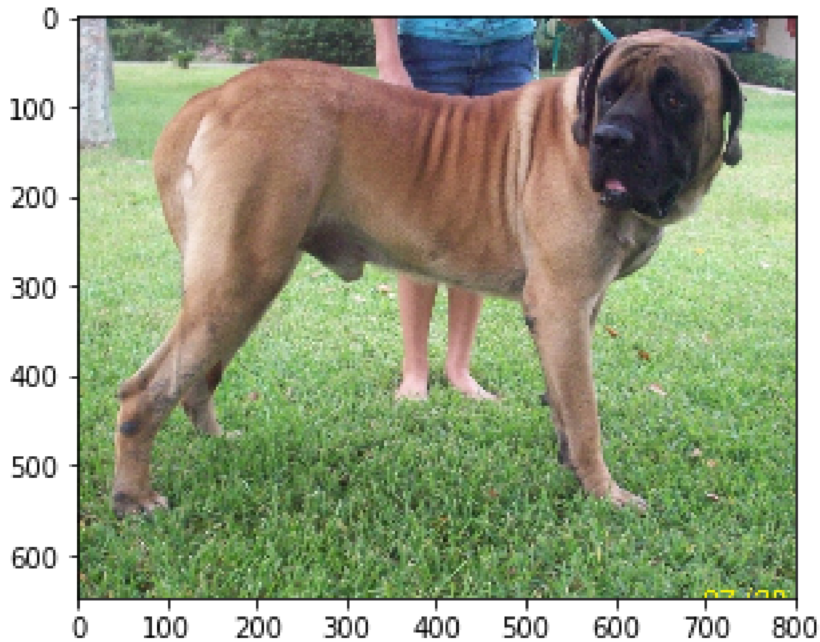
This is a Human who looks like a Brittany



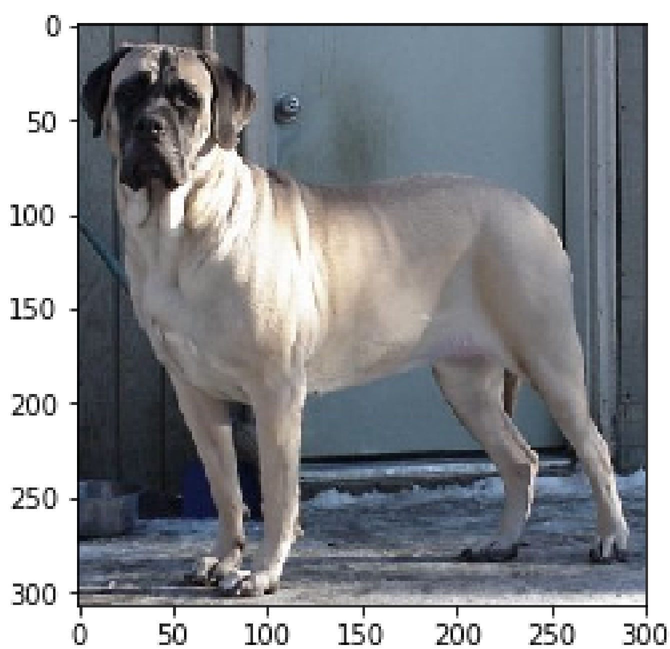
This is a Human who looks like a Welsh springer spaniel



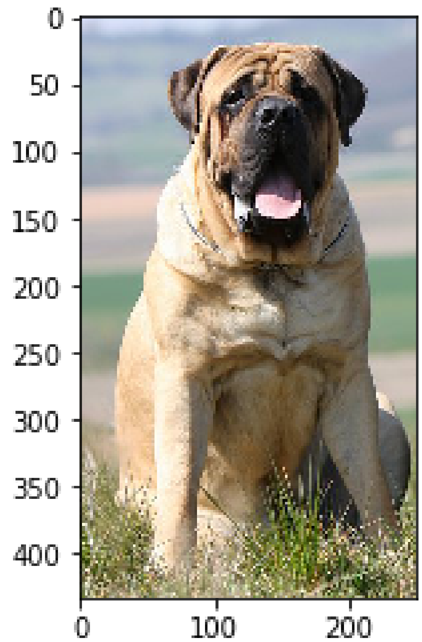
This is a Human who looks like a Welsh springer spaniel



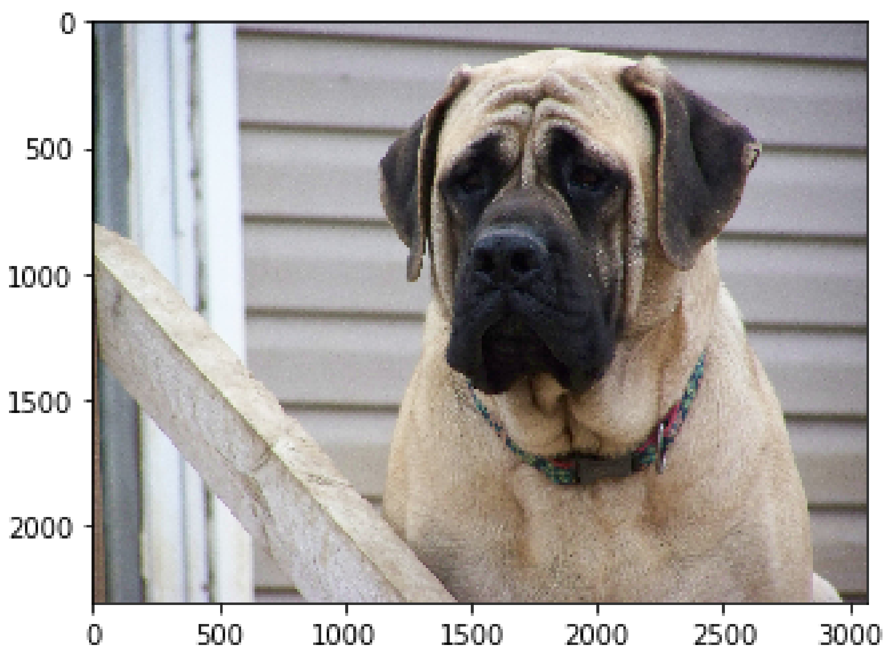
A dog has been detected which most likely to be Bullmastiff breed



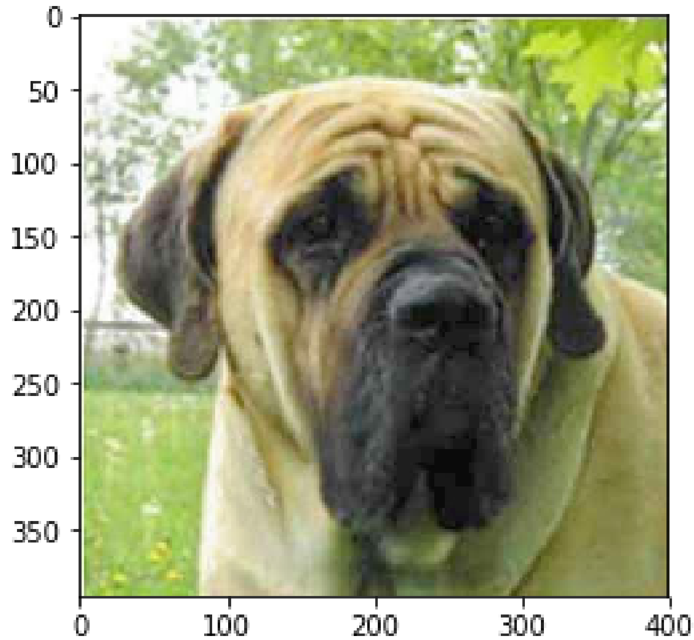
```
A dog has been detected which most likely to be Mastiff breed
```




```
A dog has been detected which most likely to be Bullmastiff breed
```



A dog has been detected which most likely to be Mastiff breed



A dog has been detected which most likely to be Mastiff breed

```
A dog has been detected which most likely to be Mastiff breed
```