dog_app

April 5, 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

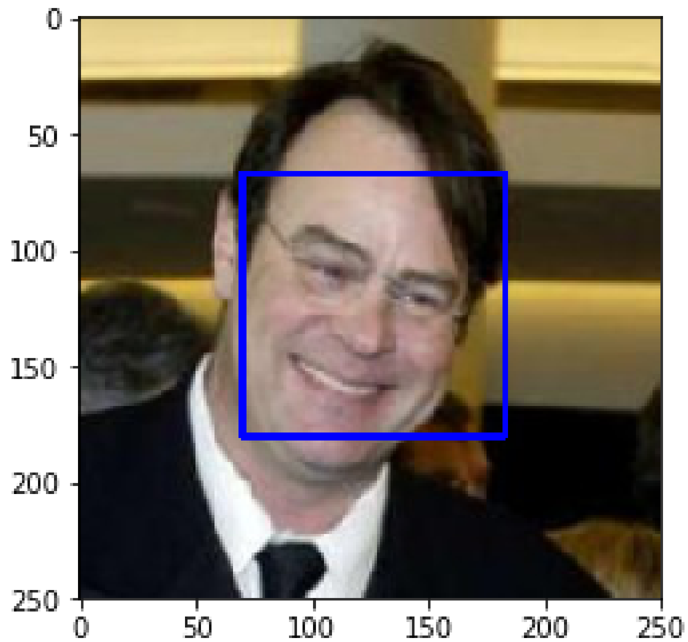
In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

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.
        human_faces_in_human = [face_detector(i) for i in tqdm((human_files_short),desc='Human i
        human_faces_in_dogs = [face_detector(i) for i in tqdm((dog_files_short),desc='Dog images
```

```
Human images: 100%|| 100/100 [00:02<00:00, 35.78it/s]
Dog images: 100%|| 100/100 [00:30<00:00,  3.30it/s]
```

```
In [5]: # Answer
        print(f'human faces percentage in the first 100 human images: {sum(human_faces_in_human)
        print(f'human faces percentage in the first 100 dogs images: {sum(human_faces_in_dogs)/l
```

```
human faces percentage in the first 100 human images: 98.0 %
human faces percentage in the first 100 dogs images: 17.0 %
```

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 [6]: ### (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 [7]: 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()
```

```
In [8]: use_cuda
```

```
Out[8]: True
```

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

1.1.4 (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 [9]: 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
    # open the image
    img = Image.open(img_path).convert('RGB')
    # set transofrm and turn to tensor
    transform = transforms.Compose([transforms.Resize(size=(224,224)),
                                    transforms.ToTensor()])
    #transform(img)
    img = transform(img).unsqueeze(0)
    if use_cuda:
        img = img.cuda()
    output = VGG16(img )
    # get the top predicted class
    class_index = torch.max(output,1)[1].item()

    return class_index #class_index # predicted class
```

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 [10]: ### returns "True" if a dog is detected in the image stored at img_path
         def dog_detector(img_path):
             ## TODO: Complete the function.
             class_index = VGG16_predict(img_path)
             return 151 < class_index < 268 # return True/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 [11]:
```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.

dog_in_human = [dog_detector(i) for i in tqdm((human_files_short),desc='Dog in Human pi
dog_in_dog = [dog_detector(i) for i in tqdm((dog_files_short),desc= 'Dog in dogs pics')
```

```
Dog in Human pics: 100%|| 100/100 [00:03<00:00, 32.32it/s]
Dog in dogs pics: 100%|| 100/100 [00:04<00:00, 21.23it/s]
```

In [12]:
```
# Answer
print(f'percentage of the images in human_files_short have a detected dog is {sum(dog_i
print(f'percentage of the images in dog_files_short have a detected dog is {sum(dog_in_
```

```
percentage of the images in human_files_short have a detected dog is 0.0%
percentage of the images in dog_files_short have a detected dog is 91.0%
```

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 [14]: # load librarires
         from PIL import Image
         import torchvision.transforms as transforms
         import torch
         import torchvision.models as models

         use_cuda = torch.cuda.is_available()
```

```
In [15]: import os
         from torchvision import datasets

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

         # specify the batch size
         batch_size = 64

         # define transforms
         normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                          std=[0.229, 0.224, 0.225])

         train_transform = transforms.Compose([transforms.RandomRotation(10),
                                               transforms.RandomResizedCrop(224),
                                               transforms.RandomHorizontalFlip(),
                                               transforms.ToTensor(),
                                               normalize
                                              ])
```

```
test_transform = transforms.Compose([transforms.Resize(256),
                                     transforms.CenterCrop(224),
                                     transforms.ToTensor(),
                                     normalize
                                    ])

show_transform = transforms.Compose([transforms.Resize(256),
                                     transforms.CenterCrop(224),
                                     transforms.ToTensor()])

# chose the data sets
train_data = datasets.ImageFolder('dogImages/train/', transform=train_transform, )
test_data = datasets.ImageFolder('dogImages/test/', transform=test_transform)
valid_data = datasets.ImageFolder('dogImages/valid/', transform=test_transform)
show_data = datasets.ImageFolder('dogImages/valid/', transform=show_transform)
```

In [16]:
```
# write the loaders
from torch.utils.data import DataLoader

train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size)
valid_loader = DataLoader(valid_data, batch_size=batch_size)
show_loader = DataLoader(show_data, batch_size=10)
```

In [17]:
```
# have a look at some images
dataiter = iter(show_loader)
images, labels = dataiter.next()
images = images.numpy()
```

In [18]:
```
def imshow(img):
    ' to show image'
    # change order of dimensions
    plt.imshow(np.transpose(img, (1, 2, 0)))
```

In [19]:
```
# loop over all images in the first batch
fig = plt.figure(figsize=(18, 6))

n_img = 10
for idx in np.arange(n_img):
    ax = fig.add_subplot(2, n_img/2, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    ax.set_title(labels[idx].item())
```

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 resized the images to be 224 x 224 pixels by RandomResizedCrop, because this is size expected by VGG16 * I augment the dataset by random rotation with 10 degrees, and random horizontal flipping

1.1.8 (IMPLEMENTATION) Model Architecture

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

```
In [20]: import torch.nn as nn
         import torch.nn.functional as F

         # here, we define the architecture of the model

         class Net(nn.Module):
             ### TODO: choose an architecture, and complete the class
             def __init__(self):
                 super(Net, self).__init__()

                 # feature extractor
                 # we use an architecture similar to VGG16 here
                 # just not as deep
                 self.Conv1_1 = nn.Conv2d(3,32,3,1,1)
                 self.B_norm1_1 = nn.BatchNorm2d(32)

                 self.Conv2_1 = nn.Conv2d(32,64,3,1,1)
                 self.B_norm2_1 = nn.BatchNorm2d(64)
                 self.Conv2_2 = nn.Conv2d(64,64,3,1,1)
                 self.B_norm2_2 = nn.BatchNorm2d(64)

                 self.Conv3_1 = nn.Conv2d(64,128,3,1,1)
```

```python
        self.B_norm3_1 = nn.BatchNorm2d(128)
        self.Conv3_2 = nn.Conv2d(128,128,3,1,1)
        self.B_norm3_2 = nn.BatchNorm2d(128)

        self.Conv4_1 = nn.Conv2d(128,256,3,1,1)
        self.B_norm4_1 = nn.BatchNorm2d(256)
        self.Conv4_2 = nn.Conv2d(256,256,3,1,1)
        self.B_norm4_2 = nn.BatchNorm2d(256)

        # classifier
        self.fc1 = nn.Linear(256*14*14,1024)
        self.B_norm5 = nn.BatchNorm1d(1024)

        self.fc2 = nn.Linear(1024,133)

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

    def forward(self, x):
        ## Define forward behavior
        # again, first the feature extractor
        x = self.Conv1_1(x)
        x = F.relu(self.B_norm1_1(x))
        x = self.pool(x)

        x = self.Conv2_1(x)
        x = F.relu(self.B_norm2_1(x))
        x = self.Conv2_2(x)
        x = F.relu(self.B_norm2_2(x))
        x = self.pool(x)

        x = self.Conv3_1(x)
        x = F.relu(self.B_norm3_1(x))
        x = self.Conv3_2(x)
        x = F.relu(self.B_norm3_2(x))
        x = self.pool(x)

        x = self.Conv4_1(x)
        x = F.relu(self.B_norm4_1(x))
        x = self.Conv4_2(x)
        x = F.relu(self.B_norm4_2(x))
        x = self.pool(x)

        # reshape and feed into the classifier
        x = x.view(-1,256*14*14)

        x = self.fc1(x)
        x = F.relu(self.B_norm5(x))
```

```
        x = F.dropout(x)

        x = F.log_softmax(self.fc2(x), dim=0)

        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
use_cuda = torch.cuda.is_available()
if use_cuda:
    model_scratch.cuda()
```

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

Answer:

- I used an architecture similar to what we have seen in the course before. It consists of a featue extractor part and a classifier. In the features part, there are several convolutional layers, follower by max pooling layers and relu layers. They are grouped together in blocks, as in the VGG16 network. The purpose of this stack is to identify descriptive features in the input data. The second part of the network consists of the classifier. In the classifier, there are linear layers, followed by relu layers and dropout layers. This stack of layers in meant to use the features from the previous stack to classify the given image.

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 [21]: import torch.optim as optim

         ### TODO: select loss function
         criterion_scratch = nn.CrossEntropyLoss()
         ### TODO: select optimizer
         optimizer_scratch = optim.Adam(params=model_scratch.parameters(), lr=0.001)
```

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 [22]: # define a dict with the data loaders
         loaders_scratch = {'train': train_loader,
                            'test': test_loader,
                            'valid': valid_loader}
```

In [23]:
```
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
import sys
```

In [24]:
```
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()
        print('Start training ...')
        for batch_idx, (data, target) in enumerate(loaders['train']):
            sys.stdout.write('Batch idx: {}/{}\r'.format(batch_idx, len(loaders['train'
            sys.stdout.flush()
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            ## find the loss and update the model parameters accordingly

            # zero the gradient
            optimizer.zero_grad()

            # forward pass through the network
            scores = model.forward(data)

            # compute the loss
            loss = criterion(scores, target)

            # perform backward pass
            loss.backward()

            # update the parameters (gradient step)
            optimizer.step()

            ## record the average training loss, using something like
            train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss)

        ######################
        # validate the model #
```

```
            ######################
            model.eval()
            print('---validating---')
            for batch_idx, (data, target) in enumerate(loaders['valid']):
                sys.stdout.write('Batch idx: {}/{}\r'.format(batch_idx, len(loaders['valid'
                sys.stdout.flush()
                # move to GPU
                if use_cuda:
                    data, target = data.cuda(), target.cuda()
                ## update the average validation loss
                scores = model.forward(data)
                loss = criterion(scores, 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:
                print('Validation loss decreased ({:.6f} --> {:.6f}). Saving model...'.form
                valid_loss_min, valid_loss))
                torch.save(model.state_dict(), save_path)
                valid_loss_min = valid_loss


        # return trained model
        return model
```

In [25]:
```
# train the model
model_scratch = train(30, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')
```

```
Start training ...
---validating---05
Epoch: 1 	Training Loss: 4.620306 	Validation Loss: 4.468390
Validation loss decreased (inf --> 4.468390). Saving model...
Start training ...
---validating---05
Epoch: 2 	Training Loss: 4.313269 	Validation Loss: 4.324039
Validation loss decreased (4.468390 --> 4.324039). Saving model...
Start training ...
---validating---05
Epoch: 3 	Training Loss: 4.114117 	Validation Loss: 4.282353
```

```
Validation loss decreased (4.324039 --> 4.282353). Saving model...
Start training ...
---validating---05
Epoch: 4        Training Loss: 4.016421         Validation Loss: 4.308033
Start training ...
---validating---05
Epoch: 5        Training Loss: 3.918866         Validation Loss: 4.230394
Validation loss decreased (4.282353 --> 4.230394). Saving model...
Start training ...
---validating---05
Epoch: 6        Training Loss: 3.802915         Validation Loss: 4.270567
Start training ...
---validating---05
Epoch: 7        Training Loss: 3.706261         Validation Loss: 4.080474
Validation loss decreased (4.230394 --> 4.080474). Saving model...
Start training ...
---validating---05
Epoch: 8        Training Loss: 3.616390         Validation Loss: 4.036614
Validation loss decreased (4.080474 --> 4.036614). Saving model...
Start training ...
---validating---05
Epoch: 9        Training Loss: 3.570717         Validation Loss: 4.003342
Validation loss decreased (4.036614 --> 4.003342). Saving model...
Start training ...
---validating---05
Epoch: 10        Training Loss: 3.467743         Validation Loss: 4.056796
Start training ...
---validating---05
Epoch: 11        Training Loss: 3.410034         Validation Loss: 4.013372
Start training ...
---validating---05
Epoch: 12        Training Loss: 3.341225         Validation Loss: 3.943544
Validation loss decreased (4.003342 --> 3.943544). Saving model...
Start training ...
---validating---05
Epoch: 13        Training Loss: 3.244808         Validation Loss: 4.058672
Start training ...
---validating---05
Epoch: 14        Training Loss: 3.198074         Validation Loss: 3.902573
Validation loss decreased (3.943544 --> 3.902573). Saving model...
Start training ...
---validating---05
Epoch: 15        Training Loss: 3.125545         Validation Loss: 3.933439
Start training ...
---validating---05
Epoch: 16        Training Loss: 3.028384         Validation Loss: 3.798406
Validation loss decreased (3.902573 --> 3.798406). Saving model...
Start training ...
```

```
---validating---05
Epoch: 17        Training Loss: 2.974738        Validation Loss: 3.963931
Start training ...
---validating---05
Epoch: 18        Training Loss: 2.915760        Validation Loss: 3.849880
Start training ...
---validating---05
Epoch: 19        Training Loss: 2.826394        Validation Loss: 3.791646
Validation loss decreased (3.798406 --> 3.791646). Saving model...
Start training ...
---validating---05
Epoch: 20        Training Loss: 2.786355        Validation Loss: 3.942865
Start training ...
---validating---05
Epoch: 21        Training Loss: 2.728284        Validation Loss: 3.731860
Validation loss decreased (3.791646 --> 3.731860). Saving model...
Start training ...
---validating---05
Epoch: 22        Training Loss: 2.649757        Validation Loss: 3.865325
Start training ...
---validating---05
Epoch: 23        Training Loss: 2.612802        Validation Loss: 3.812565
Start training ...
---validating---05
Epoch: 24        Training Loss: 2.556945        Validation Loss: 3.832611
Start training ...
---validating---05
Epoch: 25        Training Loss: 2.495964        Validation Loss: 3.603918
Validation loss decreased (3.731860 --> 3.603918). Saving model...
Start training ...
---validating---05
Epoch: 26        Training Loss: 2.410185        Validation Loss: 3.810235
Start training ...
---validating---05
Epoch: 27        Training Loss: 2.403166        Validation Loss: 3.774473
Start training ...
---validating---05
Epoch: 28        Training Loss: 2.328172        Validation Loss: 3.756681
Start training ...
---validating---05
Epoch: 29        Training Loss: 2.255671        Validation Loss: 3.928945
Start training ...
---validating---05
Epoch: 30        Training Loss: 2.189152        Validation Loss: 3.596040
Validation loss decreased (3.603918 --> 3.596040). Saving model...
```

In [26]:
```
# load the model that got the best validation accuracy
```

```
model_scratch.load_state_dict(torch.load('model_scratch.pt'))
```

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 [27]: 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 [28]: # call test function
         test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 3.572880


Test Accuracy: 19% (159/836)
```

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 [29]: ## 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 [30]: import torchvision.models as models
         import torch.optim as optim
         import torch.nn as nn

         ## TODO: Specify model architecture
         model_transfer = models.resnet50(pretrained=True)

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

         model_transfer.fc = nn.Linear(2048, len(train_data.classes))

         if use_cuda:
             model_transfer = model_transfer.cuda()

         # print(model_transfer)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.torch/models/
100%|| 102502400/102502400 [00:03<00:00, 28080033.10it/s]
```

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: - Resnet50 effectively filters and extracts the features of an image, and my model works well with Resnet50 also has good performance vs error on ImageNet classification

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 [31]: criterion_transfer = nn.CrossEntropyLoss()
         optimizer_transfer = optim.Adam(model_transfer.fc.parameters(), lr=0.001,amsgrad=True)
```

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 [32]: # train the model
         model_transfer =  train(25, loaders_transfer, model_transfer, optimizer_transfer, crite
```

```
Start training ...
---validating---05
Epoch: 1         Training Loss: 2.782555         Validation Loss: 1.018352
Validation loss decreased (inf --> 1.018352). Saving model...
Start training ...
---validating---05
Epoch: 2         Training Loss: 1.279447         Validation Loss: 0.605651
Validation loss decreased (1.018352 --> 0.605651). Saving model...
Start training ...
---validating---05
Epoch: 3         Training Loss: 1.008431         Validation Loss: 0.543063
Validation loss decreased (0.605651 --> 0.543063). Saving model...
Start training ...
---validating---05
Epoch: 4         Training Loss: 0.900607         Validation Loss: 0.538040
Validation loss decreased (0.543063 --> 0.538040). Saving model...
Start training ...
---validating---05
Epoch: 5         Training Loss: 0.818085         Validation Loss: 0.500704
Validation loss decreased (0.538040 --> 0.500704). Saving model...
Start training ...
---validating---05
Epoch: 6         Training Loss: 0.792262         Validation Loss: 0.455593
Validation loss decreased (0.500704 --> 0.455593). Saving model...
Start training ...
---validating---05
Epoch: 7         Training Loss: 0.767624         Validation Loss: 0.453252
Validation loss decreased (0.455593 --> 0.453252). Saving model...
Start training ...
---validating---05
Epoch: 8         Training Loss: 0.732162         Validation Loss: 0.416357
Validation loss decreased (0.453252 --> 0.416357). Saving model...
Start training ...
---validating---05
Epoch: 9         Training Loss: 0.693188         Validation Loss: 0.409606
Validation loss decreased (0.416357 --> 0.409606). Saving model...
Start training ...
---validating---05
Epoch: 10         Training Loss: 0.698704         Validation Loss: 0.429507
Start training ...
---validating---05
```

```
Epoch: 11         Training Loss: 0.667821         Validation Loss: 0.522698
Start training ...
---validating---05
Epoch: 12         Training Loss: 0.650582         Validation Loss: 0.414409
Start training ...
---validating---05
Epoch: 13         Training Loss: 0.651728         Validation Loss: 0.399653
Validation loss decreased (0.409606 --> 0.399653). Saving model...
Start training ...
---validating---05
Epoch: 14         Training Loss: 0.628330         Validation Loss: 0.394217
Validation loss decreased (0.399653 --> 0.394217). Saving model...
Start training ...
---validating---05
Epoch: 15         Training Loss: 0.605699         Validation Loss: 0.373923
Validation loss decreased (0.394217 --> 0.373923). Saving model...
Start training ...
---validating---05
Epoch: 16         Training Loss: 0.610582         Validation Loss: 0.397530
Start training ...
---validating---05
Epoch: 17         Training Loss: 0.590168         Validation Loss: 0.396137
Start training ...
---validating---05
Epoch: 18         Training Loss: 0.611813         Validation Loss: 0.397425
Start training ...
---validating---05
Epoch: 19         Training Loss: 0.558744         Validation Loss: 0.375788
Start training ...
---validating---05
Epoch: 20         Training Loss: 0.587040         Validation Loss: 0.406574
Start training ...
---validating---05
Epoch: 21         Training Loss: 0.572522         Validation Loss: 0.451137
Start training ...
---validating---05
Epoch: 22         Training Loss: 0.552440         Validation Loss: 0.368315
Validation loss decreased (0.373923 --> 0.368315). Saving model...
Start training ...
---validating---05
Epoch: 23         Training Loss: 0.552594         Validation Loss: 0.367721
Validation loss decreased (0.368315 --> 0.367721). Saving model...
Start training ...
---validating---05
Epoch: 24         Training Loss: 0.542572         Validation Loss: 0.403496
Start training ...
---validating---05
Epoch: 25         Training Loss: 0.558888         Validation Loss: 0.398944
```

```
In [33]: # load the model that got the best validation accuracy (uncomment the line below)
         model_transfer.load_state_dict(torch.load('model_transfer.pt', map_location=lambda stor
```

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%.

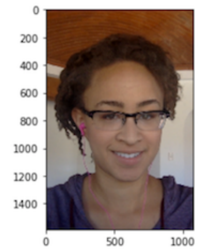
```
In [34]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.431932


Test Accuracy: 87% (729/836)
```

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

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

         def predict_breed_transfer(img_path):
             # load the image and return the predicted breed
             img = Image.open(img_path).convert('RGB')
             # set transofrm and turn to tensor
             transform = transforms.Compose([transforms.Resize(size=(224,224)),
                                            transforms.ToTensor()])
             #transform(img)
             img = transform(img).unsqueeze(0)
             # move to cuda if so
             if use_cuda:
                 img = img.cuda()
             output = model_transfer(img )
             # get the top predicted class
             class_index = torch.max(output,1)[1].item()

             return class_names[class_index] # return the class name / predicted breed
```



Sample Human Output

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 [36]:
```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.

def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    if dog_detector(img_path):
        output = predict_breed_transfer(img_path)
        img = Image.open(img_path).convert('RGB')
        print(f'Dog!!!!, He looks like a {output}.')
        plt.imshow(img)
        plt.show()
    elif face_detector(img_path):
        output = predict_breed_transfer(img_path)
        print(f'Hi, Human!')
        # use the image from dog files beside to show how it looks together
        fun =[i for i in dog_files if output.replace(' ','_') in i]
        img = Image.open(img_path).convert('RGB')
        fun_img = Image.open(fun[0])
        f, axarr = plt.subplots(1,2,gridspec_kw = {'width_ratios':[1,1]})
        axarr[0].imshow(img)
        axarr[1].imshow(fun_img)
        plt.show()
```

```
        print(f'You look like a ... {output}')
    else:
        print('Error!')
        img = Image.open(img_path).convert('RGB')
        plt.imshow(img)
        plt.show()
```

Step 6: Test Your Algorithm

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

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.

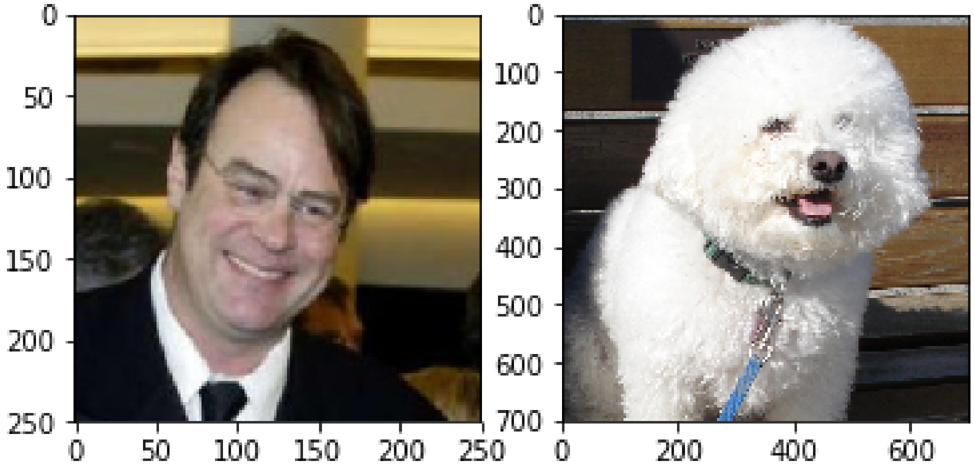
Answer: (Three possible points for improvement)

Actually I found the model works very well except for some errors in the test images as due to these images may not pass the face_detector function or not contain a dog nor a human which is considered correct for the latter case.

- Use a deeper model such as ResNet150
- train for longer
- experiment with different optimizers and learning rates

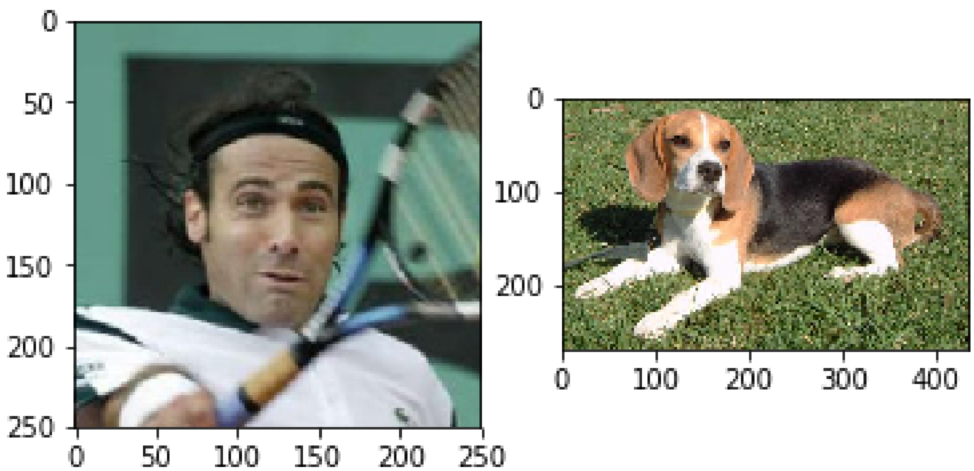
In [37]:
```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.

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

```
Hi, Human!
```



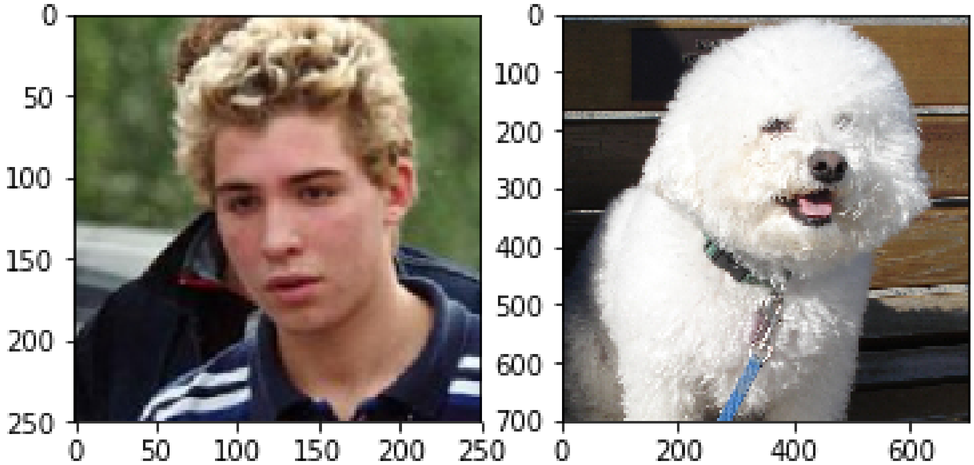
You look like a ... Bichon frise

Hi, Human!



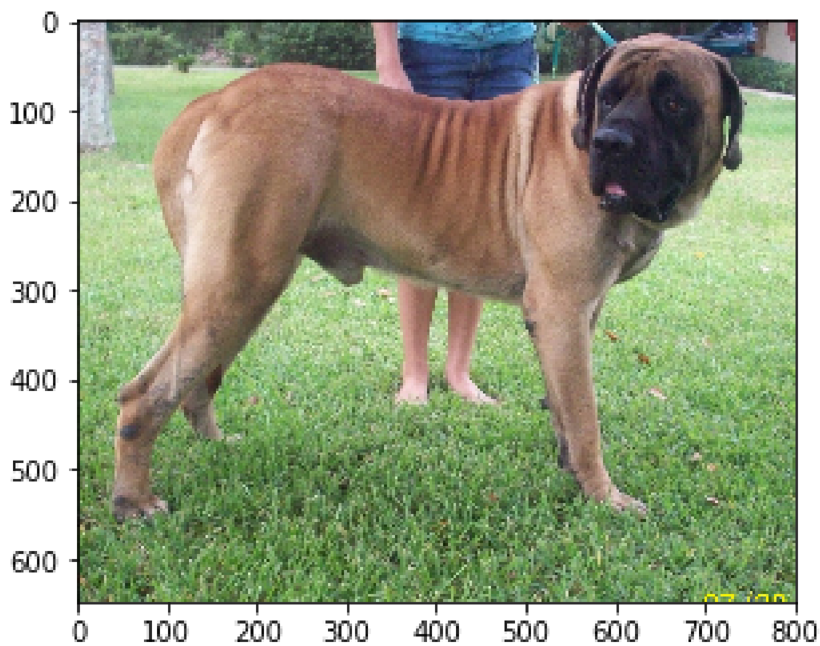
You look like a ... Beagle

Hi, Human!

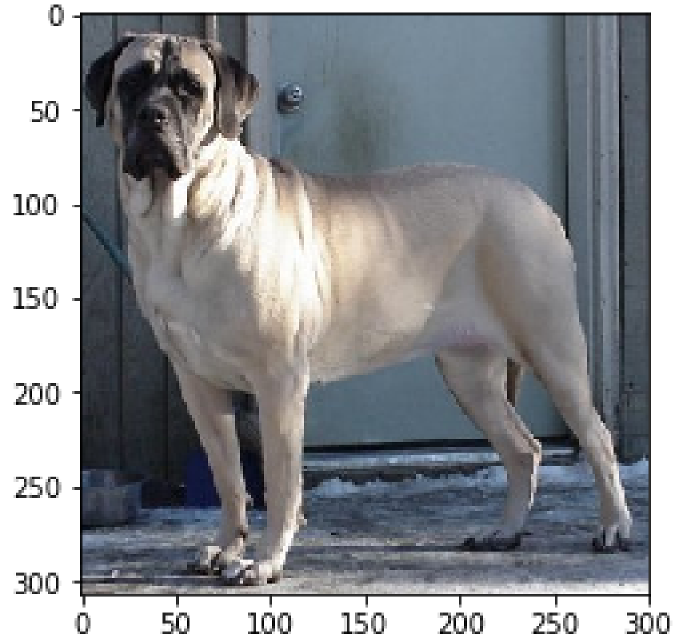


You look like a ... Bichon frise

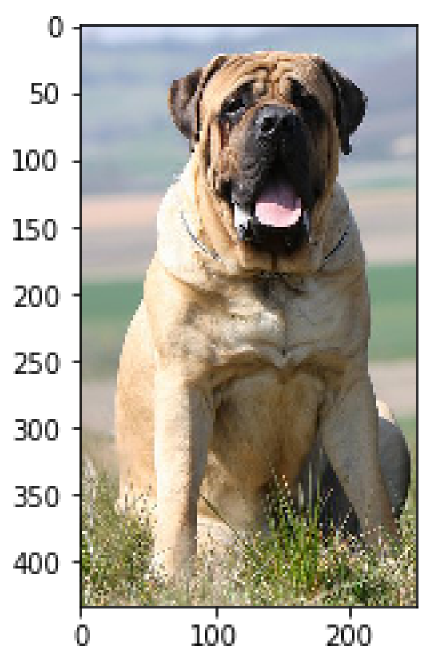
Dog!!!!, He looks like a Chinese shar-pei.



Dog!!!!, He looks like a Mastiff.



Dog!!!!, He looks like a Neapolitan mastiff.



```
In [38]: # make a new directory for test images
         ! mkdir test_imgs
```

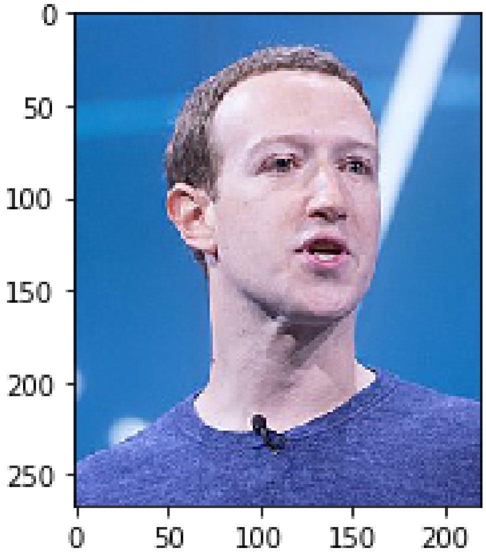
```
mkdir: cannot create directory test_imgs: File exists
```

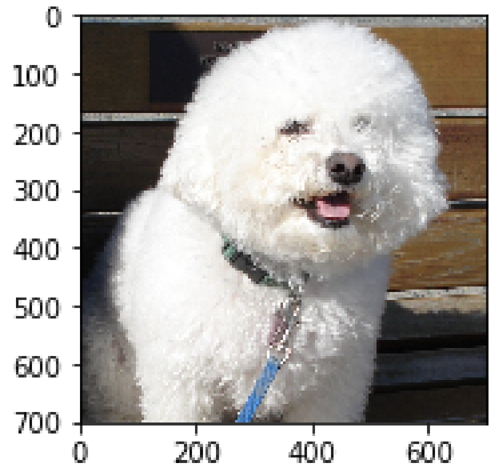
```
In [39]: # collect some images from the web for testing
         from urllib.request import urlretrieve
         links = \
         ['https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQLcgox8FlGTqAsGAdwDSp2SV9t69q5A
          'https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/All_Gizah_Pyramids.jpg/290p
          'https://upload.wikimedia.org/wikipedia/commons/thumb/e/e1/Baby_Face.JPG/220px-Baby_Fa
          'https://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/Mark_Zuckerberg_cropped.jpg
         'https://scontent-bru2-1.cdninstagram.com/vp/b46b8c23622526096b32297fd29a9285/5D1BAAEC/
                  ]
         for i,link in enumerate(links):
             urlretrieve(link,f'test_imgs/{i+1}.jpg')
```

```
In [40]: # test the app
         for i in np.array(glob('test_imgs/*')):
             run_app(i)
             print(' ')
```

```
Error!
```

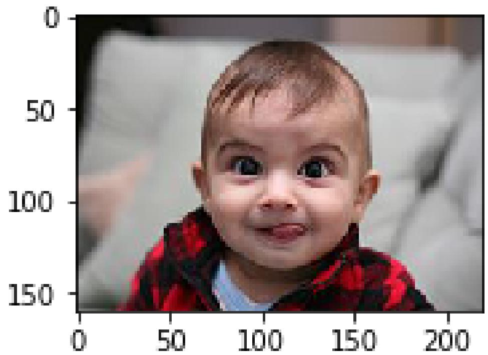
```
Hi, Human!
```

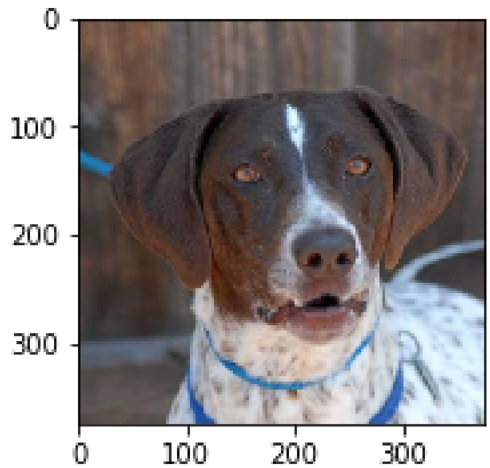





```
You look like a ... Bichon frise

Error!
```

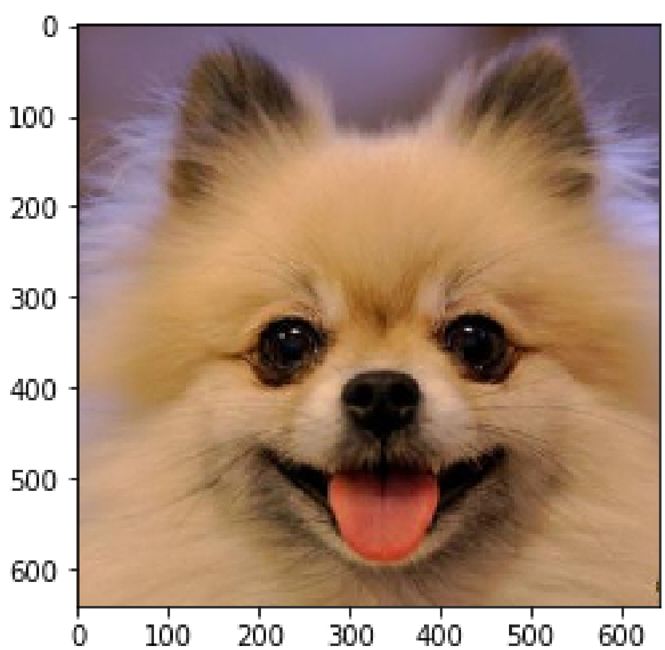
Hi, Human!





You look like a ... German shorthaired pointer

Dog!!!!, He looks like a Pomeranian.



```
In [ ]:
```