dog_app

March 16, 2021

1 Convolutional Neural Networks

1.1 Project: Write an Algorithm for a Dog Identification App

In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'(IMPLEMENTATION)'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section, and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the Jupyter Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

The rubric contains *optional* "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. If you decide to pursue the "Stand Out Suggestions", you should include the code in this Jupyter notebook.

Step 0: Import Datasets

Make sure that you've downloaded the required human and dog datasets:

Note: if you are using the Udacity workspace, you *DO NOT* need to re-download these - they can be found in the `/data` folder as noted in the cell below.

- Download the dog dataset. Unzip the folder and place it in this project's home directory, at the location `/dog_images`.
- Download the human dataset. Unzip the folder and place it in the home directory, at location `/lfw`.

Note: If you are using a Windows machine, you are encouraged to use 7zip to extract the folder.

In the code cell below, we save the file paths for both the human (LFW) dataset and dog dataset in the numpy arrays `human_files` and `dog_files`.

In [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

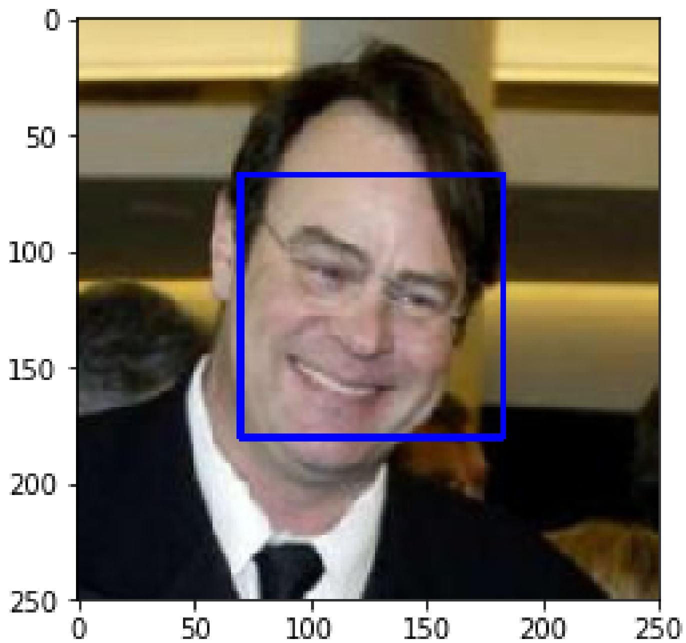
In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

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))
```

```python
    # 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:

The face dector function defined above detects 98% correctly the human faces from 100 images given in numpy array. This implies that the function face detector works neary properly in case of human faces identification. However, it also detects 17% human faces from the 100 images of dogs, which means it finds 17% dog images as human faces. Hence, we call this phenomenon as false positive, when the algorithm classifies incorrectly the dog images as human faces.

```
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 = [face_detector(human_file) for human_file in human_files_short]
        human_pct = (sum(human_faces)) / len(human_faces) * 100

        dog_faces = [face_detector(dog_file) for dog_file in dog_files_short]
        dog_pct = (sum(dog_faces) / len(dog_faces)) * 100

        print('Percentage of first 100 pictures in human files classified as humans is: {}%' .fo
        print('Percentage of first 100 pictures in dog files classified as humans is: {}%' .form
```

```
Percentage of first 100 pictures in human files classified as humans is: 98.0%
Percentage of first 100 pictures in dog files classified as humans is: 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 [5]: ### (Optional)
        ### TODO: Test performance of anotherface detection algorithm.
        ### Feel free to use as many code cells as needed.
```

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

1.1.3 Obtain Pre-trained VGG-16 Model

The code cell below downloads the VGG-16 model, along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

```
In [5]: import torch
        import torchvision.models as models

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

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

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

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.torch/models/vgg
100%|| 553433881/553433881 [00:21<00:00, 32961765.98it/s]
```

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

1.1.4 (IMPLEMENTATION) Making Predictions with a Pre-trained Model

In the next code cell, you will write a function that accepts a path to an image (such as `'dogImages/train/001.Affenpinscher/Affenpinscher_00001.jpg'`) as input and returns the index corresponding to the ImageNet class that is predicted by the pre-trained VGG-16 model. The output should always be an integer between 0 and 999, inclusive.

Before writing the function, make sure that you take the time to learn how to appropriately pre-process tensors for pre-trained models in the PyTorch documentation.

In [6]:
```
from PIL import Image
import torchvision.transforms as transforms

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

    Args:
        img_path: path to an image

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

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image
    transformation = transforms.Compose([transforms.CenterCrop(224), transforms.ToTensor

    #img = Image.open(img_path).convert('RGB')
    img_tens = Image.open(img_path).convert('RGB')
    img_tens = transformation(img_tens)[:3,:,:].unsqueeze(0)
    img_tens = img_tens.cuda()
    output = VGG16(img_tens)

    #Find max and return index of it
    if torch.cuda.is_available():
        prediction = output.cpu()

    max_indx = prediction.data.numpy().argmax()
    return max_indx
```

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 [7]:
```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    prediction = VGG16_predict(img_path)
    prediction = ((prediction <= 268) & (prediction >= 151))
```

```
        return prediction
```

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:

As described above the ideal situation here would be when our algorithm detects 0% of the human images as dogs and 100% dog images as dogs. In this case, the algorithm dog_detector falls short in away that it provides below ideal results. The dog_detector function defined above detects nearly 60% correctly the dog images in dog files. This implies that the function dog_detector works properly in 60% cases in case of dog images identification. Moreover, it detects 0% human faces in humane faces as dogs, which means dog_detector is doing perfectly fine in this situation.

```
In [8]: ### TODO: Test the performance of the dog_detector function
        ### on the images in human_files_short and dog_files_short.
        human_as_human = (np.mean([dog_detector(img) for img in human_files_short]))
        human_as_human_pct = human_as_human * 100

        dog_as_human = (np.mean([dog_detector(img) for img in dog_files_short]))
        dog_as_human_pct = dog_as_human * 100

        print('Percentage of human faces in human files detected as Dogs is: {}%'.format(human_a
        print('Percentage of dog faces in dog files detected as Dogs is: {}%'.format(dog_as_huma
```

```
Percentage of human faces in human files detected as Dogs is: 0.0%
Percentage of dog faces in dog files detected as Dogs is: 59.0%
```

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 [10]: ### (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 [9]: import os
        from torchvision import datasets

        ### TODO: Write data loaders for training, validation, and test sets
        ## Specify appropriate transforms, and batch_sizes
        data_dir = '/data/dog_images/'

        # Define path to images
        train_dir = os.path.join(data_dir, 'train')
        test_dir = os.path.join(data_dir, 'test')
        valid_dir = os.path.join(data_dir, 'valid')

        #Define Transforms
        #train transform
        train_transform = transforms.Compose([transforms.RandomHorizontalFlip(),
                                              transforms.RandomRotation(15), #20
```

```
                                                    transforms.RandomResizedCrop(224),
                                                    transforms.ToTensor(),
                                                    transforms.Normalize([0.485, 0.456, 0.406], [0.229

#test transform
test_transform = transforms.Compose([transforms.RandomResizedCrop(224),
                                    transforms.ToTensor(),
                                    transforms.Normalize([0.485, 0.456, 0.406], [0.229,

#Collect and transform images
train_data = datasets.ImageFolder(train_dir, transform=train_transform)
test_data = datasets.ImageFolder(test_dir, transform=test_transform)
valid_data = datasets.ImageFolder(valid_dir, transform=test_transform)

# prepare three data loaders
train_loader  = torch.utils.data.DataLoader(train_data, batch_size=15, num_workers=2, dr
test_loader   = torch.utils.data.DataLoader(test_data, batch_size=15, num_workers=2, dro
valid_loader  = torch.utils.data.DataLoader(valid_data, batch_size=15, num_workers=2, dr

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

Question 3: Describe your chosen procedure for preprocessing the data. - How does your code resize the images (by cropping, stretching, etc)? What size did you pick for the input tensor, and why? - Did you decide to augment the dataset? If so, how (through translations, flips, rotations, etc)? If not, why not?

Answer:

I loaded the required datasets, train, test and validation sets and then I specified dataloader for all the datasets.

As for as image size is concern, I resized image to 224 pixels, that is considered as a standard practice.

In addition to that I also choose rotation to avoid the issue of overfitting alonwith flip ([transforms.RandomHorizontalFlip()) and cropping of the images.

Then the image values were normalized with the standard means ([0.485, 0.456, 0.406]) and standard deviations ([0.229, 0.224, 0.225]).

1.1.8 (IMPLEMENTATION) Model Architecture

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

In [15]:
```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        ## Define layers of a CNN with Pooling max pooling function
```

```
        self.conv1 = nn.Conv2d(3, 32, kernel_size = (3, 3), stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size= (3, 3), stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size= (2, 2), stride=1, padding=1)
        self.conv4 = nn.Conv2d(128, 128, kernel_size= (2, 2), stride=1, padding=1)

        # Max pooling over a (2, 2) window
        self.pool = nn.MaxPool2d(2,2) #(kernel_size=2, stride=2, padding=1) #(2,2)

        ##Define image dimensions
        self.fc1 = nn.Linear(25088, 5000)
        self.fc2 = nn.Linear(5000, 512)
        self.fc3 = nn.Linear(512, 133)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        ## Conv layers
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        #Flatten
        x = x.view(-1, 25088)
        ## Linear NN
        x = self.dropout(x)
        ## Fully connected layers
        # linear layer, with relu activation function
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x

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

# instantiate the CNN
model_scratch = Net()

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

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

Answer:

The cnn model has standard parameters to build a CNN architecture, that can also be customized accrodingly. We can also note two important elements from the standard cnn architecture i.e., *CONV* and *POOL*. Where CONV are convolutional layers while POOL are pooling for translational invariance.

In our model above, each Conv2D defines CONV convolutional layer to the model, and each MaxPooling2D POOL applies max pooling to the convolutional layer. I followed the intrsuctions provided in the course and specify each convolutional layer and max pooling accordingly.

I used the MaxPooling2D type that is most popular and common in building CNNs. As we can note, as we increase the layers in the model the model becomes more powerfull in capturing the minor details of the input.

The convolutional layer takes images as input were we can also set number of filters, kernal size, padding, activation and input sapes.

The bottom part of the model includes a Flatten (flatten all inputs) layer followed by dropout. Finally, I specified the dropout, where dropout helps avoid overfitting the model.

I played around with the values to arrive the highest accuracy. The required accuracy for the model was around 10%, I obtained the accuracy near to 15%.

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 [16]: import torch.optim as optim

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

         ### TODO: select optimizer
         optimizer_scratch = optim.Adam(model_scratch.parameters(), lr=0.0001)
```

```
In [17]: from PIL import ImageFile
         ImageFile.LOAD_TRUNCATED_IMAGES = True
```

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 [18]: 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
                output = model(data)
                loss = criterion(output, target)
                optimizer.zero_grad()
                loss.backward()
                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()
            for batch_idx, (data, target) in enumerate(loaders['valid']):
                # move to GPU
                if use_cuda:
                    data, target = data.cuda(), target.cuda()
                ## update the average validation loss
                output = model(data)
                loss = criterion(output, target)
                valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss)

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

            ## TODO: save the model if validation loss has decreased
            if valid_loss <= valid_loss_min:
                print('Loss is decreasing please wait........!!!Saving the model......!!!')
                torch.save(model.state_dict(), save_path)
                valid_loss_min = valid_loss
            else:
                print("")
        # return trained model
        return model
```

In [25]:
```
# train the model
n_epochs = 20
model_scratch = train(n_epochs, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')
```

```
Epoch: 1         Training Loss: 4.279983         Validation Loss: 4.268752
```

```
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 2         Training Loss: 4.195704        Validation Loss: 4.205272
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 3         Training Loss: 4.114848        Validation Loss: 4.159376
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 4         Training Loss: 4.058909        Validation Loss: 4.108271
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 5         Training Loss: 3.997653        Validation Loss: 4.070939
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 6         Training Loss: 3.900729        Validation Loss: 3.982647
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 7         Training Loss: 3.862250        Validation Loss: 3.907566
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 8         Training Loss: 3.793994        Validation Loss: 3.882494
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 9         Training Loss: 3.726288        Validation Loss: 3.880668
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 10         Training Loss: 3.664929        Validation Loss: 3.874407
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 11         Training Loss: 3.624809        Validation Loss: 3.845572
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 12         Training Loss: 3.543462        Validation Loss: 3.890828

Training and Validation in Process
Epoch: 13         Training Loss: 3.544220        Validation Loss: 3.714831
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 14         Training Loss: 3.481562        Validation Loss: 3.769247

Training and Validation in Process
Epoch: 15         Training Loss: 3.413157        Validation Loss: 3.731093

Training and Validation in Process
Epoch: 16         Training Loss: 3.388513        Validation Loss: 3.680803
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 17         Training Loss: 3.333695        Validation Loss: 3.695065

Training and Validation in Process
Epoch: 18         Training Loss: 3.321254        Validation Loss: 3.679186
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 19         Training Loss: 3.251336        Validation Loss: 3.648333
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 20         Training Loss: 3.238676        Validation Loss: 3.653587

Training and Validation in Process
```

In [27]:
```python
# 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 [28]: 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)
                 # loss calculation
                 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: 3.743888


Test Accuracy: 15% (127/836)
```

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.nn as nn

         ## TODO: Specify model architecture
         model_transfer = models.vgg19(pretrained=True)


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

```
Downloading: "https://download.pytorch.org/models/vgg19-dcbb9e9d.pth" to /root/.torch/models/vgg
100%|| 574673361/574673361 [00:17<00:00, 33736538.21it/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:

In the below, I used Convolutional Neural Network CNN Architecture by using a pretrained vgg-19 model. In this model we will follow the transfer learning approach to our underlyong classification problem. The model adopted here is already pretained with the torchvision model archives.

This new algorithm with transfer learning performed much better and arrived at the accuracy of around (72%) that is quit above the desired accuracy level (60%).

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.SGD(model_transfer.classifier.parameters(), lr=0.001)
```

1.1.15 (IMPLEMENTATION) Train and Validate the Model

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

In [32]:
```
# train the model
n_epochs = 15
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: 7.298234 	Validation Loss: 5.214440
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 2 	Training Loss: 4.878393 	Validation Loss: 3.791075
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 3 	Training Loss: 3.788210 	Validation Loss: 2.827698
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 4 	Training Loss: 2.951922 	Validation Loss: 2.052420
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 5 	Training Loss: 2.380794 	Validation Loss: 1.708085
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 6 	Training Loss: 2.075150 	Validation Loss: 1.509628
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 7 	Training Loss: 1.858495 	Validation Loss: 1.305457
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 8 	Training Loss: 1.671076 	Validation Loss: 1.264831
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 9 	Training Loss: 1.539009 	Validation Loss: 1.214482
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 10 	Training Loss: 1.468547 	Validation Loss: 1.081357
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 11 	Training Loss: 1.413087 	Validation Loss: 1.014098
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 12 	Training Loss: 1.349573 	Validation Loss: 1.031468

Training and Validation in Process
Epoch: 13 	Training Loss: 1.303341 	Validation Loss: 0.983229
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 14 	Training Loss: 1.266887 	Validation Loss: 0.972663
Loss is decreasing please wait...!!!Saving the model...!!!
Epoch: 15 	Training Loss: 1.232830 	Validation Loss: 1.028197

Training and Validation in Process
```

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

```
Test Loss: 1.027171


Test Accuracy: 72% (609/836)
```

1.1.17 (IMPLEMENTATION) Predict Dog Breed with the Model

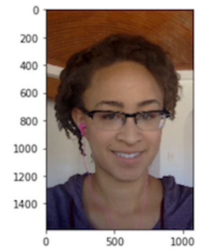
Write a function that takes an image path as input and returns the dog breed (`Affenpinscher`, `Afghan hound`, etc) that is predicted by your model.

```
In [34]: ### TODO: Write a function that takes a path to an image as input
         ### and returns the dog breed that is predicted by the model.
         data_transfer = {"train" : train_data, "valid" : valid_data, "test" : test_data}

         # 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]

         def predict_breed_transfer(img_path):
             # load the image and return the predicted breed
             transformation = transforms.Compose([transforms.CenterCrop(224),transforms.ToTensor
             img = Image.open(img_path).convert('RGB')
             img = transformation(img)[:3,:,:].unsqueeze(0)
             if use_cuda:
                 img = img.cuda()
             output = model_transfer(img)
             #Find max and return index of it
             max_indx = 0

             for i in range(len(output[0])):
                 if(output[0, max_indx] < output[0, i]):
                     max_indx = i

             return class_names[max_indx] # predicted class
```

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.

hello, human!



You look like a ...
Chinese_shar-pei

Sample Human Output

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 [35]: ### 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):
                 print("hi, Doggy!")
                 print("You belong to a dog breed of .....")
             elif face_detector(img_path):
                 print("hi, Gentleman!")
                 print("You look like a.....")
             else:
                 print("You are NOT a human nor a dog")
                 print("But it seems you may belong to the dog breed .....")
             predicted_breed = predict_breed_transfer(img_path)
             print(predicted_breed)
```

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)

The output of the model is better than what I expected given the data because it predicted all of the images of dogs correctly. In addition, I given images of humans which the model predicted them succesfully as human.

i. We can always improves the performance of our model by increasing the amount of data.

ii. It might also be a good idea to jointly use neural networks like Neural network ensemble.

iii. In order to improve the performance we could also use augmentation techniques along with playing around with some additional improvement mechansims such as hyperparameters and gradient descendent optimizer as well.

In [40]:
```
## 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)
```

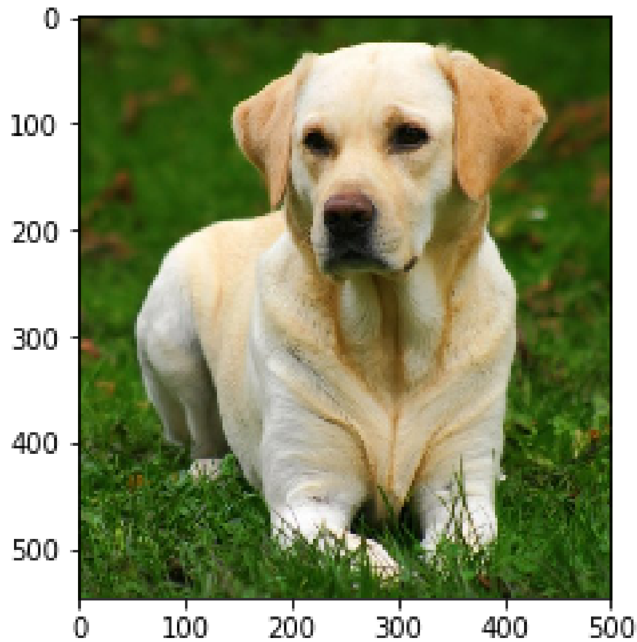
In [41]:
```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.
from PIL import Image
#import matplotlib.pyplot as plt
#import numpy as np

def display_image(img_path):
    image = Image.open(img_path, "r")
    plt.imshow(np.array(image))
```

In [42]:
```
img_path = "images/American_water_spaniel_00648.jpg"
display_image(img_path)
run_app(img_path)
```

```
hi, Doggy!
You belong to a dog breed of ...
Curly-coated retriever
```

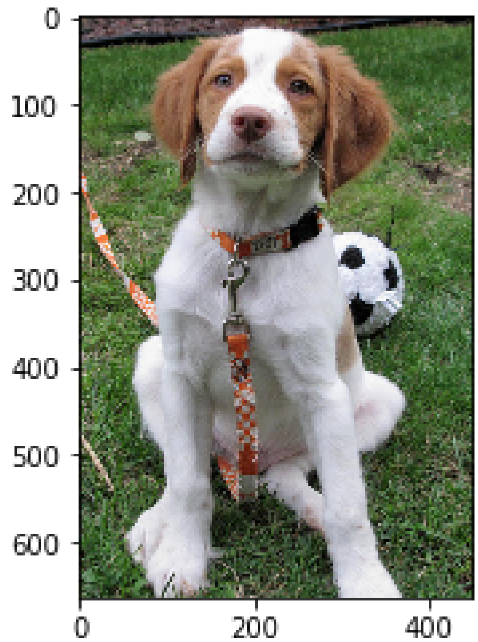

```
In [43]: img_path = "images/Labrador_retriever_06457.jpg"
         display_image(img_path)
         run_app(img_path)
```

```
hi, Doggy!
You belong to a dog breed of ...
Great pyrenees
```



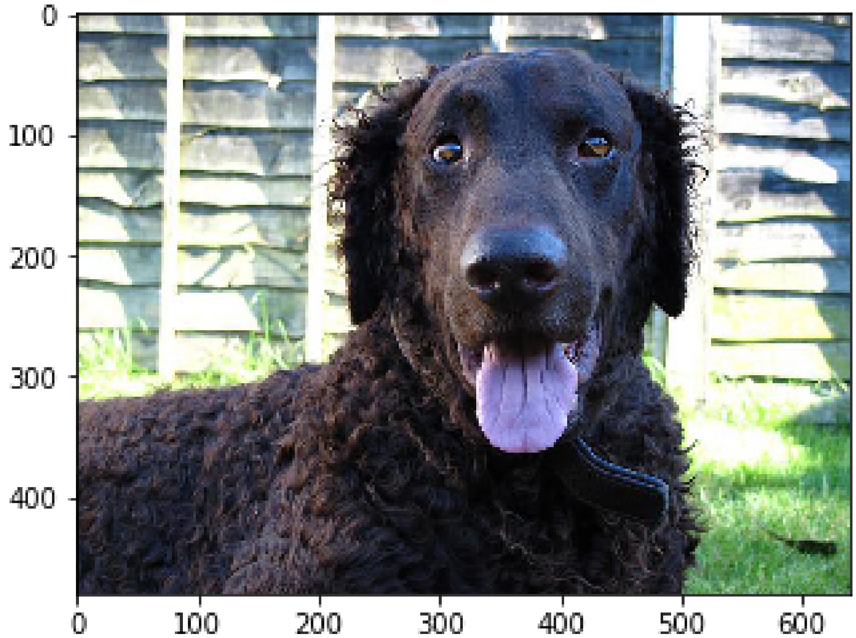

```
In [44]: img_path = "images/Brittany_02625.jpg"
         display_image(img_path)
         run_app(img_path)
```

```
hi, Doggy!
You belong to a dog breed of ...
American eskimo dog
```



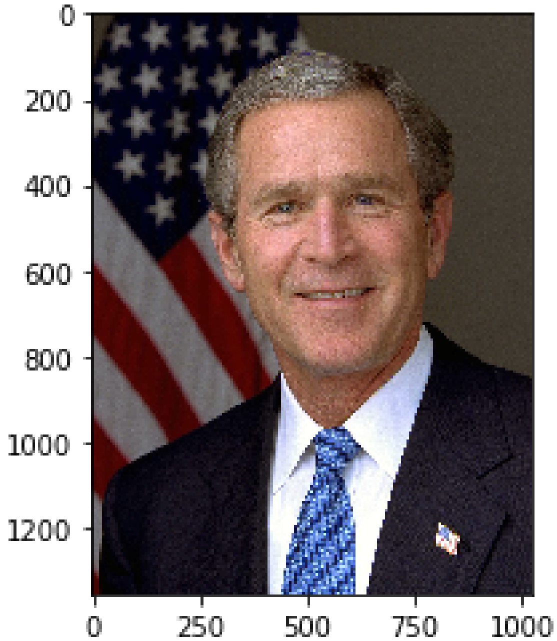

```
In [45]: img_path = "images/Curly-coated_retriever_03896.jpg"
         display_image(img_path)
         run_app(img_path)
```

```
hi, Doggy!
You belong to a dog breed of ...
Irish water spaniel
```



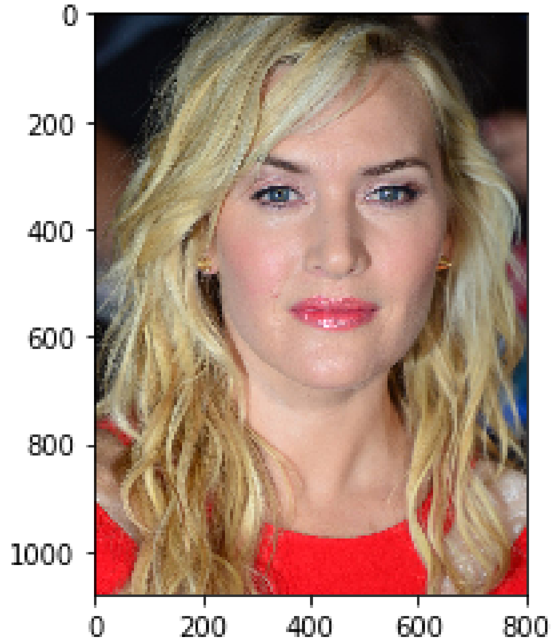
```
In [48]: img_path = "images/bush.jpeg"
         display_image(img_path)
         run_app(img_path)
```

```
hi, Gentleman!
You look like a...
American foxhound
```




```
In [53]: img_path = "images/kat.jpeg"
         display_image(img_path)
         run_app(img_path)
```

```
hi, Gentleman!
You look like a...
Nova scotia duck tolling retriever
```




```
In [ ]:
```