Machine Learning Engineer Nanodegree
Capstone Final Report By: Felicia Tamasovics
Convolution Neural Networks - Dog Breed Classification

Project Definition

Project Overview

This project aims to identify dog breeds using Convolutional Neural Networks (CNN). Throughout this course, I have learned about Image Classification problems within Deep Learning Models. I have decided to explore this more in this capstone as CNN fascinates me as I use it on a daily basis to open my iPhone using facial recognition.

An individual may want to determine the breed of a dog for many reasons. One particular reason could be in determining common behavioural traits for that specific breed. [1] For example: The following breeds have similar physical traits: Beauceron, Rottweiler, Doberman, however differing temperaments. Beaucerons are commonly known for being loving, while Rottweilers have been known to be fearless and protective. [2] Since dogs have been breed for various reasons, it is evident that some are better as a service dog or a therapy dog. It could also be beneficial to know how much physical activity and attention the breed requires. Taking care of various dogs in my personal life, I have adapted to their needs: the Doberman requires lots of physical activity so he gets taken on lots of walks/runs, whereas the Maltese Chihuahua does not so we play fetch, or short walks for activity.

I also thought it would be fun to see what dog I most resemble!

The input data was provided by Udacity and can be downloaded using the links provided in the top of the .pynb file, under section: Step 0: Import Datasets.

Problem Statement

This project will determine the breed of a dog using Machine Learning's Deep Learning CNN. The trained model will determine:

- an estimate of the dog's breed if a dog is detected
- an estimate of the dog breed that is most resembling if a human is detected
- state not a dog or human if neither a dog or human is detected

As with every model, it is important that it has a good performance. The notebook has specified that the CNN must attain at least 60% accuracy on the test set.

This project will be implemented in a web application using Flask to allow all to test out model.

Metrics

The provided notebook stated there must be at least 10% test accuracy attained during the creation of a CNN to classify dog breeds from scratch. As well as 60% accuracy on the test set when creating a CNN to classify dog breeds using Transfer Learning. The project can be evaluated on effectiveness and efficiency based on the percentages provided above. The notebook states the following: "[S]etting aside the fact that the classes are slightly imbalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%." Since the classes are imbalanced, the model will also be evaluated looking at the loss. The class imbalances can be seen in Figure 14: Distribution of Dog Photos for Each Breed for Train Dataset noticing that the amount of photos for each bread range from 26-77.

Analysis

Data Exploration

There are two different datasets, human and dog images, both provided by Udacity. All images are in a .jpg format and vary in size.

Dog Datasets:

- 8351 total images: 6680 train, 836 test, 835 validation
- 133 different breads
- Inconsistent number of photos for each breed
  - 3-10 photos of each breed in test
  - 4-9 photos of each breed in valid
  - 26-77 photos of each bread in train

Human Dataset:

- 13233 total images

Since the goal is not to identify a specific human, but rather if the image is a dog or a human, splitting into the varying train, test, and valid are not necessary. These images are solely used to test the performance of the Human Face Detector.

The number of images in each dataset were determined using the code below:

```
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]: dog_train_files = np.array(glob("/data/dog_images/train/*/*"))
        dog_test_files = np.array(glob("/data/dog_images/test/*/*"))
        dog_valid_files = np.array(glob("/data/dog_images/valid/*/*"))

        print('There are %d total dog images in train dataset.' % len(dog_train_files))
        print('There are %d total dog images in test dataset.' % len(dog_test_files))
        print('There are %d total dog images in valid dataset.' % len(dog_valid_files))

        There are 6680 total dog images in train dataset.
        There are 836 total dog images in test dataset.
        There are 835 total dog images in valid dataset.
```

Figure 1: Code Snippet

Data Visualization

While looking at photos of dogs, it was apparent that most photos were of the individual dog, however some photos contained multiple dogs of the same breed, different breed, and sometimes humans were in the photo. This can be challenging to the model as it can be confused whether to detect a human or dog. This can also be challenging to the model if different breeds are presented within a photo that has been determined to be of one type. It will make assumptions on the breed based on characteristics presented in the photo.

For example, below are two photos of a Doberman Pinscher from the valid, and train datasets. There are individual photos and photos of multiple Doberman Pinschers.

Figure 2: Doberman Pinscher

Figure 3: Doberman Pinscher

However looking at the following photos in Figure 4, 5, 6, it is clear where challenges arise for the model. During the training, there was a human in the photo for a beagle. While testing the model for a Mastiff dog, there were two Shih Tzu's in the photo, so that can be challenging in determining which breed to detect. Lastly, in the validation dataset, the photo of an Anatolian Shepherd Dog also contained a wild animal. There are many more cases just like the following for other breeds as well.

Figure 4: Train – Beagle

Figure 5: Test – Mastiff

Figure 6: Valid – Anatolian Shepherd Dog

Having 13233 total images hopefully allows for individuals of all races to be present. This will be helpful in detecting humans as people have different characteristics. Below are sample human photos.

Figure 7: Bill Gates

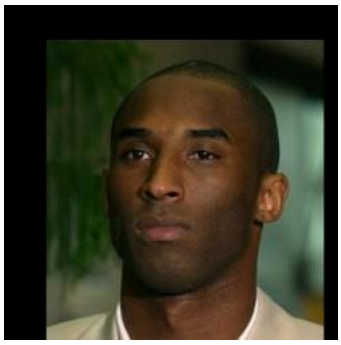


Figure 8: Kobe Bryant


Figure 9: Hassan Nasrallah


Figure 10: Hayden Panettiere


Figure 11: Phoenix Chang


Figure 12: Maria Garcia

The following code was utilized in creating the graph in Figure 14: Distribution of Dog Photos for Each Breed for Train Dataset which determines the number of photos per dog breed. It also depicted the range of photos per dog in each dataset. Similar graphs were produced using this code for test and valid with modifications to dog_path:

```
dog_path = '/data/dog_images/test'
dog_path = '/data/dog_images/valid'
```

```
In [4]: import os
        dog_path = '/data/dog_images/train/'
        class_names = os.listdir(dog_path)
        class_dict = {}
        dog_names = []
        for counts, name in enumerate(class_names):
            class_dict[name] = len(os.listdir(os.path.join(dog_path, name)))
            dog_names.append(list(class_dict)[counts].split('.')[1].replace('_', ' ').title())

In [5]: print('There are a minimum of %d dog images in train dataset.' % min(list(class_dict.values())))
        print('There are a maximum of %d dog images in train dataset.' % max(list(class_dict.values())))

        There are a minimum of 26 dog images in train dataset.
        There are a maximum of 77 dog images in train dataset.

In [6]: import seaborn as sns
        fig, ax = plt.subplots()
        fig.set_size_inches(20,25)
        ax = sns.barplot(x=list(class_dict.values()), y=dog_names)
        plt.ylabel("Dog Breeds")
        plt.xlabel("Number of Photos")
        plt.title("Distribution of Dog Breeds - Train Dataset")

Out[6]: Text(0.5,1,'Distribution of Dog Breeds - Train Dataset')
```

Figure 13: Code Snippet

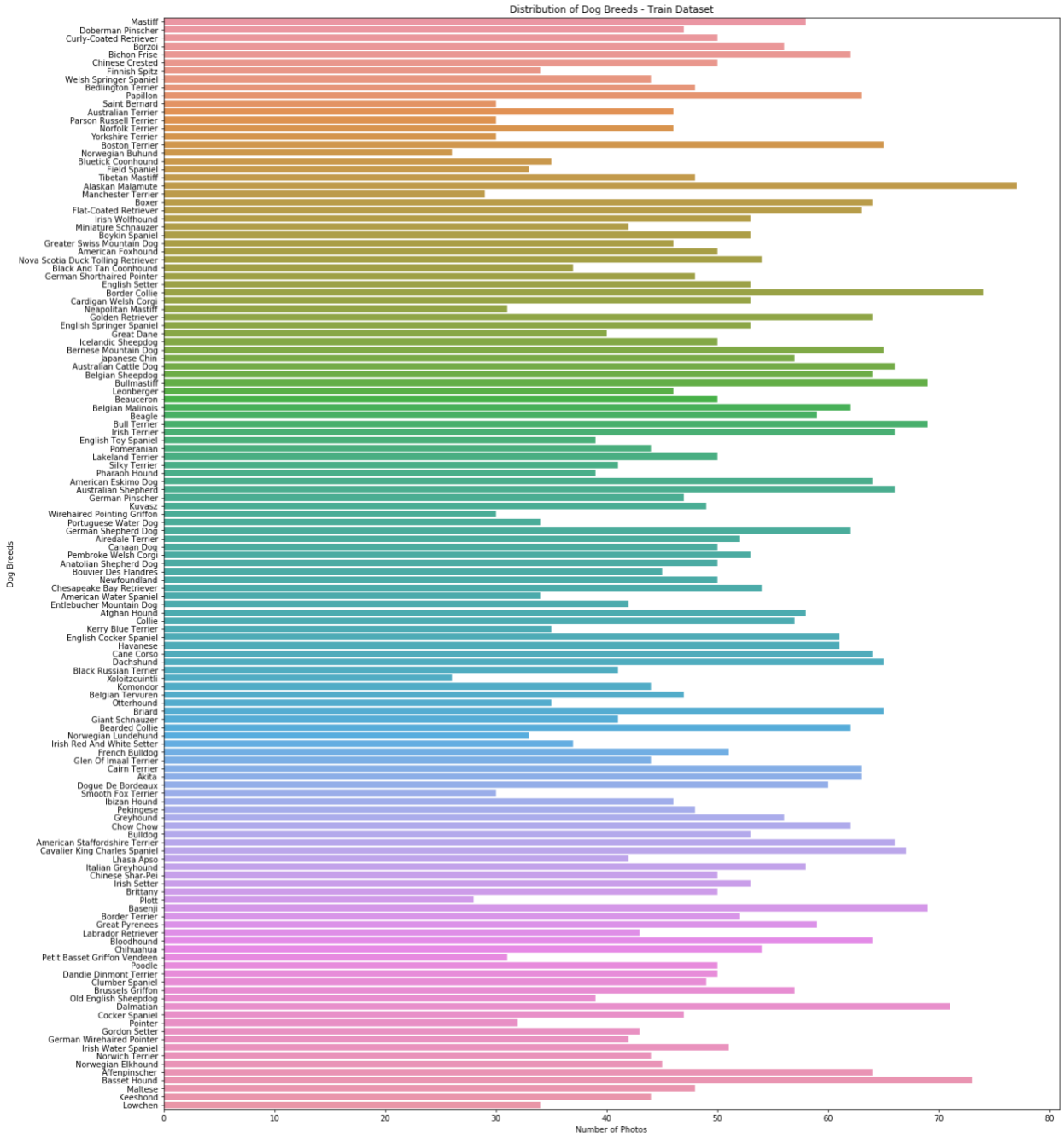


Figure 14: Distribution of Dog Photos for Each Breed for Train Dataset

Analyzing the data is important because it can showcase if datasets are imbalanced and not well profiled. The visuals provided above clearly depict that there is an imbalance with the provided dataset as some dog breeds have 26 photos while others have 77. Since the classes are imbalanced, accuracy is not the most optimal metric for the dataset. Therefore, the loss was utilized as the scoring metric when training/tuning the model rather than just accuracy alone.

Methodology

Data Preprocessing

The provided images varied in size, so it was important to process the image before passing it to the network. The images were resized, cropped, normalized, rotated, and flipped. These steps are beneficial because it enhances the images by helping the network to generalize allowing for better training observations and leading to better performance. The code in Figure 15: Preprocessing Data, depicts the image getting altered in the following ways:

Resizing the images:

- resize of 255 since this is how RGB images are normalized
- cropping size since input data is 224x224 pixels - required by the pre-trained network VGG16
- normalized with a mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225] since the colour channel was normalized separately, and therefore normalizing with the respective values is what the network expects

The images were augmented only on test dataset in the following ways:

- random rotation of 30
- random horizontal flip

The validation and testing sets are used to measure the model's performance on data it hasn't seen yet. Since this is the case, we don't want any scaling or rotation transformations, but we did need to resize then crop the images to the appropriate size.

```
In [25]: 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'
         train_dir = data_dir + '/train'
         valid_dir = data_dir + '/valid'
         test_dir = data_dir + '/test'

         data_transforms = {'train': transforms.Compose([transforms.RandomRotation(30),
                                                         transforms.RandomResizedCrop(224),
                                                         transforms.RandomHorizontalFlip(),
                                                         transforms.ToTensor(),
                                                         transforms.Normalize([0.485, 0.456, 0.406],
                                                                              [0.229, 0.224, 0.225])]),
                            'valid': transforms.Compose([transforms.Resize(255),
                                                         transforms.CenterCrop(224),
                                                         transforms.ToTensor(),
                                                         transforms.Normalize([0.485, 0.456, 0.406],
                                                                              [0.229, 0.224, 0.225])]),
                            'test': transforms.Compose([transforms.Resize(255),
                                                        transforms.CenterCrop(224),
                                                        transforms.ToTensor(),
                                                        transforms.Normalize([0.485, 0.456, 0.406],
                                                                             [0.229, 0.224, 0.225])]),
                           }

         # TODO: Load the datasets with ImageFolder
         image_datasets = {'train': datasets.ImageFolder(train_dir, transform=data_transforms['train']),
                           'valid': datasets.ImageFolder(valid_dir, transform=data_transforms['valid']),
                           'test': datasets.ImageFolder(test_dir, transform=data_transforms['test'])
                          }

         # TODO: Using the image datasets and the trainforms, define the dataloaders
         loaders_scratch = {'train': torch.utils.data.DataLoader(image_datasets['train'], batch_size=64, shuffle=True),
                            'valid': torch.utils.data.DataLoader(image_datasets['valid'], batch_size=64, shuffle=True),
                            'test' : torch.utils.data.DataLoader(image_datasets['test'], batch_size=64, shuffle=True)
                           }
```

Figure 15: Preprocessing Data

Implementation

Since the premise of the project is to determine the breed of a dog or if presented a human, to predict the breed they most represent, it is important to establish what is presented in the photo: a dog, human, neither. Therefore a human and dog detector algorithm was created and implemented. This can be seen in Figure 16: Human Detector and Figure 17: Dog Detector.

Step 1: Detect Humans

The human detector algorithm utilized the pre-trained Haar feature-based cascade classifiers from OpenCV to identify human faces. The notebook mentions it is standard procedure to convert images to grayscale. The detectMultiScale function executes the classifier stored in face_cascade and takes the grayscale image as a parameter. face_cascade is missing from the following image as it was applied before creating this function. It is provided below for reference. In 100 test images of humans, it recognized 98 as human.

```
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')
```

```
In [9]: # 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
```

```
In [10]: 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_human_files = 0
human_dog_files = 0

for human in human_files_short:
    if face_detector(human):
        human_human_files += 1
for human in dog_files_short:
    if face_detector(human):
        human_dog_files += 1

print(f"The percentage of the first 100 images in human_files have detected human faces: {human_human_files}%")
print(f"The percentage of the first 100 images in dog_files have detected human faces: {human_dog_files}%")
```

```
The percentage of the first 100 images in human_files have detected human faces: 98%
The percentage of the first 100 images in dog_files have detected human faces: 17%
```

Figure 16: Human Detector

Step 2: Detect Dogs

The dog detector algorithm was created using VGG16_predict function which can be found in Figure 18. VGG16 is missing from the screenshot, so for reference:

```
VGG16 = models.vgg16(pretrained=True)
```

It was suggested to used VGG-16 as a potential network to detect dog images in the algorithm, but was free to explore other pre-trained networks such as ResNet-50. Figure 16 shows the dog detector algorithm using VGG16 which determined 100 dogs recognized out of 100 dog images. When ResNet-50 was implemented it determined 99 dogs out of 100 dog images. Therefore, the VGG-16 model was used throughout the project even after ResNet50 was implemented since it had better predictions. This is ok since the VGG-16 model is only used for prediction and not for training. The function accepts a path to an image (such as '/data/dog_images/train/001.Affenpinscher/Affenpinscher_0001.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. For a dog to be detected, the index will be between 151 and 268 inclusive. This is reflected in the algorithm.

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.
    predicted_idx = VGG16_predict(img_path)
    result = predicted_idx >=151 and predicted_idx <=268
    return result # true/false
```

In [21]:
```
### TODO: Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.
dog_human_files = 0
dog_dog_files = 0

for dog in human_files_short:
    if dog_detector(dog):
        dog_human_files += 1
for dog in dog_files_short:
    if dog_detector(dog):
        dog_dog_files += 1

print(f"The percentage of the first 100 images in human_files have detected dogs: {dog_human_files}%")
print(f"The percentage of the first 100 images in dog_files have detected dogs: {dog_dog_files}%")
```

```
The percentage of the first 100 images in human_files have detected dogs: 0%
The percentage of the first 100 images in dog_files have detected dogs: 100%
```

Figure 17: Dog Detector using VGG-16

In [14]:
```
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

    image_transforms = transforms.Compose([transforms.Resize(255),
                                           transforms.CenterCrop(224),
                                           transforms.ToTensor(),
                                           transforms.Normalize([0.485, 0.456, 0.406],
                                                                [0.229, 0.224, 0.225])])


    tensor_transforms = image_transforms(Image.open(img_path).convert('RGB'))[:3,:,:].unsqueeze(0)

    if use_cuda:
        tensor_transforms = tensor_transforms.cuda()

    return torch.max((VGG16(tensor_transforms)), 1)[1].item() # predicted class index
```

Figure 18: VGG-16 Predict Function

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

This is where the data loaders mentioned in Section Data Preprocessing of this report was defined. The input shape is 224x224x3 since the input is an RGB image that's been cropped into 224x224. After the four convolution layers, the tensor was flattened by the final shape of the tensor (128x7x7 =25088) and then two linear layers with ReLU were added changing shape of the tensor from 25088 to 4096 to finally 133, the number of classes of dog breeds. A dropout of 0.2 was added to avoid overfitting. This architecture is shown in Figure 19. After training (using 75 epochs) and testing the model, it had a test loss of 3.706335 and a test accuracy of 14% (121/836).

In [27]:
```
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(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1)

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

        # linear layer
        self.fc1 = nn.Linear(128*14*14, 4096)
        self.fc2 = nn.Linear(4096, 133)

        # dropout layer
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        ## Define forward behavior
        x = self.pool(F.relu(self.conv1(x)))
        x = self.dropout(x)
        x = self.pool(F.relu(self.conv2(x)))
        x = self.dropout(x)
        x = self.pool(F.relu(self.conv3(x)))
        x = self.dropout(x)
        x = self.pool(F.relu(self.conv4(x)))
        x = self.dropout(x)

        #flatten
        x = x.view(-1, 128*14*14)

        #hidden layer
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

#-#-# You do 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()
```

Figure 19: CNN Architecture (from Scratch)

Refinement

Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)

Seeing that the previous model was only 14% accurate, it was decided a new model would be configured. The loaders used in Figure 15: Preprocessing Data were copied over to loaders_transfer. As discussed in the project proposal, pre-trained ResNet-50 model was investigated as it is widely known that ResNet provides good performance in image classification. This model was applied by freezing all layers, and then modifying the last layer from 2048 to 133 as that is the total number of classes for this project. This architecture is shown in Figure 20. The model was trained using the following parameters:

```
loaders_transfer = loaders_scratch.copy()
model_transfer = train(75, loaders_transfer, model_transfer, optimizer_transfer,
                       criterion_transfer, use_cuda, 'model_transfer.pt')
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer.fc.parameters(), lr=0.02)
```

After training (using 75 epochs) and testing the model, it had a test loss of 0.383980 and a test accuracy of 89% (746/836). Having a low test loss and high accuracy score, this was the chosen model.

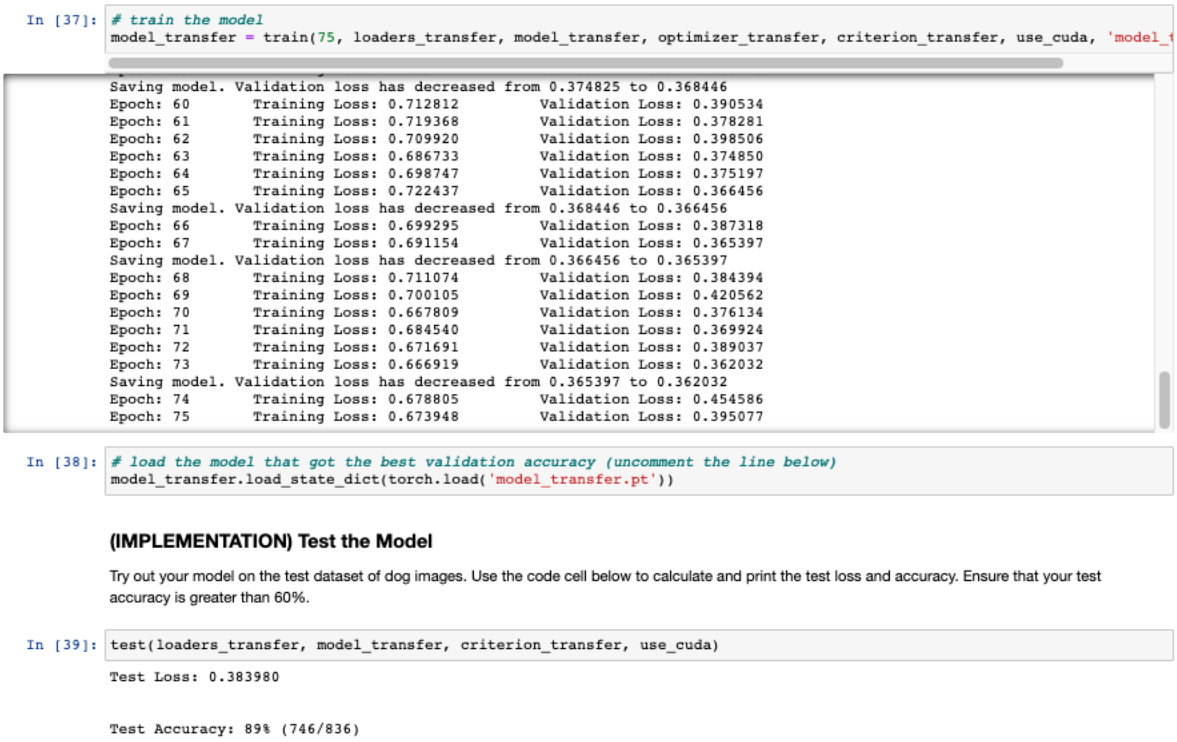
```
In [34]: import torchvision.models as models
         import torch.nn as nn

         ## 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, 133)

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

Figure 20: Final ResNet-50 CNN Architecture

Results

Model Evaluation and Validation

As discussed in the Refinement section, after training (using 75 epochs) and testing the model, it had a test loss of 0.383980 and a test accuracy of 89% (746/836). This can be seen in Figure 21.



```
In [37]: # train the model
         model_transfer = train(75, loaders_transfer, model_transfer, optimizer_transfer, criterion_transfer, use_cuda, 'model_t

Saving model. Validation loss has decreased from 0.374825 to 0.368446
Epoch: 60       Training Loss: 0.712812         Validation Loss: 0.390534
Epoch: 61       Training Loss: 0.719368         Validation Loss: 0.378281
Epoch: 62       Training Loss: 0.709920         Validation Loss: 0.398506
Epoch: 63       Training Loss: 0.686733         Validation Loss: 0.374850
Epoch: 64       Training Loss: 0.698747         Validation Loss: 0.375197
Epoch: 65       Training Loss: 0.722437         Validation Loss: 0.366456
Saving model. Validation loss has decreased from 0.368446 to 0.366456
Epoch: 66       Training Loss: 0.699295         Validation Loss: 0.387318
Epoch: 67       Training Loss: 0.691154         Validation Loss: 0.365397
Saving model. Validation loss has decreased from 0.366456 to 0.365397
Epoch: 68       Training Loss: 0.711074         Validation Loss: 0.384394
Epoch: 69       Training Loss: 0.700105         Validation Loss: 0.420562
Epoch: 70       Training Loss: 0.667809         Validation Loss: 0.376134
Epoch: 71       Training Loss: 0.684540         Validation Loss: 0.369924
Epoch: 72       Training Loss: 0.671691         Validation Loss: 0.389037
Epoch: 73       Training Loss: 0.666919         Validation Loss: 0.362032
Saving model. Validation loss has decreased from 0.365397 to 0.362032
Epoch: 74       Training Loss: 0.678805         Validation Loss: 0.454586
Epoch: 75       Training Loss: 0.673948         Validation Loss: 0.395077

In [38]: # load the model that got the best validation accuracy (uncomment the line below)
         model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

(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 [39]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)

Test Loss: 0.383980


Test Accuracy: 89% (746/836)
```

Figure 21: Results from ResNet-50 Model

Step 5&6: Write and Test Algorithm

Testing the algorithms that predict the breeds and seeing the performance is another way to visually validate the model. This algorithm was written keeping the objectives from the Problem Statement in mind. Figures 22 and 23 show the functions created to predict the breed of a dog and human.

In Figure 22, def predict_breed_transfer was challenging to do since using sigmoid did not return an expected result. A work around was done by calculating the probabilities manually for the top five expected. Returned values are the top five prediction classes along with the probability of each.

In Figure 23, def predict_breed_for_human calls predict_breed_transfer and stores the values in prob,

top_class_names, top_class. Wanting to print out the dog image that the human most represented, the path for the breed with the highest probability was determined. This was done by building off the given path '/data/dog_images/valid/' and adding to it based on the dog name along with the corresponding number that was associated to the image. For example: If human most represented an Afghan hound', the path would be created like the following:

```
top_class_names = Afghan hound, top_class = 1
file = '/data/dog_images/train/002.Afghan_hound'
```

human_dog would find all the files within that directory and join the first file it found to file Therefore, the final path for an image of the dog the human most represented is established as well as the name of the breed.

Figure 24 shows the code that produces the final result. It uses functions: face_detector, dog_detector, predict_breed_transfer, predict_breed_for_human.

In [66]:
```
### 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]

def predict_breed_transfer(img_path, model=model_transfer, topk=5):
    # load the image and return the predicted breed
    image = process_image(img_path)

    #convert from numpy to tensor - torch.Size([3, 224, 224])
    np_to_tensor = torch.tensor(image).float()

    #new tensor with a dimension of size one inserted at position 0 - torch.Size([1, 3, 224, 224])
    np_to_tensor = np_to_tensor.unsqueeze(dim=0)
    model = model.cpu()
    model.eval()

    with torch.no_grad():
        logps = model.forward(np_to_tensor)
        ps = torch.exp(logps)
        top_p, top_class = ps.topk(topk)
        p = top_p.numpy()[0]
        p0=p[0]
        p1=p[1]
        p2=p[2]
        p3=p[3]
        p4=p[4]
        p01=p0/(p0+p1+p2+p3+p4)
        p11=p1/(p0+p1+p2+p3+p4)
        p21=p2/(p0+p1+p2+p3+p4)
        p31=p3/(p0+p1+p2+p3+p4)
        p41=p4/(p0+p1+p2+p3+p4)
        tot = p01+p11+p21+p31+p41
        prob = [p01, p11, p21, p31, p41]
        top_class = top_class.numpy().tolist()[0]

        top_class_names = [class_names [i] for i in top_class]

#         if p0 >= 0.95:                          #was curious if the prediction is over 95%
#             print (p01)                         #call the dog a purebread
#             print(top_class_names[0])           #decided to leave as is so individuals could see what
                                                  #other breed the model predicted
    return prob, top_class_names, top_class
```

Figure 22: Breed Prediction Function For Dog

In [67]:
```
def predict_breed_for_human(img):
    prob, top_class_names, top_class = predict_breed_transfer(img)
    if top_class[0] == 0 or top_class[0] <= 8:
        file=("/data/dog_images/valid/00"+str(top_class[0]+1)+"."+top_class_names[0]+"/").replace(' ', '_')
    elif top_class[0] == 9 or top_class[0] <= 98:
        file=("/data/dog_images/valid/0"+str(top_class[0]+1)+"."+top_class_names[0]+"/").replace(' ', '_')
    else:
        file=("/data/dog_images/valid/"+str(top_class[0]+1)+"."+top_class_names[0]+"/").replace(' ', '_')

    human_dog = os.listdir(file)
    file = file+human_dog[0]
    top_class_name = top_class_names[0]

    return top_class_name, file
```

Figure 23: Breed Prediction Function For Human

```
In [68]: ### 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 face_detector(img_path):
                 print("Hi human")
                 plt.imshow(Image.open(img_path))
                 plt.show()
                 doggyname, filepath = predict_breed_for_human(img_path)
                 print(f'You look like a ... {doggyname}')
                 plt.imshow(Image.open(filepath))
                 plt.show()

             elif dog_detector(img_path):
                 print("Hi puppy!")
                 plt.imshow(Image.open(img_path))
                 plt.show()
                 print("I think you could be...")
                 probs, class_names, top_class = predict_breed_transfer(img_path)
                 plt.subplot(2,1,2)
                 sns.barplot(x=probs, y=class_names, color='blue');
                 plt.xlabel("Probability")
                 plt.ylabel("Dog Breeds")
                 plt.show()
             else:
                 plt.imshow(Image.open(img_path))
                 plt.show()
                 print("Could not detect a human or dog")
```

Figure 24: Run App Function

Figures 25-28 are a few example results from running the run_app function. They were ran when the image was of a dog, a bear (neither human or dog), a human, and a human using dog filter. These figures show the different outputs generated depending on the provided image. It allows for visual confirmation that the model is working as expected. Figure 25 shows the models top five predictions along with the probability of each. The top breed it predicted was a Mastiff and indeed it was. Figure 26 stated that it could not detect a human or dog, and that was correct considering the image is of a bear. Figure 27 is a photo of a human (me) and predicted as such. It stated the breed most resembling – Afghan Hound – which looks pretty accurate to me! Figure 28 is another photo of the same human (still me) with a dog filter and also produced the same breed prediction.

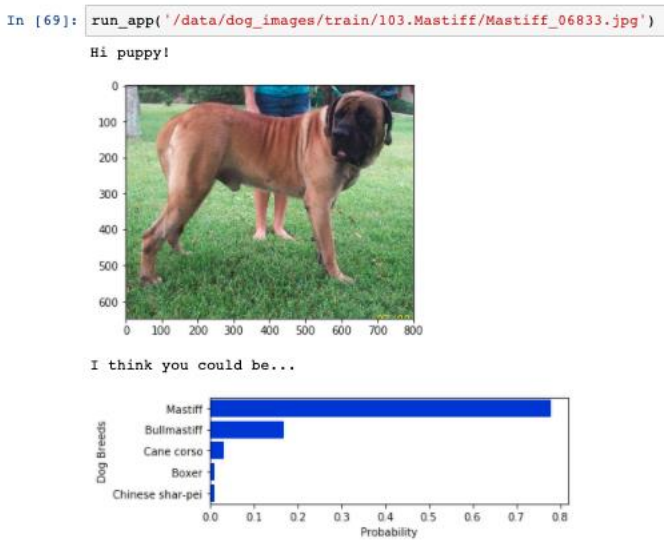


Figure 25: Sample Output of Dog

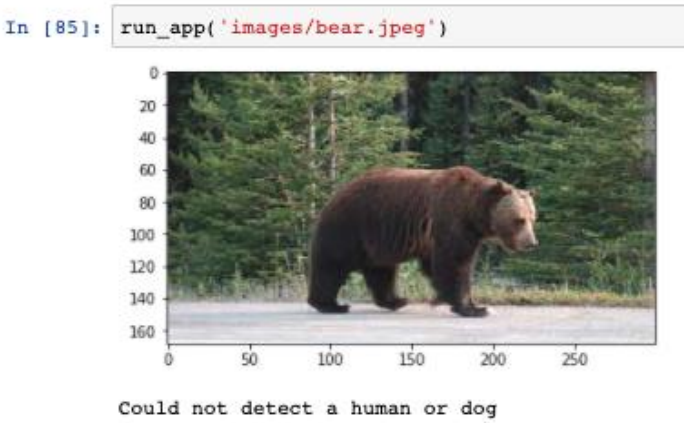


Figure 26: Sample Output of Neither Human or Dog

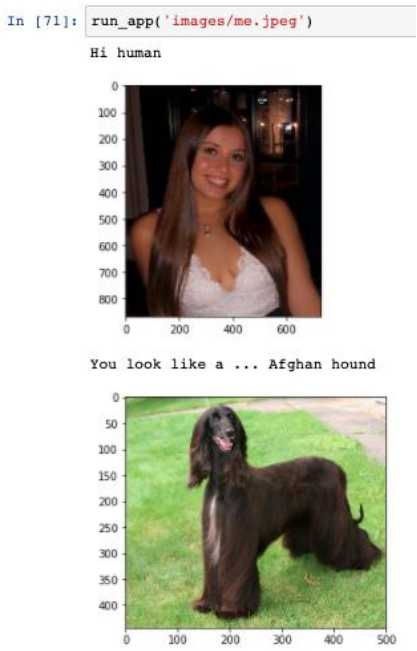


Figure 27: Sample Output of Human

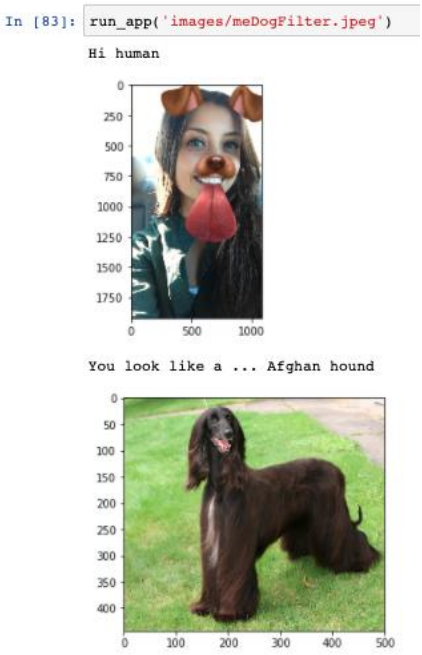


Figure 28: Sample Output of Dog

Justification

The final model is justified as it achieved the highest accuracy amongst all models that were tested throughout this capstone project. As updates were made the accuracy improved. One high accuracy rate was 87%, and after updating the number of epochs to 75 it improved to 89%. Since the notebook stated it must be at least 60%, this model should do. 89% is pretty great considering the project is harmless as it determines the breed of a dog! If the project was dealing with medical data it would be worth having a higher accuracy, as well as looking into the precision and recall. The loss 0.383980 which is an improvement to the first model that was tested, however it is still not the best compared to published model on Kaggle.

Conclusion

Reflection

- Step 0: Import Libraries and Datasets
- Step 1: Detect Humans using OpenCV's implementation of Haar feature-based cascade classifier
- Step 2: Detect Dogs using pre-trained VGG-16 Model (as a starting point)
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
- Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning) (RESNET-50)
- Step 5: Write an Algorithm that combines detected humans, detected dogs
- Step 6: Test Your Algorithm on points discussed in Problem Statement
    - an estimate of the dog's breed if a dog is detected
    - an estimate of the dog breed that is most resembling if a human is detected
    - state not a dog or human if neither a dog or human is detected
- Step 7: Deploy Model to Web Application using Flask

One aspect of the project that was challenging Step 3, creating a CNN from scratch. It was difficult creating the architecture that provided at least 10% accuracy. After lots of modifications, the final accuracy was 14%. This was done by playing around with the number of convolution layers and their parameters (ie: in_channels, out_channels, kernel_size, stride, padding).

Another aspect of the project that was very challenging was deploying the model to the web application using Flask. Having no exposure to Flask, lots of youtube videos were watched. One in particular was extremely helpful, and should be noted. [3] Although challenging, it was also the most rewarding and interesting. Seeing a web application using the model created and being able to upload dog photos that were not provided and seeing the prediction match the actual breed was worth the struggle.

Improvement

The model could be improved by adding more classes as there are currently only 133 dog breeds being accounted for. The Fédération Cynologique Internationale (World Canine Organisation) currently recognizes 354 different breeds. [4] This would help in determining the breed because if an individual uploads a breed that was not account for on the web app it would predict the breed based on the breeds it has been trained on.

Since there are not similar number of photos for each breed causing an imbalance issue, it would be beneficial to look update the dataset with the same amount of photos per breed.

As discussed in the Data Visualization section, the current datasets were trained, tested, and validated with photos that were not solely of the individual dog. Some photos included people, other breeds, and other animals. Since the premise of the project is to also determine whether a human or dog is present, it could be difficult for the model to determine which is detected - a human or dog - if both are actually present in the photo.

A minor improvement could be updating the design of the web application. It currently serves its purpose, but is not too pleasing to the eye.

Deliverables

Web Application

The model was deployed on a web application using Flask. Relevant sections of the notebook were implemented in files api.py, model.py and index.html, which can be found in the git repo. The model was loaded in the through model_transfer.pt. A bootstrap template example was used in creating the web app default page. [5]

Steps to Run Web App:

- python3 -m venv virtenv (create virtual environement)
- . virtenv/bin/activate (activate virtual environement)
- pip install module (install necessary modules)
- python pip freeze > requirements.txt (save modules that need to be installed)
- pip install -r requirements.txt (install required modules if starting a new session)
- python api.py (run web application)

Figures 29-31 show the output from the web application after uploading different images. Figure 29 is of a dog (mine) which predicted almost 50% he was a Maltese and that's technically correct considering he

is a Maltese Chihuahua. Figure 30 is the output of a bear which states it was neither a human or a dog, as we expect. Figure 31 is a photo of a human (my mom) which states she resembles a havanese. Looks pretty accurate!

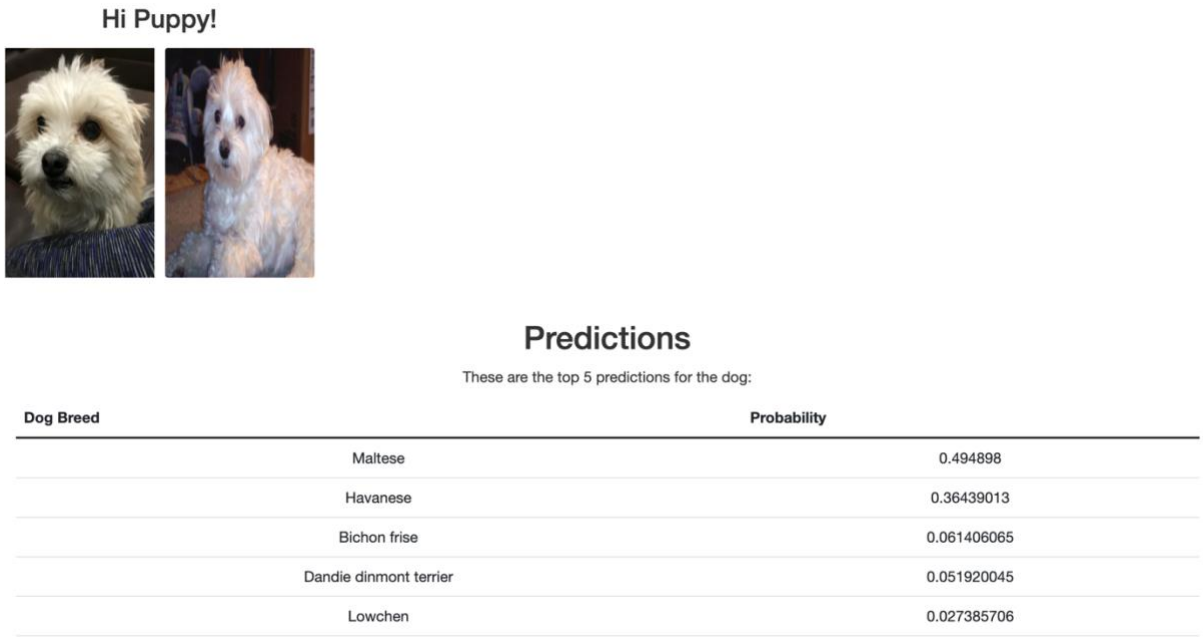


Figure 29: Sample Output of Dog on Web App

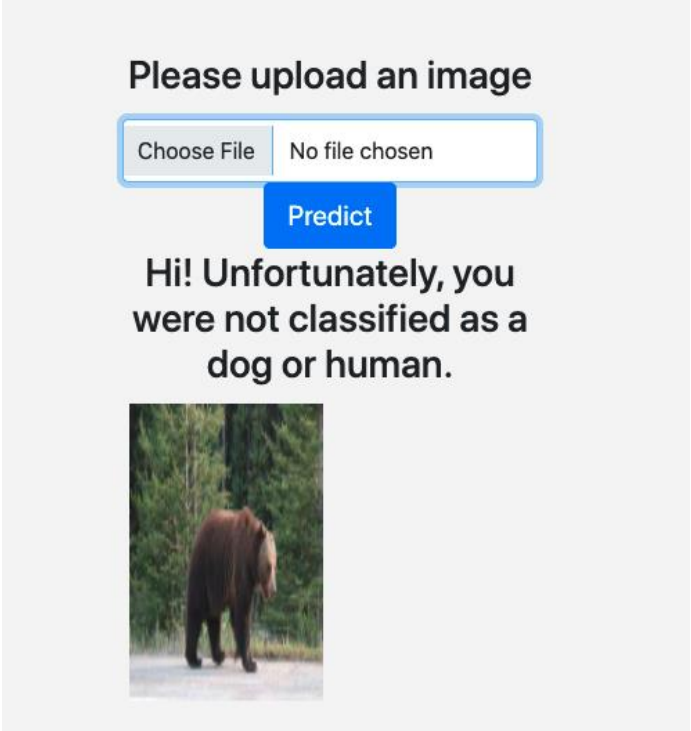


Figure 30: Sample Output of Not Human or Dog on Web App

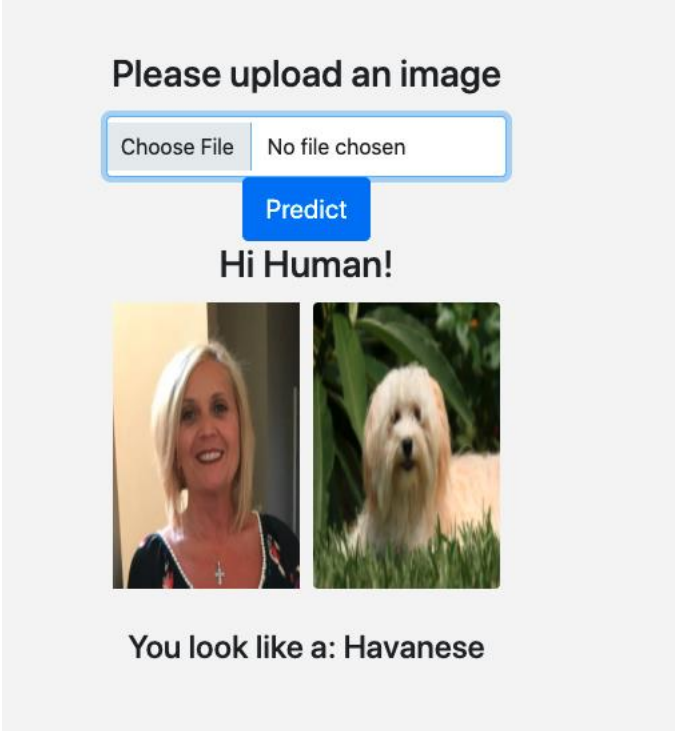


Figure 31: Sample Output of Human on Web App

References

[1] https://www.rover-time.com/3-reasons-to-know-your-dogs-breed/
[2] https://dogell.com/en/compare-dog-breeds/beauceron-vs-rottweiler-vs-doberman-pinscher
[3] https://www.youtube.com/watch?v=BUh76-xD5qU
[4] http://www.fci.be/en/Presentation-of-our-organisation-4.html
[5] https://getbootstrap.com/docs/4.0/examples/sign-in/