Machine Learning Engineer Nanodegree

Capstone Project: Dog Breed Classification

J. Taylor Isbell
December 21st, 2020

I. Definition

Project Overview

Computer vision is an active area of research which involves developing machine learning (ML) models that give computers the ability to extract information from imagery. This problem is one that we, as humans, solve almost constantly through our vision, but is surprisingly non-trivial to teach a machine how to solve. Computer vision is an extremely powerful technology and is currently being used in things like Tesla's self-driving car technology, Apple's FaceID, Google Translate, and many others.

In this project, a computer vision algorithm will be built and tested on two popular labeled datasets that include images of humans and dogs. Although there will be a definition of success for the algorithm itself, which will be discussed in the coming sections, the overall goal for this project will be to explore the process of building such an algorithm itself—from exploratory data analysis to model testing. And finally, several recommendations for improving the developed model will be made.

Problem Statement

In machine learning terminology, this problem is defined as a supervised classification task. This means that the model will be built based on example input-output pairs (i.e. "labeled data") and the output will be categorical (as opposed to numeric, or continuous). The goal of this project is to develop a model that can correctly predict the breed of a dog given only an image. The model should be able to accept an image of a dog OR a person, and it will then decide: 1. Whether the image contains a dog or a person, and 2. what dog breed the dog OR person most closely resembles.

As with most computer vision problems, deep learning will be employed as the primary tool for completing the task at hand. Deep learning is a family of machine learning methods based on artificial neural networks (ANNs), which are computing systems vaguely inspired by the biological neural networks that constitute animal brains.[1]

Metrics

The primary metric used in this project will be accuracy, which is one of the most common metrics used in any classification task. The definition of accuracy for the sake of this problem is defined below:

$$Accuracy = \frac{Images\ correctly\ classified}{Total\ images}$$

The details of the models used in this project will be discussed below, but in summary, there will be models developed to accomplish two different tasks: binary classification (i.e. human vs dog detection) and multi-class classification (i.e. inferring the correct dog breed). Since the former task only has two options, it is expected that the accuracy of those model(s) will be much higher than the multi-class model(s). In fact, the dog breed dataset includes images of 133 different dog breeds, which means that random guessing would result in less than 1% accuracy (as opposed to 50% for binary classification).

Another metric that will be used in this project is the F1 score, shown below:

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

Where precision is defined as the number of true positives divided by the number of true positives plus the number of false positives, and recall is defined as the number of true positives divided by the number of true positives plus the number of false negatives. This metric is used in classification tasks that have imbalanced datasets, or datasets where one class has more observations than the other. For this project, there are two binary classification tasks, and both tasks will have slightly imbalanced datasets. Also, for the non-binary classification task, the data is also imbalanced because some classes have more images than others. Therefore, it is important to utilize the F1 score so that the models are not over-optimistically scored based on accuracy alone.

II. Analysis

Data Exploration

[1] Artificial Neural Networks, http://en.wikipedia.org/wiki/Artificial_neural_network

The data needed for this project has already been collected and organized by Udacity. For the human images, the Labeled Faces in the Wild (LFW) dataset will be used.[2] For the dog images, the Stanford Dogs dataset will be used.[3] Both datasets are benchmark datasets that have been specifically curated, edited, and labeled for the use of training and testing computer vision models.

The human data contains 13,233 images. These images are colored and of various dimensions, but always contain a single human face and are titled by the name of human. However, please note that the names of the humans are irrelevant for the sake of this project. An example image is shown below in Figure 1.

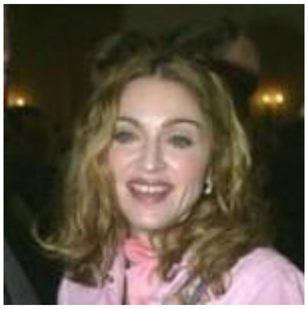


Figure 1: Image of Madonna

The dog data contains 8,351 images. Like the human images, these are also colored, of various dimensions, and are titled by the name of the dog's breed. However, in the dog images there are sometimes more than one dog (of the same breed) in a single image. An example image is shown below in Figure 2.

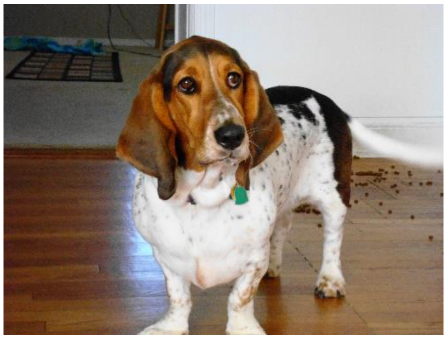


Figure 2: Image of a Basset hound

Because the dog and human datasets do not contain the same number of images, the data is therefore considered to be imbalanced. As discussed earlier, this will affect how the binary classification tasks (human and dog detector models) are evaluated.

[2] Labelled Faces in The Wild, http://vis-www.cs.umass.edu/lfw/

[3] ImageNet Dogs, http://vision.stanford.edu/aditya86/ImageNetDogs/main.html

Exploratory Visualizations

The dog dataset comes pre-split between training, validation, and test sets (for the ease of benchmarking). Figure 3 below shows the proportions of each set. It appears that the data was split 80/10/10 between the three sets.

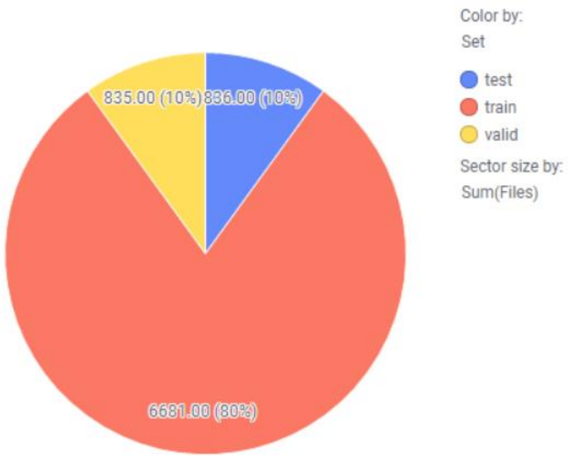
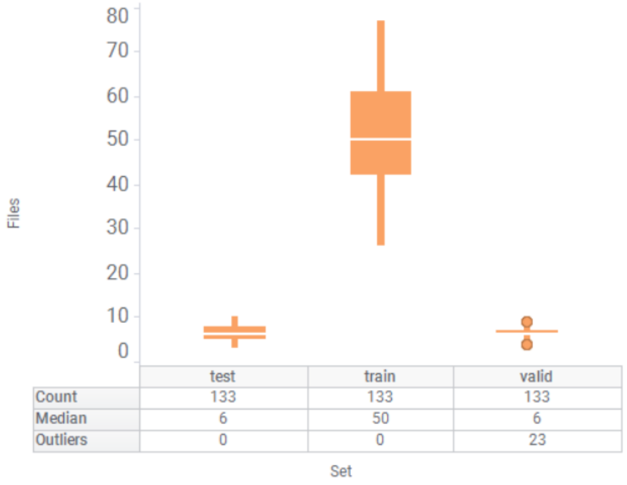


Figure 3: How the dog dataset is split between its modeling sets

Figure 4 below shows the distribution for the number of images each dog breed has attributed to it in the three sets. As shown, there are 133 dog breeds contained in this data. Furthermore, the training set has on average 50 images per dog breed, while the training and testing both have 6.



	test	train	valid
Count	133	133	133
Median	6	50	6
Outliers	0	0	23

Figure 4: box plot showing the distribution of each set

Again, this also portrays class imbalance in the data, but in a multi-class sense rather than binary. Because of this, the F1 score must also be used when evaluating the multi-class classification task (dog breed classifier). For this project, multi-class F1 score will be reported on a weighted average basis.[4]

Algorithms and Techniques

The specific model type used for this problem is known as a convolutional neural network (CNN). A CNN is a special kind of neural network that uses computational layers known as "convolutions" that reduce the complexity, and therefore the time required for model training, of 2D data like imagery.[5] To accomplish the solution described previously, there will need to be several CNN models used in sequence or parallel, each of which dedicated to a single task. For example, there will need to be one model for detecting human faces, one model for detecting dog faces, and one model for the actual dog breed classification. Therefore, each of these models will need to be evaluated separately before implementing the final data pipeline. And not only that, but the detection models will need to be tested on both the dog and human datasets. In total, there will be 5 accuracy evaluation steps in this project (dog detection model tested on dog and human data, human detection model tested on dog and human data, and the dog breed classifier model tested on dog data).

For this project, it is not required that every model be designed and trained from scratch (in fact, being able to apply architectures designed by others to new problems is one of the most important and profound features of the machine learning field!). The act of using a modified version of a pre-built model architecture is commonly referred to as "transfer learning." In this project, transfer learning will be used as well as testing out an architecture built from scratch. In the case of the human face detection model, OpenCV's implementation of Haar feature-based cascade classifiers will be used.[6] For the dog detection model, the pre-trained VGG-16 model will be used.[7] And finally, for the dog breed classification model, a built-from-scratch CNN will be tested as well as the pre-trained ResNet50 model.[8]

Benchmark

[4] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

[5] Convolutional Neural Network, http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[6] Cascade Classifier, https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html

[7] Very Deep Convolutional Networks for Large-Scale Image Recognition, https://arxiv.org/abs/1409.1556

[8] Deep Residual Learning for Image Recognition, https://arxiv.org/abs/1512.03385

Ayanzadeh and Vahidnia (2018) is a great study which aimed to benchmark several popular models using the Stanford dogs dataset.[9] In this work, models such as DenseNet-121, ResNet50, DenseNet169, and GoogleNet were tested. To tailor these pre-trained models for the specific task, a process they describe as "fine-tuning" was applied. This fine-tuning process involved only the fitting of the last few layers of the models while keeping the weights of all other layers fixed. This is also referred to as "transfer learning." They also utilized data augmentation, which is a popular tool used to prevent overfitting when training deep learning models.

Ayanzadeh and Vahidnia (2018) clearly showed that ResNet50, with fine tuning and data augmentation applied, performed the best out of all the models that were tested. This is an interesting finding and something that will be explored further in this project.

III. Methodology

Data Pre-Processing

Each of the three models employed in this project (i.e. human detector, dog detector, and dog breed classifier) require slightly different pre-processing steps.

For the human detector model, the only pre-processing step required was to convert the images from color to gray scale. For the dog detector model, VGG-16 (within the PyTorch library) was used, and hence the images needed to be converted to tensors. To do this, the images were pre-processed in the following sequence:

1. Convert to red-green-blue (RGB) color scale
2. Resize to 224x224 pixels.
3. Convert to tensor data type
4. Normalize tensor values to required specifications

Finally, the dog breed classifier model, also built using the PyTorch library, required similar pre-processing steps to the dog detector model, with some added steps to prevent overfitting. See the steps below:

1. Convert to red-green-blue (RGB) color scale
2. Train, test, and validate steps:
    a. Train: randomly crop to size 224x224 and random horizontal flips
    b. Valid: resize to 256x256 then center crop to 224x224
    c. Test: resize to 224x224

[9] Ayanzadeh, Aydin & Vahidnia, Sahand. (2018). Modified Deep Neural Networks for Dog Breeds Identification.

3. Convert to tensor data type
4. Normalize tensor values to required specifications

Implementation

The following section details the implementation process for each of the three separate models.

Human Detector

As previously discussed, a pre-trained model was used for the human detector model. See the code block below, which takes in an unprocessed image, and outputs the same image with a box around the detected face (if a face is indeed detected).

```
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()
```

Implementing this code into a function was done using the code below, which simply outputs “True” if a face is detected, and “False” if no face is detected.

```
# 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
```

Dog Detector

A pre-trained model was also used for the dog detector model. The VGG-16 model weights have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks.[10] ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.[11] The model object was first instantiated as follows:

```
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()
```

Then predictions were made from the model. These predictions were in the form of integers ranging from 0 to 999.

[10] ImageNet, http://www.image-net.org/

[11] GitHub ImageNet class idx, https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

```
from PIL import Image
import torchvision.transforms as transforms
from torch.autograd import Variable

# Set PIL to be tolerant of image files that are truncated.
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

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

def load_image(img_path):
    # Takes in an unprocessed image and performs the necessary transformat
ions (see data loaders)

    img = Image.open(img_path).convert('RGB')
    transformations = transforms.Compose([transforms.Resize(size=(224,224)
),
                                          transforms.ToTensor(),
                                          normalize])
    transformed_img = Variable(transformations(img)[:3,:,:].unsqueeze(0))
    if use_cuda:
        transformed_img = transformed_img.cuda()

    return transformed_img

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

    Args:
        img_path: path to an image

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

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image
    img = load_image(img_path)
    prediction = VGG16(img)
    return torch.max(prediction,1)[1].item() # predicted class index
```

Finally, the above code was made into a binary detection function much like the human detector.

```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path): ## TODO: Complete the function.
      return 151 <= VGG16_predict(img_path) <= 268 # true/false
```

Note that the return line specifies a specific range of integers needed to return "True." This is because ImageNet indexed their objects in such a way that all available dog breeds were grouped together between 151 and 268.

Dog Breed Classifier

In the case of the dog breed classifier, both a "built-from-scratch" as well as a pre-trained transfer learning model were tested. In fact, multiple scratch model architectures were tested, but none could match the performance of the pre-trained transfer learning model. These results will be discussed in detail in the following section.

As with the dog detector, the model object was first instantiated.

```
import torchvision.models as models import torch.nn as nn

model_transfer = models.resnet50(pretrained=True) model_transfer
```

This will return the architecture of the ResNet50 model, listing out all the arguments for every single layer. When printed out, notice the final fully connected (fc) layer:

```
(fc): Linear(in_features=2048, out_features=1000, bias=True)
```

This means that the model will attempt to assign the input images to 1 of 1000 categories. Since the dog dataset being used for this project only contains 133 dog breeds, this needed to be changed through the following:

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

model_transfer.fc = nn.Linear(2048, 133, bias=True)

fc_parameters = model_transfer.fc.parameters()

for param in fc_parameters:
    param.requires_grad = True

model_transfer
```

```
(fc): Linear(in_features=2048, out_features=133, bias=True)
```

After successfully altering the model for the task at hand, the model then had to be trained, validated, and tested on the dog dataset. These processes required refinement to achieve the desired results and will therefore be discussed in the coming sections.

Once the model achieved a sufficient accuracy during testing, it was then implemented into a classifier function that returns the actual name of the predicted dog breed.

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

def predict_breed_transfer(classes, img_path):
    # load the image and return the predicted breed

    image = load_image(img_path)
    model = model_transfer
    model.eval()
    prediction = model(image)
    idx = torch.max(prediction,1)[1].item() # predicted class index
    return classes[idx]
```

Refinement

The bulk of the refinement efforts in this project was dedicated to optimizing the dog breed classification models. For the scratch model, there were numerous options available for refinement. Examples of these options include the number and types of layers, number inputs and outputs for each layer, optimizer and loss functions, learning rate, and number of training epochs.

Based on various well-known CNN models, the scratch model was built based on a general architecture that is split into two sections: convolutional layers and fully connected layers. Each convolutional layer is activated via the rectified linear unit (ReLU) function and followed by a max pooling layer. Then the fully connected layers are succeeded by a dropout layer to minimize overfitting. Deciding on the number of layers and the configuration of each layer (i.e. stride, kernel size, padding, output channels, etc.) were mainly decided based on trial and error. Though it is required that the number of output channels for the final fully connected layer should be 133 to reflect the number of breeds included in the training dataset.

From beginning to end, roughly 10 different models were tested. The accuracy results of these models ranged from 7% to 15%. The main parameters that were easiest and most significant to alter were the learning rate and the configuration of each layer. In the end, an architecture was chosen which took the shortest amount of time to train while also passing the minimum accuracy threshold of 10%. This architecture reduced the tensor size from 224 to 7 through the 3 convolutional layers. Dropout was included before each fully connected layer to help minimize overfitting, and ReLU was chosen as the activation function as it is widely considered the best option for achieving superior performance while also maintaining low training times.

For the transfer learning model, much less refinement was required, as the architecture itself was deemed to be static. A few iterations of the model were executed with varying training epochs and learning rate until the minimum accuracy threshold of 65% was attained.

IV. Results

Model Evaluation and Validation

For the scratch dog breed classifier model, the final architecture is shown below:

```
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__()

        #Input channels
        self.conv1 = torch.nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1)
        self.conv2 = torch.nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1)
        self.conv3 = torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.pool = torch.nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.fc1 = torch.nn.Linear(7 * 7 * 128, 500)
        self.fc2 = torch.nn.Linear(500, 133)

        self.dropout = nn.Dropout(0.3)

    def forward(self, x):

        #Computes the activation of the first convolution
        #Size changes from 3x224x224 to 32x112x112
        x = F.relu(self.conv1(x))

        #Size changes from 32x112x112 to 32x56x56
        x = self.pool(x)

        #Size changes from 32x56x56 to 64x28x28
        x = F.relu(self.conv2(x))

        #Size changes from 64x28x28 to 64x14x14
        x = self.pool(x)

        #Size changes from 64x14x14 to 128x14x14
        x = F.relu(self.conv3(x))

        #Size changes from 128x14x14 to 128x7x7
        x = self.pool(x)

        #Flatten
        x = x.view(-1, 128 * 7 * 7)

        #Computes the activation of the first fully connected layer
        #Size changes from 1x(64*14*14) to 1x500
        x = self.dropout(x)
        x = F.relu(self.fc1(x))

        #Computes the second fully connected layer (activation applied later)
        #Size changes from 1x500 to 1x133
        x = self.dropout(x)
        x = self.fc2(x)
        return(x)
```

This model architecture was then trained and tested, using the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer function with a learning rate of 0.05. The training and testing results are shown below.

```
# train the model
model_scratch = train(10, loaders_scratch, model_scratch, optimizer_scratc
h, criterion_scratch, use_cuda, 'model_scratch.pt')
```

```
Epoch: 1        Training Loss: 4.084643        Validation Loss: 3.802421
Validation loss decreased (inf --> 3.802421).  Saving model ...
Epoch: 2        Training Loss: 4.018176        Validation Loss: 3.958538
Epoch: 3        Training Loss: 3.956521        Validation Loss: 3.720663
Validation loss decreased (3.802421 --> 3.720663).  Saving model ...
Epoch: 4        Training Loss: 3.938784        Validation Loss: 3.916544
Epoch: 5        Training Loss: 3.896763        Validation Loss: 3.666155
Validation loss decreased (3.720663 --> 3.666155).  Saving model ...
Epoch: 6        Training Loss: 3.845506        Validation Loss: 3.717946
Epoch: 7        Training Loss: 3.789072        Validation Loss: 3.632002
Validation loss decreased (3.666155 --> 3.632002).  Saving model ...
Epoch: 8        Training Loss: 3.789092        Validation Loss: 3.712640
Epoch: 9        Training Loss: 3.726013        Validation Loss: 3.700356
Epoch: 10       Training Loss: 3.706768        Validation Loss: 3.489647
Validation loss decreased (3.632002 --> 3.489647).  Saving model ...
```

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

```
Test Loss: 3.624341
Test Accuracy: 14% (124/836)
F1 Score: 0.12
```

For the transfer learning model, the training and evaluation steps were almost identical. One key difference, however, is that Adam was used for the optimizer function rather than SGD, and a learning rate of 0.01 was used rather than 0.05. The final model achieved an accuracy of 71% (594/836) with an F1 score of 0.69.

Justification

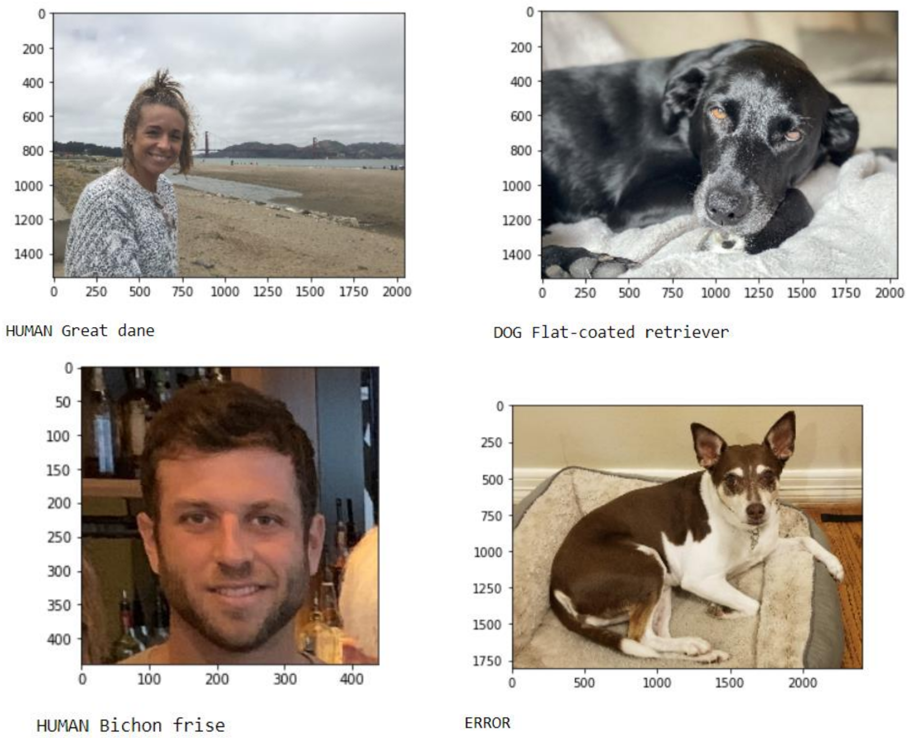
Compared to the benchmark study discussed previously, this project shares a lot of common methodologies. ResNet50 was chosen for the transfer learning model mostly due to the results shown in the benchmark study. However, since the model performed above the minimum accuracy threshold, other models were not explored, as was done in the benchmark study. Data augmentation and “fine-tuning” (i.e. transfer learning) were used in this project to achieve better results.

The performance of the dog breed classifier model in this project was not as good as the benchmark study (71% vs 90% accuracy). This is likely due to the fact that in the benchmark study, multiple layers were trained, while only the final fully-connected layer was trained in this project. Therefore, the benchmark model certainly required more computational resources and time to train. Because of this trade-off, both results are entirely acceptable, and the “best model” only depends on the constraints of the problem (i.e. optimizing for performance vs. model simplicity).

V. Conclusion

Free-Form Visualization

For further testing, some new, unprocessed images were fed to the final model. The figure below shows the results, which include the input image, the output of the human/dog detector model, and the output of the dog breed classifier model.



```
HUMAN Great dane
```

```
DOG Flat-coated retriever
```

```
HUMAN Bichon frise
```

```
ERROR
```

Figure 5: model results for new images

The above figure shows that even though the model tested well on the test dataset, there is still room for improvement. In the bottom right image, for instance, the model could not even detect that there was a dog in the image (or it detected that there was a dog AND a human), much less the dog’s breed.

Reflection

To summarize this project, the problem, defined as a supervised classification task was to develop a model that can correctly predict the breed of a dog given only an image. This was accomplished using a labeled human and dog dataset, both of which had to be pre-processed in particular ways. After the pre-processing, several models were built to accomplish single tasks. The modeling efforts involved testing multiple different architectures that were pre-trained, slightly modified, and completely built from scratch. In the end, the optimal models were packaged together in a single function that took in an image and output whether there was a human or dog detected and what dog breed the human or dog most closely resembled.

There were several interesting and challenging aspects of this project. One interesting aspect was shear amount of freedom there is when it comes to developing a CNN network from scratch. Networks can be built with an infinite combination of different convolutional layers, with different kernel sizes, and many other unique characteristics, but finding an optimal architecture that is both accurate and easily trainable can be incredibly difficult. Another interesting aspect was the unique functionality offered within the PyTorch package. Although it can seem daunting at first, PyTorch is incredibly thorough with the functions that they offer especially those surrounding data pre-processing, and their documentation is incredibly easy to navigate and understand. Despite the somewhat steep learning curve, it made the entire workflow easier and quicker in the long run.

Improvement

Although the model tested above the minimum accuracy requirement of 60%, when fed the images shown in the Free-Form Visualization subsection, it did not perform as well as expected. This might be due to the images being less clear than those in the training and testing sets. However, below are several possible points of improvement for the algorithm:

- Adding more random transformations to the test loader.
- Training more than just the final layer in the ResNet50 transfer learning model.
- Optimizing learning rate, training epochs, hyperparameters, etc.
- Adding the option to display a frame around the detected face
- Adding the capability to detect multiple dogs/humans in a single image