# Machine Learning Engineer Nanodegree

## Capstone Project: Dog-Breed Classifier

**Yash Naik**
**August 7, 2020**

## I. Definition

### Project Overview

This is Convolutional Neural Networks or CNN project. A CNN is a "deep learning algorithm that takes in an input image and assigns weights to different objects in the image" [1] so that it is easier to differentiate between the objects or the images. This project is part of Udacity's Machine Learning Engineer nanodegree program. This Machine Learning program can detect a human face and also identify the breed of a dog, given a picture. If, however, only a human face is detected, it gives an estimate of the dog breed that most resembles the face. This project is based upon a Deep Learning Model which utilizes Convolutional Neural Network to distinguish between 133 breeds of dogs and make predictions with a test accuracy of 86% given a set of images.

### Problem Statement

This is real-world project, which when given an image of a dog should provide an estimate of the dog's breed. On the other hand, if given an image of a human, the program should identify the resembling dog breed. The goal of this project is to understand the challenges involved in "piecing together various Machine Learning models" [2] in order to perform several tasks in a Data process Pipeline" [3].

### Metrics

In this project, I have used the "accuracy metric" on the test images to determine the efficiency of our Deep Learning Model, for both the scratch CNN model as well as the transfer learning model.

$$\textbf{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

**TP:** True Positives
**TN:** True Negatives
**FP:** False Positives
**FN:** False Negatives

"As a rule of thumb, accuracy can tell us whether a model is being trained correctly and how it performs generally" [4]. Accuracy, here, determines how many predictions have been made correctly by a model. However, it may not always be a reliable indicator of how good a model is, especially if the data is skewed. But, in this project, since our data is not so much imbalances, I will be using the Accuracy metric.

# II. Analysis

## Data Exploration

In this Project, we have been supplied with two datasets by Udacity-

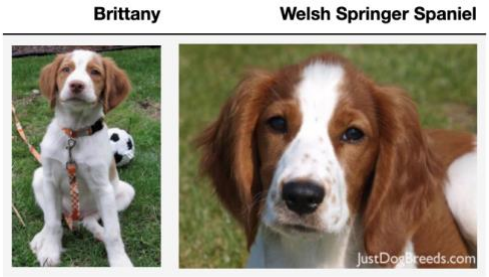
a) Dog Dataset: This dataset contains (.jpeg) images of 133 different breeds of dogs, around the world.
b) Human Dataset: This dataset contains images with many different human faces, with a view to train our model to identify a dog breed that closely resembles that human face.

While exploring these two datasets, I have found that,

- There is a total of 8,351 dog images.
- Of the total dog images, 80% (i.e. 6,680) images are for training the model
- 10% of the remaining dog images i.e. (10% of 1,671) around 835 images for validation
- Remaining 10% of the total dog images (i.e. 836 images) for testing our model
- There is a total of 13,233 human images

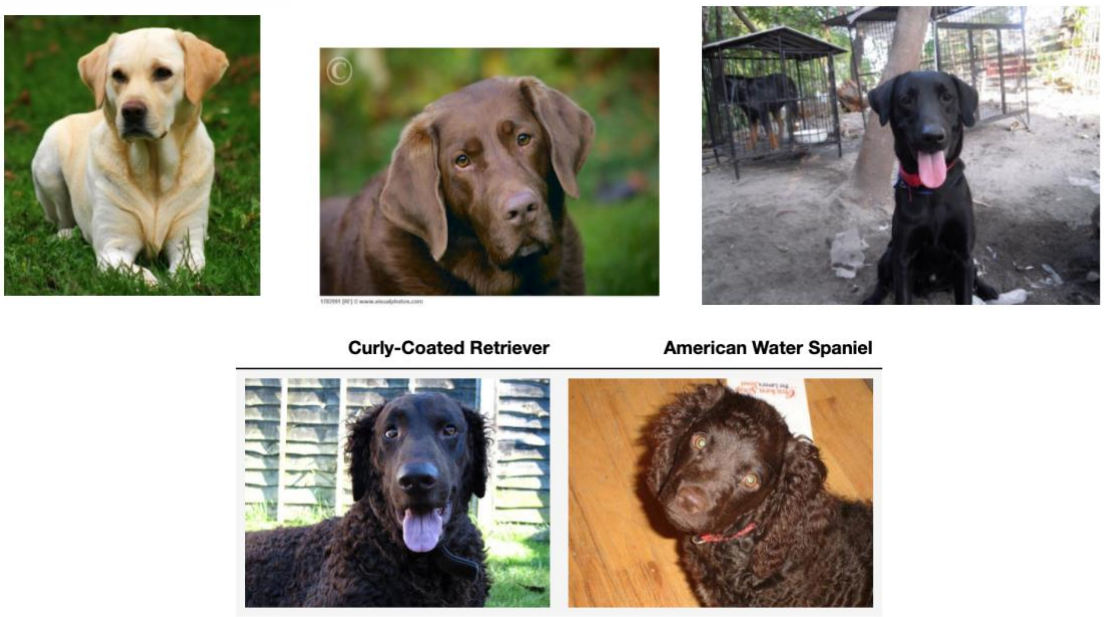
## Data Visualization

Upon visualizing the dataset, I observed how subtle the different breeds of dog are. Some of the breeds are so similar that even most humans will fail to identify them correctly, let alone a computer. Looking below, we can observe these two breeds of dogs. They almost look identical, so it might be difficult for the deep learning model to differentiate between them also.




Also, another such example with subtle inter-class variation would be the breed of Labrador retrievers. The dogs belonging to this breed come in so many different shades. For example,

yellow, chocolate, or black Labradors; or Siberian husky and the Alaskan Malamute, or the curly-coated retriever and the American Water Spaniel, etc.

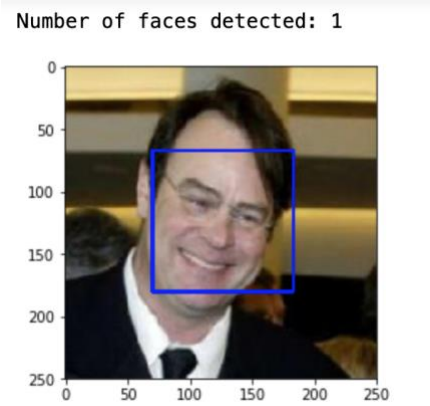



# Benchmark model

In order to effectively tackle the given problem, I will be utilizing several different Machine Learning models and techniques at hand, for different phases of the project.
For the dog detector, I will be training a VGG-16 model with pretrained weights on ImageNet dataset.
For the scratch CNN model, I will base my model on VGG-16 because it is one of the most commonly used models to train on ImageNet datasets. As per Udacity's guidelines, I need to cross the benchmark of at least 10% test accuracy.
For the "transfer learning" CNN model [5], I propose to use ResNet-152 model to train the dog images because unlike VGG, this does not hugely increase the training time when the number of layers for the neural network are increased. Resnet works on the principle of "Identity shortcut connection" [6] and skips 1 or more layers, to prevent overfitting and thereby, decreasing training time. For the transfer learning, I need to achieve a test accuracy of at least 60% to match Udacity's benchmark.

## Detecting Human Faces

For this particular algorithm, I have incorporated OpenCV's implementation of "Haar-Cascade Classifiers" [7], which is open source and available for everyone to use. OpenCV is an open source computer vision library with numerous "programming functions aimed at real-time computer vision" [8] tasks. This is the first part of our data pipeline.




This algorithm uses Haar-Cascade feature of the OpenCV library to correctly identify a human face and construct a rectangle around the face. “Haar-Cascade” is one of the machine learning algorithms “used to identify objects in an image or a video”. [9]
The face detection function, in turn, takes the path of an image as the parameter and returns (zero = 0) True, if a human face is detected otherwise returns False that is a 1.

## Detecting Dogs

In order to tackle this hurdle, I used a pre-trained CNN model, which detects the images of the dogs. I used the VGG-16 model with all the weights that are trained on ImageNet dataset. ImageNet is a “very large, popular dataset used for image classification” [10] and many other tasks involving computer vision.

This model takes in an image and gives a prediction that is “derived from 1000 possible categories in ImageNet” [11], given an object in the image.

The VGG16_predict method takes in the path of an image as a parameter. A bunch of transformations are performed on the image which are “chained together using the Compose method” [12]. Following which, the image is normalized and converted into a tensor. This is a necessary step because it helps in changing the image data “within a specific range and reducing the skewing of the data” [13] which decreases the training time for the model. Finally, the transformed image is passed through the VGG-16 model which returns an integer index in Range 0-999 that corresponds to the class in ImageNet dataset as predicted by the model.

## Writing the Dog Detector

This Dog Detector method takes in the path of an image and call in the predictor method (VGG16_predict), which returns an integer index. The function checks if the index lies between 150 and 268 and returns “True”, which means that a dog has been detected by the model.
Our Pre-trained dog detector model is then assessed by passing in the 100 image files from each of the “human_files” and “dog_files” dataset, for which the following predictions have been made by the model-

```
percentage of the images in human_files that have detected a dog:  0.0 %
percentage of the images in dog_files that have detected a dog:  100.0 %
```

# III. Methodology

## Creating a CNN to Classify Dog Breeds (from Scratch)

After having defined the face detectors for humans and the Dog detector for detected dogs in an image, I created my own Convolutional neural network from scratch. Since, my dog detector algorithm was designed on the VGG-16 model, I decided to base my CNN on the VGG model. VGG-16 is one of the most commonly used CNN architectures and I have further simplified the architecture to solve the problem at hand. The CNN that I created detects dogs in an image and classifies them according to the various dog breed that are in our dataset. Although Udacity's requirement for this scratch CNN was to "attain at least 10% test accuracy" my model was able to achieve 15% accuracy.

## Preprocessing the Data

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

train_dataset = datasets.ImageFolder(train_path,  transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(15),
            transforms.ToTensor(),
            normalize,
        ]))

val_dataset = datasets.ImageFolder(val_path,  transforms.Compose([
            transforms.Resize(size=(224,224)),
            transforms.ToTensor(),
            normalize,
        ]))

test_dataset = datasets.ImageFolder(test_path,  transforms.Compose([
             transforms.Resize(size=(224,224)),
            transforms.ToTensor(),
            normalize,
        ]))

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size= batch_size, num_workers = num_workers,
                                           shuffle = True)

val_loader = torch.utils.data.DataLoader(val_dataset, batch_size= batch_size, num_workers = num_workers)

test_loader = torch.utils.data.DataLoader(test_dataset, batch_size= batch_size, num_workers = num_workers)
```

My code resizes the image by performing "RandomResizedCrop" method which crops the image to 224 x 224 px which is the standard format for vgg-16 CNN architecture. I performed augmentation on the dataset by performing "RandomHorizontalFlip" and "RandomRotation" methods to randomize the data. This is however, only done on the training dataset. For the validation and testing dataset I have resized the images to 224 x 224 instead.

## Model Architecture

```
# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(28*28*128, 512)
        self.fc2 = nn.Linear(512, 133)
        self.dropout = nn.Dropout(0.25)
        self.batch_norm = nn.BatchNorm1d(512)

    def forward(self, x):

        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 = x.view(-1, 28*28*128)

        x = F.relu(self.batch_norm(self.fc1(x)))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        return x
```

I have chosen 3 convolution layers followed by max pooling at each step to decrease the size of the image by a factor of 2, till the size of the image comes to 28 x 28, which is then fed into the deep fully connected layers. Furthermore, I have added dropout regularization to reduce overfitting and batch-normalization to normalize all the features and speed up the training.

The first conv2d layer stars with an input image of 3 x 224 x 224 (for 3 RGB colors), with a kernel size of 3 and padding of 1 for 3x3 conv layer followed by MaxPool2d(2,2) which decreases the size of the image. The output of this layer is fed into the input of the second conv. layer and again Max-pooling is performed which further decreases the size of the image, which is finally fed into the third conv. layer and max-pooling further decreases the size of the image to 28 x 28 x 128. -This image is then input to a fully connected layer where ReLU activation is performed, and the image features are normalized (Batch Normalization) to increase the training speed and some features are dropped out (0.25) to prefect overfitting. The output is then fed into a second fully connected dense layer which decreases the nodes from 512 to 133 since our given problem has 133 breeds of dog.

This version of my scratch CNN is a little different from standard VGG model because I made my last layer as Fully-connected instead of a “softmax” layer because, later in the training, I have used “CrossEntropyLoss” function to calculate the loss for the model which inherently performs a “log-softmax” function.

## Optimizing the model and Specifying Data Loss

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

### TODO: select optimizer
optimizer_scratch = optim.SGD(model_scratch.parameters(), lr=0.02)
```

## Training the model

I trained my scratch CNN model for 15 epochs with a batch size of 20 and saved the state of the model for whichever epoch the validation-loss decreased.

```
# train the model
model_scratch = train(15, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')

# load the model that got the best validation accuracy
model_scratch.load_state_dict(torch.load('model_scratch.pt'))
```

```
Epoch: 1        Training Loss: 4.232032         Validation Loss: 4.269612
Validation loss decreased (inf --> 4.269612).  Saving model ...
Epoch: 2        Training Loss: 4.172868         Validation Loss: 4.030271
Validation loss decreased (4.269612 --> 4.030271).  Saving model ...
Epoch: 3        Training Loss: 4.120142         Validation Loss: 3.987528
Validation loss decreased (4.030271 --> 3.987528).  Saving model ...
Epoch: 4        Training Loss: 4.026708         Validation Loss: 4.133102
Epoch: 5        Training Loss: 3.996762         Validation Loss: 3.963289
Validation loss decreased (3.987528 --> 3.963289).  Saving model ...
Epoch: 6        Training Loss: 3.951417         Validation Loss: 4.079301
Epoch: 7        Training Loss: 3.878445         Validation Loss: 3.828208
Validation loss decreased (3.963289 --> 3.828208).  Saving model ...
Epoch: 8        Training Loss: 3.854692         Validation Loss: 3.911236
Epoch: 9        Training Loss: 3.791086         Validation Loss: 3.709005
Validation loss decreased (3.828208 --> 3.709005).  Saving model ...
Epoch: 10       Training Loss: 3.725961         Validation Loss: 3.809441
Epoch: 11       Training Loss: 3.698325         Validation Loss: 3.696568
Validation loss decreased (3.709005 --> 3.696568).  Saving model ...
Epoch: 12       Training Loss: 3.649747         Validation Loss: 3.718037
Epoch: 13       Training Loss: 3.603442         Validation Loss: 3.804966
Epoch: 14       Training Loss: 3.577233         Validation Loss: 3.714612
Epoch: 15       Training Loss: 3.534542         Validation Loss: 3.746774
```

## Testing the model

```
Test Loss: 3.691103


Test Accuracy: 15% (126/836)
```

Upon testing the accuracy of my scratch model, I passed all the dog images that were stored in the test dataset, into the model and achieved an accuracy of 15% which passed Udacity's benchmark.

## Creating a CNN using Transfer Learning

To further increase the efficiency of our model, we will use transfer learning on our scratch CNN model. Transfer learning is a process in deep learning, “where a model developed for a task is reused as the starting point for a model on a second task” [5]. For this task, I decided to use the ResNet-152 pretrained model because, unlike VGG, this does not hugely increase the training time when the number of layers for the neural network are increased. Resnet works on the principle of "Identity shortcut connection" [14] and skips 1 or more layers, to prevent overfitting and thereby, decreasing training time.

For transfer learning, since we are only concerned with the last layer of our network, I deactivated all the initial layers chose the second fully connected layer and fed it 2048 channels and output of 133 (i.e. the number of dog breeds)

## Optimizing, Specifying Data Loss and Training

```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer.fc.parameters(), lr=0.01)
```

To optimize this transfer learning model, I used “CrossEntropyLoss” and SGD optimizer to increase the efficiency of the training, running it for 15 epochs, with batch size 20.

```
Validation loss decreased (0.537860 --> 0.515074).  Saving model ...
Epoch: 13       Training Loss: 1.050583         Validation Loss: 0.507841
Validation loss decreased (0.515074 --> 0.507841).  Saving model ...
Epoch: 14       Training Loss: 1.033682         Validation Loss: 0.500831
Validation loss decreased (0.507841 --> 0.500831).  Saving model ...
Epoch: 15       Training Loss: 1.001018         Validation Loss: 0.466092
Validation loss decreased (0.500831 --> 0.466092).  Saving model ...
```

## Testing the model

Optimize on testing this transfer learning CNN model, I was able to achieve a testing accuracy of 86%. Hitting 26% over Udacity’s minimum criteria of 60% and increasing the overall efficacy from initial test accuracy of 15% of my scratch CNN model.

**(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%.

```
test(transfer_loaders, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.521917


Test Accuracy: 86% (722/836)
```

## Predicting the final model

After training the scratch model and achieving a higher test accuracy, now, it's time to predict our new model. The "predict_breed_transfer" function will take in the path of an image as the parameter and perform various transformations like- Resizing, Cropping, and converting the image to a tensor, and normalize the image data. After these feature extractions, the method will send the transformed tensor through the transfer-learning model (ResNet-152) and eventually return a predicted dog breed like – "Affenpinscher, Afghan hound, Labrador", etc.

## Writing the Algorithm for the Dog Breed Classifier

Finally, it is time to take our model for a spin! This algorithm will be apt at detecting whether there is a human present in the image or a dog. Furthermore, if a dog is detected it will go on predicting the appropriate breed for that dog. On the other hand, if a human face is detected, it will predict a dog breed that closely resembles the human's face. However, if neither a human nor a dog is detected by the model, it will classify the object as neither.

```
### 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('Human Face Detected!!')
        plt.imshow(Image.open(img_path))
        plt.show()

        print(f'This human looks like a ... {predict_breed_transfer(img_path)}')
        print('\n----------------------------------\n')

    elif dog_detector(img_path):
        plt.imshow(Image.open(img_path))
        plt.show()

        print(f'Analysing the Breed..... looks like a ... {predict_breed_transfer(img_path)}')
        print('\n----------------------------------\n')

    else:
        plt.imshow(Image.open(img_path))
        plt.show()
        print('Neither a human face or a dog detected in this image.')
        print('\n----------------------------------\n')
```

# IV. Results

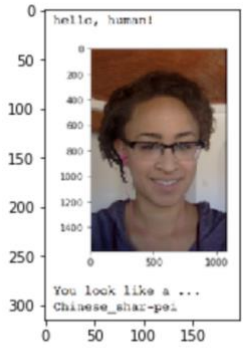
## Testing the Algorithm

In this section, I will be evaluating and validating my Deep Learning model and the breed classifier algorithm as a whole. My model has exceeded Udacity's benchmark of 10% accuracy for scratch CNN by giving a test accuracy of 15% and for the Transfer-learning model, it has produced 86% test accuracy which is 26% higher than Udacity's minimum requirement (60%). The model has been trained on ResNet-152 CNN and uses "CrossEntropyLoss" and SGD optimizer to increase the efficiency of the training, running it for 15 epochs, with batch size 20 and a learning rate of 0.01. I was able to achieve a testing accuracy of 86%.

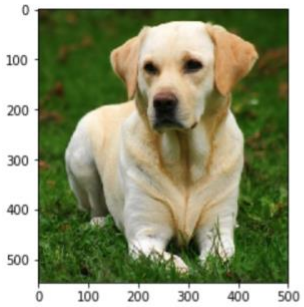
```
for img_file in os.listdir('./images'):
    img_path = os.path.join('./images', img_file)
    run_app(img_path)
```

These images were provided in the workspace already for testing purposes. These contain a mix of human faces as well as different breeds of dogs.

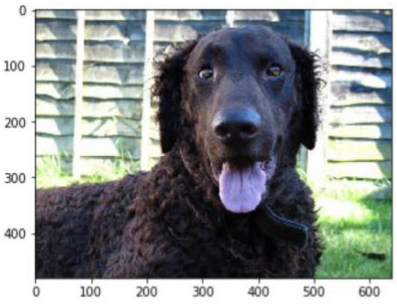
```
Human Face Detected!!
```



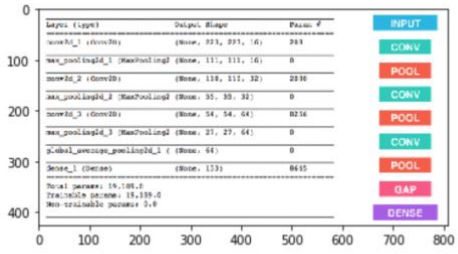

```
This human looks like a ... Yorkshire terrier
```



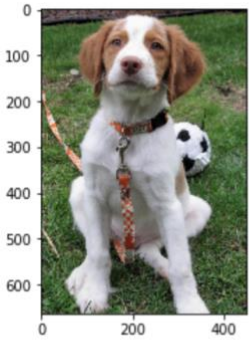
```
Analysing the Breed..... looks like a ... Labrador retriever
```



```
Analysing the Breed..... looks like a ... Curly-coated retriever
```



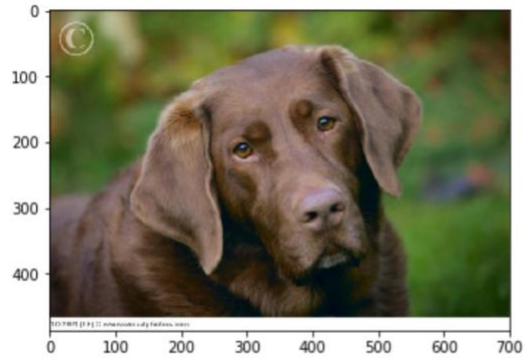

Neither a human face or a dog detected in this image.



Analysing the Breed..... looks like a ... Brittany

Analysing the Breed..... looks like a ... Labrador retriever

Analysing the Breed..... looks like a ... Boykin spaniel



```
Analysing the Breed..... looks like a ... Labrador retriever
```

## Checking Robustness of the Model

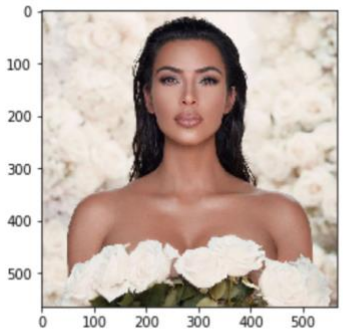
Trying to do more vigourous testing by passing some images from the internet and my personal dog, “Pluto’s” picture.

```
for img_file in os.listdir('./test_images'):
    img_path = os.path.join('./test_images', img_file)
    run_app(img_path)
```

```
Analysing the Breed..... looks like a ... Bulldog
Human Face Detected!!
```

```
This human looks like a ... Afghan hound
```

```
Human Face Detected!!
```



```
This human looks like a ... Cocker spaniel
```

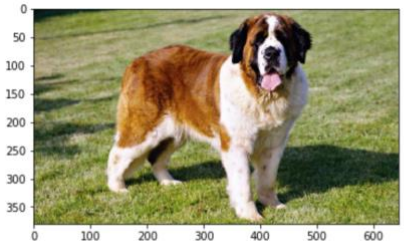
```
Analysing the Breed..... looks like a ... Golden retriever
```

```
Analysing the Breed..... looks like a ... Beauceron
```

PLUTO (my dog)

```
Human Face Detected!!
```

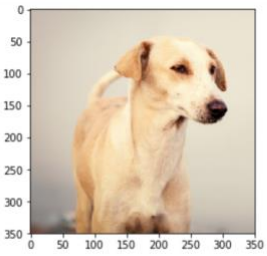
```
This human looks like a ... Bullmastiff
```



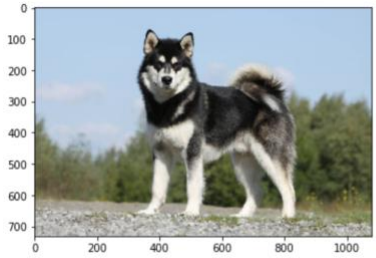
```
Analysing the Breed..... looks like a ... Saint bernard
```

```
Human Face Detected!!
```

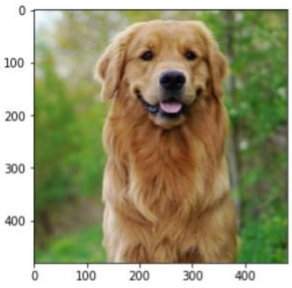
```
This human looks like a ... Dachshund
```



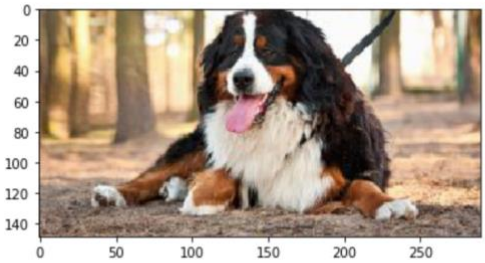
```
Analysing the Breed..... looks like a ... Greyhound
```



```
Analysing the Breed..... looks like a ... Alaskan malamute
```

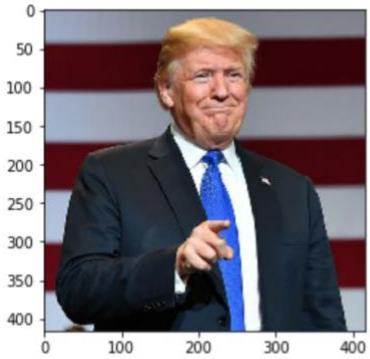


```
Analysing the Breed..... looks like a ... Golden retriever
```



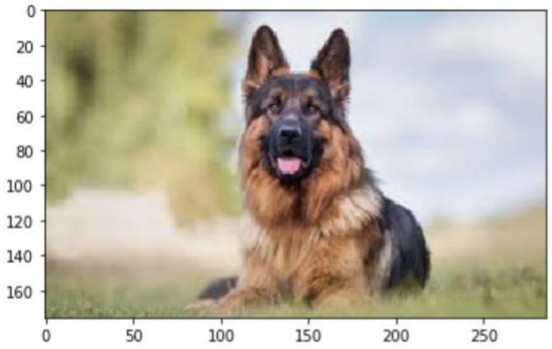
```
Analysing the Breed..... looks like a ... Bernese mountain dog
```

```
Human Face Detected!!
```



```
This human looks like a ... English cocker spaniel
```

```
Analysing the Breed..... looks like a ... Doberman pinscher
```

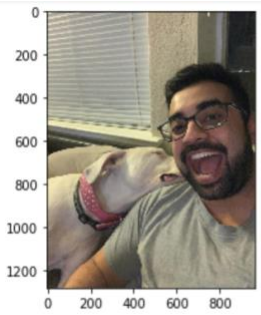


```
Analysing the Breed..... looks like a ... German shepherd dog
```

# V. Conclusion

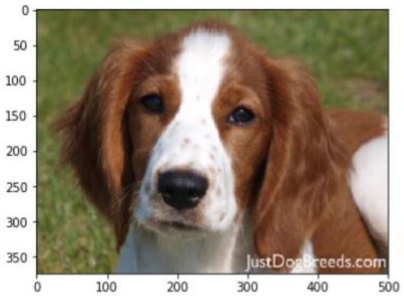
These days facial recognition and computer vision tasks, involving object detection through images, is gaining a lot of popularity. This project was based on one such real world application. Here, my objective was to design my own deep learning model which could successlfy differentiate between a human and a dog, given a picture.

I was really happy to see the results of the dog breed predictor model using Resnet-152 CNN. The model didn't require much tuning and provided really high test accuracy of 86% compared to Udacity's benchmark requirement of 60% test accuracy. For many of the dog images the progaram was able to correctly identify the appropriate dog breed. However, as with all the application, this model also have some flaws, after all it has ony 86% test accuracy, there is still a 14% chance that it will make a wrong prediction, which is a bit high to be honest. While testing the model, I was able to observe some of these flaws and outliers that the model failed to account for –




```
Analysing the Breed..... looks like a ... Italian greyhound
```

For instance, this was an outlier that the model couldn't tacke effectively. Here is a picture of me with one of my friends' dog (Dogo Argetino). The model had to tackle an extremely difficult task of having to defferentiate between a dog and a human in the same image, as it has only been trained so far on images that had dogs and humans separately. Thus, I guess, the model got confused decided to ignore my presence and also wrongly predicted the dog's breed.




```
Analysing the Breed..... looks like a ... Irish red and white setter
```

Another instance, where the model loses accuray is differentiating breeds with minimal differences. There are several dog breeds that look so much similar that even most humans fail to

distinguish them. Similarly, this ResNet-152 model was also not able to correctly distinguish this dog's breed (picture above) which is actually supposed to be a "Welsh Spring Spaniel", but looks very similar to the breed predicted by the model.

On the other hand, analysing the predictions made for humans resembling certain dog breeds, I doubt how much accurate they are. Since, being a human, I can't say if someone resembles a dog or not. Also, there is no metric availabe that could validate whether a particular human actually resembles a certain dog breed or if the model is just guessing.

This model's test accuracy can further be increased to 90% and above by-

- Increasing the training dataset, i.e. training the model with more images of will further increase the accuracy of prediction on the test data.

- Increasing the accuracy of the face_detector algorithm by using some other deep learning architecture like VGG-16 or ResNet.

- Fine tuning the training hyperarameters of our Scratch CNN model like- number of epochs, dropout regularization, batch-size, learning rate, etc. will also improve the overall accuracy of our predictions.

- Increasing the number of dog breeds. Only 133 breeds are provided in the dataset, on the contrary around 350 breeds are present in the world, including mixed-breed dogs.

In near future, I would like to further develop this machine learning model into a mobile application and deploy it on the app store. I could not add these amendmends yet because of time constraints and budget issues for deployement.

# References

[1] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way; Saha, Sumit; December, 2018; **https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53**

[2] [3] Dog Breed Classifier; Machine Learning Engineer, Udacity; May 2020; https://classroom.udacity.com/nanodegrees/nd009t/parts/2f120d8a-e90a-4bc0-9f4e-43c71c504879/modules/2c37ba18-d9dc-4a94-abb9-066216ccace1/lessons/4f0118c0-20fc-482a-81d6-b27507355985/concepts/65160313-7054-4ffb-8263-793e2a166d69

**[4]** Evaluation Metrics for Machine Learning; Nicholson, Chris; 2019; https://pathmind.com/wiki/accuracy-precision-recall-f1

[5] A Gentle Introduction to Transfer learning for Deep Learning; Deep Learning For Computer Vision; Brownlee, Jason; September, 2019; https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[6] Residual Blocks- Building Blocks of ResNet; Sahoo, Sabyasachi; November, 2018; https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec

[7] Face Detection Using Haar Cascades; Mordvintsev, Alexander & K. Abid; 2013; Revision 43532856; https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

[8] Wikipedia; https://en.wikipedia.org/wiki/OpenCV

[9] Deep Learning with Haar Cascade Explained; Berger, Will; http://www.willberger.org/cascade-haar-explained/

[10] [11] Dog Breed Classifier; Machine Learning Engineer, Udacity; May 2020; https://classroom.udacity.com/nanodegrees/nd009t/parts/2f120d8a-e90a-4bc0-9f4e-43c71c504879/modules/2c37ba18-d9dc-4a94-abb9-066216ccace1/lessons/4f0118c0-20fc-482a-81d6-b27507355985/concepts/300e50a9-8e85-448e-928e-c10dc64cdf7e

[12] Torchvision. Transforms; Pytorch; Torch Contributors; 2019; https://pytorch.org/docs/stable/torchvision/transforms.html

[13] Understanding transform.Normalize(); Pytorch; Bhushans23; October 2018; https://discuss.pytorch.org/t/understanding-transform-normalize/21730/2

[14] Residual Blocks- Building Blocks of ResNet; Sahoo, Sabyasachi; November, 2018; https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec