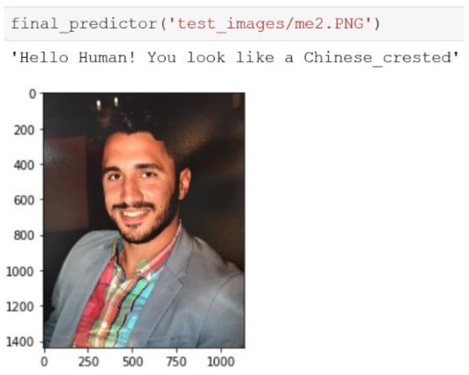
Project Overview

The goal of this project is to accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling.

Example:



```
final_predictor('test_images/me2.PNG')
'Hello Human! You look like a Chinese_crested'
```

Import Datasets

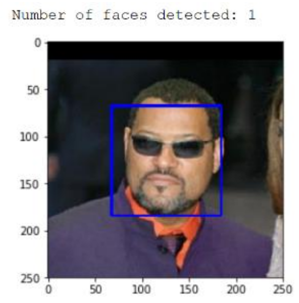
The Dog Dataset consisted of 8351 dog images and 133 dog categories, while the human Dataset consisted of 13233 images. The dog dataset was split into a training set (6680), a validation set (835) and testing set (836).

Detect Humans

I used OpenCV's implementation of “Haar feature-based cascade classifiers” to detect human faces in images. OpenCV provides many pre-trained face detectors, stored as XML files on GitHub. I downloaded one of these detectors and stored it in the haarcascades directory.

The human images first must be converted to grayscale (to reduce dimensionality from color images) prior to being assigned to the “faces” variable using the **detectMultiScale** function. Faces is now a NumPy array of detected faces, where each row corresponds to a detected face. Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. The first two entries in the array specify the horizontal and vertical positions of the top left corner of the bounding box. The last two entries in the array specify the width and height of the box.

Example:



After creating a function to detect human faces I tested it against 100 human and 100 images. The results were Human File Accuracy: 99% and Dog File Accuracy: 12%

Detect Dogs & Data Preprocessing

I used a pre-trained ResNet-50 model to detect dogs in images along with weights that have been trained on ImageNet however, When using TensorFlow as backend, Keras CNNs require a 4D array (a 4D tensor) as input, with shape (nb_samples, rows, columns, channels), where nb_samples corresponds to the total number of images (or samples), rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

In order to prepare the file paths, I created a function which takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224x224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since I was working with color images, each image has three channels. The overall shape in this case is (nb_samples, 224, 224, 3) where nb_samples corresponds to the number of 3D tensors in the dataset.

The final Preprocessing part to getting the 4D tensor ready which is also common in pre-trained models, is the additional normalization step where the mean pixel must be subtracted from every pixel in each image.

ResNet-50 Predictions

When predicting we essentially return an array who’s each entry is the model's predicted probability that the image belongs to the specific ImageNet category. Additionally, I take the argmax of the predicted probability vector, we obtain an integer corresponding to the model's predicted object class, which we can identify with an object category using a dictionary.

Percentage of Dogs in Human files: 1%, Percentage of Dogs in Dog files: 100%

Create a CNN to Classify Dog Breeds (from Scratch)

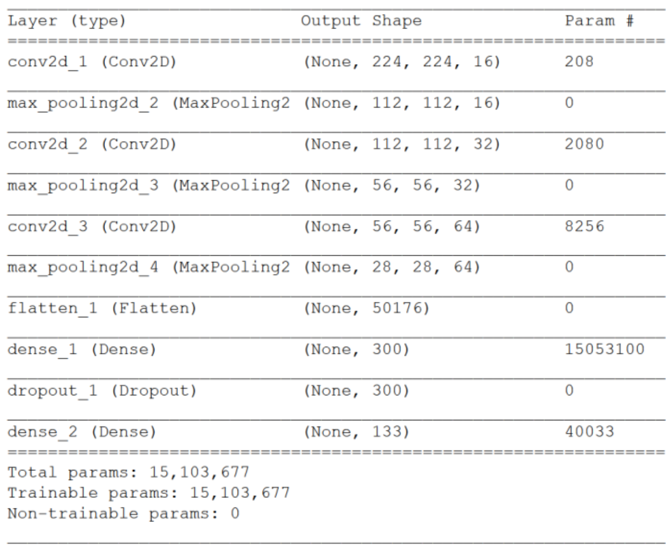
Random Chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imbalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1% (feeling confident).

Model Architecture

1. I began by creating Convoluted layers followed by MaxPooling layers.

2. I made sure the pooling size on the MaxPooling layers is the same as the kernel size used in the Convoluted Layers
    - the max pooling layers are used after each convoluted layer in order to reduce the dimensionality as the size of the filters is increasing in each convoluted layer.
    - if a greater number of filters and layers was used then the Global Average Pooling layer would have been applied.
3. The Flatten Layer is used once there is no more information left to discover. It transforms the array to a vector in order to add fully connected layers. These layers allow for the object to be detected.
4. The number of Dense nodes is arbitrary
5. The Dropout layer will prevent overfitting as well as train the nodes whose weights are not impactful.
6. The final layer has 133 nodes (the number of breeds we are trying to classify) and it's using the probabilistic 'SoftMax' activation function which solidifies that our Probabilities add up to 1.

Note: The Relu activation function is used throughout each layer besides the final one in order to assist with the vanishing gradient problem.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 300)               15053100
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               40033
=================================================================
Total params: 15,103,677
Trainable params: 15,103,677
Non-trainable params: 0
_________________________________________________________________
```

The model was compiled using the “rmsprop” optimizer, “categorical_crossentropy” as the loss, and “accuracy” as the metric.I used model checkpointing during training to save the model that attains the best validation loss. My model Test accuracy was 7.77% (6% better than random chance) which I guess we can count it as a win?

<u>Use a CNN to Classify Dog Breeds (Transfer Learning)</u>

I chose the VGG-16 model where the last convolutional output is fed as input to the model. I only add a global average pooling layer and a fully connected layer, where the latter contains one node for each dog category and is equipped with a SoftMax.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 133)               68229
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
_________________________________________________________________
```

The VGG-16 is compiled with the same parameters as mine as well as utilizing model checkpointing. The Accuracy increased to 45%

<u>Using VGG-19</u>

For the last part of this project, I chose to use an even more powerful algorithm with pre-computed bottleneck features while applying Transfer Learning.

<u>VGG-19 Model Architecture</u>

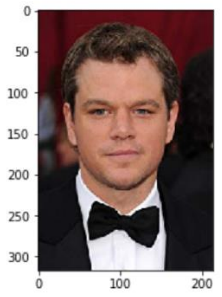
- Layer 1 - Since The dataset is relatively small and the VGG19 architecture is quite complex I added a Global Average Pooling layer as the input to prevent -overfitting.
- Layer 2 - Now all 3-D features are reduced into a vector, so I decided to use a fully connected layer due it's linear operation
- Layer 3 - Dropout layer to prevent overfitting.
- Layer 4 - The final Dense layer has 133 nodes representing all the breed types, followed with a SoftMax activation function in order to maintain the overall probability between all the breed equal to 1.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_22  (None, 512)               0
_________________________________________________________________
dense_39 (Dense)             (None, 500)               256500
_________________________________________________________________
dropout_19 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_40 (Dense)             (None, 133)               66633
=================================================================
Total params: 323,133
Trainable params: 323,133
Non-trainable params: 0
_________________________________________________________________
```

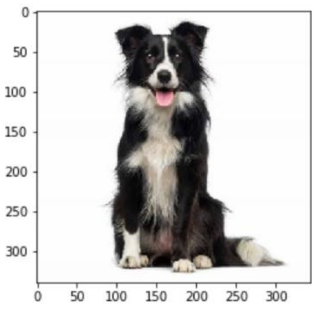
The model was compiled with loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'] and the Test Accuracy Score was 78%!!

```
'The dog breed in the image is German_shepherd_dog'
```

```
'Hello Human! You look like a Ibizan_hound'
```



```
'The dog breed in the image is Border_collie'
```



```
'The dog breed in the image is Chihuahua'
```