Capstone Project - Dog Breed Classifier
A Journey to Achieve the Best Model

Data Scientist Nanodegree
Merna Ahmed
August 1st, 2022

Project Overview

Welcome to the dog breed classifier project. This project uses Convolutional Neural Networks (CNNs)! In this project, you will learn how to build a pipeline to process real-world, user-supplied images. Given an image of a dog, your algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed.

In this real-world setting, you will need to put together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed.

Before we start our journey, let’s check our basics:

Metrics

The metric that we use to measure how well our model is performing in classifying the dog breed of images is “accuracy”. Accuracy is the percentage of total items classified correctly by our model. It can be given as:

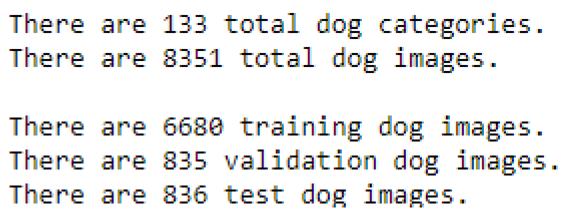
$$accuracy\ \% = \frac{total\ number\ of\ correct\ predictions}{total\ number\ of\ images}$$

The accuracy would be an appropriate metric for judging our model, as it provides a solid measure of how well our model performs and produces a number between 0 and 100 as the total number of testing images are 100.

The Dog Images Dataset

Before we start our modeling journey, let’s take a quick look regarding the main dataset:

The dog images dataset contains a total of 8351 images of dogs. These images have been split into training, testing, and validation images with a split of 80% training data, 10% testing data, and 10% validation data. Thus, there are 6680 training images, 836 testing images, and 835 validation images.



```
There are 133 total dog categories.
There are 8351 total dog images.

There are 6680 training dog images.
There are 835 validation dog images.
There are 836 test dog images.
```

Now, Let’s Start our modelling Journey with the following phases:

I. Detect (Dogs or Humans).
II. Classify dog Breeds with many approaches and test them
III. Test the final approach.

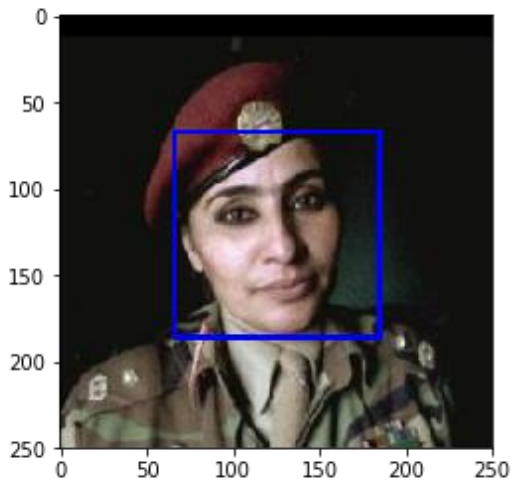
I. Detect (Dogs or Humans)

Step 1: Detect Humans

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

The main code can be found in the attached file (report.html).

we use a pre-trained classifier for our human face detector. Thus, the human dataset is not required for training.



Results of using this model:

- Percentage of correctly classified humans is 100% → **Excellent**
- Percentage of correctly classified dogs is 11% → **Need to be enhanced.**

Step 2: Detect Dogs

In this step, we use a pre-trained ResNet-50 model to detect dogs in images. So we downloaded the ResNet-50 model, along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks.

ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories. Given an image, this pre-trained ResNet-50 model returns a prediction (derived from the available categories in ImageNet) for the object that is contained in the image.

Data Processing

In this step, we are using Tensorflow as a backend so, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input with the following shape:

(nb_samples,rows,columns,channels)

where **nb_samples** corresponds to the total number of images (or samples), and **rows**, **columns**, and **channels** correspond to the number of rows, columns, and channels for each image, respectively.

Now, we are ready to use ResNet-50 to detect Dogs.

Results of using this model:

- Percentage of correctly classified humans is 0% → **we better use OpenCV model for humans**
- Percentage of correctly classified dogs is 100% → **Excellent**

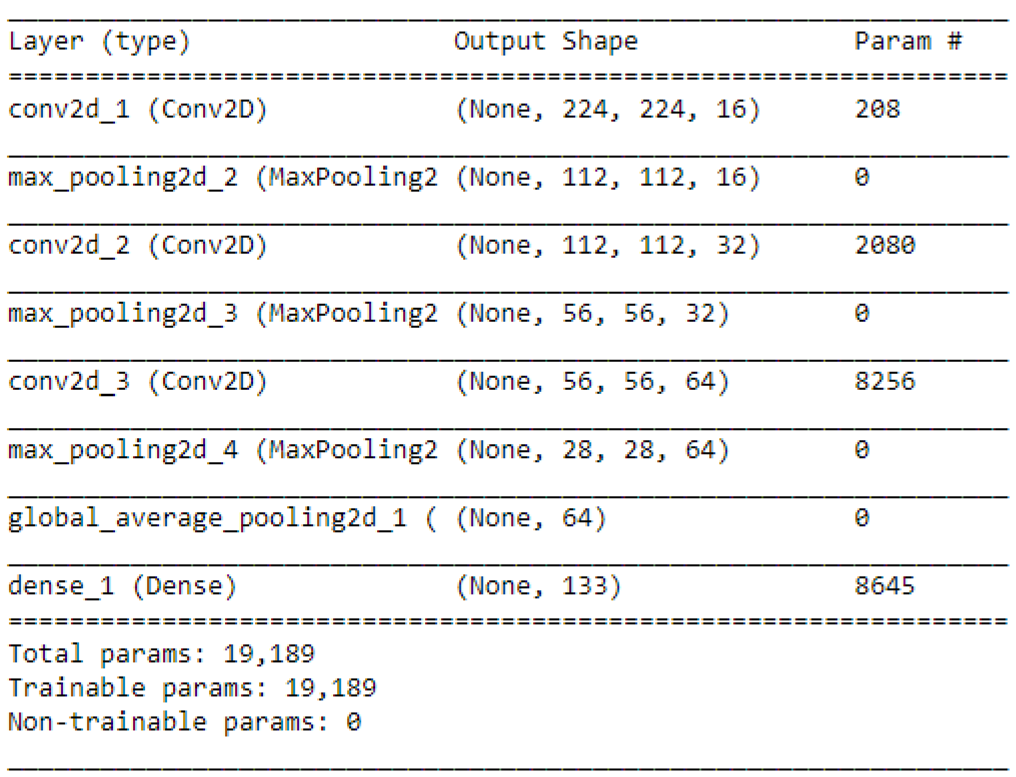
After we got a good look about Dogs/Human detectors, let’s create our own model from scratch using CNN (without transfer learning).

II. Classify dog Breeds with many approaches

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

1. Create a model with Keras Sequential API (Already provided hint).
2. Add the following layers:
    a. Convolution layer with input shape (224,224,3)
    b. Maxpooling layer with pool_size 2 and activation function "relu".
    c. Convolution layer with 32 filters, kernel size "2", and activation function "relu".
    d. Maxpooling layer with pool_size 2.
    e. Convolution layer with 64 filters, kernel size "2", and activation function "relu".
    f. Maxpooling layer with pool_size "2".
3. Global average pooling layer.
4. Final output "dense" layer with 133 output category and "softmax" activation function.

Here is the summary of the model:



```
_________________________________________________________________
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
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               8645
=================================================================
Total params: 19,189
Trainable params: 19,189
Non-trainable params: 0
_________________________________________________________________
```

Then we compile and train our model which took a very long time (you can always check the code in the attached file report.html).

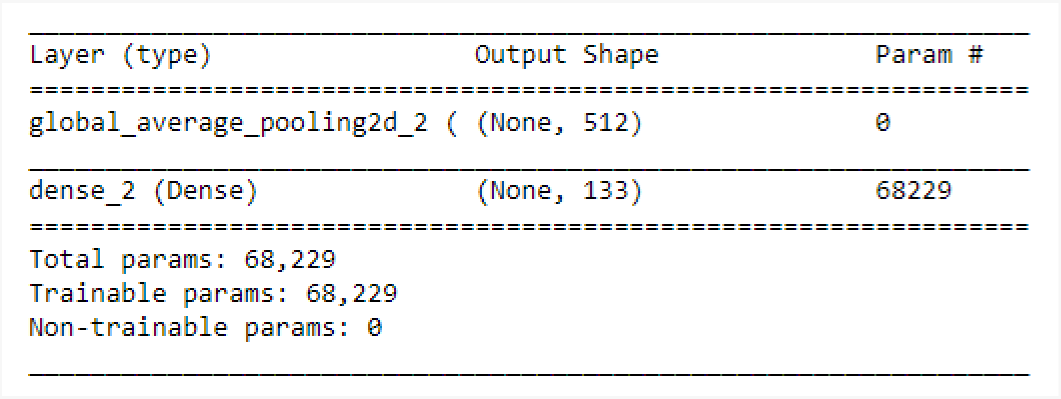
The result: Test accuracy: **4.5 %** **Very Bad Actually ☹** **so it is time to switch to transfer learning**

Step 4: Use a CNN to Classify Dog Breeds (first trial)

Here we are going to train a CNN using transfer learning.

Model Structure

Using VGG16 model but We will only add a global average pooling layer & 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_2 ( (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               68229
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
_________________________________________________________________
```

Then we compile and train our model (you can always check the code in the attached file report.html).

The result: accuracy: **42 %** → **A very good enhancement but we can do better.**

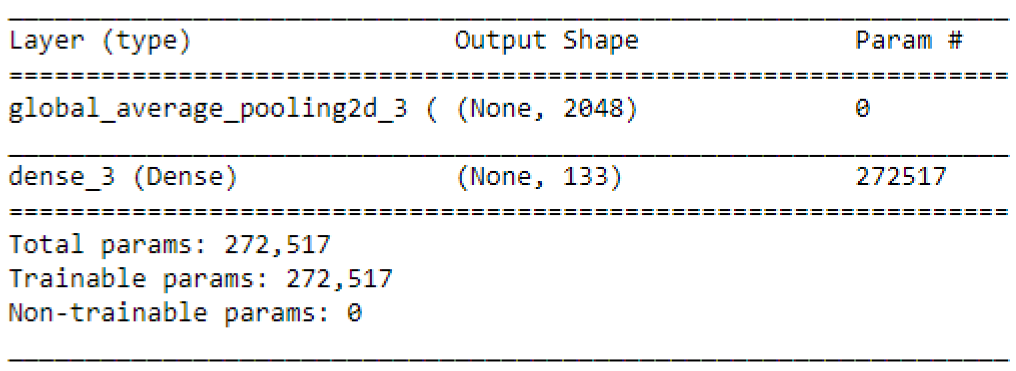
Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning) - (second trial)

Now, we will use transfer learning to create a CNN that can identify dog breed from images.

In Step 4, we used transfer learning to create a CNN using VGG-16 bottleneck features. In this step, we are going to use the bottleneck features from ResNet-50.

Model Structure

Using ResNet-50 model but We will only add a global average pooling layer & 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_3 ( (None, 2048)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 133)               272517    
=================================================================
Total params: 272,517
Trainable params: 272,517
Non-trainable params: 0
_________________________________________________________________
```

Then we compile and train our model (you can always check the code in the attached file report.html).

<u>The result:</u> accuracy: **82 %** → **Very Satisfying result.**

Now, we have excellent detectors from phase I and very good breed classifier from phase II step.5 … Now, it is time for phase III … Testing.

III. Testing the Final Approach

Let’s recap what is the needed functionality:

- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide output that indicates an error.

After combining the codes of the detectors and the classifier, we got a ML pipeline to take images as an input and give us the classification as an output.

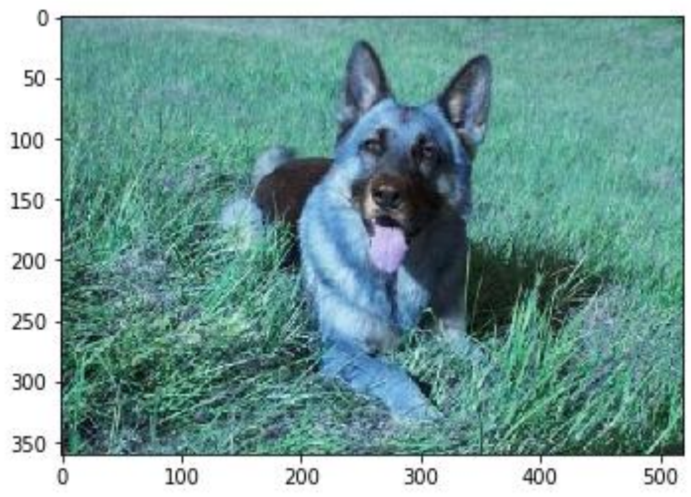
Testing steps:

1. Provide 3 Dog images.
2. Provide 3 human images.
3. Provide 3 images which do not contain neither dogs nor humans.

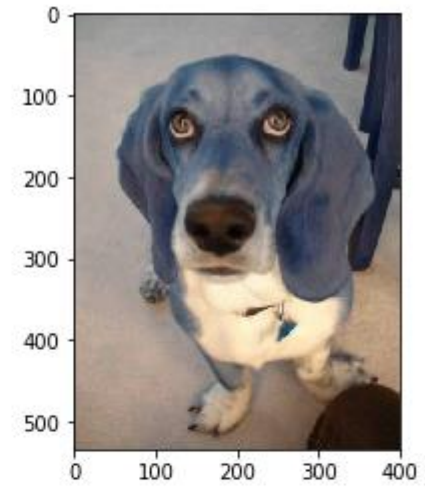
Testing Results

1. **Provide 3 Dog images.**

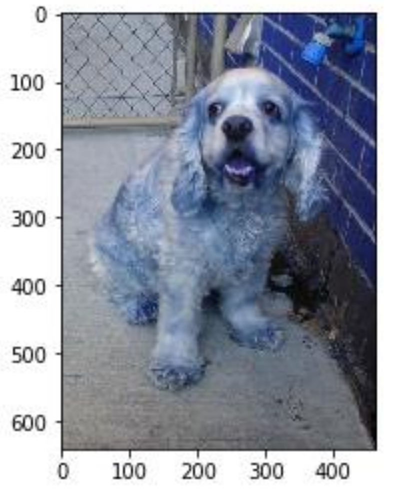
```
German_shepherd_dog
```



```
Basset_hound
```

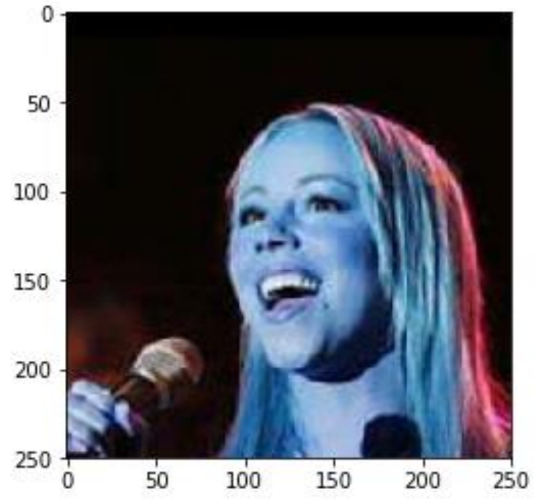


```
Cocker_spaniel
```

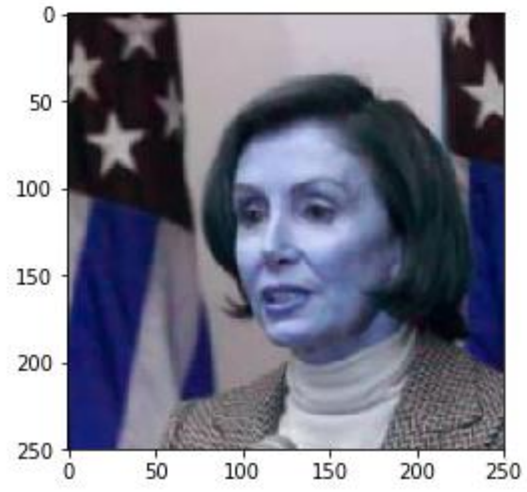


2. Provide 3 human images.

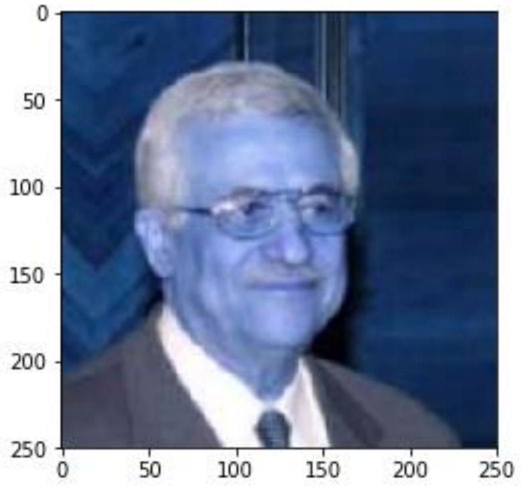
```
Bearded_collie
```



```
Silky_terrier
```

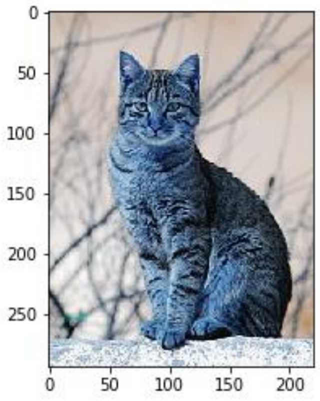


```
Beagle
```

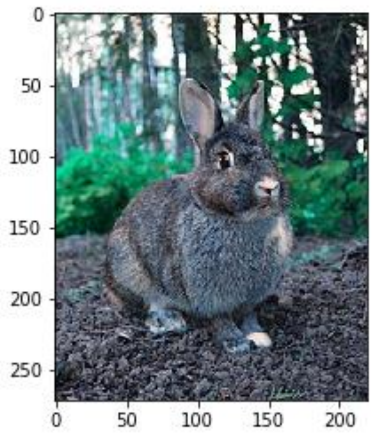


3. Provide 3 images which do not contain neither dogs nor humans.

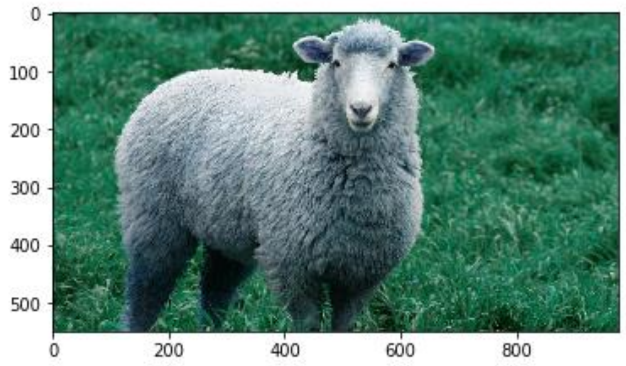
```
Error, No human or dog identified
```



```
Error, No human or dog identified
```



```
Error, No human or dog identified
```



My Conclusion

1. Using pre-trained CNNs is much better than creating your own (re-inventing the wheel).
2. ResNet-50 is one of the best computer vision models (I may need some more researches to determine what is the number 1 model in such cases).
3. Pre-processing the images is an essential step.
4. Choosing a Metric to be your base (here we have chosen the accuracy.. we can use F1 score also) is an important approach.

Possible points of improvement

1. Using data augmentation (we escaped it here (optional) but it would be a great improvement).
2. Implementing a method to detect many humans/dogs in an image.
3. Detecting humans and dogs in the same image.

Reference

https://github.com/MernaAhmedElshanawany/Dog-Breed---Capstone-Project