**Dog Breed Classification Using CNN Model**



**Project Overview**

Being a Data Scientist, Deep Learning is one of the must-to-have knowledge and skills. This project is to develop an algorithm and create a CNN (Convolutional Neural Networks) model with the two tasks as:

1. Detect the object on a provided image as a dog, a human, or neither.
2. If the object was detected as a dog, based on the given classes of the world-wide dog breeds, the model would be able to identify the dog breed.

Why Convolutional Neural Networks model?

CNN is one of the most Artificial Neural Networks as [regularized](https://en.wikipedia.org/wiki/Regularization_(mathematics)" \o "Regularization (mathematics)) versions of MLP ([Multilayer Perceptrons](https://en.wikipedia.org/wiki/Multilayer_perceptron" \o "Multilayer perceptron)). So, A standard CNN is comprised of convolutional layers, sampling layers and then followed by fully connected layers (i.e. each neuron in one layer is connected to all neurons in the next layer).

CNN uses relatively little pre-processing compared to other [image classification algorithms](https://en.wikipedia.org/wiki/Image_classification" \o "Image classification). It takes advantage of the hierarchical pattern in data and assembles more complex patterns using smaller and simpler patterns. With the help of CNN, we can use the large amount of data more effectively and accurately.

Therefore, CNN is typically applied for Image Classification from an input image with a single object and to an output as a class label from a list of object categories.

There are a few packages available o build the CNN model such as PyTorch, Keras, etc. In this project, we use Keras library for dog breed identification.

(<https://keras.io/getting-started/sequential-model-guide/>)

**Project Process**

In addition to the dog classification, we also introduce the function to identify human face to see if the image is on a human, which is not from CNN model.

Below is the flow to build the model.

*Step 0: Import Datasets*

*Step 1: Detect Human*

*Step 2: Detect Dogs*

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

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

*Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning)*

*Step 6: Write your own Algorithm*

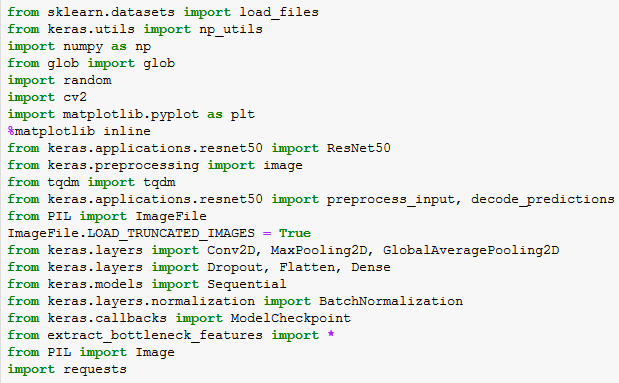
*Step 7: Test your own Algorithm*

**Metric**

In the project, we split the input data set into training and testing. We use the accuracy of the testing dataset to measure the performance of the built CNN models.

**Library Preparation**

The following package/library will be used in this project for data loading, data processing, data/image visualization, model development, etc.

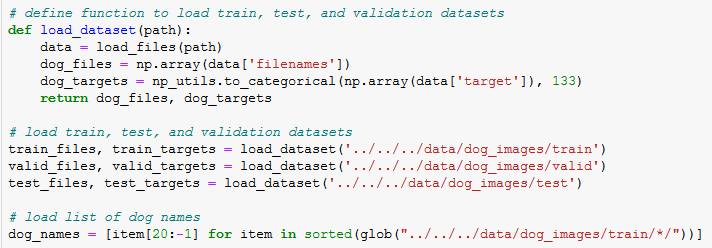


### *In the following section, the details of the project process will be explained step by step. All the corresponding codes can be found in the [github depository](https://github.com/yz14689/DSNP_Capstone).*

### **Step 0: Import Datasets**

Three set of datasets with dog images and the data of dog breed names were loaded via a pre-defined loading function:

* *train\_files, valid\_files, test\_files* - numpy arrays containing file paths to images
* *train\_targets, valid\_targets, test\_targets* - numpy arrays containing onehot-encoded classification labels
* *dog\_names* - list of string-valued dog breed names for translating labels

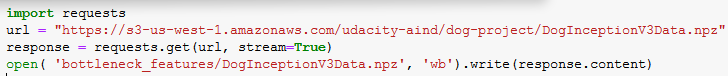


With the data, we have total 8,351 doc images (split on 80%/10%/10% for training/validation /testing, respectively) and 133 dog categories (from *dog\_names*) which will be used in final classification.

Additionally, we import a dataset of total 13,233 human images for human face detector.



Moreover, there might be some other data needed to complete this project, if not available, including the bottleneck features and the sample image for final testing.



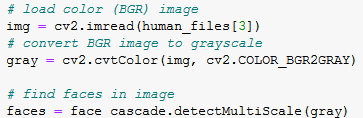


### **Step 1: Detect Human**

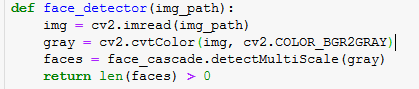
OpenCV's implementation of [Haar feature-based cascade classifiers](http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html" \t "http://localhost:8888/notebooks/Desktop/Yan/OnlineCourse/Udacity/DataScientistNarodegree/Projects/Project_6_Capstone/MyProject/DogBreed/_blank) is used to detect human faces in images. OpenCV provides many pre-trained face detectors, while we have downloaded one of these detectors for this project (*“haarcascade\_frontalface\_alt.xml”*).



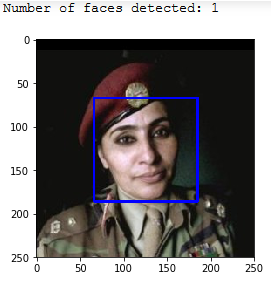
Before using any of the face detectors, the standard procedure is to convert the images to grayscale. The detectMultiScale function executes the saved classifier/detector and takes the grayscale image as a parameter.



After it, an array with four entries will be created, which specify various positions of the bounding box. Using such an arrary, we can develop a face\_detector which takes a string-valued file path to an image as input and appears in the code block below.



The result of using this face detector for a given image is as below:



Also, be testing this detector on another set of 100 images for dog and human each, we got 100% success in human images and only 11% in dog images. It implies that the fact detector works great for human face, but not for dog. 

So, what should we do to better identify dogs?

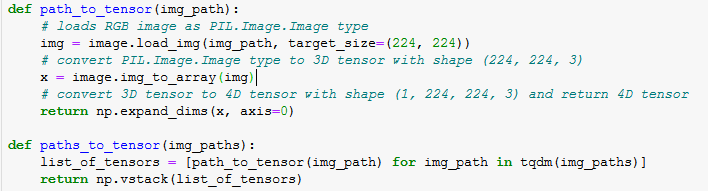
### **Step 2: Detect Dogs**

We use use one of the most popular pre-trained model, ResNet-50, to detect dog from the images. ResNet-50 was pre-trained with more than 10 million images into 1000 categories.

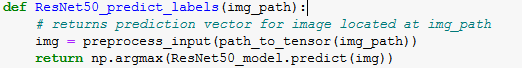


When using TensorFlow as backend, Keras CNNs require a 4D array (a.k.a.”4D tensor”) as input, corresponding to the number of rows, columns, and channels for each image.

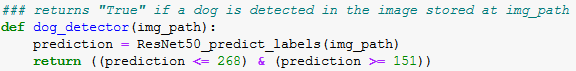
A pre-defined function, *path\_to\_tensor*, receives the file path of a color image as input and returns a 4D tensor suitable for Keras CNN. The function first loads the image and resizes it to a square image that is *224×224* pixels, and then convert the image to an array, which is then resized to a 4D tensor. In this case, the color images have three channels like *(224,224,3)* presenting the colors. This function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape (*nb\_samples,224,224,3)*, while *nb\_samples* presents the number of images.



In order to use ResNet-50, additional processing, *Normalization*, is required as well, implemented with the imported function, *preprocess\_input*.Normalization is to subtract the mean pixel calculated from all pixels in all images from every pixel in each image. With this step done, we can start using ResNet-50 to detect dogs on the image.



It is noticed that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268 at the [dictionary](https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a" \t "http://localhost:8888/notebooks/Desktop/Yan/OnlineCourse/Udacity/DataScientistNarodegree/Projects/Project_6_Capstone/MyProject/DogBreed/_blank). Thus, to check to see if an image is predicted to contain a dog by ResNet-50 model, we need only check if the *ResNet50\_predict\_labels* function returns a value between 151 and 268 (inclusive).



With this dog detector tested over the same set of 100 human images and 100 dog images, we have the dog images 100% identified correctly, but nothing for human images.



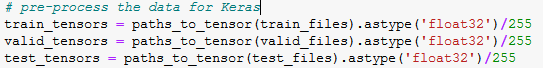
So, the pre-trained ResNet-50 model works great for detecting dog from image.

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

Now, we have the functions to identify the dog/human from the image.

Next, what we do is to classify the dog breed if a dog was detected on the image.

Pre-process data ;



Build the model architecture with mix of layerse;

Compile and train the built model;

Load model with best valiation loss;

Test model by the accuracy;



Compile and Train the model

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

(shown as example using VGG-19 model)

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

Use InceptionV3 model

Step 6: Write my own Algorithm -

Use the predefined face detector function and dog detector function

Step 7: Test my own Algorithm -

Test the algorithm wiht various images, including samples selected from internet

### **Implementation**

Metrics, algorithm, techniques

Complication of coding

Using a model with convolutional layers is effective here as the model learns from the data provided what patterns to look for when filtering the images, in this case dogs. Since the shapes that the model is seeking in the images are quite complicated, the dimensionality becomes large and there are many filters. This can lead to overfitting, therefore the max pooling layers have been implemented in order to reduce the filter dimensionality. A more extreme version of this was used in the form of the global average pooling layer. This reduces an array of filters to a vector, where each filter is represented by its average. Since the convolutional layers are only locally connected, I believe this limits the accuracy of the model. The introduction of a fully connected dense layer allows each image to be classified as any breed. Activating the final dense layer with a softmax function allows the model to output a probability of an image belonging to each breed of 133 classes in this case. This greatly improved the accuracy. Finally, I introduced dropout layers in an attempt to improve the accuracy, which did yield some improvement. This is another method of preventing overfitting, allowing nodes with smaller weights to have an impact on the output.

### **Refinement**

Metrics, algorithm, techniques

Complication of coding

## Results

**Model evaluation**

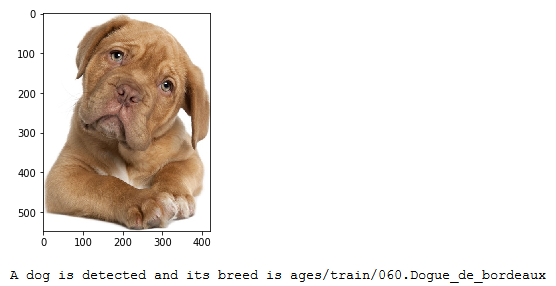
Final model parameters

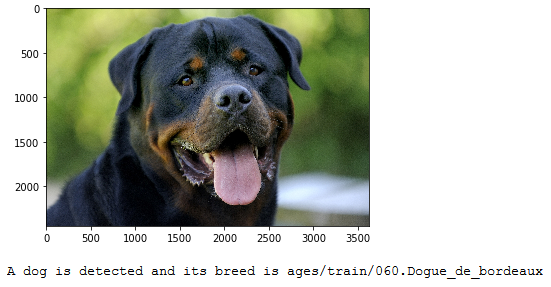
This was done using only the global average pooling layer and dense layer activated with softmax initially.

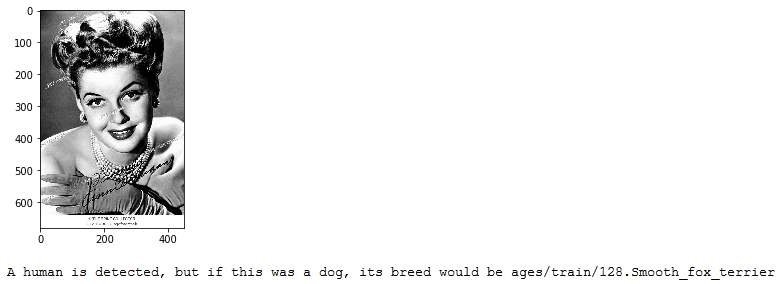
I tested the model with some additional layers as I did in my own model previously. Though in my model the greatest improvement I saw was the implementation of the softmax activation function, so I did not expect much improvement if any.

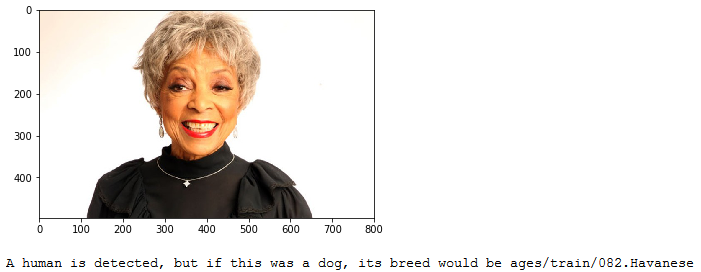
Reading the Keras documenation the different models, it appears that the only model to be better than Xception on the ImageNet validation set is InceptionResNetV2. However this has a depth of 572 compared to Xception's 126, therefore greatly increasing the runtime. I thought this was too much of a compromise for the small increase in accuracy.

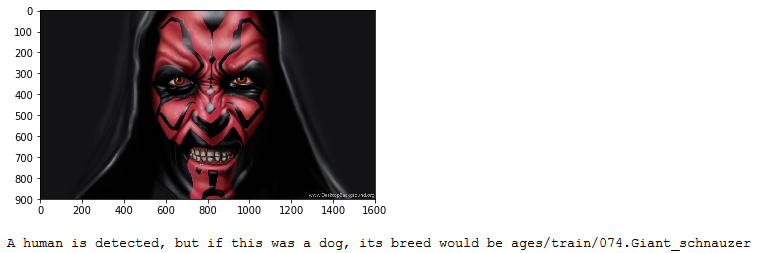


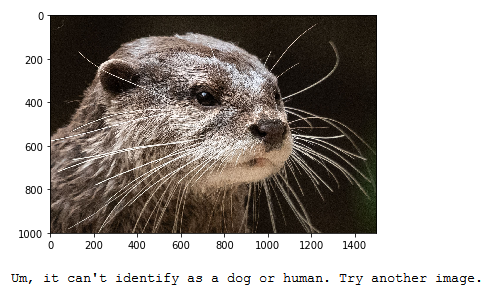


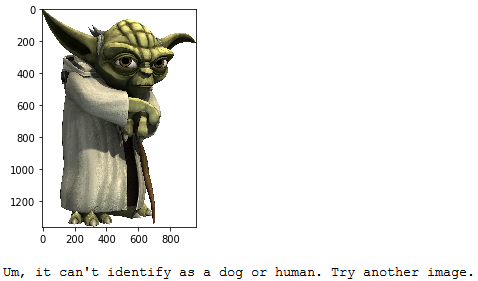












## Conclusion

Increase number of epoches

Augmentate training data (but can't be too much)

Tune some of the model parameters

Adjust the pre-processing method

Change loss function

Reduce batch size

Increase the breeds of dog and train more images

#### What I learned:

* The most important thing that I learned in this practice is that **the algorithm will just learn what you train them for**.

In this case we had just data for humans and dogs and wanted the algorithm to detect if it was not a human or thing. This was only possible by using the Imagenet Classifier for ResNet50 above, since it was trained on a whole range of images and could pick out dogs from everything else.

* **Combining algorithms helps to get better results**

Even though both bottleneck features I tried gave good result, their combined result was even better

* **Getting to Apps from models** can be achieved by combining classifiers with a different focus

The App would not have been possible to derive from a single Classifier without a huge amount of additinal data, but by combining several algorithms in a smart way, it was possible to build the app and make it reject images that were neither human or dog even though it was never trained on such images. This was acchieved by using a ready made Classifier from Keras, that had not much to do with the task at hand.

* **Transfer Learning is very powerful!**

Distinguishing the dog breeds is quite hard. I could never do this with 86% accuracy. So I am impressed how easy it was to build such a powerful classfier with transfer learning.

## Refer**ence:**

1. <https://en.wikipedia.org/wiki/Convolutional_neural_network>
2. <http://deeplearning.net/tutorial/lenet.html>
3. <http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/>