Capstone Project Report
CNN Dog Breed Classifier

Javier Martinez

1. DEFINITION

Project Overview

The field of research where this project belongs is computer vision. Computer vision could be defined as the process whereby a machine could get information about some image and transform this data into the something that is intelligible to the human. According to Wikipedia, computer vision "is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve "automatic visual understanding".

This task is very difficult, because requires and high level of abstraction that is created by the human and because of that is almost impossible to build a system than can resolve the problem with traditional programming.

It is commonly accepted that the father of Computer Vision is Larry Roberts, who discussed the possibilities of extracting 3D geometrical information from 2D perspective views of blocks (polyhedral) (Aloimonos, 1992). Since then, there have had a lot of progress, that includes change both algorithms and hardware.

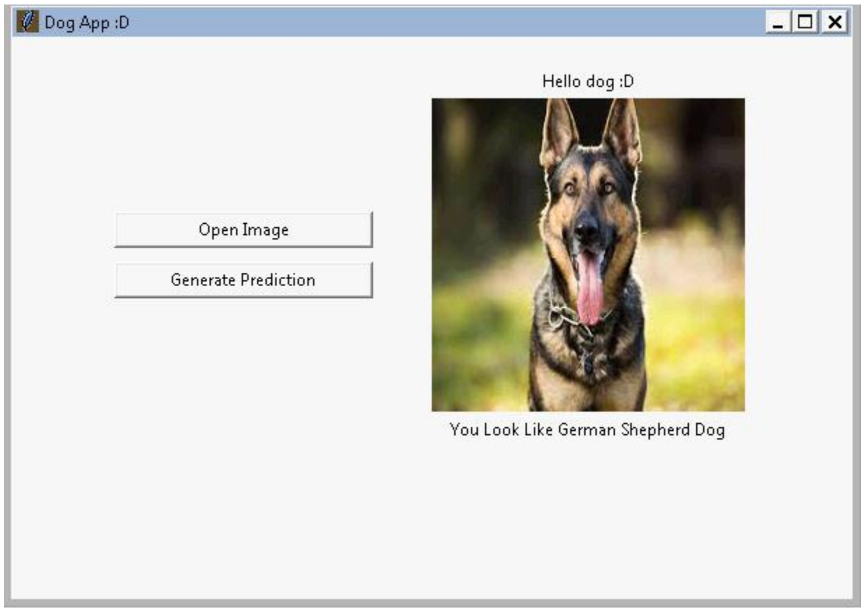
In recent time, the main developments could be resumed in two names: Convolutional Neural Network and GPUs. A convolution neural network is a special deep learning structure that allow create new features, like edges. This bring a lot of the abstraction and allow to the model adjust the parameters that are needed to create a good model. However, that requires a lot of computational power that means hours or may be moths of training. Fortunately, many of these operations can be done in parallel and with the use of GPUs1 it's possible to train a CNN in less time. (NG, 2019)

The develop of this field has allowed to improve different systems, like systems that can find images related to a search, several image recognition systems that are used to identify people, systems that can identify disease in x rays images, etc.

In particular, in this project we built an algorithm that tried to accomplish two objectives:

- First, the algorithm determines if a photo is a human, a dog or neither.
- Second it predicts what the breed of the picture is.

My final product is a tkinter app that allows selecting a picture and then generating a prediction. Below there is an image of the app



Problem Statement:

As were stated in the Udacity page the problem is “to 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.”

Evaluation Metrics

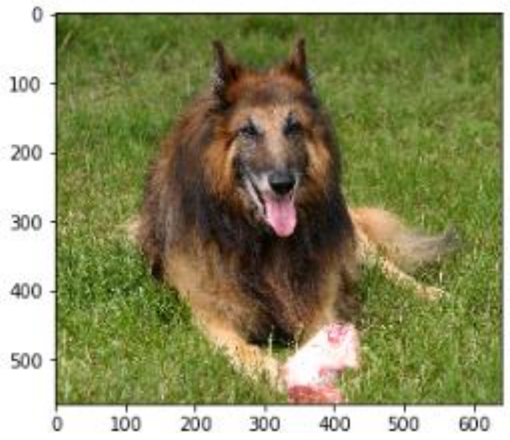
As the notebook from Udacity stated, we measure all the models with the accuracy metric. The breed model is required to have almost 60% of accuracy

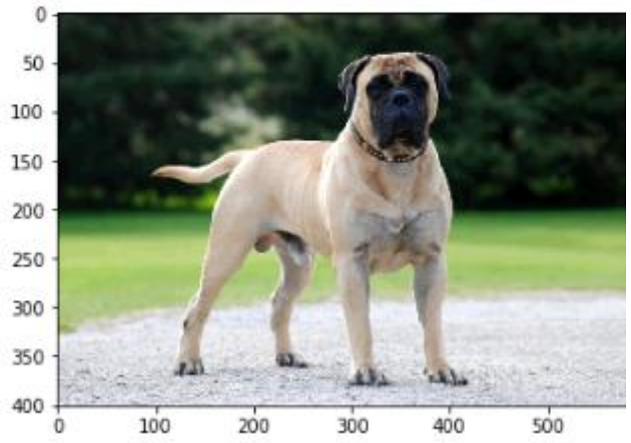
2. ANALYSIS

Data

- The dataset includes 13233 images of humans and 8351 images of dogs.
- Every image of humans is of size (250,250,3) while the images of dogs don’t have a constant size.
- 6680 images belong to the train data for dogs.
- Found 835 images belong to the validation data for dogs.
- Found 836 belong to the test data for dogs.

Below there is a few sample of dogs

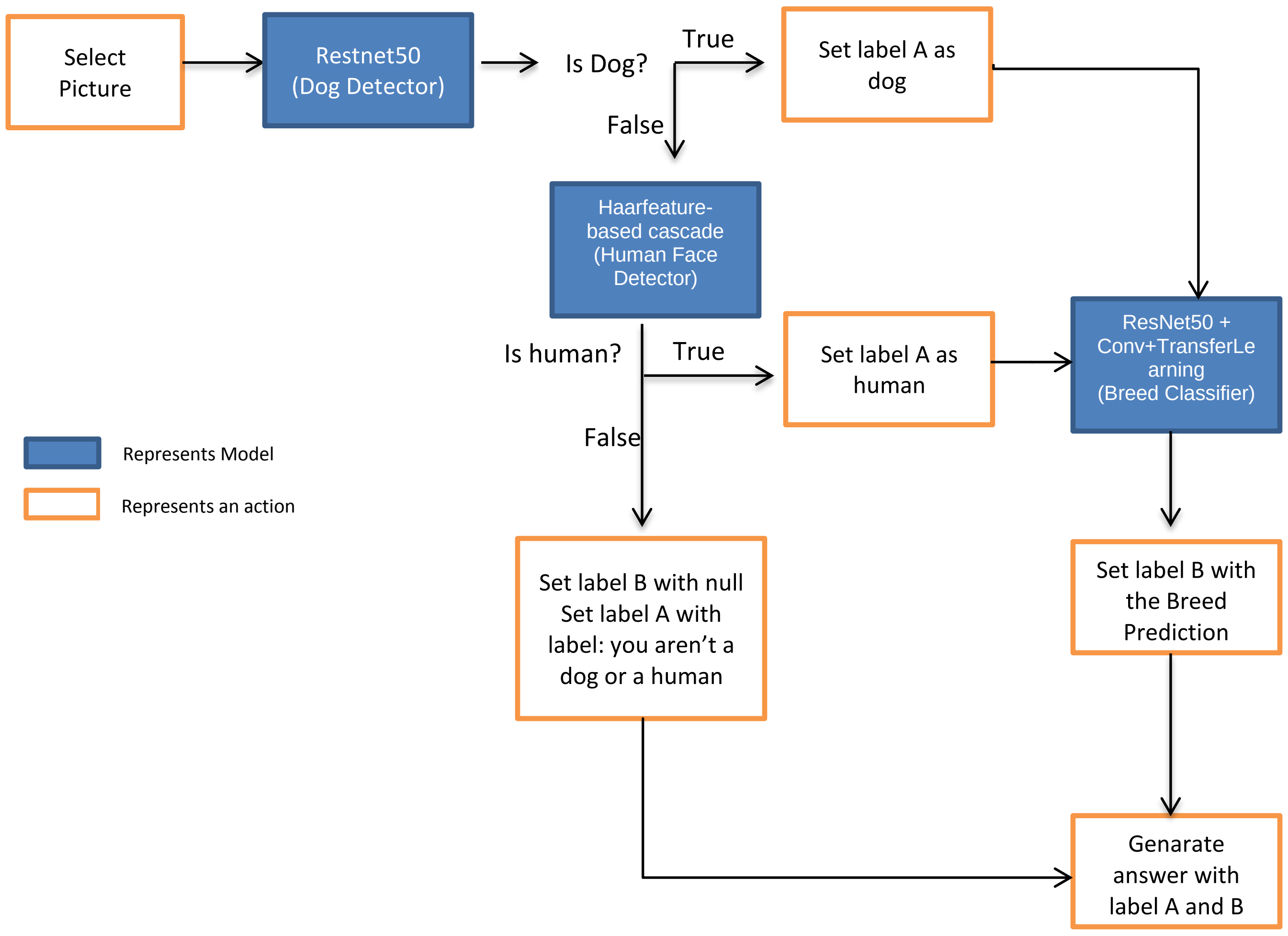




These pictures appear to be centered and with good resolution

Algorithms and Techniques

To solve the problem I used the following pipeline:


Select Picture
Restnet50 (Dog Detector)
Is Dog?
True
False
Set label A as dog
Haarfeature-based cascade (Human Face Detector)
Is human?
True
False
Set label A as human
ResNet50 + Conv+TransferLearning (Breed Classifier)
Represents Model
Represents an action
Set label B with null Set label A with label: you aren't a dog or a human
Set label B with the Breed Prediction
Genarate answer with label A and B

The algorithm has three models:

1. Dog Detector: It is a resnet50 model using tensorflow2. If the model generate a category between 151 and 268, it will be classified as Dog. (I didn't have to train the model)
2. Human Detector: If the resnet50 doesn't predict as a Dog the input will be pass through the Haarfeature-based cascade model using opencv. This model detects faces, so if there are one or more faces then it will classify as a Human. (I didn't have to train the model)

3. If the algorithm detect a human or dog, the input is passed through the Breed Classifier Model. This model is a resnet50 with two additional convolutional layers. It was retrained with all the layers freezed, except the two additional convolutional layers. This model predicts what breed is the most likely

I preferred to use tensorflow 2.0 over pytorch because it was very easy both create the model and train the model (it just takes to invoke the train method over the model instance).

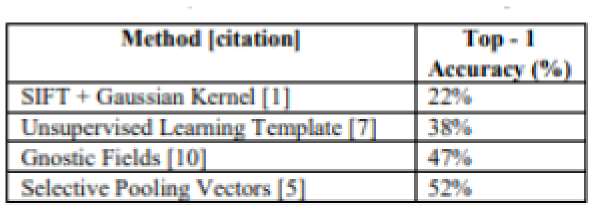
The model used to train the breed classifier was the following:

Resnet50 (175 layers)
All the weights where freezed
Conv2D(filters = 80, kernel_size=(2,2), activation='relu')
MaxPool2D()(x)
layers.BatchNormalization()
layers.Dropout(0.2)
Conv2D(filters = 240, kernel_size=(2,2), activation='relu')
MaxPool2D()(x)
layers.BatchNormalization()
layers.Dropout(0.2)
layers.Flatten()(x)
layers.Dense(133 , activation='softmax')(x)

I used google colab, because it is very easy to install software and it is possible to use GPU or TPU for free.

Benchmark

After a search in internet I find the following table (Hsu[1]) that resume a benchmark for the problem:



Method [citation]	Top - 1 Accuracy (%)
SIFT + Gaussian Kernel [1]	22%
Unsupervised Learning Template [7]	38%
Gnostic Fields [10]	47%
Selective Pooling Vectors [5]	52%

Furthermore, is required to obtain at least 60% of accuracy in the breed classifier.

[1] Hsu, D. (n.d.). Using Convolutional Neural Networks to Classify Dog Breeds. *Stanford University*. Retrieved from http://cs231n.stanford.edu/reports/2015/pdfs/fcdh_FinalReport.pdf

3. Methodology

Data Preprocessing:

The data that was used is very uniform, as you could see in the previous samples of pictures of dogs. However, it was useful to do a little of data augmentation because it could improve the performance of the model, reducing the possibility of overfitting. Thanks to the train generator class of tensorflow I could do the following:

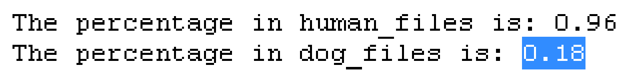
- Horizontal Flip = True
- Rotation Range = 45
- Width Shift Range = 0.3
- Height Shift Range = 0.3
- Vertical Flip =True

Implementation And Refinement

To solve the problem, I tried to use pytorch, but I didn't feel comfortable with the syntax, hence I tried to use tensorflow. I also saw a little of performance boost using tensorflow in the dog detector function, therefore I decided to build all the application with tensorflow, with the exception of the face detector.

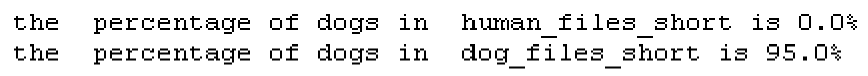
The opencv model is very good detecting faces, but has a big problem with false positives. Fortunately, the resnet50 model has a very good precision and recall so I could put first the resnet50 model to reduce the possibility of classifying a dog as human. Below I show the performance of every model

Face Detector (opencv):



```
The percentage in human_files is: 0.96
The percentage in dog_files is: 0.18
```

Dog Detector (tensorflow resnet50):



```
the  percentage of dogs in  human_files_short is 0.0%
the  percentage of dogs in  dog_files_short is 95.0%
```

I was keen to try to use SageMaker in this project, and also I want to use tensorflow.keras because I have more training in this tool. I spend a lot of days trying to run a train script over Amazon Sagemaker, but I failed. After several tries I found that there was a bug in the sagemaker estimator library. I filled a bug report, and I felt a little discouraged to continue using this platform. You could find the report in the next link https://github.com/aws/sagemaker-python-sdk/issues/1133

Next I decided to give a try into the udacity containers. However it was a little upsetting after several minutes of me being away from my pc the container could be restarted. Every time that it happened I had to reinstall all, and that was unbearable.

My following step was to use my desktop. I don't have a GPU, so it took a lot of time training the models, but at least I didn't have to start from the beginning every time. I got some decent results (close to 60%) but I wasn't totally satisfied with the final result. Fortunately, I knew google colab and I decided to use it.

To use GPU with tensorflow it is required to install two libraries: tensorflow and tensorflow-gpu. It was a very good decision because in my desktop the model took approximately 32 hour to train, while in google colab just took one hour or less. This allowed me to tried different configurations, until I finally get a result that report an accuracy close to 80%

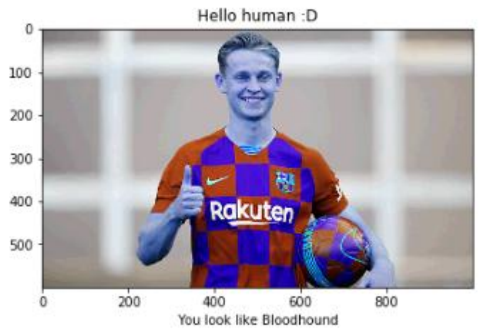
To train the breed classifier I tried different combination, but all had a very poor performance. Below I show some of the configurations that I tried

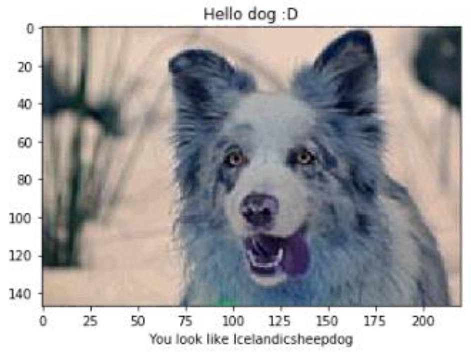
- Using only Dense layers
- Using Convolution without batch normalization or dropout
- Mistakenly use a final Dense layer with less of 133 neurons
- Have to final Dense Layers.
- Tried to use 8 or 32 as batch size.

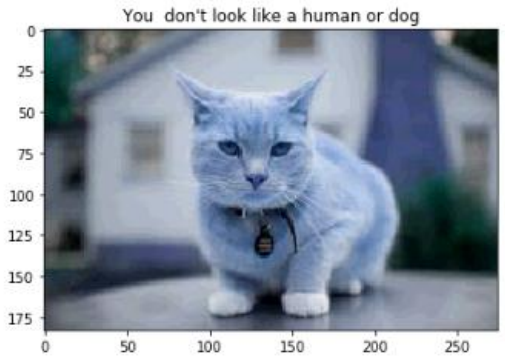
Finally I was able to use the configuration that gave an acceptable accuracy: two convolutional layers with normalization and a dense layer as output.

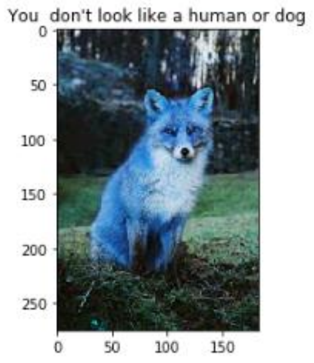
4. Results

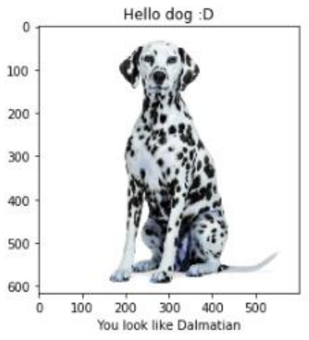
I finally get a result that report an accuracy **close to 80%** using the test dog dataset. I also run a test over 8 images, as you can see in the next table

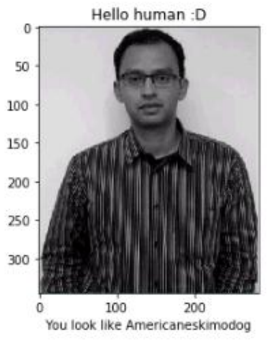


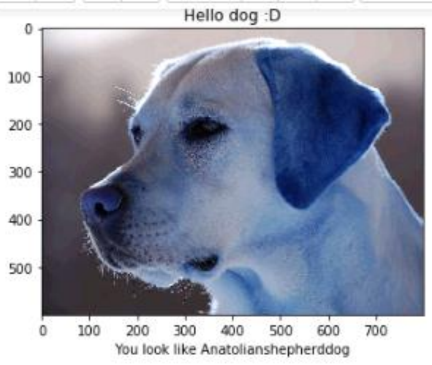


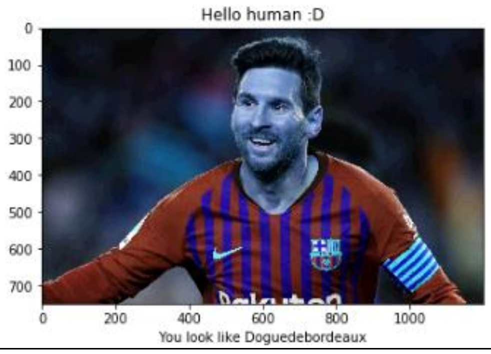












Win this little sample I could have two conclusions:

- The algorithm is good differentiating between human, dogs and other object.
- When we had a picture of a dog in a high close zoom it would predict a wrong breed

This means that we could improve our algorithm if we have more images from dog of every breed and also include some pictures only of the head of every breed. Furthermore, it could be interesting to have a dog detector that allows finding more than one dog in a picture.

The accuracy achieved by the model (80%) overcomes every value in the benchmark that was stated at beginning of this document and also overcomes the value required by Udacity.

5. Tkinter App

I created a tkinter that simplify the use of the algorithm. It contains two .py files:

- **app.py**: Have the logic of the app (tkinter)
- **classifier.py**: Run the pipeline showed in the algorithm and technique section.(tensorflow, opencv)

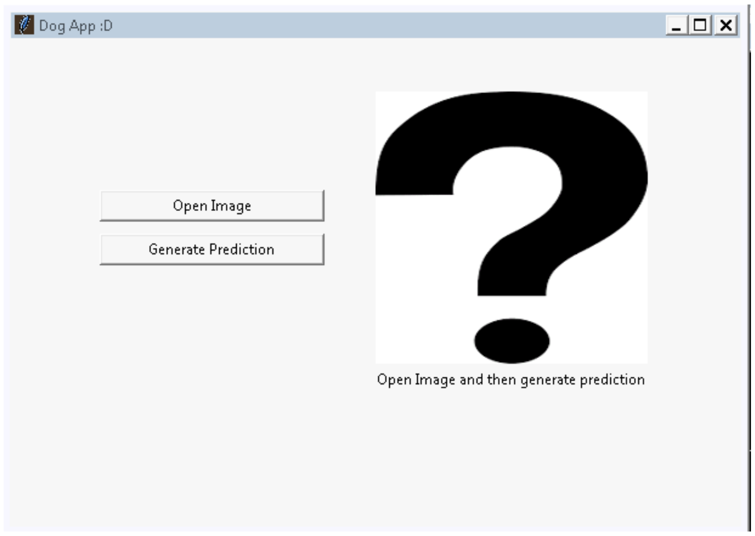
The app works with the following libraries:

- Runs with python 3.7
- Tensorflow 2.0
- Pillow 6.2.1
- Opencv 4.1.2
- Numpy 1.17.4

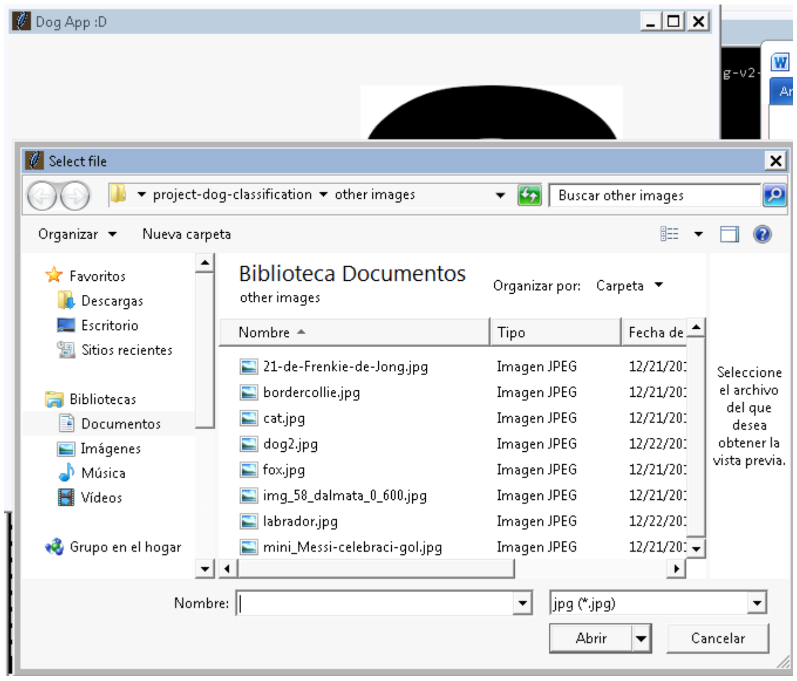
All the packages were installed with conda, with the exception on **opencv**. I had a lot of problems with opencv, *therefore I recommend to install opencv with pip*.

You could get a version of this app in my github. https://github.com/javemar/project-dog-classification/tree/master/tk_app

When you open the app you will see the following:

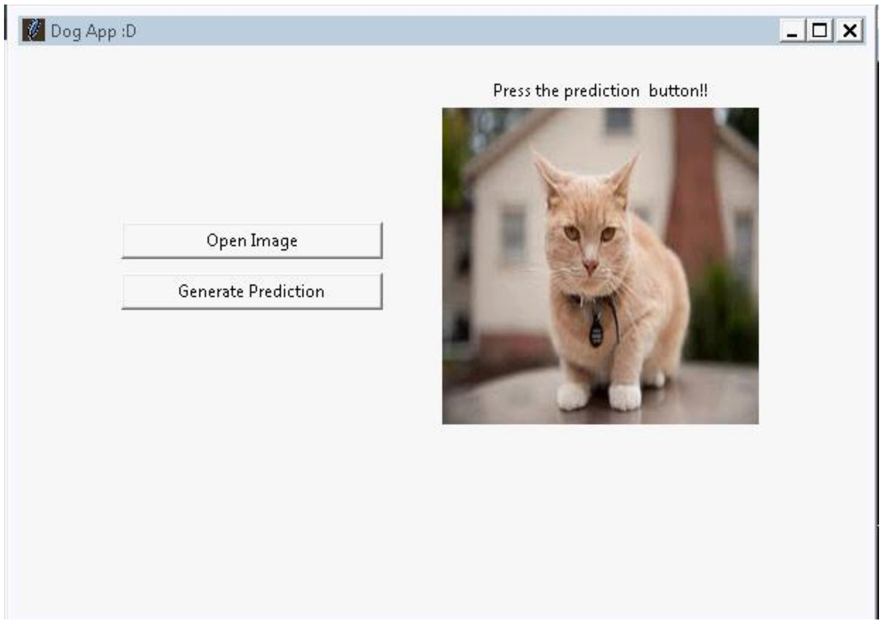


First you will need to provide a path using the button ‘Open Image’:



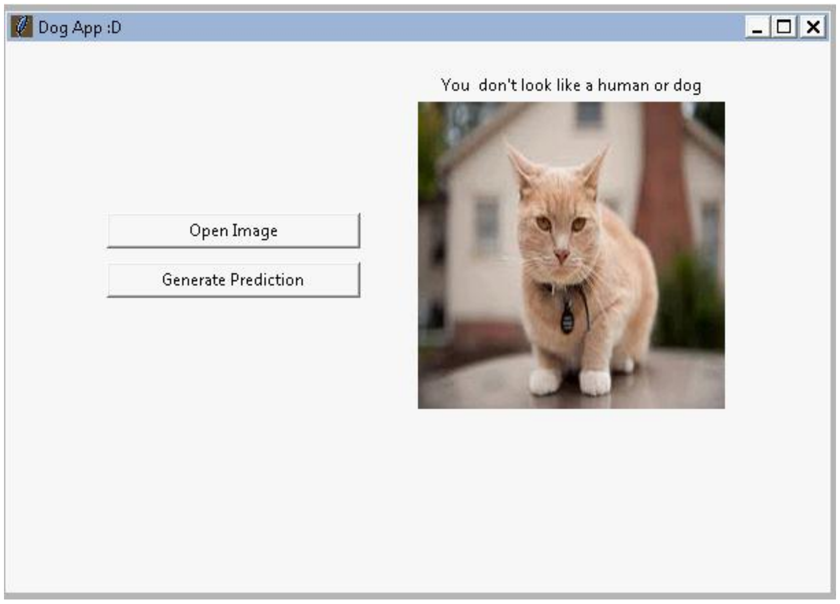
Nombre	Tipo	Fecha de
21-de-Frenkie-de-Jong.jpg	Imagen JPEG	12/21/20
bordercollie.jpg	Imagen JPEG	12/21/20
cat.jpg	Imagen JPEG	12/21/20
dog2.jpg	Imagen JPEG	12/22/20
fox.jpg	Imagen JPEG	12/21/20
img_58_dalmata_0_600.jpg	Imagen JPEG	12/21/20
labrador.jpg	Imagen JPEG	12/22/20
mini_Messi-celebraci-gol.jpg	Imagen JPEG	12/21/20

After that, you will see your picture in the app:



Press the prediction button!!

Finally, press the button ‘Generate Prediction’



You don't look like a human or dog