Dog breed classifier

Machine Learning Engineering Nanodegree

Sriram Pallapothu

09.14.2020

Project Overview

A dog is a man's best friend. You may have come across an adorable dog and you pet it and walk away with a smile. Or you may have seen an aggressive dog and wished it does not attack you. Every day we see many dogs and we may not know its breed

In this project, I will build an application that will let the user upload a dog's photo, and then the application will try to predict the breed.

The application was trained on the dog images dataset and human image dataset which is Udacity's data hosted on AWS.

Problem statement

The goal is to create an application that will accept an image and identify the dog breed. This application can also identify a human image and identify a closest looking breed of dog.

The steps followed in achieving our goal -

1. Download and import human and dog images
2. Process the data

3. Identify human images
4. Identify dog images
5. Train a model to classify dog breeds
6. Build an application that will accept images which use the trained model
7. Display the results (dog breed)

Datasets and inputs

- The dog and human image that is used for training is obtained from Udacity's AWS dataset
- The dog image data has 3 folders train, valid and test. Each folder has 133 folders with each folder for one breed.
- The human image data has 5749 folders with human images.
- A pre-trained VGG-16 model, along with weights to detect dog images, was trained on the ImageNet dataset which has a huge collection of images.

Solution statement

I begin by using OpenCV's Cascade Classifier for detecting human images. I can also use one of the pre-trained face detectors available at OpenCV[1].

For detecting the dog image, I will use the pre-trained VGG-16 model along with its weights. This pre-trained model accepts an image and gives the output as an integer between 0 and 999 which is one of the categories of images at ImageNet class

After classifying the image to be a dog or human, it will be passed into a CNN (Convolutional Neural Networks) to predict the breed if it's a dog. Based on the distinctive features of the dog breed the model will classify the dog in one of the 133 dog breeds.

Benchmark model

There are a total of 133 breeds and we need to pick 1 breed. A wild guess would have an accuracy of less than 1%. So, our initial model from scratch is deemed to be a good model if the accuracy is at least 10%

The model can be improved by using transfer learning and this must have an accuracy of at least 60%.

Evaluation metrics

As this is a multi-class classification, I am going to use the Multi-label margin Loss function to evaluate the model.

The multi-label margin loss function creates a criterion that optimizes the multi-classification margin loss between input X and output Y, both being a 2D tensor[3].

Project design

1. Firstly, I will start by downloading the datasets of human and dog from AWS.
2. The images could be in random dimensions, orientations, etc. so, pre-processing should be performed to make all images of the same size, by resize, crop, and change it to tensor. Then add some randomness by rotating, cropping, scaling, mirroring.
3. To detect human face I will use OpenCV's Haar feature-based cascade classifiers[1], which have many pre-trained face detectors.
4. To detect a dog image we can use ImageNet's pre-trained VGG-16 model which will classify the image from one of the 1000 categories[4].
5. Next, create a CNN to classify the dog breed from scratch with at least 10% accuracy
6. Create a CNN with transfer learning and train and test the model to give at least 60% accuracy
7. Finally, an algorithm that will put all the previous steps together and return the dog breed if the image is identified as a dog. Return a resembling dog breed if the image is identified as human. Return an error if its neither a dog nor a human

References

1. OpenCV cascade classifier - https://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html
2. ImageNet - http://www.image-net.org/
3. Multi-label margin loss - https://pytorch.org/docs/stable/generated/torch.nn.MultiLabelMarginLoss.html#torch.nn.MultiLabelMarginLoss
4. 1000 classifications of ImageNet - https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a