Capstone Project Report
Machine Learning Engineer
Nanodegree

Abdoulaye Beavogui Koma
August 27, 2020

Dog Breed Classifier

I. DEFINITION

1. Project Overview

According to the American Society for the Prevention of Cruelty to Animals, approximately 6.5 million companion animals enter U.S. animal shelters nationwide every year. Of those, approximately 3.3 million are dogs [1], which accounts for roughly 50.8% of the population. During the onboarding process of these dogs, it is critical to identify the breed in order to provide the most effective treatment early. With the accelerated pace in digital image processing, artificial intelligence and computer vision, we can accelerate the dog breed identification process so that dog caretakers can focus on the most important aspect: taking care of sheltered dogs.

This document reports on tackling this particular problem through Machine Learning (ML). By leveraging a set of ML artifacts, we have developed a dog breed classifier application that can accelerate the identification of dogs during the onboarding process. This application can be embedded with a computer, mobile and/or web application.

2. Problem Statement

The challenge is to develop an application that can first identify whether a human or a dog is present in a given image and estimate the corresponding or most resembling dog breed. To tackle the problem, we built a set of detectors (human, dog and dog breed estimator) through the combination of OpenCV and Convoluted Neural Network (CNN) artifacts to parse between humans and dogs and estimate the dog breed. The artifacts were trained and/or tested on a dataset provided by Udacity, Inc. containing JPEG sounds. The report will describe all the steps taken to achieve the results.

3. Metrics

We considered the following metrics to evaluate the performance and quality of the application:

[1] "Pet Statistics: How many pets are in the United States? How many animals are in shelters?" https://www.aspca.org/animal-homelessness/shelter-intake-and-surrender/pet-statistics

- Response time: Time it takes for a particular set of instructions to run. We'll be looking at how long it takes to run different modules and models to identify opportunities for optimization

$$Response\ time = end\ time - start\ time$$

- Accuracy of the detectors: Number of true positives and true negatives (or correct identifications) over the entire dataset. This metric will tell us how well our application detects humans, dogs and estimates the corresponding dog breed

$$Accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

- Precision of the detectors (optional): Number of true positives over all positives. A lower number of false positives will lead to a higher precision

$$Precision = \frac{true\ positives}{true\ positives + false\ positives}$$

- Recall (or sensitivity) of the detectors (optional): Number of true positives over the sum of true positives and false negatives. A lower number of false negatives will lead to a higher recall

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

- F1 score of the detectors (optional): A metric that considers both precision and recall (harmonic mean). It is relevant when provided an unbalanced class distribution. The closer to 1, the more precise and accurate the application is

$$F1\ score = 2 \times \frac{Precision\ \times\ Recall}{Precision + Recall}$$

II. ANALYSIS

1. Data Exploration

The dataset provided includes 21,584 colored images in a JPEG format:

- Dog dataset contains 8351 images parsed in 133 folders, each corresponding to a unique dog breed. The images come in all shape and sizes and some images may contain multiple dogs or a human with a dog. The dataset is imbalanced as we see can an uneven class distribution of dog breed images. The plot below shows the image count

available per dog breed, where 9 out of 133 breeds contain more than 124 images (above one standard deviation away from the mean)

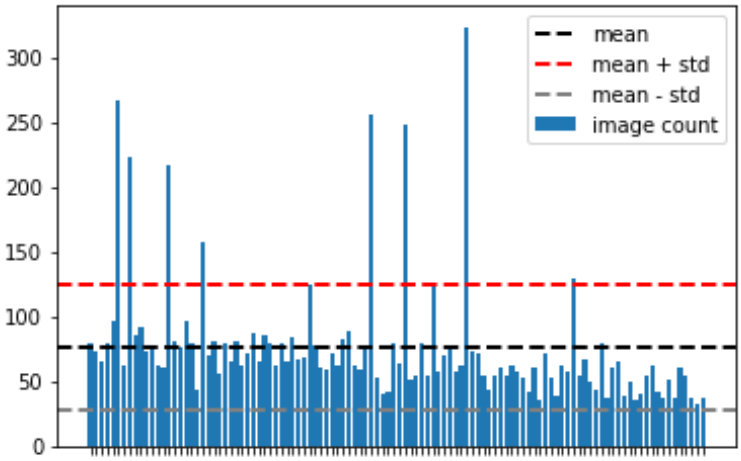


Fig1. Class distribution of dog breed image count per dog breed

- Human dataset contains 13233 images of humans. All images are of size 250x250 and are center cropped to the human face

2. Algorithm and Techniques

The application is composed of four main modules: human detector, dog detector and dog breed estimator and run in the following order:

- Image Preprocessor: read and process an image
- Dog detector: identify a dog in the image
- Human detector: identify a human in the image
- Dog estimator: provide an estimate of dog breed in the image

Fig2. Example of the application output

If the application cannot identify a human or a dog in the image, the application will report that it's neither a dog nor a human and will not provide a dog breed estimation.

a. Human detector

Its purpose is to identify a human in a given image. This detector is developed leveraging OpenCV open-source artifacts, more specifically the Haar feature-based cascade classifiers which specialize in detecting human faces in images. The program first loads the Haar feature-based cascade classifier (provided as an XML file by OpenCV on their GitHub page). An image is then loaded, converted into grayscale and fed to the classifier. The classifier returns a NumPy array with each row containing the coordinates of a detected face, but the program only needs to check if the returned array is not empty (a non-zero object means at least one human face was identified).

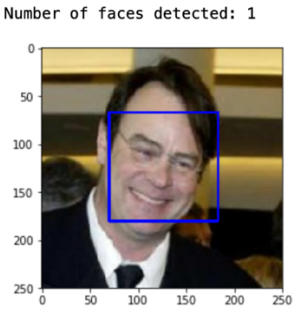


Fig2. Example of an identified face using the OpenCV Haar algorithm

b. Dog detector

Its purpose is to identify a dog within a given image. This detector is developed leveraging pretrained Pytorch CNN classifiers. These artifacts were trained on ImageNet to classify up to 1000 animals from images, including 117 breeds of dogs. The program first loads the pretrained CNN classifier. An image is then loaded, preprocessed and fed to the CNN classifier. The CNN classifier returns an index value between 0 and 999, representing a prediction of a respective category in ImageNet. Since the ImageNet dictionary labels indexes all values between 151 to 268 (inclusive) as dogs, the program only evaluates if the returned index is within 151 and 268.

c. Dog breed estimator

Its purpose is to estimate the most resembling dog breed in a given image. This detector is developed leveraging a pretrained Pytorch CNN feature detector with a custom classifier.

In order to train the classifier, we first parse the image dataset in three sets:

- Training set: this set is used to train the CNN classifiers. We apply transformations such as cropping, flipping, rotating and scaling to the set as it will to improve the model's performance in generalizing dataset

- Validation set: this set is used to measure the model's performance and helps us determine when our model starts overfitting. We do not apply any transformations to this set

- Testing set: This set is used to measure the trained model's performance on data it has not seen yet. We do not apply any transformations to this set

Upon loading the pretrained CNN, the feature detector's parameters are locked and integrated with a perceptron with an output size of 133, with each output representing class of dog breed. To classify, we use a Categorical Cross-entropy with the negative log likelihood loss function to penalize heavily error in the prediction during the training.

The model is trained through a defined set of iterations. through each iteration, it is trained with the training dataset and tested with the validation dataset. For each successfully lowered validation loss, we save the trained model to a directory for the testing phase. After the training, the model is finally tested on the testing set.

The selected pretrained model for the dog estimator will be the same as the dog detector pretrained model.

d. Benchmark

There are many widely popular open-source models available to detect humans, dogs and dog breeds in a given image higher than 80%. As an example, a team from the Istanbul Technical Institute provided the following accuracy of their dog breed classifier models: ResNet-50 (89.66%), DenseNet-121 (85.37%), DenseNet-169 (84.01%) and GoogleNet (82.08%) [2].

Our goal in this project is to achieve at least an 80% accuracy, precision, recall and F1 score on each detector and estimator modules in the application, including the CNN performance on the testing set. We will benchmark our models (more specifically the dog breed classifier: CNN scratch vs CNN pretrained) in terms of performance and accuracy to identify the best approach in designing the application.

III. Methodology and results

1. Data Preprocessing

[2] "Modified Deep Neural Networks for Dog Breeds Identification" https://www.researchgate.net/publication/325384896_Modified_Deep_Neural_Networks_for_Dog_Breeds_Identification

To leverage the ML/CV models for this project, we need to preprocess and transform the data in the appropriate format

a. Using OpenCV classifiers

The Haar feature-based cascade classifiers expect images in grayscale as it is standard procedure. The detectMultiScale function of the cascade classifier takes grayscale image as an input.

b. Using CNN pre-trained classifiers

Per the guidance of PyTorch, the pre-trained CNN models used expect input images normalized in tensors representing 3-channel RGB images of shape (3 x height x width), where both height and width are 224 pixels long. The images are be loaded into a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]. We leverage the PIL module from Pillow to preprocess the images.

2. Implementation, evaluation and refinement

a. Application flowchart

The application is a python program that takes an image path as an input and follows the following flowchart:

Step 1: The application preprocesses the image and feeds the image path to dog detector. The dog detector returns a TRUE/FALSE indicator on whether or not a dog was identified in the image. If a dog is identified, the application proceeds to step 3

Step 2: The application feeds the image path to human detector. The human detector returns a TRUE/FALSE indicator on whether or not a human was identified in the image. If a human is identified, the application proceeds to step 3

Step 3: The application feeds image path to dog breed estimator. The estimator returns the dog's breed class with the highest probability. The program then prepares a message for display providing the dog's breed

Step 4: if neither a dog not a human is detected, the program prepares a message confirming it did not identify either classes

Step 5: Display the image with the appropriate message

b. Human detector

To implement this module, we benchmarked two Haar feature-based cascade classifiers in terms of accuracy and runtime performance on 500 images and observed the following results:

- **haarcascade_frontalface_alt:**
  - Accuracy: 0.934
  - Precision: 0.9387454710144927
  - Recall: 0.9339999999999999
  - **F1 score: 0.9338210521249458**
  - **Time of execution: 6min 12s**

- **haarcascade_frontalface_alt2:**
  - Accuracy: 0.892
  - Precision: 0.9104471360841668
  - Recall: 0.892
  - **F1 score: 0.890772722307851**
  - **Time of execution: 5min 51s**

We observe a higher F1 score (93.4%) with Haarcascade_frontalface_alt classifier as it provides a higher score, but a slightly slower response time over 500 images.

c. Dog detector

To implement this module, we benchmarked three pretrained PyTorch CNN classifiers in terms of accuracy and runtime performance on 500 images and observed the following results:

- **VGG-16:**
  - Accuracy: 0.983
  - Precision: 0.9830173886259905
  - Recall: 0.983
  - **F1 score: 0.982999846998623**
  - **Runtime: 25.5s**

- **ResNet-50:**
  - Accuracy: 0.987
  - Precision: 0.9872358221379147
  - Recall: 0.987
  - **F1 score: 0.9869984268096439**
  - **Runtime: 19.9s**

- **DenseNet-121:**
  - Accuracy: 0.977
  - Precision: 0.9776897840481655
  - Recall: 0.977

- **F1 score: 0.9769916940015346**
- **Runtime: 31.2s**

We observe a higher F1 score (98.8%) with the ResNet-50 classifier as it provides a higher F1 score and faster runtime response time over 500 images.

d. Dog breed estimator

To implement this module, we benchmarked two PyTorch CNN classifiers (scratch vs pretrained) in terms of accuracy and runtime performance on a testing set (836 images) and observed the following results:

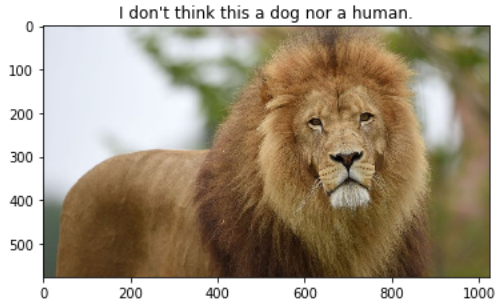
- **CNN model from scratch - Convoluted feature detector downsizing image size from 224x224x3 to a 56x56x12 + (37632,512,133) perceptron classifier:**
  - Accuracy: 0.11722488038277512
  - Precision: 0.08347584357147496
  - Recall: 0.09791442892946653
  - F1 score: 0.07453304414580311
  - **Runtime: 11.5s**
  - **Training time: 1h 31min 5s**

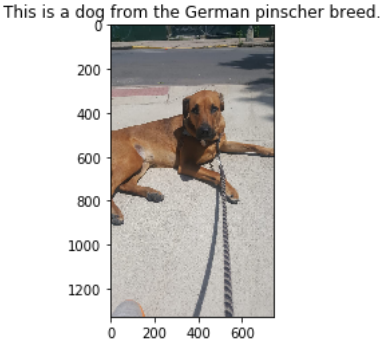
- **Pretrained CNN model - ResNet-50 feature detector with a (2048, 133) perceptron classifier:**
  - Test Accuracy: 0.8744019138755981
  - Test Precision: 0.8851674641148326
  - Test Recall: 0.861296097386323
  - Test F1 score: 0.8593547619281459
  - **Runtime: 11.4s**
  - **Training time: 47min 44s**

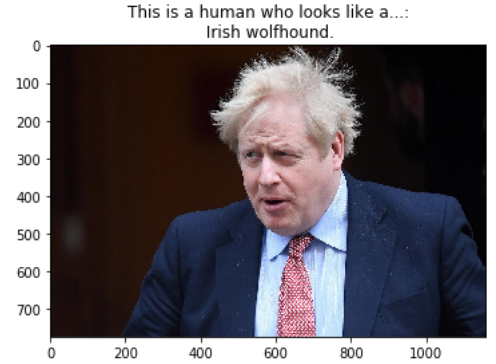
It is clear that leveraging a pretrained CNN model increases substantially the accuracy and F1 score (85.7% F1 score for pretrained CNN vs 7.5% F1 score for scratch CNN) of the model with a much shorter amount of training effort.

3. Justification

Below is shown a set of runtime results:







The application correctly identifies humans from animals, dogs from animals, and correctly estimates the dog breed. The overall accuracy, precision, recall and F1 score of each application detector is above 85%, well above our expectations. We do see a few ways to improve the algorithm:

- Leverage CNN artifacts for human detector as it may generate a higher F1 score
- Train the CNN artifacts on a large dataset as it may generate a higher F1 score
- Identify a more lightweight CNN model for dog estimator as it over 90MB in size
- Deploy this application onto a cloud platform (AWS / Google Cloud) to allow ease of use and/or testing of real-time performance at a local, regional and/or global scale