Dog Breed Classifier

Thiago Francisco Martins

Capstone Project Report

July 10th, 2020

I. Definition

Project Overview

At first, the idea of classifying a dog seems simple, but even for humans that might be a difficult task. The idea of looking to a dog and knowing which breed it is sounds something simple but effective for owners that are looking to adopt. Users of such technology could look to a dog they like and apply the algorithm to retrieve not only which breed is the dog they are looking at, but also retrieve important information about that breed such as temperament, habits, abilities and special cares. That would increase the right choice when adopting a pet and owners would be much more informed about the breed.

Problem Statement

In this project, the goal is to build a machine learning algorithm that given an image of a dog, it will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed. The idea is to have a robust, fast and reliable algorithm that detects with high accuracy a dog's breed. The algorithm will also estimates the most close dog breed if a supplied image is from a human. In order to achieve this goal a CNN network will be trained using a method called transfer learning, in which we use a pre-trained network that tackles a similar problem and apply it to our case with some modifications.

Metrics

In this project, two metrics were chosen to evaluate the performance of the algorithm. The first one is accuracy, which measures how many true positives (correct predictions) out of the total of instances the network predicted correctly.

1) Accuracy

$$\frac{TruePositives}{TotalInstances}$$

Equation 1: Accuracy metric

Where True positives are the number of instances that were correctly predicted

2) Multi Log Loss

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

Where,

N	*No of Rows in Test set*
M	*No of Fault Delivery Classes*
Y_{ij}	*1 if observation belongs to Class j; else 0*
P_{ij}	*Predicted Probability that observation belong to Class j*

Equation 2: Multi Class Log Loss

A similar competition in Kaggle: [1] Dog Breed Identification, used the same kind of evaluation and it accurately represents not only if the classes are labeled correct but also how confident our model is when it predicted classes are equal to the target.

II. Analysis

Data Exploration

In this project, we will use the dataset provided by the Udacity [2] notebook. Analyzing the data, we can see that the data contains 133 dog breeds and has 8750 images. Please find below a graph representing how the number of instances are distributed across the different labels in the train folder of the dataset provided by the notebook. In Figure 1: Distribution of number of images per dog class, We can see that there is around between 35 to 60 images for each class. That might be few images for a high accuracy classifier but we will experiment with this data to see if we can achieve reasonable results. An additional human dataset will be used to detect if a human is present on the picture.

Exploratory Visualization

The dog dataset has 5750 images from humans, which is pretty high to our goal. [3] Dog dataset, 8750 images (6680 for training, 836 for testing, and 835 for validation). Training set: 133 classes. Around 40 images per class. [4] Human dataset, 5750 images, one image per person.

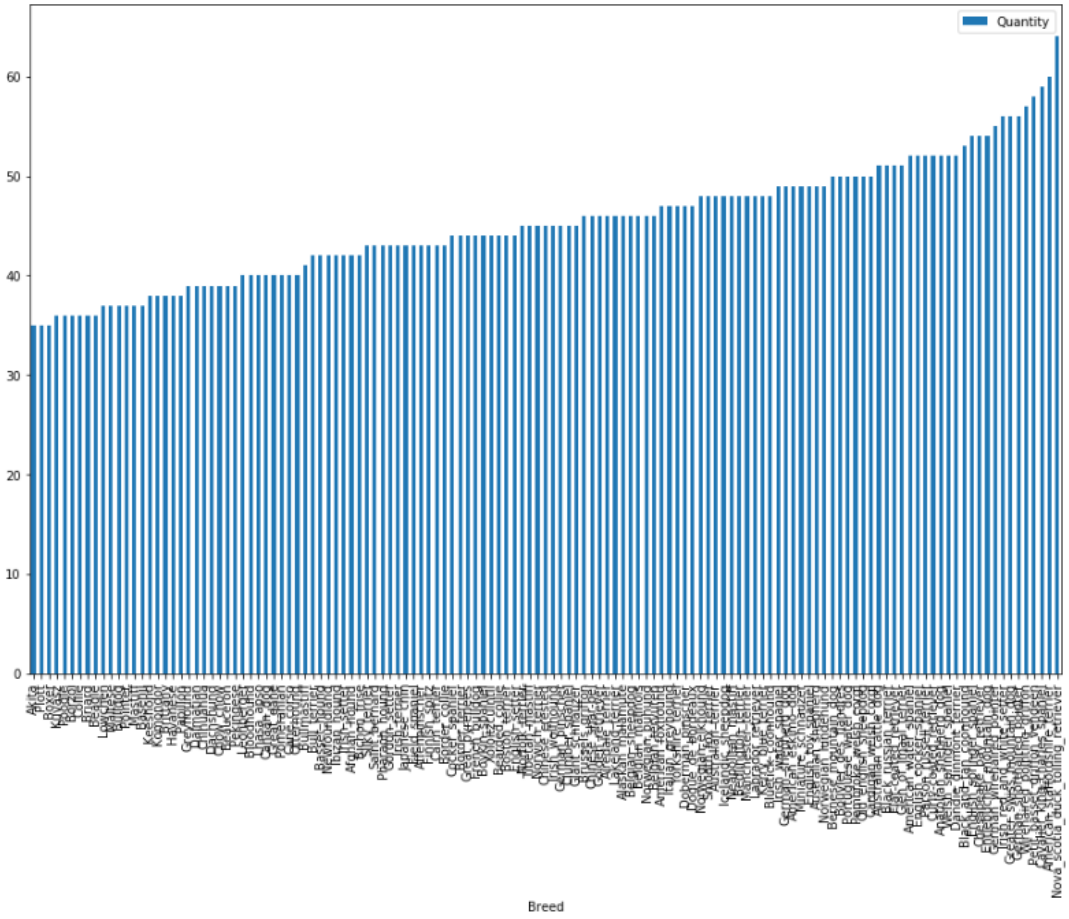

Figure 1: Distribution of number of images per dog class

Figure 2: Example of an image from the dog dataset

Regarding our human faces dataset, we have 13233 total images from 5749 different people, averaging 2.3 images per class. The lowest number of images we have per person is 1, while the maximum is 530 (Images from George W. Bush).

Figure 3: Example of an imge from the human face dataset

Algorithms and techniques

The main steps of the algorithm are described in **Figure 4**. It is basically composed of 3 main process steps.

- The Human face detector
- The Dog detector
- The Dog breed classifier

Basically we provide the algorithm with an image file and we apply a process to check if the image belongs to a human face. If it does not belong to a human, we apply a second algorithm to check if the image is from a dog. After these 2 steps we will have an idea if the image belongs to a dog, human or neither of them (In which case we will raise an error message).

Regardless if the image is from a dog or a human we will apply the dog breed classifier to see which dog breed is the most close to the picture provided. Then the algorithm can print if it thinks it is a human or a dog, and which dog breed is the most similar. Figure describes a workflow of the solution

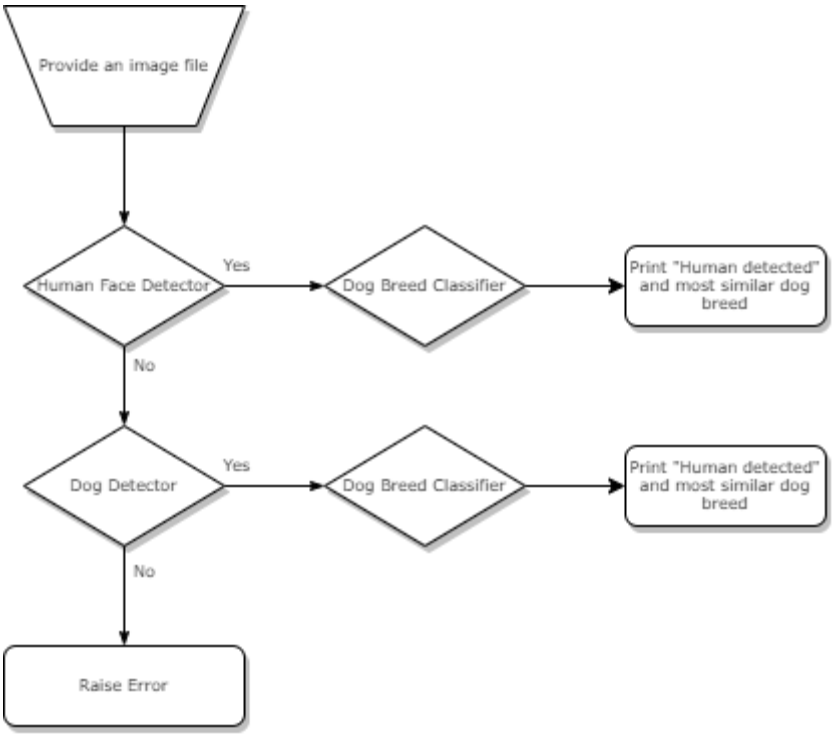

Figure 4: Workflow of the algorithm

The dog detector and dog breed classifier are designed using Convolutional Neural Networks, they are a type of network suitable to image classification and detection tasks. It is created using convolution layers.

Neurons in the first convolutional layer are not connected to every single pixel in the input image (like they were in previous chapters), but only to pixels in their receptive fields (see Figure 14-2). In turn, each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer. This architecture allows the network to concentrate on small low-level features in the first hidden layer, then assemble them into larger higher-level features in the next hidden layer, and so on.

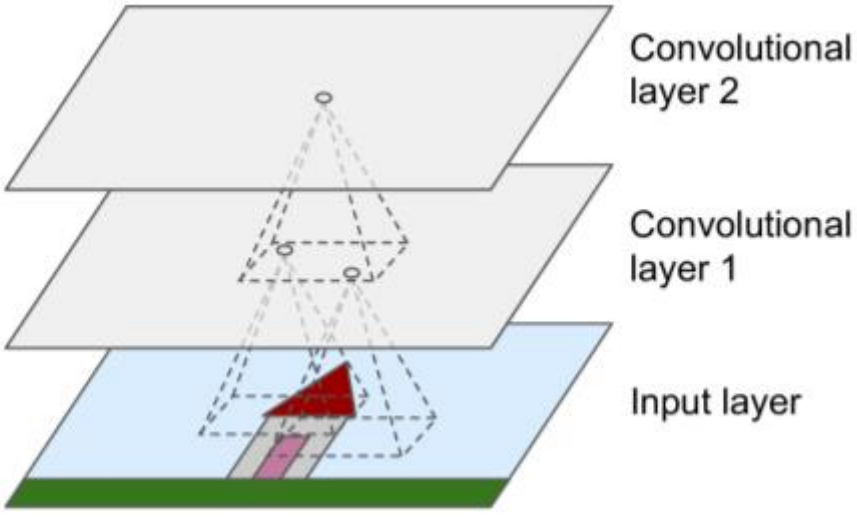

Figure 5: Convolutional Layers

Another layer present in CNNs are the pooling layers, their goal is to subsample (i.e., shrink) the input image in order to reduce the computational load, the memory usage, and the number of parameters (thereby limiting the risk of overfitting). This type of subsample could be a maximum (Where we take the pixel with the maximum value), average, or minimum. Convolutional layers and pooling layers are the heart of this type of network, the best ones usually have several of them stacked in determined sequence.

A VGG-16 network for instance has a very simple and classical architecture, with 2 or 3 convolutional layers, a pooling layer, then again 2 or 3 convolutional layers, a pooling layer, and so on (with a total of just 16 convolutional layers), plus a final dense network with 2 hidden layers and the output softmax layer.

1) Human face detector

The human face detector is built using one of the built-in functions of OpenCV, the cascade classifier, which is designed using [5] Haar cascades, a technique that trains several weak classifiers into one strong classifier that can identify faces. There is already a XML file that were obtained trained specifically for face detection purposes, and we make use of such file.

2) Dog Detector

In the dog detector we simply make use of a pre-trained VGG-16 [6] model that according to their documentation [7] outputs a range of values an image is from a dog.

3) Dog Breed Classifier

We designed 2 approaches to tackle the problem of identifying which breed a dog belongs to. Both approaches make use of Convolutional Neural Networks, networks designed for better image recognition.

The first one is to build a CNN model from scratch, with all layers and weights having to learn entirely from the dataset. And another using transfer learning. We will take the one that outputs the best result and apply to our pipeline described in **Figure 4**

Benchmark

As we have 133 classes, a random guess would yield an accuracy of 0.75%. We would like to achieve much better performance than random, so we defines that for our CNN created from scratch, a good output value would be around 10%, as it would 13 times better than a random guess.

When it comes to transfer learning, as the network we will make use of was trained on thousands of images Imagenet Large Scale Visual Recognition Challenge 2014 [8], we expects a much higher accuracy, of 60% or above. These goals (10% and 60%) are outlined by the Udacity Notebook [2] as the minimum requirements for the solution to be accepted.

III. Methodology

Data preprocessing

In order to feed our CNN, we need to convert an image format file to a reasonable format, like a Numpy array or a tensor so we feed the network with numbers. We also apply some augmentation techniques that consist of modifying the training instances randomly so we have more variability on the training set and reduce the risk of overfitting. The following steps were applied.

1) Resize the image: Some files are too big to feed a network, if we don't reduce the size of the input, the network might become too big to fit in memory. We reduced the image sizes to 299x299 images, same size as the network we will use for transfer learning
2) Convert Jpeg files to numbers: Our network need numbers to work, we convert images in the RGB format to a numeric input tensor.
3) Rescale: Pixel values vary from 0 to 255. These values don't work well with neural network as a convergence might be more difficult to achieve and training time might be slower. We rescale the pixel values so they fall into the 0 to 1 range.

After these steps the network is ready to be fed by the inputs, we apply them regardless if we are working with training, validation or test set. However, specifically for the training set we apply data augmentation to improve the performance of the network.

1) Randomly flip the image horizontally: This step is taken to avoid overfitting and add more variability to the training set.
2) Randomly flip the image vertically: This step is taken to avoid overfitting and add more variability to the training set.
3) Randomly rotate the image 45 degrees: This step is taken to randomly alter the image so the network learns to identify a dog breed regardless if the dog is not in a straight position.
4) Randomly alter the brightness: This step is important so the network learns to identify a dog breed regardless if the image is underexposed or overexposed

Figure 6: A few example of the augmented dataset

We make use of a built-in function from tensorflow, ImageDataGenerator to ease these steps and prepare the data. This function also batches, and optionally shuffle the data.

We also create a numeric array representing the different dog classes that we can encounter and the network will learn based on these numeric values.

After all these steps, we are ready to train our network.

Implementation

The first implementation we come up is to build a convolutional neural network from scratch. This network is created using Keras and Tensorflow and the design resembles a simple CNN. The first step is to build four sequences of convolutional layers followed by a Max Pooling Layer. After that we add a Dropout layer to increase the accuracy of the network and avoid overfitting. The last steps consists of a Global Average Pooling layer to condense the network followed by a 500 dense network with Relu activation function to improve speed) and the last layer, a Softmax layer, appropriate to classification tasks such as this one.

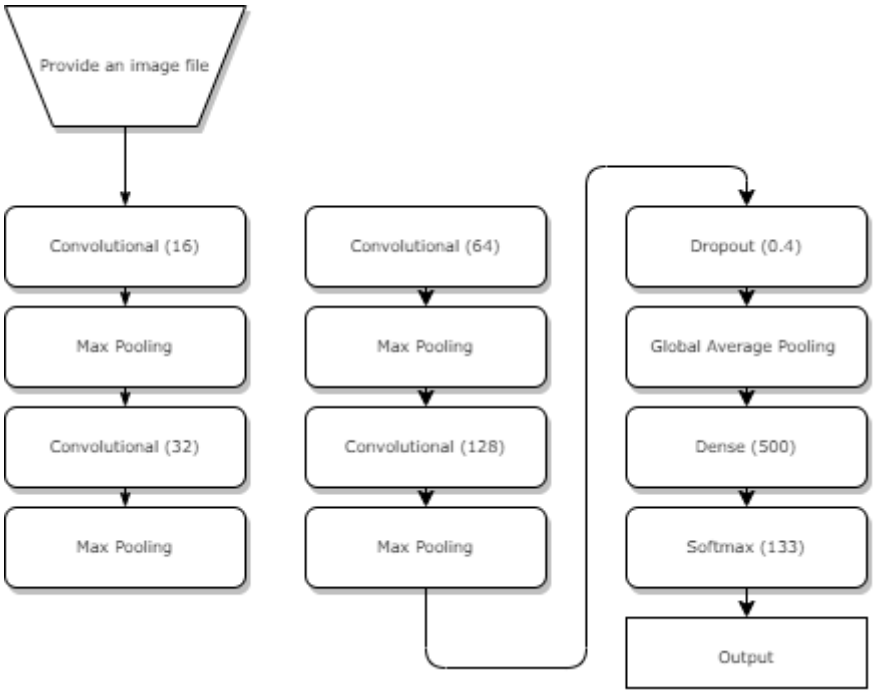

Figure 7: CNN model from scratch

The metric chosen to train the network is the accuracy, where we have a good sense of how well the network is performing. We also used categorical cross entropy loss, well suited to classification problems and use the Nadam optimizer, which is an adaptive learning rate algorithm that converges slightly faster. We created a function where all these inputs can be fed along with the image tensors.

Refinement

Although our network is capable of achieving good results, a better idea is to use a better and pre-trained network that tackles a similar task and tweak it so it can solve our own problem. This technique is called transfer learning and it is widely used when someone doesn't have enough training instances, lack computer resources and needs a reliable network. This is exactly our case so we apply this technique to improve our accuracy.

The network chosen is the Xception [9], it merges the ideas of GoogLeNet and ResNet (winners of the 2014 and 2015 challenges respectively), but it makes some modifications to improve accuracy and performance. This type of network use less parameters, less memory and less computations than regular convolutional layers, and in general they even perform better. So it is a perfect fit given our circumstances of limited resources.

The step we take is to download the Xception [9] network and get rid only of the last layer, then we add a Global Average Pooling layer to reduce the computational power that we will need and add one last softmax layer so it can learn to distinguish our 133 different dog breeds.

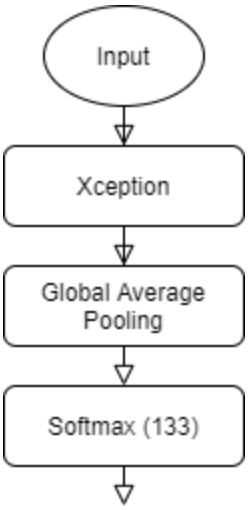

Figure 8: The transfer learning network

The process of training consist in freeze the layers correspondent to the Xception [9] , so the network don't change significantly the weights already learned previously on the Xception [9] network, and then train the network until the validation error doesn't improve for 10 epochs. After that, we save the model in which the epoch performed better on the validation set and then we train the network again the entire network but this time unfreezing the all layers, given the network more freedom to learn new aspects.

IV. Results

Model Evaluation and Validation

In order to accurately evaluate the capability of the network to work well with unseen data we previously set aside a test set that has dog in numerous conditions (In dark environments, with pose changes, occlusion and others). We believe that such variability help us to evaluate better how our network will perform in unseen data.

We designed two different models and we have 2 different metrics. These four cases are discussed below

1) Accuracy on model created from scratch.

The network created from scratch achieved an accuracy of 10.765%

2) Multi Log Loss on model created from scratch.

The network created from scratch achieved a multi log loss of 4.222

3) Accuracy on model using transfer learning.

The network using transfer learning achieved an accuracy of 86.722%

4) Multi Log Loss on model using transfer learning

The network using transfer learning achieved a multi log loss of 0.410

We believe these results can be trusted to work in real world as we fed the network with never seen data. The network is not robust to some aspects such as occlusion, multiple dogs in one picture or robust to other image quality aspects other than brightness, however the results are promising as a random classifier would 0.75% accuracy while our network achieved 86.722%.

Justification

Both of the implementations developed achieved their benchmark. The model created from scratch achieved a 10.765% accuracy, which is slightly better than the goal of 10%.

Also, our model using transfer learning achieved much better results than expected, we achieved 86.722% accuracy while the goal was 60%. Almost 50% more than we were expecting. This of course is due to the network we used, the Xception [9], which was trained in a much more challenging dataset.

V. Conclusion

Free-Form Visualization

After testing the performance of the algorithm we can finally test our entire algorithm, described at Figure 4: Workflow of the algorithm. We selected a few samples of dogs and people to check the results.

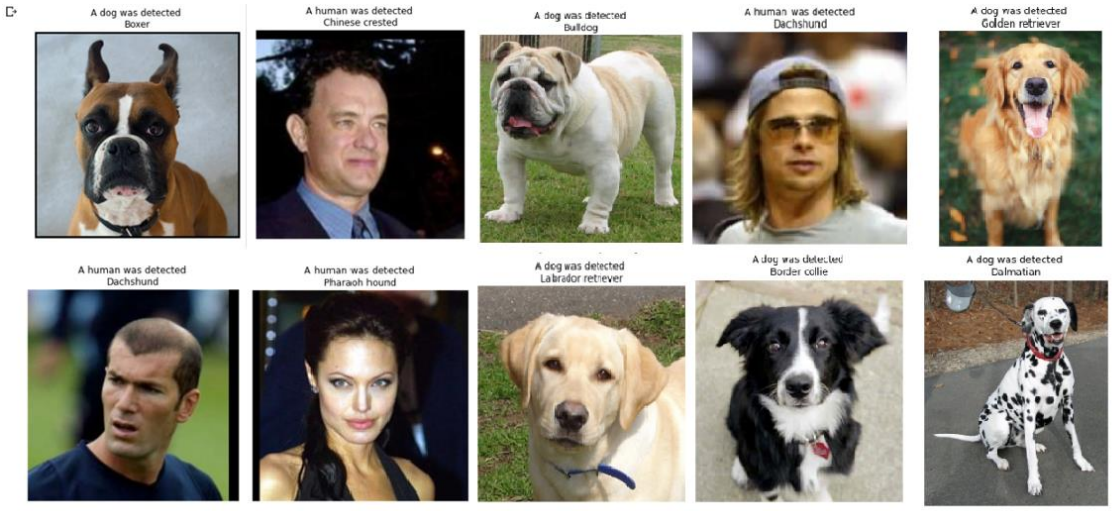

Figure 9: Examples of output

Reflection

This project was fun to implement and proportionated a better understanding of how CNN works. There was a need not only how to develop a reliable model but also how to handle the data and apply pre-process steps to the images such as data augmentation.

The main challenge we faced in this project was to handle tensors in tensorflow. The final solution end up using built-in functions of keras but the initial idea was to take advantage of tensor graph characteristic of the problem and use it in order to have better training speed and performance. It is a nice to have feature.

Another complicate problem is the case of using a second data along with the one we used. Other datasets have a different name notation and some of them don't have the same breeds, which makes it harder to mix two datasets together.

Improvement

Further improvements are possible for this project. The project was trained in a small dataset considering it only has around 50 instances per class and a lot of other corner cases such as multiple dogs in an image or obstruction weren't handle.

To improve the performance of the Dog Classifier network, there is a few points to tackle

1) Increase the number of training instances
2) Try out other powerful CNN architectures such as ResNet or SENet
3) Generate other data augmentation cases such as regarding color saturation, JPEG quality, contrast, translation
4) Insert another network before the dog breed classifier that will detect the region of the image that belongs to a dog and then crop only the region of interest
5) Implement regularization techniques such as l1 norm
6) Get rid of some layers of the Xception [9] network

References

[1] Kaggle: Dog Breed Identification Competition
https://www.kaggle.com/c/dog-breed-identification
[2] Udacity Dog Breed Classification
https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
[3] Stanford Dog Dataset
http://vision.stanford.edu/aditya86/ImageNetDogs/main.html
[4] Labeled Faces in the Wild Dataset
http://vis-www.cs.umass.edu/lfw/
[5] Viola and Jones - Rapid object detection using a boosted cascade of simple feature
https://www.merl.com/publications/docs/TR2004-043.pdf
[6] K. Simon- yan and A. Zisserman - VGG-16 Network
https://arxiv.org/abs/1409.1556
[7] VGG-16 Labels
https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a
[8] Imagenet Challenge 2014
http://www.image-net.org/challenges/LSVRC/2014/
[9] François Chollet - Xception: Deep Learning with Depthwise Separable Convolutions
https://arxiv.org/abs/1610.02357