Machine Learning Nanodegree Capstone Report

Dog Breed Classifier using CNN

Rahul Garg

27, September 2020

Definition Project

Overview

The Dog breed classifier is a well-known problem in ML. The problem is to identify the breed of dog, if a dog image is given as input, if supplied an image of a human, we have to identify the resembling dog breed. The idea is to build a pipeline that can process real-world user-supplied images and identify an estimate of the canine's breed. This is a multi-class classification problem where we can use supervised machine learning to solve this problem. Therefore, each owner should at least know the breed(s) of their dog(s). There can be a web app that returns the estimated breed of a dog given a user-supplied image of a dog or human (if people want to know which breed they look like).

Problem Statement

The aim of the project is to build a pipeline to process real-world, user-supplied images. The algorithm will identify an estimate of the dog's breed given an image. When the image is of a human, the algorithm will choose an estimate of a dog breed that resembles the human. If neither a dog nor a human is detected, then an error message is output. Therefore, the models in place should be capable of detecting a dog or human in an image, classify the dog to its breed and classify a dog breed that the human resembles.

The goal of the project is to build a machine learning model that can be used within a web app to process real-world, user-supplied images. The algorithm has to perform two tasks:

- **Dogface detector:** Given an image of a dog, the algorithm will identify an estimate of the canine's breed.
- **Human face detector:** If supplied an image of a human, the code will identify the resembling dog breed.

Metrics

Accuracy will be the main metric used to test both the benchmark model and the solution model. This is so because the problem at hand is a classification task, where the model should classify the images accurately.

Accuracy = (T P +T N)/(T P +T N+F P +FN)

True Positive(TP), True Negative(TN), False Positive(FP) & False Negative(FN).

The validation loss value is defined by cross-entropy loss, which is also called the log loss or multi-class loss in some platforms (e.g. Kaggle). The loss value measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy values increase if the predicted probability diverges from the actual label3.

Data Exploration

For this project, the input format must be of image type, because we want to input an image and identify the breed of the dog. The dataset has pictures of dogs and humans.

Dog images dataset:

The dog image dataset has 8351 total images that are sorted into the train (6,680 Images), test (836 Images), and valid (835 Images) directories. Each of this directory (train, test, valid) has 133 folders corresponding to dog breeds.

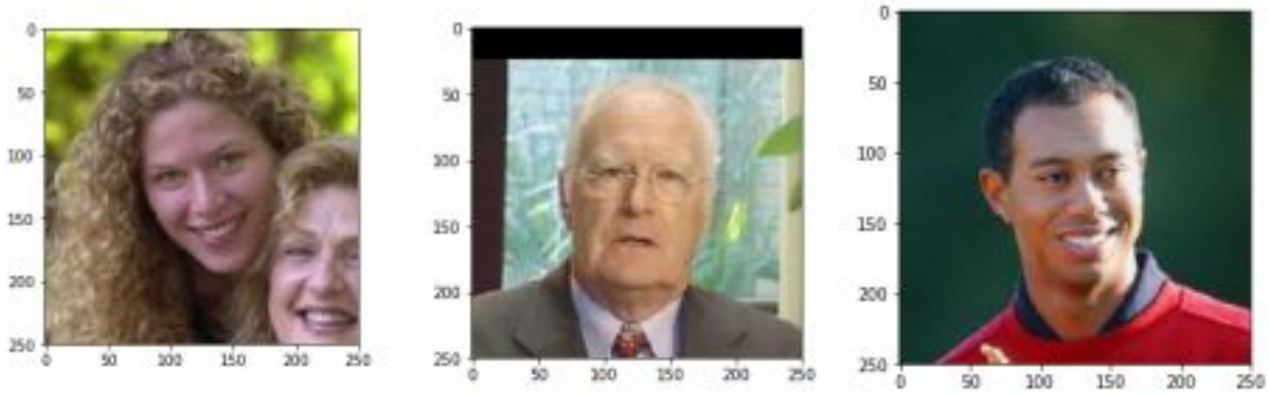
The images are of different sizes and different backgrounds, some images are not full-sized. The data is not balanced because the number of images provided for each breed varies. Few have 4 images while some have 8 images. Human images dataset: The human dataset contains 13233 total human images which are sorted by names of

humans (5750 folders). All images are of size 250x250. Images have different backgrounds and different angles. The data is not balanced because we have 1 image for some people and many images for some.

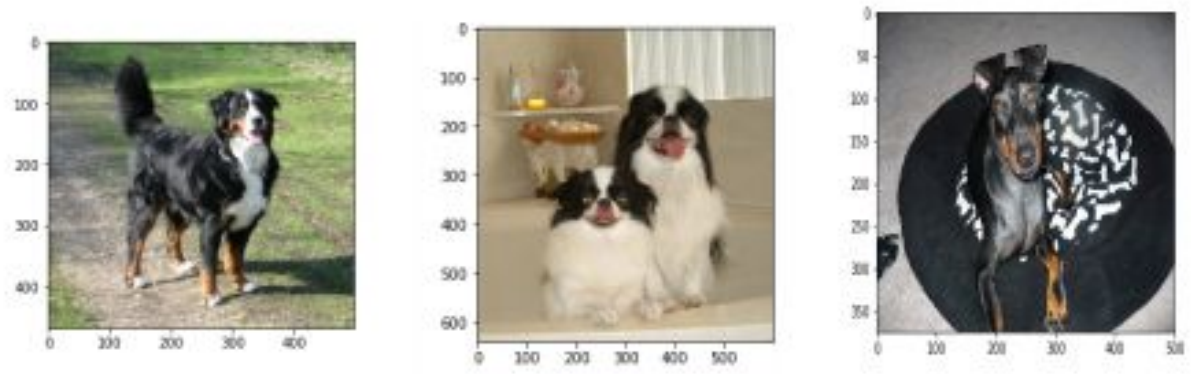
The random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imbalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Dog Dataset download link:
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

Human Dataset download link:
https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip



Sample image of humans



Sample Images of dogs

Algorithms and Techniques

The solution of the project follows six main steps.

First, we need to explore the datasets in order to understand how to use them and choose the proper algorithms.

Second, implement a Haar feature-based cascade classifier using OpenCV in order to detect faces in the human dataset.

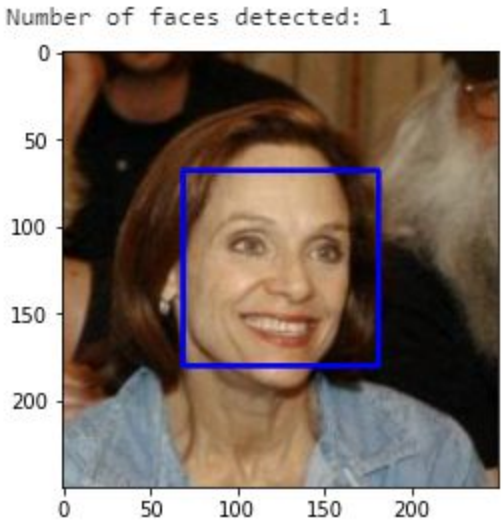
In the third step, I will use a pre-trained VGG16 model in order to detect dogs in the dog's dataset

Fourth, I will create a LeNet[2] like architecture that uses CNN in order to classify the 133 dogs breeds and have an accuracy greater than 10%.

In step five, I will use the transfer learning technique in order to a pre-trained ResNet50 architecture and continue the training with the dog's dataset. The minimum accuracy required is 60% on the test set.

Seventh, I will write a custom algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither.

In the last step, I will Test the Algorithm with some random images found online.



Benchmarks

The benchmark for the model can be referenced to the Kaggle leaderboard for dog breed identification competition. The target for this model is to reach a multiclass loss score of less than 0.01, which is in the top 100 of the competition. The other benchmark will be 90% prediction accuracy, which will be used as the upper limit. The benchmark set by Udacity will be 60% prediction accuracy, which will be used as the lower limit. The final performance of the model will sit in between the two limits.

Methodology

Data Preprocessing

The training, test, validation images are resized and center cropped into 224x224 pixels, then randomly flipped in the horizontal direction before transforming into tensors. The transformed data are organized into train, test, and validation directories, respectively. The corresponding reprocessing code blocks are shown in Figure 6. The reason for resizing, cropping is to achieve a uniform input data format. The random flipping increases the data variation in terms of features, thus making the dataset more robust.

Implementation

I have built a CNN model from scratch to solve the problem. The model has 3 convolutional layers. All convolutional layers have a kernel size of 3 and stride 1. The first con layer (conv1) takes the 224*224 input image and the final Conv layer (conv3) produces an output size of 128. The ReLU activation function is used here. The pooling layer of (2,2) is used which will reduce the input size by 2. We have two fully connected layers that finally produce 133-dimensional output. A dropout of 0.25 is added to avoid.

```
-----------
Epoch: 24
Training Loss: 4.110414         Validation Loss: 3.991432       Time: 3m 39s
Validation loss decreased (3.999423 --> 3.991432).  Saving model ...
-----------
Epoch: 25
Training Loss: 4.096873         Validation Loss: 3.966757       Time: 3m 39s
Validation loss decreased (3.991432 --> 3.966757).  Saving model ...
***********
Current time - 20:49:0
***********
-----------
Epoch: 26
Training Loss: 4.099367         Validation Loss: 4.040556       Time: 3m 40s
-----------
Epoch: 27
Training Loss: 4.080561         Validation Loss: 3.936662       Time: 3m 39s
Validation loss decreased (3.966757 --> 3.936662).  Saving model ...
-----------
Epoch: 28
Training Loss: 4.047440         Validation Loss: 3.951099       Time: 3m 39s
-----------
Epoch: 29
Training Loss: 4.035146         Validation Loss: 3.920111       Time: 3m 40s
Validation loss decreased (3.936662 --> 3.920111).  Saving model ...
-----------
Epoch: 30
Training Loss: 4.027959         Validation Loss: 3.947044       Time: 3m 40s
***********
Current time - 21:7:2
***********
```

Training epochs iterations

Refinement

Several structures have been tested with the scratch CNN model. Specifically, the number of filters (8 vs 16 in the first layer), pooling, and striding (striding=0 vs striding=2). The scratch model is trained using GPU mode in the Udacity container environment. The worst and best accuracy after 20 epochs is 8% and 11%, respectively. No drastic improvement is found using limited filter variations and pooling

strategies. Therefore, I conclude that the structure of scratch CNN is too basic for achieving higher accuracy. Because the accuracy metric achieved by the scratch CNN model is a far cry from the proposed accuracy (60%-90%) in this project. Therefore, using a more established pre-trained model makes sense. In this project, the DenseNet161 pre-trained model is used to serve as the real dog breed classifier. The DenseNet161 model is tuned to match the 133 class outputs for this project. The DenseNet161 model should theoretically produce much higher accuracy.

```
Validation loss decreased (0.498791 --> 0.479552).  Saving model ...

-----------
Epoch: 8
Training Loss: 1.148708         Validation Loss: 0.465031
Training Accuracy: 0.671707     Validation Accuracy: 0.862275
Current time - 21:52:0
Epoch Time: 4m 23s
Validation loss decreased (0.479552 --> 0.465031).  Saving model ...

-----------
Epoch: 9
Training Loss: 1.089020         Validation Loss: 0.444891
Training Accuracy: 0.689671     Validation Accuracy: 0.861078
Current time - 21:56:3
Epoch Time: 4m 23s
Validation loss decreased (0.465031 --> 0.444891).  Saving model ...

-----------
Epoch: 10
Training Loss: 1.030070         Validation Loss: 0.442452
Training Accuracy: 0.701647     Validation Accuracy: 0.863473
Current time - 22:0:5
Epoch Time: 4m 24s
Validation loss decreased (0.444891 --> 0.442452).  Saving model ...
```

Results after using some refinements

Model Evaluation and Validation Human Face detector:

The human face detector function was created using OpenCV's implementation of Haar feature-based cascade classifiers. 98% of human faces were detected in the first 100 images of the human face dataset and 17% of human faces detected in the first 100 images of the dog dataset. Dog Face detector: The dog detector function was created using a pre-trained DenseNet161 model. 100% of dog faces were detected in the first

100 images of the dog dataset and 1% of dog faces detected in the first 100 images of the human dataset. CNN using transfer learning:
The CNN model created using transfer learning with DenseNet161 architecture was trained for 5 epochs, and the final model produced an accuracy of 81% on test data. The model correctly predicted breeds for 680 images out of 836 total images.
Accuracy on test data: 81% (680/836)

Results

Justification

The best prediction accuracy I have achieved is 77% and the best test loss score is ~1.51. These results are still a fair distance away from the proposed performance. I have concluded the following ways to potentially improve the accuracy of my implementation.

1. Given more training time and epochs, the accuracy can be improved.
2. Larger input dataset for training can improve the accuracy of the model.
3. Manipulation on the training images to make the input dataset more robust.
4. Play with layer structures and hyperparameter tuning.
5. Examine the relevance of a different pre-trained model.

Neither!

Dogs Detected: Golden retriever

Dogs Detected: Basenji

Dogs Detected: Pomeranian

Neither!

Improvement

For this task, I think is possible to improve the model in several ways:

• try some different model architectures like ResNet, etc...

• hyper-parameters fine-tuning (change the optimizer, batch size, learning rate, etc...)

• improve de dataset with more images and more data augmentation

References

1. Paul Viola and Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2):137–154, 2004.

2. Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on, volume 1, pages I–900. IEEE, 2002.
3. ml-cheatsheet, https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html
4. VGG16 1000 categories, https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a
5. A Walkthrough of Convolutional Neural Network — Hyperparameter Tuning, https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network7f474f91d7bd
6. Overview of convolutional neural network in image classification, https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/