Capstone Project Proposal - Dog Classifier

Definition

Project Overview

Image classification is an extensively studied problem in the field of Machine Learning (ML). In this project, which is the Capstone project of Udacity's Machine Learning Engineer Nanodegree, I applied built several Machine Learning models to address the problem of image classification for dog breeds. Specifically, I built a pipeline to predict the breed of a dog if an image of dogs supplied by users and predict the closest resembling breed of dog if an image of a human is supplied by users. The pipeline contains a human and dog face classifier to detect whether a human or dog is in the image, and Convolutional Neural Network (CNN) — trained using transfer learning — to classify the human/ dog face accordingly. The training, validation, and testing data used to retrain the CNN was provided by Udacity.

Domain Background

Machine learning (ML) has become more prominent in business and scientific applications over the past decade, driven by advances in computing technology (e.g. GPUs) and the ML algorithms. A common application of ML is in the field of computer vision, in which machines are trained to interpret and understand the visual world. Using digital images from cameras and ML models, machines are used to identify and classify objects — and then react to what they "see."

This project focuses on the problem of classifying dog breeds. This problem has been extensively studied and has been the topic of past Kaggle competitions (i.e., Dog Breed Classification Competition) and numerous academic papers (e.g., Dog Breed Identification Using Deep Learning).

As a dog lover and owner, this application of ML aligns with my personal interest and I look forward to deploying an ML-powered web app that can be used by my friends and family.

Problem Statement

There are two main problems addressed by this project.

First, we classify user-provided images of dogs into the appropriate dog breed. Second, we classify user-provided images of humans and return the dog breed that the human bears the closest resemblance to.

This is a supervised learning problem — our dog images are divided into breed classes.

Capstone Project Proposal - Dog Classifier

Metrics

The performance of our human/dog face detector and dog breed classification CNN was measured based on each's *accuracy*. In this case, accuracy is calculated by:

Accuracy (%) = correctly classified images / all images classified

This accuracy calculation was performed on the human face detector, dog face detector, and the dog breed classification CNN. For the CNN, the accuracy score for the test images was used for model evaluation.

This is a suitable metric for evaluation as we are interested in how well the model performs in both face detection and dog breed classification.

Analysis

Data Exploration & Data Visualization

The project used two datasets provided by Udacity:

#1: Dog image data
8,351 total dog images, pre-sorted into training (6,680 images), testing (836 images), and validation (835 Images) folders. Within each folder, images are sorted into dog breeds. In total, we have 133 folders (i.e., dog breeds). The dog images are of different sizes, backgrounds, and angles which should provide enough variety for us to train on.

Looking at the training data, there seems to be a fairly even distribution between categories for the top 10 breeds and the bottom 10 breeds. The top breeds have 80-90 counts in the training data, whereas the bottom 10 breeds have 30-40 examines in the training data. This suggests sample imbalance will not be a problem in training the model.

Capstone Project Proposal - Dog Classifier

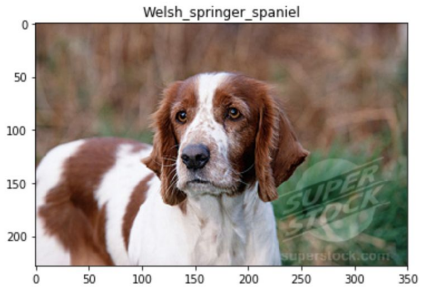
#	Dog Type	Count
0	Alaskan_malamute	95
1	Border_collie	92
2	Basset_hound	91
3	Dalmatian	88
4	Bull_terrier	86
5	Bullmastiff	85
6	Basenji	85
7	Cavalier_king_charles_spaniel	83
8	Australian_cattle_dog	82
9	Australian_shepherd	82
10	Irish_terrier	81

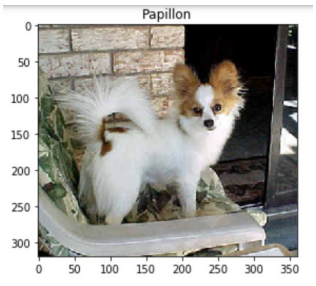
Top 10 dog categories in training data

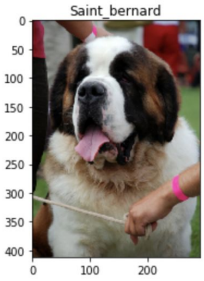
122	Neapolitan_mastiff	38
123	Petit_basset_griffon_vendeen	38
124	Parson_russell_terrier	37
125	Yorkshire_terrier	37
126	Smooth_fox_terrier	37
127	Saint_bernard	36
128	Wirehaired_pointing_griffon	36
129	Manchester_terrier	35
130	Plott	34
131	Norwegian_buhund	32
132	Xoloitzcuintli	32

Bottom 10 dog categories in training data

Below are sample images of different dog breeds from the dataset. As we can see, the images are different dimensions and some have borders. Hence, we will need to standardize the images for model training purposes.







#2: [Human face data]
13,233 total human images. These are sorted by name of the human (presumably). The human images are of the same size, but different backgrounds, positions, and angles.

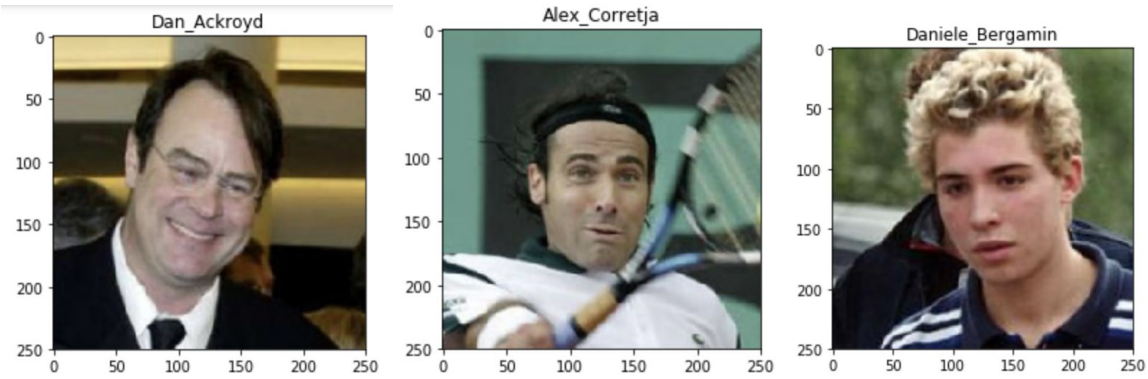
The human face data is fairly unbalanced. The top 5 people have significantly more images (i.e.,

100-500) than others (<50-100). However, since we are not training our human face detector on this data nor are we classifying humans, this imbalance should not pose a problem to our project.

#	Human Type	Count
0	George_W_Bush	529
1	Colin_Powell	235
2	Tony_Blair	143
3	Donald_Rumsfeld	120
4	Gerhard_Schroeder	108
5	Ariel_Sharon	76
6	Hugo_Chavez	70
7	Junichiro_Koizumi	59
8	Jean_Chretien	54
9	John_Ashcroft	52
10	Serena_Williams	51

Top 10 human category counts

Below are sample images of different people from the dataset. Since we are not training any models on this data nor does the OpenCV implementation have input size requirements, there is no resizing necessary on this data. However, we will need to transform these images to gray for use with the module.



Algorithms and Techniques

The pipeline is in two parts.

1. **Face Detectors:** I have created a dog detector and a human detector to detect whether there is a dog or human in the image. If the detectors do not detect either, then I return an error message.

   The dog detector is based on a pre-trained VGG-16, loaded from Pytorch. VGG-16 is suitable for this task as it has been trained on ImageNet data (~14 million images) and achieved an accuracy of 92.7%. By selecting the VGG-16 output categories for dogs (i.e., output index 151 to 268 inclusive), we effectively built a binary dog classifier as all

other indexes were classified as no dog.

The human detector is based on OpenCV's implementation of the Haar feature-based classifier (see link here).

2. **Dog Breed Classifier:** I created several Convolutional Neural Networks (CNN) to classify the dog breed for images that have dogs in them and closest resembling dog breed for images with a human in it. The CNNs will be trained in two ways: from scratch using a reference architecture and via transfer learning by taking the existing weights of a pre-train model (i.e., ResNet101). In the transfer learning instance, I retrained the last fully connected layer since I need to change the output classification from 1,000 (by default) to 133 (for 133 dog breeds in our dataset).

Benchmark

The benchmark for the Dog Breed Classifier CNN model created from scratch is an accuracy of greater than 10%. This is sufficient to confirm the model works because a random guess would give us <1% accuracy since there are 133 breeds (i.e., 1 divide by 133 = <1%).

The benchmark model for our Dog Breed Classifier CNN model created via transfer learning must have an accuracy greater than 60%, as per the project requirements.

Methodology

Data Preprocessing

The data preprocessing step for **training the model** was as follows:

1. Randomly crop and resize training images to 224 x 224 - this allows us to deal with occlusions and partial images, and standardizes our final input training images to 224x224.
2. Randomly horizontally flip and rotate training images - this serves as data augmentation and lets us deal with different positions and angles.
3. Normalize the training data (images) - this improves CNN performance and speeds up training

The data preprocessing step for **validation and testing the model** was as follows:

1. Resize training images to 224 x 224 - this is the input format required by ResNet models
2. Normalize the image - inline with the training step as the model is trained on normalized images

There was no need to crop or augment (e.g., flip, rotate) the validation and testing data since they are not being used to train the model.

Implementation

The **face detectors** were created using existing models. The dog detector is created using a pre-trained VGG-16, loaded from Pytorch. I classified the VGG-16 output categories for dogs (i.e., output index 151 to 268 inclusive) as “1” (i.e., dog detected) and all other categories as “0” (i.e., no dog detected). The human detector is created using OpenCV’s implementation of the Haar feature-based classifier.

The **dog breed classifier** involved creating two CNNs.

First, I created a CNN from scratch (“model_scratch)”. This model starts with 3 convolutional layers, each followed by a Relu activation and max-pooling layer. All convolution layers have a kernel size of 3x3, stride of 1, and padding of 1. The image size decreased by a factor of 2 with each convolution layer. Then, I flatten the image (28 * 28 * 128) for the first fully connected layer. After that, I applied a dropout layer (p = 0.25) and then a second fully connected layer.

Second, I created a CNN using transfer learning. To do so, I used a pre-trained ResNet101 from Pytorch’s library. To repurpose the model for our dog breed classification purposes, I retrained the final fully connected layer to output 133 classifications —rather than the original 1,000—in order to match the 133 dog breeds in Udacity’s dog database.

Refinement

I applied several refinements to the CNN model to reach the final accuracy of ~82%. Starting with the CNN built from scratch, I initially had a low accuracy (<10%) after training for 10 epochs. To improve this, I increased the learning rate (from 0.05 to 0.1) and epochs (from 10 to 20) as I saw that the validation accuracy was falling with each epoch. This lifted accuracy beyond the 10% benchmark for the CNN from scratch. However, I was not able to break the 15% limit. Hence, a transfer learning-based CNN was the logical solution.

The transfer learning-based CNN —built using ResNet101— far exceed the performance benchmark (which was 60%) at the first go. The model achieved an 82% accuracy after 10 epochs.

Capstone Project Proposal - Dog Classifier

Results

Model Evaluation and Validation

The **dog detector** achieved 100% accuracy based on 100 dog and 100 human images from Udacity's datasets. In other words, it identified dogs in 100 out of 100 images of dogs and did not identify any dogs in the human images.

```
In human images, incorrectly identified dog faces in  0  out of 100 images.
In dog images, correctly identified dog faces in 100  out of 100 images.
```

Dog detector results

The **human detector** achieved 98% accuracy based on 100 human images from Udacity's datasets. In other words, it identified humans in 98 out of 100 images of humans. However, it also generated 17% false positives when applied to 100 dog images from Udacity's datasets.

```
In human images, correctly identified  98  out of 100 images.
In dog images, incorrectly identified human faces in 17  out of 100 images.
```

Human detector results

Justification

The **dog and human detectors** are sufficient for the initial filter purposes. Since the dog detector was far more accurate (i.e., no false negatives nor false positives), my pipeline starts with applying the dog detector first. If no dog is detected, then a human detector is then applied to look for humans.
My final CNN—based transfer learning on ResNet101—has an accuracy of 82%, which is far above the 60% requirement for the project.

Future Direction and Improvements

While 82% accuracy is good for the dog breed classification CNN, it is still resulting in roughly 2 out of 10 mistakes. This is fairly significant from a user experience perspective and warrants further improvement. For a consumer-facing application, I would want to refine the model to reach 90-95% accuracy.

Potential ideas to improve the model include:

1. Training for longer (i.e., more epochs)
2. Experimenting with different hyperparameters (e.g., learning rates) and optimizers (E.g., Adam) 3)

3. Experimenting with different pre-trained models (e.g., Inception_v3, VGG-16...)