Project Report

Dog Breed Classifier

Overview

The project was the capstone project of Udacity's Machine Learning Engineer nanodegree.

The aim of the project in the Machine Learning Engineer is to create a model that is able to **identify a breed of dog** if given a photo or image as input. If the photo or image contains a human face , then the application will return the breed of dog that most resembles this person or alien .

The strategy laid out for solving this problem, given in the notebook provided by Udacity, is as follows:

- Step 0: Import Datasets
- Step 1: Detect Humans
- Step 2: Detect Dogs
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
- Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 6: Write your Algorithm
- Step 7: Test Your Algorithm

Step 0: Import Datasets

The datasets are provided by Udacity through the following links.

- dog images for training the models
- human faces for detector
- all other downloads to ensure smooth running of the notebook are available in the repository.

The first thing we do is load all the libraries and packages that have been used throughout the notebook.

As you can see, we have used extensively from keras for creating the CNN, we have also used sklearn for dataset loading, OpenCV and PIL for image work, matplotlib for viewing the images and numpy for processing tensors.

tqdm provides a smart progress meter so you can see how your for loops are progressing and glob is used to find all pathnames matching a specified pattern

After loading our libraries, we use the load dataset function from sklearn to import our datasets for our dog breed model training.

The dog_names variable stores a list of the names for the classes which we will use in our final prediction model. Depending on the path for where you have the images you may need to change the 35 in the item[35:-1] to a bigger or smaller number.

If everything has worked you will see 133 different dog breeds and ~8.3k dog images.

Step 1: Detect Humans

```
# load filenames in shuffled human dataset
human_files = np.array(glob("../../../data/lfw/*/*"))
```

We use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. OpenCV provides many pre-trained face detectors. The code below instantiates the Haar Cascade Classifier from OpenCV and is then used in the face detector function to determine if a supplied image contains a face or not.

```
# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')# returns "True" if face is detected in image stored at img_path
```

- Humans correctly identified: 96%
- Dogs recognised has humans: 17%

The results are not perfect but acceptable.

Step 2: Detect Dogs

For the dog detector we have used the pretrained VGG-16 model. To use this model with our images, we need to process our images into the correct tensor size for the model. This is often

one of the most challenging parts in the image classification process, the preprocessing of the images.

We used the normalized data as the input to the model

Now we are ready to make predictions. After completing the preprocessing steps, uses the predict function to predict.

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

This is a very interesting exercise to do and although it is not used in the final web application it is very useful for understanding CNNs and how they work. We will be utilising transfer learning for the final image detector.

The full code is in the notebook and you can follow along experimenting and creating your own code from scratch.

The network I chose was close to that presented in the class which consisted of 5 convolutional layers with 1 max pooling layers to reduce the dimensionality and increase the depth.

I added a couple of fully connected layers with the final layer having 133 nodes to match our classes of dog breeds.

Dropouts were added to reduce the possibility of overfitting. The default settings with Adam were used as the optimizer for the loss function.

Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning)

In this section of the Jupyter notebook, we are walked through using one of the pretrained networks available for use with keras.

The most important concept to understand, especially if you have been used to using PyTorch for transfer learning is the use of bottleneck features.

Bottleneck features is the concept of taking a pre-trained model in our case here VGG16 and chopping off the top classifying layer, and then inputing this “chopped” VGG16 as the first layer into our model.

The bottleneck features are the last activation maps in the VGG16, (the fully-connected layers for classifying has been cut off) thus making it now an effective feature extractor.

I haven’t included any code here as we will follow the same process in step 5 with the Resnet50 pretrained model used for transfer learning. Check out the code in the Jupyter notebook.

Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning)

Ok, so let's build our model in keras.

I decided to use the VGG16 model.

The variables contain images that have already been put through the bottleneck extractor. This will make the training of our model very quick, as we are applying images where the main features for identification have already been isolated. This means we will have only a small number of parameters or weights to backpropogate through.

Our fully-connected layers will now just correlate the patterns for our 133 classes coming from the bottleneck features in order to train, validate and test.

Next we need to design our model. The design information can be seen in the notebook itself. So after designing we need to train our model. And After that testing is done, in which our model performed well.

It got a accuracy of 77% which is good.

Step 6: Write your Algorithm

Here we are creating our algorithm to analyze any image. The algorithm accepts a file path and:

- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide output that indicates an error.

The algorithm collects together functions we used previously to create a final output and shows the image.

Step 7: Test Your Algorithm

The results were pretty good for the images the model was shown. All the images were correctly identified, but a more robust testing of dog breeds would be required.