Data Scientist Nanodegree
Convolutional Neural Networks
Project: Write an Algorithm for a Dog Identification App

Classify images of dogs according to their breed.

Feddah Alotaibi

Project Definition

Project Overview

The project aims to classify images of dogs according to their breed using deep learning neural network as part of Udacity's Data Scientist Nano Degree. In this project, I developed an algorithm that could be used as part of a mobile or web app. The code will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling. The image below displays potential sample output of your finished project.

Problem Statement

Preforming image classification of various breeds of dogs using deep learning, conventional neural network. Developing an algorithm that detects whether the image provided is of human or dog. Not only detecting humans and dogs in images, but also predicting dog breed from images. Using CNN that classifies dog breeds.

Metrics

Because the problem is classification so using classification accuracy is suitable.

Analysis

Data Exploration

There are 133 total dog categories in dataset of 8351 dog images
Splitting the data into three sets/
There are 6680 training dog images.
There are 835 validation dog images.
There are 836 test dog images.

Methodology

The steps 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
In this project we have used Keras to build our Convolutional Neural Network (CNN) to make the dog predictions, though it would also be possible to use PyTorch, which we have used in a previous project in this Data Science nanodegree to classify flower species from a given image.

Data preprocrssing

To start working on the images, we need to process them 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.
Using Keras, the images had to be in four dimensions, formatted in a way that (number of images, row size of image in pixels, column size of image in pixels, number of color channels).
In the path_to_tensor function we are processing a single image, so the output is (1,224,224,3), where 1 image, 224 pixels wide, 224 pixels high, and 3 colours red, green and blue
The function paths_to_tensor then stacks the images returned from path_to_tensor into a 4D tensor with the number of images from training, validation or test datasets depending on which img_path is called.

- path_to_tensor(img_path): used to load RGB image as PIL.Image.Image type
- image.load_img(img_path, target_size=(224, 224)): used to convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
- image.img_to_array(img): used to convert 3D tensor to 4D tensor

Impelementation

Implemented using Keras transfer learning VG16 and Inception pre trained model
Refinement

To get final CNN architecture the following steps were followed

- Create convolutional layers by applying kernel/ feature maps
- Apply Max pool for translational invariance
- Flatten the inputs
- Create NN
- Train the model
- Predict

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 223, 223, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 54, 54, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 27, 27, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 46656)             0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               9331400
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               26733
=================================================================
Total params: 9,368,677
Trainable params: 9,368,677
Non-trainable params: 0
```

- The first model achived 8.0144% accuracy

Transfer Learning:

Why Does Transfer Learning Work So Well ?

To learn why transfer learning works so well, we must first look at what the different layers of a convolutional neural network are really learning.

When we train a deep convolutional neural network on a dataset of images, during the training process, the images are passed through the network by applying several filters on the images at each layer. The values of the filter matrices are multiplied with the activations of the image at each layer. The activations coming out of the final layer are used to find out which class the image belongs to.

When we train a deep network, out goal is to find the optimum values on each of these filter matrices so that when an image is propagated through the network, the output activations can be used to accurately find the class to which the image belongs. The process used to find these filter matrix values is gradient descent.

When we train a conv net on the imagenet dataset and then take a look at what the filters on each layer of the conv net has learnt to recognize, or what each filter gets activated by, we are able to see something really interesting.

Thus, to reduce training time without sacrificing accuracy, a CNN using transfer learning was used.

The model uses the the pre-trained VGG-16 model as a fixed feature extractor, where the last convolutional output of VGG-16 is fed as input to the model. A global average pooling layer and a fully connected layer were added, where the latter contains one node for each dog category and is equipped with a softmax.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 133)               68229
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
_________________________________________________________________
```

However, the accuacry for this model was poor it only achieved 41.5072%

The next model, I used a different bottleneck feature from a different pre-trained model.
Inception bottleneck features was used

In Convolutional Neural Networks (CNNs), a large part of the work is to choose the right layer to apply, among the most common options (1x1 filter, 3x3 filter, 5x5 filter or max-pooling). All we need is to find the optimal local construction and to repeat it spatially. he goal of the inception module is to act as a "multi-level feature extractor" by computing 1×1, 3×3, and 5×5 convolutions within the *same* module of the network — the output of these filters are then stacked along the channel dimension and before being fed into the next layer in the network. The subsequent manifestations have simply been called *Inception vN* where N refers to the version number put out by Google. The weights for Inception V3 are smaller than both VGG and ResNet, coming in at 96MB.

Inception architecture:

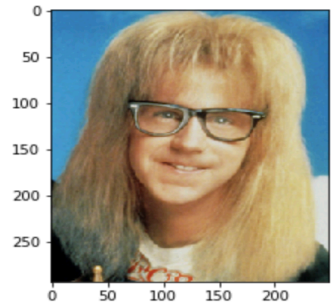
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 133)               272517
=================================================================
Total params: 272,517
Trainable params: 272,517
Non-trainable params: 0
_________________________________________________________________
```

The test accuracy obtained was 79.9043% Which is good

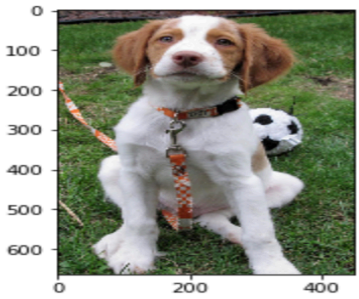
Result

Using transfer learning with inception have improved the model, the test accuracy obtained was 79.9043% Which is good

```
hello, it's a human!
You look like a ...
ages/train/082.Havanese
```



```
hello, it's a dog breed!
your predicted breed is ...
ages/train/037.Brittany
```



```
hello, it's a dog breed!
your predicted breed is ...
ages/train/009.American_water_spaniel
```

As we can see the model was able to correctly perform required tasks alos answering the following questions. What kind of dog does the algorithm think that __you__ look like? If you have a dog, does it predict your dog's breed accurately? If you have a cat, does it mistakenly think that your cat is a dog?

Conclusion

In conclusion, the model obtained nearly 80% Which is good, however there are more area that would have improved the model if taking into consideration.

I think the model would have been improved by training more data. Also, performing data augmentation through the use of rotation, rescale and other transformation and also to prevent overfitting. Optimizing number of layers would have improved the neural network as well

In addition to performing hyperparameter optimization. Considering the above I'm sure we could increase the testing accuracy of the model to above 90%.