Dog Breeds Classifier - Udacity DSND Capstone Project

Renee Shiyan Liu

Abstract

This project utilizes Convolutional Neural Network — CNN — technique to train a deep learning model through an existing set of 8351 dog images, where the dogs are pre-classified into 133 unique breeds. Specifically in this project, I used transfer learning technique where Keras' InceptionV3 model is employed as the base with a custom-defined classifier layer to build a new dog image classifier. This algorithm first uses human and dog face detectors to differentiate if the incoming image is a dog or a human face, then classifies it into certain breed. The final product of this project is a web application deployed using FLASK to provide users an interesting interactive experience where a user can get a breed prediction with 79% accuracy after uploading a dog or non-dog image.

Keywords: Deep Learning, Image Classification, CNN

---

1. Project Definition

1.1. What is the core task of this project?

The core task of this project has three main steps as follows

- First, use an existing CNN model as the feature extraction layer via transfer learning technique before adding the last classifier layer

- Second, use the existing set of dog images to train this newly obtained model with a reasonable accuracy

- The last step is to deploy this trained model as a web application using Flask. The end goal of this project is to allow a user to upload her favorite image, either a dog or a non-dog, and predicts for her what the dog breed is. If it is human photo, the application tells her what dog breed this human mostly resemble with.
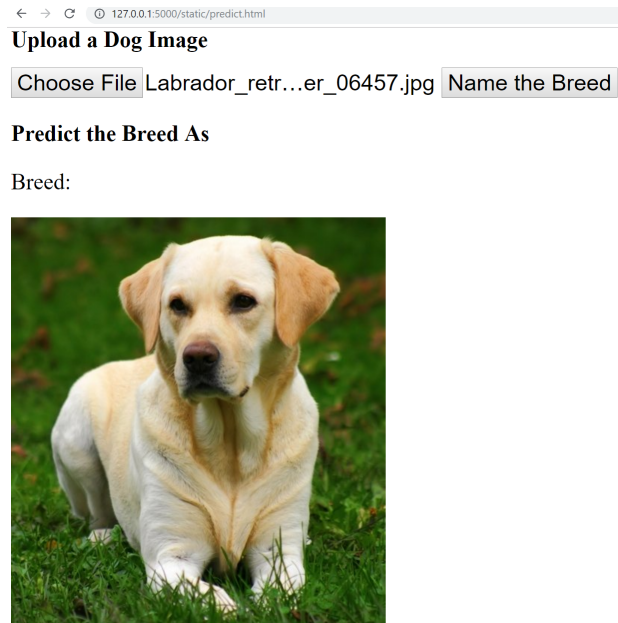
Figure 1: Project Web Application UI

1.2. Evaluation Metrics

After training the model using the training data set (6680 out of 8351 total images) with validation data set (835 out of 8351 total images), accuracy score is calculated on the test data set (another 836 images). The accuracy score is the metrics of this algorithm. Overall, the algorithm in this project achieved a 79% accuracy.

I chose to use accuracy as the model metrics for two reasons. The first reason is that it is very intuitve and straightforward especially in image classification space. The score tells you directly how the model performs. The second reason is that the training data has relatively balanced classes. In other words, there are reasonably amount of dog representations in each breed. Under relatively balanced classes, using accuracy is completely justifiable.

It is definitely not a perfect metrics because the classes are not entirely uniformly distributed but they are neither severely imbalanced. (As the histgram shows in the following section)

2

2. Analysis

2.1. Data Exploration and Visualization
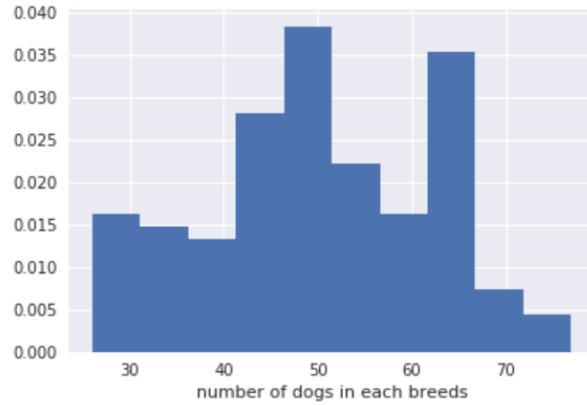
Raw images in the training set are of different sizes. This implies that pre-processing the images into uniform sizes and standard shape are needed.

3. Methodology

3.1. Data Preprocessing

As suggested in the previous section, three preprocessing steps are done before using any Keras pre-trained model. The three steps are as follows,

1. reordering the color channels from Red, Green, Blue (RGB) to Blue, Green, Red (BGR).
2. resizing any incoming image to 224 by 224 pixels.
3. Normalization. This step is required if we want to employ pre-trained Keras models such as ResNet-50 and InceptionV3. In this extra step, each pixel from an image must be subtracted by the mean pixels (expressed in RGB order) which are calculated from all the image.

3.2. Implementation

In order to build a Image classification model, we need to first refine the features of each incoming image. The is to say that we have to "pick" the features that mostly capture the "soul" of an image.

This is especially important and necessary if the predictors are image pixels because they are very large in size after being reshaped into pixels.

For example, a 224 by 224 image will have 224×224 features. This idea is very similar to dimension reduction techniques such as PCA. The end product is called bottleneck features.

The best method to achieve this goal is to use Convolutional Neural Network (CNN) to do feature extraction. Why? It is because images contains lots of pixels that make dense layers very slow. CNN comes in handy because it doesn't require the model to start with looking at each single pixel at the beginning. Instead, it uses patterns to "look" at the whole image and look for features first. This largely reduces the number of inputs for the model to train efficiently.

Noted, in practice (as it is in this project), we don't train the CNN model from scratch because it takes too long and we won't have enough training images to obtain a satisfying model. Thus, I use transfer learning technique to directly obtain the infrastructure of the feature extraction layer for my model.

Specifically, I used InceptionV3 model from Keras as my feature extractor. This is done through directly downloading the corresponding bottleneck features for training, validating, and test data set prepared by Udacity. The custom-classification layer – the last layer – is build up using a GlobalAveragePooling2D layer, two dropout layers and two Dense layers.

GlobalAveragePooling2d layer is to retain the most important features among the bottleneck features that are fed in. This step is crucial because it largely reduces the complexity this model is dealing with. Then, dropout layers prevent overfitting because I want to achieve a good accuracy in the testing stage. Dense layers are to make sure that the model takes into account all the information provided by the refined bottleneck features.

This architecture is suitable to this Image Classification task because we need accuracy and we have already largely reduced the dimensionality of the input features. See the model summary below.

I then compiled the model using accuracy as the model performance metric and a RMSprop optimizer. I used 20 epochs to train the data, the validaton accuracy is 82%. Finally, I saved the weights of the trained model for re-use in the future.

3.3. Solution Refinement

I decided upon 20 epochs because it gives me the best validation accuracy among other choices. However, if more time and computational powers are

4

```
Layer (type)                    Output Shape              Param #
=================================================================
global_average_pooling2d_1 (    (None, 2048)              0
_____
dropout_1 (Dropout)             (None, 2048)              0
_____
dense_1 (Dense)                 (None, 1024)              2098176
_____
dropout_2 (Dropout)             (None, 1024)              0
_____
dense_2 (Dense)                 (None, 133)               136325
=================================================================
Total params: 2,234,501
Trainable params: 2,234,501
Non-trainable params: 0
```

Figure 2: Model Architecture Summary

allowed, I am sure I can find a better number of Epochs to gain even higher accuracy.

Similar experimentation can be done on finding a better learning rate and optimizer. In this project, I used 0.001 as my learning rate and RMSprop as my optimizer. I can try Adam or other optimizer later on.

4. Results

The trained model achieves a 78.9% of accuracy when used on the test data set. Overall, a satisfactory result.

Aside from the two points I have discussed in the previous section on how to improve the model, we can also try to use other transferred CNN model to compare which one provides higher accuracy.

5. Conclusion

To recapture the pipeline for this project,

1. Preprocess an incoming image into a 4D array that is suitable to be fed into Keras Model.
2. Use InceptonV3 as the transfer-learning CNN layers to extract bottle-neck features.
3. Build a custom classifier features using Dense layers.
4. Train and validate the model.

5. Save the trained model weights for future use.
6. Reload the model with the saved weights to check its accuracy on the test data set.

What I found particularly difficult about this project is to make the correct input dimention for the image to be ready to use. Another difficult part is how to understand the large amount of parameter values. I think I need to find more resources to understand how to use them than just storing them.

As this is an end-to-end project, the most challenging part is actually figure out how to deploy this algorithm successuflly online. I have trouble making the connection between the backend model and front-end user input.

How do I want to improve this project in the future? Besides trying out different pre-trained model and tinker with different hyperparameter values, I want to figure out a way to filter the incoming image so that each image contains mostly the object of interest. For example, if a supplied image is a dog running in the distance, I want to be able to transform the image such that the dog image can be blown up or the irrelevant parts of the image can be deleted at first.

6. References

1. [Classification Metrics](#)