Capstone Project Ali UPT

Data Science Nanodegree November 18th, 2021

Project Overview

This project uses Convolutional Neural Networks (CNNs)! In this project, I built a ML algorithm to process real-world, user-supplied images. By giving an image, the algorithm gives an estimate of whether it is a dog and its breed or it is a human face or neither. If supplied an image of a human, the code will also identify the resembling dog breed. This is a multi-class classification ML problem that use supervised machine learning to unravel the patterns.

Problem Statement

The ultimate goal of this project is to build a ML algorithm that can be fed any images so that it can tell us if it is a human, dog or unrelated object.

The algorithm has two detectors:

Dog face detector: Given an image of a dog, return the estimate of the predicted breed.

Human face detector: If supplied an image of a human, return the resembling dog breed.

Data Exploration

For the classification purpose, the input data must the images of both human, and dogs so that the algorithm identify the breed of the dogs and also human and their resembling dog breeds.

I used the following two datasets for training the models:

- The dog dataset = 8351 images with 133 classes

  This dataset included 6680, 835, 836 images for training, validation and test

- The human dataset = 13233 images

I imported a dataset of dog images. I populated a few variables through the use of the load_files function from the scikit-learn library:

train_files, valid_files, test_files - numpy arrays containing file paths to images

train_targets, valid_targets, test_targets - numpy arrays containing onehot-encoded classification labels

dog_names - list of string-valued dog breed names for translating labels

Subsequently, the OpenCV's implementation of Haar feature-based cascade classifiers was used to detect human faces in images. For detecting different breed of dogs, I used a pre-trained ResNet-50 model to detect dogs in images. The provided block of codes, first downloads the ResNet-50 model along with the weights on the trained ImageNet.

The images have 3 channels, red, green, and blue. They are normalized using a mean of 0.485, 0.456, and 0.406 respectively, and a standard deviation of 0.229, 0.224, and 0.225 respectively. These values have been taken from the PyTorch documentation. What this essentially does is it changes each value in the image matrix as

$$image = \frac{image - mean}{std}$$

Normalization helps get data within a range and reduces the skewness which helps learn faster and better.

Evaluation Metrics

To evaluate the multiclass classifier, I used the Accuracy as the base of evaluations. By calculating, how far each prediction is from the true label, the algorithm learns from each prediction and minimize the distances in each iteration.

Accuracy=Number of items correctly classified/ All classified items

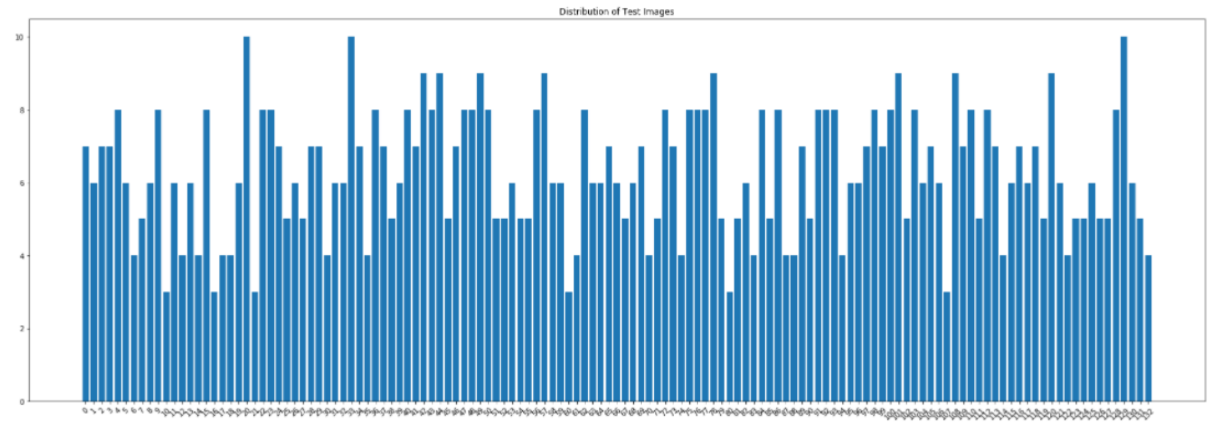


Figure 1. The class distribution of the datasets.

The base metric used for model evaluation is often Accuracy. This metric useful when we have symmetric datasets. Given that we had this property in our dataset, I decided to use this evaluation metric as mentioned above to determine the model performance.

Data Visualization

The dataset contains images that mainly are a single portrait of the dogs with the associated breed or single portrait of human.

However, some images are complicated given that there are multiple objects in one shot, which make the detection process by the classifier extremely tough.

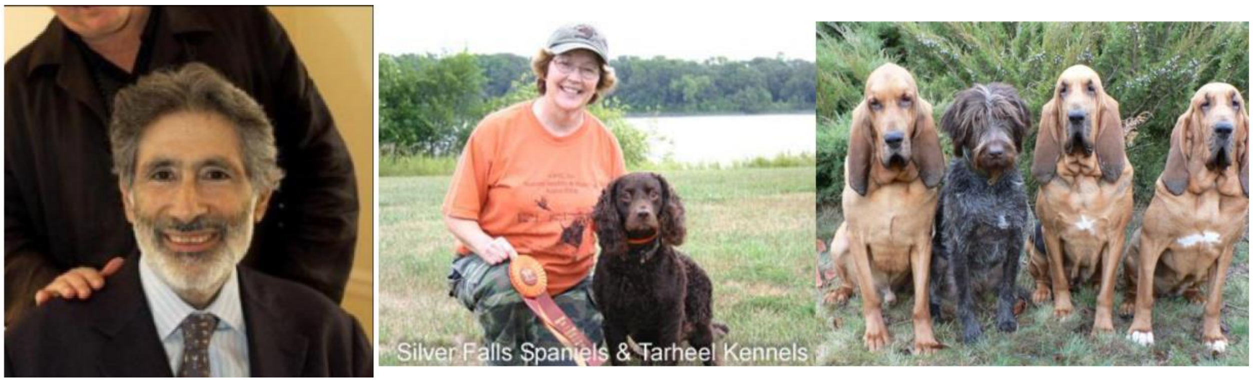


Figure 3. Sample images of input data set.

It is worth mentioning that some of the breed such as the ones given below are so similar and even human will have difficulties to recognize the breed. Therefore, image recognition is the challenging task even by classifier algorithm:

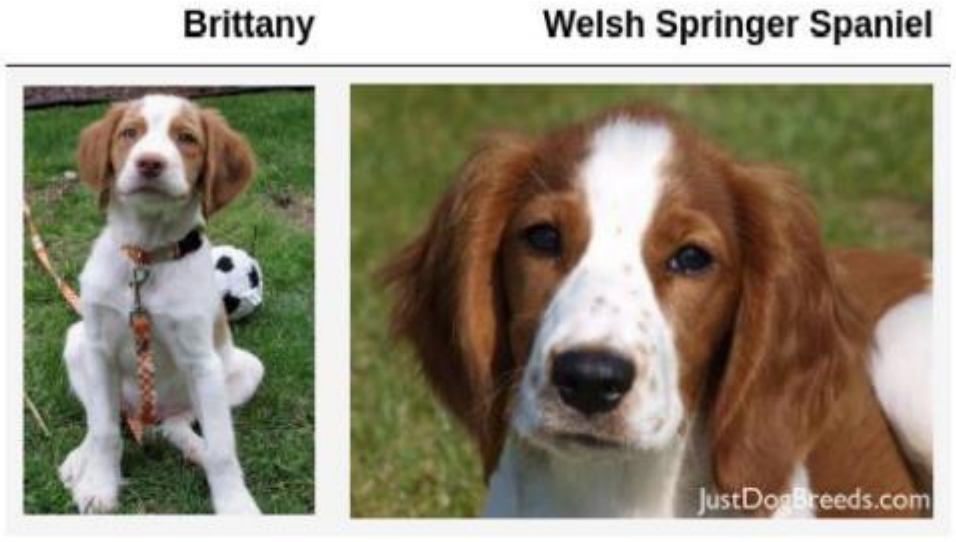


Figure 4. Sample images of input data set.

Methodology and analysis

The convolutional neural networks (CNNs) are the class of deep learning neural networks. CNNs are the one of the robust techniques out there for image recognition. They are widespread techniques in the visual imagery and are mostly used in the image classification many apps. They are used in most commonly in Facebook's photo tagging to self-driving cars. In this project, first I used the available algorithms like OpenCV's implementation of Haar feature based cascade classifiers for the human image recognition. Subsequently, I used a pretrained VGG16 model and evaluated its performance by accuracy metric. Lastly, I built a CNN classifier images from the scratch and used the Accuracy, Precision, Recall and F1 score to evaluate the model performance.

Benchmark

After defining the functions that can detect human and dogs faces. Subsequently, I created a CNN that classifies dog breeds. We built a CNN model from the scratch which attained the test accuracy of at least 1%. The CNN model created from scratch must have accuracy of at least 10%. This can confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Data Preprocessing

It is worth noting that we used images of 224*224 as an input and normalization technique was applied for train, valid and test datasets. Additionally, augmentation technique was applied on training dataset to make sure that we reduced the overfitting on the dataset. Subsequently, we used random rotation and horizontal flip of images so that we eliminate the selection bias. Lastly, we converted images into tensor passed it to the classifier algorithm.

Implementation

For the classification purpose, I constructed a CNN with 3 convolutional layers along with 3 max pooling layers. I used max pooling layer to reduce the dimensionality (reduce the input size). Additionally, I increased the intralayer filter size from 16 to 64 to boost the accuracy. I used Relu as an activation function to improve neural networks efficiency by speeding up training. I also used a dropout of 0.4 to prevent the overfitting.

Refinement

Even through the model that I built has reasonable accuracy and met the benchmarking requirements but I used transfer learning to create a CNN, which capable of detecting dog breed from raw images. By defining an appropriate architecture of model, I could obtain the accuracy of around 85%. In this step, I used Xception bottleneck features from a pre-trained model. In this step, I used transfer learning to train a CNN in order to reduce training time without sacrificing accuracy

```
I predicted this to be a human
 But if you were a dog, I guess you would be a ... ages/train/012.Australian_shepherd!
```

```
Hum...this is a tough one! I guess this is neither dog nor human. It must be something else!
```

```
It should be a dog!
 And I guess you would be a ... ages/train/006.American_eskimo_dog!
```

Figure 5. Sample outputs predicted using the model.

Model Evaluation and Validation

For detecting human faces, I created human face detector function using OpenCV's implementation of Haar feature based cascade classifiers and the following result were obtained:

Percentage of the human faces in first 100 human files: 100%

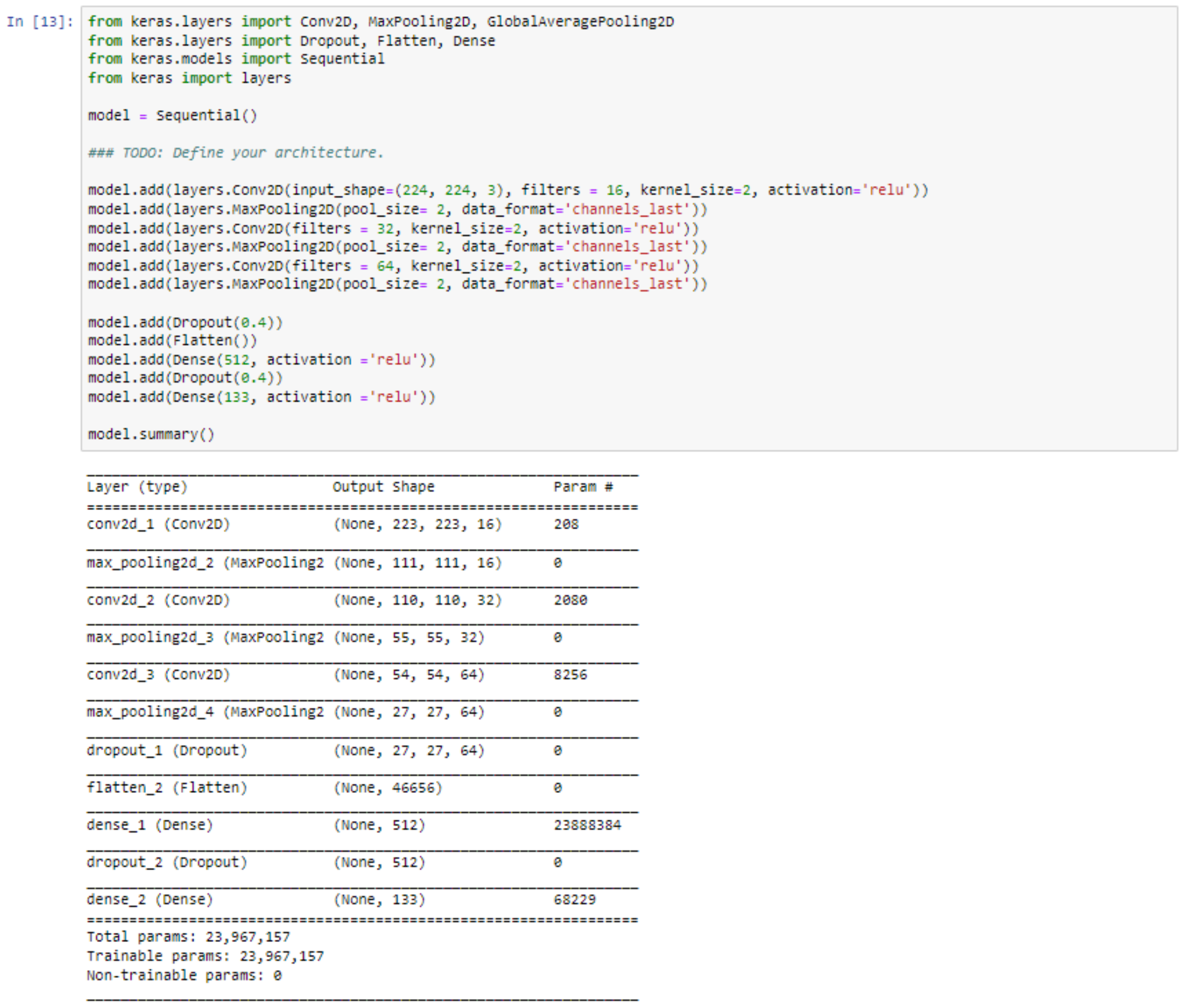
Percentage of the human faces in first 100 dog files: 11%

For detecting human faces, we created pre-trained VGG16 model and the following result were obtained:

Percentage of the dog faces in first 100 human files: 0%

Percentage of the dog faces in first 100 dog files: 100%

CNN using transfer learning: The CNN model was created using transfer learning with ResNet101 architecture was trained for 20 epochs, and the final model produced an accuracy of 85% on test data. For the classification purpose, I constructed a CNN with 3 convolutional layers along with 3 max pooling layers. I used max pooling layer to reduce the dimensionality. Additionally, I increased the intralayer filter size from 16 to 64 to boost the accuracy. I used Relu as an activation function to improve neural networks efficiency by speeding up training.



In [13]:

```python
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential
from keras import layers

model = Sequential()

### TODO: Define your architecture.

model.add(layers.Conv2D(input_shape=(224, 224, 3), filters = 16, kernel_size=2, activation='relu'))
model.add(layers.MaxPooling2D(pool_size= 2, data_format='channels_last'))
model.add(layers.Conv2D(filters = 32, kernel_size=2, activation='relu'))
model.add(layers.MaxPooling2D(pool_size= 2, data_format='channels_last'))
model.add(layers.Conv2D(filters = 64, kernel_size=2, activation='relu'))
model.add(layers.MaxPooling2D(pool_size= 2, data_format='channels_last'))

model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(512, activation ='relu'))
model.add(Dropout(0.4))
model.add(Dense(133, activation ='relu'))

model.summary()
```

```
_________________________________________________________________
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         
_________________________________________________________________
dropout_1 (Dropout)          (None, 27, 27, 64)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 46656)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               23888384  
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 133)               68229     
=================================================================
Total params: 23,967,157
Trainable params: 23,967,157
Non-trainable params: 0
_________________________________________________________________
```

Justification

Since the accuracy tremendously got improved by around 85% compared to the initial architecture using transfer learning approach, using this method can be justified.

Improvement

Defiantly, the model can be improved by increasing the size of input data, especially for the training dataset. Using different CNNs architecture paired with transfer learning approach can also be considered for model performance improvement.

Reflection

For conducting this project, I followed the steps given below:

- Step 0: Dataset exploration
- Step 1: Detect Humans using a Haar feature-based cascade classifiers
- Step 2: Detect Dogs using a pretrained network
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch).
- Step 4: Create a CNN to Classify Dog Breeds using Transfer Learning and a using a ResNet50 architecture.
- Step 5: Write a custom Algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither.
- Step 6: Test the Algorithm with some random images found online.

The abovementioned steps were taken to build and analyze data structure and model performance. Please note that the app performance can be significantly improved by increasing the size of the dataset and trying out different algorithm.

.

References:

https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/

https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2

https://en.wikipedia.org/wiki/Precision_and_recall

The detailed data analysis for this project can be found in my GitHub Link.