Machine Learning Engineer Nanodegree

Capstone Project

Dog Breed Classifier

Jingxian Lin

April 20th, 2020

I. Definition

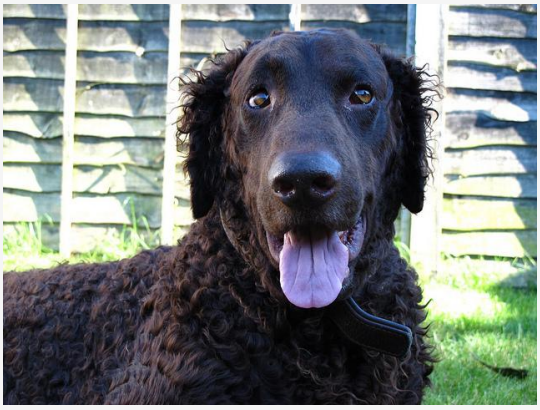
Project Overview

The task of assigning breed to dogs from images is considered exceptionally challenging. To see why, consider that *even a human* would have trouble distinguishing between a Brittany and a Welsh Springer Spaniel.

Brittany	**Welsh Springer Spaniel**
	

It is not difficult to find other dog breed pairs with minimal inter-class variation (for instance, Curly-Coated Retrievers and American Water Spaniels).

Curly-Coated Retriever **American Water Spaniel**




Likewise, recalling that convolutional neural networks can be used to classify images with high accuracy and at scale, the goal of this capstone project is to apply deep learning techniques to the classification of dog breeds.

Problem Statement

The main objective of this project will be to build a pipeline to process real-world, user-supplied images. Given an image of a dog, your algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed.

A strategy for solving this problem is to try using different neural network architectures such as Convolutional Neural Networks (CNNs) from scratch and using Transfer Learning based on various pre-trained models, with the target to attain a test accuracy of at least ten times better than the benchmark.

Metrics

The evaluation metric that can be used to quantify the performance of both the benchmark model and the solution model is the accuracy score.

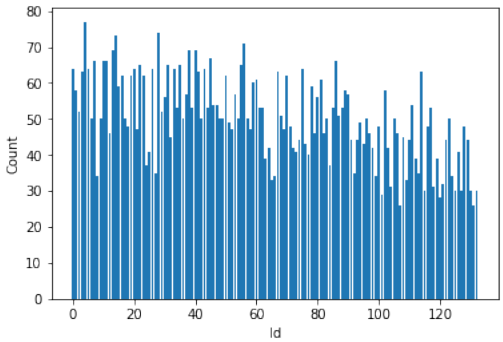
As a matter of justification, plotting the count of each dog breed in the next section demonstrates that the dataset is relatively symmetrical.

II. Analysis

Data Exploration

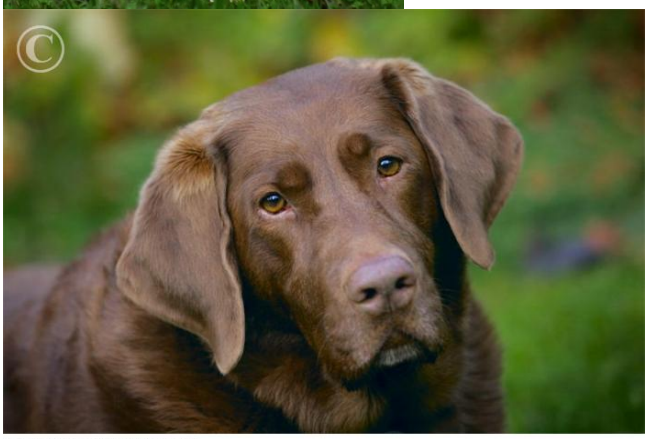
Both the human (LFW) dataset and dog dataset are provided by Udacity. There are 13233 total human images and 8351 total dog images with 133 total dog categories.

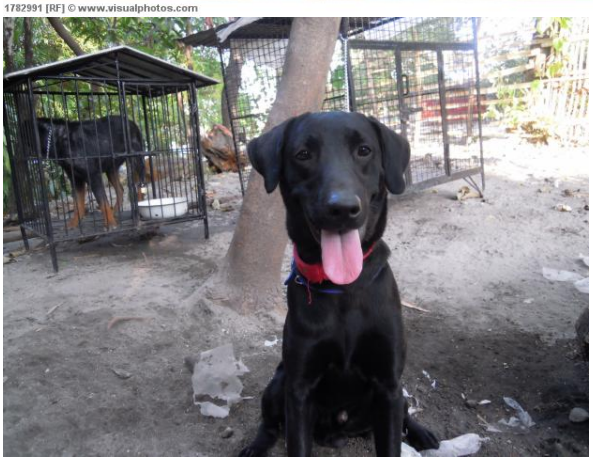
The label distribution is shown below, and the resolution of dog images varies a lot, listing several examples here: 375 x 500, 487 x 600, 640 x 613, which is pretty clear in the exploratory data analysis session. Resizing the images is generally required before feeding data into the model, because CNN structure presumes that the input would have a certain shape, like 224 x 224 x 3, an important procedure.



Exploratory Visualization

Sample images from the downloaded dog dataset are presented below. Their various sizes need to be considered in preprocessing steps.





Algorithms and Techniques

Use transfer learning to create a CNN that can identify dog breed from images. This way would reach higher accuracy than the following benchmark model, because the architecture is based on the pre-trained model as a feature extractor, and a global average pooling layer and a fully connected layer are added.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1   (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               68229
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
```

The layers in CNNs are organized into three dimensions, width x height x depth, and the nodes in one layer do not necessarily connect to all nodes in the subsequent layer, but often just a sub region of it. This allows the CNN to perform two critical stages: One is the feature extraction—a filter window slides over the input and extracts a sum of the convolution at each location then stored in the feature map, a pooling process is included between CNN layers where typically the max value in each window is taken, decreasing the feature map size but retaining the significant data, this reduces the dimensionality of the network, the training time, and the likelihood of overfitting; the other is the classification, where the 3D data within the network is flattened into a 1D vector to be output.

Benchmark

Setting aside the fact that the classes are slightly imbalanced, a random guess will provide a correct answer roughly 1 in 133 times,

which corresponds to an accuracy of less than 1%. A better benchmark model will be to create a CNN to classify dog breeds from scratch, which should improve the accuracy.

III. Methodology

Data Preprocessing

Resize the images to (224, 224, 3) and rescale the images by dividing every pixel in every image by 255, and split the data into train, test, and validation sets.

```
def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and retur
n 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_p
aths)]
    return np.vstack(list_of_tensors)

train_tensors = paths_to_tensor(train_files).astype('float32')/255
valid_tensors = paths_to_tensor(valid_files).astype('float32')/255
test_tensors = paths_to_tensor(test_files).astype('float32')/255
```

Implementation

For architecture constructing, define the CNN structure, specify loss function and optimizer. Three sets of convolutional and max pooling layers are used to extract complex features to distinguish different dog breeds. A flatten layer and a fully connected layer are added, then an output layer with SoftMax activation.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
conv1 (Conv2D)               (None, 112, 112, 16)      1216
_________________________________________________________________
activation_50 (Activation)   (None, 112, 112, 16)      0
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 111, 111, 16)      0
_________________________________________________________________
conv2 (Conv2D)               (None, 56, 56, 32)        12832
_________________________________________________________________
relu2 (Activation)           (None, 56, 56, 32)        0
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 28, 28, 32)        0
_________________________________________________________________
conv3 (Conv2D)               (None, 28, 28, 64)        18496
_________________________________________________________________
activation_51 (Activation)   (None, 28, 28, 64)        0
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 14, 14, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 12544)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12544)             0
_________________________________________________________________
dense1 (Dense)               (None, 64)                802880
_________________________________________________________________
activation_52 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
output (Dense)               (None, 133)               8645
=================================================================
Total params: 844,069
Trainable params: 844,069
Non-trainable params: 0
_________________________________________________________________
```

Refinement

Creating a CNN to predict dog breed from scratch achieves a test accuracy score: ~6%, so the next step is to use transfer learning to create a CNN that attains greatly improved accuracy.

```
bottleneck_features = np.load('/data/bottleneck_features/DogVGG16Data.npz
')
```

```
train_VGG16 = bottleneck_features['train']
valid_VGG16 = bottleneck_features['valid']
test_VGG16 = bottleneck_features['test']
VGG16_model = Sequential()
VGG16_model.add(GlobalAveragePooling2D(input_shape=train_VGG16.shape[1:]))
VGG16_model.add(Dense(133, activation='softmax'))
```

Train and validate the proposed model, save the final model parameters, test the model based on the evaluation metric presented above.

```
# get index of predicted dog breed for each image in test set
VGG16_predictions = [np.argmax(VGG16_model.predict(np.expand_dims(feature, axis=0))) for feature in test_VGG16]

# report test accuracy
test_accuracy = 100*np.sum(np.array(VGG16_predictions)==np.argmax(test_targets, axis=1))/len(VGG16_predictions)
print('Test accuracy: %.4f%%' % test_accuracy)
```

```
Test accuracy: 43.3014%
```

Pick a different pre-trained model: ResNet-50, use the corresponding bottleneck features, create a CNN to classify dog breed.

```
# Model architecture including a Global Average Pooling layer and an output layer

resnet_model = Sequential()
resnet_model.add(GlobalAveragePooling2D(input_shape=train_resnet.shape[1:]))
resnet_model.add(Dense(133, activation='softmax'))

resnet_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2   (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               272517
=================================================================
Total params: 272,517
Trainable params: 272,517
Non-trainable params: 0
_________________________________________________________________
```

```
resnet_predictions = [np.argmax(resnet_model.predict(np.expand_dims(feature, axis=0))) for feature in test_resnet]
```

```
# report test accuracy
test_accuracy = 100*np.sum(np.array(resnet_predictions)==np.argmax(test_targe
ts, axis=1))/len(resnet_predictions)
print('Test accuracy: %.4f%%' % test_accuracy)
Test accuracy: 80.1435%
```

IV. Results

Model Evaluation and Validation

Extract the bottleneck features corresponding to the chosen CNN model, supply the bottleneck features as input to the model to return the predicted vector: the argmax of this prediction vector gives the index of the predicted dog breed, use the dog_names array defined before to return the corresponding breed.

```
def dog_breed_predictor(img_path):
    # Extract the bottleneck features for Resnet CNN model
    bottleneck_feature = extract_Resnet50(path_to_tensor(img_path))

    # Load the best model
    resnet_model.load_weights('saved_models/weights.best.resnet.hdf5')
    # Obtain predicted vector
    predicted_vector = resnet_model.predict(bottleneck_feature)
    # Return dog breed that is predicted by the model
    return dog_names[np.argmax(predicted_vector)]
```

Justification

The final model achieves a classification accuracy of 80% on the test set, much higher than the benchmark. Transfer learning based on ResNet-50 is better than VGG-16.

Model	Classification Accuracy
CNN from Scratch	6%

Transfer Learning based on VGG-16	43%
Transfer Learning based on ResNet-50	80%

V. Conclusion

Free-Form Visualization

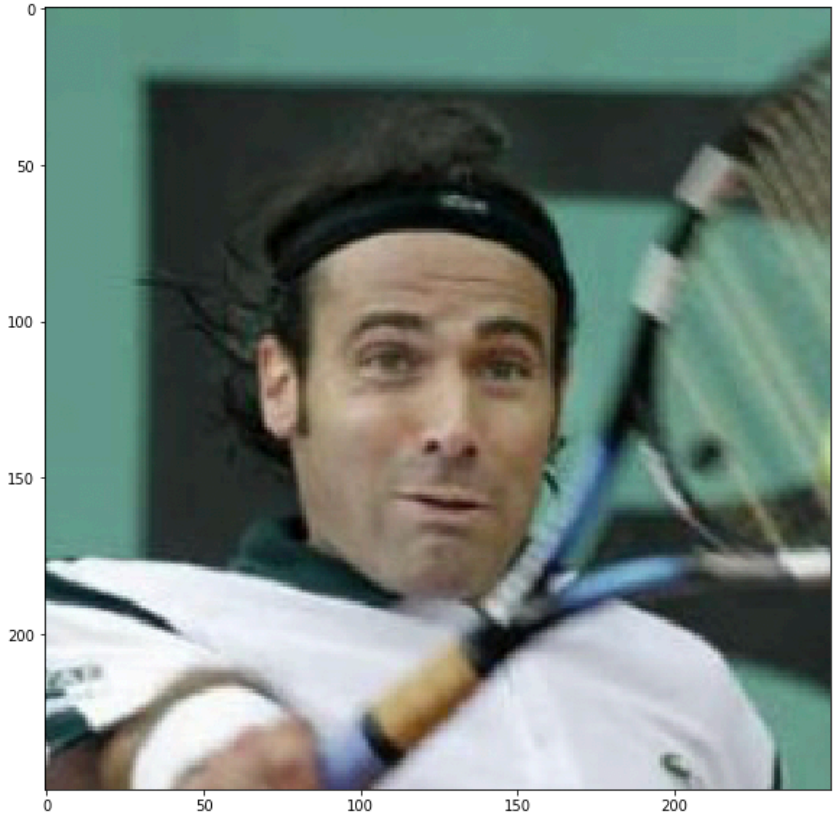
Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

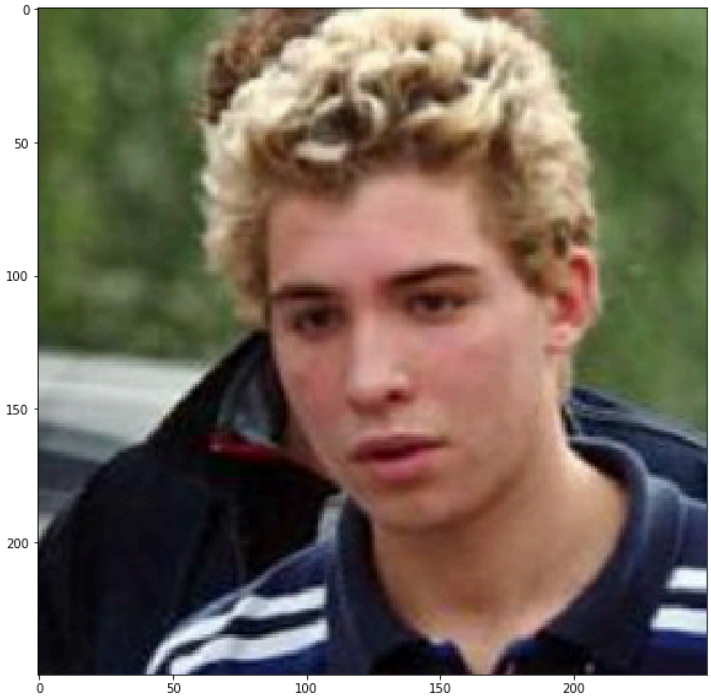
- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide output that indicates an error.

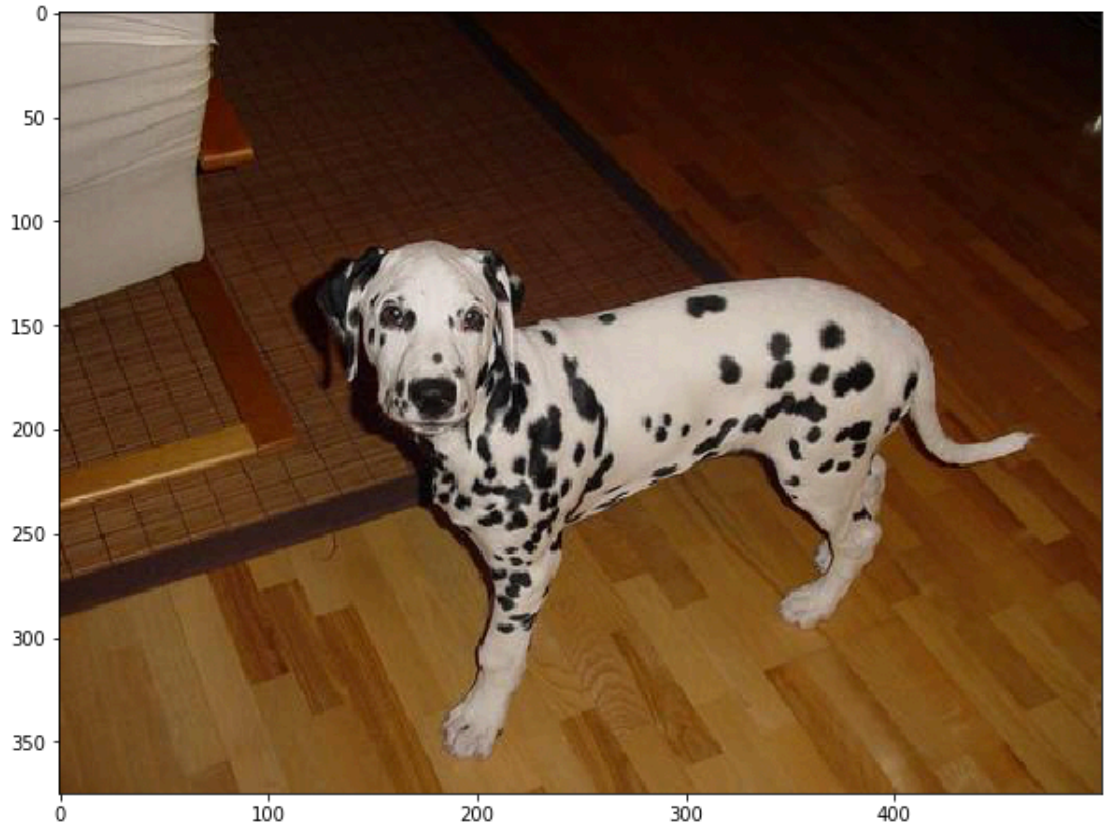
```
def my_predictor(img_path):
    # Load the image
    img= mpimg.imread(img_path)
    plt.figure(figsize=(10,10))
    plt.imshow(img)

    # Use dog_detector and dog_breed_predictor functions to detect dogs an
d predict their breed
    if dog_detector(img_path) == True:
        print("It is a dog!.Dog breed is:", dog_breed_predictor(img_path))
    elif dog_detector(img_path) == False:
         print("It is a human!.You resemble the dog breed:", dog_breed_pre
dictor(img_path))
    else:
        print("An error has occured")

for infile in np.hstack((human_files[:3], test_files[:3])):
     my_predictor(infile)
```

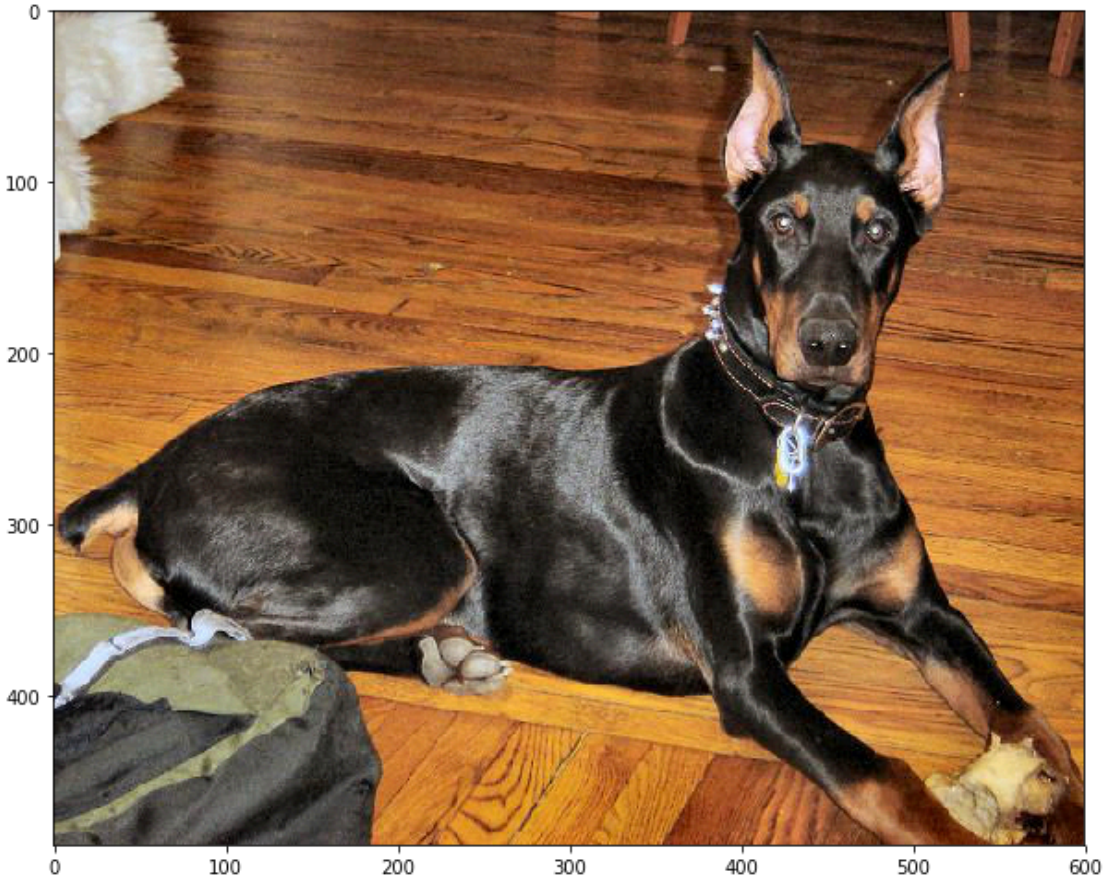
```
It is a human!.You resemble the dog breed: Beagle
It is a human!.You resemble the dog breed: Italian_greyhound
It is a human!.You resemble the dog breed: Silky_terrier
It is a dog!.Dog breed is: Dalmatian
It is a dog!.Dog breed is: Doberman_pinscher
It is a dog!.Dog breed is: Papillon
```

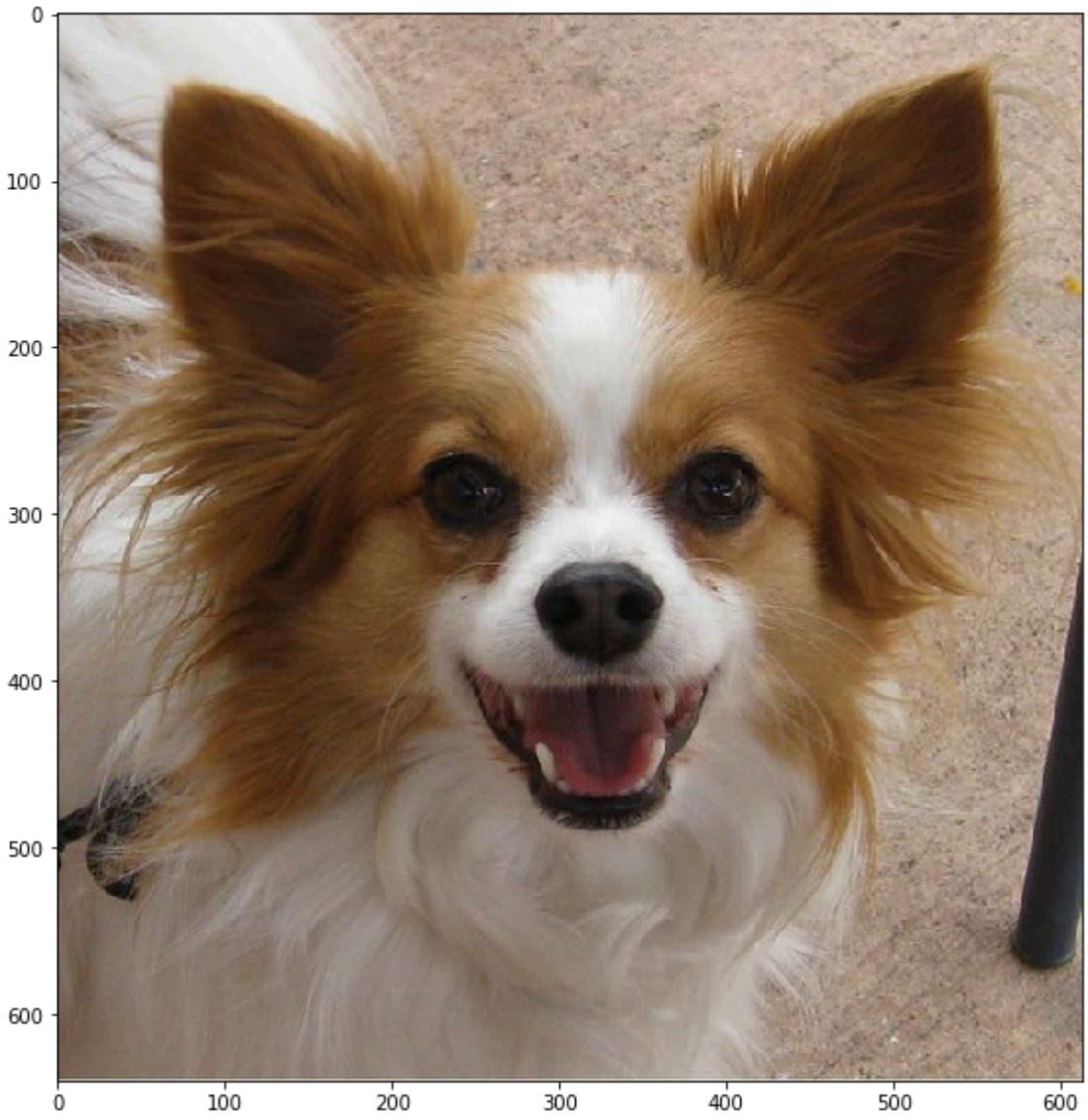

0
50
100
150
200
0
50
100
150
200


0
50
100
150
200
0
50
100
150
200


0
50
100
150
200
0
50
100
150
200


0
50
100
150
200
250
300
350
0
100
200
300
400


0
100
200
300
400
0
100
200
300
400
500
600



Reflection

Several steps are taken in this project:

- Import Datasets
- Exploratory Data Analysis
- Data Preprocessing
- Create a CNN to Classify Dog Breeds (from Scratch)
- Create a CNN to Classify Dog Breeds (using Transfer Learning)

Improvement

Finally, provide 3 possible points of improvement. The output is better than I expected. First, there is imbalance between different dog breeds, more training data will help; Second, try more transfer learning with VGG19, InceptionV3 or Xception; Third, model fusion can further improve.