Data Scientist Nanodegree Capstone Project

Image Classification: Dogs Breeds

Yifan Sun @ April 2020

Definition

Project Overview

The goal of this project is to develop a machine learning model to classify dog images into different breeds, by using a data set that consists of 8351 images classified into 133 breeds. This project is not driven by practical needs, but is motivated by learning to apply state of the art machine learning techniques to image classification. In particular, I am motivated to compare the classifier performance with versus without applying transfer learning techniques.

Problem Statements

The goal is to create an multi-class image classifier that classifies dog images into one of 133 breeds. I will focus on using state of the art image classification technique of Convolutionary Neural Network (CNN). We want to compare the performance of two approaches:

1. Construct and train CNN from scratch
2. Apply transfer learning and fine-tune a pre-trained CNN. (We will use ResNet-50 as an example).

Metrics

The classifiers will be evaluated with accuracy, a common metric for classification problems. In the context of multi-class image classification,

Accuracy = Number of Test Images Classified Correctly / Total Number of Test Images

Analysis

Data Exploration

The data set consists of 8351 dog images classified into 133 breeds. These are color photos of varying photo sizes. The Udacity provided data set has already been divided into a training set of 6680 images, a validation set of 835 images and a test set of 836 images. Some examples are presented below.

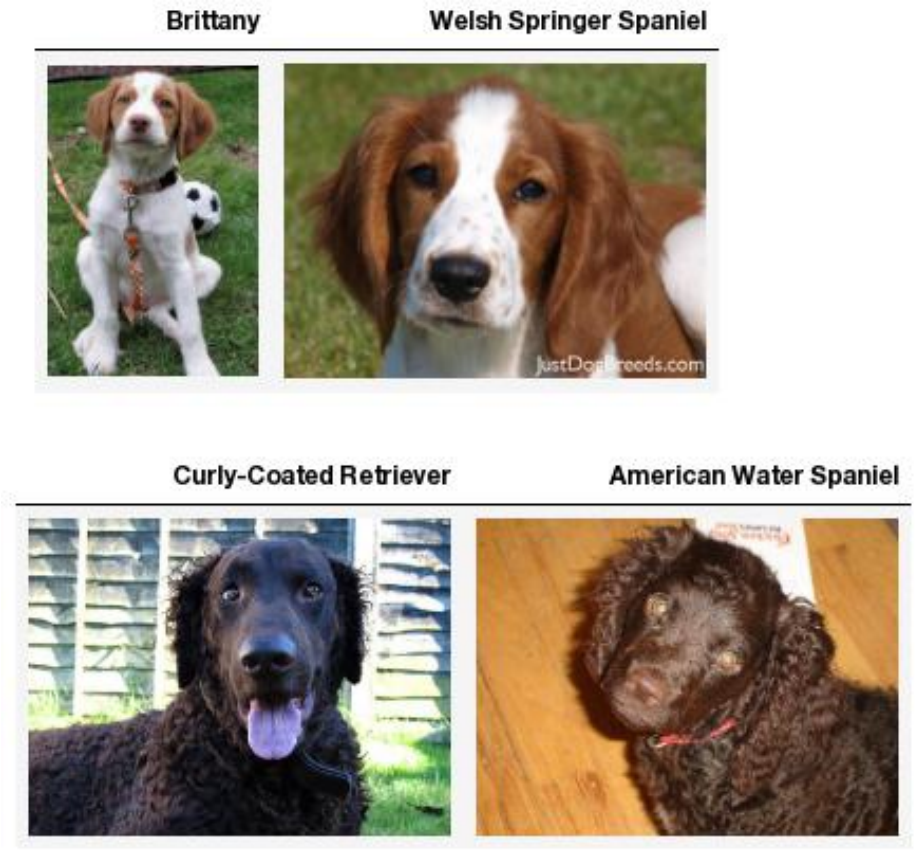

Figure 1: Example Dog Images from the Data Set

Exploitative Data Visualization

I want to examine whether there is roughly equal number of images for each breed, and whether the data distribution differs across the training, validation and testing data sets. Thus, I used a bar chart to visualize the count of images for each breed. The bar chart indicates that each breed have roughly equal number of images, for all of the training, validation, and testing data sets. However, the entire bar chart is too large to present here. Thus, in the project report, I will present a bar chart of the first five breeds (in alphabetical order) as an example. The complete bar char can be found in the Jupyter notebook of this project.

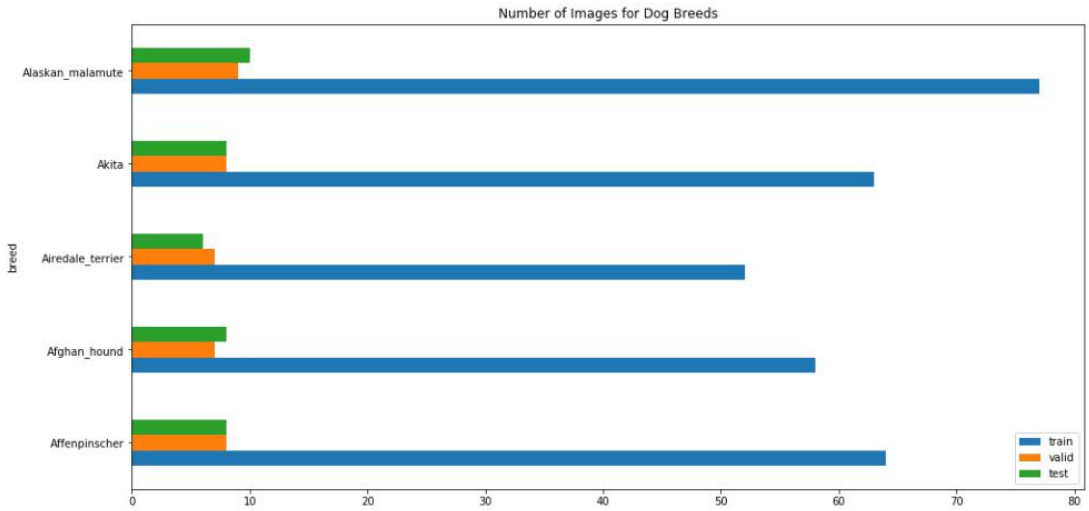

Figure 2: Number of Images for Dog Breeds in the Data Set, for First Five Breeds (in alphabetical order). Note: The complete bar chart for all 133 breeds is available in the project notebook.

Methodology

Data Preprocessing

The data preprocessing consists of the following steps:
1. The images have been randomly divided into 80% training, 10% validation, and 10% testing data sets.
2. To train CNN from scratch, the images of varying sizes are resized to shape of 224 x 224 pixels, and normalize by dividing each pixel value by 255.
3. To speed up training in finetuning the ResNet-50 classifier, the images are run through pre-trained standard ResNet-50 to extract their bottleneck features. Thus, each image is pre-processed into a feature vector of 2048 float numbers.

Implementation

Convolutionary Neural Network (CNN) is the state of the art image classification technique. Thus, this project focuses on CNN techniques. I will compare classification performance of two alternative options based on CNN techniques: a CNN trained from scratch versus a CNN finetunned from ResNet-50. Their implementation are outlined below.

First, for the training a CNN from scratch option, I have started with a fairly standard neural network architecture given in the project notebook. After some refinement (to be presented in the next section), I have settled with the network architecture below.

Layer (type)	Output Shape	Param #
input_33 (InputLayer)	(None, 224, 224, 3)	0
dropout_10 (Dropout)	(None, 224, 224, 3)	0
conv2d_7 (Conv2D)	(None, 224, 224, 16)	448
max_pooling2d_35 (MaxPooling	(None, 112, 112, 16)	0
conv2d_8 (Conv2D)	(None, 112, 112, 32)	4640
max_pooling2d_36 (MaxPooling	(None, 56, 56, 32)	0
flatten_5 (Flatten)	(None, 100352)	0
dense_12 (Dense)	(None, 128)	12845184
dropout_11 (Dropout)	(None, 128)	0
batch_normalization_6 (Batch	(None, 128)	512
dense_13 (Dense)	(None, 133)	17157

Second, for the finetuning ResNet-50 option, I have started with an example architecture given in the project notebook (which was used to finetune the VGG16 model). After some refinement (to be presented in the next section), I have settled with the network architecture below.

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	(None, 1, 1, 2048)	0
global_average_pooling2d_3 (	(None, 2048)	0
dense_10 (Dense)	(None, 256)	524544
dropout_9 (Dropout)	(None, 256)	0
batch_normalization_5 (Batch	(None, 256)	1024
dense_11 (Dense)	(None, 133)	34181

Refinement

As discussed in the previous section, for both options, I have started with an example provided in the original project notebook. These examples are representative of the standard industry practice when the project was developed. On top of the example start point, I have added the following refinement.

For the first option of training a CNN from scratch, the refinements include:
1. I have added Dropout layers to make the model generalize better.
2. I have added the Batch Normalization layer to speed up training.
3. I then started parameter tunning. In particular, I have tried to alter model size (e.g., alter the number of CNN filters, alter the number of params in fully connected layer, add more fully connected layers, and etc.). I found that
    a) When model size is too big, training would either fail due to out-of-memory error, or the model would overfit with better training accuracy but worse validation accuracy.
    b) When model size is too small, model performance would be significantly worse.
    c) For other parameters I tried, the models perform more or less the same in terms of accuracy. But bigger models takes longer to train and consumes more computation resource.
4. Finally, I have chosen the following: the smallest (simplest) CNN network that have relatively good performance among all models tested.

For the second option of finetunning ResNet-50, the refinements include:
1. I have added Dropout layers to make the model generalize better.
2. I have added the Batch Normalization layer to speed up training.
3. I have added a dense layer before the output layer. The goal is to enable the neural network to learn to combine simple features into complex ones to increase model performance.

Results

Model Evaluation and Validation

The same data are used to train, validate and test both models. The models are trained for several epochs and one with best accuracy on the validation data is selected. Finally, the models are evaluated on the test data in terms of accuracy.

The baseline of comparison is to predict the breed at random. Since there are 133 breeds, the baseline accuracy is 1/133 = 0.75%.

Option 1 (Training CNN from scratch) has accuracy of 2.99%. This is not great, but much better than the baseline.

Option 2 (Finetuning ResNet-50) has accuracy of 80.38%. This is head and shoulders above both the baseline and option 1.

Justification

Option 1 (Training CNN from scratch) is much better than the baseline of random prediction, otherwise the model have not learned anything useful. However, an accuracy of 2.99% is still pretty low. There are three main factors of bad performance here. First, there is as many as 133 categories, and it is hard to predict accurately. Second, there is such limited amount of training data that the model have not learned enough useful features to make accurate predictions. Third, constrained by both limited training data and limited computation resource, I was not able to use a deep neural network in option 1, which would resulted in better performance.

Option 2 (Finetuning ResNet-50) has achieved a high accuracy of 80.38%. It has clearly benefit a lot from transfer learning techniques: a pre-trained deep CNN ResNet-50 is used to extract useful features out of the images. Given these extracted features, even a fairly simple neural network trained on on limited amount of data can achieve good performance.

Conclusion

Reflection

The execution of this project consists of the following steps:
1. The project goal is set as classifying dog images into several breeds.
2. The dataset is provided by Udacity, along with data preprocessing code and result.
3. Accuracy is chosen as the metric to measure the success of the project. The accuracy of 0.75% from a dummy random classifier is set as the baseline.
4. The scope of the project is set to compare two options: train a CNN classifier from scratch versus finetune an existing deep CNN, ResNet-50.
5. For each option, a classifier is trained, refined, and evaluated. Evaluation indicates that option 1 resulted in an accuracy of 2.99% while option 2 resulted in an accuracy of 80.38%.

I find it interesting that finetunning an existing deep CNN brings such a dramatic improvement to classifier performance. I am motivated to invest more efforts to study transfer learning techniques in the future.

Improvement

Several techniques can potentially be applied to further improve classifier performance.

First, we can test other pretrained deep CNN such as VGG19, Inception, Xception, and etc.

Second, thanks to ensemble learning techniques, we can use multiple rather than one pretrained deep CNN to increase classifier performance.

Third, we can apply data augmentation techniques to increase amount of training data. For example, we can create new training images by fliping and rotating existing images. This is particularly useful when there is limited amount of training data.