CNN Project: Dog Breed Classifier

A Capstone Report

By

PRAVEEN BANDARU

Submitted to Udacity

in partial fulfillment of the requirements for the degree of

Machine Learning Engineer

Nanodegree Program

March 2020

Definition

Project Overview

Image classification [1] is a supervised learning [2] problem: define a set of target classes (objects to identify in images) and train a model to recognize them using labeled example photos. A breakthrough in building models for image classification came with the discovery that a convolutional neural network [3] (CNN) could be used to progressively extract higher- and higher-level representations of the image content. Instead of preprocessing the data to derive features like textures and shapes, a CNN takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they constitute.

In this project, I created a code pipeline using transfer learning to create a CNN that can identify dog breed from images. We used a pretrained ResNet50 model from Torchvision Model Zoo for our Dog Classification problem. This pretrained model was trained on ImageNet dataset and has the last Fully Connected layer with 1000 out features. These pre-trained networks demonstrate a strong ability to generalize to images outside the ImageNet dataset via transfer learning. We make modifications in the pre-existing model by fine-tuning the model. We have frozen the parameters, so we don't back-propogate through them and replaced the last Fully Connected layer with a Linear layer having 133 out features equal to the number of classes in our Dog dataset. Our pipeline 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 a sample output of dog detection pipeline.

Problem Statement

In this project, we will learn how to build a pipeline to process real-world, user-supplied images. Given an image of a dog, our 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. Along with exploring state-of-the-art CNN models for classification, we will make important design decisions about the user experience for our app. Our goal is that by completing this project, we understand the challenges involved in piecing together a series of models designed to perform various tasks in a data processing pipeline. Each model has its strengths and weaknesses, and engineering a real-world application often involves solving many problems without a perfect answer. Our imperfect solution will nonetheless create a fun user experience!

Evaluation Metrics

The model's performance will be evaluated based on the test set accuracy score, i.e., we want to see how the model will perform when it encounters an image it has never seen before. Accuracy is a common metric for image classification, it takes into account both the true positives and true negatives with equal weight.

Accuracy = (true positives + true negatives) / dataset size

We achieved a test accuracy of 85% after running our model on the test dataset.

Analysis

Data Exploration

The required human [4] and dog [5] datasets were provided by Udacity as downloadable links in the *dog_app.ipynb* notebook. They can be also found in the /data folder in the Udacity workspace. There are 13233 total human images and 8351 total dog images which spanned across 133 breeds. The dog images were already split into *test*, *train* and *valid* subfolders and each of the *test*, *train* and *valid* folders were further divided into 133 breeds subfolders. In the *dog_app.ipynb* notebook, we save the file paths for both the human (LFW) dataset and dog dataset in the numpy arrays *human_files* and *dog_files*.

Exploratory Visualization

Here is an exploratory visualization of the training data set.

The following plot shows a sample image for the first 50 classes from the training data set.

Fig. 1 Images from the first 50 classes of the dog training data set

Next, I plotted a bar chart showing how the data is distributed across different classes. It shows the total count of images from each class.

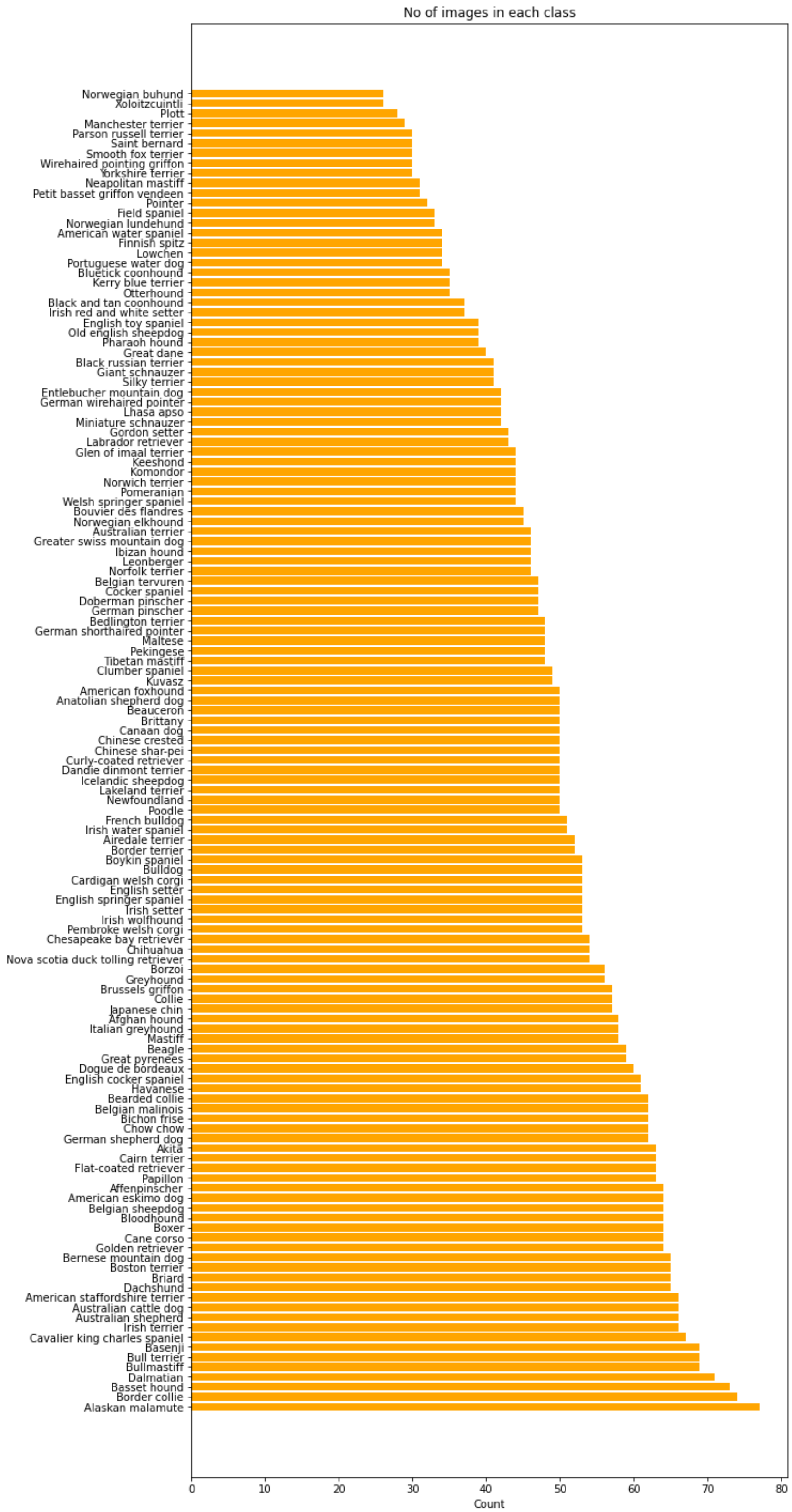

Fig. 2 Bar chart showing total count of images from each class

Algorithms and Techniques

Our goal in this project is to create a CNN from scratch as well as by using transfer learning to classify dog breeds. To start, the CNN receives an input feature map: a three-dimensional matrix where the size of the first two dimensions corresponds to the length and width of the images in pixels. The size of the third dimension is 3 (corresponding to the 3 channels of a color image: red, green, and blue). The CNN comprises a stack of modules, each of which performs three operations: Convolution [6], Rectified Linear Unit [7] (ReLU) and Pooling [8].

At the end of a convolutional neural network are one or more fully connected layers [9] (when two layers are "fully connected," every node in the first layer is connected to every node in the second layer). Their job is to perform classification based on the features extracted by the convolutions. Here we use a final fully connected layer having 133 output features to classify the input image into one of the 133 dog breed classes.

As with any machine learning model, a key concern when training a convolutional neural network is overfitting [10]: a model so tuned to the specifics of the training data that it is unable to generalize to new examples. Two techniques to prevent overfitting when building a CNN are: Data augmentation and Dropout regularization. Here we use Dropout to prevent overfitting.

Training a convolutional neural network to perform image classification tasks typically requires an extremely large amount of training data, and can be very time-consuming, taking days or even weeks to complete. But what if you could leverage existing image models trained on enormous datasets, such as Inception, and adapt them for use in your own classification tasks? One common technique for leveraging pretrained models is feature extraction: retrieving intermediate representations produced by the pretrained model, and then feeding these representations into a new model as input.

Here we use a pretrained ResNet50 [12] model to create a CNN based on transfer learning. Here we replace the final fully connected layer of the pretrained ResNet50 model with our own having 133 output features corresponding to the 133 dog breeds.

Benchmark

The CNN model built from scratch by us will serve as a benchmark model to validate the performance of our final optimized model built using transfer learning. This benchmark model also ascertain that the dog breed classification problem is clearly defined and solvable.

Methodology

Data Preprocessing

The dog dataset was already separated into train, valid and test datasets and each dataset was further separated into 133 folders corresponding to the 133 dog breeds.

We applied the following preprocessing and data augmentation techniques:

- Randomly resized and cropped the images to 224x224 using *transforms.RandomResizedCrop(224)*
- Randomly flipped the images horizontally using *transforms.RandomHorizontalFlip()*
- Randomly rotated the images using *transforms.RandomRotation(10)*
- Normalized the images using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]

Implementation

The implementation consists of the following two stages:

1. **Create a CNN to Classify Dog Breeds (from Scratch):**
   Here we will create a CNN that classifies dog breeds from scratch and use it as a benchmark model to evaluate the CNN model which we create using Transfer Learning.

2. **Create a CNN to Classify Dog Breeds (using Transfer Learning):**
   We will now use transfer learning to create a CNN that can identify dog breed from images.

References

1. https://www.sciencedirect.com/topics/computer-science/image-classification
2. https://en.wikipedia.org/wiki/Supervised_learning
3. https://wikipedia.org/wiki/Convolutional_neural_network
4. https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip
5. https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
6. https://en.wikipedia.org/wiki/Convolutional_neural_network#Convolutional_layer
7. https://en.wikipedia.org/wiki/Convolutional_neural_network#ReLU_layer
8. https://en.wikipedia.org/wiki/Convolutional_neural_network#Pooling_layer
9. https://en.wikipedia.org/wiki/Convolutional_neural_network#Fully_connected_layer
10. https://en.wikipedia.org/wiki/Overfitting
11. https://arxiv.org/abs/1409.1556
12. https://arxiv.org/abs/1512.03385