CNN Project: Dog Breed Classifier

A Capstone Proposal

By

PRAVEEN BANDARU

Submitted to Udacity

in partial fulfillment of the requirements for the degree of

Machine Learning Engineer

Nanodegree Program

February 2020

1. DOMAIN BACKGROUND

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. Early computer vision models relied on raw pixel data as the input to the model. However, raw pixel data alone doesn't provide a sufficiently stable representation to encompass the myriad variations of an object as captured in an image. The position of the object, background behind the object, ambient lighting, camera angle, and camera focus all can produce fluctuation in raw pixel data; these differences are significant enough that they cannot be corrected for by taking weighted averages of pixel RGB values.

To model objects more flexibly, classic computer vision models added new features derived from pixel data, such as color histograms [3], textures, and shapes. The downside of this approach was that feature engineering [4] became a real burden, as there were so many inputs to tweak. For a cat classifier, which colors were most relevant? How flexible should the shape definitions be? Because features needed to be tuned so precisely, building robust models was quite challenging, and accuracy suffered.

A breakthrough in building models for image classification came with the discovery that a convolutional neural network [5] (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.

2. 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!

3. DATASETS AND INPUTS

The required human [6] and dog [7] 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*.

4. SOLUTION STATEMENT

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 [8], Rectified Linear Unit [9] (ReLU) and Pooling [10].

At the end of a convolutional neural network are one or more fully connected layers [11] (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 [12]: 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 VGG [13] /ResNet [14] model to create a CNN based on transfer learning. Here we replace the final fully connected layer of the pretrained VGG/ResNet model with our own having 133 output features corresponding to the 133 dog breeds.

5. BENCHMARK MODEL

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.

6. 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.

7. PROJECT DESIGN

This project design consists of the following steps as described in the *dog_app.ipynb* notebook.

i. Import Datasets: Here we download and import the required human and dog datasets.

ii. Detect Humans: Here we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

iii. Detect Dogs: Here we use a pre-trained model to detect dogs in images.

iv. Create a CNN to Classify Dog Breeds (from Scratch): Here we will create a CNN that classifies dog breeds from scratch.

v. 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.

vi. Write your Algorithm: Here we 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.

vii. Test Your Algorithm: In this section, we will take our new algorithm for a spin! What kind of dog does the algorithm think that you look like? If you have a dog, does it predict your dog's breed accurately? If you have a cat, does it mistakenly think that your cat is a dog?

References

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

11. https://en.wikipedia.org/wiki/Convolutional_neural_network#Fully_connected_layer
12. https://en.wikipedia.org/wiki/Overfitting
13. https://arxiv.org/abs/1409.1556
14. https://arxiv.org/abs/1512.03385