Machine Learning Engineer Nanodegree

Capstone Project

Wahib Farhat October 10th, 2021

I. Definition

Project Overview

This is an image classification project for identifying dog breeds. Identifying dog breeds is useful for finding dogs with preferred characteristics, like growth size, whether they are suited to indoors or outdoors and the toes and quantity of food they eat. It can also be used for targeted advertising. Food and toys for specific breeds of dogs can be advertised to users with those dogs.

Problem Statement

Given an image of a dog, identify its breed. If the image is an image of a human, identify the resembling dog breed. A good approach to solve this problem would be to apply deep learning using a Convolutional Neural Network. We will define a CNN architecture and train it using a set images of dogs labelled with the correct breed. Ideally, the model resulting from the training should be able to identify breeds of dogs from new images.

Metrics

Our performance metric is accuracy since this is a common metric for classification algorithms. This is the sum of true positives and true negatives, divided by the size of the dataset. The accuracy test will be done on the 133 test images included in the dataset.

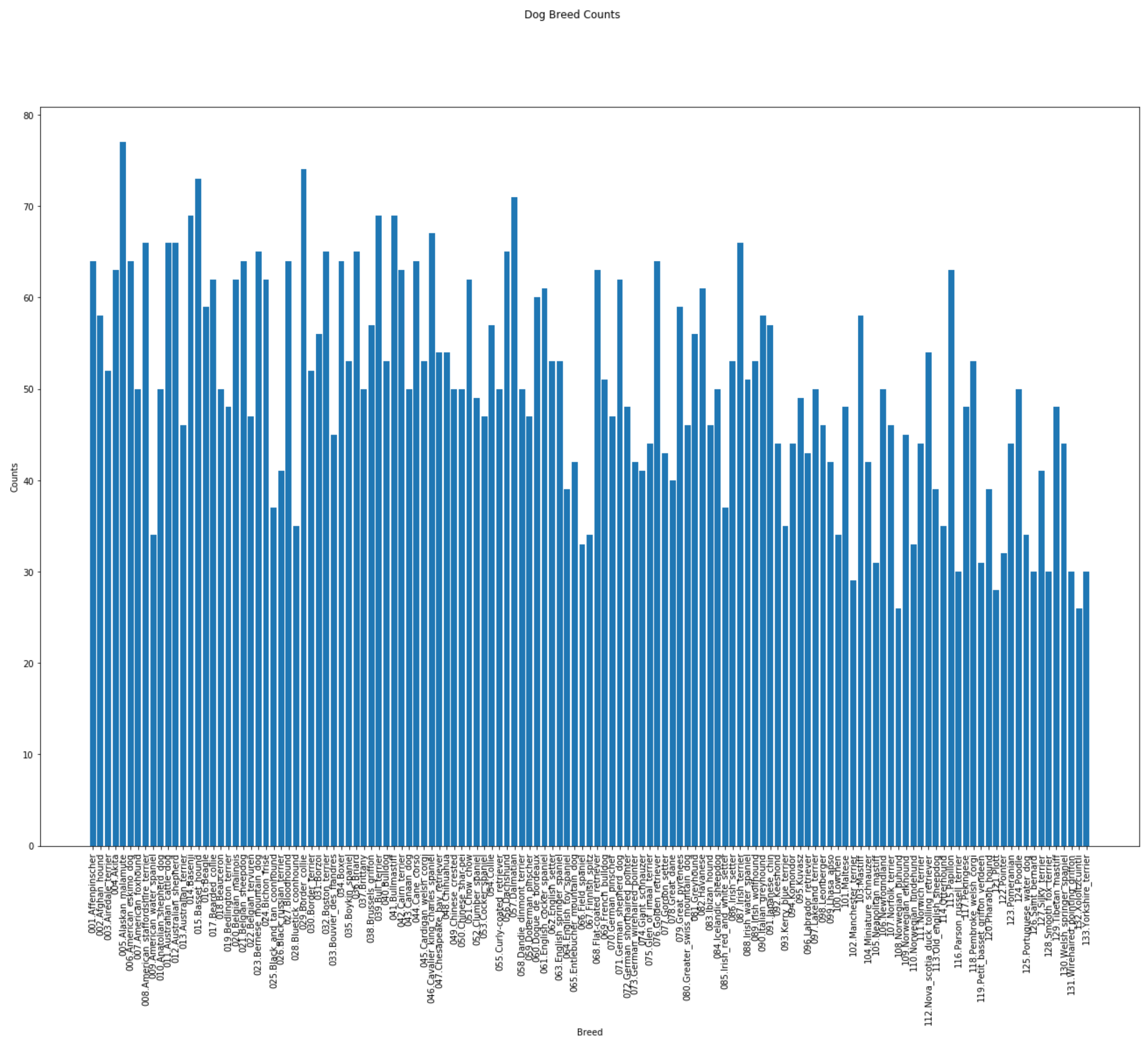
II. Analysis

Data Exploration

The dataset from Udacity has 8351 images https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip. It has already been split into training, test and validation data. Our focus of the exploration will be on the training data. The training data consists of 6680 colored images of dogs, labelled by breed. There is a lot of variation in the images sizes and aspect ratios, ranging from 112x120 to 4278x3744. There is also significant variation in the dog poses and the lighting conditions. Some of the images have other objects and even humans near the dog. The most occuring breed is the Alaskan Malamute with 77 training examples while the least occurring breed is Xoloitzcuintli. However, the training data is fairly balanced with most breeds having 40-60 images.

Exploratory Visualization

In this section, you will need to provide some form of visualization that summarizes or extracts a relevant characteristic or feature about the data. The visualization should adequately support the data being used. Discuss why this visualization was chosen and how it is relevant. Questions to ask yourself when writing this section:



This bar chart indicates the number of training images per dog breed. This visualization was chosen because it shows the distribution of training images across the various breeds. Most of the breeds have 40-60 training images.

Algorithms and Techniques

We will use a Convolutional Neural Network for this project. CNNs are a good fit for image classification tasks because they are feature location invariant. Since we can have dogs at different parts of the image and in different orientations, we can take advantage of this. We will pass each image channel through three convolutional layers, applying a maxpooling layer in between them to prevent overfitting. After that, the data will be flattened and passed through two fully connected layers in order to make a prediction. We will train the model for 30 epochs, checking the model accuracy on the validation set. Whenever the validation accuracy reduces, we will save the model parameters, overwriting the previous saved model.

Benchmark

In this section, you will need to provide a clearly defined benchmark result or threshold for comparing across performances obtained by your solution. The reasoning behind the benchmark (in the case where it is not an established result) should be discussed. Questions to ask yourself when writing this section: The accuracy threshold for this project are at least 10% accuracy for our CNN model built from scratch and at least 60% accuracy for the ResNet50 model. These thresholds were chosen because we have a relatively small training set.

III. Methodology

Data Preprocessing

Since the images have varying sizes, the images will have to be resized. The dog positions in the images and their poses also vary so it will be a good idea to perform a random resized crop in order to generalize the model and prevent overfitting.

Implementation

The CNN architecture consisted of 3 layers with sizes (3x32), (32x64), (64x128). I used a kernel size of 3 and applied a padding of 1 pixel to keep the output image size constant. I used a 2x2 MaxPooling layer to downsample the output at each layer. I then flattened the output of the conv2d layers before passing it through my two fully connected layers. The input size of my first fully connected layer (128 * 28 * 28) was calculated as follows:

128 is the number of output channels of the last conv2d layer 28x28 is the size of the image after the original image (224x224) is downsampled 3 times by a 2x2 MaxPooling layer (224/(2^3))

Before passing the output through each fully connected layer, I apply a Dropout layer of 0.5 to prevent overfitting.

The complete CNN class is defined below

```
    import torch.nn as nn
    import torch.nn.functional as F

    # define the CNN architecture
    class Net(nn.Module):
        def __init__(self):
            super(Net, self).__init__()
            ## Define layers of a CNN
            self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
            self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
            self.conv3 = nn.Conv2d(64, 128, 3, padding=1)

            self.pool = nn.MaxPool2d(2, 2)

            self.fc1 = nn.Linear(128 * 28 * 28, 500)
            self.fc2 = nn.Linear(500, 133)

            self.dropout = nn.Dropout(0.5)

        def forward(self, x):
            ## Define forward behavior
            x = self.pool(F.relu(self.conv1(x)))
            x = self.pool(F.relu(self.conv2(x)))
            x = self.pool(F.relu(self.conv3(x)))

            x = x.view(-1, 128 * 28 * 28)

            x = self.dropout(x)

            x = F.relu(self.fc1(x))

            x = self.dropout(x)

            x = self.fc2(x)
            return x

    # instantiate the CNN
    model_scratch = Net()

    # move tensors to GPU if CUDA is available
    if use_cuda:
        model_scratch.cuda()
```

The next step was to load the data using PyTorch's DataLoader class. I enabled shuffling on the training data loader. I then proceeded to train the model using cross entropy loss function and the stochastic gradient descent loss function.

Before running the first epoch, I set the variable tracking validation loss to infinity. That way, model will be saved since the validation will reduce. Each epoch afterward checks the validation loss to determine if the model needs to be overwritten.

We will then use the model to predict the breeds of dogs in the test data images. We will divide the number of correctly predicted images by the total number of images in the test dataset to get our accuracy metric.

Refinement

My initial model had a very low accuracy (around 2%). Judging from the training loss which was low compared to the validation and test loss, I concluded that my model was overfitting the test data. To address this I increased the odds of the dropout layer from 0.2 to 0.5. I also used random resizing and cropping on the training data. These actions dramatically increased my test accuracy to 15%.

IV. Results

Model Evaluation and Validation

The final model chosen had 15% accuracy on the test data (133 correct predictions out of 836 images). It was chosen because it had a higher accuracy than the 10% benchmark.

Justification

The final model with transfer learning (ResNet50) had an accuracy of 85% which was higher than the 60% benchmark.

Based on the results above, the final solution is significant enough to solve the problem.

V. Conclusion

Free-Form Visualization

Examples of tests of the model on new image data that was not in the Udacity dataset.

```
well well, a dog!
```

```
specifically a Mastiff
```

```
well well, a dog!
```

```
specifically a Bull terrier
```

The breed predictions were accurate.

Reflection

The entire end-to-end solution can be summarized in the following steps: - Download the image data - Transform and load the data, applying image augmentation. - Define CNN architecture - Train the CNN using training data while checking validation accuracy - Test the trained model against the test dataset

One aspect I found difficult was preventing overfitting. The training accuracy was increasing while the test and validation accuracy reduced. Using a combination of a higher dropout and data augmentation, I was able to overcome this. The CNN performs well on image recognition tasks because it is able to analyse and detect patterns at varying parts of the images. The final model and solution fit my expectations for the dog breed prediction and I think it can be used in a general setting for people who want to detect dog breeds.

Improvement

We could have further increased accuracy by training for more epochs as the validation error was still reducing at 30 epochs. Other pretrained models like VGG-16 and Inception-v3 could also offer some improvement.