Machine Learning Engineer Nanodegree

Capstone Proposal

Arda Atahan İbiş August 3rd, 2020

Proposal

Domain Background

Throughout centuries, dogs have been an integral part of the human life. Due to their high running speed, acute sense of smell and hearing, hunting skills, and friendliness dogs have been cared as house pets, trained for service in order to assist such as the police, fire department, and etc. As the need for dogs in the human life increased, different dog breeds have been interbred in order to have a "new breed" that has the pros of the two parent breeds. Therefore, there are a wide variety of dog breeds, 339 according to the World Canine Organization (FCI)[1], each of which attains a relatively different role in the human life. Given the wide variety of dog breeds, it is virtually impossible for any human to predict the breed of a dog with high accuracy.

Problem Statement

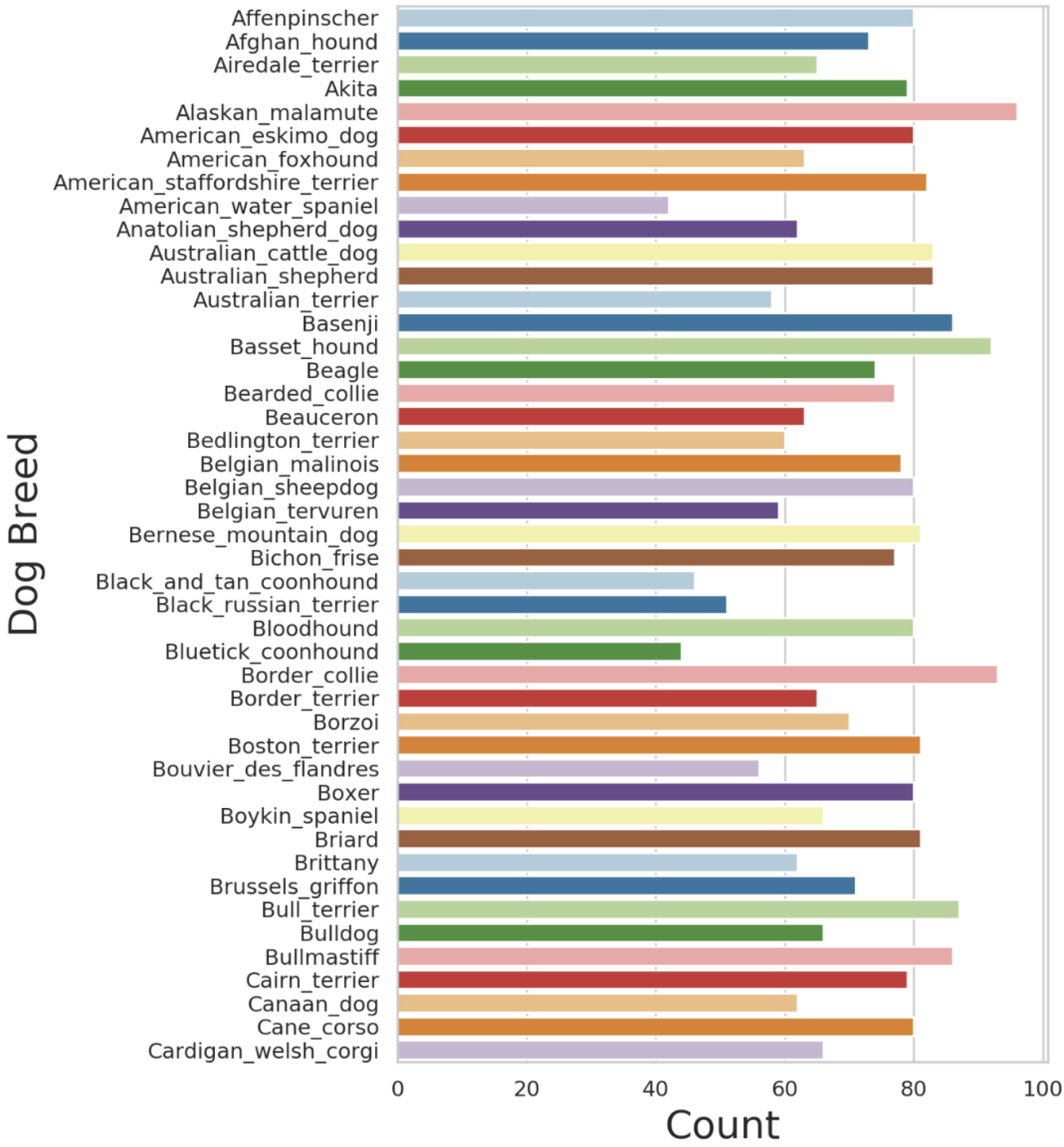
As aforementioned, there are a wide variety of dog breeds in the world and it is impossible for a human to accurately predict each breed from a given dog image. This problem can be thought of as a multi-class classification problem. Each dog breed has its own label and every dog belongs to a single breed. The goal in this problem is to accurately label, ideally more accurate than humans, a dog's breed from a given dog image and ultimately inform the user about the breed of his/her dog. In addition, if the user provides an image of another human, the application needs to inform the user that it has detected a human face.

Datasets and Inputs

There are two main datasets that are going to be used in this project. Both of these datasets have been provided by Udacity.

Firstly, the dog dataset (can be downloaded here) consists of 8351 dog images partitioned into 133 different breeds. This is the dataset that is going to be used in order to train, validate, and test the model that is going to be used in order to classify given dog images the model has not seen before as it contains a variety of images of certain breeds that we want to classify. The dataset is divided into training, validation, and test sets. The training set has a total of 6680 different images. The test and validation sets have 836 and 835 images respectively. Each image is of varying size, i.e. there is not any standard size across the dataset. In addition, all images are RGB images so that the input data has 3 different channels one for each color. One limitation to this dataset might be it falls short in providing enough data in order for the model to be able to classify all 339 different breeds in the world.

In the bar chart below the number of dog images and their corresponding breeds can be found for the first 45 breed out of the 133 breeds[2]:



In the above chart, it can be seen that the count of input image across different classes are relatively balanced. Therefore, it can be claimed that we have enough balanced data that is adequate to avoid class imbalance problem.

Another dataset that is provided is the human dataset (can be downloaded here). The human dataset is not relevant for our classification problem. Its role in the overall application is to detect any human faces that is potentially uploaded by the user and prevent the human image to be classified as a dog breed. The human dataset consists of 13233 total human images with a standard size of (250, 250). Like dog images, human images are also RGB.

Solution Statement

As aforementioned, the problem can be seen as a classification problem in the domain of machine learning. More specifically, I found that a deep learning model would prove a performant way to tackle

this problem. Therefore, the solution I propose to this classification problem is a convolutional neural network, classifying the images into 133 different classes, as they are particularly effective to find patterns given an input image and classify them. I will utilize Pytorch as my main deep learning framework and I will use transfer learning in order to classify dog breeds. The main purpose of using transfer learning is that the given training data is not enough to train a convnet from scratch. Another reason to use transfer learning is that Pytorch gives access to pretrained state-of-the-art convnet architectures which surely saves me a lot of time training and experimenting with different models. The effectiveness of convnets, or any deep learning model in general, can be expressed in a wide variety of evaluation metrics as the actual class labels and the predicted class labels are being provided.

Evaluation Metrics

The sole evaluation metric I will make use of in this problem is accuracy as the dataset is balanced. Accuracy can be defined as follows[3]:

Accuracy is the number of correctly predicted data points out of all the data points. More formally, it is defined as the number of true positives and true negatives divided by the number of true positives, true negatives, false positives, and false negatives.

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{All Samples}}$$

Benchmark Model

With my solution model I will compare my results to a benchmark model and a benchmark result. Ideally, I expect my model to surpass both of these benchmarks in terms of the evaluation metric, that is accuracy.

The benchmark model is going to be the convnet I built and trained with the given input dog training data. Then, I validated and tested this model with the validation and test sets. The network architecture I used for this model is as follows (with relu activation at the end of each convolutional layer and a dropout of probability 0.1 before the fully connected layer):

```
- (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
- (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
- (conv_bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
- (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
- (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
- (conv_bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
- (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
- (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
- (conv_bn3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
- (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
- (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
- (conv_bn4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
- (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
- (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
- (conv_bn5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
- (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
- (fc6): Linear(in_features=12544, out_features=133, bias=True)
```

The accuracy of this model on the test set was found to be 29%. So, ideally, I need to be scoring more than 29% with a pretrained model.

Another benchmark result is the one that was provided by Udacity which is 60%. So, my model should also be classifying input dog images with an accuracy of higher than 60%.

Project Design

1. Human face detector: Here a function that utilizes OpenCV's implementation of Haar feature-based cascade classifiers will be used in order to determine whether an image contains a human face or not.

2. Dog detector: Here a function that utilizes a pretrained VGG16 model is going to output the class index of the image. The model was trained with the ImageNet dataset with over 14 million images[4]. The model outputs 1000 classes however only class indices that are between 151 and 268 (inclusive) signal that a dog is detected in the image, which is enough for the sake of detecting dogs. I will not further train this model, I will only use it in order to detect whether an image contains a dog or not.

3. Data augmentation and normalization: I will first load in the data that is separated into training, validation, and test sets. Then the training set is going to have a data augmentation step, which will ideally include random rotations, flips, crops, and also color jitter. Then every image in each set is going to be resized to (224, 224) and converted to tensors and normalized.

4. Dog breed classifier (scratch): In this step I will write my own dog breed classifier model using PyTorch and ConvNets. I will experiment with different architectures and find the one that is the most performant. I will use Udacity's dog dataset in order to train, validate, and test my model.

5. Model selection: Then I will load in a pretrained model, that best suits the problem at hand, available in Pytorch as outlined in the documentation[5].

6. Model modification: After deciding upon a pretrained model, I will slightly modify the model to tackle the classification problem. More specifically, I will likely freeze the convolutional layers as I do not have enough data to further tune them and it would take a lot of time to do so. I will also modify the final fully connected layer to output 133 different classes instead of the default value. I am planning to only train the fully connected layer in the training stage in order to tune its hyperparameters and increase its accuracy for the classification task.

7. Training and validation: I will train the model with the given training data set in order for it to be able to classify dog breeds. I will also use the validation set in order to see, while training, how the model can perform on data that it has not been exposed to before.

8. Testing: After the training and validation stage has finished, I will test the accuracy of the model using the test set. If the test accuracy is more than 60% (and surpasses my custom benchmark model) I will consider it as a "successfull" model and most likely not try to further improve it for the sake of this project. Else, if it underperforms, I will experiment with different pretrained convnet architectures and/or further train the convolutional layers.

9. Writing the algorithm: In this step I will write a function that is first going to load the new image and show it to the user. Then, the function is going to determine whether the given input image contains a human or not using a prewritten function that utilizes OpenCV's implementation of Haar feature-based cascade classifiers[6]. If the face belongs to a human, the function will only output the word "Human". Else if the face does not belong to a human, the function then will check if the image belongs to a dog, also utilizing another prewritten function that uses a pretrained VGG16 model to detect dog faces. If the image belongs to a dog, my custom transfer learning model will then take the image as input and output the likelihood of each class. Then the class with the highest likelihood is going to be chosen and the corresponding class label is going to be printed to the user. Else if the image containes neither a human nor a dog, the function is going to print "Neither dog nor human".

10. Testing the algorithm: I will provide at least 6 different images that is not in the provided data set and see if the model can predict them accurately.

References

1. Dog breeds
2. Data visualization chart
3. Accuracy
4. ImageNet stats
5. Pretrained Pytorch models
6. Haar feature-based cascade classifiers