Dog Breed Classifier with CNN

Nishanth Kumar Gunda

Machine Learning Engineer Nanodegree

October 6, 2023

Definition

Project Overview

A dog breed classifier can estimate the breed of a dog from an image. This can be useful for dog owners who are unsure of their dog's breed and want to confirm it with someone or a web app by simply showing a picture. Dog breed classification is a research topic in computer vision, which is a field of artificial intelligence that deals with the understanding and interpretation of images.

Computer vision is a field of artificial intelligence that enables machines to understand and interpret images and videos. It is one of the key technologies behind self-driving cars and other applications that allow the digital world to interact with the physical world.

Most computer vision algorithms use convolutional neural networks (CNNs). CNNs are a type of machine learning model that are specifically designed to extract features from spatial data, such as images and videos. This is in contrast to traditional feedforward neural networks, which flatten the input data into a feature vector before learning from it.

CNNs have been shown to achieve state-of-the-art results on a variety of computer vision tasks, including image classification, object detection, and image segmentation. However, they require large amounts of training data to train effectively. One popular dataset for training CNNs is ImageNet, which contains over 14 million labeled images.

In this project, we built a CNN model to classify dog breeds. The training dataset was provided by Udacity and contained 8,351 dog images and 13,233 human images. The entire dog dataset has 133 breed classes, from Affenpinscher to Yorkshire Terrier. This means that a random guess of the dog breed would be correct about 1 in 133 times, corresponding to an accuracy of less than 1%.

Problem Statement

In this project, we will learn how to build a pipeline to process real-world, user-supplied images and identify the breed of the dog or the most similar dog breed to the human in the image. If the image contains neither a dog nor a human, the algorithm will indicate this. We will use transfer learning to achieve an accuracy of at least 60%. This will involve freezing the weights of all the layers in the network except for the final fully connected layer.

In summary, the following are what we need to achieve:

- The algorithm can tell if the image contains a dog or a human, or neither a dog
- nor a human
- In the presence of a dog or a human, the algorithm can give an estimate
- The accuracy of the estimate should be at least 60%
- To achieve the high accuracy, we can build a CNN model using transfer learning

Metrics

I will use accuracy as the evaluation metric because it is a common and easy-to-understand metric that is well-suited for both binary and multiclass classification problems, even if the classes are slightly imbalanced. Accuracy is calculated by counting the number of correct predictions for the dog images in the test dataset.

Analysis

Data Exploration

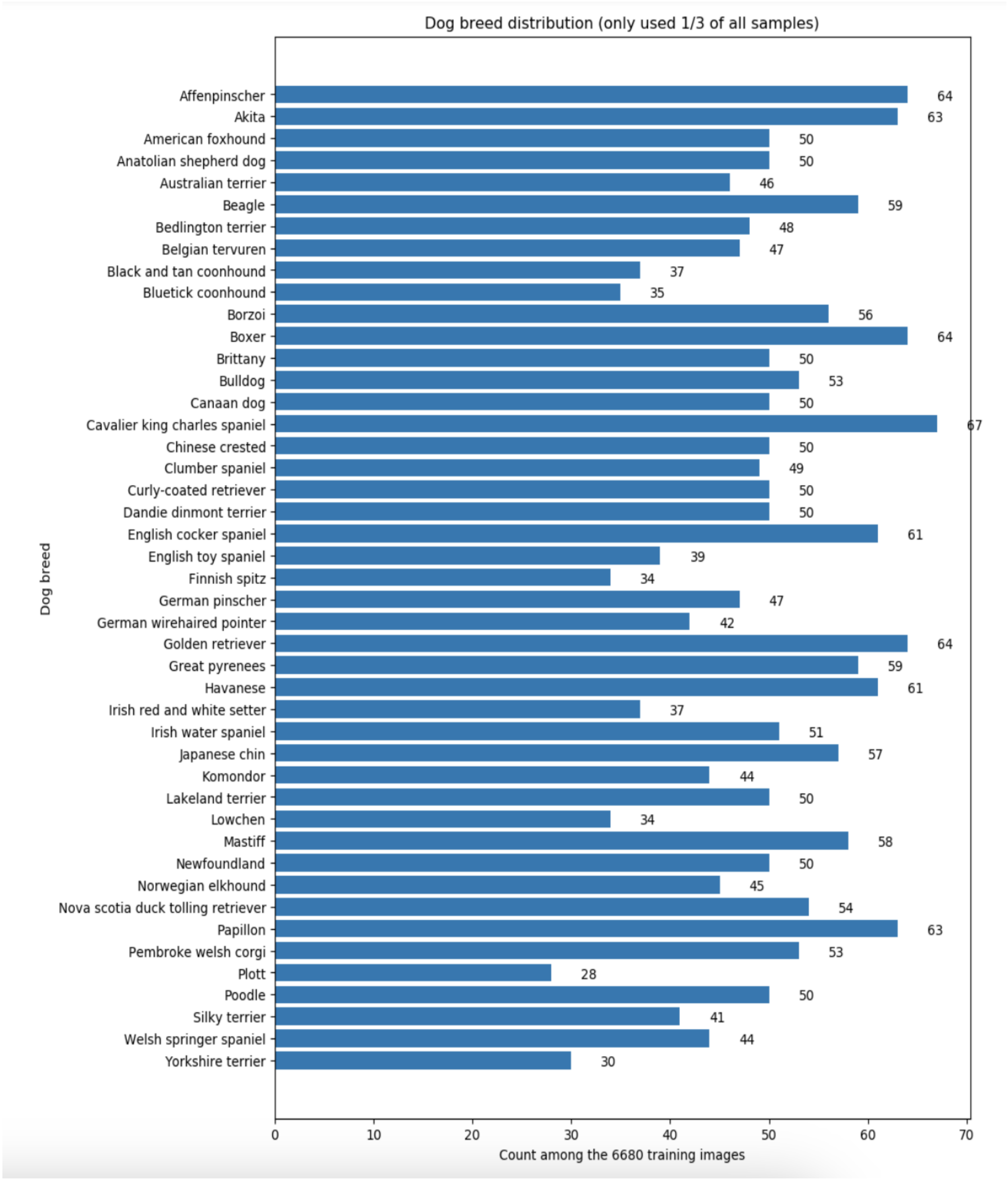
Udacity has provided two datasets for this project: a dog dataset with 8,351 images and a human dataset with 13,233 images. Since I will be using the Udacity workspace, I do not need to download the datasets from Amazon S3, as they are already stored in the /data folder. The human dataset can be used to detect human faces and to test the performance of the dog breed classifier. The dog dataset will be used to train the classifier. The dog images are located in the /data/dog_images folder and are divided into three categories:

- 6680 images under /data/dog_images/train for training
- 835 images under /data/dog_images/valid for validation
- 836 images under /data/dog_images/test for testing

The entire dog dataset contains 133 breed classes, from Affenpinscher to Yorkshire Terrier. The training data contains 6,680 images from all 133 breeds. The following visualization shows the distribution of 1/3 of the dog breeds, which were selected randomly by taking the first breed of every 3 dog breeds.

The distribution of different dog breeds is slightly imbalanced, but this should not be a problem. The average number of images per dog breed is about 50.
[1]

[1] Udacity. The dog dataset. URL:
https://s3-us-west-1.amazonaws.com/udacity-aind/dogproject/dogImages.zip
Udacity. The human dataset. URL: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip



Algorithms and Techniques

To address the problems in this project, I will design a human face detector and a dog detector to identify humans and dogs in images. Then, I will use transfer learning to design a CNN model, which I will compare to a benchmark model built from scratch. The specific parts of my approach are:

1. **A human face detector**
   OpenCV3 provides many pre-trained face detectors on GitHub, stored as XML files. We can download one of these detectors and store it locally to detect human faces in sample images. The human face detector is based on OpenCV's implementation of the Haar feature-based cascade classifier.

2. **A dog detector**
   I will use the pre-trained VGG-16 model, which was trained on ImageNet, to design the dog detector. The VGG-16 model is a large and popular deep learning model that can be used for image classification and other vision tasks. I will use the PyTorch machine learning framework to access the VGG-16 model.

   As a dog detector, I will use the loaded VGG-16 model to predict the breed of the dog in an image. I will not update the weights of the VGG-16 model.

   If the VGG-16 model predicts that the image contains a dog breed between 151 and 268, then the dog detector will conclude that the image contains a dog.

3. **A CNN to classify dog breeds from scratch**
   I will use this model as a benchmark model, and I will discuss it in more detail in the next section. Since the input dataset is not large, I will use PyTorch to design a relatively simple CNN architecture with 4 convolutional and pooling layers, followed by 2 fully connected layers.

4. **A CNN to classify dog breeds using transfer learning**
   Transfer learning is a useful technique for training a dog breed classifier with a small dataset. I will use the pre-trained ResNet-50 model, which has a good top-1 accuracy (76.130) and top-5 accuracy (92.862) on the ImageNet dataset [5]. I will replace the last fully connected layer of the ResNet-50 model with a new one with 133 output nodes, which corresponds to the number of dog breeds in my dataset. I will freeze the weights of all the other layers in the network.

5. **A dog breed classifier**
   To build the dog breed classifier, I will first use the dog detector and the human face detector to determine if the image contains a dog or a human. If it contains neither a dog nor a human, I will return an error message. If it contains a dog, I will use the CNN model from transfer learning to predict the breed.

[2]

[2] OpenCV. URL: https://opencv.org/
PyTorch. URL: https://pytorch.org/

Benchmark

In this project, I will build a CNN model from scratch as the benchmark model, because I want to know how much the transfer learning can improve the prediction accuracy. Considering the small amount of the training dataset, the architecture of the CNN model from scratch will be relatively simple, consisting of 4 convolution layers and 2 fully connected layers. I expect that the test accuracy of this benchmark model is at least 10%.

Methodology

Data Preprocessing

The data exploratory visualization section showed that the distribution of dog breeds in the dataset is slightly imbalanced, but this imbalance is not significant enough to cause a problem.

I will focus on applying data augmentation, normalization, and resizing to the input images for training and testing, both for the model from scratch and the model using transfer learning. The model using transfer learning is based on the pre-trained ResNet-50 model, which requires input images of 224x224 pixels. For convenience, I also designed the model from scratch to take 224x224 images as input. I used the PyTorch module torchvision.transforms to perform the data transformations.

To augment the training dataset, I first randomly cropped a portion of each image and then resized it to 224x224 pixels using bilinear interpolation. I also randomly flipped some of the images horizontally. For the validation and test datasets, I first resized each image to 256x256 pixels and then center cropped it to 224x224 pixels. After resizing, I normalized all of the images.

Implementation

In the section of algorithms and techniques, I listed 5 components that we will implement. Here, I would explain their implementations in more detail.

1. **A human face detector**
   The human face detector is based on OpenCV's implementation of the Haar feature-based cascade classifier. To use the detector, we first convert the image to grayscale and then apply the pre-trained classifier. Each detected face is returned as a 1D array with four entries, which specify the bounding box of the detected face. If the array is not empty, at least one face has been detected. We tested the performance of the face detector on 100 human images and 100 dog images.

[3]

[3] ResNet-50 model accuracy scores were found on PyTorch website. URL: https://pytorch.org/vision/stable/models.html

The detector successfully detected 98 human images and only missed 2. However, it also detected 17 dog images out of 100, which is higher than the expected percentage. Based on this result, I will apply a dog detector before the human face detector in the final implementation of the dog breed classifier.

2. **A dog detector**

   The dog detector is based on the pre-trained VGG-16 model, which predicts the object in an image from 1000 possible categories. The prediction indexes for dogs are 151-268, so I only need to check if the predicted index is in this range to determine if the image contains a dog. Since VGG-16 takes 224x224 pixel images as input, I also need to resize and normalize any input images.

   We evaluated the dog detector on 100 human images and 100 dog images. It successfully detected all of the dog images and none of the human images. Therefore, in the final dog breed classifier app, I will use the dog detector to check the input image before using the human face detector.

3. **A CNN to classify dog breeds from scratch**

   To establish a baseline, I trained a CNN from scratch using PyTorch. The training dataset was small, so I designed a relatively simple architecture with 4 convolutional and pooling layers, followed by 2 fully connected layers. As a common practice, the number of channels in the convolutional layers increased, while the height and width of the image decreased through max pooling. A dropout layer with a probability of 0.3 was applied after the first fully connected layer to prevent overfitting. The architecture is as follows:
   - 1st Convolution (3 input channels, 6 output channels, and kernel size 5) + Relu activation + Max pooling (kernel size 2)
   - 2nd Convolution (6 input channels, 16 output channels, and kernel size 5) + Relu activation + Max pooling (kernel size 2)
   - 3rd Convolution (16 input channels, 32 output channels, and kernel size 4) + Relu activation + Max pooling (kernel size 2)
   - 4th Convolution (32 input channels, 64 output channels, and kernel size 4) + Relu activation + Max pooling (kernel size 2)
   - Flatten all dimensions except batch
   - 1st Fully connected layer (7744, 1024) + Relu activation + Dropout (p=0.3)
   - 2nd Fully connected layer (1024, 133) as output layer

[4] ImageNet. URL: https://www.image-net.org/

4. **A CNN to classify dog breeds using transfer learning**
   I used PyTorch to load the pre-trained ResNet-50 model for transfer learning, due to its good top-1 (76.130) and top-5 (92.862) accuracy. I froze the weights of all layers except the last fully connected layer by setting requires_grad = False. I replaced the last fully connected layer in ResNet-50 (2048, 1000) with a new one (2048, 133) with random weights, and only trained this new layer.

   ```
   (fc): Linear(in_features=2048, out_features=133, bias=True)
   ```

5. **A dog breed classifier**
   To build the dog breed classifier, I will first use the dog detector and the human face detector to determine if the image contains a dog or a human. If it contains neither a dog nor a human, I will return an error message. If it contains a dog, I will use the CNN model from transfer learning to predict the breed. Please note that the input image must be resized to 224x224 pixels and normalized before being passed to the CNN predictor.

Refinement

The architecture of the CNN model, the pre-trained model for transfer learning, the optimizer, the number of epochs, data augmentation, and the loss function all affect the prediction accuracy. I experimented with different architectures for the model from scratch and different epoch numbers for the model using transfer learning, but the refinement only resulted in a small improvement in accuracy.

Results

Model Evaluation and Validation

During training, I validated both models with the validation dataset after each epoch and saved the model with the lowest validation loss. After training, I tested the models with the test dataset and calculated their prediction accuracy.
To train the model from scratch:

1. I set the number of epochs to 100, which was enough for this training.
2. I used the cross-entropy loss function by calling the `CrossEntropyLoss()` function in PyTorch.
3. For the optimizer, I selected stochastic gradient descent (SGD) with a learning rate of 0.001 and a momentum value of 0.9, because SGD works very well for image classification.

The figure above shows the evolution of the training loss and the validation loss over the epochs during training. From the figure, we can see that the model probably converged around 60 epochs, and after that, the validation loss tended to increase even though the training loss continued to decrease. However, I was not concerned about overfitting, since I saved the model with the lowest validation loss.

To train the model using transfer learning, I set the number of epochs to 50 (instead of 100, because 50 was enough). I also used the cross-entropy loss function by calling the `CrossEntropyLoss()` function in PyTorch and the SGD optimizer with a learning rate of 0.001 and a momentum value of 0.9 by calling the `SGD()` function. The following figure shows the evolution of the training loss and the validation loss over the epochs during training.

Justification

Both models were tested with 836 testing images, and following are the accuracy scores:

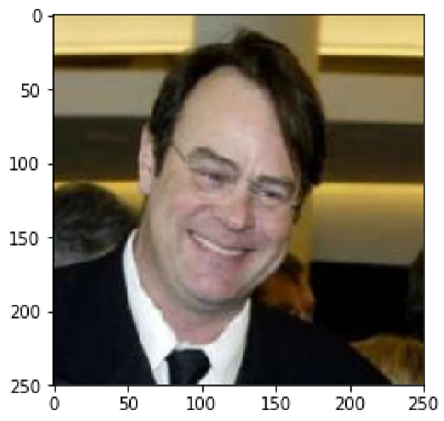
- 27% for model from scratch (benchmark model)
- 86% for models using transfer learning.

Both models have achieved the pre-defined accuracy threshold: 10% for benchmark model, and 60% for model using transfer learning. The model using transfer learning was used for the dog breed classifier, since 86% accuracy was enough to address the problem of this project.

Conclusion

Free-Form Visualization

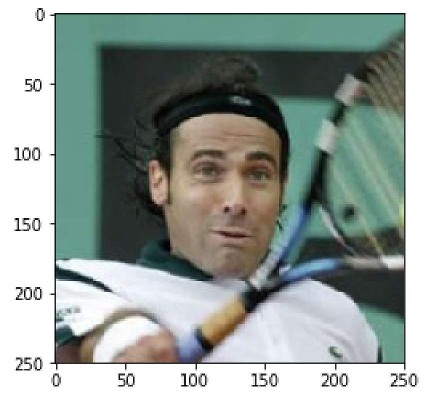
Hello Human!



You look like a Nova scotia duck tolling retriever

==

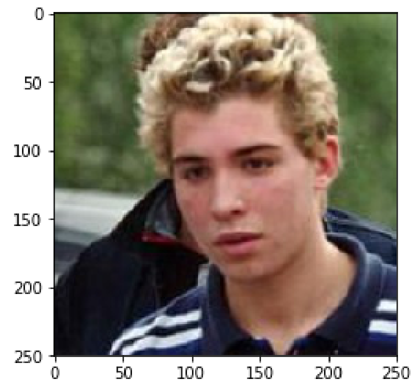
Hello Human!



You look like a Nova scotia duck tolling retriever

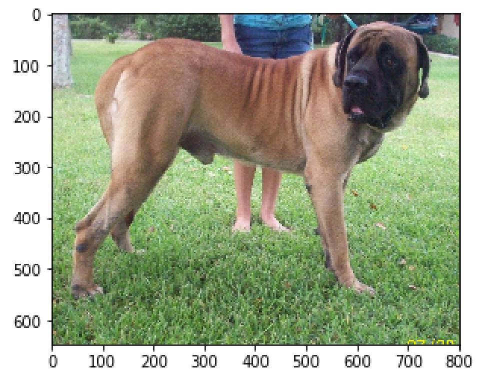
==

Hello Human!



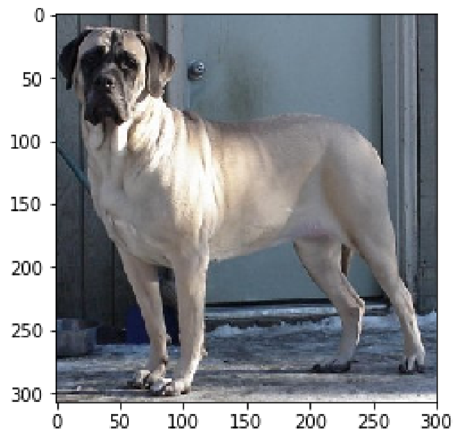
You look like a Kerry blue terrier

==



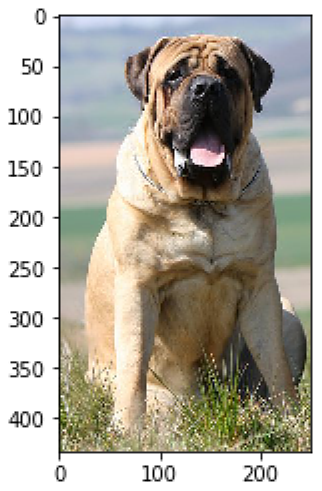
This is a picture of a ... Bullmastiff

==



```
This is a picture of a ... Mastiff

========================================
```



```
This is a picture of a ... Mastiff

========================================
```

Reflection

The process used for this project can be summarized using the following steps:

1. Import datasets
2. Design a human face detector
3. Design a dog detector
4. Preprocess the data, and divide the data into test/validation/test datasets
5. Create a CNN model to classify dog breeds from scratch (benchmark model)

6. Create a CNN model to classify dog breeds using transfer learning.
7. Combine the two detectors and CNN model using transfer learning to form the app of dog breed classifier
8. Test the app of dog breed classifier with random images

I found the step 6 the most difficult, so I had to familiarize myself with the transfer learning process using PyTorch.

Improvement

The output seems to be very good.
Here are three possible points for algorithm improvement:

- Improve the face detector algorithm by building a new neural network using transfer learning and the VGG16 network.
- Use transfer learning on the dog detector to increase accuracy.
- Increase the accuracy of the dog breed predictor by increasing the number of training episodes and possibly using a better feature detector, such as the one from the InceptionV3 network.