Classification of Dog Breeds

Shrey Samdani

December 2019

1 Definition

1.1 Project Overview

This project aims to make the first steps towards developing a dog breed classification algorithm that could be used as part of a mobile or web app. At the end of this project, the code 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.

1.2 Problem Statement

This project will need to piece together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed. Below are the steps that will lead us to solve this problem.

1. Import Datasets
2. Detect Humans
3. Detect Dogs
4. Create a CNN to Classify Dog Breeds (from Scratch)
5. Create a CNN to Classify Dog Breeds (using Transfer Learning)
6. Write the Algorithm
7. Test the Algorithm

1.3 Metrics

There are two evaluation metrics we will use to evaluate the performance of our model. The first is the prediction accuracy of our model. This is a relevant metric as it is easy to visualize and interpret - we will have a good understanding

of how our model is doing. The second is the cross-entropy loss (log loss) of our model. This metric does not provide an obvious understanding of our performance, but rather a relative understanding. It is a metric we can use to compare across models to understand which model is performing better.

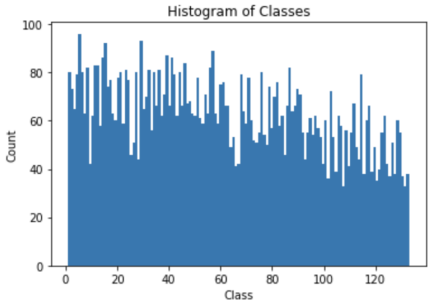
2 Analysis

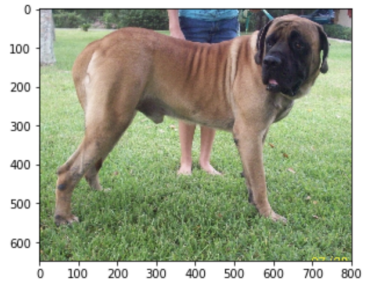
2.1 Data Exploration

The datasets for this project are provided by Udacity. The dataset with images of humans contains 13233 total images of humans. The other dataset contains 8351 dog images along with their breed. There is some imbalance in our dataset, as there are more human photos than dog photos, but this is not an issue as we will only be training on the dog photos. Within the dog photos, there is also some imbalance between the 133 different classes, which may pose an issue during training. The images are also in different shapes and sizes, with varying locations of where in the image the dog actually stands. The average size of an image is 567x520 pixels.

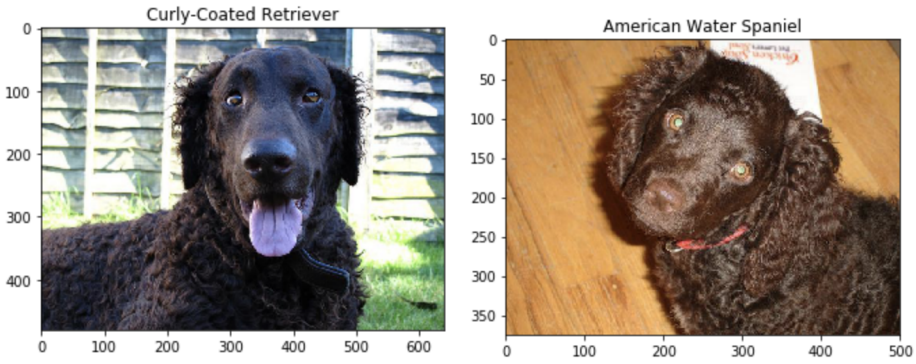
2.2 Exploratory Visualization

We can see below the histogram of the classes and an example image. The histogram shows the distribution of the class labels, which we can see to be somewhat imbalanced.





Another issue we run into is the similarity of looks between certain dog breeds. Consider the example below:



These two breeds look very similar, which may cause difficulties in their classifications. There are more such examples in our data.

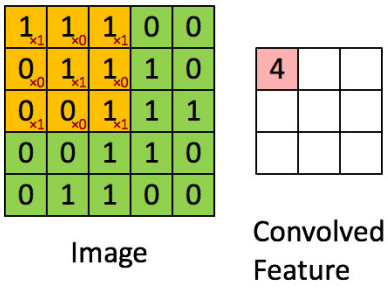
2.3 Algorithms and Techniques

There are two main algorithms/models that will be used in this project.

1. Haar feature-based cascade classifiers The first is the use of a Haar feature-based cascade classifiers provided to us by OpenCV. This classifier is one that can identify the location or presence of a human face. In our project, we import this classifier and treat it as a black-box, as it is well tested and works with good accuracy.

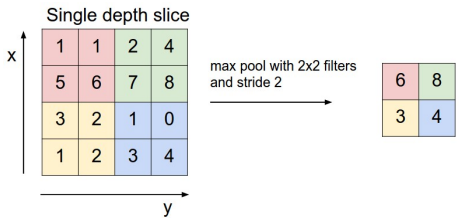
2. The second model that needs attention is a Convolutional Neural Network (CNN). This is a type of neural network that builds upon a naive neural network by adding preprocessing layers in the form of convolutional layers and pooling layers.

   A convolutional layer applies a sliding square filter to an image and extracts the sum of an element-wise multiplication over this filter. To visualize this, observe the image below.



Source: medium.com

A max pooling layer, on the other hand, takes a sliding filter and finds the max of the elements within the filter. To visualize this, observe the image below.



Source: http://cs231n.github.io/convolutional-networks/

A CNN will take combinations of these layers and add them to simpler neural nets. In practice, CNNs have been found to perform well on image data.

2.4 Benchmark

For the CNN created from scratch, we hope to attain an accuracy of at least 10%. Given that a random guess has a 1/133 probability of being correct, 10% accuracy would be a good benchmark to show that our model does better than random guessing (chosen arbitrarily). For the transfer learned model, we hope to attain an accuracy of at least 60%. Since this model is based on a more well tested neural net (90%+ accuracy on other image sets), we should hope to achieve at least 60% (lower due to the high number of classes we have).

3 Methodology

3.1 Data Preprocessing

To preprocess our data, we first standardize our inputs to a similar metric. In specific, we first resize the image to be a 256x256 pixel image. We then crop the center 224x224 pixels of the image. After we have our image, we normalize it accoridng to a set mean ([0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]) where each element corresponds to R,G,B values of the image.

For the training data specifically, we augment our training space by applying transformations. Instead of taking a center crop as we did above, we take a random crop of 224x224 pixels. Next, we apply random rotaions of degree size 15, randomize the brightness/contrast/saturation, and apply random horizontal flips. We apply these transformations in the hope that the model will learn features rather than memorizing input images.

3.2 Implementation

From above, there are 4 separate implementations that we aimed to complete:

1. Detect Humans For this task, as mentioned above, we used OpenCV's Haar feature-based cascade classifier. This performed as we expected to, with no errors.

2. Detect Dogs For this task, we do not need perfect classification of the dog breeds. PyTorch provides a pretrained VGG 16 CNN, a well known network that will classify if an image has the breed of a dog, or if it does not. We apply the preprocessing defined above, pass in the image into our model, and record the output. Like the human detection, this model performed quite well.

3. Create a CNN to Classify Dog Breeds (from Scratch) This is where things started becoming trickier. To create the CNN, we starteed with a simpler 'unit'. It is common to group together a convolutional layer, a batch normalization, and a relu activation. As such, I created a class to combine these three functions into one unit. I then added created a sequential model by alternating 2 units and a max pooling layer. As I added more units, I increased the output size until it was 256. In the end, in order to make the output classify a dog breed, I added a fully connected layer with 133 outputs.

   As a result, the model looked like the following:

```
  Net(
(unit1): Unit(
  (conv): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (bn): BatchNorm2d(16)
  (relu): ReLU()
)
(unit3): Unit(
  (conv): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
  (bn): BatchNorm2d(16)
  (relu): ReLU()
)
(pool1): MaxPool2d(kernel_size=2, stride=2, padding=0)
(unit4): Unit(
  (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (bn): BatchNorm2d(32)
  (relu): ReLU()
)
(unit6): Unit(
  (conv): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (bn): BatchNorm2d(32)
  (relu): ReLU()
)
(pool2): MaxPool2d(kernel_size=2, stride=2, padding=0)
(unit7): Unit(
  (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (bn): BatchNorm2d(64)
  (relu): ReLU()
)
(unit9): Unit(
  (conv): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (bn): BatchNorm2d(64)
  (relu): ReLU()
)
(pool3): MaxPool2d(kernel_size=2, stride=2, padding=0)
(unit10): Unit(
  (conv): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (bn): BatchNorm2d(128)
  (relu): ReLU()
)
```

```
  (unit12): Unit(
    (conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
    (bn): BatchNorm2d(128)
    (relu): ReLU()
  )
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0)
  (unit13): Unit(
    (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
    (bn): BatchNorm2d(256)
    (relu): ReLU()
  )
  (unit15): Unit(
    (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
    (bn): BatchNorm2d(256)
    (relu): ReLU()
  )
  (seq): Sequential(
    (0): Unit(
      (conv): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(16)
      (relu): ReLU()
    )
    (1): Unit(
      (conv): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(16)
      (relu): ReLU()
    )
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0)
    (3): Unit(
      (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(32)
      (relu): ReLU()
    )
    (4): Unit(
      (conv): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(32)
      (relu): ReLU()
    )
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0)
    (6): Unit(
      (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(64)
      (relu): ReLU()
    )
    (7): Unit(
      (conv): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(64)
      (relu): ReLU()
    )
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0)
    (9): Unit(
      (conv): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(128)
      (relu): ReLU()
    )
    (10): Unit(
      (conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(128)
      (relu): ReLU()
    )
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0)
    (12): Unit(
      (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(256)
      (relu): ReLU()
    )
    (13): Unit(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
      (bn): BatchNorm2d(256)
      (relu): ReLU()
    )
  )
  (fc): Linear(in_features=9216, out_features=133, bias=True)
)
```

4. Create a CNN to Classify Dog Breeds (using Transfer Learning) For this task, I imported a well known pre-trained CNN known as the DenseNet 161, a model where each layer is connected to every other layer in front of it. This model has parameters that have been trained extensively, and we want to transfer its knowledge to account for our dataset. To do so, we remove the final layer of the network, the fully connected layer. We instead replace it with a fully connected layer of our own that has 133

different outputs. Since we only need to train this last layer, we freeze each of the other layers and train our model only on the last layer for 15 epochs. As a result, we are left with a variant of DenseNet 161 that is fitted to our dataset.

For implementations of the actual models, please see the corresponding Jupyter notebook.

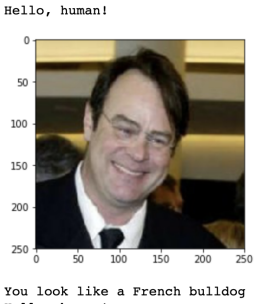
3.3 Refinement

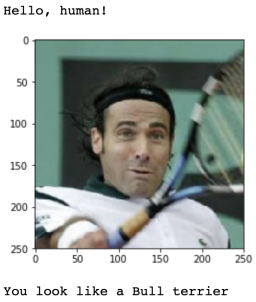
For tasks 1,2 and 4 above, the initial creation of our model met our standards and needed no current refinement. Task 3, however, did not initially meet the criteria specified in the beginning. The initial model was very simple, and alternated between a max pooling layer and a unit. Since we did not achieve the desired accuracy, one improvement was to increase the complexity of the model. As such, I continued to add more units, making the model more complex. The models were also initially being trained on 10 epochs. However, with such training, it was noted that the validation accuracy only decreased each subsequent epoch; the model never overfit. As a result, I increased the number of training epochs to 15. By iterating over this process by changing the number of epochs and the complexity of the model, I was able to achieve the desired benchmarks.

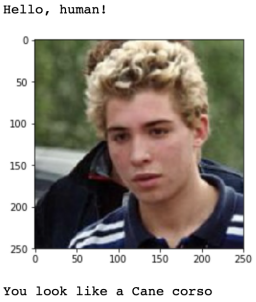
4 Results/Conclusion

The CNN made from scratch performed with 12% accuracy, exceeding the 10% benchmark and the transfer learned network performed with 71% accuracy, also exceeding the 60% benchmark. In the final result, we use the transfer learned model, as it performs with much better accuracy. In order to test out the benchmark, we can evaluate our model on 6 sample images below:

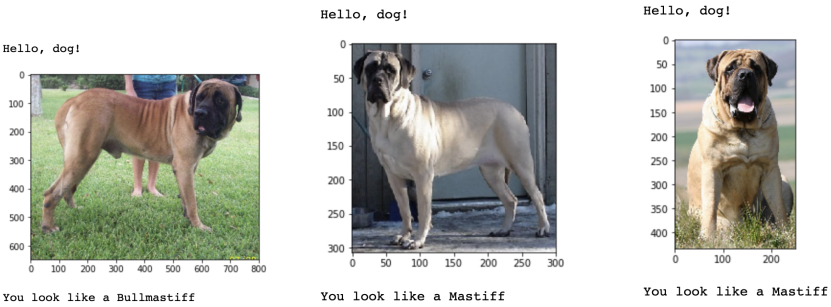
Humans:







Dogs:



We can clearly see that our models perform as expected to.

From an overall perspective, this project expresses the power of CNNs and transfer learning to create models that can predict difficult tasks with relatively high accuracy. It was clear that starting off with a model from scratch posed a lot of difficulties, but transfer learning a model could build on top of what was already done to achieve our results. In the future, we can hope to increase our accuracy to even higher numbers and implement a production version of the algorithm.