Capstone Project Report **Chris Francis**

Machine Learning Engineer Nanodegree April 11, 2020

Dog Breed Classification using CNN

Definition

Project Overview

Image classification is an exciting area of research and is an important problem with many applications. The task I have chosen is to identify the most resembling dog breed from an image. Hence, it is a multi-class classification problem. There was a Kaggle competition with a similar theme: Dog Breed Identification.

In this project, I created a CNN model from scratch and a CNN model using transfer learning to identify the closest resembling dog breed from an image. The trained model was saved and then later loaded into a web app made using Flask. The web app works on a local machine(it is not deployed) and allows users to upload an image and determines whether the uploaded image contains a human, dog, or neither.

- ➔ If a dog is detected in the image, it returns the predicted breed.
- ➔ If a human is detected in the image, it returns the resembling dog breed.
- ➔ If neither is detected in the image, it provides an output that indicates an error.

Problem Statement

The problem is to employ machine learning to identify dog breeds from images and to integrate a web app for receiving images supplied by users. This problem has two subtasks as follows:

- ➔ Identifying the most likely breed of dog, if an image of a dog is supplied.
- ➔ Identifying the closest resembling dog breed if the image of a human face is supplied.

A potential solution could be to use Convolutional Neural Networks(CNN). CNNs are popular in image classification problems due to their high accuracy in such tasks.

Metrics

Since the dataset is unbalanced, accuracy is not the best choice as a metric. Accuracy has a yes or no character. It counts the number of times the predicted label matches the actual label. Multi-class log loss is a better option for this problem, since it considers the uncertainty of the predicted label by checking how much the prediction varies from the actual label. The Kaggle competition on Dog Breed Identification also used multi-class log loss for evaluation. In the code, I have used cross-entropy loss, which is basically the same as log loss.

For binary classification, log loss is defined as:

$$-(y\log(p) + (1-y)log(1-p))$$

where y is the actual label and p is the predicted probability.

For multi-class classification, we take the sum of log loss values for each class prediction in the observation:

$$-\sum_{c=1}^{M} y_c \log(p_c)$$

where c represents a particular class, M is the total number of classes, y_c is 1 if the image actually belongs to class c and is 0 if the image actually belongs to some other class than class c, and p_c is the predicted probability of the image belonging to class c.

Analysis

Data Exploration

There are two datasets for this problem. One consisting of images of dogs and the other consisting of images of humans.

Dog Images Dataset

This dataset consists of 8351 images of dogs divided into train, validation and test datasets. Each of these sets has 133 folders - each folder containing images of a particular breed of dog. The number of images supplied for each breed varies. Hence the data is not perfectly balanced. Backgrounds, sizes, lighting and angles may vary from image to image.

Fig 1: Sample images from the Dog Images Dataset

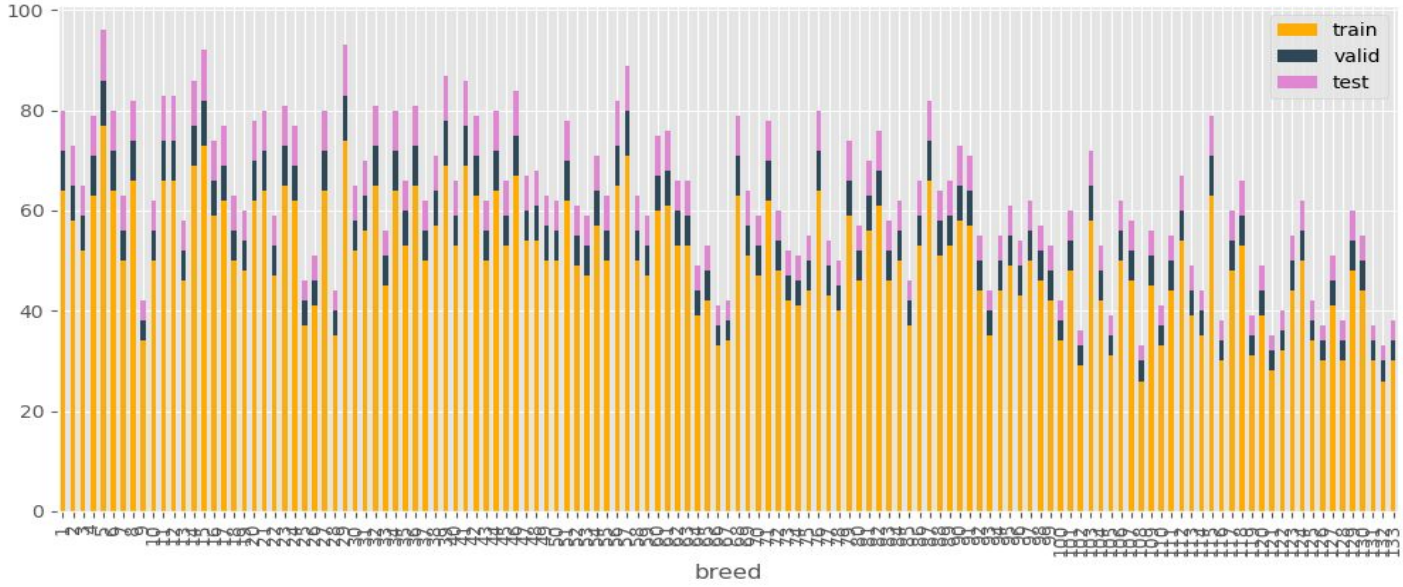
Human Images Dataset

This dataset consists of 13233 images of humans. All images are of the size 250 x 250 pixels. Here too, the dataset is not perfectly balanced as the number of images of some people are higher than others. Backgrounds, lighting and angles may vary from image to image. Also, there could be more than one human face in an image.

Fig 2: Sample images from the Human Images Dataset

Exploratory Visualization

The following plot shows the distribution of the images in the Dog Images Dataset:



Plot 1: The y-axis has the count of images and the x-axis has all the 133 breeds represented by numbers.

The plot shows how the number of images of each type of breed is different. This means that the dataset is imbalanced. This is why I have used log loss as a metric instead of accuracy. The plot also gives a sense of how large the number of classes is.

Algorithms and Techniques

I used Convolutional Neural Networks(CNN), which are state of the art in image classification. The large requirement of training data for CNN was met by the dataset.

CNN made from scratch

In the CNN made from scratch, I used three convolutional layers of stride = 1 and kernel size = 3. Layer one has three input channels and the final convolutional layer has 128 as output size. A (2,2) pooling layer was used and two fully connected layers were used, whose output has 133 dimensions. The dropout is used to prevent the model from over-fitting. The activation function used is relu.

CNN made using transfer learning

I used the ResNet152 model which is pre-trained on the Imagenet dataset. I had earlier tried a ResNet101 model but the loss was greater for ResNet101. Also, loss is a better metric than accuracy in this problem as the dataset is imbalanced.

Steps:

1. Import pretrained ResNet152 model.

2. Change out_features of fully-connected layer to 133, since there are 133 classes in this classification problem.

I think this architecture is suitable for the current problem since ResNet is a model that works very well in image classification and is also easily available for use as a PyTorch Model.

Benchmark

As a benchmark I am choosing a minimum accuracy of 10 percent for the CNN written from scratch and a minimum accuracy of 60 percent for the CNN implemented via transfer learning. These benchmarks are in accordance with the values suggested by Udacity in the project workspace.

However, since I received a suggestion in my proposal review to use the evaluation metric that I have chosen for setting a benchmark, I have decided to use log loss(cross entropy loss) for setting benchmarks. Thus, as benchmarks, I am setting a log loss of 4 for the CNN created from scratch and a log loss of 0.7 for the CNN created using transfer learning.

Methodology

Image preprocessing

I have used the size 224x224 since pre-trained models usually use that size for input.

- ➔ I applied normalization on train, test and validation datasets.
- ➔ I applied RandomResizedCrop, RandomRotation and RandomHorizontalFlip to the train dataset.
- ➔ I applied Resize to the test and validation datasets.

Human face detection

I used OpenCV's implementation of Haar feature-based cascade classifiers to create a function that detects human faces in images. This step can be substituted with a deep learning based approach.

Dog detection

I used the pre-trained VGG-16 model to create a function that detects dogs in images. Models like VGG-19 or Inception-v3 or ResNet50 can be used too.

CNN from scratch

I used three convolutional layers of stride = 1 and kernel size = 3. Layer one has three input channels and the final convolutional layer has 128 as output size. A (2,2) pooling layer was used and two fully connected layers were used, whose output has 133 dimensions. The dropout is used to prevent the model from over-fitting. The activation function used is relu. I trained, validated and tested this model and saved the model as 'model_scratch.pt'.

CNN using transfer learning

I created a CNN using transfer learning with a ResNet152 model to classify dog breeds from images. This was done using PyTorch and the model was trained, validated, tested and saved as 'model_transfer.pt'.

Algorithm to process images

Wrote an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. The dog detection and human detection functions made earlier were used in this algorithm to detect dogs and human faces.

- ➔ If a dog was detected in the image, it was passed to the CNN and the predicted breed was returned.
- ➔ If a human was detected in the image, it was passed to the CNN and the predicted resembling breed was returned.
- ➔ If neither was detected in the image, an output that indicates an error was returned.

Web App

I used the same algorithm and loaded the trained model 'model_transfer.pt' in the web app. The app was made using flask and had two web pages. The first page asks the user to choose an image from their computer. This image gets copied to the directory of the Flask web app and the python script performs inference and outputs a predicted breed. This prediction is shown on the next web page. The output is in accordance with the 3 cases mentioned in the above section. The user can then return and try another image. Once the app is stopped, the script deletes all the copies of the images collected from the user.

Results

Model Evaluation and Validation

Dog Detection Function:

This function is made using the VGG-16 model and was tested on 100 images of dogs and 100 images of humans.

It had an accuracy of 100% of the time on both the sets.

Human Face Detection Function

This function was made using OpenCV's Haar feature-based cascade classifiers and was tested on 100 images of dogs and 100 images of humans.

It had an accuracy of 98% with human images and 83% with dog images.

CNN made from scratch

The model achieved an accuracy of 24% in 100 epochs of training which is more than twice of the benchmark(10%). It also achieved a loss of 3.090314 which is better than the benchmark of 4.

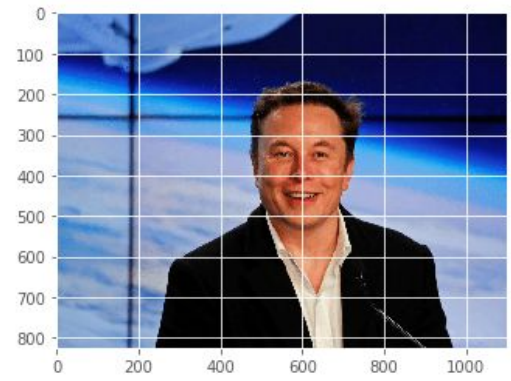
CNN made using transfer learning

The model achieved an accuracy of 84% in 10 epochs of training which is more than the benchmark of 60%. It also achieved a loss of 0.503607 which is better than the benchmark of 0.7.

Testing with other images from my computer

I tried out the following images and got results as follows:

```
Hi! You seem to be a human!
```



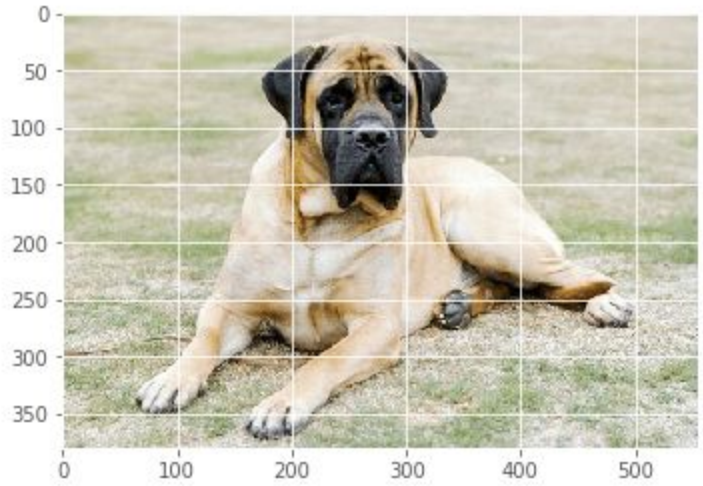

```
You look like a:  Afghan hound
```

```
Hi! You seem to be a dog!
```


```
You look like a:  Chinese crested
```

Which is correct, it is a Chinese crested.

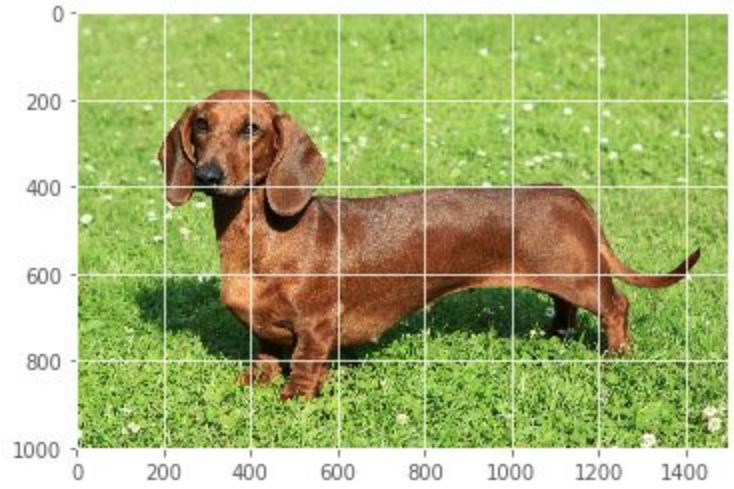
```
Hi! You seem to be a dog!
```




```
You look like a:  Mastiff
```

This is also correct. It is an image of a Mastiff.

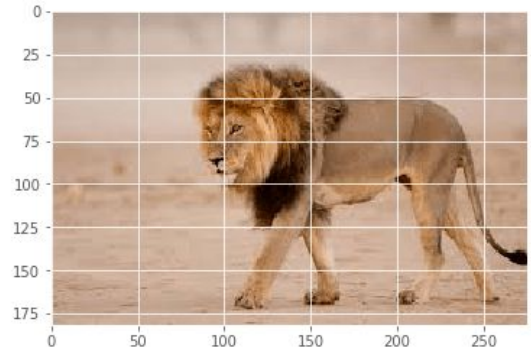
```
Hi! You seem to be a human!
```




```
You look like a:  Dachshund
```

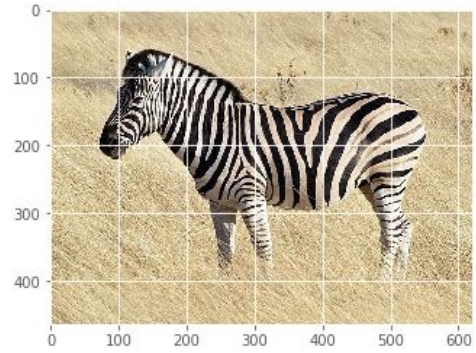
It detects the dog wrongly as a human, but the breed is correct, it is a Dachshund.

```
Image is not of a dog or a human. Please try with an image of a dog or a human.
```



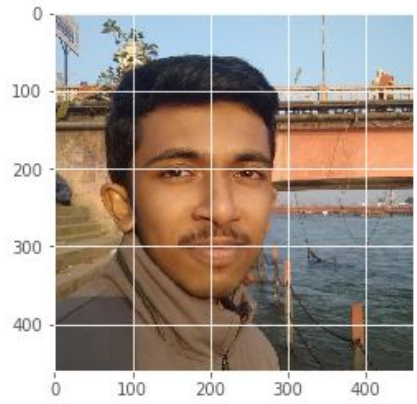
This is also correct, since it's neither a dog nor a human.

```
Image is not of a dog or a human. Please try with an image of a dog or a human.
```



This is also correct, since it's neither a dog nor a human.

```
Hi! You seem to be a human!
```




```
You look like a:  Yorkshire terrier
```

And finally I got to know that I look like a Yorkshire Terrier.

Justification

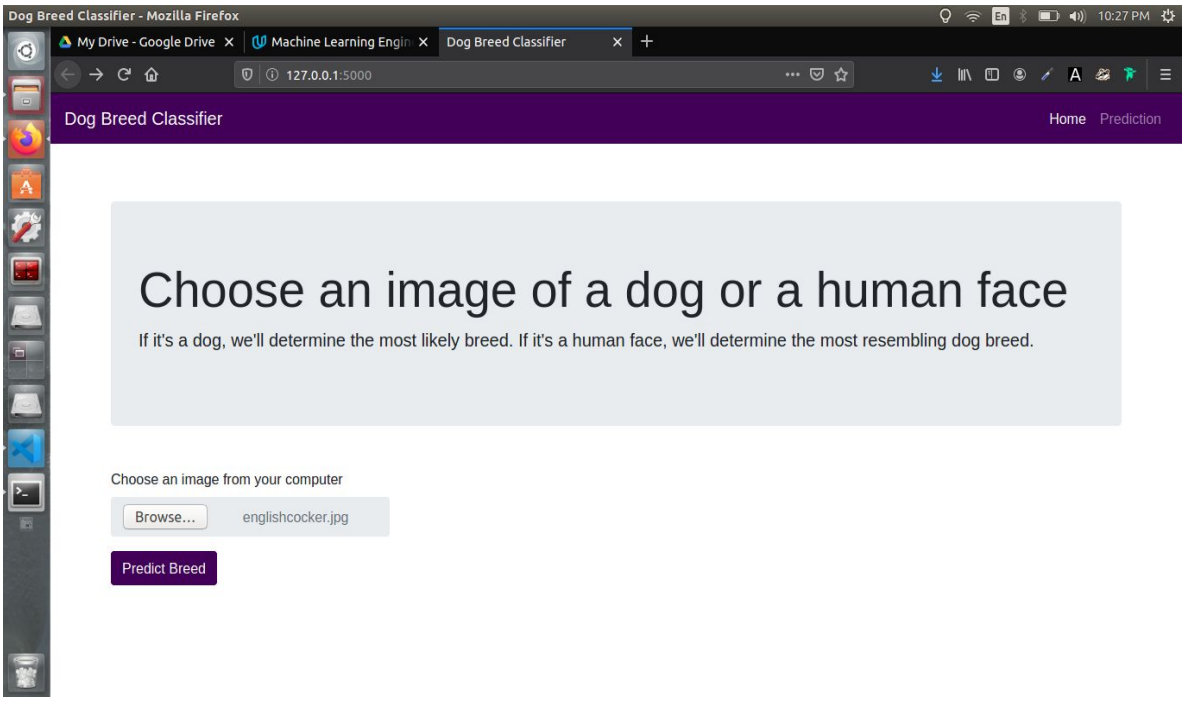
The model performed better than I expected, especially since it was a multi-class classification problem with 133 classes and an imbalanced dataset. It performed better than the benchmarks I had set.

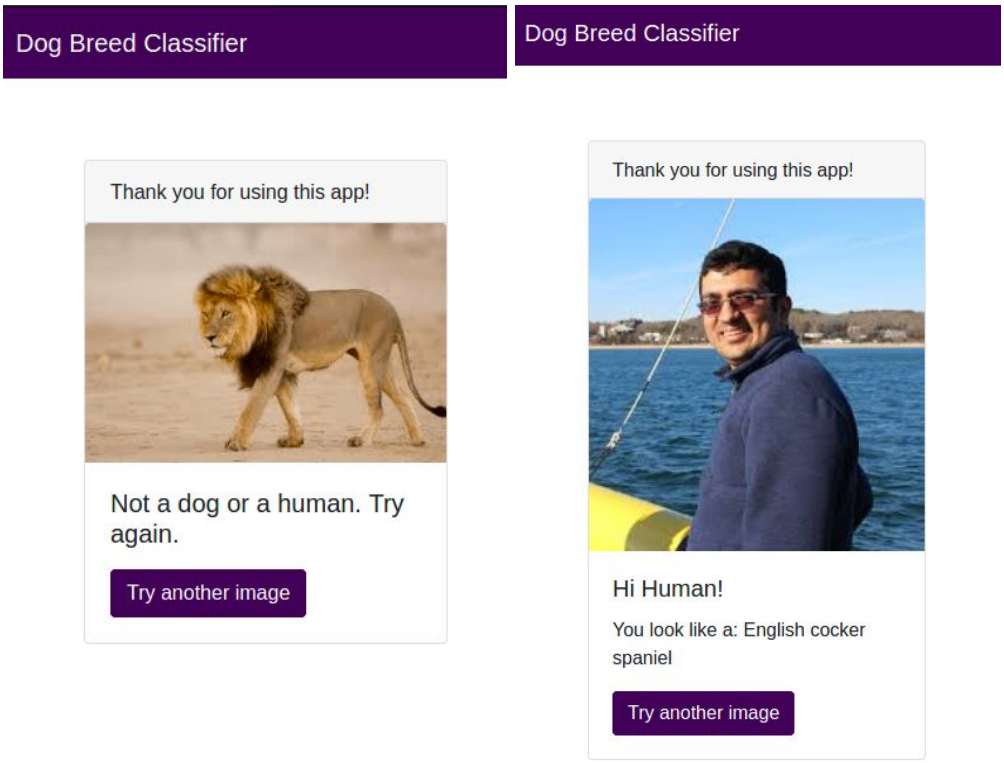
Conclusion

Web App Images

Here is the link to a YouTube video in which I show my web app working.

Here are some screenshots:





Reflection

I enjoyed working on this project and it gave me the confidence to do bigger projects. The hardest steps for me were building the CNN from scratch and using the saved model to create a working web app.

Improvement

Points for improvement are:

1. Perform hyper-parameter tuning.
2. Use better image augmentation techniques on the images.
3. Collect more data. (more number of images)
4. Use a better model for dog detection such as VGG-19.
5. Use a deep learning algorithm for detecting human faces instead of haar cascades classifier.

References

1. An article on CNN: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
2. PyTorch Documentation: https://pytorch.org/docs/stable/nn.html
3. PyTorch Documentation: https://pytorch.org/docs/stable/nn.functional.html
4. Log Loss: http://wiki.fast.ai/index.php/Log_Loss
5. An article on Capstone Projects: https://medium.com/@mmatterr/tips-for-udacity-machine-learning-engineering-nanodegree-capstone-project-1110ce5b8cd0
6. A GitHub repo on CNN benchmarks: https://github.com/jcjohnson/cnn-benchmarks
7. PyTorch pretrained models: https://pytorch.org/docs/stable/torchvision/models.html
8. Kaggle Competition on Dog Breed Identification: https://www.kaggle.com/c/dog-breed-identification/overview/description