**Capstone Project Report**
**Machine Learning Engineering Nanodegree**
**October 2020**

# Classifying Dog Breeds

## Project Overview

Computer vision is a field in which machine learning and in particular deep learning has been able to provide significant breakthroughs. Through the use of Convolution Neural Networks, the computer is able to 'learn' about features in the images effectively. They were inspired by biology, where the connectivity pattern between neurons is similar to that of the visual cortex.
Classifying dog breeds is a well known problem and will be used to illustrate the above. This project will allow me to learn about transfer learning, as we will use various pre-trained models. As future work, I would look to focus on deployment and create a web app utilising AWS SageMaker.

## Problem statement

The problem is a multi-class classification problem in the field of computer vision. There are two aspects to the problem:
1. Given an image of a dog, we want our model to accurately predict the dog's breed
2. Given an image of a human face, we want our model to identify the closest resembling dog breed.

To tackle this problem, we will use Convolution Neural Networks (CNN) due to their known success in the field of computer vision. Details of the proposed model architectures are discussed in the Algorithms and Techniques section below.

## Evaluation metrics

When training our model, we will use **cross entropy loss** to choose the best model, whilst iterating over many epochs. This metric is suited to multi-class classification problems and will ensure our model's performance can be relied on.
When evaluating our overall model performance on test data, we will use **accuracy** as it gives a clear indication of how well the model has identified dog breeds.

## Data exploration

The data have been provided by Udacity. The input type should be images, as we want to detect and classify dog breeds.

Dog images dataset: The data has been already split into three folders, for train, test and validation purposes. Each folder is further subdivided into 133 folders representing the different dog breeds we are seeking to classify. Note that we are dealing with imbalanced data as there is not the same number of images for each class (breed).

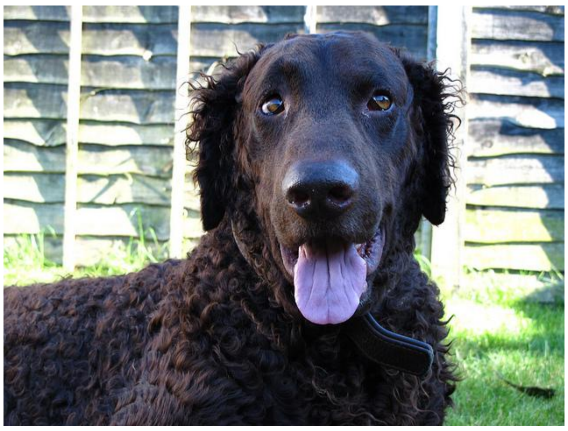
Human images dataset: The data has been split into folders according to names. There are a total of 13233 human face images divided into 5749 folders. Again we note that there is not the same number of images in each folder.

The preprocessing steps we will need to complete are as follows:

- the human images will need to be converted to grayscale before using OpenCV's Haar Cascade classifier and a bounding box for each detected face will need to be established
- The dog images will also need to be converted to grayscale and resized to 224 x 224 pixels. We will need to transform the input to a 4D tensor to use Keras CNNs.

## Exploratory Visualization

Sample images are provided below.



As discussed above, we note the need to resize the images and convert to grayscale.

## Algorithms and Techniques

We will use Convolution Neural Networks (CNN) to solve this problem due to their known success in the field of computer vision. We will use existing algorithms and pre-trained models to ensure better results (making use of transfer learning tools.) In particular, we will use the following:

- To detect human faces, we will use OpenCV's Haar Cascades classifiers
- To detect dog images, we will use a pretrained VGG16 model, which was trained on the ImageNet dataset
- To detect breeds, we will both implement a CNN from scratch and use a pre-trained Resnet-101 model.

## Benchmark model

A random guessing model would accurately predict the dog breed once in 133, as there are 133 classes. This would give an accuracy score of less than 1%.

We would expect our CNN model implemented from scratch to achieve at least 10% accuracy on the test set.

Our CNN model which uses transfer learning should achieve at least 60% accuracy.

## Data Preprocessing

The images were all resized to 224 x 224 pixels and converted to grayscale.

For the training set, we performed image augmentation to combat overfitting. We introduced random rotation and random horizontal flips.

The images were reshaped to tensors, which is the required input format for CNNs.

## Implementation

A CNN was implemented from scratch to tackle the problem.

It has 3 convolutional layers and 2 fully connected layers. The first convolutional layer has input 3 corresponding to the three channels and the final convolutional layer has an output of 128. We added a pooling layer to reduce the input size. The final fully connected layer has an output of 133 corresponding to the 133 classes. After seeing the initial performance, we added a dropout layer and normalisation layer to combat overfitting.

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=100352, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=133, bias=True)
  (dropout): Dropout(p=0.2)
  (batch_norm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

CNN from scratch model architecture

## Refinement

Whilst our CNN from scratch model met our benchmarking requirements, we explored transfer learning tools to improve performance.

I selected a pre-trained VGG16 model.This model is suitable as it has been trained on ImageNet data and the classification task at hand is similar.
I changed the classifier to be 1 fully connected layer, with output 133 corresponding to the number of classes.

CNN from scratch had an accuracy of 15% whereas this model increased accuracy to 80%, showing an impressive improvement.

## Model Evaluation and Validation

<u>Human face detector</u>
We used OpenCV's implementation of Haar Cascade classifiers.
We tested performance on the first 100 human and first 100 dog images in our dataset. The results were as follows:
Percentage of human faces detected in the human images: 98%
Percentage of human faces detected in the dog images: 17%
Whilst not perfect, the model performed adequately for our purposes.

<u>Dog detector</u>
We used a pre-trained VGG16 model.
We tested performance on the first 100 human and first 100 dog images in our dataset. The results were as follows:
Percentage of dogs detected in the human images: 0%
Percentage of dogs detected in the dog images: 96%
The model performed well, with no misclassifications.

CNN from scratch
We trained the CNN model described above for 20 epochs, specifying our loss function to be Cross Entropy and the optimiser to be SGD.
The test accuracy was 15%, correctly predicting 133 out of 836 images.

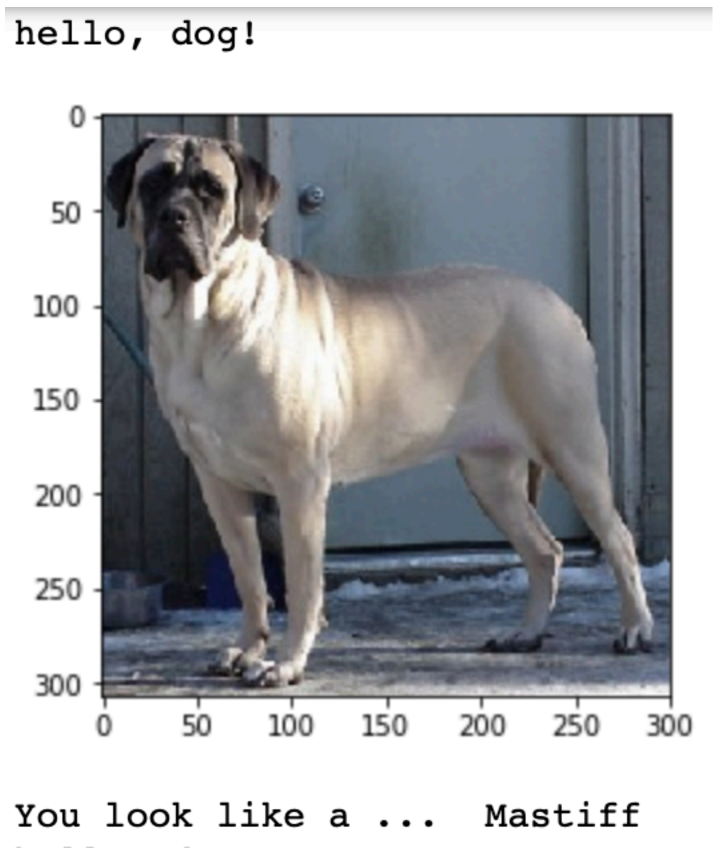
CNN using transfer learning
We trained the transfer learning (VGG16) CNN model for 10 epochs specifying our loss function to be Cross Entropy and the optimiser to be ADAM.
The test accuracy was 80%, correctly predicting 673 out of 836 images.

## Justification

The models performed better than expecting, exceeding our benchmark.
The CNN from scratch model achieved an accuracy of 15% (benchmark was 10%) and the CNN with transfer learning model achieved an accuracy of 80% (benchmark was 60%).

We will use the CNN with transfer learning model for the next step of the project (beyond the scope of this Capstone) to create a web app. With an accuracy of 80%, it performs sufficiently well for our purposes. Note that the human eye would struggle to identify some very similar dog breeds, especially with 133 different classes. As such we are pleased with our machine learning result.

## Sample output



```
hello, dog!
```

```
You look like a ...  Mastiff
```

## Improvements/ next steps

The output is good, I am pleased with the performance of the algorithm. However, to improve performance further, I would suggest the following steps:

- gather additional data/images
- further tweaking hyperparameters (e.g. learning rate, optimizer choice)
- running the model over a higher number of epochs
- Trying different transfer learning models, e.g. Resnet for the breed classification instead of VGG16

## References

- Udacity Repository: https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
- Pytorch Documentation: https://pytorch.org/docs/master/
- https://debuggercafe.com/transfer-learning-with-pytorch/