Machine Learning Engineer Nanodegree

Capstone Project Report

Chidimma Nweke

November 27, 2020

Definition

Project Overview

There are currently over 400 different breeds of dogs across the world, both of pure and cross breeds. We love our furry friends and we love to know them half as well as they know us. Before now, identifying dog breeds was not as tedious as it is now, as there were less hybrids and more of pure breeds, and one could tell them apart by just taking a rough look at them. But recently, there are more breeds with close resemblance that it takes even a professional some time to successfully identify these breeds.

In this project, I created an app using deep Convolutional Networks and Haarcascade face detectors to detect and classify the breeds of dogs present in an image or return the closest breed a human in an image resembles.

Problem Statement

The goal is to create an app using deep learning. The task involved includes the following:

1. Download, explore and preprocess the dog images and the lfw data.
2. Build face and dog detector models.
3. Train a CNN to classify the dog breeds.
4. Bundle the models into a function which can be deployed as a web app.

The final application is expected to be useful for classifying dog breeds.

Metrics

- ❖ Model accuracy

Model accuracy here is the percentage ratio of the correctly classified images to the total number of images in the dataset.

Mathematically:

$$accurcy = \frac{correct\ predictions}{size\ of\ dataset}\ x\ 100\%$$

This metric takes only the true negatives and positives into consideration.

- ❖ Cross Entropy Loss

This metric actually involves two steps:

1. The first applies a softmax function to any output it sees.
2. Then applies Negative Log Like Likelihood Loss (NLLLoss).

It then returns the average loss over a batch of data.

Mathematically:

$\ell_{CE}(p, t) = -[t \log p + (1 - t) \log(p - 1)]$

Where: t = target, and p = prediction

Analysis

Data Exploration

The dataset for this project is the dog dataset provided by Udacity, and Labelled Faces in the Wild (LFW) dataset which was created and managed by researchers at the University of Massachusetts. Both datasets can be downloaded here and here.

By analysis:

The dog dataset is made up of 8,351 images, 133 labels which is the different dog breeds. The dataset is already split into train, test and validation files. The train file contains 6650 images and all the 133 labels. The dataset is fairly balanced with about 50 images for each label while the test and validation files contain 836 and 835 dog images respectively.

In The lfw (labelled faces in the wild) dataset consists 13,233 faces of 5,749 people which were collected from the web, each face labelled with the name of the person in the image. 1680 people captured in the dataset has two or more distinct photos.

The dog images data will be used to train a model to classify dogs according to their breeds.

Samples of images contained in the datasets are shown below.

Fig. 1 Sample Images in the dog images dataset.

Fig. 2 Sample Images in the lfw dataset.

Algorithms and Techniques

The model for detecting faces in images is a haarcascade face detector. This is a pre-trained model which can be implemented using the opencv library. The algorithm detects a face by extracting features from an image and uses these features to classify an image as having a face or not. It was trained using both images with faces and those without faces. However, like any other imperfect model, it has a tendency of reporting false positives and false negatives which entails detecting faces where there are none and not detecting faces where there are faces.

For dog detection and breed classification, the models are both convolutional neural network classifiers, trained on datasets which are different but has similar contents, both can be used interchangeably with the predict functions. For the dog detection, I used a VGG-16 model. The VGG-16 convolutional neural networks model was one of the leading models from the ImageNet challenge in 2012. It was trained on over 14 million images (dog images included). The output of a pretrained VGG-16 model is 1000 classes. In these 1000 classes, (which is in the range 0 – 999 inclusive), from 151 – 268 represents dog classes.

For the dog breed classification, the model is also a CNN model. Its "predict" function outputs a class from the 133 different dog breeds which an input image might belong to. Like any other model, one sure problem is that for every image, the classifier will always return a breed even if the image has cow or a building in it! To account for this problem, prior to feeding the classifier an image, image detector models are used to detect the correct contents of the image, else we might discover that our car or our buildings actually belong to some dog breed!

There are some factors about the dog breeds which makes this task a bit challenging. Some dogs which are of the same breed come in different shades of colours and some different breeds are very close in resemblance, for example, the "Brittany" and "Welsh Springer Spanier", successfully classifying these breeds will require a lot of robustness in the model.

Fig. 3 "Welsh Springer Spanier"-*left* and "Brittany"-*right.*

Optimization

The face detector can not be tuned because the model contains only the save model artefacts. However, a better face detector model can be designed and used in place of the haarcascade model.

The dog detector which in this case is pretrained model contains parameters which can be finetuned. But that won't be necessary. The model is already robust. However, we can try other pretrained models which are available in pytorch such as Inception-v3, ResNet-series[1], AlexNet, etc.

The dog breed classifier which is built from scratch can be also be optimized. Possible suggestions to this optimization process include:

- Network architecture
  - Total number of layers in the network
  - Varying dropout probabilities between layers
  - Tweaking the parameters of the pooling, convolutional and dropout layers.
- Hyperparameters
  - Batch size

[1] Different ResNet architectures which include ResNet-18, 34, 50, 101, 152

- Epochs
- Learning rate
- Optimizer type

Steps taken to pre-process the data (see. Methodology)

Benchmark

- For training a classifier from scratch, a model accuracy of more than **10%** is to be achieved.
- Using transfer learning, a model accuracy of more than **60%** is to be achieved.

The values are clearly estimated, given the time and hardware available to train the classifiers.

Methodology

Data Pre-processing and Preparation

I have three models which accept images in different forms. The face detector model accepts images in grayscale format while the dog detector and dog breed classifier accept images in tensor format.

For the face detector, the image is simply converted into a grayscale. But for the other classifiers, a transformer is defined which takes in the pre-processes and augments the train data, and pre-processes the test and validation data. The image pre-processing in the transformation job follows this pattern:

1. Image is scaled.
2. Image is cropped to the desired input size accepted by the model.
3. Image is converted to tensor
4. Image is normalized.

For the train data, which is augmented during pre-processing, the images go through the following steps:

1. Image is flipped horizontally,
2. Image is rotated randomly at an angle of 45 degrees.
3. Image is scaled and resized to the desired input size accepted by the model.
4. Image is converted to tensor.
5. Image is normalized.

Tweaking the rotation angles and adding more augmentation processes may increase model accuracy and generalization, improving performance.

Implementation

Face Detection

The OpenCV library is used to implement the haarcascade pre-trained image detection model and passed some to the pre-processed face images into the classifier to detect. The face detector signifies that a face is detected in an image by drawing a box(es) on the area of where the face(s) is detected. Samples of images with faces as detected by the face detector is shown below.

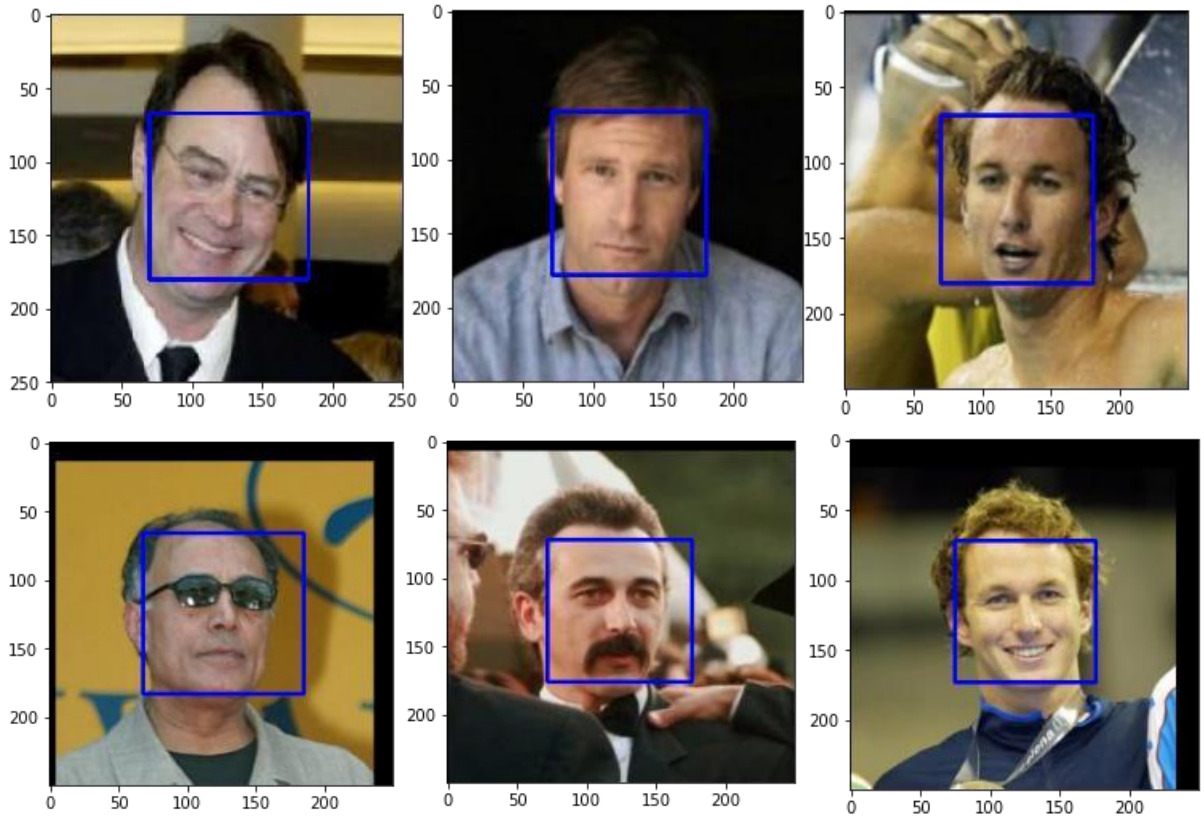

Fig. 4 Samples of Images with detected faces.

Dog Detection

The VGG net detects the presence of a dog in image by producing the classes that corresponds to dog classes in its original artefact.

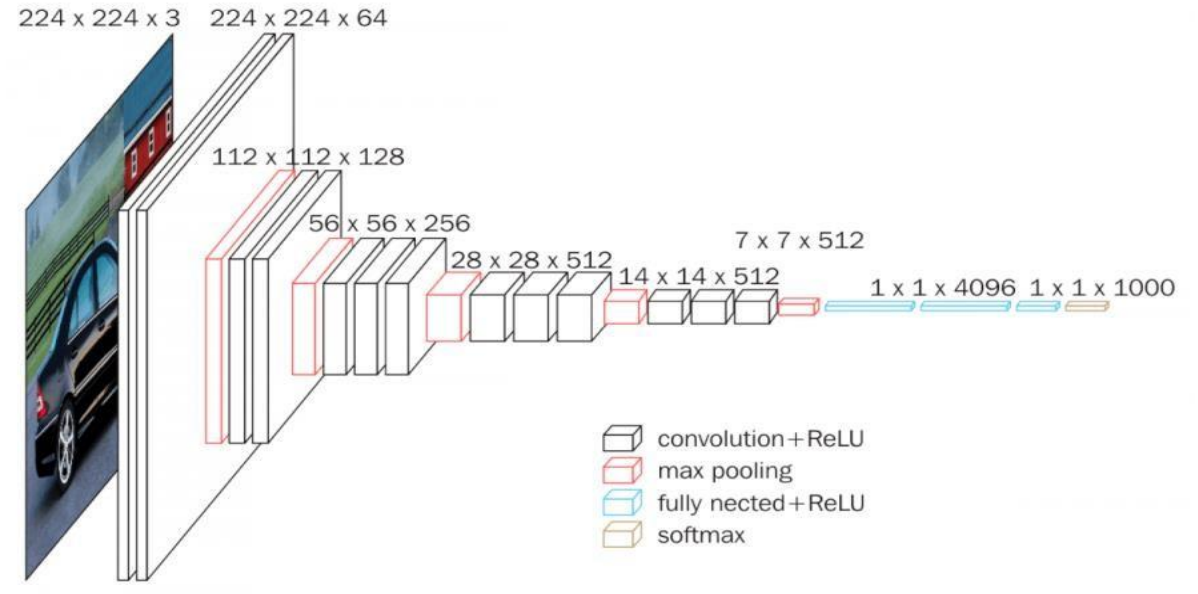

Fig. 4 The VGG-16 network architecture (source: popular networks)

Building the dog breed classifier

In step 3 of the jupyter notebook titled "Create a CNN to classify Dog Breeds (from scratch)"

I designed a network architecture which is made up of five **(5)** convolutional layers, max pooling layers with different strides and pool, four fully connected layers, batch normalizer for the each of the layers, and dropout layers with dropout probability of **0.25** and **0.15**. The steps taken to design and train the network are outlined below:

1. Create a transform job that augments only the training data
2. Create dataloaders with a specified batch size and number of subprocesses
3. Define the network architecture
4. Specify the optimizer and criterion
5. Train the model, saving the model with the lowest validation loss
6. Load saved model and test it on the test set.
7. If accuracy is not higher than the specified bench mark (see. Optimization)

The summary of the resulting model is shown below.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 224, 224]             448
       BatchNorm2d-2         [-1, 16, 224, 224]              32
         MaxPool2d-3         [-1, 16, 112, 112]               0
            Conv2d-4         [-1, 32, 112, 112]           4,640
       BatchNorm2d-5         [-1, 32, 112, 112]              64
         MaxPool2d-6           [-1, 32, 56, 56]               0
            Conv2d-7           [-1, 64, 56, 56]          18,496
       BatchNorm2d-8           [-1, 64, 56, 56]             128
         MaxPool2d-9           [-1, 64, 28, 28]               0
           Conv2d-10          [-1, 128, 28, 28]          73,856
      BatchNorm2d-11          [-1, 128, 28, 28]             256
        MaxPool2d-12          [-1, 128, 14, 14]               0
           Conv2d-13          [-1, 256, 14, 14]         295,168
      BatchNorm2d-14          [-1, 256, 14, 14]             512
        MaxPool2d-15            [-1, 256, 4, 4]               0
           Linear-16                 [-1, 2048]       8,390,656
      BatchNorm1d-17                 [-1, 2048]           4,096
          Dropout-18                 [-1, 2048]               0
           Linear-19                 [-1, 1024]       2,098,176
      BatchNorm1d-20                 [-1, 1024]           2,048
          Dropout-21                 [-1, 1024]               0
           Linear-22                  [-1, 500]         512,500
      BatchNorm1d-23                  [-1, 500]           1,000
          Dropout-24                  [-1, 500]               0
           Linear-25                  [-1, 133]          66,633
================================================================
Total params: 11,468,709
Trainable params: 11,468,709
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 26.72
Params size (MB): 43.75
Estimated Total Size (MB): 71.04
----------------------------------------------------------------
```

Transfer Learning

Training deep learning models generally takes a lot of time and computational power. It is easier and more efficient some times to use pretrained models which were trained to perform tasks similar to the one you have. So here I streamlined the number of outputs of the VGG-16

network to match the number of dog breeds in the dog images dataset which is 133, added dropout layers with probabilities of **0.1** and **0.15** between the first and second fully connected layers respectively, then applied LogSoftMax function to the output and trained the network.

Refinement

The CNN built from scratch, though not so great, performed really well with an accuracy of **45%** which is **35%** higher than the benchmark accuracy of **10%**. The initial result of the model was **11%**, but it improved with the following techniques:

- Applying batch normalization
- Varying dropout probabilities between layers
- Applying max pooling layers with different strides
- Training the model for longer epochs

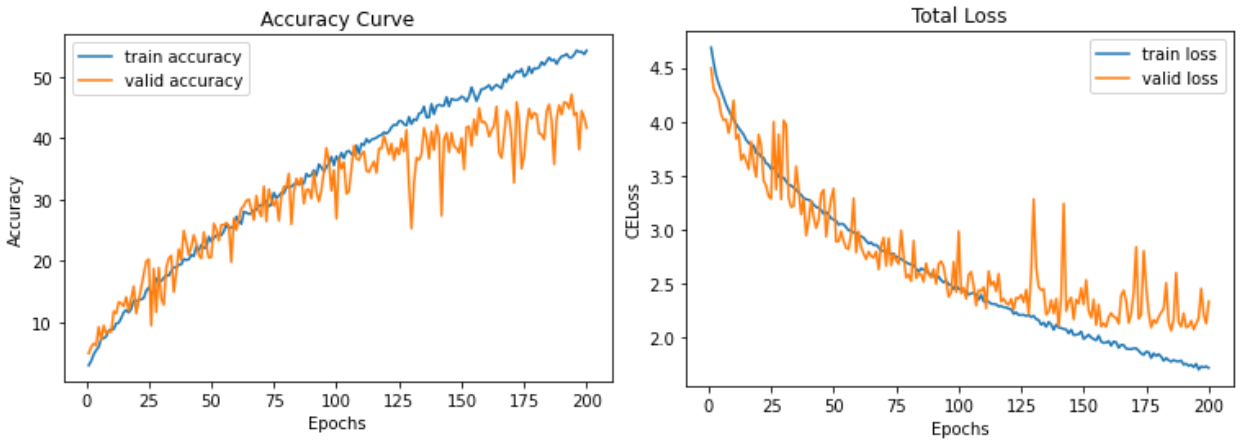

Fig. 5 Accuracy and Loss curve for CNN trained from scratch

The training processed showed steady increase and decrease in the accuracy and loss. The divergence in the curves indicates overfitting which might be as a result of fewer samples of the individual classes in the dataset, therefore, the more the model trains, the more it learns the data closely, the more it is unable to generalize. The model complexity and weight decay can also be factors.

The CNN trained using transfer learning (for steps on design, see. Transfer Learning) achieved an accuracy of **85%** on the test set. The validation curves for the model is shown below.

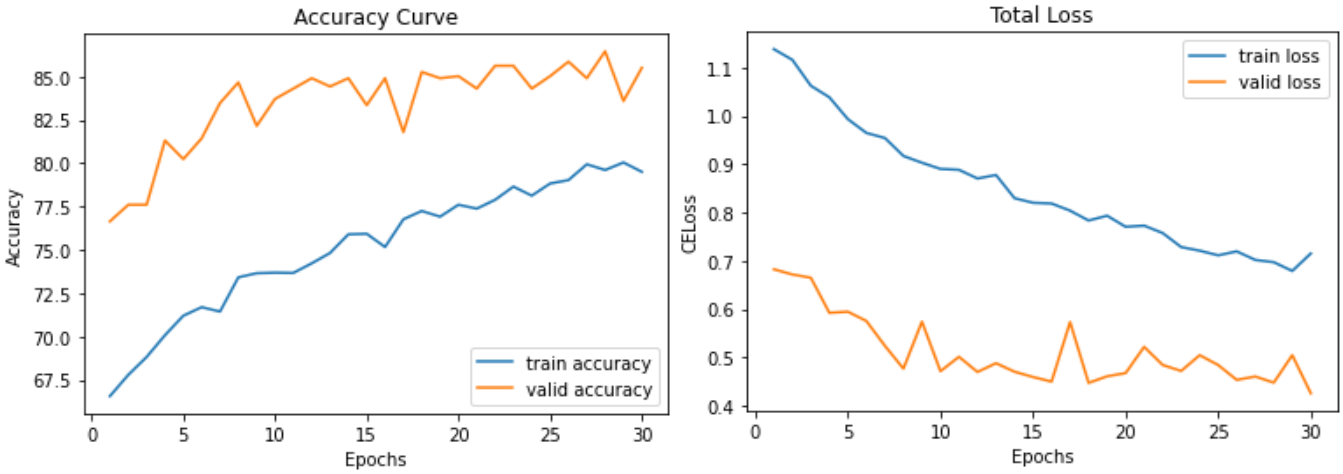

Fig 6. Accuracy and Loss curve for CNN trained using transfer learning.

Here, we can see that the accuracy and loss are way better than that of the model trained from scratch for 200 epochs. The explosive gap between the train and validation curve is not exactly as a result of overfitting or underfitting, and it does not look like a good fit either. One possible explanation might be due to the presence of the dropout layers, as the they turn off some of the network nodes during training but the full network is used during evaluation. However, as the number of epochs increases, the training curve and the validation curve converges, and overfitting sets in, but the validation accuracy decreases albeit the loss decreased.

Results

Model Evaluation and Validation

The face detection model performed well it is glory on most of the images. The clarity of the contents of the image posed somewhat of a problem to it, for example, when passed a dog image with human faces in it, it detected the human faces and classified the image wrongly. For some reasons, it could not detect human faces in perfect human images and also due to its love for faces,

it detected some on dogs! But this is not much of our problem because we are aiming at classifying dog breeds and not face detection.

The dog detector was almost perfect, it detected dogs in approximately 98% of images with dogs passed to it.

For the dog breed classifiers, during training, the models were evaluated using the holdout/validation set. The final parameters of the model were chosen because the models performed better with them.

Justification

Feeding the a few images to the final app which included images of dogs, humans and objects, the models were able to detect the contents of the image and the classifier got most of the dog breeds correct. The lapses mostly are in the face detectors, some images with human faces had no faces detected.

For images with unique contents, the object detectors perform well, but for images with a lot in the background, they don't do much.

More images should be added to each class in the dogImages dataset and if possible, more classes should be added.

Additional Features

The primary problem here is to detect objects in images and classify dog breeds, however, I have added additional functionality and feature to make the app more fun and precise.

Dog Mutts Functionality

A Mutt or Mongrel[2] dog is a dog that belongs to more than one breed. Normally, the breed classification model returns a probability for each of the 133 classes and the predict function returns the class with the highest probability. It is alright to say that a dog is of a particular pure breed if the class returned by the predict function has a probability higher than **0.65%**[3], else, the dog in the input image might be a mongrel. In this project, I added

[2] Also Mixed-Breed dogs, different from designer dogs or cross-breeds which are intentionally bred.
[3] The rest of the values are distributed among the remaining 122 classes.

functionality for dogs which might have two or three different breeds as their ancestors. To do this, thresholds of **0.65** and **0.35**[4] are defined for the two breed mutts and thresholds of **0.45** and **0.15** are defined for the three breed mutts. If the highest-class probability returned is ***lower*** than the first threshold of **0.65,** and the second highest-class probability returned is ***greater*** than the second threshold of **0.35,** the dog in the image is considered a mutt of the two breeds. Similarly, for the three breed mutts, if the highest-class probability returned is ***lower*** than the first threshold of **0.45,** and the second and third highest-class probability returned are both ***greater*** than the second threshold of **0.15,** the dog is considered a mutt of the three classes.

Filter Overlay

To make the app more fun, for every image with a detected face, I added a function that overlays dog ears and nose filters over the detected face, just like in Snapchat and similar apps.

Web App Deployment

Using the python flask framework, I created an app which takes in an image, classifies the object(s) in the image as dog(s), human(s) or neither and if the image has dog or human detected, the dog which the human or dog looks like is displayed. Also, if the detected object(s) in the image is human, dog ears and nose filters are applied to the human face(s) in the image.

I faced a huge setback trying to deploy the app to the cloud as the cloud services provide only a limited storage for free accounts. However, with Heroku, there is an option to upload the huge files to amazon s3 bucket, but there might be charges, so the final web app was not deployed to the cloud.

To test out the app on your local computer, follow these steps:

1. Navigate to this github repository and fork it.
2. Open your command prompt. If you have anaconda installed, use the anaconda command prompt. (The following commands are for windows, if you are on a Mac/Linux, check equivalent commands online)
3. On your local computer, navigate to the folder where the repo is stored, open the requirements folder and copy either the .yml file or

[4] Considering the remaining probability values which are distributed among the remaining classes.

the .txt file and paste in the root folder, then set up a virtual environment with the file using the following commands:

```
# Using Conda,
$conda create env -f requirements_conda.yml

# Using pip
$py -m venv env
$pip install -r requirements.txt
```

4. Activate the virtual environment using either of the following commands:

```
# Using Conda,
$conda activate env

# Using pip
$.\env\Scripts\activate
# if the above throws an error, try:
$\\env\Scripts\activate
```

5. Run the following commands below to deploy the app on your local runtime.

```
$set FLASK_APP=dog_app.py
$set FLASK_ENV=development
$flask run
```

The output should be similar as the one shown in the image below.

```
(flask_env) C:\Users\NGabriel\Desktop\Capstone_Dog_App>flask run
 * Serving Flask app "dog_app.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 232-891-288
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

6. Copy and paste the link on your browser.

The web app is made up of two web pages, one to get the file, and the other to display the results.

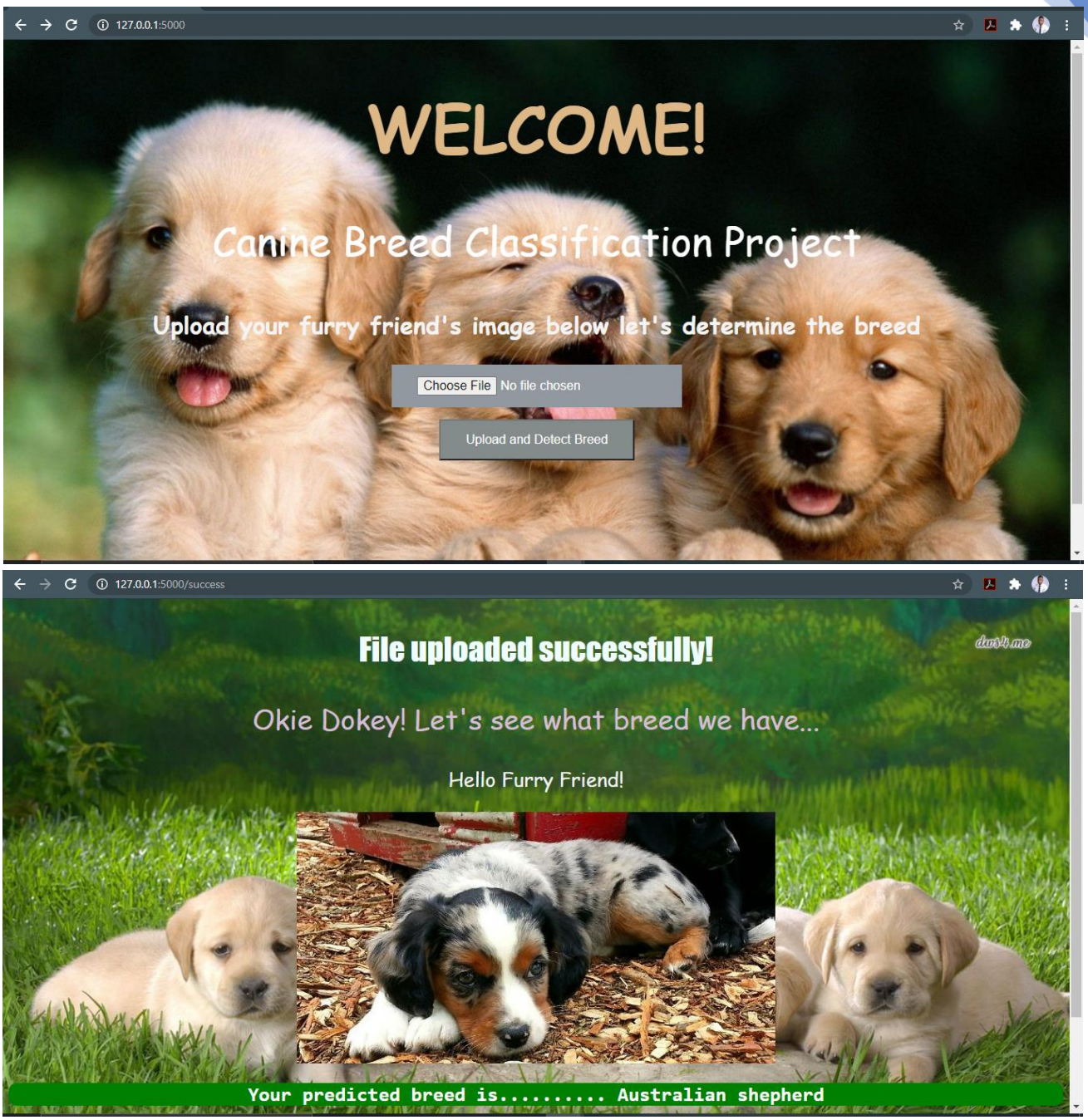

Fig 7. The pages of the web app

Conclusion

Free Form Visualization

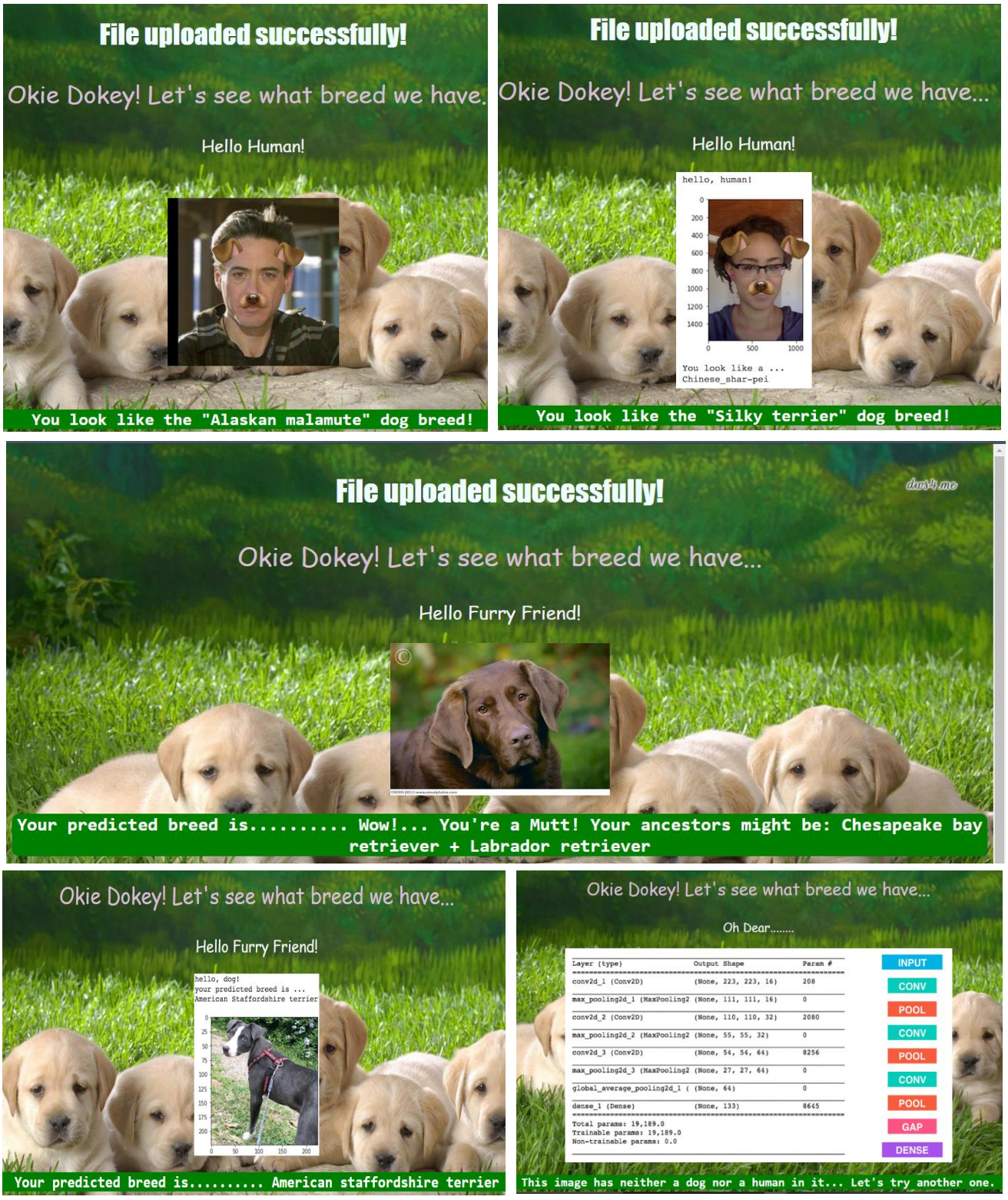

Fig 8. Results produced by the web app for different images

Reflection

The end to end development process for this project can be summarized into the following steps:

1. An initially defined problem with links to public datasets
2. Datasets were downloaded and object detectors were built
3. Relevant datasets were preprocessed and augmented
4. With the given benchmark, the model classifier was trained, multiple times until the best model is achieved
5. A predict function was defined with dog mutt functionality added, to translate the outputs from the classifier into human readable texts
6. For overlaying filters, a facial keypoints detection model was built and trained
7. Using the flask framework, the web app was built.

Every step taken to complete this project was very much as educative as the project. The time I spent with pytorch during project was worth it. Getting stuck in each step of the project led me to have read a lot of articles (both related and non-related to the project) on deep learning, and I am even more eager to explore the field and contribute to my society with what I have learnt.

Possible Improvements

As already discussed, the app was not deployed to the cloud so it was not possible to measure the actual time it takes for user to get their outputs after uploading their files. On the local runtime it was a bit slower, as it took up to two (2) seconds before the success page is displayed.

The front end of the web app could really use a lot of work to improve the interface and increase user experience.

With in-depth knowledge of mutts[5], better thresholds to determine if the dog(s) present in the image might be mutt or not will be set, thereby, classifying the breeds better.

[5] See Dog Mutts Functionality