Udacity Machine Learning Engineer Nanodegree

Capstone Project Report

Convolutional Neural Networks Application in Dog Identification App

Jingyi Han

July 5th, 2021

Table of Content

1. Definition

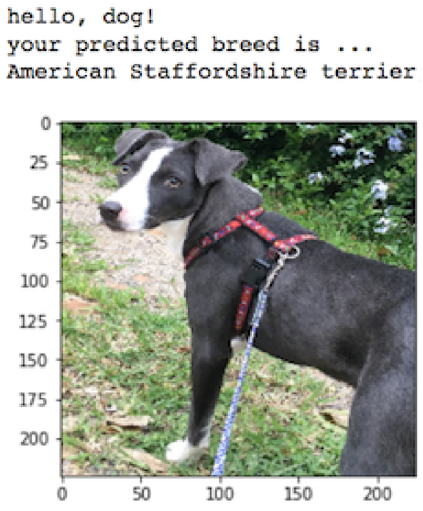
1.1 Project Overview

Image recognition, as a subset of computer vision, allows machines to gain an understanding of digital images using artificial intelligence algorithms. The improvement in the image recognition field enables us to automatic relevant daily tasks, enhance efficiency and bring fun into our daily life. This capstone project will investigate dog breed identification and build an algorithm to produce an estimation of a dog breed given a dog image or dog breed that best ensembles the provided human image.

1.2 Problem Statement

This project aims to develop a dog breed identification algorithm that could be used as part of a mobile or web app. Taken user-supplied images as input, the program would provide the estimation of the dog's breed. When the algorithm detects a dog in the supplied image, it will estimate the dog breed. However, if a human is detected, it will provide an estimate of the dog breed that is most resembling. The image below displays the potential sample output of the finished project.

Sample output [1]



```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```

1.3 Metrics

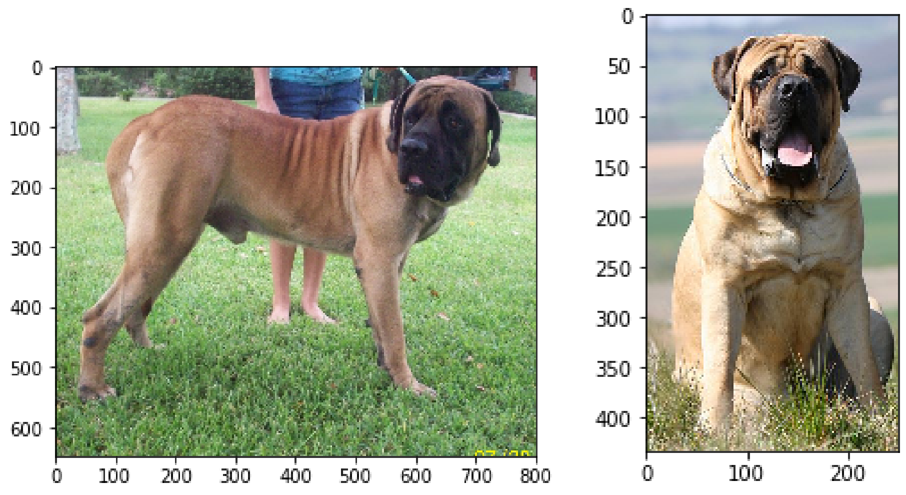
Test accuracy will be calculated as an evaluation metric of the model performance. As suggested by the Udacity project notebook [1], certain test accuracies must be reached for each implementation. For the CNN algorithm created from scratch, test accuracy of at least 10% must be attained. For the CCN algorithm created using transfer learning, at least 60% accuracy on the test set must be attained.

2. Analysis

2.1 Data Exploration

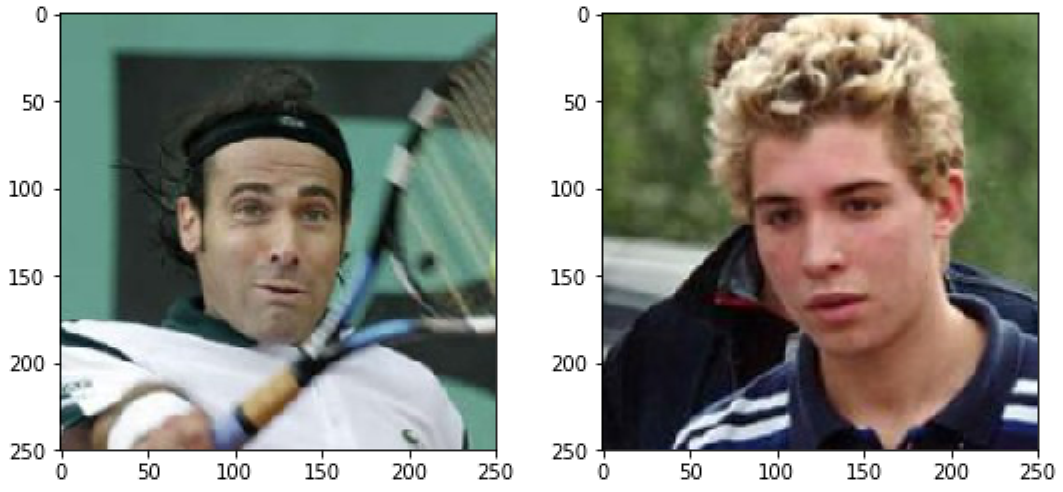
Dog images are downloaded for dog breed identification. 133 distinct breeds will be included and separated into a training set, validation set, and testing set. A total of 6,680 images are included for model training, 835 images are included for validation, and 836 images are included for testing. For training, Alaskan Malamute has the highest number of images which contains 77 images, and Norwegian Buhund has the lowest number of images which contains 26 images. The following pictures show some example dog images used for model training, validating and testing.

Mastiff data [7]

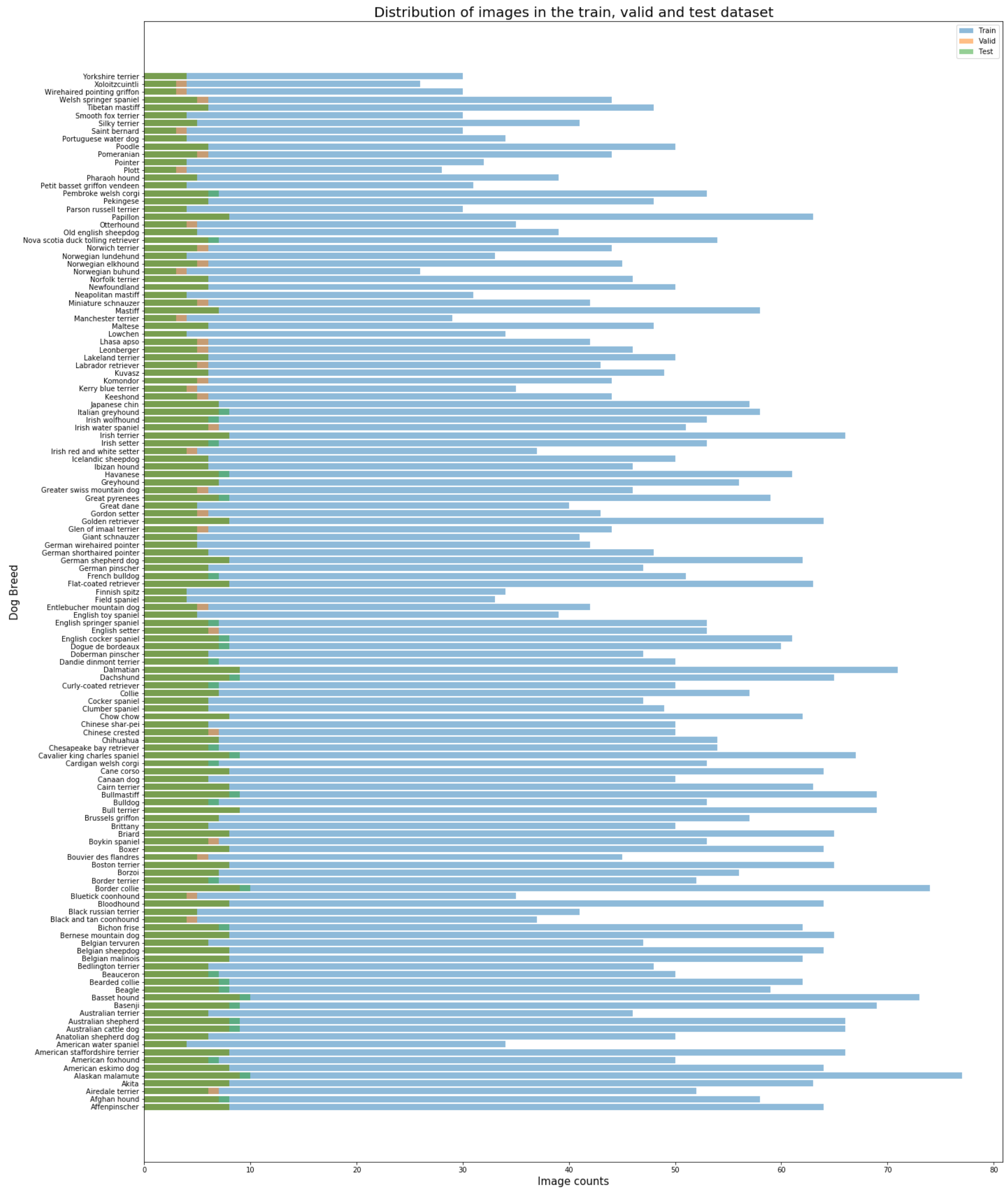


Human images are included as additional testing data set. 13,233 total images of humans will be downloaded for 5,749 distinct people. Out of this dataset, George W. Bush has the highest number of images, which contains 530 images in the folder. The rest of the people have as least 1 image stored in the folder. The following pictures show some exampling human images used for model training, validating and testing.

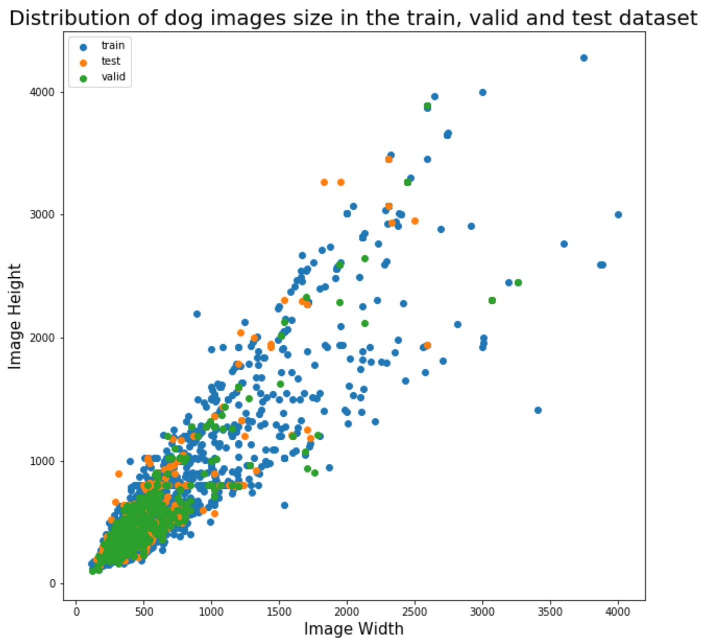
Human face data [7]



The number of pictures included for each dog breed is shown below. It can be observed that training data have slight data imbalance given each breed of dog has a different number of figures provided. Alaskan malamute has the most number of training images (77 images included). Norwegian buhund has the least number of training images (26 images included). Since each breed has at least 26 images, no additional images were added to the training set.



Lastly, the image sizes and color were analyzed. All images for both bog and human have RGB as 3. For human images, all images have a width being 250 pixels and a height being 250 pixels. For dog images, the minimum width is 113 pixels in the training set and the minimum height is 105 pixels. The overall distribution for dog images is shown in the figure below. This information was useful in determining the scaled image size during data preprocessing.



2.2 Algorithms and Techniques

Firstly, two pre-implemented detection models were used to identify humans and dogs in user supply images. Specifically, OpenCV's implementation of Haar feature-based cascade classifiers [2] was used to detect human faces in images, and the pre-trained VGG-16 Model [3] was used to detect dogs in images. Secondly, algorithms for dog breed classification were implemented. Convolutional Neural Networks (CNN) [4] has been a well-studied deep learning algorithm in the domain of computer vision that enabled machines to conduct tasks such as image recognition and natural language processing. Hence, a CNN algorithm was selected for the purpose of dog breed identification using user-provided images and was implemented in two ways, from scratch [5] and using transfer learning [6] with PyTorch, as solutions to the dog breed identification problem. A variety of image classification models were tested, including resnet18, restnet34, resnet50, and resnet101. Detailed implementation of the aforementioned models and techniques will be discussed in Section 3.2.

2.3 Benchmark

A random guess would provide a correct answer approximately 1 in 133 times, which corresponds to an accuracy of less than 1%. Hence, the CNN algorithms should have an accuracy higher than the random guess. In addition, the CNN algorithm created from scratch would be used as the benchmark model. CNN algorithm implemented using transform function should improve the model performance.

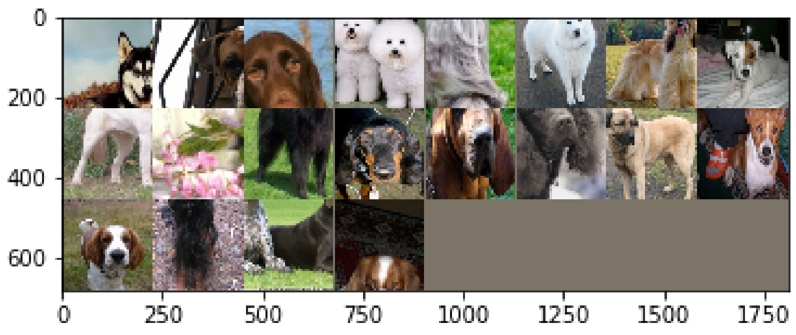
3. Methodology

3.1 Data Preprocessing

For face detection using Haar feature-based cascade classifier pre-trained model [2], images were first converted to grey scaled images. For dog detection using VGG-16 Model [3], the images were resized to 256*256 pixels, then center cropped to 224*224 pixels for data augmentation. The images were then transformed to tensor type and normalized.

For dog classification models from scratch and using resnet models, training data were resized to 256*256 pixels, then randomly cropped to 224*224 pixels for data augmentation and random horizontal flipped. The images were then transformed to tensor type and normalized. Testing and validation data were resized to 256*256 pixels, then center cropped to 224*224 pixels for data augmentation. Training, testing, and validation data were separated into batches with 20 samples and a number of subprocesses to use for data loading as 1 to speed up the training process. The image below shows an exampling of the batch input.

Training Input



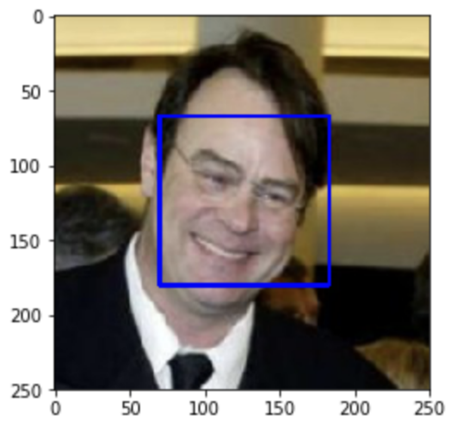
3.2 Implementation

For face detection, Haar feature-based cascade classifiers [2] from openCV were used. The model was stored as CML files on github location [8]. The model was downloaded with the following script:

```
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')
```

Then, preprocessed data was provided to the model. The model output would produce the number of faces detected in the image. An example output of the face detection is shown below:

```
Number of faces detected: 1
```



For dog detection, VGG-16 [2] from PyTorch pre-trained was imported from torchvision.models package. The model has been trained on ImageNet, which contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories [1]. Categories between 151 and 268, inclusive, are designated to dog breed. Hence, the model output will be an integer between 1 to 1000, and if the integer falls between 151 to 268, it is considered that a dog was detected from the image.

For the dog classifier from scratch, a simplified resnet18 model was constructed. As a resnet18 model will be tested later in the project, the simplified model would be compared with the original model. This simplification allowed less convolution layers and less computation. The model is implemented as the following:

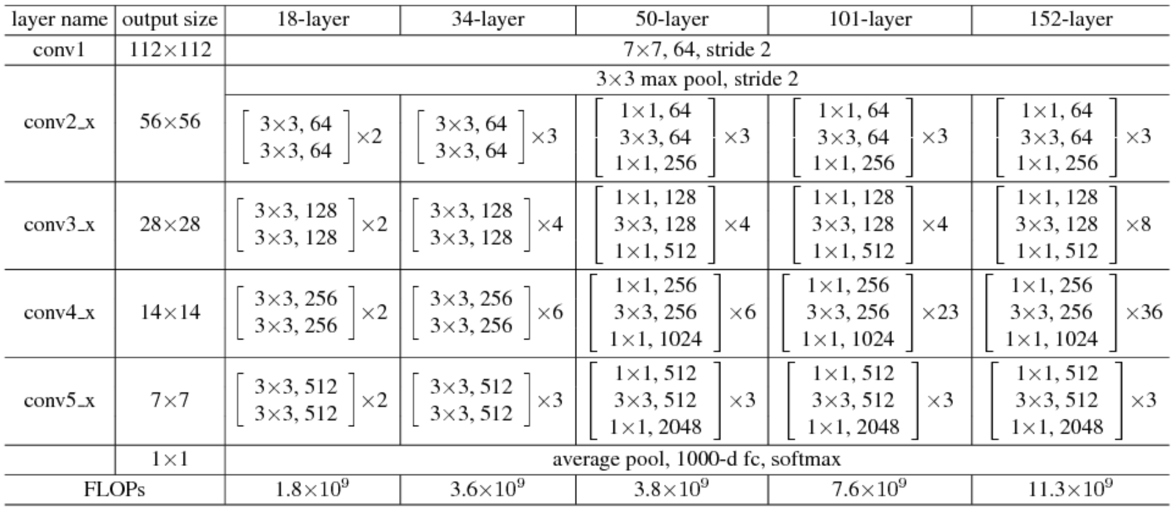
- Convolutional Layer with 3 input dimensions, 64 output dimensions, kernel size as 3, stride as 2 and padding as 1
- Relu Activation function
- Polling layer, kernel size as 2, stride as 2 and padding as 0
- Convolutional Layer with 64 input dimensions, 64 output dimensions, kernel size as 3, stride as 1 and padding as 1
- Relu Activation function
- Polling layer, kernel size as 2, stride as 2 and padding as 0
- Convolutional Layer with 64 input dimensions, 64 output dimensions, kernel size as 3, stride as 2 and padding as 1
- Relu Activation function
- Polling layer, kernel size as 2, stride as 2 and padding as 0
- Convolutional Layer with 64 input dimensions, 128 output dimensions, kernel size as 3, stride as 2 and padding as 1
- Relu Activation function
- Polling layer, kernel size as 2, stride as 2 and padding as 0
- Convolutional Layer with 128 input dimensions, 128 output dimensions, kernel size as 3, stride as 2 and padding as 1
- Relu Activation function
- Polling layer, kernel size as 2, stride as 2 and padding as 0

- Convolutional Layer with 128 input dimensions, 256 output dimensions, kernel size as 3, stride as 2 and padding as 1
- Relu Activation function
- Batch Normalization with 256 input dimensions
- Polling layer, kernel size as 2, stride as 2 and padding as 0

For model training, the loss function was specified as cross-entropy loss and stochastic gradient descent with learning rate as 0.01 was used for the optimizer.

For transfer learning implementation of the dog classifier, resnet101 model was used. Resnet18, restnet34, restnet50, and restnet101 were compared respectively, and it was observed that restnet101 provided predictions with the highest accuracy. Testing results will be discussed in detail in section 3.3 and section 4.2. The implementation of these models is shown below.

Table 3: resnet model implementation [9]



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

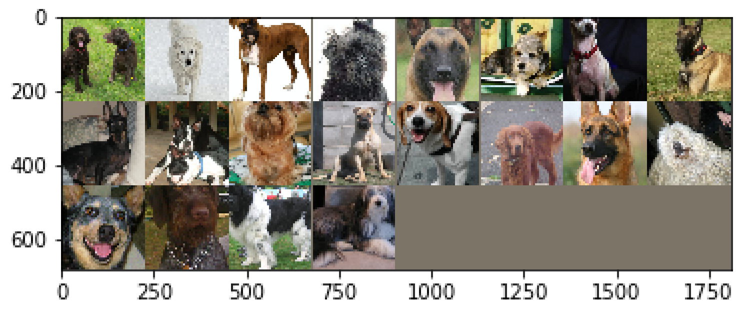
For model training, the loss function was specified as cross-entropy loss and stochastic gradient descent with a learning rate of 0.001 and momentum as 0.9 was used for the optimizer.

Lastly, an algorithm was written to combine the above features together. It accepts a file path to an image and first determines whether the image contains a human or a dog and returns the predicted breed through combining the Human detection model, dog detection model, and the CNN classification model. The algorithm would provide an error indicator when the provided image does not contain a human or bog.

3.3 Refinement

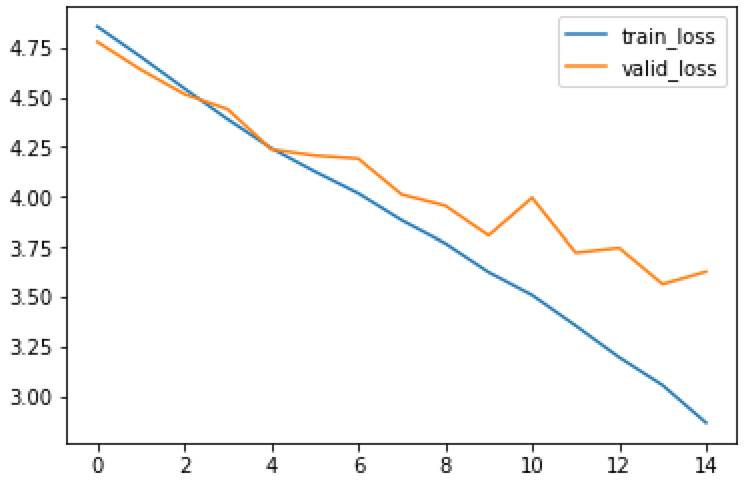
A few refinements were implemented and testing during the model development process. Firstly, different data preprocessing steps were taken for training data during the development of the model from scratch. The data preprocessing steps were set the same as testing and validation, where data were resized to 256*256 pixels, then center cropped to 224*224 pixels for data augmentation. An example of the data input is shown below.

Training Input - before the final version



With the implementation, overfitting issues were observed and only an accuracy of 16% was achieved.

Model from scratch - before final version Accuracy: 16%



Secondly, for transfer learning implementation of the dog classifier, resnet18, resnet34, resnet50, and resnet101 were tested. According to PyTorch documentation, a model with layers would produce higher prediction accuracy. The four models are still investigated to see how much accuracy improvement and time consumption for the dataset used in this project. Respectively, resnet18 produced 85% accuracy, resnet34 produced 88% accuracy, restnet50 produced 88% accuracy and resnet101 produced 90% accuracy. It was also observed that more time was taken during training as the number of layers in the model increased.

4. Results

4.1 Model Evaluation and Validation

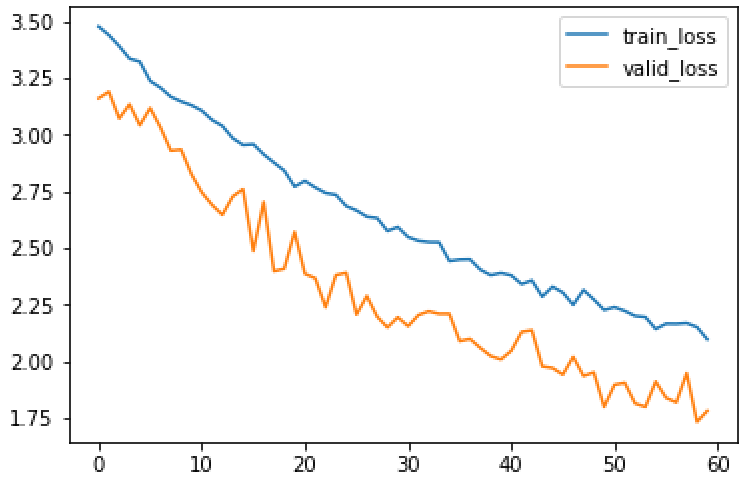
For face detection, to verify the accuracy of the face detector, the first 100 images in human images and 100 images in dog images were tested respectively. 98% of the human image was identified successfully and 83% of dog faces were identified correctly as non-human.

For dog detection, VGG-16 [2] from pytorch pre-trained was used. To verify the accuracy of the dog detector, the first 100 images in human images and 100 images in dog images were tested respectively.

99% of the human image was identified successfully as non-dog and 100% of dog faces were identified correctly.

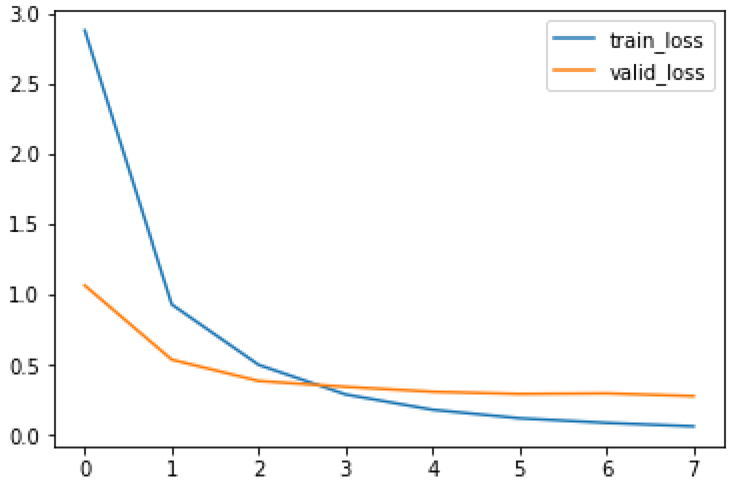
For the dog classifier implemented from scratch, a simplified resnet18 model was constructed. This simplification allowed fewer convolution layers and less computation. 60 iterations were used and the plot below shows that train loss was mostly above validation loss, which means the model did not overfit. However, only 54% accuracy was achieved. Since the accuracy is above 10% as specified metrics for the model implemented from scratch, no further investigation was performed.

Model from scratch Accuracy: 54%

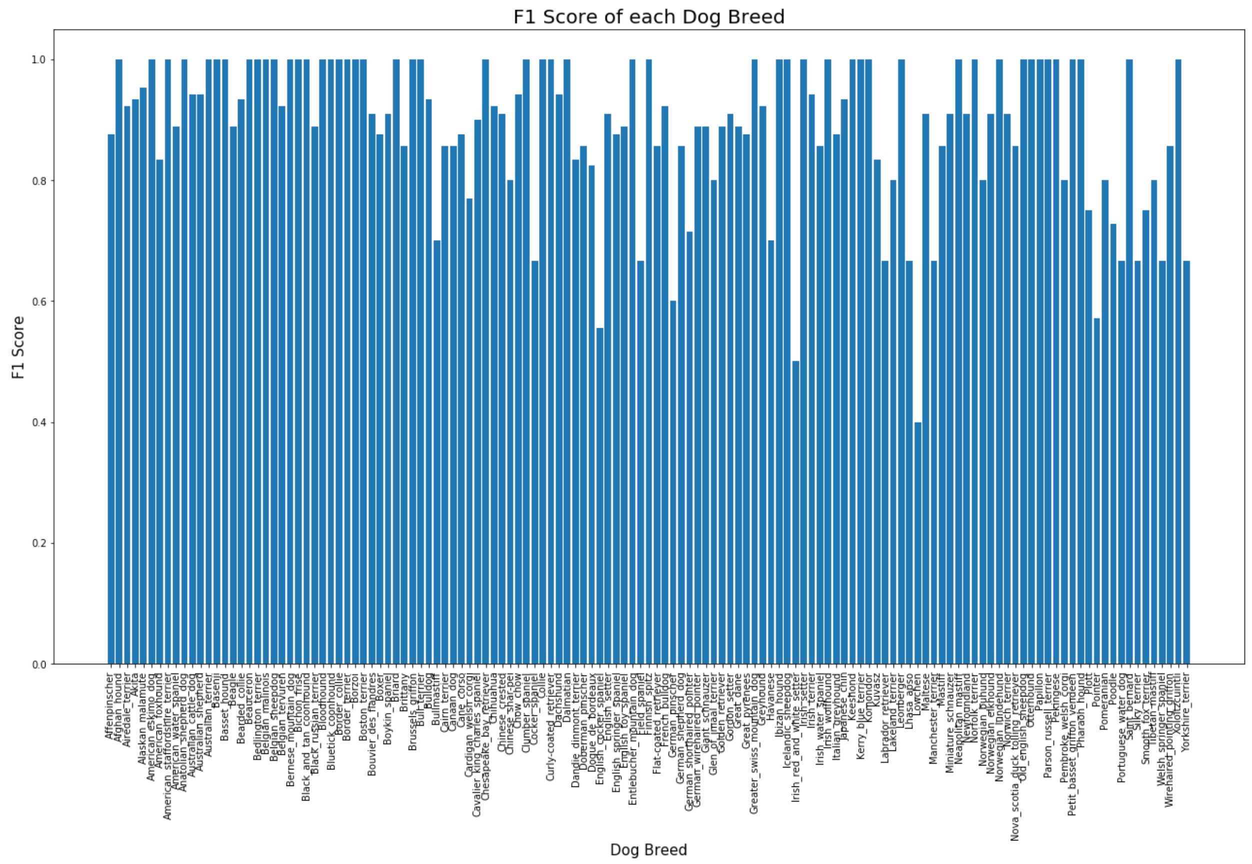


For dog classification using resnet model, resnet101 was selected since it produced higher accuracy compare to models with fewer layers. Hence, it should be noted that the training time for the model was also longer. 7 iterations were sufficient to achieve high accuracy. From the training loss and validation loss plot below, it can be observed that the model overfitted for the last few iterations.

Resnet101 Accuracy: 90%

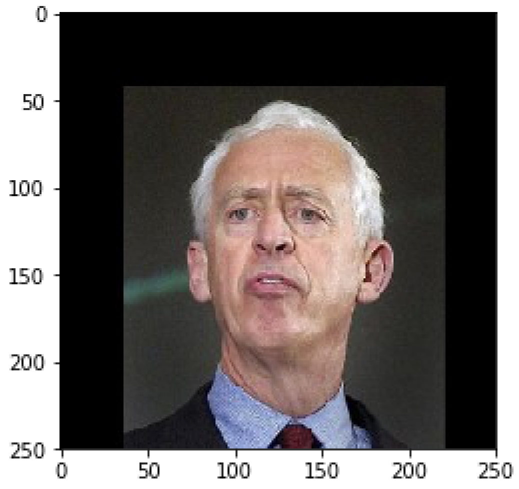


The F1 score for each dog breed is shown below. It can be noticed that the majority of the breed have an F1 score above 0.6. Lowchen has the lowest F1 score which is around 0.4. From data exploration, it can be observed that Lowchen has fewer training images. Also, Lowchen is very similar to many long hair breeds such as Maltese, and it can have many color variations; hence, increased the difficulty of the prediction.



For the final app algorithm, the top three mostly alike breeds are produced with corresponding probability. The following shows a testing output for a human image.

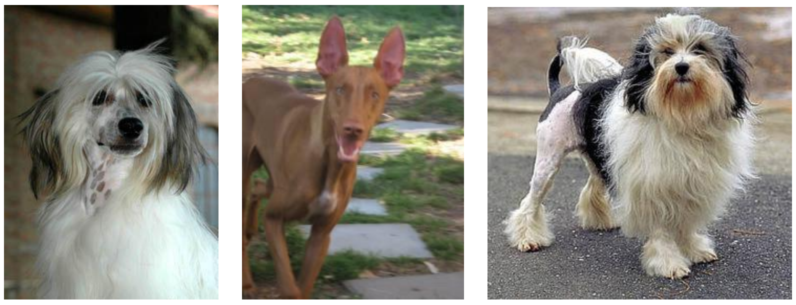
Hello, human!



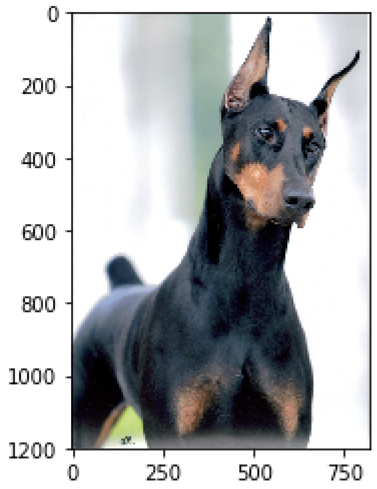
```
You look like a ...
Top 1 breed: Chinese_crested, class probability: 0.04227
Top 2 breed: Pharaoh_hound, class probability: 0.03328
Top 3 breed: Lowchen, class probability: 0.02044
```

As shown above, the top 3 breeds were Chinese crested, Pharaoh hound, and Lowchen. To verify the results, one figure for each breed is displayed below. The prediction seems to be reasonable as Chinese crested and Lowchen resembles his hair color and mouth. Pharaoh hound resembles ear feature and oblong face feature.

Chinese crested (left), Pharaoh hound (middle), Lowchen (right)



Below shows an exampling output for a dog image with the top 3 breeds with the highest probability. Knowing the model achieved 90% accuracy for breed prediction, the output for dog breed classification would be mostly correct. This top 3 feature is used for the 10% of times when a breed was falsely identified, the app would still provide some additional information for consideration.

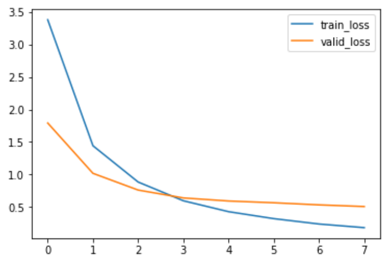


Top 1 breed: Doberman_pinscher, class probability: 0.98857
Top 2 breed: German_pinscher, class probability: 0.00815
Top 3 breed: Great_dane, class probability: 0.00163

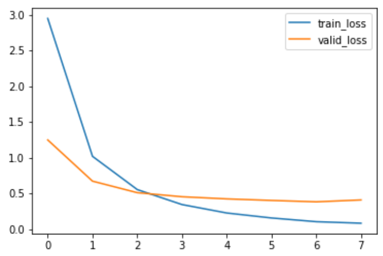
4.2 Justification

Overall, the final delivered app algorithm has excessed the accuracy criteria of 60% and the model implemented from scratch has excessed the accuracy criteria of 10%. In addition, the accuracy of the model implemented from scratch also excessed the benchmark, random guess, and the final transfer learn model exceed the accuracy of the model implemented from scratch. To justify that the final model selection was the optimal selection. The training results for resnet18, restnet34, and restnet50 are shown below. All of the models used 7 iterations and all show some degree of overfitting. Less iteration could potentially be used. Since resnet101 shows the highest accuracy out of the four models, it was selected to be the model used for the final app algorithm.

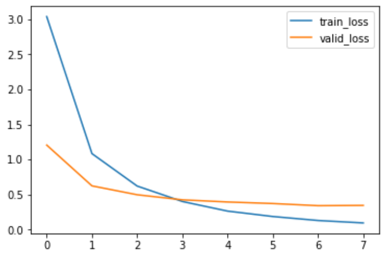
Resnet18 Accuracy: 85%



Resnet34 Accuracy: 88%



Resnet50 Accuracy: 88%



5. Future Improvements

Firstly, I would like to learn more about the process of building an image classification model from scratch and improve the prediction accuracy. Secondly, I would like to include more breeds in the classification training data. Lastly, notice the course also provide some instruction on building an interactive website using Flask, I would like to build a web app for this project and provide a user interface.

6. Reference

[1]https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/dog_app.ipynb
[2]https://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html
[3]https://pytorch.org/vision/stable/models.html
[4]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[5]https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
[6]https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
[7]https://github.com/udacity/deep-learning-v2-pytorch.git
[8]https://github.com/opencv/opencv/tree/master/data/haarcascades
[9]https://pytorch.org/hub/pytorch_vision_resnet/