Capstone Project Report: Machine Learning Engineer Nanodegree

Dog Breed Classifier using CNN

Srijan Sahay
October 24th, 2020

Domain Background

Image classification is one of the fundamental and the trending topic in the field of Machine Learning and Deep Learning. This also include real world image processing and computer vision problems.

Today, researchers are trying to classify different objects in an image. Classifying of animal and its breed is also one of them and is highly appreciated among animal lovers.

In this project, we will be using Convolutional Neural Network (CNN) to predict the breed of the dog in the given image. Also, if a human image is given, we will be predicting the label that closely resembles dog breed.

Problem Statement

The primary goal of this project is to do the following tasks using CNN:

i. Given an image of the dog, the model should predict its breed
ii. If human is detected in image, the model should predict resembling dog breed

To implement the solution, we will create 3 different model.

The first model will classify human face present in the image using the Haar Cascade algorithm.

The second model will be a CNN custom model to classify dog in the image. This model is created from scratch defining the number of convolution layer, pooling, and batch normalization within it.

The third model will be based on transfer learning using Resnet50 algorithm to improve over model and compare it with the earlier created models. Resnet50 is a convolutional neural network that is 50 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

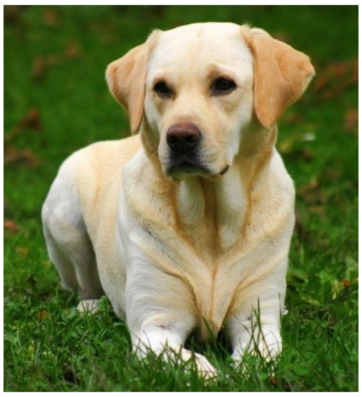
The main aim of this project is to create a robust classifier that can successfully determine breed of the dog image. We will be creating a pipeline where given an image of a dog, the algorithm will identify the breed of the dog. If there exists a human in the image, the code will return the resembling dog breed.

Datasets and Inputs

For this project, the dataset is provided by Udacity. The dataset contains the image of dogs and humans.

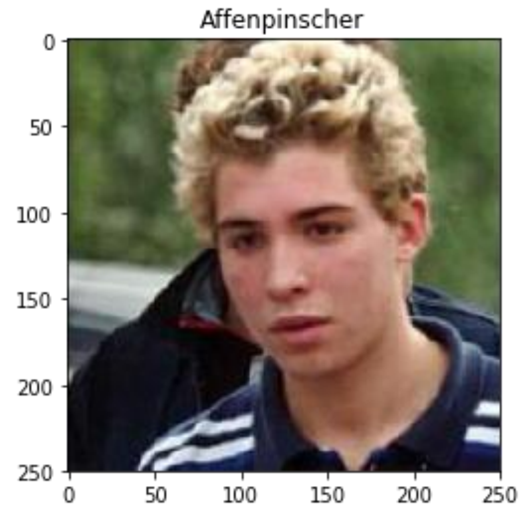
Dog Dataset: The dog dataset has 8351 images and is saved in the location /dog_images. The dimension of each image is not same. We have 133 types of dog breed and the dataset is imbalanced.

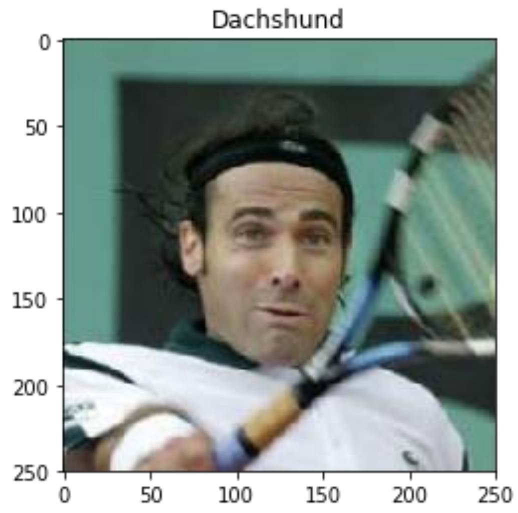
The dog image with different dimensions.



Human Dataset: The human dataset has 13233 images and is saved in the location /lfw. The dimension of each image is same i.e. 250X250

Human image has same dimension.





Metrics

Accuracy score is the main metric used for the validating the performance of the dog breed classifier. Below is the formula to find accuracy:

Accuracy = **Number of items correctly classified/ All classified items**

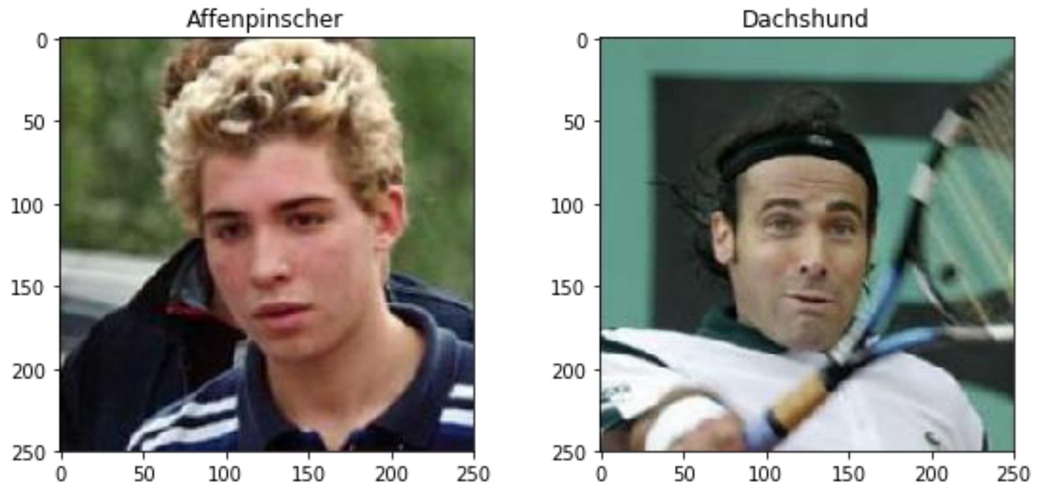
Also, accuracy score will be used to determine the performance of human face detector and dog detector.

Data Exploration & Visualization

There are 13233 total human images, which can help us train a model to identify the human image. And there are 8351 total dog images for dog classification.

For human face images, they all have the same dimension: 250×250. Images have different background and different angles. The data is not balanced because we have 1 image for some people and many images for some.

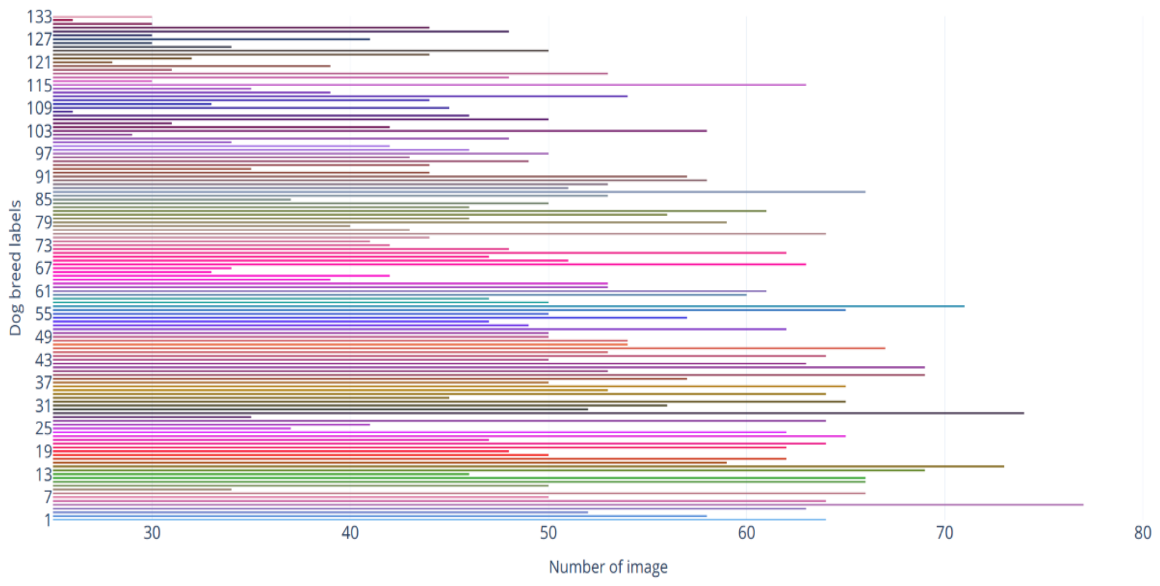
Human image has same dimension.



For dog images, the images are of different sizes and different backgrounds, some images are not full-sized. The data is not balanced because the number of images provided for each breed varies. Few have 4 images while some have 8 images.

The dog image with different dimensions.

Below image gives us a clear visualization of the number dogs in each bread.



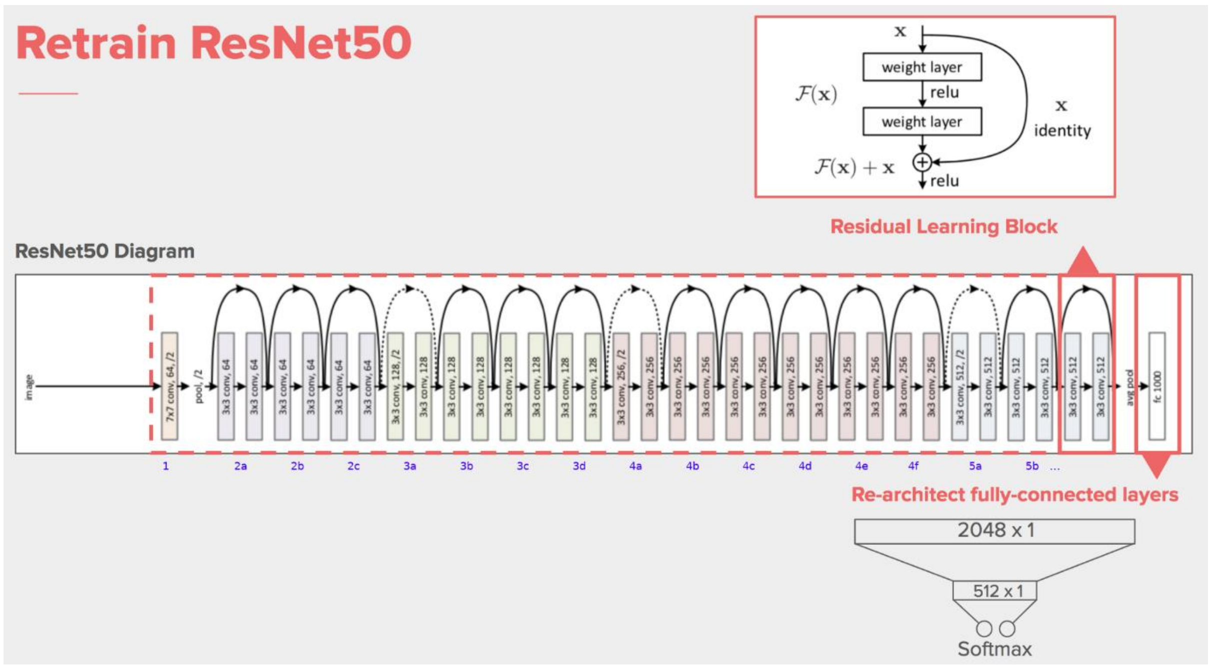
Algorithms and Techniques

Convolutional Neural Network can be used to solve the problem of multiclass image classification. A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. For solving this multiclass problem, Convolutional Neural Network (CNN) is used which is a state-of-the-art algorithm for image classification problem.

We have used three different models to solve this problem.

1. The first model will classify human face present in the image using OpenCV which uses the Haar Cascade algorithm.
2. The second model will be a CNN custom model to classify dog in the image. This model is created from scratch defining the number of convolution layer, pooling, and batch normalization within it.
3. The third model will be based on transfer learning using Resnet50 algorithm to improve over model and compare it with the earlier created models. Resnet50 is a convolutional neural network that is 50 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.
   As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224 X 224.

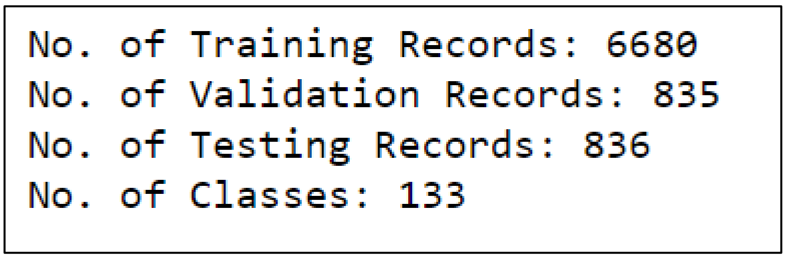
Below is the image of a ResNet-50 architecture:



For creating any machine learning or deep learning-based model, the foremost step is to pre-process the dataset to improve the results of training and create a robust feature sets than can accurately solve a given problem. Following are the steps that are involved in data pre-processing:

1. Resize the images in the dataset to a fixed size of 224*224 pixels.
2. Perform augmentation like flips and rotations of the images to include more variation in the dataset.
3. Normalize the pixel values of the images.
4. Convert the images into appropriate tensor the feed it in the model for training.

The dataset was split in the following ratios:



```
No. of Training Records: 6680
No. of Validation Records: 835
No. of Testing Records: 836
No. of Classes: 133
```

Implementation

We have divided this project in several steps like first performing human face detection and dog detection to extract human and dog region of interest (ROI).

Detect human image using OpenCV which in turn uses a Haar feature based cascade classifiers.

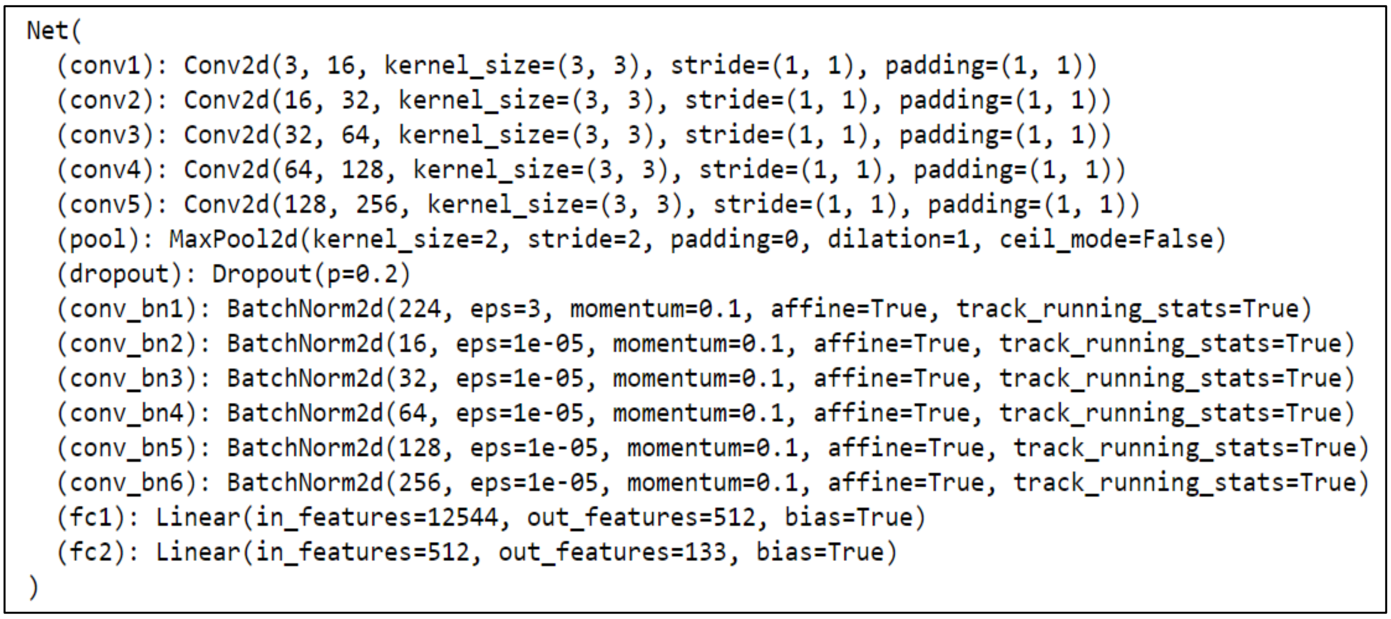
Create a dog detector using a pretrained VGG-16 model. VGG-16 is one the CNN architecture which was trained initially on ImageNet dataset.

To test the performance of the VGG-16 for the dog detection, we tested the first 100 human images and the first 100 dog images and we found that 0.02% of the human images have dog and 100% of dog images where detected for dog.

We trained a custom model to predict breed of the dog given an input image. For this we use CNN model.

Finally, we train a CNN model using transfer and compare the results.

To summarize, the architecture of custom CNN model is as follows:



```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout(p=0.2)
  (conv_bn1): BatchNorm2d(224, eps=3, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=12544, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=133, bias=True)
)
```

Further, dropout hyperparameter is used to avoid overfitting. However, this custom CNN model performs poorly, and accuracy obtained on test dataset is just 23%. To improve the performance transfer learning technique is used.

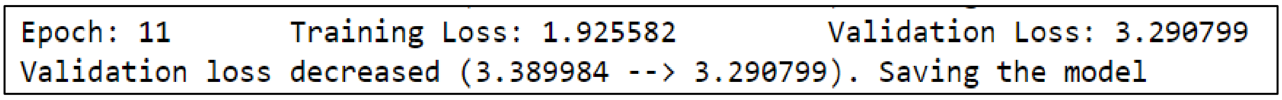
For transfer learning, Resnet50 architecture is used and initial layers are fined tuned. Then, this model was trained for 15 epochs and there is significant improvement in the result. The accuracy obtained is 81%. This really shows that if we have small dataset it is better to used transfer learning technique than training the custom CNN model.

Benchmark

Benchmark Model Result

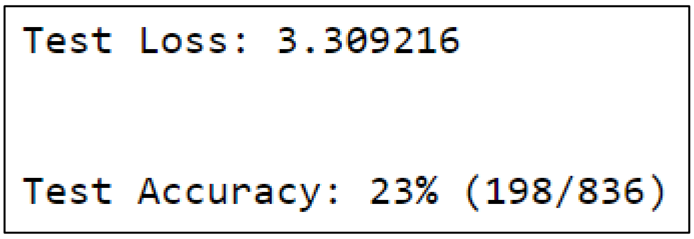
The CNN model we customized can be a benchmark model. During the training process, we saved the model only when the validation loss decreases. For the saved model, we calculated that the training loss was 1.925582 and the validation loss was 3.290799.

The accuracy of test data was 23% (198/836) and test loss was 3.309216.



```
Epoch: 11       Training Loss: 1.925582         Validation Loss: 3.290799
Validation loss decreased (3.389984 --> 3.290799). Saving the model
```

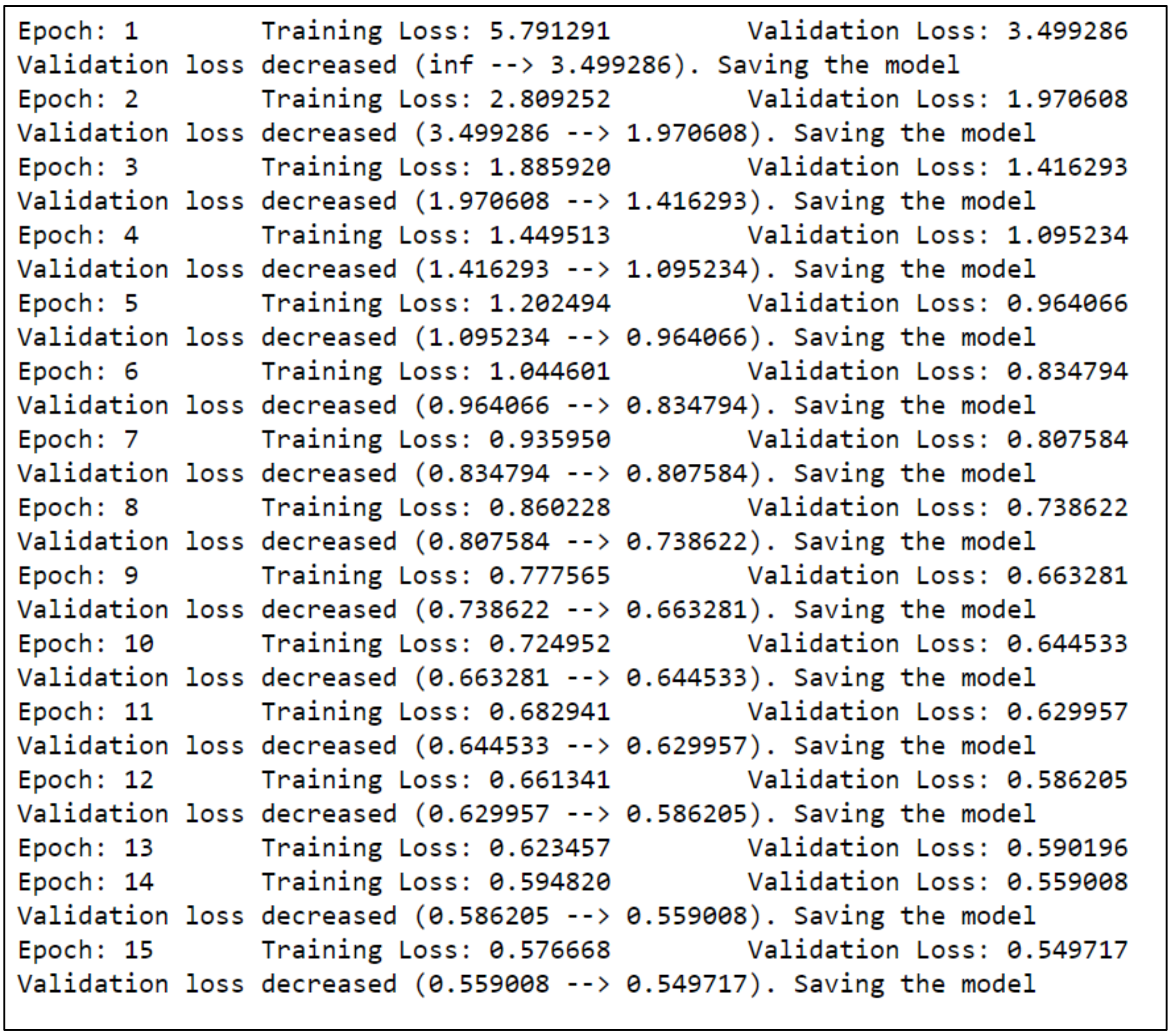
The accuracy of test data was 23% (198/836) and test loss was 3.309216.



```
Test Loss: 3.309216


Test Accuracy: 23% (198/836)
```

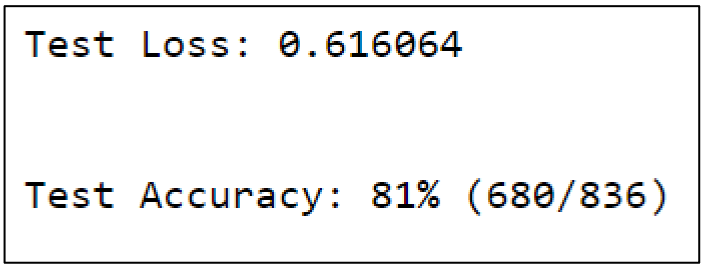
Transfer Learning Model Result

We used ResNet50 to create this model. During the training process, we saved the model only when the validation loss decreases. We ran this model for 15 epochs only. For the saved model, we calculated that the training loss was 0.576668 and the validation loss was 0.549717.



```
Epoch: 1        Training Loss: 5.791291         Validation Loss: 3.499286
Validation loss decreased (inf --> 3.499286). Saving the model
Epoch: 2        Training Loss: 2.809252         Validation Loss: 1.970608
Validation loss decreased (3.499286 --> 1.970608). Saving the model
Epoch: 3        Training Loss: 1.885920         Validation Loss: 1.416293
Validation loss decreased (1.970608 --> 1.416293). Saving the model
Epoch: 4        Training Loss: 1.449513         Validation Loss: 1.095234
Validation loss decreased (1.416293 --> 1.095234). Saving the model
Epoch: 5        Training Loss: 1.202494         Validation Loss: 0.964066
Validation loss decreased (1.095234 --> 0.964066). Saving the model
Epoch: 6        Training Loss: 1.044601         Validation Loss: 0.834794
Validation loss decreased (0.964066 --> 0.834794). Saving the model
Epoch: 7        Training Loss: 0.935950         Validation Loss: 0.807584
Validation loss decreased (0.834794 --> 0.807584). Saving the model
Epoch: 8        Training Loss: 0.860228         Validation Loss: 0.738622
Validation loss decreased (0.807584 --> 0.738622). Saving the model
Epoch: 9        Training Loss: 0.777565         Validation Loss: 0.663281
Validation loss decreased (0.738622 --> 0.663281). Saving the model
Epoch: 10       Training Loss: 0.724952         Validation Loss: 0.644533
Validation loss decreased (0.663281 --> 0.644533). Saving the model
Epoch: 11       Training Loss: 0.682941         Validation Loss: 0.629957
Validation loss decreased (0.644533 --> 0.629957). Saving the model
Epoch: 12       Training Loss: 0.661341         Validation Loss: 0.586205
Validation loss decreased (0.629957 --> 0.586205). Saving the model
Epoch: 13       Training Loss: 0.623457         Validation Loss: 0.590196
Epoch: 14       Training Loss: 0.594820         Validation Loss: 0.559008
Validation loss decreased (0.586205 --> 0.559008). Saving the model
Epoch: 15       Training Loss: 0.576668         Validation Loss: 0.549717
Validation loss decreased (0.559008 --> 0.549717). Saving the model
```

The accuracy of test data was 81% (680/836) and test loss was 0.616064.



```
Test Loss: 0.616064


Test Accuracy: 81% (680/836)
```

We found that for transfer learning model, model has better fit as both training and validation loss decreases to a similar level.

Justification

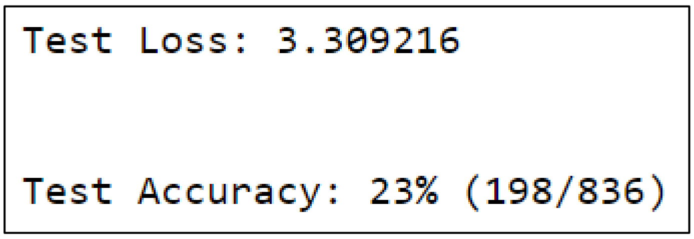
As we have comparatively less dataset for this project, the custom CNN model will have less accuracy. The CNN model created from scratch must have accuracy of at least 10%. This can confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%. Moreover, we will be also using transfer learning technique to improve the performance of this model. Thus, the benchmark for creating CNN model using transfer learning should be at least 60% and this much higher than a custom CNN model.

Formula for **Accuracy = Number of items correctly classified/ All classified items**

Model Evaluation

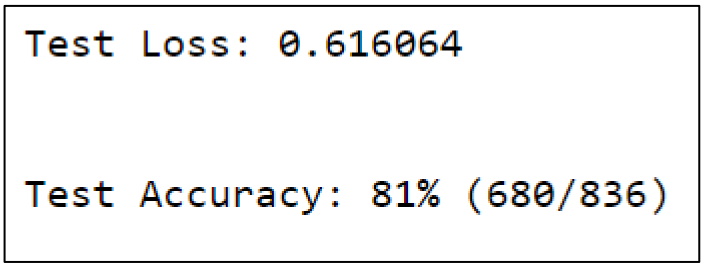
As we have used 2 models in total to classify the dog breed:

1. Custom Dog Classifier (Benchmark Model): This classifier model gave 23% accuracy.



```
Test Loss: 3.309216


Test Accuracy: 23% (198/836)
```

2. Resnet50 model (Transfer Learning Model): This classifier model gave 81% accuracy.



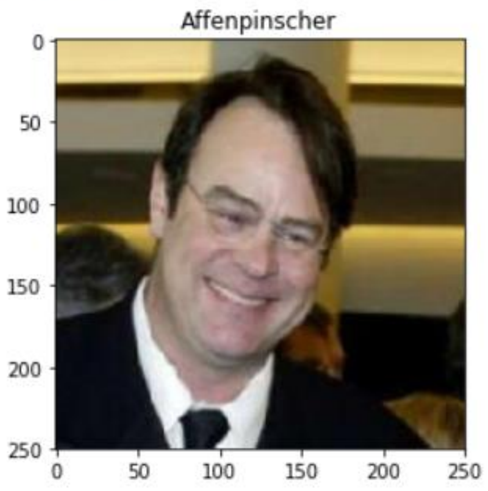
```
Test Loss: 0.616064


Test Accuracy: 81% (680/836)
```

Justification

I think the model performance is better than expected and the performance is appreciable for a complex real-world problem. The model created using transfer learning have an accuracy of 81% compared to the CNN model created from scratch which had only 23% accuracy.

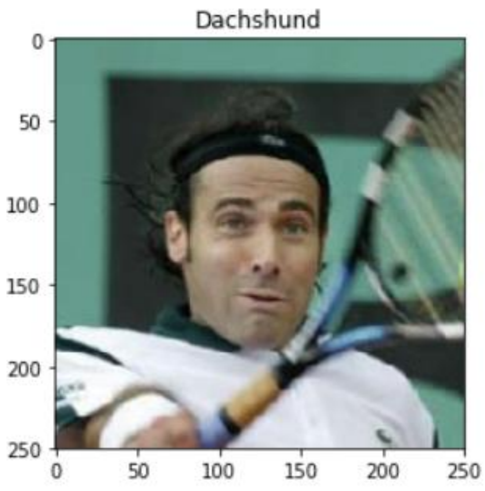
Some of the result where model was able to classify dog breed and human into certain dog breeds are:

```
Hello Human!
```



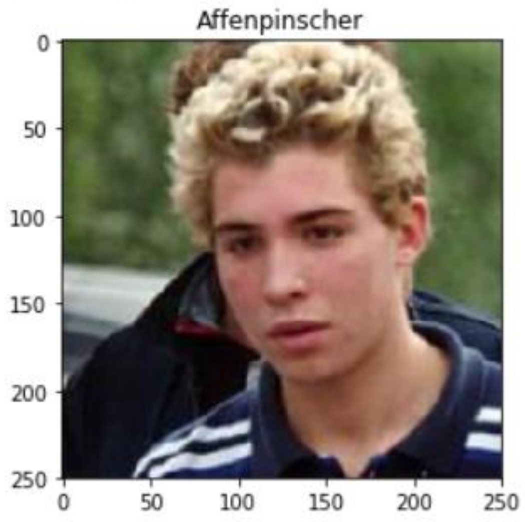
```
You look like: Affenpinscher
```

```
Hello Human!
```



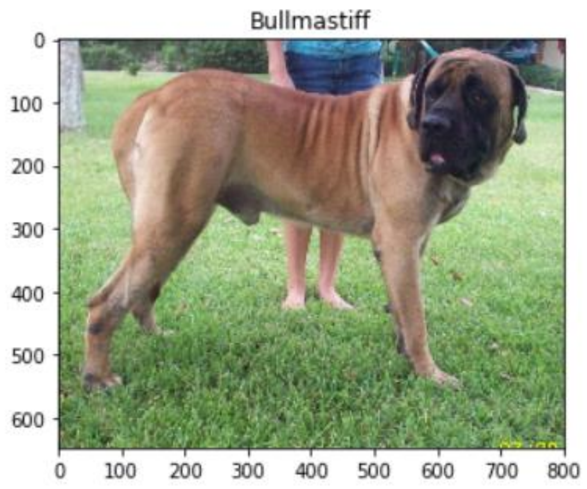
```
You look like: Dachshund
```

```
Hello Human!
```

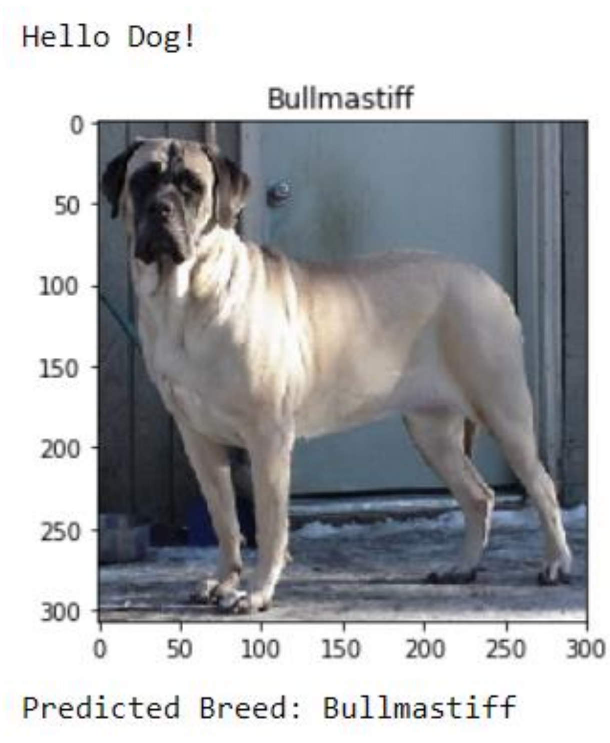


```
You look like: Affenpinscher
```

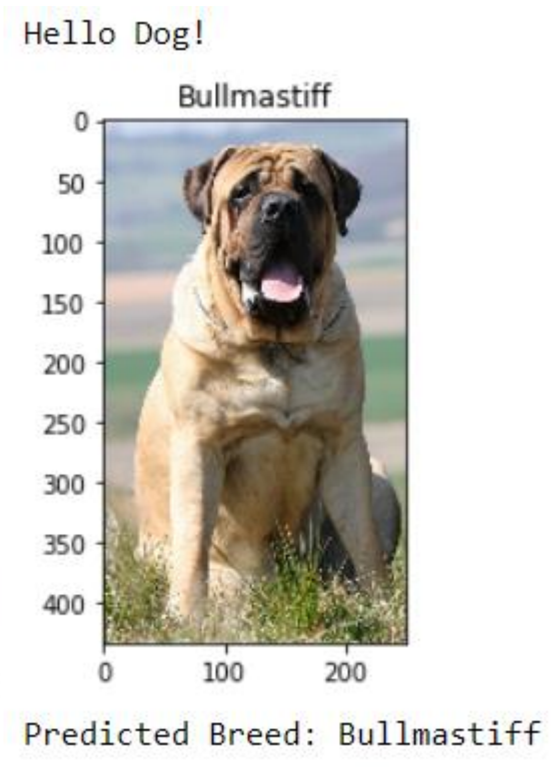
```
Hello Dog!
```



```
Predicted Breed: Bullmastiff
```



```
Hello Dog!
Predicted Breed: Bullmastiff
```



```
Hello Dog!
Predicted Breed: Bullmastiff
```

References

Original repo for Project - GitHub: https://github.com/udacity/deep-learningv2pytorch/blob/master/project-dog-classification

Pytorch Documentation: https://pytorch.org/docs/master/

Resnet50: https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html#resnet50

ImageNet training in Pytorch: https://github.com/pytorch/examples/blob/97304e232807082c2e7b54c597615dc0ad8f6173/imagenet/main.py#L197-L198