PROJECT REPORT

DOG BREED CLASSIFIER WITH CNNs

Paulo Victor de Sousa Moura May/2021

PROJECT OVERVIEW

Humans interact with the world in many ways. One of the most important of these interactions is through vision. Humans can easily interpret the three-dimensional world and analyze and identify objects seen. Today's modern computers can work with captured images, and the increasing use of the Internet in recent decades has provided us with gigantic amounts of image data. As a result, a lot of development is taking place to make a machine extract meaning from the images.

The field of image processing is one of the subfields of machine and deep learning, along with signal processing and many others and is an active area of research. This field is based on image processing by means of mathematical algorithms and uses these algorithms to fulfill many objectives, such as classification, forecasting and even image generation, such as the use of Adversarial Neural Networks (GANs). Some real-world examples include optical character recognition, medical images, self-driving cars, face recognition, object classification, etc. With the advancement in computer hardware, we have been able to perform increasingly expensive tasks, which are one of the reasons why deep learning has taken off. An innovative application of deep neural networks that emerged in the 2012 ImageNet challenge was the use of “Convolutional neural networks” to win the ImageNet competition.

A convolutional neural network, CNN for short, is a specialized type of neural network for processing data that has the structure of a grid. Classification of images with CNN came to light in 1994 with LeNet5 by Yann LeCun.

Given that people can be helped by an image classifier that recognizes dog breeds, in this project, I am interested in seeing how convolutional neural networks work to recognize dog breeds using a dog breed classifier.

In this project, “dog breed classification” problem was explored, where we used pre-trained weights from a VGG-16 model, which was trained on a very large data set, and we trained our own fully connected network to be able to classify images into your correct dog breeds. The data to be used to train our model is provided by Udacity and is explained in more detail in the “Data Exploration” section. The next section describes the problem definition in more detail.

PROBLEM STATEMENT

How can we use convolutional neural networks to recognize the dog breed classifier?

One of the main challenges in the field of image processing is to allow a system to recognize the appearance of things and what are the differences between them. The aim of this project is to build an algorithm that will be able to distinguish between dog breeds and overcome the challenge that many dog breeds would be very similar to. Based on a dog's image, an algorithm should provide an estimate of the dog's breed. If an image of a person is provided, the algorithm should reproduce the most similar breed of dog.

The main goal of this project is to create a CNN to predict a breed of dog from a given image. If the image shows only one human, the algorithm will detect the human and return the dog breed to the human face. To achieve the project objective, a CNN must be made from scratch or a pre-trained model must be

used. Creating a CNN from scratch and comparing performance to a pre-trained model is a mandatory task in this project.

Therefore, we face the following main problems in this project.

1. Create a model to detect a human
2. Create a model to predict the breed of dog
3. Create an algorithm, which will return the expected breed of dog for the image.
4. Achieve 60 percent or more accuracy in the breed classification

METRICS

The goal is to compare the performance of my model with that of the reference model. Therefore, the percentage of accuracy would be an appropriate metric for judging the performance of our model, as it provides a solid measure of how well our model is performing, limited between 0 and 100. Also, because the benchmark model specifies only the precision.

$$accuracy\ \% = \frac{number\ of\ correctly\ predicted\ images}{total\ number\ of\ images} \times 100$$

Additionally, due to unbalanced data set, we can evaluate performance using the F1-Score metric, which is calculated from Precision and Recall metrics.

$$F1Score = 2 * Precision * \frac{Recall}{Precision + Recall}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

Where TP stands for the number of true positive, FP stands for the number of false positives, FN stands for the number of false negatives and FP stands for the number of false positive.

ANALYSIS

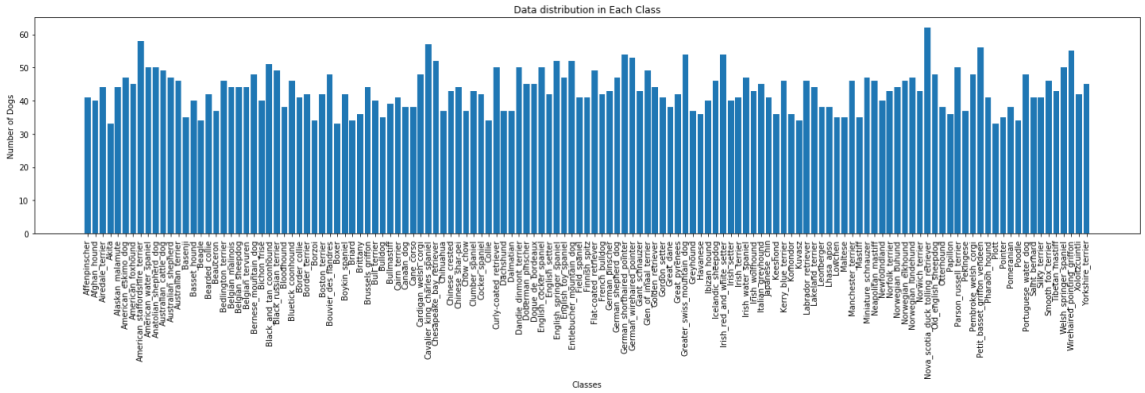
Data Exploration

The dataset to be used in this project is provided by Udacity, which contains images of humans and dogs as well. This dataset is already separated into a training, testing and validation folder.

To classify dog breeds, we will use a dataset that contains 8,351 images of dogs. Therefore, we have 6,680 images to train the model, 836 images for testing and 835 images for validation, which means that 80% of the total data was kept for training and 10% each for validation and testing. The entire data set contains 133 different breeds.

To detect humans, we will use the LFW (Labeled Face in the Wild) dataset from the University of Massachusetts. The LFW dataset consists of face photos. Each face was tagged with the person's name. This data set contains 5,750 common images of ordinary people. These images are used to test the performance of the Human Face Detector.

The study on the distribution of data gives us information about the balance or imbalance in the data. If there is an imbalance in the data beyond a certain threshold, we should see if the data is balanced by adding relevant images. If the balance in the data is comparatively close to the limit, it is a good idea to proceed with the operation.



From the figure we can realize that the data set is imbalanced, it does not have a similar amount of data for each of the classes. The number of images in each class varies from 26 images to 77 images, mean of 50 images per breed. This imbalance can also be seen in the test and validation sets.

Algorithms and Techniques

The solution is to design a CNN model that can estimate as much as possible the breed of a dog that is present in a photo. To do this, one must recognize beforehand whether a person or a dog is present in the image. If a person is present, the similar breed of dog is identified. If a dog is present, an estimate of the breed of the dog is provided.

The solution can be explained in four modules each one with different approaches and algorithms:

- **Face Detector**: First, we need to find a way to detect human faces. To detect humans, we use OpenCV's implementation of cascade classifiers based on Haar Cascades. OpenCV provides many pre-trained face detectors, stored as XML files on GitHub. We use the XML frontface_default. The only input parameter for the Haar classifier is the XML file that contains the pre-trained facial detector. I also compared the performance between OpenCV with haarcascade_frontalface_alt.xml and *Face Recognition* library, and decide to select OpenCV due both lib performance was quite similar

OpenCV haarcascade_frontalface_alt.xml:

- Humans detected in human_files_short: 100
- Humans detected in dog_files_short: 8

Face Recognition Lib:

- Humans detected in human_files_short: 101
- Humans detected in dog_files_short: 7

1Face Detection performance comparison between OpenCV and Face Lib

- **Dog Detection**: In order to classify whether the image is of a dog, we realize that the ImageNet dataset has several labeled dog images. It was used a pre-trained model, the VGG-16 model, which was trained on the ImageNet data set, a very large and very popular data set used for image classification and other vision tasks. We feed our input image into the pre-trained VGG-16 model and obtain a numerical output, which corresponds to the expected output class. If we see the classes corresponding to these indexes, we will find that the categories corresponding to the dogs appear in an uninterrupted sequence and correspond to the keys of the dictionary 151-268, if the output of the VGG-16 model is within this range, we can assume that the image has a dog in it. The entry for our VGG-16 model is a standardized 224 × 224 image tensor. As we are using the model's pre-trained weights, there is no training step involved. It was evaluated the performance with other model architectures such as IncepctionV3 and ResNet-50, however the overall results were similar between them. Considered that outcome the VGG-16 model was selected.

```
Model: VGG-16
0.00% dogs found on human files
100.00% dogs found on dogs files

_________________________________________________________________
Model: InceptionV3
0.00% dogs found on human files
100.00% dogs found on dogs files

_________________________________________________________________
Model: ResNet-50
0.00% dogs found on human files
100.00% dogs found on dogs files
```

2 Dog detection comparison among different model architectures

- **Dog Breed Classifier**: It was applied the technique transfer learning. In transfer learning, a pretrained model is used which is then only modified according to the intended use. Following choose made for the Dog Detector, a pre- trained VGG16 model was selected. Pre-trained models used for the ImageNet challenge were trained on a very large data set and, therefore, their convolutional layers have significant information about the image, such as the different borders, orientation variations, etc. This information that the model has learned can be applied to our problem as well. Thus, we use the pre-trained weights from the model's convolutional layer and retrain our own fully connected layer added to it, which will be used to produce a probabilistic output from the 133 output classes according to our use case scenario.
- **Demo Web Application**: It was developed a web application using Flask and deployed as an AWS ECS (Fargate) service. Flask is a lightweight WSGI web application framework and the app were wrapped as a Docker image in order to expedite the deployment task. This simple web app enables the dog classification tasks through a straightforward user interface.

BENCHMARK

The following conditions served as benchmarks for our model:

● Building a CNN from scratch that should exceed 10% accuracy. This is a challenging task since image processing models need many iterations and times that take a long time to finish training.

● Using a pre-trained model. It must achieve an accuracy of at least 60%, so that it can be used successfully in a dog breed classifier application.

METHODOLOGY

Data Preprocesing

Not all images have the same resolution, and as observed before, the distribution of the images is not the same in the various dog breeds. Here there is a slight data imbalance.

For the data preprocessing step the PyTorch module *torchvision.transforms* was used.

All images of the section test and valid are first reduced to 256x256 pixels. Then a CenterCrop to 224x224 pixels is performed.

All images of the section train are first reduced to 256x256 pixels. Afterwards, data augmentation is applied to counteract the data imbalance. After the images are reduced in size, a RandomResizedCrop to 224x224 pixels is performed. Then a RandomHorizontalFlip is performed and finally a RandomRotation of 30 degrees.

```
train_transforms = transforms.Compose([transforms.Resize(size=256),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.RandomRotation(30),
                                       transforms.ToTensor()])

validTest_transforms = transforms.Compose([transforms.Resize(size=256),
                                           transforms.CenterCrop(224),
                                           transforms.ToTensor()])
```

I decide for not include an image normalization step since, after several attempts running few epochs, the training accuracy was higher when not normalizing the input.

Implementation

During the initial experiments training the model from scratch I decided for using 5 convolutional layers for this project and 3 fully connected layers. A

Dropout layer is added at the end so that it will reduce the complexity of the network.

5 convolutional layers are created with max-pooling layers in between them to learn a hierarchy of high-level features. Max pooling layer is added to reduce the dimensionality. Dropout was used along with the flattening layer before using the fully connected layer to reduce overfitting and ensure that the network generalizes well. The number of nodes in the last fully connected layer was set up as 133. ReLu activation function was used for most of the layers.

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=6400, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=500, bias=True)
  (fc3): Linear(in_features=500, out_features=133, bias=True)
  (conv_bn): BatchNorm2d(224, eps=3, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

The final version of the app was implement considering the following steps:

- **Human Face Detector**

  As mentioned in the section, “Algorithms and Techniques”, we use OpenCV's implementation of Haar cascade classifiers to detect human faces in images. As said in the “Data Preprocessing” section, the image was converted into grayscale, using the OpenCV method, “cvtColor”. The XML containing the pre-trained classifier was loaded using OpenCV’s “CascadeClassifier”. To find the number of faces in the image, the method detectMultiScale() was used. This returns the number of human faces that the classifier has detected. A function called face_detector() was written that would take in an image path a parameter, load it, convert it to grayscale, and use detectMultiScale() to get the number of human faces in the image. The return value is a boolean, which returns

true if the number of humans is greater than 0 and returns false otherwise.

- **Dog Detector**

  The dog detector is made using a VGG16 pre-trained model. PyTorch already provides many pre-trained models, readily available for use. The VGG 16 model was obtained and loaded with its pre-trained weights. In the predict function, the transformations described in the "Data Preprocessing" step are implemented, and the image is fed into the model. If the output lies in 151 and 268 (inclusive), we claim that the image contains a dog.

- **Dog Breed Classifier**

  At the heart of our project is our dog breed classifier. The dog classifier makes use of a VGG-16 model. Data processing is like that done in the dog detector. The VGG-16 model's pre-trained weights from the convolutional layer have been taken and used for the dog breed classifier model. We train our own fully connected layer, which takes 25088 input features, has a first layer with 4096 nodes, and finally has 512 nodes in the second hidden layer, and 133 output classes in the final layer. The model is trained using Stochastic Gradient Descent, with a learning rate of 0.001 and momentum of 0.9. The model was trained for 25 epochs, and its performance was measured according to the section "Metrics".

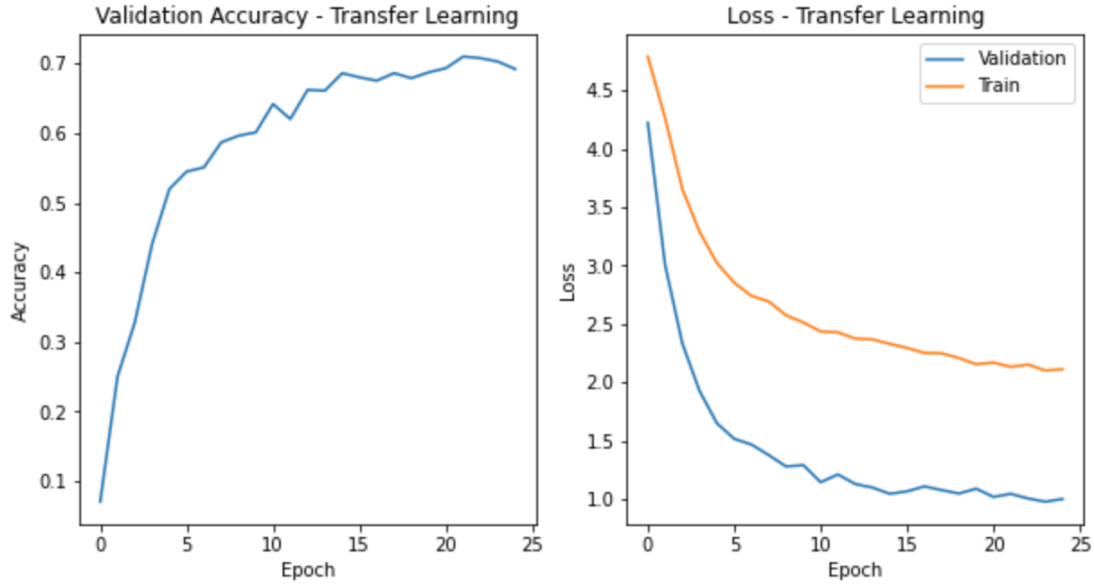
```
model_transfer = models.vgg16(pretrained=True)

for param in model_transfer.parameters():
    param.requires_grad = False

# Define dog breed classifier
model_transfer.classifier = nn.Sequential(nn.Linear(25088, 4096),
                                          nn.ReLU(), nn.Dropout(0.5),
                                          nn.Linear(4096, 512),
                                          nn.ReLU(),
                                          nn.Dropout(0.5),
                                          nn.Linear(512, 133))
```

Refinement

The initial model consisted of Stochastic Gradient Descent with a learning rate of 0.05, up to 20 epochs, which achieved an accuracy of 65%. By using a learning rate of 0.001 and increasing the number of epochs to 25, we were able to increase it to 70%.



RESULTS

Model Evaluation and Validation

The model was evaluated using a validation set during training, and then on the test set. The best performing hyperparameters were chosen from all the combinations.

The final model uses Stochastic Gradient Descent with a learning rate of 0.001 and has a test accuracy of 70%. This sufficiently meets our expectations and can be used in our final app.

To verify the robustness of the model, I compared the performance against other metrics such as Precision, Recall and F1-Score.

	precision	recall	f1-score
accuracy			0.71
macro avg	0.71	0.69	0.67
weighted avg	0.73	0.71	0.69

3 Final Model Metrics

	precision	recall	f1-score
001.Affenpinscher	0.86	0.75	0.80
002.Afghan_hound	0.89	1.00	0.94
003.Airedale_terrier	1.00	0.83	0.91
004.Akita	0.50	0.25	0.33
005.Alaskan_malamute	0.82	0.90	0.86
006.American_eskimo_dog	0.88	0.88	0.88
007.American_foxhound	0.58	1.00	0.74
008.American_staffordshire_terrier	0.78	0.88	0.82
009.American_water_spaniel	1.00	0.25	0.40
010.Anatolian_shepherd_dog	1.00	0.83	0.91
011.Australian_cattle_dog	0.80	0.89	0.84
012.Australian_shepherd	0.88	0.78	0.82
013.Australian_terrier	1.00	0.83	0.91
014.Basenji	0.89	0.89	0.89
015.Basset_hound	0.90	0.90	0.90
016.Beagle	1.00	0.38	0.55
017.Bearded_collie	0.50	0.75	0.60
018.Beauceron	0.83	0.71	0.77
019.Bedlington_terrier	1.00	0.67	0.80
020.Belgian_malinois	0.55	0.75	0.63
021.Belgian_sheepdog	0.54	0.88	0.67
022.Belgian_tervuren	0.75	0.50	0.60
023.Bernese_mountain_dog	1.00	0.88	0.93
024.Bichon_frise	1.00	0.75	0.86
025.Black_and_tan_coonhound	1.00	1.00	1.00
026.Black_russian_terrier	0.57	0.80	0.67
027.Bloodhound	1.00	0.88	0.93
028.Bluetick_coonhound	0.80	1.00	0.89
029.Border_collie	0.83	1.00	0.91
030.Border_terrier	0.86	0.86	0.86
031.Borzoi	1.00	0.71	0.83
032.Boston_terrier	1.00	0.88	0.93
033.Bouvier_des_flandres	1.00	0.20	0.33
034.Boxer	0.70	0.88	0.78
035.Boykin_spaniel	0.50	0.33	0.40
036.Briard	0.88	0.88	0.88
037.Brittany	0.50	1.00	0.67
038.Brussels_griffon	1.00	0.86	0.92
039.Bull_terrier	0.89	0.89	0.89
040.Bulldog	0.86	0.86	0.86
041.Bullmastiff	0.56	0.56	0.56
042.Cairn_terrier	0.75	0.38	0.50
043.Canaan_dog	0.50	0.33	0.40
044.Cane_corso	0.55	0.75	0.63
045.Cardigan_welsh_corgi	0.56	0.71	0.63
046.Cavalier_king_charles_spaniel	0.64	0.78	0.70
047.Chesapeake_bay_retriever	0.80	0.57	0.67
048.Chihuahua	1.00	0.43	0.60
049.Chinese_crested	0.86	1.00	0.92
050.Chinese_shar-pei	1.00	1.00	1.00
051.Chow_chow	1.00	0.62	0.77
052.Clumber_spaniel	1.00	0.67	0.80
053.Cocker_spaniel	0.50	0.17	0.25
054.Collie	0.78	1.00	0.88
055.Curly-coated_retriever	0.40	0.86	0.55
056.Dachshund	0.67	0.44	0.53
057.Dalmatian	1.00	1.00	1.00
058.Dandie_dinmont_terrier	0.83	0.71	0.77
059.Doberman_pinscher	0.50	0.83	0.62
060.Dogue_de_bordeaux	0.88	0.88	0.88
061.English_cocker_spaniel	0.50	0.25	0.33
062.English_setter	0.57	0.67	0.62
063.English_springer_spaniel	0.62	0.71	0.67
064.English_toy_spaniel	0.67	0.40	0.50
065.Entlebucher_mountain_dog	1.00	0.60	0.75
066.Field_spaniel	0.40	0.50	0.44
067.Finnish_spitz	0.33	0.25	0.29
068.Flat-coated_retriever	0.71	0.62	0.67
069.French_bulldog	0.78	1.00	0.88
070.German_pinscher	0.33	0.17	0.22
071.German_shepherd_dog	0.86	0.75	0.80
072.German_shorthaired_pointer	0.50	1.00	0.67
073.German_wirehaired_pointer	0.50	0.60	0.55
074.Giant_schnauzer	0.50	0.80	0.62
075.Glen_of_imaal_terrier	0.33	0.40	0.36
076.Golden_retriever	0.56	0.62	0.59
077.Gordon_setter	0.56	1.00	0.71
078.Great_dane	0.50	0.40	0.44
079.Great_pyrenees	0.55	0.75	0.63
080.Greater_swiss_mountain_dog	0.71	1.00	0.83
081.Greyhound	0.80	0.57	0.67
082.Havanese	0.83	0.62	0.71
083.Ibizan_hound	1.00	1.00	1.00
084.Icelandic_sheepdog	0.44	0.67	0.53
085.Irish_red_and_white_setter	1.00	0.25	0.40
086.Irish_setter	0.58	1.00	0.74
087.Irish_terrier	0.55	0.75	0.63
088.Irish_water_spaniel	0.50	0.17	0.25
089.Irish_wolfhound	0.60	0.43	0.50
090.Italian_greyhound	0.50	0.75	0.60
091.Japanese_chin	0.78	1.00	0.88
092.Keeshond	1.00	1.00	1.00
093.Kerry_blue_terrier	1.00	0.50	0.67
094.Komondor	1.00	1.00	1.00
095.Kuvasz	0.67	0.33	0.44
096.Labrador_retriever	1.00	0.80	0.89
097.Lakeland_terrier	1.00	0.33	0.50
098.Leonberger	0.83	1.00	0.91
099.Lhasa_apso	0.44	0.80	0.57
100.Lowchen	0.00	0.00	0.00
101.Maltese	0.71	0.83	0.77
102.Manchester_terrier	0.60	1.00	0.75
103.Mastiff	0.43	0.43	0.43
104.Miniature_schnauzer	0.71	1.00	0.83
105.Neapolitan_mastiff	0.60	0.75	0.67
106.Newfoundland	0.67	1.00	0.80
107.Norfolk_terrier	0.57	0.67	0.62
108.Norwegian_buhund	0.00	0.00	0.00
109.Norwegian_elkhound	0.80	0.80	0.80
110.Norwegian_lundehund	1.00	0.50	0.67
111.Norwich_terrier	0.50	1.00	0.67
112.Nova_scotia_duck_tolling_retriever	1.00	0.71	0.83
113.Old_english_sheepdog	0.00	0.00	0.00
114.Otterhound	0.50	0.50	0.50
115.Papillon	0.70	0.88	0.78
116.Parson_russell_terrier	1.00	0.50	0.67
117.Pekingese	0.83	0.83	0.83
118.Pembroke_welsh_corgi	0.50	0.57	0.53
119.Petit_basset_griffon_vendeen	0.80	1.00	0.89
120.Pharaoh_hound	1.00	1.00	1.00
121.Plott	0.00	0.00	0.00
122.Pointer	0.67	0.50	0.57
123.Pomeranian	1.00	0.80	0.89
124.Poodle	0.50	0.50	0.50
125.Portuguese_water_dog	0.40	0.50	0.44
126.Saint_bernard	0.67	0.67	0.67
127.Silky_terrier	0.67	0.40	0.50
128.Smooth_fox_terrier	0.67	0.50	0.57
129.Tibetan_mastiff	1.00	0.50	0.67
130.Welsh_springer_spaniel	0.67	0.40	0.50
131.Wirehaired_pointing_griffon	0.50	0.67	0.57
132.Xoloitzcuintli	0.75	1.00	0.86
133.Yorkshire_terrier	0.57	1.00	0.73

4 Model Metrics Per Class

Justification

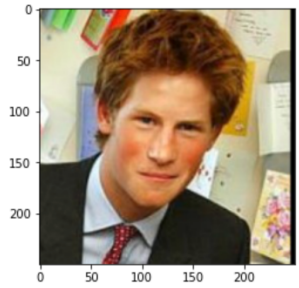
The final model has an accuracy of 70% on the test dataset, which is a better performance than our benchmark of 60%. With the discussions made before on the complexity of classifying dog images, this is a reasonable accuracy that we can use in our dog breed classifier app.

CONCLUSION

Free-Form Visualization

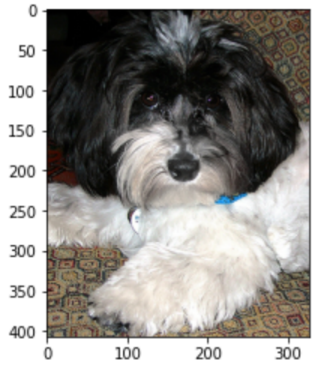
Here is provided some screenshots of the output produced by our model.

```
Hello, Human
```



```
 If you were a dog you could look like a  Nova scotia duck tolling retriever
Top  2 prediction:  20.89% - Irish setter
Top  3 prediction:  20.56% - Brittany
Top  4 prediction:  18.54% - Dachshund
Top  5 prediction:  16.09% - Dogue de bordeaux
```

```
Woof Woof... it is a dog
```

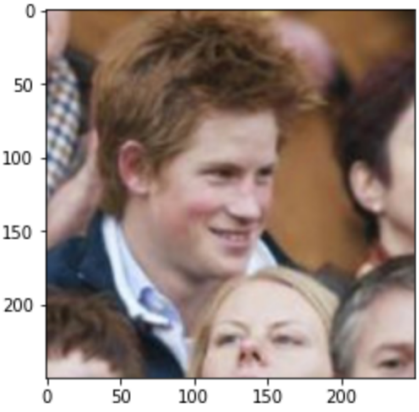


```
Predicted breed: Havanese
Top  2 prediction:  23.06% - Miniature schnauzer
Top  3 prediction:  19.17% - Lowchen
Top  4 prediction:  17.12% - Petit basset griffon vendeen
Top  5 prediction:   4.66% - Bearded collie
```

```
Woof Woof... it is a dog
```


```
Predicted breed: Lowchen
Top  2 prediction:  31.58% - Havanese
Top  3 prediction:  17.21% - Bearded collie
Top  4 prediction:  15.33% - Lhasa apso
Top  5 prediction:   3.09% - Maltese
```

```
Hello, Human
```




```
 If you were a dog you could look like a  Brittany
Top  2 prediction:  24.87% - Clumber spaniel
Top  3 prediction:  17.73% - Welsh springer spaniel
Top  4 prediction:  15.59% - Japanese chin
Top  5 prediction:  13.07% - Bulldog
```

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant public dataset of images were found
2. The images were downloaded
3. A Human Face Detector was developed
4. A dog detector was developed
5. The images were pre-processed, and data augmentation was applied
6. A Model from Scratch was developed, trained and tested
7. A Model with transfer learning was developed, trained and tested
8. An application was designed and developed to show the user the top 3 predictions for a given image
9. A website was designed and developed to show prediction for a given image
10. The GitHub repository has been clearly documented and provided with README files

The most difficult aspect of the project was to be able to tune our final dog breed classifier for the best results.

Improvements

Some areas of improvement in this project have been listed below:

- Improve face detecting algorithm, replacing OpenCV Haar Cascades with some robust Object Detection network trained to detect faces or use Dlib capabilities to do so.
- Better control the scenario where both dog and human are present in the same picture
- Try different alternatives for classifier layers of the transfer learning model. Try to change only the FC layer and compare the performance.
- Expand training with more epochs and test the influence of the batch_size in the overall performance

- Try to improve model's accuracy by fine-tuning hyperparameters and testing different backbone models (Inceptionv3, ResNet-50, VGG-19)
- Leverage SageMaker SDK and API for model training and model deployment purposes.