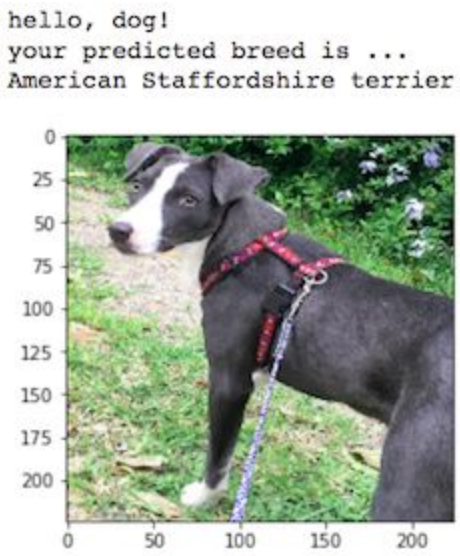
Project Overview:
In this project, I will make the first steps towards developing an algorithm that could be used as part of a mobile or web app. At the end of this project, my code will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. 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.



Problem Statement:
Developing an algorithm that could be used as part of a mobile or web app. At the end of this project, the code will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling.

Following are the steps are taken to solve the problem:

- Step 0: Import Datasets
- Step 1: Detect Humans
- Step 2: Detect Dogs
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
- Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 5: Writing the Algorithm which works as an interface.
- Step 6: Testing the Algorithm

Metrics:
To measure the performance of the model. I have used the accuracy on the test dataset. While training the model, I have used the accuracy of the training dataset to

measure if the model is training or not and validation dataset accuracy to avoid the overfitting problem of the model.

Data Exploration:

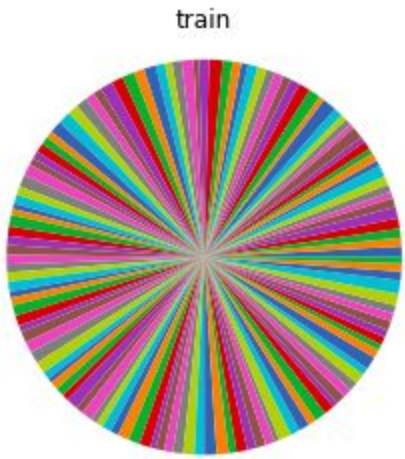
Dog images dataset and Human faces dataset was provided by Udacity.
Dog images dataset is divided into three folders train, valid and test. Each folder contains 133 subfolders for different species of dog breeds which are:

001.Affenpinscher
002.Afghan_hound
003.Airedale_terrier
004.Akita
005.Alaskan_malamute
006.American_eskimo_dog
007.American_foxhound
008.American_staffordshire_terrier
009.American_water_spaniel
010.Anatolian_shepherd_dog
011.Australian_cattle_dog
012.Australian_shepherd
013.Australian_terrier
014.Basenji
015.Basset_hound
016.Beagle
017.Bearded_collie
018.Beauceron
019.Bedlington_terrier
020.Belgian_malinois
021.Belgian_sheepdog
022.Belgian_tervuren
023.Bernese_mountain_dog
024.Bichon_frise
025.Black_and_tan_coonhound
026.Black_russian_terrier
027.Bloodhound
028.Bluetick_coonhound
029.Border_collie
030.Border_terrier
031.Borzoi
032.Boston_terrier
033.Bouvier_des_flandres
034.Boxer
035.Boykin_spaniel
068.Flat-coated_retriever
069.French_bulldog
070.German_pinscher
071.German_shepherd_dog
072.German_shorthaired_pointer
073.German_wirehaired_pointer
074.Giant_schnauzer
075.Glen_of_imaal_terrier
076.Golden_retriever
077.Gordon_setter
078.Great_dane
079.Great_pyrenees
080.Greater_swiss_mountain_dog
081.Greyhound
082.Havanese
083.Ibizan_hound
084.Icelandic_sheepdog
085.Irish_red_and_white_setter
086.Irish_setter
087.Irish_terrier
088.Irish_water_spaniel
089.Irish_wolfhound
090.Italian_greyhound
091.Japanese_chin
092.Keeshond
093.Kerry_blue_terrier
094.Komondor
095.Kuvasz
096.Labrador_retriever
097.Lakeland_terrier
098.Leonberger
099.Lhasa_apso
100.Lowchen
101.Maltese
102.Manchester_terrier

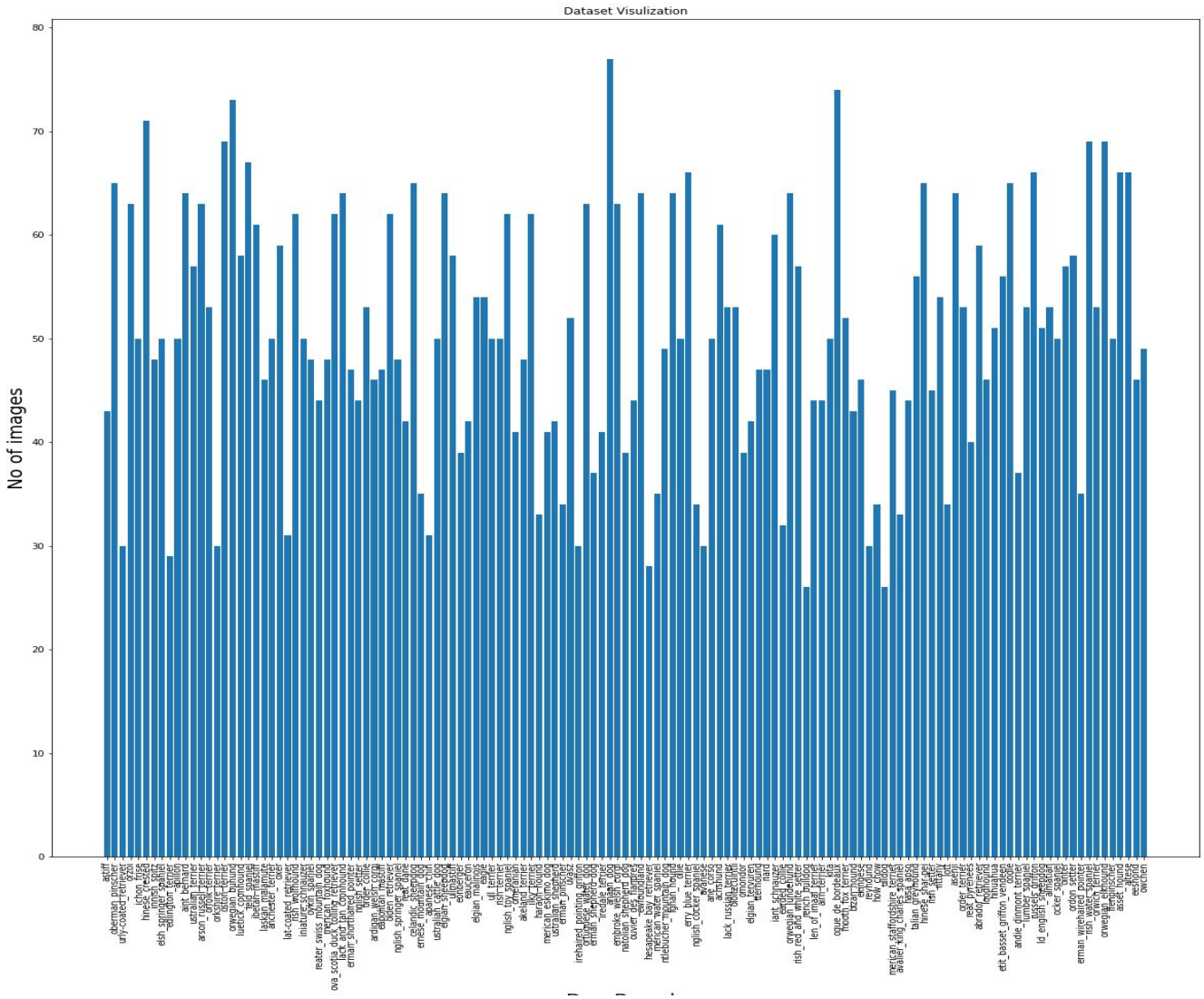
036.Briard
037.Brittany
038.Brussels_griffon
039.Bull_terrier
040.Bulldog
041.Bullmastiff
042.Cairn_terrier
043.Canaan_dog
044.Cane_corso
045.Cardigan_welsh_corgi
046.Cavalier_king_charles_spaniel
047.Chesapeake_bay_retriever
048.Chihuahua
049.Chinese_crested
050.Chinese_shar-pei
051.Chow_chow
052.Clumber_spaniel
053.Cocker_spaniel
054.Collie
055.Curly-coated_retriever
056.Dachshund
057.Dalmatian
058.Dandie_dinmont_terrier
059.Doberman_pinscher
060.Dogue_de_bordeaux
061.English_cocker_spaniel
062.English_setter
063.English_springer_spaniel
064.English_toy_spaniel
065.Entlebucher_mountain_dog
066.Field_spaniel
067.Finnish_spitz
103.Mastiff
104.Miniature_schnauzer
105.Neapolitan_mastiff
106.Newfoundland
107.Norfolk_terrier
108.Norwegian_buhund
109.Norwegian_elkhound
110.Norwegian_lundehund
111.Norwich_terrier
112.Nova_scotia_duck_tolling_retriever
113.Old_english_sheepdog
114.Otterhound
115.Papillon
116.Parson_russell_terrier
117.Pekingese
118.Pembroke_welsh_corgi
119.Petit_basset_griffon_vendeen
120.Pharaoh_hound
121.Plott
122.Pointer
123.Pomeranian
124.Poodle
125.Portuguese_water_dog
126.Saint_bernard
127.Silky_terrier
128.Smooth_fox_terrier
129.Tibetan_mastiff
130.Welsh_springer_spaniel
131.Wirehaired_pointing_griffon
132.Xoloitzcuintli
133.Yorkshire_terrier

Similarly, Human images dataset contains subfolders of human faces with the name of the person as the name of the folder. Overall, there are 13233 total human images and 8351 total dog images, approximately there are 50 images per breed.
The resolutions of the images vary with images, so in the preprocessing part, I have resized all the images to 3 x 224 x 224.

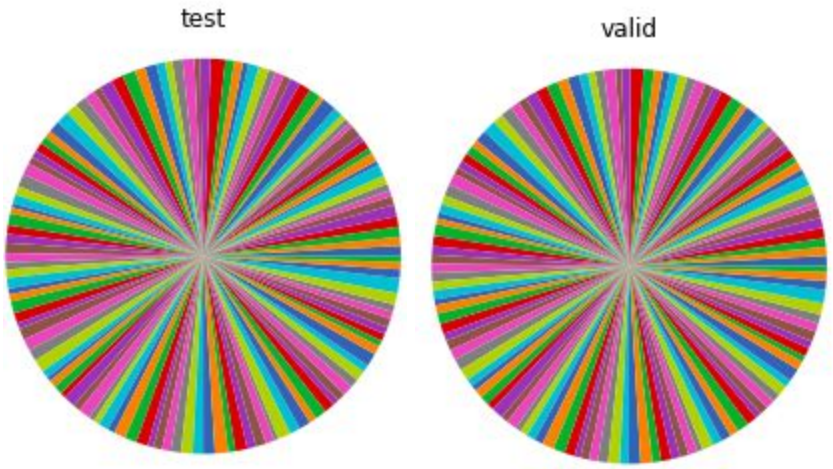
Exploratory Visualization:
Here is the piechart for the training dataset which depicts approximately that all the dog breeds have the almost same number of images ie approximately 50.



However, the exact count can be depicted from the following bar graph:



The pie charts for the test and validation dataset are almost similar:



Algorithms and Techniques:
I have written an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide an output that indicates an error.

So, first I have passed the image to the dog_detector as it had a 100% accuracy on the test data if the passed image is a dog. If the passed image is a dog then I have plotted the image and printed the message along with its breed.

If the passed image is not of dog then I have checked if the passed image is of human with the face_detector, if the human face is detected then I have printed *Human Detected!!* then I have also printed the message that which dog breed is most similar to that face.

If the passed image is neither dog nor human then I have printed the error message and plotted the image.

As I have used CNN here, I will define some basic terms here first then explain about the CNN (Convolutional Neural Network).

So, first what is convolution? **convolution** is a mathematical operation on two functions (f and g) that produces a third function expressing how the shape of one is modified by the other. On images, we apply a 2d convolution in which we start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data,

performing an element-wise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

Typical image convolution is used in neural networks as a form of blurring (though there are other kinds of convolutions that are closer to spatial derivatives). This blurring ensures that the neural network will respond similarly to slightly displaced versions of an image or feature.

Benchmark: The model can be tested with different kinds of images if the image is detected to be a dog or human or something else. It can be tested with other animals' images like a cat and checked whether the passed image is identified as a dog or other. So, as per the instructions in the notebook, an accuracy of 10% is expected on the model designed from scratch and a minimum of 60% accuracy is expected for the transfer learning model.

I have used the cross-entropy loss which can be mathematically defined as:

```
loss(x, class) = weight[class] * (-x[class] + log(\sum_j exp(x[j])))
```

The input expected into the cross-entropy loss function from pytorch is a 2d tensor of size (minibatch, C)

Shape:

- Input: (N,C) where C = number of classes
- Target: (N) where each value is 0 <= targets[i] <= C-1
- Output: scalar. If reduce is `False`, then (N) instead.

There are 16 loss functions defined in the pytorch source code which are:
`nn.L1Loss:` criterion that measures the mean absolute value of the element-wise difference between input x and target y
`nn.MSELoss:` criterion that measures the mean squared error between n elements in the input x and target y
`nn.CrossEntropyLoss:` combines LogSoftMax and NLLLoss in one single class
`nn.NLLLoss:` negative log-likelihood loss
`nn.PoissonNLLLoss:` Negative log-likelihood loss with the Poisson distribution of target
`nn.NLLLoss2d:` negative log-likelihood loss, but for image inputs
`nn.KLDivLoss:` useful distance measure for continuous distributions and is often useful when performing direct regression over the space of (discretely sampled) continuous output distributions.
`nn.BCELoss:` measures the Binary Cross Entropy between the target and the output
`nn.BCEWithLogitsLoss:` combines a Sigmoid layer and the BCELoss in one single class

`nn.MarginRankingLoss:` measures the loss gave inputs x1, x2, two 1D mini-batch Tensor`s, and a label 1D mini-batch tensor `y with values (1 or -1)
`nn.HingeEmbeddingLoss:` Measures the loss gave an input tensor x and a labels tensor y containing values (1 or -1). This is usually used for measuring whether two inputs are similar or dissimilar
`nn.MultiLabelMarginLoss:` criterion that optimizes a multi-class multi-classification hinge loss (margin-based loss) between input x (a 2D mini-batch Tensor) and output y
`nn.SmoothL1Loss:` criterion that uses a squared term if the absolute element-wise error falls below 1
`nn.SoftMarginLoss:` criterion that optimizes a two-class classification logistic loss between input x (a 2D mini-batch Tensor) and target y (which is a tensor containing either 1 or -1).
`nn.MultiLabelSoftMarginLoss:` criterion that optimizes a multi-label one-versus-all loss based on max-entropy, between input x (a 2D mini-batch Tensor) and target y (a binary 2D Tensor)
`nn.TripletMarginLoss:` criterion that measures the triplet loss given an input tensors x1, x2, x3 and a margin with a value greater than 0

Out of the 16 loss functions, first I checked for all the loss functions which can be applied to the images then I further refined for multi-class classification problems, which left me with 5 loss functions as **`CrossEntropyLoss`** is a combination of the two loss functions ie LogSoftMax and NLLLoss and useful when training a classification problem with C classes. This is useful when we have an unbalanced training set.

Data Preprocessing:

For the training data preprocessing I have used *RandomResizedCrop* which crops the image of random size (default: of 0.08 to 1.0) of the original size and a random aspect ratio (default: of 3/4 to 4/3) of the original aspect ratio is made. This crop is finally resized to the given size. Then I have used *RandomHorizontalFlip* with a probability of 0.5 which will horizontally flip the given PIL Image randomly with a given probability. Then I have also used the *RandomVerticalFlip* with a probability of 0.5 which will vertically flip the given PIL Image randomly with a given probability. After that, I have converted the image to tensor and normalized the image.

For the validation dataset, I have simply used the *RandomResizedCrop* then converted to tensor and normalized it.

For the test dataset, I have used the *Resize* which resize the input PIL Image to the given size. Then I have converted to tensor and normalized it.

After applying the transforms to the images I have loaded the images with shuffling to avoid overfitting problems, with the batch_size of 50 and the number of workers as 0.

Implementation:

For detecting humans, I have used the haar cascades from OpenCV, which gave me around 97% accuracy on human faces as this is not quite satisfactory, then I had tried the python dlib library face_detector which improved the accuracy to 100% on first 100 human files.

For the detection of dogs, I have used the vgg16 pretrained model from pytorch which has the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from 'Chihuahua' to 'Mexican hairless'. It has an accuracy of 100% on the dogs' images dataset.

For the implementation of the model from scratch first I have created a CNN model which was really a decisive work, so I did some research and referred some papers and works related to the dog breed classifiers. After going through them I decided to go with 5 convolutional layers with max-pooling layers then I flattened the parameters and applied two layers of perceptrons which I used to classify among the various breeds. I also applied the dropouts to the linear layers to avoid overfitting and leaving the nodes untrained. However, this model didn't give much better accuracy (21%), so I further refined my model with Transfer Learning which is discussed below.

Refinement:

After trying my own model, I looked into which discusses the advantages of transfer learning so, I applied transfer learning on the resnet50 model which improved the accuracy to 76%. So, first I froze all the layers, then changed the last layer as output features of resnet50 are 1000, while our classifier needs just 133 classes. After changing the last layer I trained the last layer of the model for around 100 epochs which I think can be optimized with the use of scheduler. Also after training about 200 epochs, I got an accuracy of 86%.

Model Evaluation and Validation:

After training the model, the model was tested on the test dataset which gave an accuracy of 86% after training for 200 epochs and 76% after training for 100 epochs. The validation loss for the final model was 1.112262%.

Justification:

After making a comparison with the benchmark result, I think the project is successfully completed ie my model_scratch has obtained an accuracy of 21% on the test dataset and an accuracy of 76% with model_transfer.

Here are the some sample outputs: