Machine Learning Engineer Nanodegree

Capstone Project

By Pooja Vishnoi

Project Overview: The project is about building a convolutional neural network aimed at processing and identifying user- supplied supplied images. In this project particularly, identification of dog breeds is done.

Problem Statement: Write an Algorithm for a Dog Identification App that identifies an image as a dog or a human and depicts the breed of a dog, given the image of a dog.

Strategy to solve the problem
1. Import Datasets
2. Detect Humans using Harr Feature Based Cascade Classifier
3. Detect Dogs using pre trained VGG 16 model
4. Create a CNN to Classify Dog Breeds (from Scratch)
5. Create a CNN to Classify Dog Breeds (using Transfer Learning)
6. Write your Algorithm
7. Test Your Algorithm

Metrics: The CNN model to classigy dog breed from scratch should have an accuracy of atleast 10%. Also, the CNN model to classify dog breeds using transfer learning should have an accuracy of atleast 60%.

Data Exploration and visualisation:

The dog dataset contains 8351 dog images and human dataset contains 13233 human images.
To get insight into how the CNN models decompose the images into pixels to perform face recognition, histograms are used. With the help of histograms, I analysed the no. of pixels at different intensity values. It

helps to get an idea about contrast, brightness , intensity distribution etc of an image.

Some of the human and dog images that I analysed are:

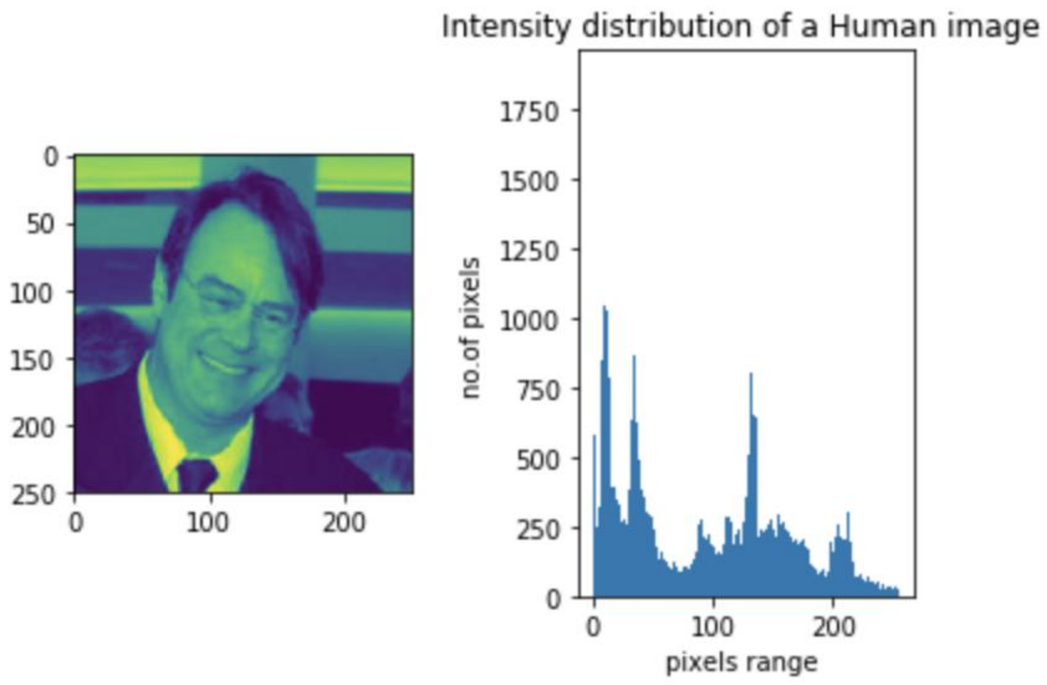


Fig 1

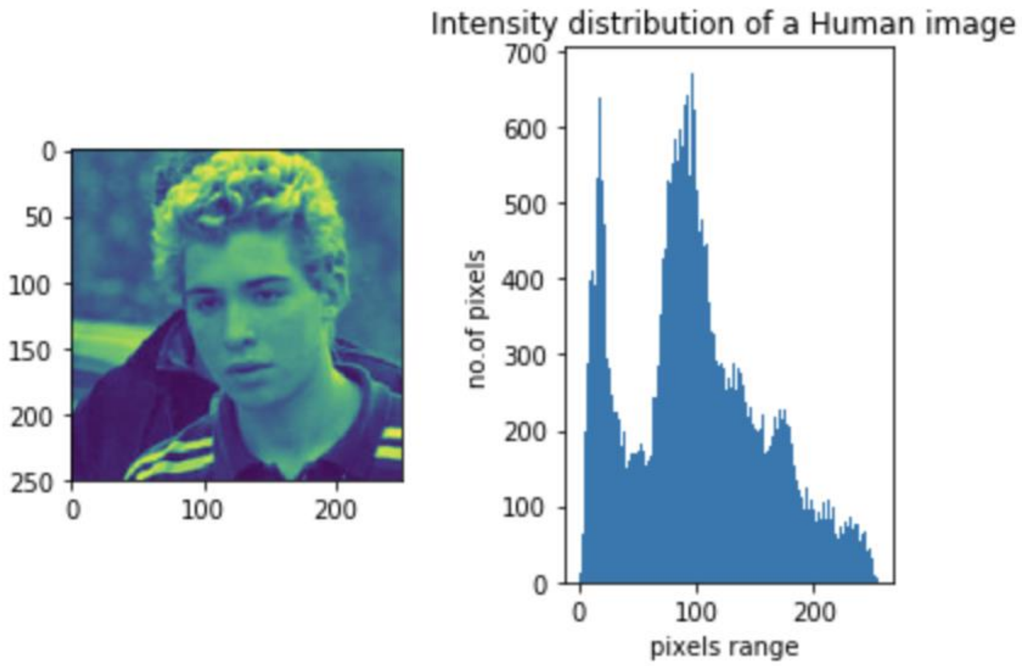

Intensity distribution of a Human image
no.of pixels
pixels range

Fig 2

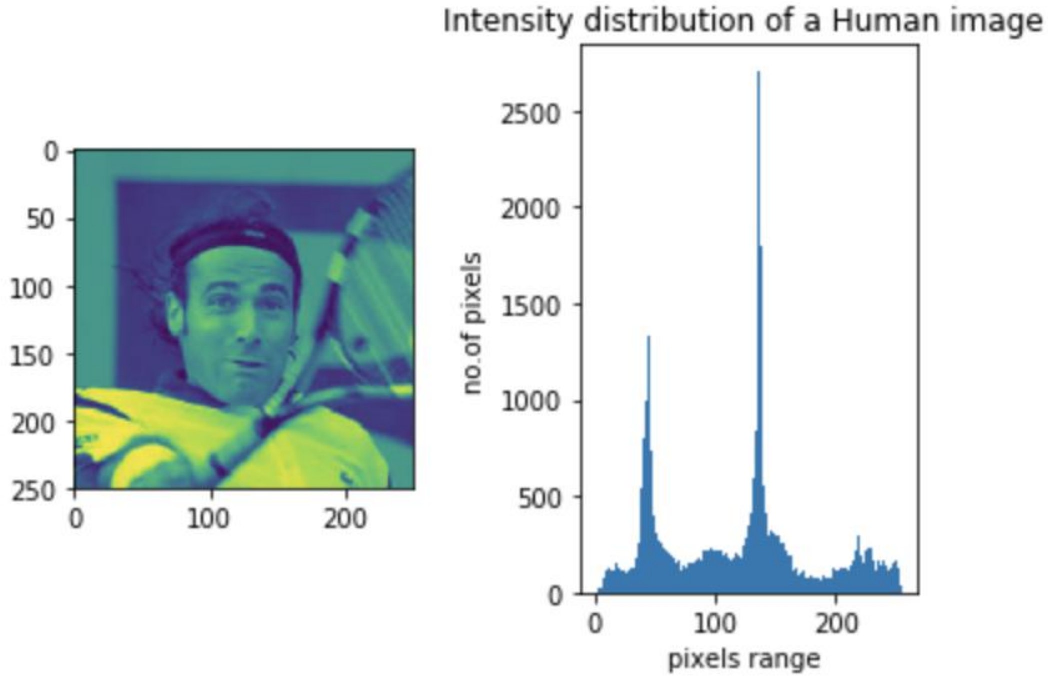

Intensity distribution of a Human image
no.of pixels
pixels range

Fig 3

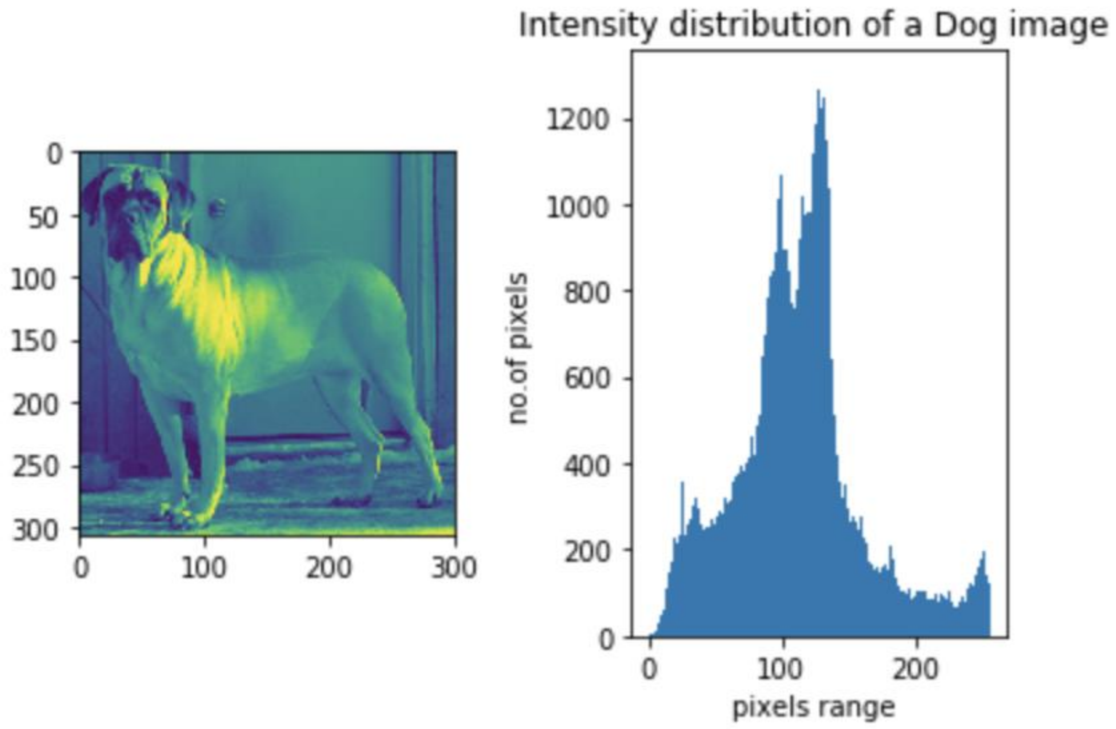
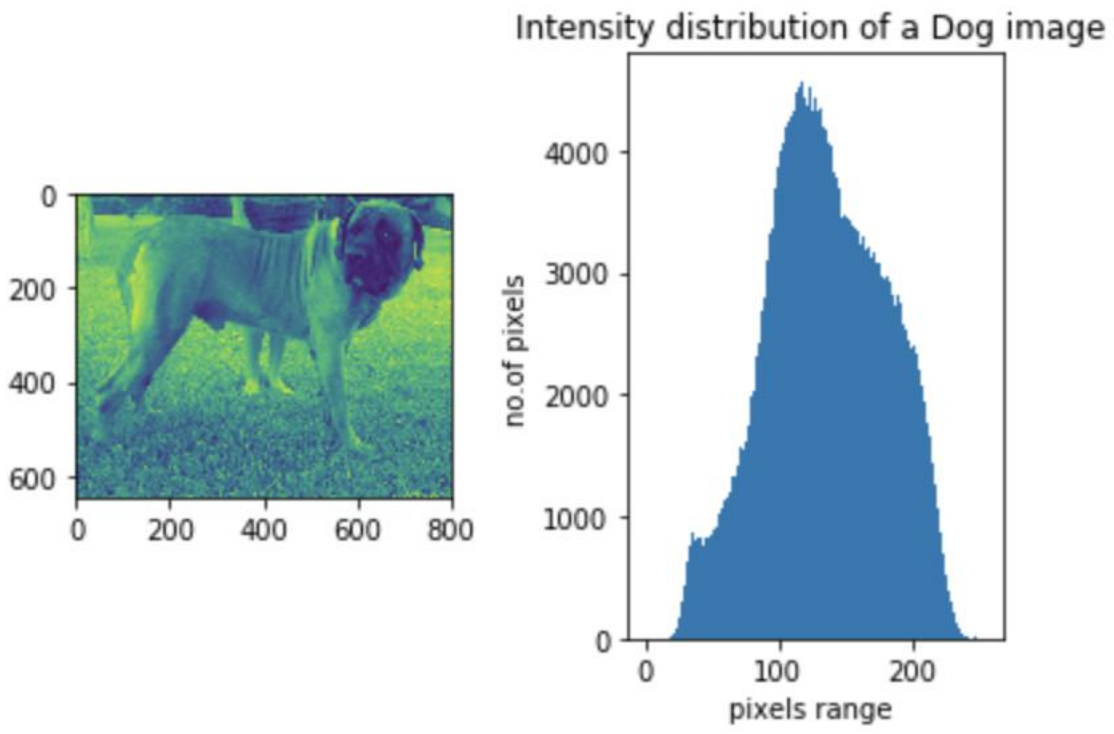
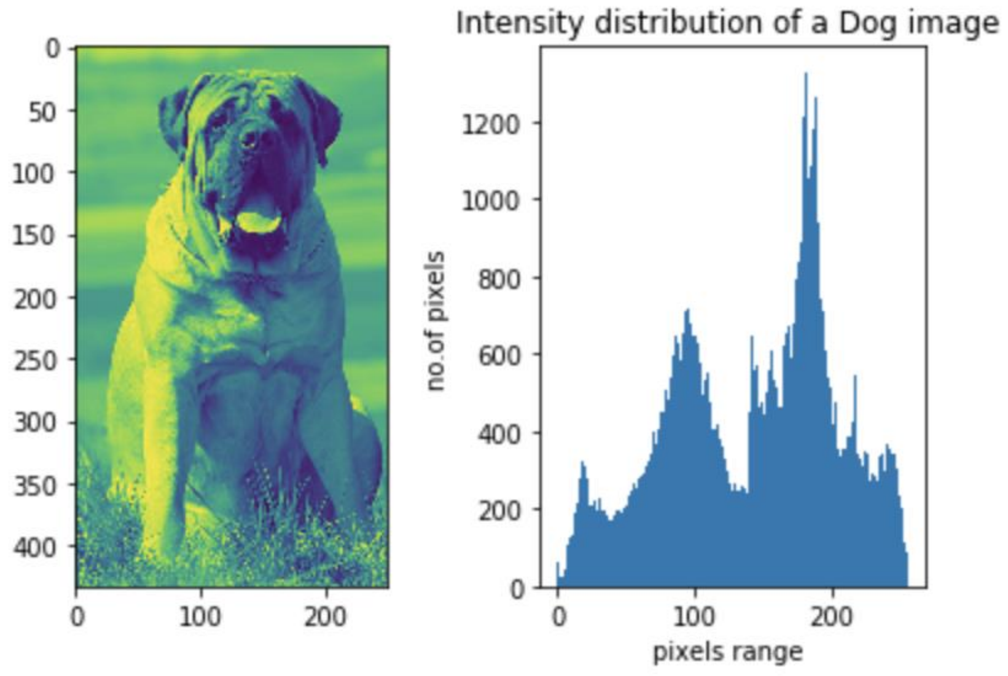

Intensity distribution of a Dog image
no.of pixels
pixels range
0
50
100
150
200
250
300
0
100
200
300
0
200
400
600
800
1000
1200
0
100
200

Fig4



Intensity distribution of a Dog image
no.of pixels
pixels range

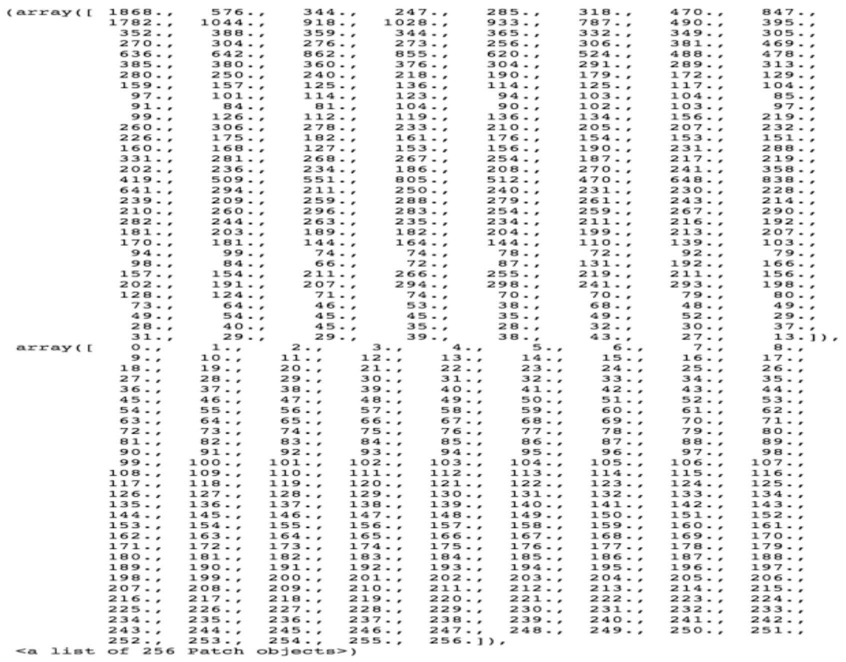
Fig 5



Intensity distribution of a Dog image
no.of pixels
pixels range

Fig 6

Algorithms and Techniques
Computer sees an image as an array of pixel.
For example, the Fig 1 translates as following for a computer



```
(array([ 1868.,   576.,   344.,   247.,   285.,   318.,   470.,   847.,
         1782.,  1044.,   918.,  1028.,   933.,   787.,   490.,   395.,
          352.,   388.,   359.,   344.,   365.,   332.,   349.,   305.,
          270.,   304.,   276.,   273.,   256.,   306.,   381.,   469.,
          636.,   642.,   862.,   855.,   620.,   524.,   488.,   478.,
          385.,   380.,   360.,   376.,   304.,   291.,   289.,   313.,
          280.,   250.,   240.,   218.,   190.,   179.,   172.,   129.,
          159.,   157.,   125.,   136.,   114.,   125.,   117.,   104.,
           97.,   101.,   114.,   123.,    94.,   103.,   104.,    85.,
           91.,    84.,    81.,   104.,    90.,   102.,   103.,    97.,
           99.,   126.,   112.,   119.,   136.,   134.,   156.,   219.,
          260.,   306.,   278.,   233.,   210.,   205.,   207.,   232.,
          226.,   175.,   182.,   161.,   176.,   154.,   153.,   151.,
          160.,   168.,   127.,   153.,   156.,   190.,   231.,   288.,
          331.,   281.,   268.,   267.,   254.,   187.,   217.,   219.,
          202.,   236.,   234.,   186.,   208.,   270.,   241.,   358.,
          419.,   509.,   551.,   805.,   512.,   470.,   648.,   838.,
          641.,   294.,   211.,   250.,   240.,   231.,   230.,   228.,
          239.,   209.,   259.,   288.,   279.,   261.,   243.,   214.,
          210.,   260.,   296.,   283.,   254.,   259.,   267.,   290.,
          282.,   244.,   263.,   235.,   234.,   211.,   216.,   192.,
          181.,   203.,   189.,   182.,   204.,   199.,   213.,   207.,
          170.,   181.,   144.,   164.,   144.,   110.,   139.,   103.,
           94.,    99.,    74.,    74.,    78.,    72.,    92.,    79.,
           98.,    84.,    66.,    72.,    87.,   131.,   192.,   166.,
          157.,   154.,   211.,   266.,   255.,   219.,   211.,   156.,
          202.,   191.,   207.,   294.,   298.,   241.,   293.,   198.,
          128.,   124.,    71.,    74.,    70.,    70.,    79.,    80.,
           73.,    64.,    46.,    53.,    38.,    68.,    48.,    49.,
           49.,    54.,    45.,    45.,    35.,    49.,    52.,    29.,
           28.,    40.,    45.,    35.,    28.,    32.,    30.,    37.,
           31.,    29.,    29.,    39.,    38.,    43.,    27.,    13.]),
 array([   0.,    1.,    2.,    3.,    4.,    5.,    6.,    7.,    8.,
           9.,   10.,   11.,   12.,   13.,   14.,   15.,   16.,   17.,
          18.,   19.,   20.,   21.,   22.,   23.,   24.,   25.,   26.,
          27.,   28.,   29.,   30.,   31.,   32.,   33.,   34.,   35.,
          36.,   37.,   38.,   39.,   40.,   41.,   42.,   43.,   44.,
          45.,   46.,   47.,   48.,   49.,   50.,   51.,   52.,   53.,
          54.,   55.,   56.,   57.,   58.,   59.,   60.,   61.,   62.,
          63.,   64.,   65.,   66.,   67.,   68.,   69.,   70.,   71.,
          72.,   73.,   74.,   75.,   76.,   77.,   78.,   79.,   80.,
          81.,   82.,   83.,   84.,   85.,   86.,   87.,   88.,   89.,
          90.,   91.,   92.,   93.,   94.,   95.,   96.,   97.,   98.,
          99.,  100.,  101.,  102.,  103.,  104.,  105.,  106.,  107.,
         108.,  109.,  110.,  111.,  112.,  113.,  114.,  115.,  116.,
         117.,  118.,  119.,  120.,  121.,  122.,  123.,  124.,  125.,
         126.,  127.,  128.,  129.,  130.,  131.,  132.,  133.,  134.,
         135.,  136.,  137.,  138.,  139.,  140.,  141.,  142.,  143.,
         144.,  145.,  146.,  147.,  148.,  149.,  150.,  151.,  152.,
         153.,  154.,  155.,  156.,  157.,  158.,  159.,  160.,  161.,
         162.,  163.,  164.,  165.,  166.,  167.,  168.,  169.,  170.,
         171.,  172.,  173.,  174.,  175.,  176.,  177.,  178.,  179.,
         180.,  181.,  182.,  183.,  184.,  185.,  186.,  187.,  188.,
         189.,  190.,  191.,  192.,  193.,  194.,  195.,  196.,  197.,
         198.,  199.,  200.,  201.,  202.,  203.,  204.,  205.,  206.,
         207.,  208.,  209.,  210.,  211.,  212.,  213.,  214.,  215.,
         216.,  217.,  218.,  219.,  220.,  221.,  222.,  223.,  224.,
         225.,  226.,  227.,  228.,  229.,  230.,  231.,  232.,  233.,
         234.,  235.,  236.,  237.,  238.,  239.,  240.,  241.,  242.,
         243.,  244.,  245.,  246.,  247.,  248.,  249.,  250.,  251.,
         252.,  253.,  254.,  255.,  256.]),
 <a list of 256 Patch objects>)
```

Fig 7

For each point along the height and width of the image, the array contains its intensity which is a value between 0 and 255.
CNN architecture –
Convolutional neural network consists of four parts –

1. Convolutional Layer
2. The Non linear layer
3. Pooling Layer
4. Fully Connected Layer

Convolutional Layer - Any CNN model consists of a number of these layers.

The input matrix in the Fig 7 is passed into the first convolutional layer. Computer software selects a smaller matrix, say for example a 3*3 matrix and applies it over the input matrix. The algorithm involves dot product of the filter matrix and keeps moving forward step by step (based on the value of stride) at a time till the original matrix is covered. In this way, a smaller output matrix is produced which is then passed on to second convolution layer and so on.
In this manner, feature extraction is done.

Non Linear Layer - After each convolution layer, an activation function is applied, to ensure that the image has the non linearity property intact.

Pooling Layer – The aim of the pooling layer is to downsample the image, so that the already extracted features are taken out of further processing.

Fully Connected Layer – The output of the convolutional layers goes as input into the FC layer which outputs the result as a N dimensional array where N is the number of classification categories present.

Strides - Stride is the number of shifts over the input matrix. When the stride is 1 then we move the filters by one step at a time. When the stride is 2 then we move the filters 2 steps at a time and so on.

Padding – Sometimes, it is possible that while traversing the input matrix, filter does not fit perfectly on the input image. In that case we have two options:

- Pad the picture with zeros (zero-padding) so that the filter fits on the input matrix.
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image

Transfer Learning - Transfer learning (TL) is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. In this case, I used the VGG 16 model for breed classification.

Benchmark Model:

The VGG16 model is considered to be one of the excellent vision model architecture till date. VGG16 is a convolution neural net (CNN) architecture which instead of having a large number of hyper-parameter focusses on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

Data Preprocessing:

As part of data preprocessing, I have resized the image to 256 pixels and then cropped the image to 224 pixels after which I have performed horizontal flip to augment the data. Every pixel is also rescaled by dividing it by 252 with the help of transform.ToTensor() method.

Implementation:

1. Human Detection – I have used OpenCV's implementation of Haar feature-based cascade classifier to detect human faces in images. First a BGR image is coverted into a gray scale image and after that a face classifier is applied.

2. Dog Detection- It is done with the help of a pre-trained model of VGG-16 where it predicts an input image as a dog if index value is between 151 and 268(inclusive).

3. Dog Breed Classification

This is done in two ways, one from scratch and another using the pre-trained model. The two ways are described below:-

- I have used 5 convolution layers with all convolution having kernel size = 3, stride =1; padding =1. Also max pooling of 2*2 is applied to reduce the spatial size of the feature set. Batch normalization is applied after each max pooling so that the model is stabilised because of the normalisation of the output produced by the previous layer. RELU activation function is applied after each convolution layer so that the non linearity is still maintained. Layer 1: (3,16) input channels =3 , output channels = 16 Layer 2: (16,32) input channels = 16 , output channels = 32 Layer 3: (32,64) input channels =32 , output channels = 64 Layer 4: (64,128) input channels =64 , output channels = 128 Layer 5: (128,256) input channels =128 , output channels = 256 after this there is a fully connected layer with input = 9216 and output = 133 i.e equal to the number of dog breeds. It takes the input of the final convolution layer and classifies the image into one of the types(in this case the dog breeds.

- Another model is VGG16 using transfer learning.

4. Final app algorithm

Final algorithm is written where if a human is detected, it classifies it as human and if a dog is detected his breed is shown.

Refinement:

It was difficult to get the right combination of convolution layers and the parameters involved to get the desired result. I tried to understand the VGG model and the role of different layers involved and then tweaked the parameters a number of times to get the desired output. Also, while training and validating the model, it was crucial to get the number of epochs right. Initially, having a higher epoch led to a very long processing time which was practically infeasible. Very less epoch time resulted in poor accuracy. Finally, after a few attempts I settled to a value of 10 for epoch.

Model Evaluation and Validation: Going back to the simple OpenCV human face detector, it performed quite well. The human face detector was able to identify 98% of all humans, however, it also found that 17% of the dogs had human faces. The dog detector was able to confirm that 99% of the dogs were dogs and identified 2% of the humans as dogs. With the dog breed classification model model, the testing accuracy was 13% and with the VGG16 model used in transfer learning the accuracy achieved was of 86%.

Justification: The 11% accuracy in case of the CNN model implemented from scratch is due to the less number of convolutional layers and fully connected layers. The model is similar to VGG16 model, and when VGG 16 model is used which has 16 convolutional layers and 2 fully connected layers, the model yields 86% accuracy.

Free-Form Visualization: The two important aspects of the project are the detection algorithm and the data set. While it is important to have a good algorithm, it is also very crucial to have a large dataset to avoid the problem of underfitting. Also, a good algorithm is one which is generalized and doesnot over fit the data. Both underfitting and overfitting can result in a poor performing model on new data. I believe that the model that I developed from scratch underfitted the data, hence there was only 11% percent accuracy. Also, the VGG 16 model performed with an accuracy of 86% and hence can be termed as a generalized model. Suppose that the number of convolutional layers is increased further beyond 16 with an aim to increase its performance on training data. The result could be inverse as the model may overfit the training data and perform poorly on new datasets because of lack of generalization. Therefore, it is very important to understand the appropriate algorithm and also allow the algorithm to train on sufficient data to ensure good accuracy.

Reflection: In this project, starting from face detection with the help of opencv haarcascade classifier to developing CNN algorithm for breed classification, everything was covered. It was particularly challenging to develop and write CNN model from scratch and get desired accuracy. It required a firm understanding of the maths involved in the calculation of the parameters involved in the convolution layers and the FC layer. Further, with the loss function and optimizer function, I got an insight into the performance of the model at each iteration. Also, the concept of transfer learning was quite new to me. After creating a model from scratch, I realized that it was difficult to train the model every time as it was pretty time consuming. There was a significant improvement in performance as well, when I used the pretrained VGG 16 model because of both the algorithm of the VGG 16 model and the fact that it was a pretrained model. Comparison of the training, validation loss on each iteration and testing accuracy of this model with that of the model I built from scratch made clear the scope of improvement in my model.

Improvement:
1. When the model is built from scratch using CNN classifier is 13% .
2. When the model is built using transfer learning the test accuracy is 86%.
Following improvements can be made in the implementation:

1. the depth in my model was quite less. For better results the depth of convolution layers can be increased.
2. There can be more fully connected layers so that the model fits better on training data and naturally gives more accurate results.
3. Then number of epochs can be increased.
4. The algorithm can be made better for identifying multiple dogs and humans in a single image.