Capstone Project **Rakesh Kumar**

Machine Learning Engineer Nanodegree **23-09-2019**

Project Report
On

Dog Breed Classification

Project Overview:- In this project, we will learn how to build a pipeline to process real-world, user-supplied images. This is basic and typical machine learning classification problem where we will classify 133 categories of dogs. To solve problems related to images with classifications, neural network is the best approach because it works better in classifying images of multiple categories. To check accuracy of model we will use percentage of how many images classified correctly out of total images. Given an image of a dog, our algorithm will identify an estimate of the dog breed and if an image is of a human, the code will identify the resembling dog breed. We will explore the state-of-the-art CNN models and transfer learning for classification. As part of the process, I also created a function to detect if an image was a human face or a dog. I utilized the Udacity training space with pytorch which was GPU-enabled to allow for the models to run quickly. Cats and dogs are often the subject of projects, but the technology can be applied to identifying letters, faces, and tumors. While this technology often finds itself as part of a fictional plot, it has just as many real uses than it does fantasy ones. In this project, we will make the steps towards developing an algorithm that could be used as part of a mobile or web app.

Problem Statement:- Our goal is that by completing this project, we will understand the challenges involved in piecing together a series of models designed to perform various tasks in a data processing pipeline. Each model has its strengths and weaknesses, and engineering a real-world application often involves solving many problems without a perfect answer. Since there are many algorithms available but we will try to use pre-trained and light model, so that whenever this model will be deployed as an app it will work fast and accurately. The purpose of the project is to use a convolutional neural network (CNN) to distinguish dog breeds. First, I attempted to build a CNN from scratch but the results were poor (26% accuracy). This was an expected result because image recognition requires more complex feature detection.

A transfer learning approach, was a much more accurate approach. Transfer learning focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. In transfer learning, you reuse the feature extraction part and re-train the classification part. Since the feature extraction process is the most complex modeling challenge, reusing it allows you to train a new model with less computational resources and training time. The end goal of the project is to provide a predicted breed for dog images and for humans, the function will tell you which dog the human most resembles.

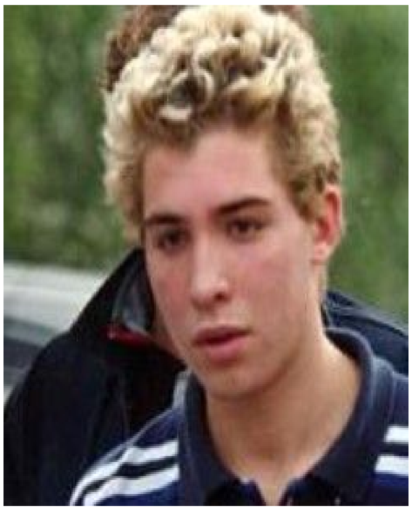
Metrics:- Percentage of total items classified correctly,

True Positive/total no. of images

The models were evaluated using an accuracy metric utilizing a test set. The human and dog detectors were tested on 100 images of each. There are 6680 training dog images, 835 validation dog images and 836 test dog images.
Since we are just checking how many out of total images is classified correctly. Because we are also using pre-trained model where dog categories lies between index 151 and 268. So if index is between given index then count 1 otherwise count 0 towards total number of correctly classified. Same is for human face detection but without any index. Because for human face model is purely trained on human faces.

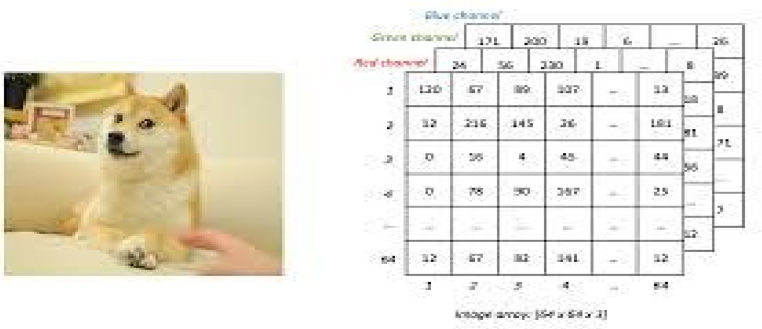
Datasets and Inputs:- Since this project is available in udacity workspace, so dataset is available there, but if someone want to do on their own they are providing link for download. Dataset contains 13233 human and 8351 dog images. And in both dataset there are images for training, validation, and testing. These images contains required data for better training and testing of model. Datasets contains almost every type of images to learn and detect dog image and human image. The total categories of dog images is 133. Data in dataset is imbalanced, But there is not much difference between total count of every image. Since we need to classify images of dogs as well as humans because we have to specify resembled category of dog, so we need human images as well.
Since we have to classify images we train the model with images of dog and human. So we need lots of images for better training, that our model can generalize well.

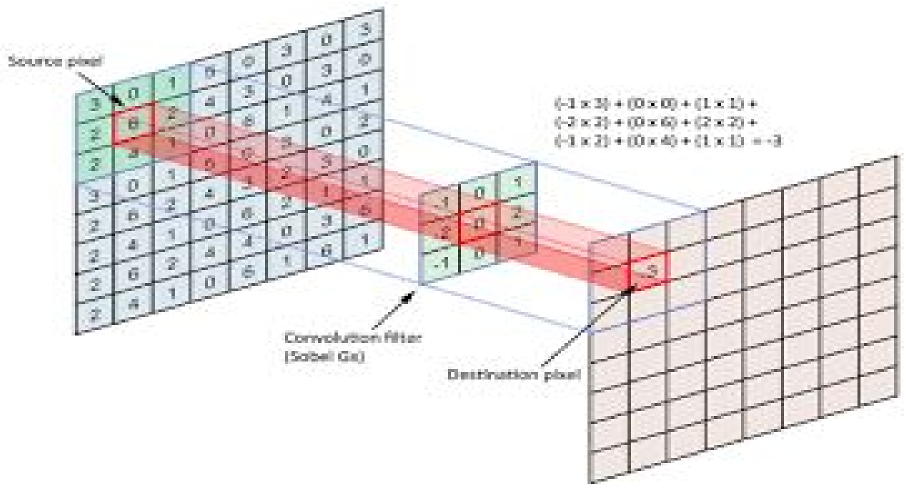
Exploratory Visualization:- For exploratory visualization below are the different samples of dataset. As we can see the first two images are identical but are of different categories.

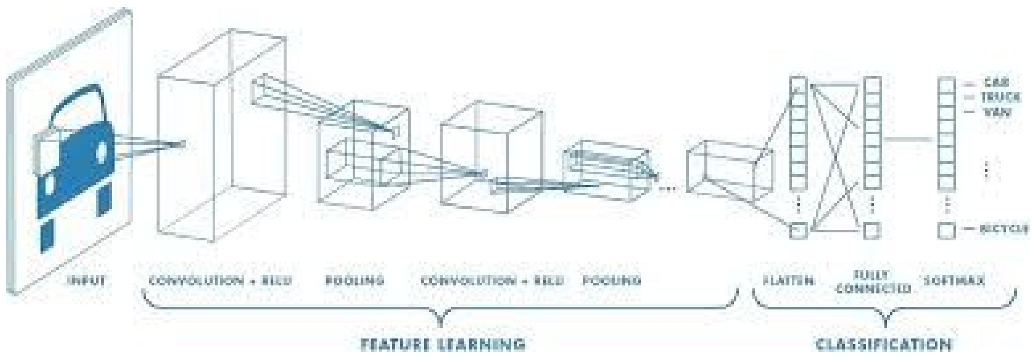


Each image has different dog with different positions at different angles and different colour. We need these types of images because during training model has to learn about how to detect dog in every angle of image with so much difference or with so minor difference. In human dataset as well we different types of images of human as we can see one is smiling and the other one is playing tennis.

Algorithms and Techniques:- Since this is Classification problem and so it’s come under supervised machine learning problem. And for image classification neural network is the best option. Since other algorithms can also do good but neural is best among them. And in this project I am using convolution neural network with transfer learning. Object recognition and classification using convolutional neural networks (CNN)s has been the topic of many research projects in recent years. The popularity of CNNs can be attributed to the fact that they are capable of recognizing and classifying a wide variety of objects and images. A lot of research has gone in to the training of CNNs on a variety of datasets, most focusing on the optimization of a particular network’s performance. Such research has extended into what is known as fine-grained classification tasks. In these problems, the dataset used contains classes which can have minor, or few, differences between them. Additionally, such datasets may also contain classes which vary in a number of different ways, within themselves. Convolutional neural networks (CNN) have been used to great effect in applications such as object classification, scene recognition, and other applications. In many situations, we can imagine the features that are learned by the CNNs in the process of training. However, when the objects the CNNs are trying to categorize share many similar features, such as the breeds of dogs, it becomes hard to imagine the specific features that CNNs must learn in order to categorize these dogs correctly. In the current age, when nearly everyone carries a high-resolution camera in their pockets, such systems could be of great use in recognizing dog or other animal breeds, and even plant species from photos taken in real time and uploaded as queries. In this research, we will be training CNNs on the Dogs and human Dataset, using transfer learning. However, the focus of this project is to analyze the features used by the trained CNNs. Convolutional Neural Networks CNNs are based on a number of concepts within image processing and computer vision. As the name suggests, convolution is a major component to neural networks. Convolution Layer The convolution (CONV) layer takes matrices, or images, which are just matrices of integer values. Then a feature matrix, convolves the input image, creating a feature map. There are multiple feature matrices, or detectors. Additionally, there are usually multiple CONV layers in a single network. An example of a simplified convolution (showing a single filter) can be seen in figures.



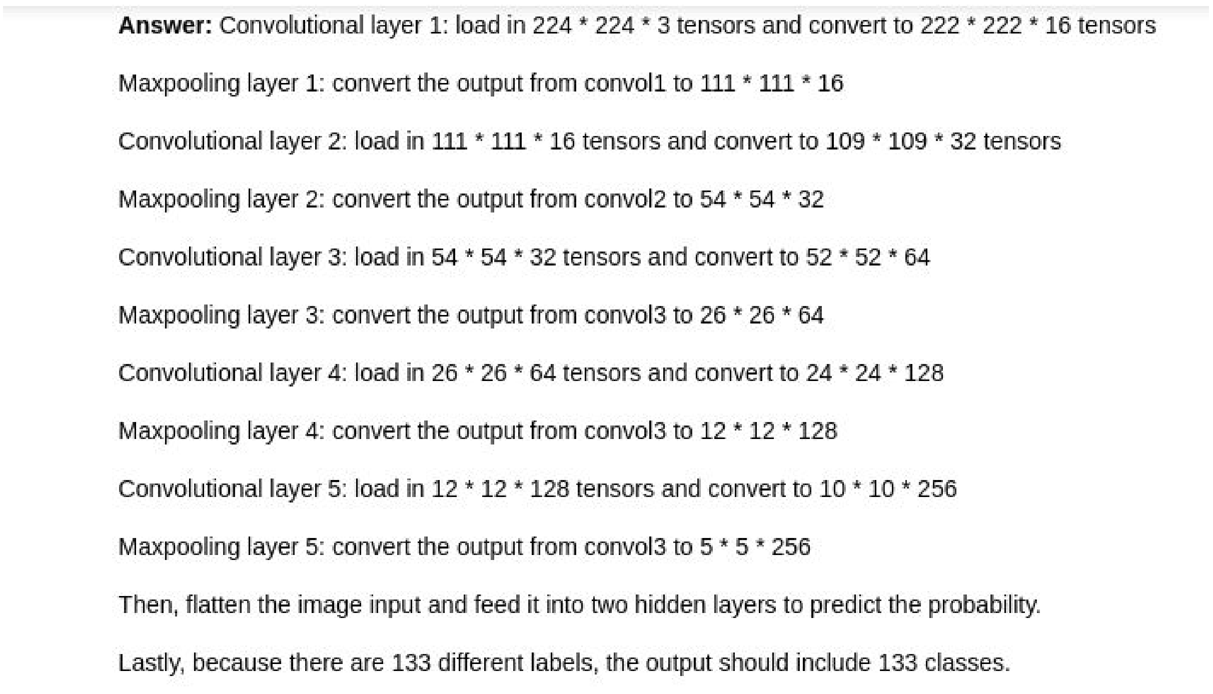

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel



Benchmark Model:- For benchmark model we will use OpenCV algorithm to detect human faces. For this we will use some images from the dataset for detection and then check how many will it detect exactly out of total. OpenCV provides many pre-trained face detectors which are very useful for benchmark of any particular model. And for dog images we will use VGG16, which is also trained on a very large dataset of images for classification and other vision tasks. So it also helps us in initial phase of development.

Data Preprocessing:- Since images in dataset are not of one dimension and we need to preprocess them, so that our model can process them efficiently and output don’t affect by size or dimensions. So in this project for training we resize each image to 256 px and convert to 224*224 px using centercrop. Then normalize them using standard deviation and mean. We also use horizontal flip to flip horizontally which gives us varieties of each image and provide one more dimension of image type for neural network to learn. And since we working on pytorch we need to convert them into tensors.
For validation and testing i perform less preprocessing. I just resize each image to 224*224 px and normalize. And convert them into tensors for further use. Since for training we need to do a lot of preprocessing for better training, so that our model can learn better. But in case of validation and testing we just check how well the model is performing on random data. Therefore we just do little preprocessing and generalize on that data. Because end user can feed any type of image of any dimension.

Implementation:- Now we will use Convolution Neural Network. The first trial in building this model is building a CNN from scratch, without the use of transfer learning. Based on what we know about the purpose of transfer learning, it is likely that our results will be poor. We will define convolution layers and we will apply maxpooling at each layer. And activation function will be ReLU function. At the end we will flatten the input. For loss function we will use cross-entropy, because Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1.
For Optimizer we will use Adam Optimizer, because adam is an adaptive learning rate method, it computes individual learning rates for different parameters. After that we will define training methods to train our model with some number of epochs and calculate training loss and validation loss. Then for testing we will define test method and calculate test loss and accuracy.
For the basic CNN, I chose the following architecture:-



Answer: Convolutional layer 1: load in 224 * 224 * 3 tensors and convert to 222 * 222 * 16 tensors

Maxpooling layer 1: convert the output from convol1 to 111 * 111 * 16

Convolutional layer 2: load in 111 * 111 * 16 tensors and convert to 109 * 109 * 32 tensors

Maxpooling layer 2: convert the output from convol2 to 54 * 54 * 32

Convolutional layer 3: load in 54 * 54 * 32 tensors and convert to 52 * 52 * 64

Maxpooling layer 3: convert the output from convol3 to 26 * 26 * 64

Convolutional layer 4: load in 26 * 26 * 64 tensors and convert to 24 * 24 * 128

Maxpooling layer 4: convert the output from convol3 to 12 * 12 * 128

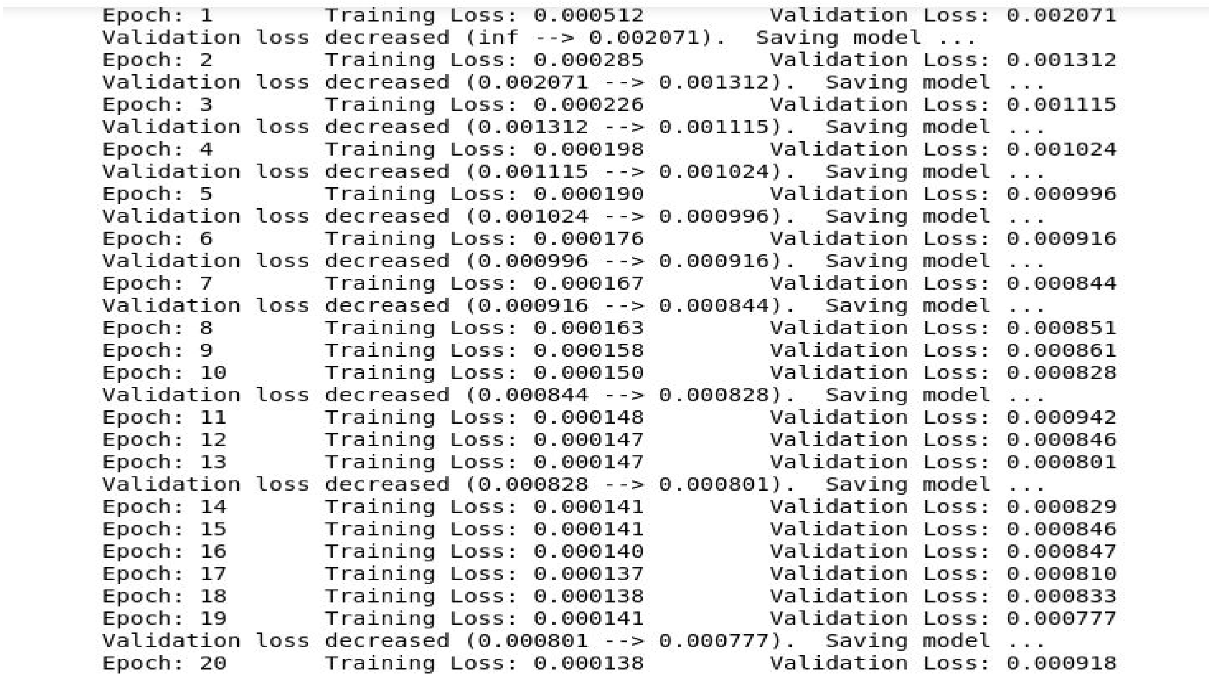
Convolutional layer 5: load in 12 * 12 * 128 tensors and convert to 10 * 10 * 256

Maxpooling layer 5: convert the output from convol3 to 5 * 5 * 256

Then, flatten the image input and feed it into two hidden layers to predict the probability.

Lastly, because there are 133 different labels, the output should include 133 classes.

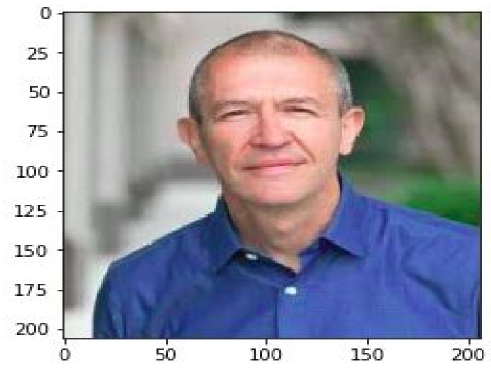
Refinement:- For better prediction of the images we will also use transfer learning. Transfer learning make use of the knowledge gained while solving one problem and applying it to a different but related problem. Because in this project we will also predict resembling dog breed if a human is predicted. For that we will use Resnet18 pre-trained model. For this part we will use same optimizer and loss function as in convolution network. Then we will use already defined training and testing methods. After testing data on test images I got 78% accuracy which is good right now.



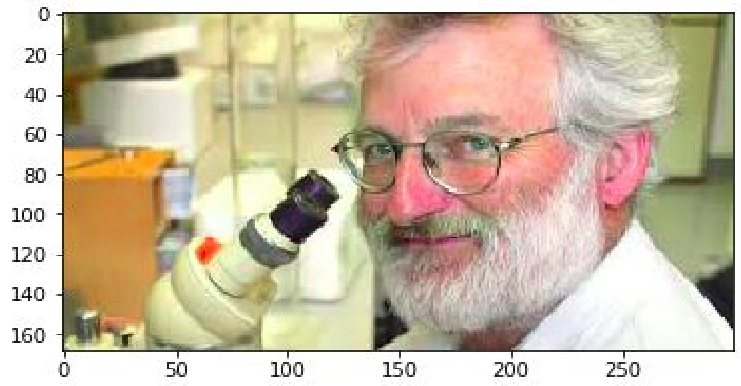
```
Epoch: 1        Training Loss: 0.000512         Validation Loss: 0.002071
Validation loss decreased (inf --> 0.002071).  Saving model ...
Epoch: 2        Training Loss: 0.000285         Validation Loss: 0.001312
Validation loss decreased (0.002071 --> 0.001312).  Saving model ...
Epoch: 3        Training Loss: 0.000226         Validation Loss: 0.001115
Validation loss decreased (0.001312 --> 0.001115).  Saving model ...
Epoch: 4        Training Loss: 0.000198         Validation Loss: 0.001024
Validation loss decreased (0.001115 --> 0.001024).  Saving model ...
Epoch: 5        Training Loss: 0.000190         Validation Loss: 0.000996
Validation loss decreased (0.001024 --> 0.000996).  Saving model ...
Epoch: 6        Training Loss: 0.000176         Validation Loss: 0.000916
Validation loss decreased (0.000996 --> 0.000916).  Saving model ...
Epoch: 7        Training Loss: 0.000167         Validation Loss: 0.000844
Validation loss decreased (0.000916 --> 0.000844).  Saving model ...
Epoch: 8        Training Loss: 0.000163         Validation Loss: 0.000851
Epoch: 9        Training Loss: 0.000158         Validation Loss: 0.000861
Epoch: 10       Training Loss: 0.000150         Validation Loss: 0.000828
Validation loss decreased (0.000844 --> 0.000828).  Saving model ...
Epoch: 11       Training Loss: 0.000148         Validation Loss: 0.000942
Epoch: 12       Training Loss: 0.000147         Validation Loss: 0.000846
Epoch: 13       Training Loss: 0.000147         Validation Loss: 0.000801
Validation loss decreased (0.000828 --> 0.000801).  Saving model ...
Epoch: 14       Training Loss: 0.000141         Validation Loss: 0.000829
Epoch: 15       Training Loss: 0.000141         Validation Loss: 0.000846
Epoch: 16       Training Loss: 0.000140         Validation Loss: 0.000847
Epoch: 17       Training Loss: 0.000137         Validation Loss: 0.000810
Epoch: 18       Training Loss: 0.000138         Validation Loss: 0.000833
Epoch: 19       Training Loss: 0.000141         Validation Loss: 0.000777
Validation loss decreased (0.000801 --> 0.000777).  Saving model ...
Epoch: 20       Training Loss: 0.000138         Validation Loss: 0.000918
```

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 100% of the dogs were dogs and correctly confirmed that 0% of the humans were dogs. Below there is a human face which model does not recognizing neither dog nor human. But overall model is performing better.

```
Dogs Detected!
It looks like a Pekingese
```

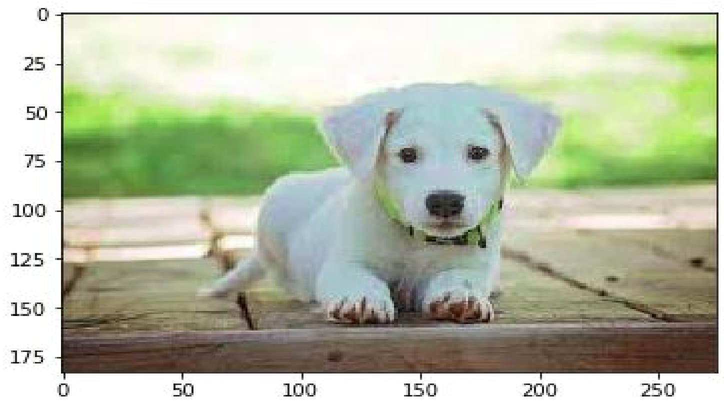


```
Hello, human!
If you were a dog..You may look like a Irish water spaniel
```

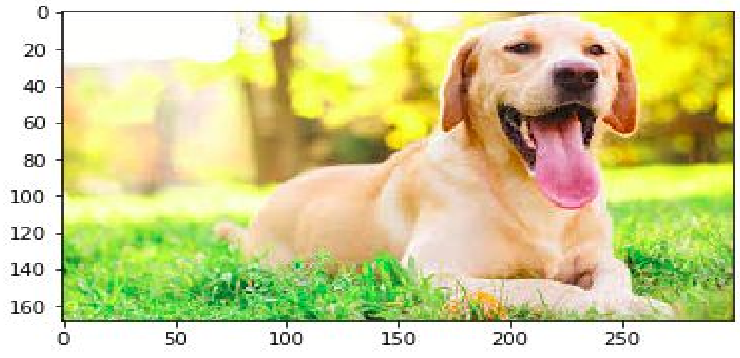


```
Error! Can't detect anything..
```

```
Hello, human!
If you were a dog..You may look like a American staffordshire terrier
```



```
Dogs Detected!
It looks like a Komondor
```



```
Dogs Detected!
It looks like a Kuvasz
```

Justification:- Since it performing better in recognizing dog images better and also good in detecting human images as it is not detecting human face. This is because in our benchmark model it performed 100% accurately in detecting dog images but it perform 98% accurately on human face detection. Therefore it is performing the same as our benchmark model which is good because it performing as we expect.

Improvement:- I was not able to improve above 80%. I could have augmented the dataset to improve its ability to generalize. I also could try the other pre-trained models that were available in the project such as Xception, Resnet 50, VGG19, etc. Or if we add more images in datasets then it will better for model to learn more from them and generalize well. Adding more will increase variation in dataset. The pre-trained networks will continue to evolve and make recognition a ‘plug-and-play’ aspect of our everyday life. I think the biggest challenge is the accuracy. I worked hard to get above 80 and it wasn’t possible given the work I did here. That is WAY too low to be used in everyday life. We can’t make visual mistakes one out of every 5 times we look at something. Given more time, and possibly more motivation, I would want to improve the training set and the model to get 99+% accuracy.