Machine learning Engineer nanodegree

Capstone Project report

Dog Breed Classification with CNN

1.Definition

Project Overview:

As many people now days are seeking for buying a dog and are new to this business the first thing to know is the breed of the dogs in order to know which one will suit them the most and their environment and know more about their behavior therefor the overview of this project is to build a model which is used to identify the breed of a dog by just giving an image to the model as an input and it will classify it by extracting the features of the dog then detecting the breed of the dog . If the model is supplied with a human image it has to identify the resembling dog breed.

This model will be so useful for dog owners as it will prevent them from getting scammed and it will also save them lots of time and money for identifying the breed of their dog .

Problem Statement

Building a pipeline to process real world, user supplied images that can be used in an application .The model has to perform two tasks.

Human face detection : If the model is given a human image it has to identify the resembling dog breed.

Dog detection : when the model is given an image of a dog it has to detect it’s breed

Metrics :

The metrics that could be used in evaluating the performance of the model

- Accuracy : which is the number of correct specifiacations over all the specifications
- Recall: which is the ratio of the true positives over the true positives and the false negatives

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

True Positive + False Negative = Actual Positive

- Precision : The ratio of true positives over the true positives and the false positives

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

True Positive + False Positive = Total Predicted Positive

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

In my case I am going to use the accuracy as an evalution metric as after the visualization of the data I have observed that it is beyound still and is balanced to test the classification work I have also tried the f1 score as a metric and it’s output wasn’t promising at all in whi

2.Analysis

Data Exploration

The dataset provided by Udacity contains images of dogs and humans

The dog images datasets : (8151 image) of dogs with different sizes and backgrounds which are divided into training , testing and validation.

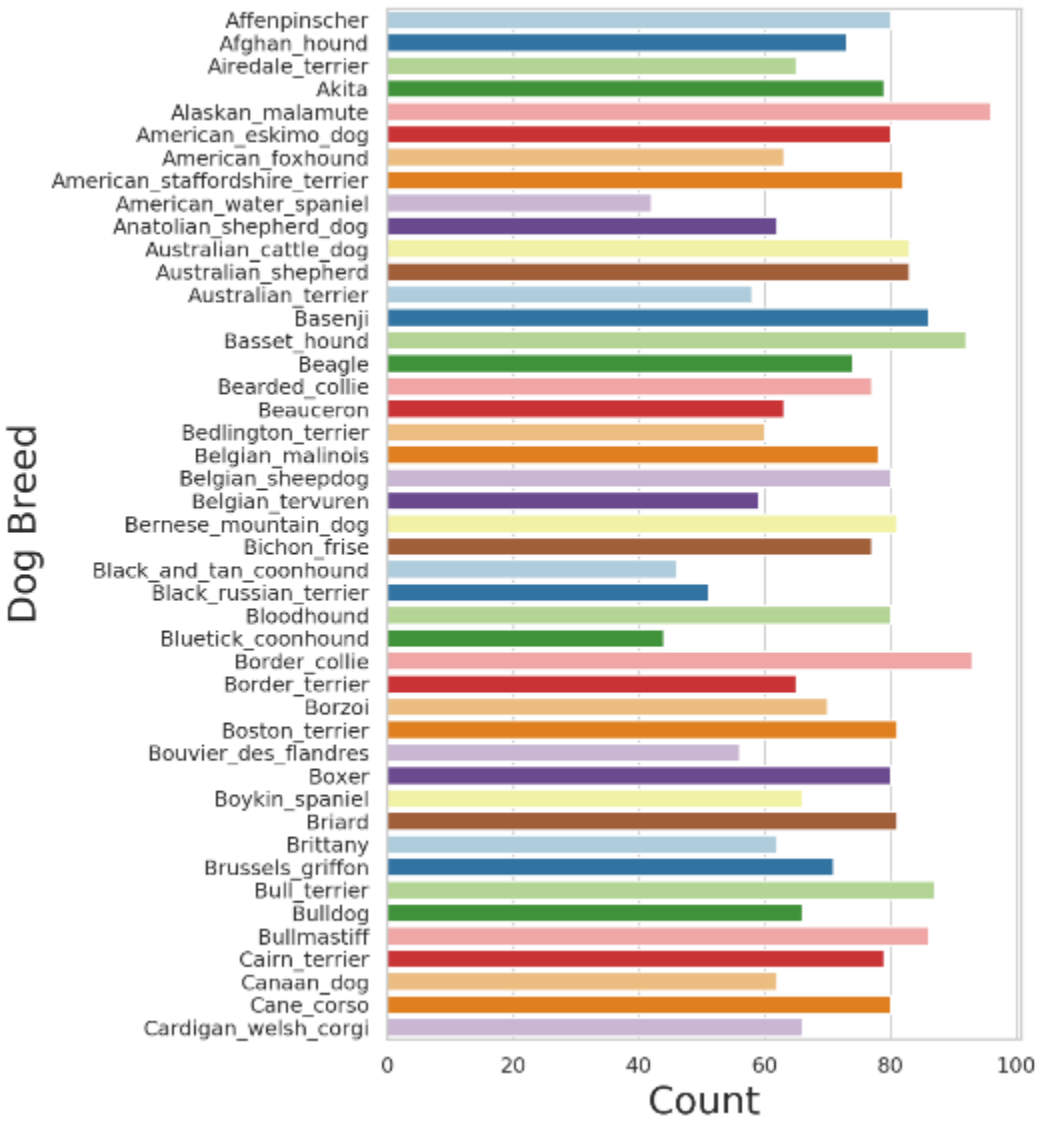
Training (6,680) image , testing(836) image and validation (835) image . The data is also divided to directories with 133 folder each . The number of images of each breed varies widely which makes the data unbalanced .

The humans images datasets : (13233) images of humans all with same size of 250X250 but the pictures are taken of different humans and different angles which also makes the data unbalanced.

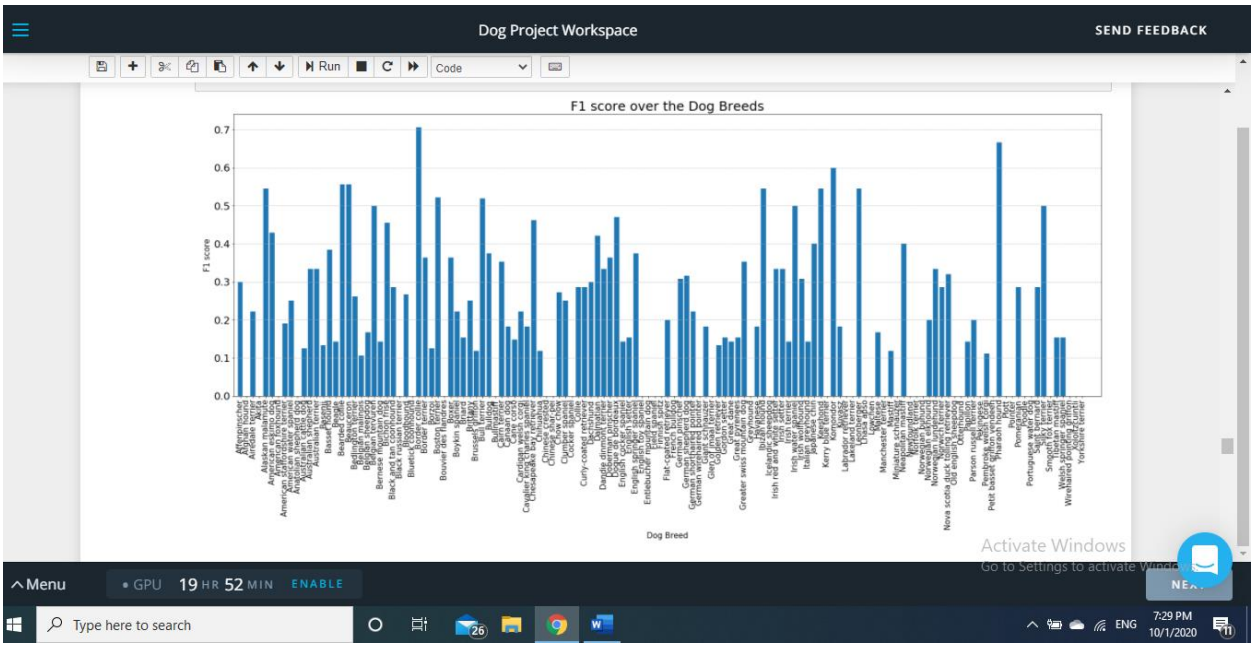
Exploratory Visualization

In this section I will show some plots to visualize the data and make it easier to understand

The first plot shows the number of each breed in the dataset



The plot below shows the f1 score to with relation with the dog breeds it shows that the border collie has the highest f1 score which means that it works best with the model



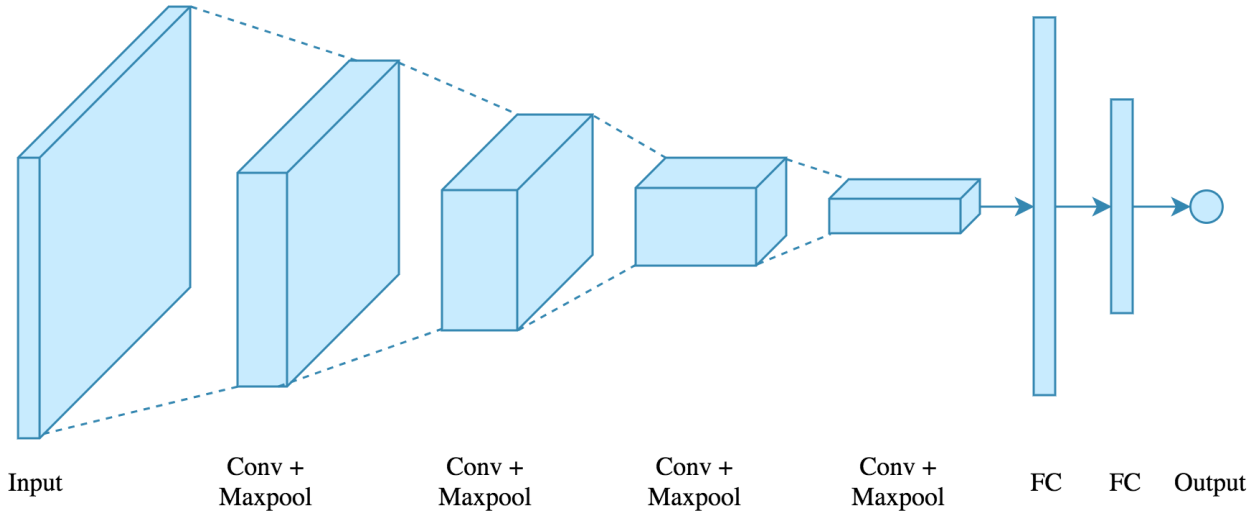
The images have different backgrounds

Algorithms and Techniques

As many of the algorithms that am using in building this model are based on the idea of CNN which is convolutional neural networks I will explain what it is pithiness , the CNN

is a neural network that is used mainly for image processing and classification and it contains one or more convolutional layers inspired by the neurons in the human brain and for these purposes I used it in the classification of the dog breed in my model .

The image below shows the architecture of the CNN model from scratch



Another algorithm which I used is the **Cascade Classifier** from OpenCV as it got so well training from positive and negative images for example images that includes faces and images that doesn't so that is why I used it to detect if an image includes a human or not and I implemented it using pytorch .

VGG-16 : I have used VGG-16 as it is known to be too great in detecting plenty of objects in a single image especially animals and that's the reason behind choosing it of dog

breed classification , The VGG-16 model is a CNN model which has 16 layers deep as overtime it became so rich in features extraction.

Transfer learning ResNet50: Resnet is a CNN model which consists of 50 layers deep and it is also very good in detecting objects as it can detect 1000 object in a single image .

Benchmark

I have set the benchmark of my CNN model built from scratch to have an accuracy of at least 10% and this proves that the model is working because random guess will roughly provide 1 out of 133 which corresponds to an accuracy of 1% so more than 10% proves the success of my model .

And the benchmark for the transfer learning model to be above 60%.

3.Methodology

Data Preprocessing

As the images were having different resolutions which made the data a little bit imbalance I used PyTorch module

torchvision.transforms and reduced the resolution to 224X224 then I applied normalization to the train , valid and test datasets and augmented the training data to reduce overfitting and underfitting .The training data was randomly rotated and randomly horizontally flipped then converted the images to tensor.

Implementation

Human face detector I have used the Cascade Classifiers to check whether a person's face is included in the image or not ,I have used the cascade classifier of haarcascade_fronalface_alt.xml file and implemented the face_detector function as it is able to recognize 100% of humans as humans**.**

Dog detector I used the pretrained VGG-16 model to be able to recognize dogs in the images as explained in the algorithms section I have implemented the model using PyTorch the the open source machine learning framework . I started by transforming the image of the model to 224X224 pixels which is the input size of the model then converted it to the pytorch tensor then normalized it with the same parameters as those of used when training the VGG-16 model then passed the image to the prediction to get the index value .

The dog breed classifier from scratch I have designed ,trained and tested the model using pytorch from scratch and I made it with only three convolutional neural networks to avoid complexity each with kernel size 3 ,padding 1 and stride 1 and then activated the relu function which outputs zero unless the input is positive it will return it directly and a pooling layer of kernel size 2 which reduces the amount of parameters and reduces the computational time of the network and a dropout with probability of 0.25 . the optimizer that I have used to reduce losses was the (SGD) which is the stochastic gradient descent optimizer as it is good with image classification the other hyperparameters that I used was setting the learning rate to 0.2 which worked fine with my model and set 15 epochs and the images below shows the training loss and validation loss progress over epochs .

```
Epoch 1, Batch 1 loss: 4.893456
Epoch 1, Batch 101 loss: 4.861359
Epoch 1, Batch 201 loss: 4.835721
Epoch 1, Batch 301 loss: 4.811738
Epoch: 1        Training Loss: 4.802269         Validation Loss: 4.699832
Validation loss decreased (inf --> 4.699832).  Saving model ...
Epoch 2, Batch 1 loss: 4.922375
Epoch 2, Batch 101 loss: 4.685390
Epoch 2, Batch 201 loss: 4.665530
Epoch 2, Batch 301 loss: 4.650681
Epoch: 2        Training Loss: 4.648099         Validation Loss: 4.554887
Validation loss decreased (4.699832 --> 4.554887).  Saving model ...
Epoch 3, Batch 1 loss: 4.537521
Epoch 3, Batch 101 loss: 4.592939
Epoch 3, Batch 201 loss: 4.555274
Epoch 3, Batch 301 loss: 4.545338
Epoch: 3        Training Loss: 4.543783         Validation Loss: 4.473463
Validation loss decreased (4.554887 --> 4.473463).  Saving model ...
Epoch 4, Batch 1 loss: 4.794885
Epoch 4, Batch 101 loss: 4.473694
Epoch 4, Batch 201 loss: 4.468399
Epoch 4, Batch 301 loss: 4.459466
Epoch: 4        Training Loss: 4.451112         Validation Loss: 4.352239
Validation loss decreased (4.473463 --> 4.352239).  Saving model ...
Epoch 5, Batch 1 loss: 4.268179
Epoch 5, Batch 101 loss: 4.392054
Epoch 5, Batch 201 loss: 4.394542
Epoch 5, Batch 301 loss: 4.377995
Epoch: 5        Training Loss: 4.377793         Validation Loss: 4.311018
Validation loss decreased (4.352239 --> 4.311018).  Saving model ...
Epoch 6, Batch 1 loss: 4.654929
Epoch 6, Batch 101 loss: 4.329735
Epoch 6, Batch 201 loss: 4.305745
Epoch 6, Batch 301 loss: 4.316312
Epoch: 6        Training Loss: 4.313151         Validation Loss: 4.261076
Validation loss decreased (4.311018 --> 4.261076).  Saving model ...
Epoch 7, Batch 1 loss: 4.534790
Epoch 7, Batch 101 loss: 4.283104
Epoch 7, Batch 201 loss: 4.269339
Epoch 7, Batch 301 loss: 4.264398
Epoch: 7        Training Loss: 4.261946         Validation Loss: 4.160961
Validation loss decreased (4.261076 --> 4.160961).  Saving model ...
Epoch 8, Batch 1 loss: 3.936312
```

```
Epoch 8, Batch 101 loss: 4.188739
Epoch 8, Batch 201 loss: 4.194172
Epoch 8, Batch 301 loss: 4.195583
Epoch: 8        Training Loss: 4.198020         Validation Loss: 4.239044
Epoch 9, Batch 1 loss: 4.345009
Epoch 9, Batch 101 loss: 4.114996
Epoch 9, Batch 201 loss: 4.129331
Epoch 9, Batch 301 loss: 4.137250
Epoch: 9        Training Loss: 4.132874         Validation Loss: 4.249936
Epoch 10, Batch 1 loss: 3.800079
Epoch 10, Batch 101 loss: 4.056664
Epoch 10, Batch 201 loss: 4.064674
Epoch 10, Batch 301 loss: 4.077953
Epoch: 10       Training Loss: 4.077172         Validation Loss: 4.134114
Validation loss decreased (4.160961 --> 4.134114).  Saving model ...
Epoch 11, Batch 1 loss: 4.343294
Epoch 11, Batch 101 loss: 4.002690
Epoch 11, Batch 201 loss: 4.032941
Epoch 11, Batch 301 loss: 4.033110
Epoch: 11       Training Loss: 4.028441         Validation Loss: 3.998078
Validation loss decreased (4.134114 --> 3.998078).  Saving model ...
Epoch 12, Batch 1 loss: 3.509154
Epoch 12, Batch 101 loss: 3.969872
Epoch 12, Batch 201 loss: 3.977506
Epoch 12, Batch 301 loss: 3.980928
Epoch: 12       Training Loss: 3.979130         Validation Loss: 3.964167
Validation loss decreased (3.998078 --> 3.964167).  Saving model ...
Epoch 13, Batch 1 loss: 3.954685
Epoch 13, Batch 101 loss: 3.899901
Epoch 13, Batch 201 loss: 3.890163
Epoch 13, Batch 301 loss: 3.886095
Epoch: 13       Training Loss: 3.890691         Validation Loss: 3.831596
Validation loss decreased (3.964167 --> 3.831596).  Saving model ...
Epoch 14, Batch 1 loss: 4.141478
Epoch 14, Batch 101 loss: 3.792190
Epoch 14, Batch 201 loss: 3.806967
Epoch 14, Batch 301 loss: 3.829901
Epoch: 14       Training Loss: 3.828298         Validation Loss: 3.877468
Epoch 15, Batch 1 loss: 3.902372
Epoch 15, Batch 101 loss: 3.844658
Epoch 15, Batch 201 loss: 3.817249
Epoch 15, Batch 301 loss: 3.795112
Epoch: 15       Training Loss: 3.799385         Validation Loss: 3.860865
```

And then I have stored the weigh and model of with the lowest validation loss of epoch in the training function.

Dog breed classifier with transfer learning I used transfer learning technique to implement the pretrained model ResNet-50 using pytorch and I have explained the ResNet-50

model in the algorithm section ,I also choose the SGD optimizer as in the model build from scratch as it was working very well here too and set the learning rate to 0.2 and made only five epochs which where enough

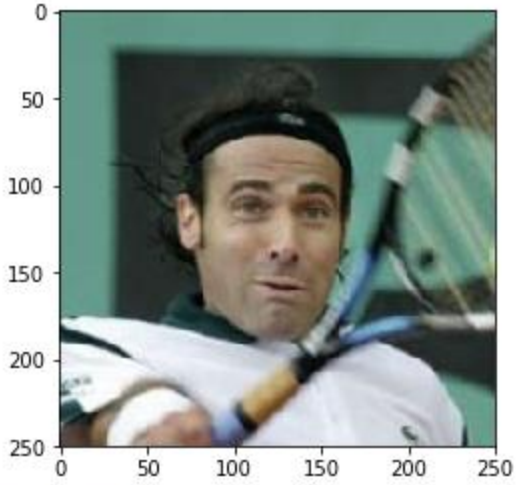
```
In [17]: # train the model
model_transfer =model_transfer =  train(5, loaders_transfer, model_transfer, optimizer_transfer, criterion_transfer, use_cuda, 'model_tra
nsfer.pt')
# load the model that got the best validation accuracy (uncomment the line below)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

```
Epoch 1, Batch 1 loss: 10.137520
Epoch 1, Batch 101 loss: 7.280512
Epoch 1, Batch 201 loss: 6.167127
Epoch 1, Batch 301 loss: 5.490372
Epoch: 1        Training Loss: 5.307045         Validation Loss: 3.037261
Validation loss decreased (inf --> 3.037261).  Saving model ...
Epoch 2, Batch 1 loss: 3.271444
Epoch 2, Batch 101 loss: 3.194010
Epoch 2, Batch 201 loss: 3.044270
Epoch 2, Batch 301 loss: 2.915280
Epoch: 2        Training Loss: 2.866323         Validation Loss: 1.729430
Validation loss decreased (3.037261 --> 1.729430).  Saving model ...
Epoch 3, Batch 1 loss: 2.228527
Epoch 3, Batch 101 loss: 2.273277
Epoch 3, Batch 201 loss: 2.251157
Epoch 3, Batch 301 loss: 2.165162
Epoch: 3        Training Loss: 2.147053         Validation Loss: 1.262911
Validation loss decreased (1.729430 --> 1.262911).  Saving model ...
Epoch 4, Batch 1 loss: 1.971799
Epoch 4, Batch 101 loss: 1.873842
Epoch 4, Batch 201 loss: 1.834987
Epoch 4, Batch 301 loss: 1.811499
Epoch: 4        Training Loss: 1.801424         Validation Loss: 1.011081
Validation loss decreased (1.262911 --> 1.011081).  Saving model ...
Epoch 5, Batch 1 loss: 2.010596
Epoch 5, Batch 101 loss: 1.686984
Epoch 5, Batch 201 loss: 1.664438
Epoch 5, Batch 301 loss: 1.643347
Epoch: 5        Training Loss: 1.620880         Validation Loss: 0.934591
Validation loss decreased (1.011081 --> 0.934591).  Saving model ...
```

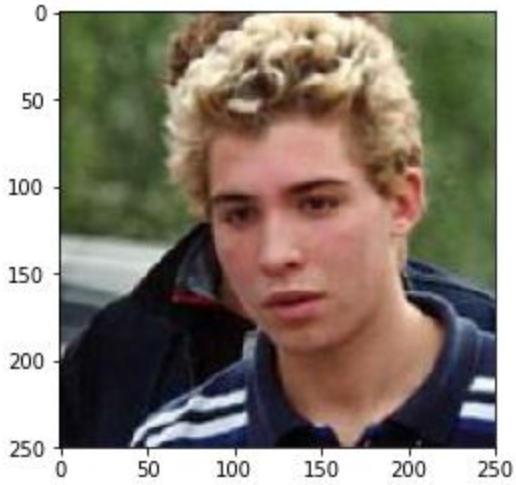
Refinement

In the dog breed classifier with transfer learning I have used ResNet-50 with accuracy of 71% .The CNN model created form scratch accuracy was 14% and the final model was so successful in identifying the dog breeds and also the breed that looks close to the human the images below shows the output of my model and how successful it is .

```
Detected Human!
Predicted breed:  English toy spaniel
```

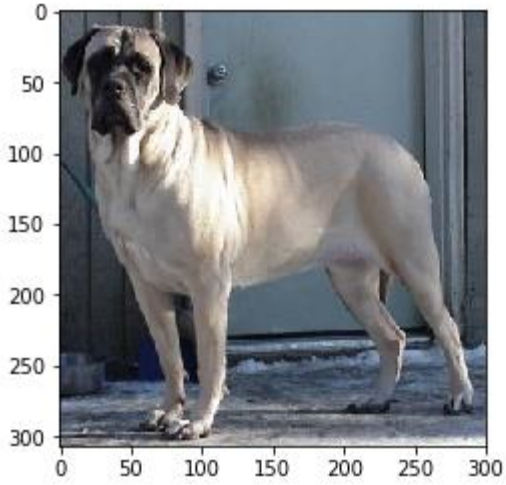



```
Detected Human!
Predicted breed:  Petit basset griffon vendeen
```



```
Detected Dog!
Predicted breed:  Cane corso
```


```
Detected Dog!
Predicted breed:  Bullmastiff
```




```
Detected Dog!
Predicted breed:  Bullmastiff
```

4.Results

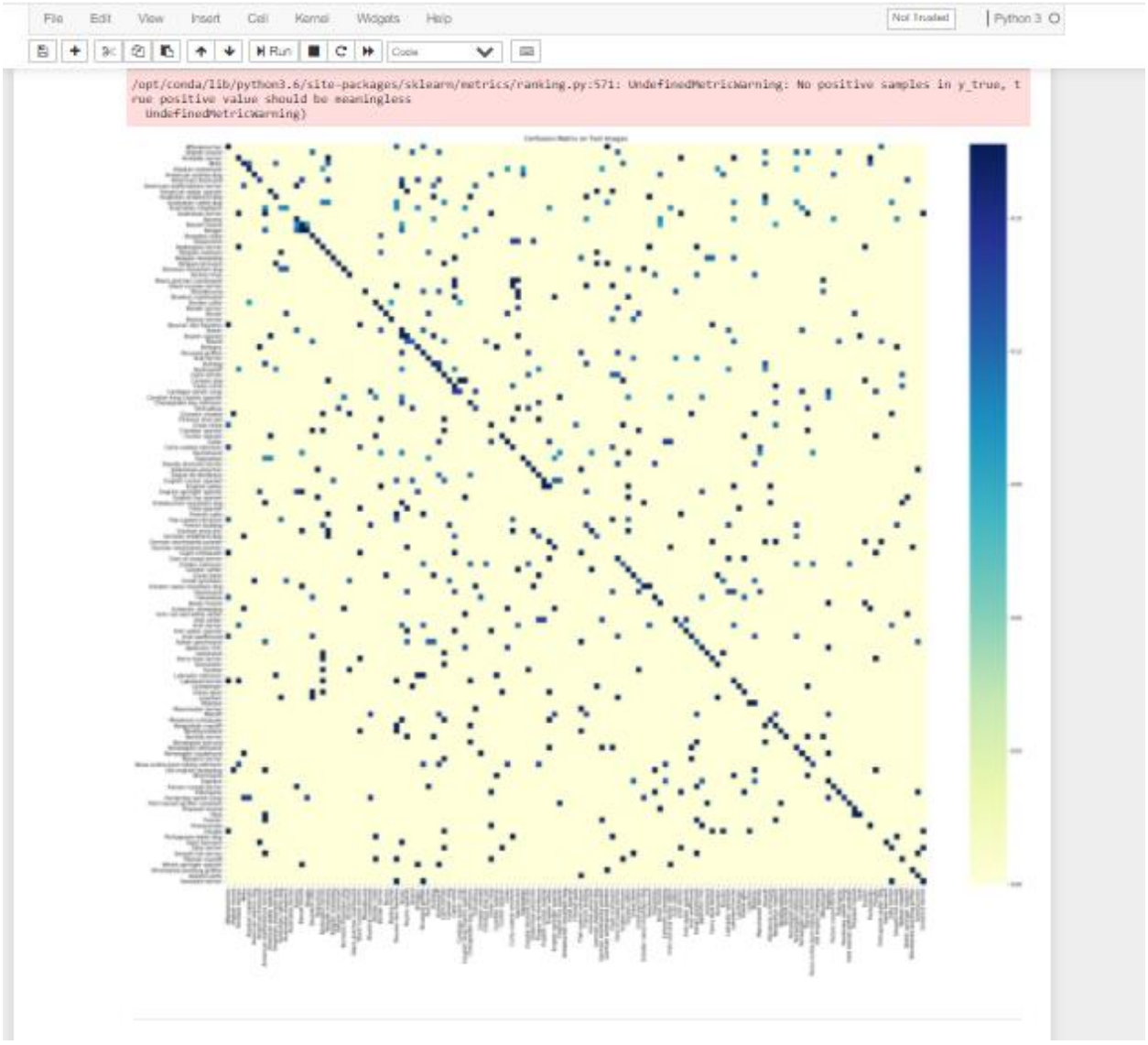
Model Evaluation and Validation

Dog face detector : The dog face detector function was created using VGG-16 model 100% of dog faces were

detected in the first 100 images of dog dataset and 1% of dog faces detected in first 100 image of human dataset.

Human face detector :For implementing the human face detector function I used the Haar feature based cascade classifier form python OpenCv 98% of human faces were detected in the first 100 images of human face dataset and 17% of human faces detected in first 100 images of dog dataset.

CNN using transfer learning : the model is created using trans fer learning with ResNet-50 which is a CNN model which consi sts of 50 layers deep and It is so good in detecting objects as i t can detect 1000 object in a single image but it needs images of input size 224 X224 and the final model accuracy was 71% on test data 597/836 and the Test Loss: 0.958761 and the con fusion matrix for the ResNet-50 transfer learning model show s with results that it can classify dogs and humans into certain dog breeds and the F1 score plot on the tested data shows th at a lot of dog breeds have high F1 score which indicates that the transfer learning model is more robust for different dog b reeds .



Justification

For me the performance of my model was great as the accuracy of the transfer learning model is 71% which exceeds the benchmark by 11 % and also the accuracy of the CNN model build from scratch was fine as it's accuracy was 14% which exceeded the benchmark by 4 % I believe that the final

model is significant enough to have adequately solved the problem.

5.Conclusion

Reflection

1.Identifying the problem and looking for releavant datasets

2.Downloading the dataset

3.Implementing the human face detector

4.Implementing the dog face detector

5.Implementing the CNN model from scratch

6.Implementing the transfer learning model and training and testing it

7.Documenting the project

Improvement

The model could be improved by increasing the convolutional layers but I have avoided complexity and the algorithm could also be improved by increasing the dataset and hyperparameter tunning and one of the methods also that could have improved the algorithm was implementing ResNet-101 which is very deep neural network.

References

https://pytorch.org/docs/stable/index.html

https://github.com/Abdelrahman13-coder/Capstone-project-Dog-Breed/blob/master/dog_app.ipynb

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

https://www.youtube.com/channel/UCxxljM6JkSvJVSD_T90ZnMw

https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html#:~:text=It%20is%20a%20machine%20learning,detect%20objects%20in%20other%20images.&text=Initially%2C%20the%20algorithm

https://www.mathworks.com/help/deeplearning/ref/vgg16.html#:~:text=Description,%2C%20pencil%2C%20and%20many%20animals.%20needs%20a,faces)%20to%20train%20the%20classifier.

https://www.mathworks.com/help/deeplearning/ref/resnet50.html#:~:text=ResNet%2D50%20is%20a%20convolutional,%2C%20pencil%2C%20and%20many%20animals.

https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7#:~:text=A%20Convolutional%20neural%20network%20(CNN,a%20filter%20over%20the%20input.