DOG BREED CLASSIFIER

Udacity Machine Learning Engineer Capstone Project Report

Photo by Karsten Winegeart on Unsplash

Rahul Chauhan

30.10.2020

1. DEFINITION

Project Overview

In this project we aimed to build a neural network based multiclass classifier that could identify a breed of dog if given a photo or image as input.

The model can even detect human faces and try to output the nearest breed of the 133 dogs that it was trained on.

We used neural networks or convolutional neural networks and it’s various architectures to achieve the same.

Problem Statement

Biodiversity is one of the most complex and vital features of our planet. Biodiversity simply means ‘the variety of life on earth’. We can see this variety on several levels. Among these diverse animals we call one a man’s best friend. Even though they are all individually different we can differentiate them from their visual and other features into different breeds. We wish to train a machine to be able to recognise and classify these numerous species. Also for fun, we will try to classify humans into dog breeds that the trained model thinks looks similar to them.

Metrics

We used accuracy as our metric for evaluation. Accuracy is the proportion of correct classifications among all classifications.

$$\boxed{accuracy := \frac{\text{correct classifications}}{\text{number of classifications}}}$$

Accuracy is a simple and useful measure if we have the same amount of samples per class, which is the case with the dataset that we are working on.

When we use accuracy, we assign equal cost to false positives and false negatives. Since we do not prefer any one breed over others and misclassification of all breeds have similar penalty and correct classifications similar rewards, accuracy can give us a good idea about the effectiveness of our classifier.

2. ANALYSIS

Data Exploration

The dog_names variable stores a list of the names for the classes to use in our prediction model. Based on the path name, we see a total of 8351 images of dogs belonging to 133 different dog breeds which are then categorized into 6680, 835, and 836 images in training, validation, and testing.

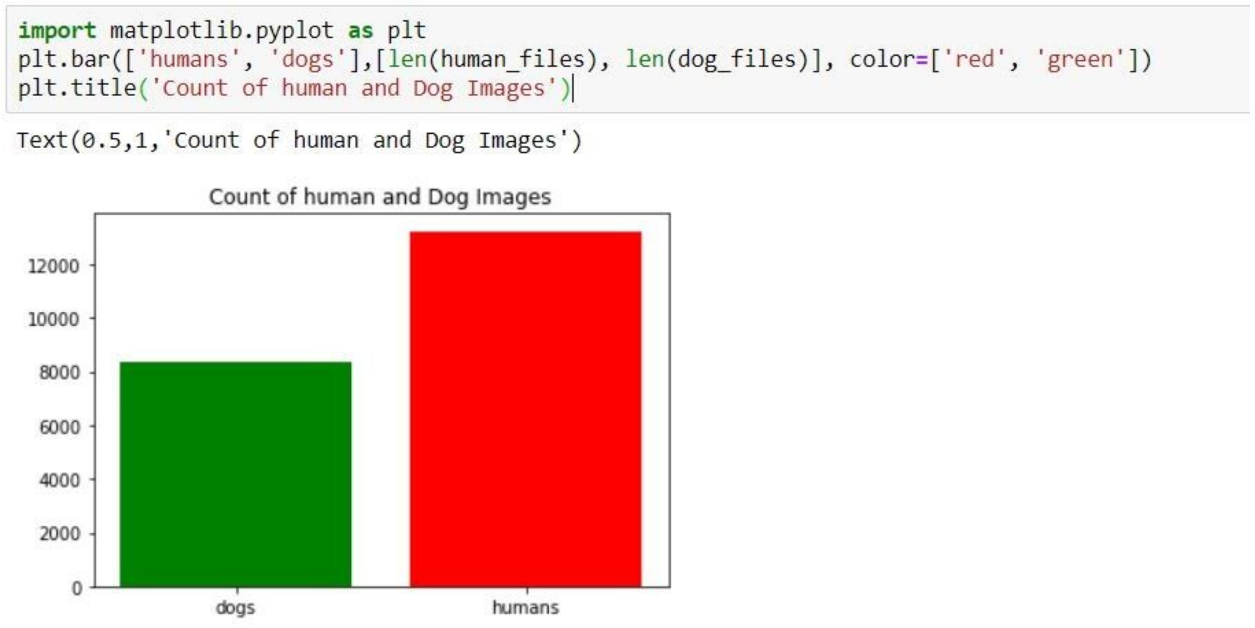
Dog Images — The dog images provided are available in the repository within the Images directory further organized into the train, valid and test subfolders

Human Faces — An exhaustive dataset of faces of celebrities have also been added to the repository in the lfw folder

Haarcascades — ML-based approach where a cascade function is trained from a lot of positive and negative images, and used to detect objects in other images. The algorithm uses the Haar frontal face to detect humans. So the expectation is that an image with the frontal features clearly defined is required

Test Images — A folder with certain test images have been added to be able to check the effectiveness of the algorithm.

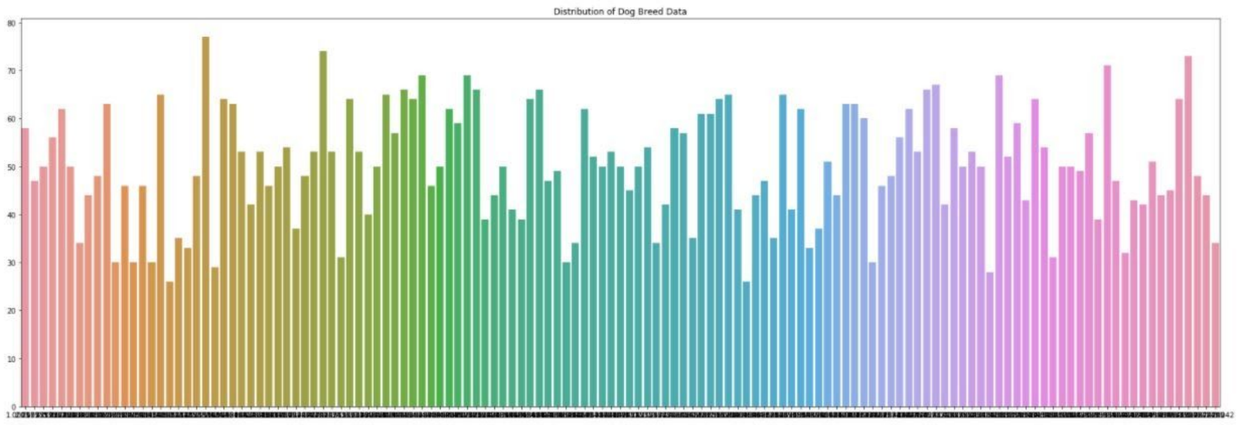
1. We check for samples available



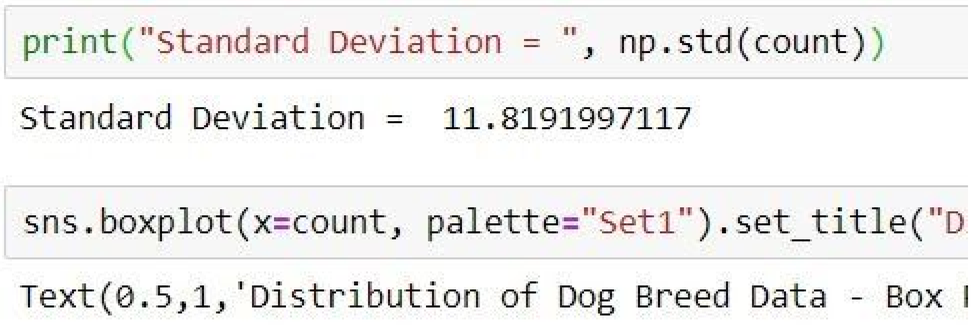
```
import matplotlib.pyplot as plt
plt.bar(['humans', 'dogs'],[len(human_files), len(dog_files)], color=['red', 'green'])
plt.title('Count of human and Dog Images')
```

```
Text(0.5,1,'Count of human and Dog Images')
```

2. Distribution of breed in dog_photos



3. Check for outliers

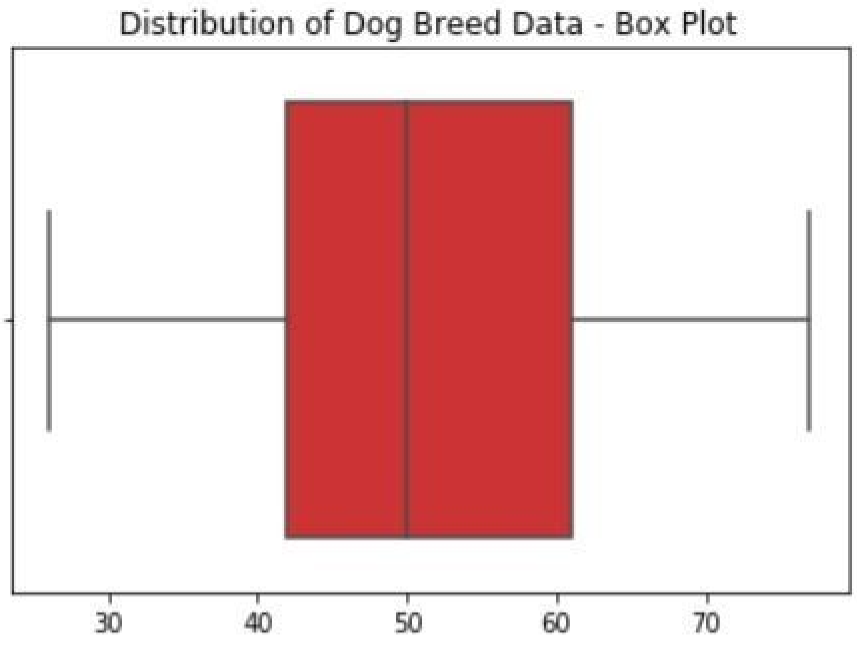


```
print("Standard Deviation = ", np.std(count))
```

```
Standard Deviation =  11.8191997117
```

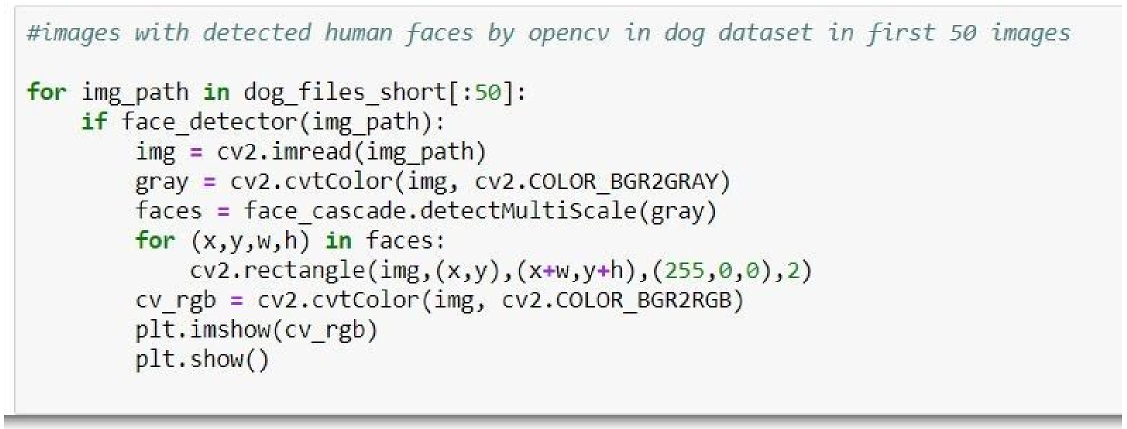
```
sns.boxplot(x=count, palette="Set1").set_title("D
```

```
Text(0.5,1,'Distribution of Dog Breed Data - Box 
```

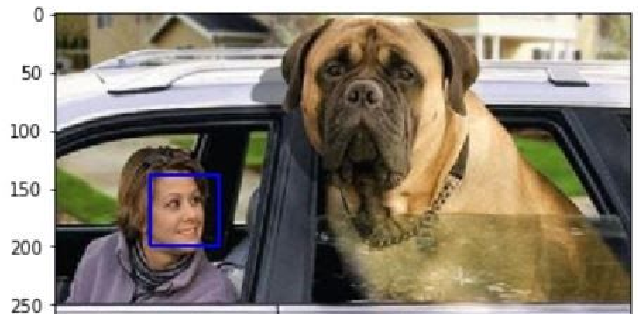


Exploratory Visualisation

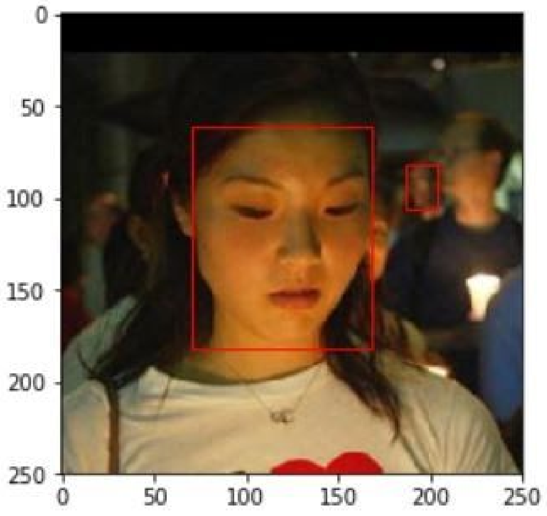
1. Found humans with both dogs and humans

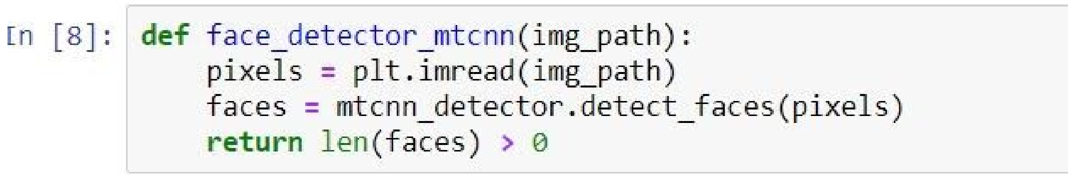


```
#images with detected human faces by opencv in dog dataset in first 50 images

for img_path in dog_files_short[:50]:
    if face_detector(img_path):
        img = cv2.imread(img_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray)
        for (x,y,w,h) in faces:
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(cv_rgb)
        plt.show()
```



2. Explored Neural Network based algorithms for face detection

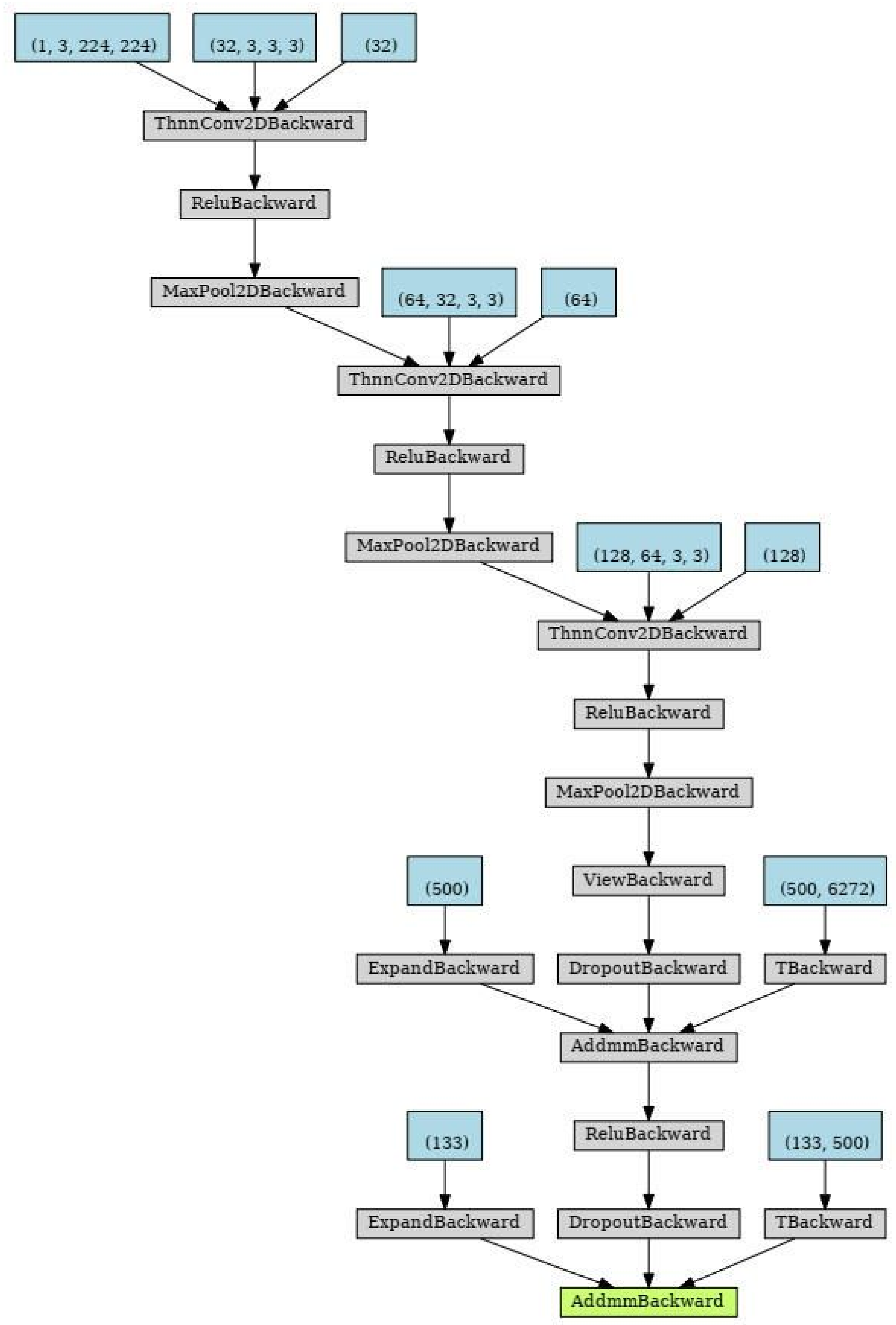




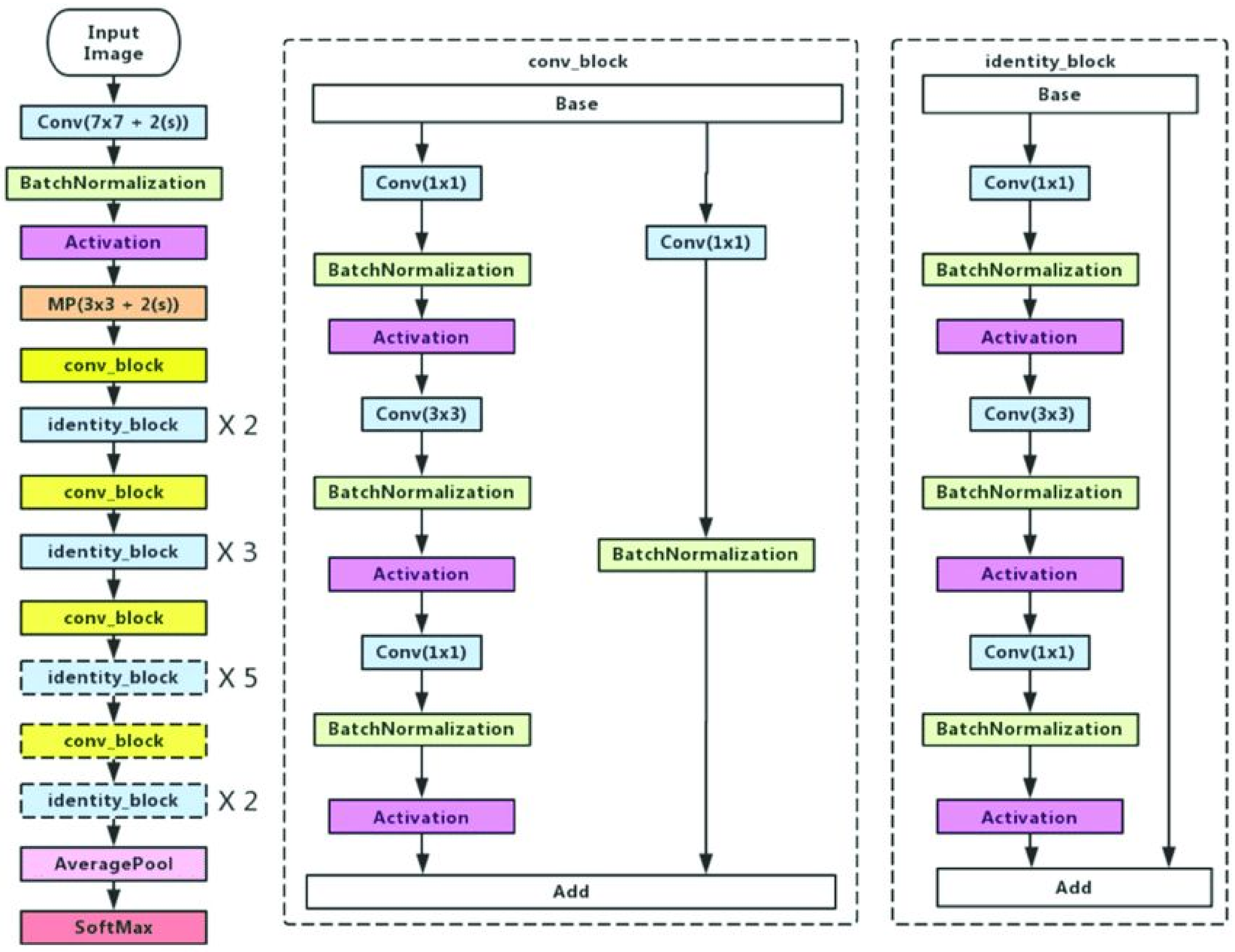
```
In [8]: def face_detector_mtcnn(img_path):
            pixels = plt.imread(img_path)
            faces = mtcnn_detector.detect_faces(pixels)
            return len(faces) > 0
```

Algorithms and Techniques

1. Created a basic convolutional neural network from scratch with the following architecture.



2. ResNet50 Architecture (A pretrained ResNet was obtained)



ResNet50 Architecture : Creative Commons License

Source:
https://www.researchgate.net/figure/Left-ResNet50-architecture-Blocks-with-dotted-line-represents-modules-that-might-be_fig3_3313648
7

Benchmark

We used the thresholds provided by Udacity to benchmark our model.

We were required to obtain an accuracy of 10% and above for the scratch implementation and an accuracy of 60% and above for the transfer learning implementation.

We also compared our results with those available on Kaggle for a similar problem and used the 75th percentile result (77 % accuracy) as our benchmark.

3. METHODOLOGY

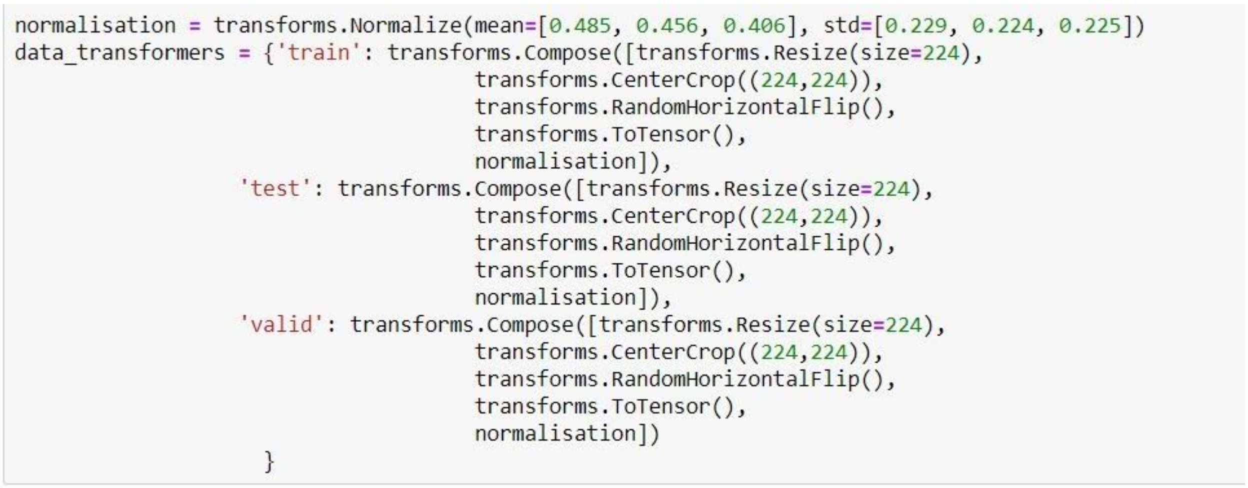
Data Preprocessing

Since most machine learning models require data in a certain format before they can begin analysing them, we need to preprocess data before feeding it to the model.

We also normalised and resized the data before feeding it into the network.

Synthetically modifying data in this manner is called data augmentation. In the real world images can be taken in various lighting conditions, various angles and orientation, to build a model that can work with that kind of test data we use artificial techniques to create this kind of data and train our model on it to make it more robust.

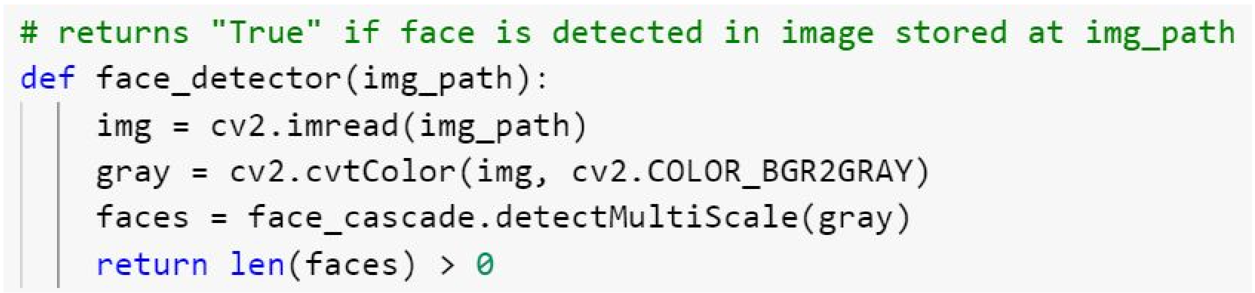
Finally we converted them to tensors as per demands of the models.



```
normalisation = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
data_transformers = {'train': transforms.Compose([transforms.Resize(size=224),
                                     transforms.CenterCrop((224,224)),
                                     transforms.RandomHorizontalFlip(),
                                     transforms.ToTensor(),
                                     normalisation]),
                   'test': transforms.Compose([transforms.Resize(size=224),
                                     transforms.CenterCrop((224,224)),
                                     transforms.RandomHorizontalFlip(),
                                     transforms.ToTensor(),
                                     normalisation]),
                   'valid': transforms.Compose([transforms.Resize(size=224),
                                     transforms.CenterCrop((224,224)),
                                     transforms.RandomHorizontalFlip(),
                                     transforms.ToTensor(),
                                     normalisation])
                     }
```

Implementation

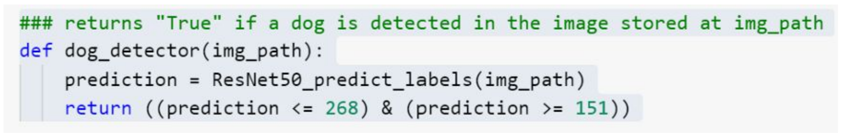
1. OpenCV's implementation of face detection classifier was used to detect humans in provided images. This is a Harr feature based cascade classifier.



```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

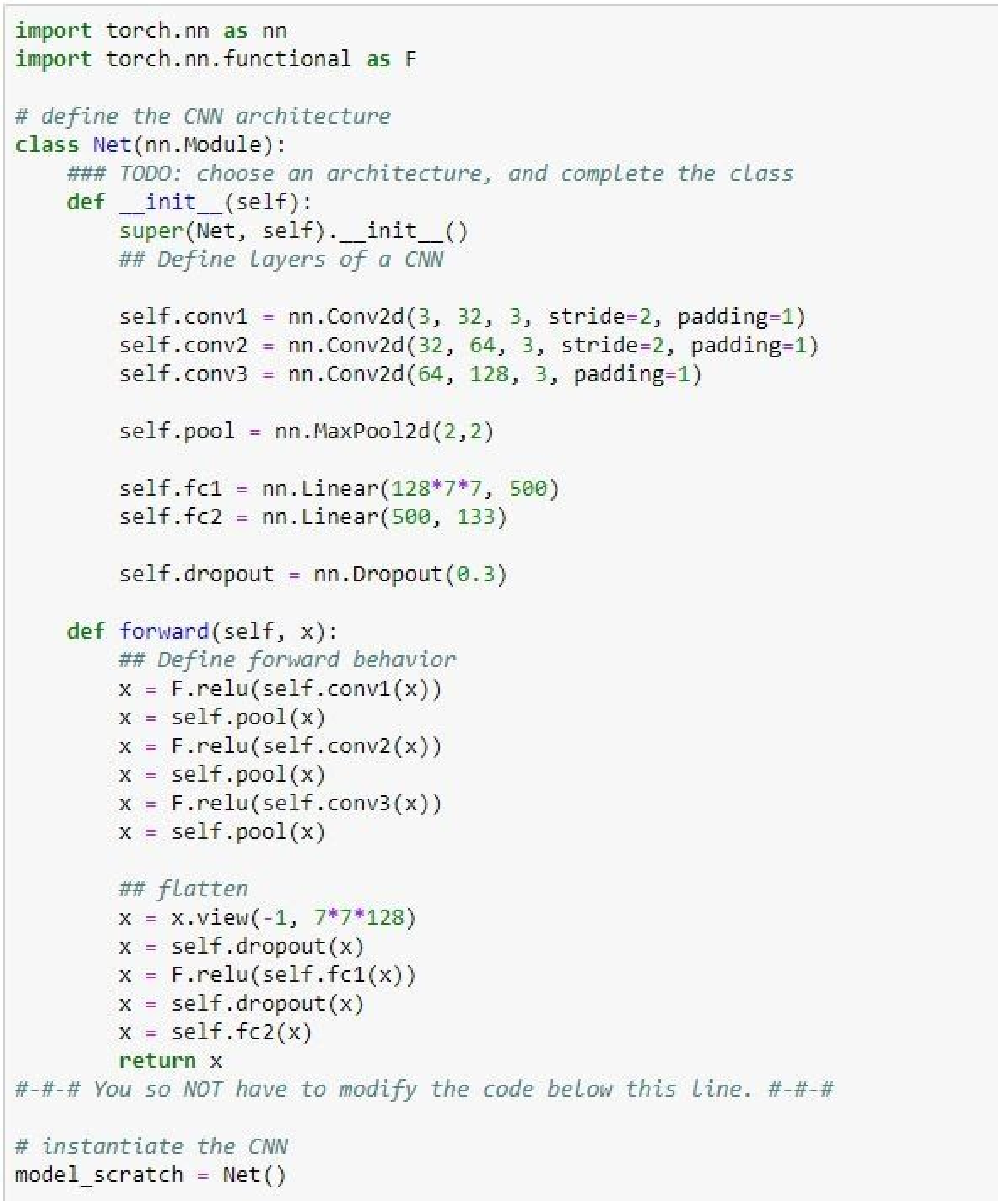
2. Here, we use a pre-trained ResNet-50 model to detect dogs in images. Our first line of code downloads the ResNet-50 model, along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks.

```
# define ResNet50 model
ResNet50_model = ResNet50(weights='imagenet')
```



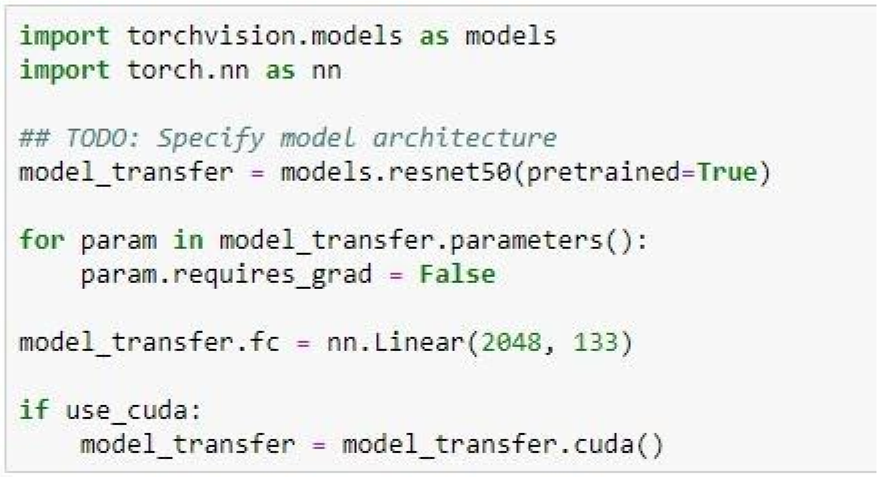
```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    prediction = ResNet50_predict_labels(img_path)
    return ((prediction <= 268) & (prediction >= 151))
```

3. Then we created a model from scratch to classify our images



```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN

        self.conv1 = nn.Conv2d(3, 32, 3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)

        self.pool = nn.MaxPool2d(2,2)

        self.fc1 = nn.Linear(128*7*7, 500)
        self.fc2 = nn.Linear(500, 133)

        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)

        ## flatten
        x = x.view(-1, 7*7*128)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
#-#-# You so NOT have to modify the code below this line. #-#-#

# instantiate the CNN
model_scratch = Net()
```

4. We then used a pretrained model for the same purpose of identification.

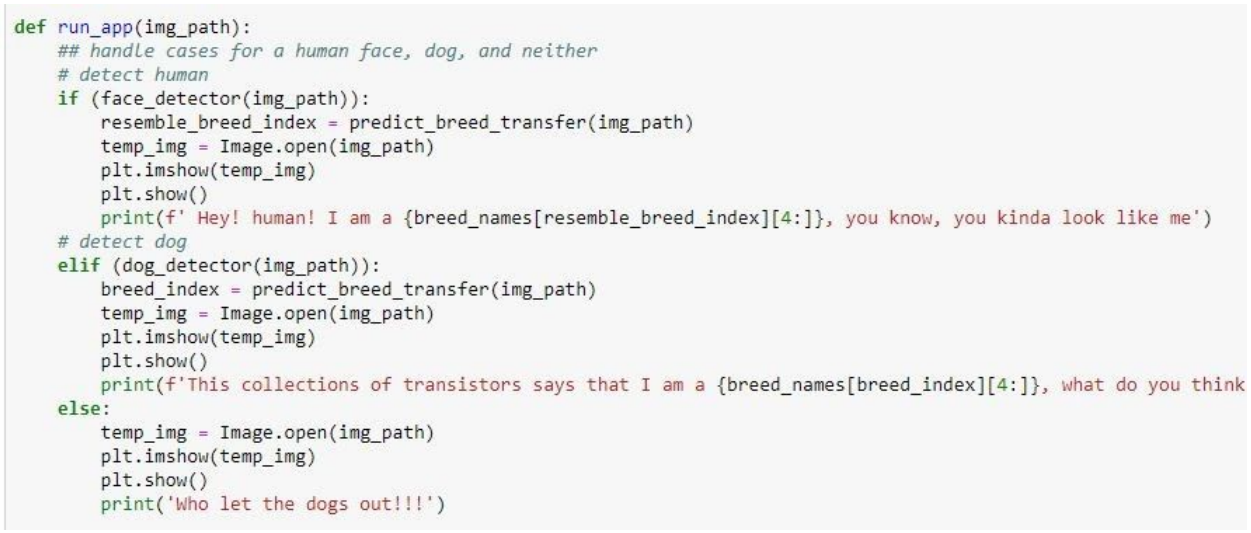


```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
model_transfer = models.resnet50(pretrained=True)

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

model_transfer.fc = nn.Linear(2048, 133)

if use_cuda:
    model_transfer = model_transfer.cuda()
```

5. We wrote a algorithm to classify and give us the predicted output

   [These are the steps mentioned in Udacity provided jupyter notebook]

- 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 output that indicates an error.



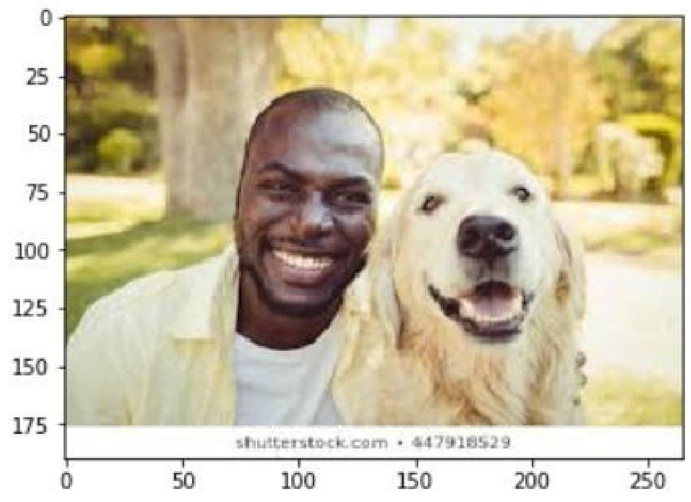
```
def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    # detect human
    if (face_detector(img_path)):
        resemble_breed_index = predict_breed_transfer(img_path)
        temp_img = Image.open(img_path)
        plt.imshow(temp_img)
        plt.show()
        print(f' Hey! human! I am a {breed_names[resemble_breed_index][4:]}, you know, you kinda look like me')
    # detect dog
    elif (dog_detector(img_path)):
        breed_index = predict_breed_transfer(img_path)
        temp_img = Image.open(img_path)
        plt.imshow(temp_img)
        plt.show()
        print(f'This collections of transistors says that I am a {breed_names[breed_index][4:]}, what do you think
    else:
        temp_img = Image.open(img_path)
        plt.imshow(temp_img)
        plt.show()
        print('Who let the dogs out!!!')
```

4. RESULTS

Model Evaluation and Validation

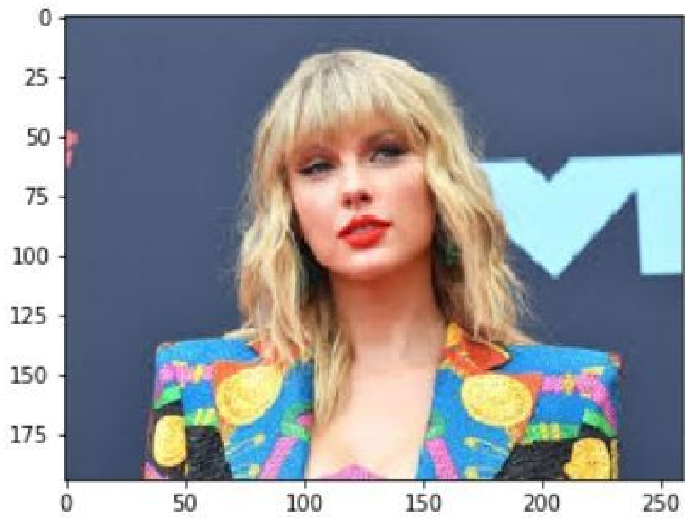
We take our new algorithm for a spin! What kind of dog does the algorithm think that a man looks like? If we have a dog, does it predict the dog's breed accurately? If we have a cat, does it mistakenly think that our cat is a dog?

```
#An image with both man and a dog.
run_app('my_dog_photos/mananddog.jpg')
```



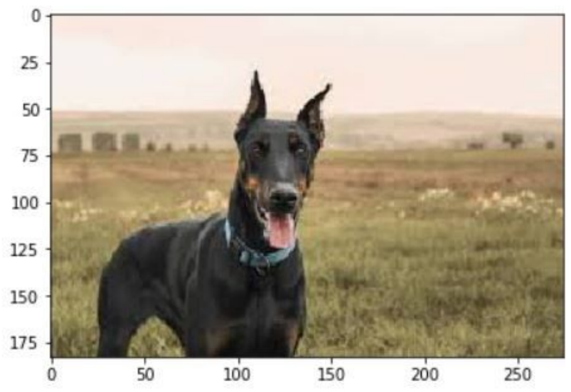
```
Hey! human! I am a Great_pyrenees, you know, you kinda look like me
```

```
#Taylor Swift
run_app('my_dog_photos/taylor.jpg')
```



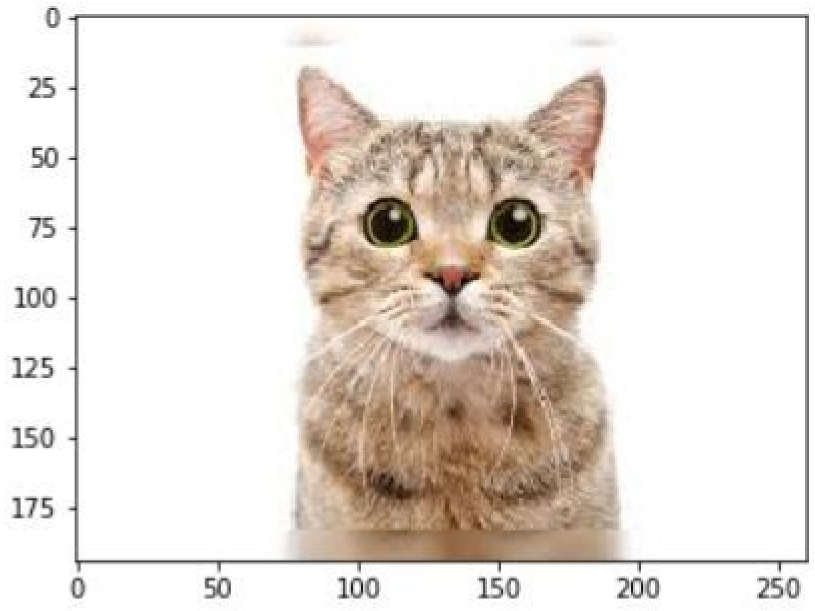
```
Hey! human! I am a Chinese_crested, you know, you kinda look like me
```

```
#Doberman
run_app('my_dog_photos/doberman.jpg')
```



```
This collections of transistors says that I am a Doberman_pinscher, what do you think, is it correct
```

```
#cat
run_app('my_dog_photos/cat.jpg')
```



```
Who let the dogs out!!!
```

Justification

The output accuracy is good given the small number of epochs the algorithms ran on, we can see that the training or validation error hadn't reached saturation yet. Simply training further might improve the accuracy. (I had to stop it at lower epochs because of time constraints and the fact that this workspace stops if left idle or connection is lost.

Other hyperparameters such as batch size can be tuned to further improve the accuracy

Since neural networks are data hungry, providing more images can help them get better. Natural images or data augmentation can help them generalise better and thus prove valuable in the real world too.

5. CONCLUSION

The project was fun and packed with things to learn. Although we did good enough given the time constraints, we could also have looked more deeply into the images themselves that we used for training the dog breeds.

We could have increased the data augmentation by changing hue, saturation and other properties of the images. We saw that a simple neural network not only took a lot of time but also failed to provide satisfactory results.

I am still thinking of a better procedure to handle images that have both humans and dogs in them. Will keep improving the project, but have to stop here due to my nanodegree time constraints.

6. REFERENCES

https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://neurohive.io/en/popular-networks/vgg16/

https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035