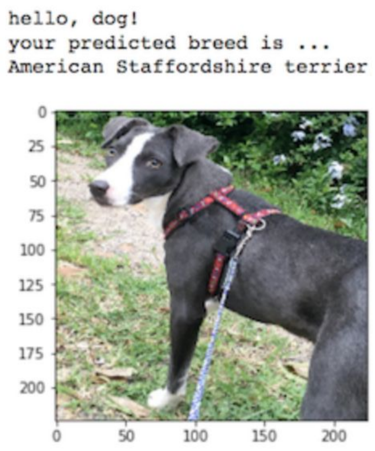
Dog Breed Classifier

Problem Statement

In this project, we will try to classify dogs as per their breeds. This is a supervised learning problem as we will be using the images and its labels to train a model. At the end of this project, the code should 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 potential sample output of the finished project:



```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```

Metrics

We will be using Accuracy as the metric. Even though the data is not evenly balanced, the dataset is not heavily unbalanced. There are still enough samples for each class. We will also be using data augmentation to ensure we have a more diverse dataset.

Test accuracy = number of predictions which matches the true label / Total number of samples in the test set.

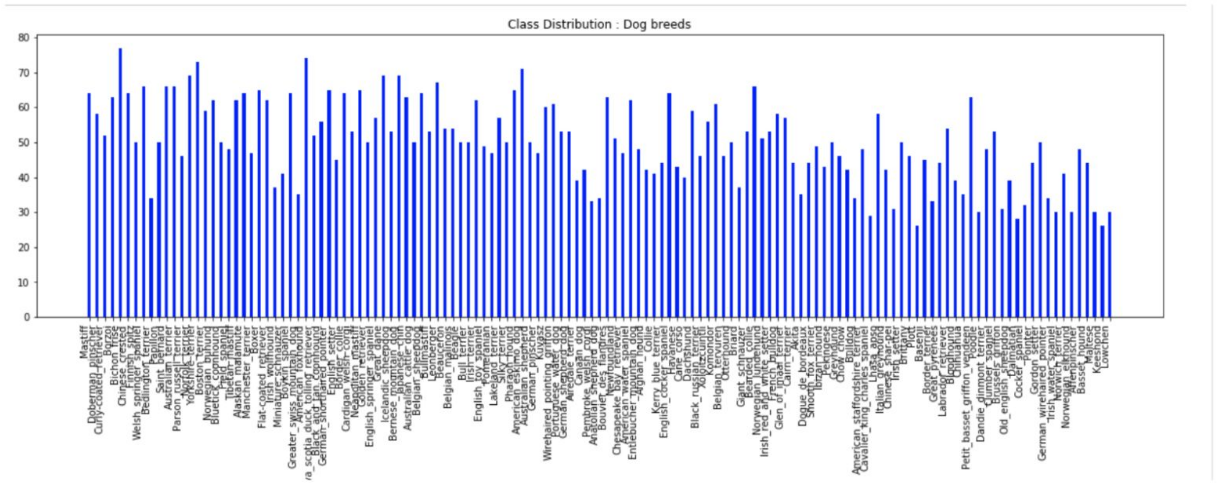
Dataset

There are two datasets used for this project which are provided by Udacity:

- dog dataset : The dog dataset consists of 8,351 dog images across 133 breeds. The dataset would be divided between training, validation and test set. The dataset will be distributed as follows:
    - Number of training images: 6680
    - Number of validation images: 835
    - Number of test images: 836
- human dataset : The human dataset consists of 13,233 human images.

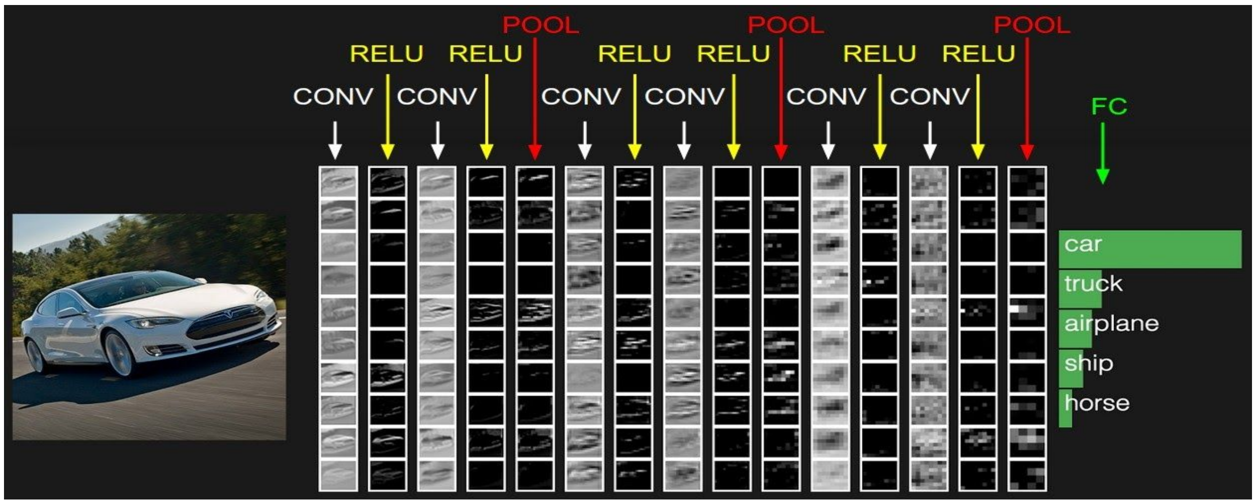
Exploratory Visualization

The class distribution of the training set is not evenly balanced as shown below:



Algorithms and Techniques

Convolutional Neural Networks are a type of NNs which uses much less number of parameters to produce good results for computer vision tasks like image classification, segmentation, object detection, etc. CNNs are normally quite deep and consist of different types of hidden layers like convolutional layer, pooling layer, batchnorm layer and fully connected layers as well. A lot of success in applying deep learning in computer vision tasks comes from using CNNs. But we also do not train a model from scratch for a new application. We have pretrained models trained on huge datasets like Imagenet, easily accessible through Pytorch. This reduces the training time for a new model trained on a different dataset as the model weights don't start from random and the model is able to determine shapes and figures easily.



Example of a Convnet architecture. Source: [cs231 Stanford]

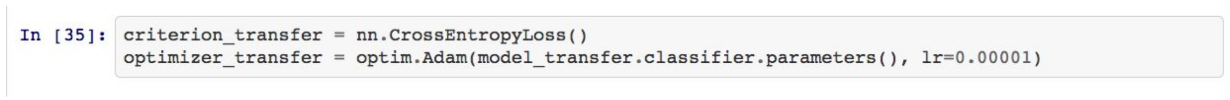
The CNN architectures consist of following layers:

- Convolutional layer: It is used to extract features from an image.
- ReLU layer: It adds non linearity to the output from the convolutional layer.
- Maxpool layer: It decreases the dimensionality by selecting the max value in it's filter.
- Batchnorm layer: A batchnorm layer normalizes the input for the next convolutional later.
- Dropout layer: It helps in creating a robust model by randomly switching off some of the neurons during an iteration so that the other neurons also had to learn what these neurons were learning.
- Linear layer or fully connected layer: This is a plain neural network consisting of hidden layers where each neuron in a layer is connected to the next or previous layer.

Following hyperparameters are required for training a models:

- Learning rate: It decides how large or small the gradient updates are to the weights.
- Batch size: The number of samples used for a single training step.
- Number of epochs: The number of iterations to train a model.

We will be using Cross entropy loss and Adam optimizer to train the model.



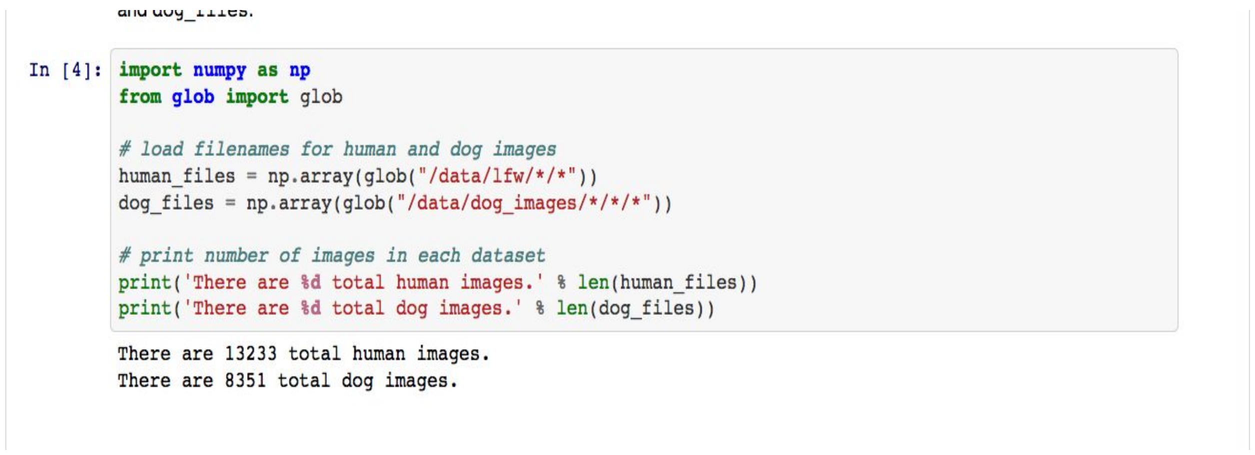
```
In [35]: criterion_transfer = nn.CrossEntropyLoss()
         optimizer_transfer = optim.Adam(model_transfer.classifier.parameters(), lr=0.00001)
```

Benchmark

The benchmark for CNN from scratch was 10% and for the transfer learning model was 60%.

Methodology

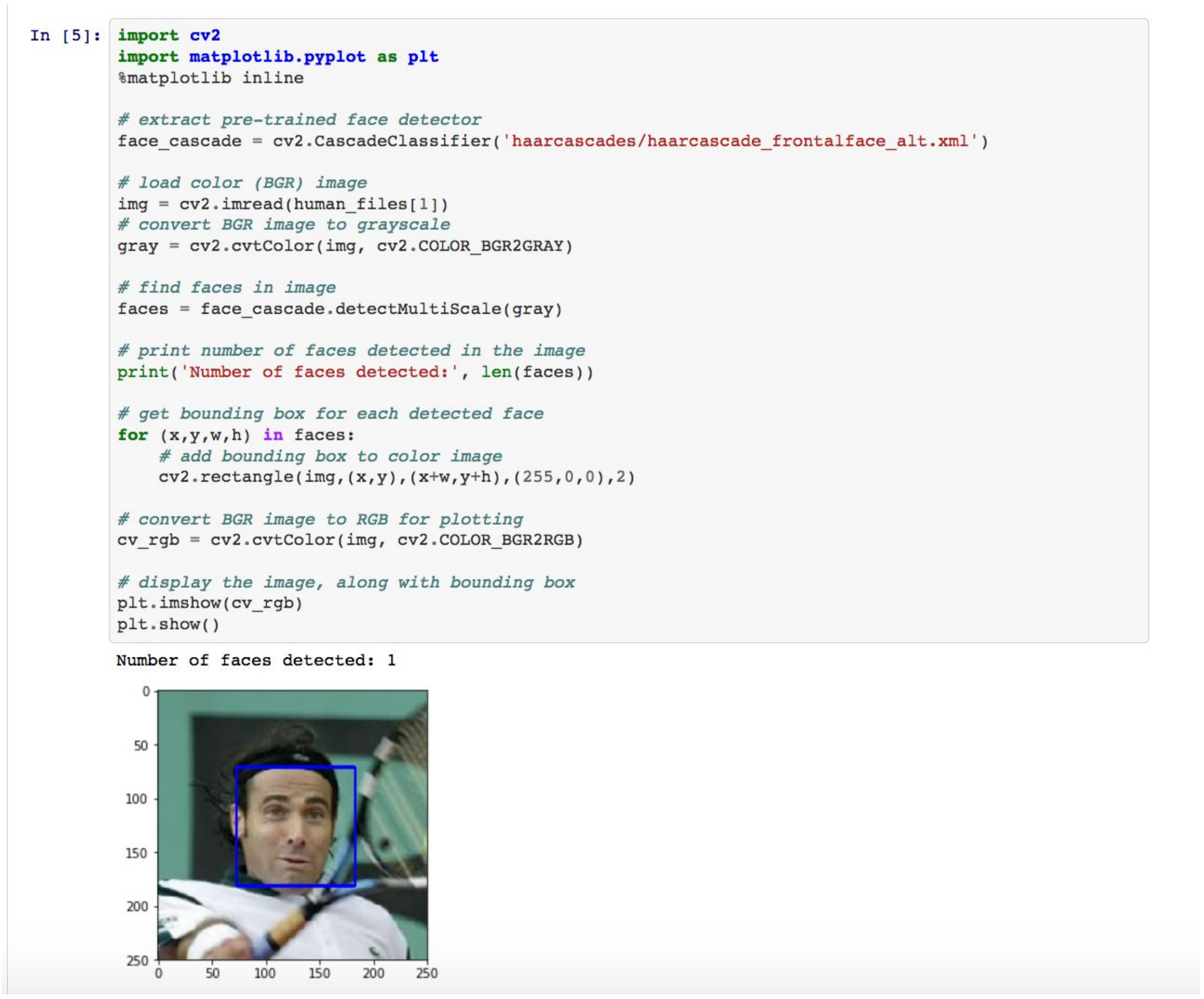
1. Import Datasets



and dog_files.

```
In [4]: import numpy as np
        from glob import glob

        # load filenames for human and dog images
        human_files = np.array(glob("/data/lfw/*/*"))
        dog_files = np.array(glob("/data/dog_images/*/*/*"))

        # print number of images in each dataset
        print('There are %d total human images.' % len(human_files))
        print('There are %d total dog images.' % len(dog_files))

        There are 13233 total human images.
        There are 8351 total dog images.
```

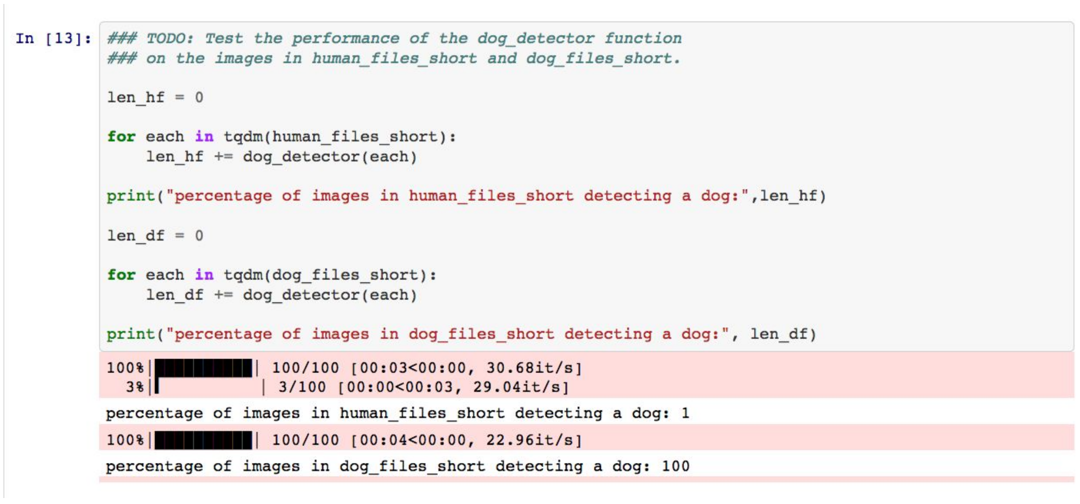
2. We will use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.



```
In [5]: import cv2
        import matplotlib.pyplot as plt
        %matplotlib inline

        # extract pre-trained face detector
        face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

        # load color (BGR) image
        img = cv2.imread(human_files[1])
        # convert BGR image to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # find faces in image
        faces = face_cascade.detectMultiScale(gray)

        # print number of faces detected in the image
        print('Number of faces detected:', len(faces))

        # get bounding box for each detected face
        for (x,y,w,h) in faces:
            # add bounding box to color image
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

        # convert BGR image to RGB for plotting
        cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # display the image, along with bounding box
        plt.imshow(cv_rgb)
        plt.show()
```

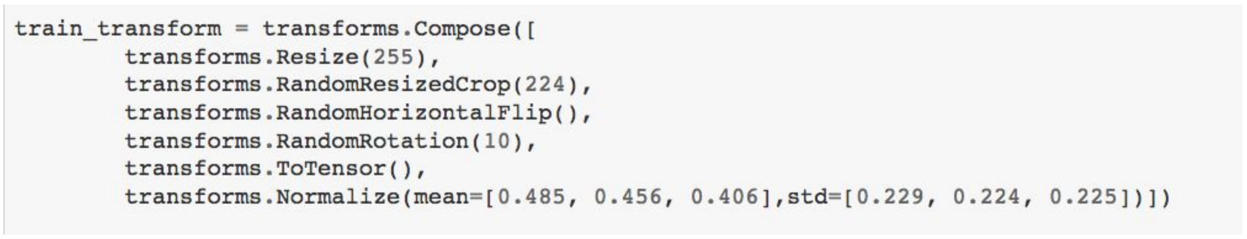
```
Number of faces detected: 1
```

3. We use a pre-trained model(VGG16) to detect dogs in images.



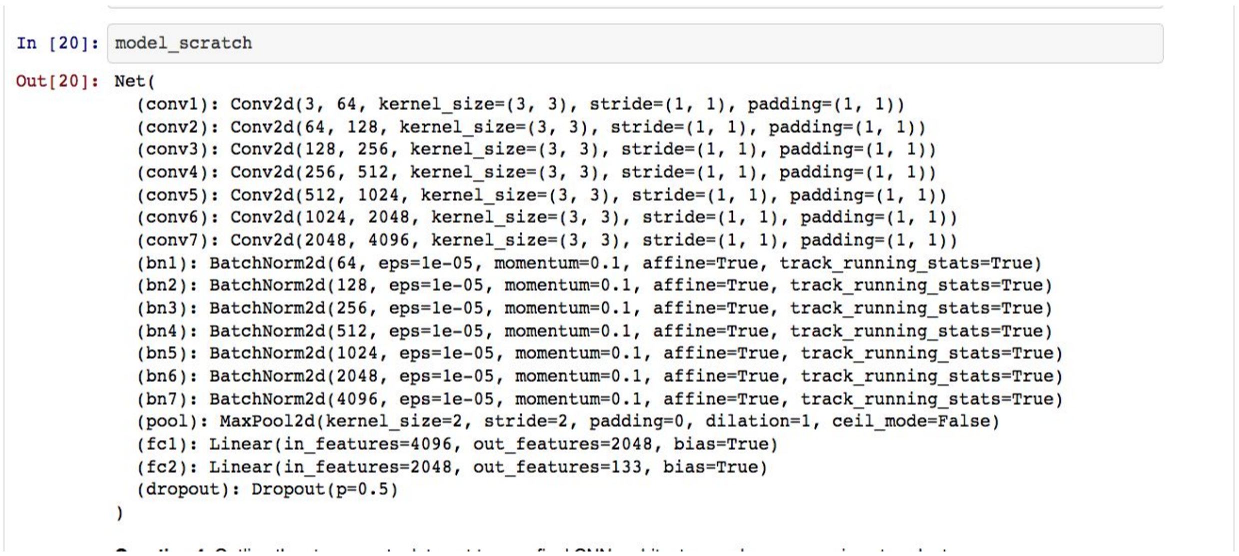
```
In [13]: ### TODO: Test the performance of the dog_detector function
         ### on the images in human_files_short and dog_files_short.

         len_hf = 0

         for each in tqdm(human_files_short):
             len_hf += dog_detector(each)

         print("percentage of images in human_files_short detecting a dog:",len_hf)

         len_df = 0

         for each in tqdm(dog_files_short):
             len_df += dog_detector(each)

         print("percentage of images in dog_files_short detecting a dog:", len_df)

         100%|██████████| 100/100 [00:03<00:00, 30.68it/s]
           3%|▎         | 3/100 [00:00<00:03, 29.04it/s]
         percentage of images in human_files_short detecting a dog: 1
         100%|██████████| 100/100 [00:04<00:00, 22.96it/s]
         percentage of images in dog_files_short detecting a dog: 100
```

4. Preprocessing: We would resize and normalize the input images. We would also use some transforms like flip and rotate to augment the dataset.



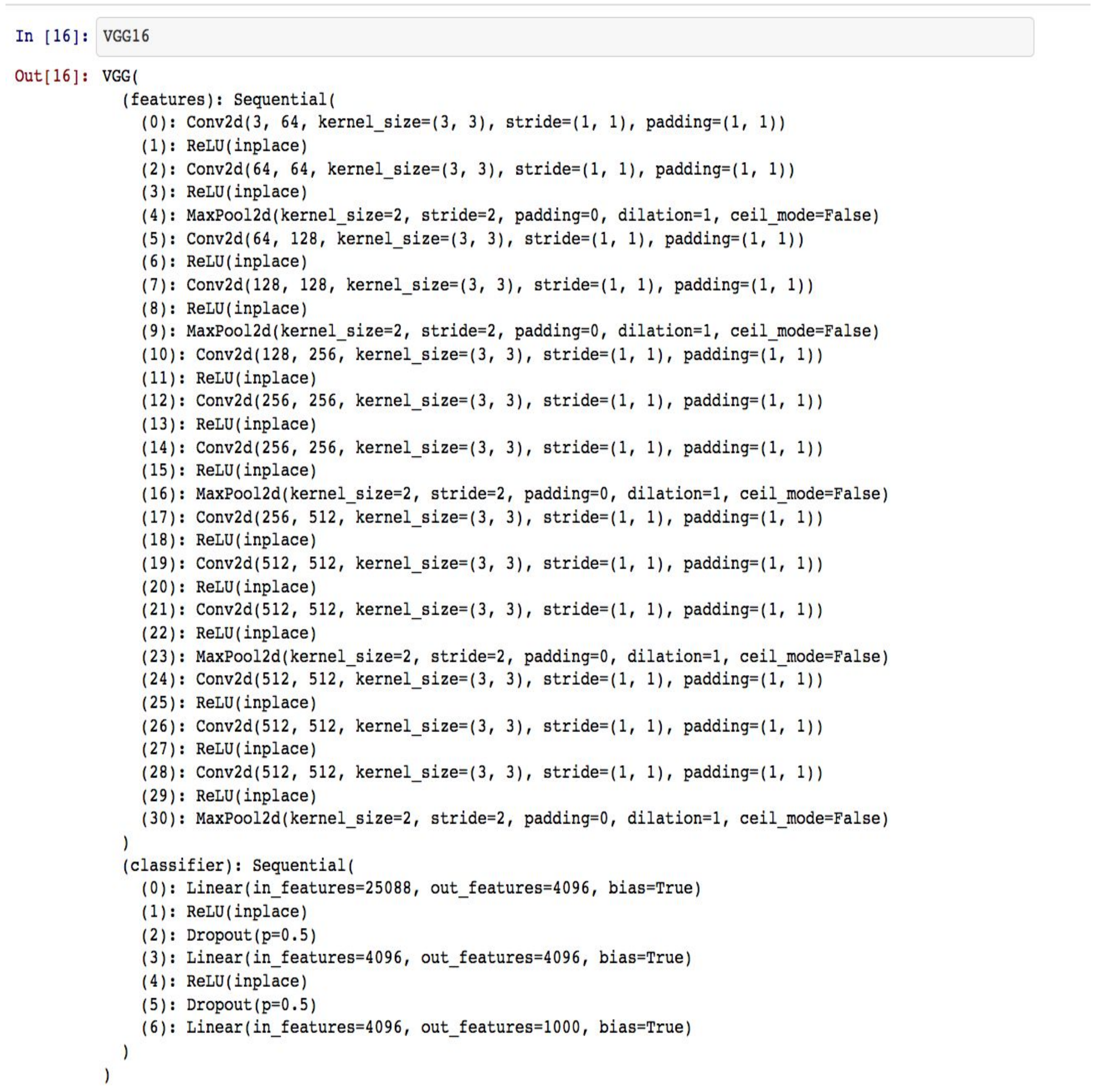
```
train_transform = transforms.Compose([
        transforms.Resize(255),
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])])
```

5. Create and train a CNN model from scratch to classify dog breeds. The architecture of the model:



```
In [20]: model_scratch

Out[20]: Net(
           (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (conv5): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (conv6): Conv2d(1024, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (conv7): Conv2d(2048, 4096, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
           (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
           (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
           (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
           (bn4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
           (bn5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
           (bn6): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
           (bn7): BatchNorm2d(4096, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
           (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
           (fc1): Linear(in_features=4096, out_features=2048, bias=True)
           (fc2): Linear(in_features=2048, out_features=133, bias=True)
           (dropout): Dropout(p=0.5)
         )
```

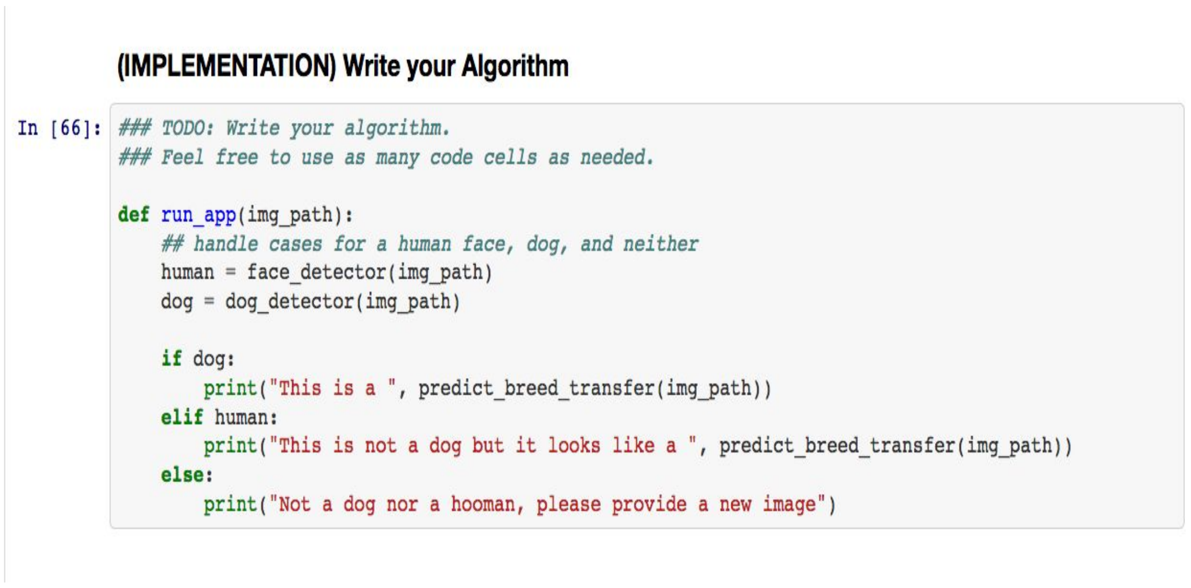
6. Use a pretrained model and finetune it on the dog training set to classify dog breeds. We have used a pretrained model of VGG16 which is trained on the Imagenet dataset.



```
In [16]: VGG16

Out[16]: VGG(
           (features): Sequential(
             (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (1): ReLU(inplace)
             (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (3): ReLU(inplace)
             (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
             (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (6): ReLU(inplace)
             (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (8): ReLU(inplace)
             (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
             (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (11): ReLU(inplace)
             (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (13): ReLU(inplace)
             (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (15): ReLU(inplace)
             (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
             (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (18): ReLU(inplace)
             (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (20): ReLU(inplace)
             (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (22): ReLU(inplace)
             (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
             (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (25): ReLU(inplace)
             (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (27): ReLU(inplace)
             (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
             (29): ReLU(inplace)
             (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
           )
           (classifier): Sequential(
             (0): Linear(in_features=25088, out_features=4096, bias=True)
             (1): ReLU(inplace)
             (2): Dropout(p=0.5)
             (3): Linear(in_features=4096, out_features=4096, bias=True)
             (4): ReLU(inplace)
             (5): Dropout(p=0.5)
             (6): Linear(in_features=4096, out_features=1000, bias=True)
           )
         )
```

We are only changing the last layer for this architecture so that the number of output nodes matches that of our problem(133). The weights of the model are frozen and are not trained except for the last layer.

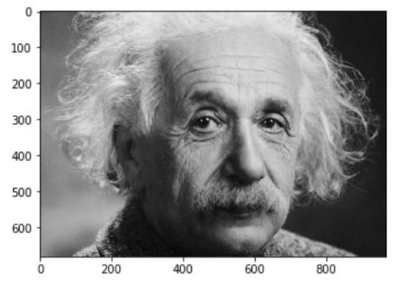
7. Algorithm accepts a file path to an image and first determines whether the image contains a human, dog, or neither.
    - If a **dog** is detected in the image, return the predicted breed.
    - If a **human** is detected in the image, return the resembling dog breed.
    - if **neither** is detected in the image, provide output that indicates an error.



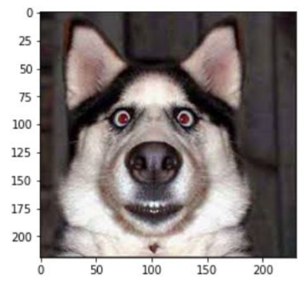
(IMPLEMENTATION) Write your Algorithm

```
In [66]: ### TODO: Write your algorithm.
         ### Feel free to use as many code cells as needed.

         def run_app(img_path):
             ## handle cases for a human face, dog, and neither
             human = face_detector(img_path)
             dog = dog_detector(img_path)

             if dog:
                 print("This is a ", predict_breed_transfer(img_path))
             elif human:
                 print("This is not a dog but it looks like a ", predict_breed_transfer(img_path))
             else:
                 print("Not a dog nor a hooman, please provide a new image")
```

8. The algorithm is tested on sample images. Some test results:



```
This is not a dog but it looks like a  Chinese crested
None
```



```
This is a  Alaskan malamute
None
```

Results

Model	Test Accuracy
CNN from scratch	36%
Transfer learning using VGG16	85%