Age Detection

Talluri Surya Teja

15EE35028

- I have manually chose data for classes (online standard datasets are of huge size... around 2GB)
- I chose 2 classes (child and adult) of age 2 and 20.
- each class have a data of around 30 images

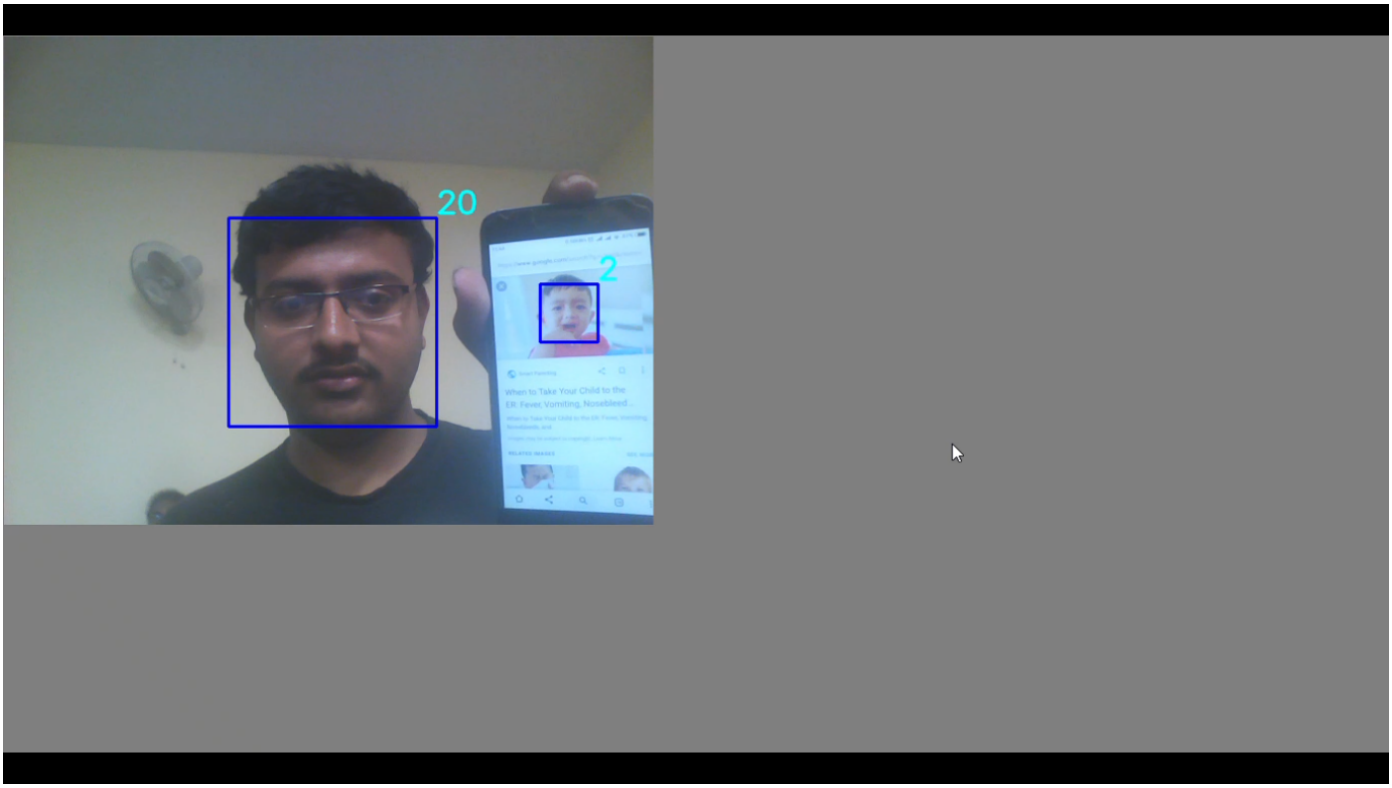
Preparing Data

- After getting images i generated face log for those images and saved them

Training Model

- I split this face logs into train and validation data
- I have extracted HOG features
- Using this features, I have trained a SVM model and acheived an accuracy of 100% on both Train and Validation

Real Time Testing



Code

I have used Python 3.7 version to write the code (.py file)

- After training the model I have used Pickle library to save the model and reload during test phase
- I have used arguments for the file
- For testing we have to run: python code_file.py --train "path to trained model"

Histogram of Oriented Gradients

In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions.

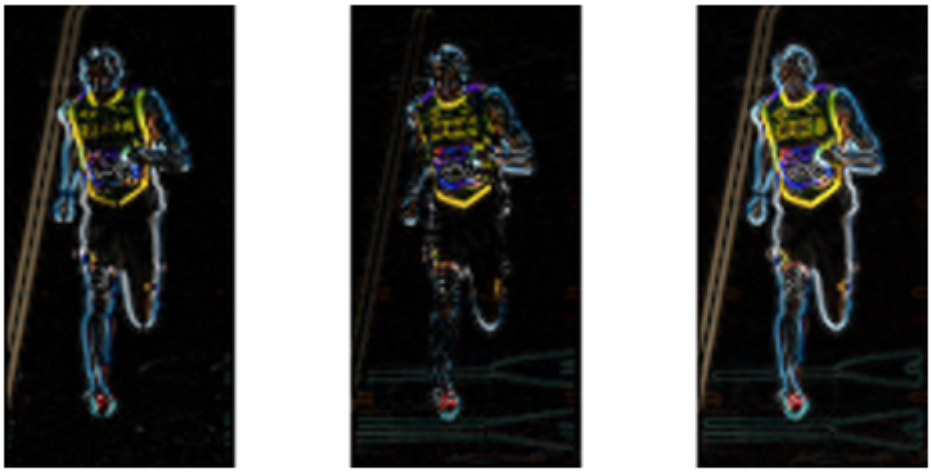
To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

```
[-1 0 1], [-1 0 1].T
```

Next, we can find the magnitude and direction of gradient using the following formula

$$g = \sqrt{g_x^2 + g_y^2}$$
$$\theta = \arctan \frac{g_y}{g_x}$$

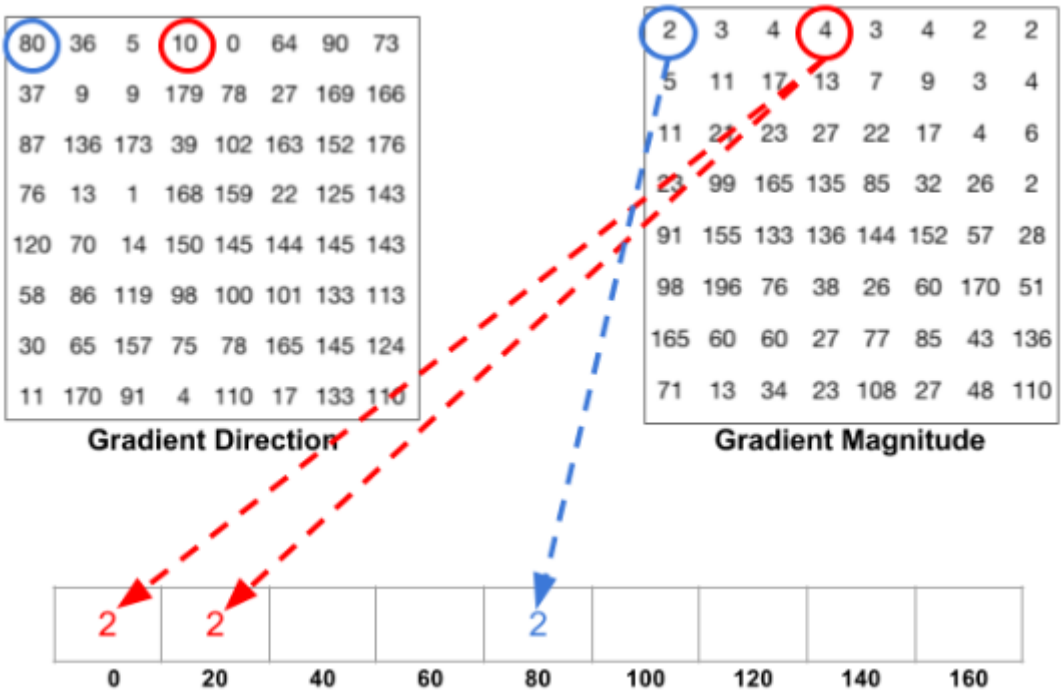
The figure below shoews gradient

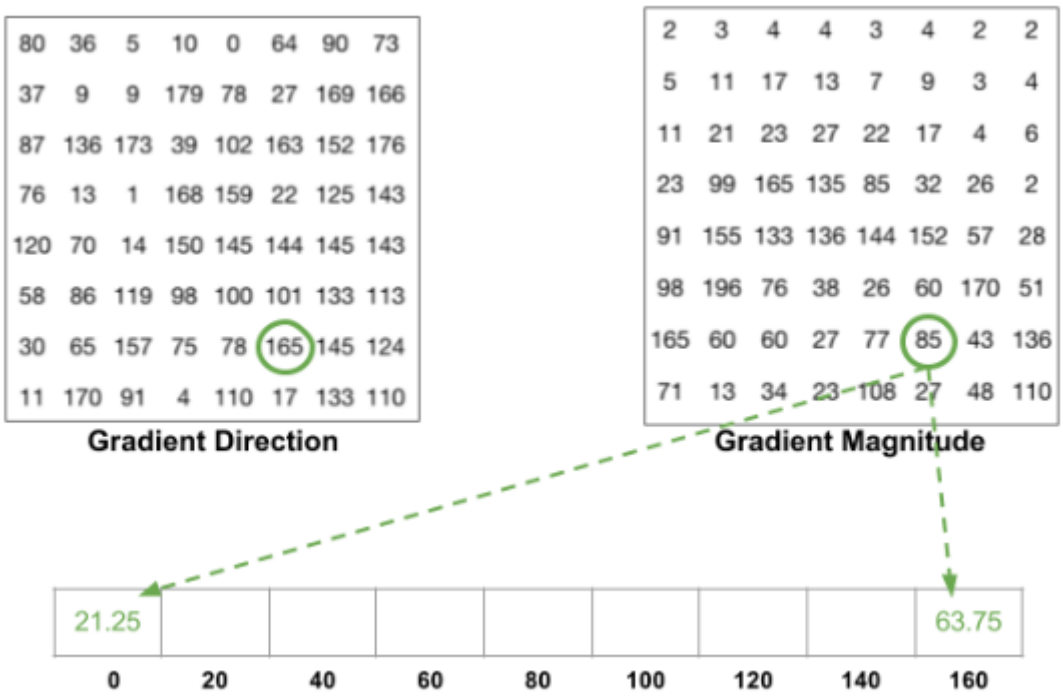


Left : Absolute value of x-gradient. Center : Absolute value of y-gradient.
Right : Magnitude of gradient.

the gradient angles are between 0 and 180 degrees instead of 0 to 360 degrees. These are called “unsigned” gradients because a gradient and it’s negative are represented by the same numbers.

Next we will calculate histograms, we will divide image into grids and divide each grid into bins. Based on angle value add magnitude to different bin





After that we will normalize the histograms by taking blocks which contains 4 or more small blocks. we would like to “normalize” the histogram so they are not affected by lighting variations.

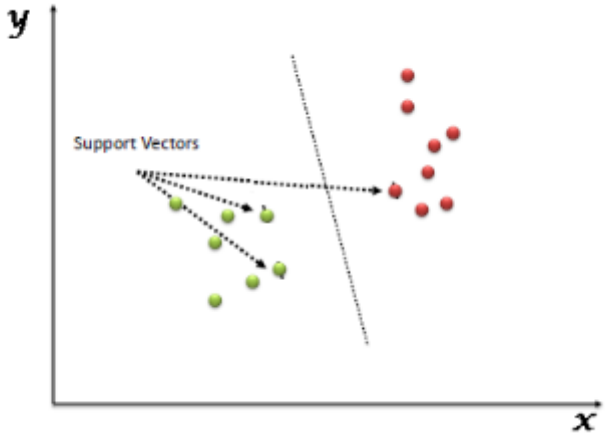
After the normalization, we will append them to generate final features.

- I have taken color image of size (150,150) for feature caluculation

Model Used

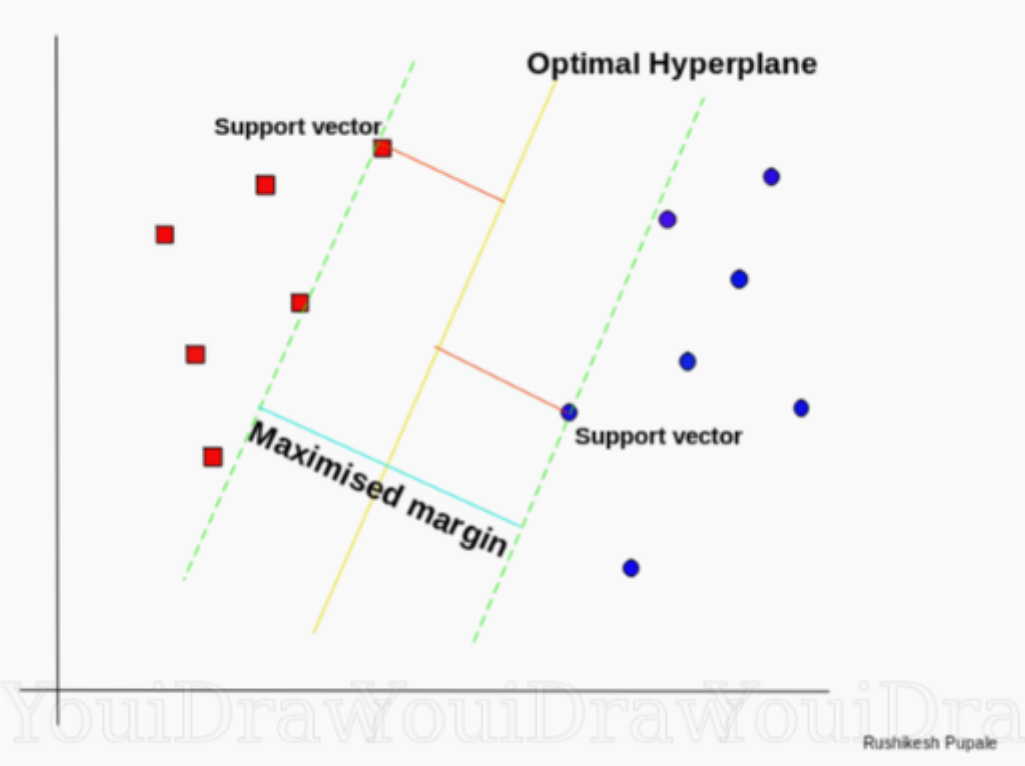
Support Vector Machine

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well



Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

According to the SVM algorithm we find the points closest to the line from both the classes.These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.



Optimal Hyperplane using the SVM algorithm

In [ ]:

```
import cv2
import os
from skimage import feature
import numpy as np
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
import argparse
import pickle as pkl

ap = argparse.ArgumentParser()
ap.add_argument("-model", "--test",
                help="saved model")
args = vars(ap.parse_args())

data_paths = [os.path.join(r, n) for r, _, f in os.walk('data') for n in f]

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

def detect(gray, frame, model, save = False):
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        age =  model.predict(hog(cv2.resize(gray[y:y+h, x:x+w], (150,150))).reshape(1,-1))
        cv2.putText(frame, age[0], (x+w,y-5), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,0), 2,
        if save:
            cv2.imwrite('data/'+str(i)+'.jpg', frame[y:y+h, x:x+w])
    return frame

def hog(image):
    if isinstance(image, str):
        image = cv2.imread(image)
        image = cv2.resize(image, (150,150))

    image = feature.hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(3,
                        visualize=False, visualise=None, transform_sqrt=True, feature_vecto
    return image


data = np.asarray([hog(i) for i in tqdm(data_paths)])
labels = np.asarray([i.split('\\')[1] for i in tqdm(data_paths)])

arr = np.random.permutation(len(labels))

data = data[arr]
labels = labels[arr]

x_train, x_test, y_train, y_test = train_test_split(data, labels, train_size = 0.7, random_

model = LinearSVC(random_state=42)
model.fit(x_train, y_train)
print('train accuracy: ', model.score(x_train, y_train))
print('test accuracy: ', model.score(x_test, y_test))
pkl.dump(model, open('svm_model_hog', 'wb'))

if args['test'] is not None:
    model = pkl.load(open(args['test'], 'rb'))#### Support Vector Machine
```

```
“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used
![image.png](attachment:image.png)

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machi

According to the SVM algorithm we find the points closest to the line from both the classes
![image.png](attachment:image.png)
    video_capture = cv2.VideoCapture(0)
    while True:
        _, frame = video_capture.read()

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        canvas = detect(gray, frame, model)
        cv2.imshow('Video', canvas)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    video_capture.release()
    cv2.destroyAllWindows()
```