Face Detection Using Viola-Jones Object Detection Framework

Akash Ghimire (12194814) BIjay Pariyar (12194945) Keshav Adhikari (12194874) Thapa Pradip (12194940)

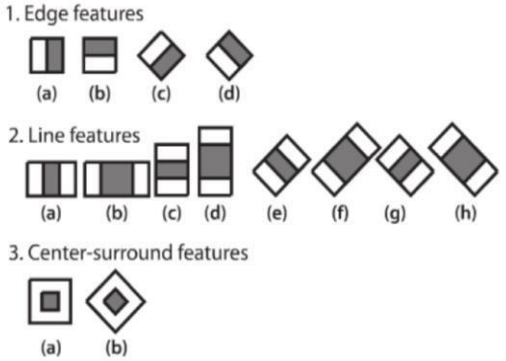
Project Objective:

In this project we implement the **face-based object detection algorithm** proposed by **Viola** and **Jones** (2001). This Project uses **Haar Cascade** technique along with **adaBoost** training system for face detection by taking hundreds of positive face images with respect to thousands of negative non-face images.

Face detection employs classifiers, which are algorithms that determine whether something in a picture is a face(1) or not a face (0). To improve accuracy, classifiers have been trained to recognize faces in tens of thousands to millions of photo. Haar Cascade is one of the most common classifiers in OpenCV.

1) Haar Cascade:

- The Haar Cascade classifier divides the pixels in the image into squares depending on their functions using the Haar Wavelet approach. The "features" that are detected are computed using "integral image" principles. The Adaboost learning algorithm is used by Haar Cascades, which picks a small number of crucial features from a huge set to get an effective output. Cascading techniques are then used by classifiers to recognize faces in images. Some Haar-Features are listed below.



Face Detection determines the locations and sizes of human faces in arbitrary (digital) images.

In **Face Recognition**, the use of Face Detection comes first to determine and isolate a face before it can be recognized.

Fig 1: Haar feature extraction kernels

2) Datasets:

There are two types of data set we have used for this project:

- Custom dataset
- Public dataset

a) Custom dataset:

- For custom dataset we have used Q51 13 megapixels camera, in order to maintain picture quality and capture several images per minute we have used photo burst application.
- For custom dataset we have divided each team members image dataset into 6 sub folders of 6 challenging aspects of object detection according to the variations and orientations and illumination conditions:

i) Viewpoint variation

ii) Deformation

iii) Occlusion

iv) Illumination conditions

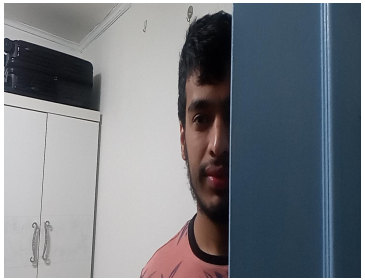
v) Cluttered or textured background &

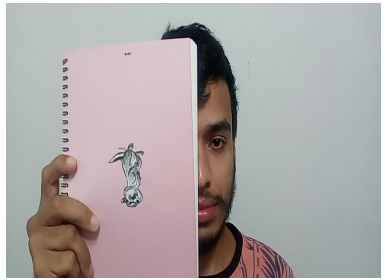
vi) Intra class variation

i) Viewpoint Variation

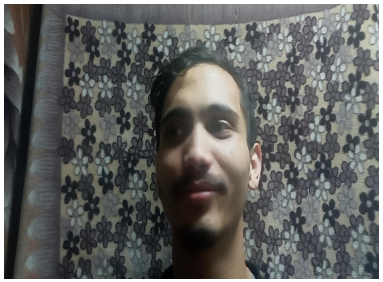


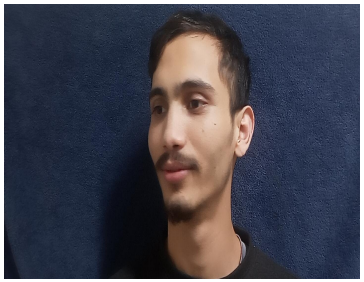

ii) Occlusion



iii) Cluttered or textured Background






iv) Illumination conditions




v) Intra-class variation

vi) Deformation



Custom Dataset:

- we have taken various wide range as well as short range images along with vertical orientations.

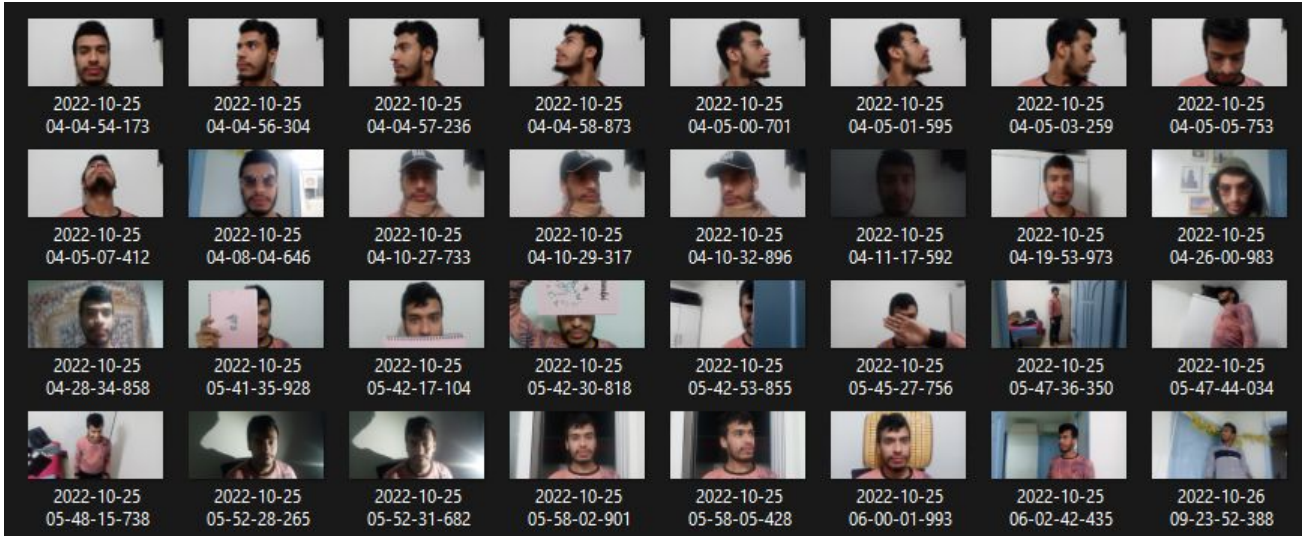


fig 2: Custom data-set with 6 different challenging aspects of object detection

Positive and Negative Images

- A positive image is one containing an object that must be detected.

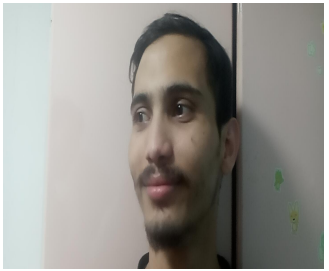



fig3: custom made positive images

- And a negative image is everything else which we don't have to detect.




fig4: custom made negative images

b) Public Dataset:

i) Positive Samples

- For positive samples we have used dataset from CelebA Dataset
- 10,177 of identities
- 202,599 Samples of Faces
- Variation: Eyeglasses, Hat, Bangs. Different emotion,and so on.

ii) Negative Sample

- For negative samples we took datasets from kaggle.
- Non-face datasets as in furniture, vegetables, notebooks etc., available in the surrounding.

b) Dataset size: Public and Custom Dataset

Custom Dataset				Public Dataset			
Train		Test		Train		Test	
p	n	p	n	p	n	p	n
776	3264	194	–	7368	32766	2298	–

3) Dataset Preprocessing

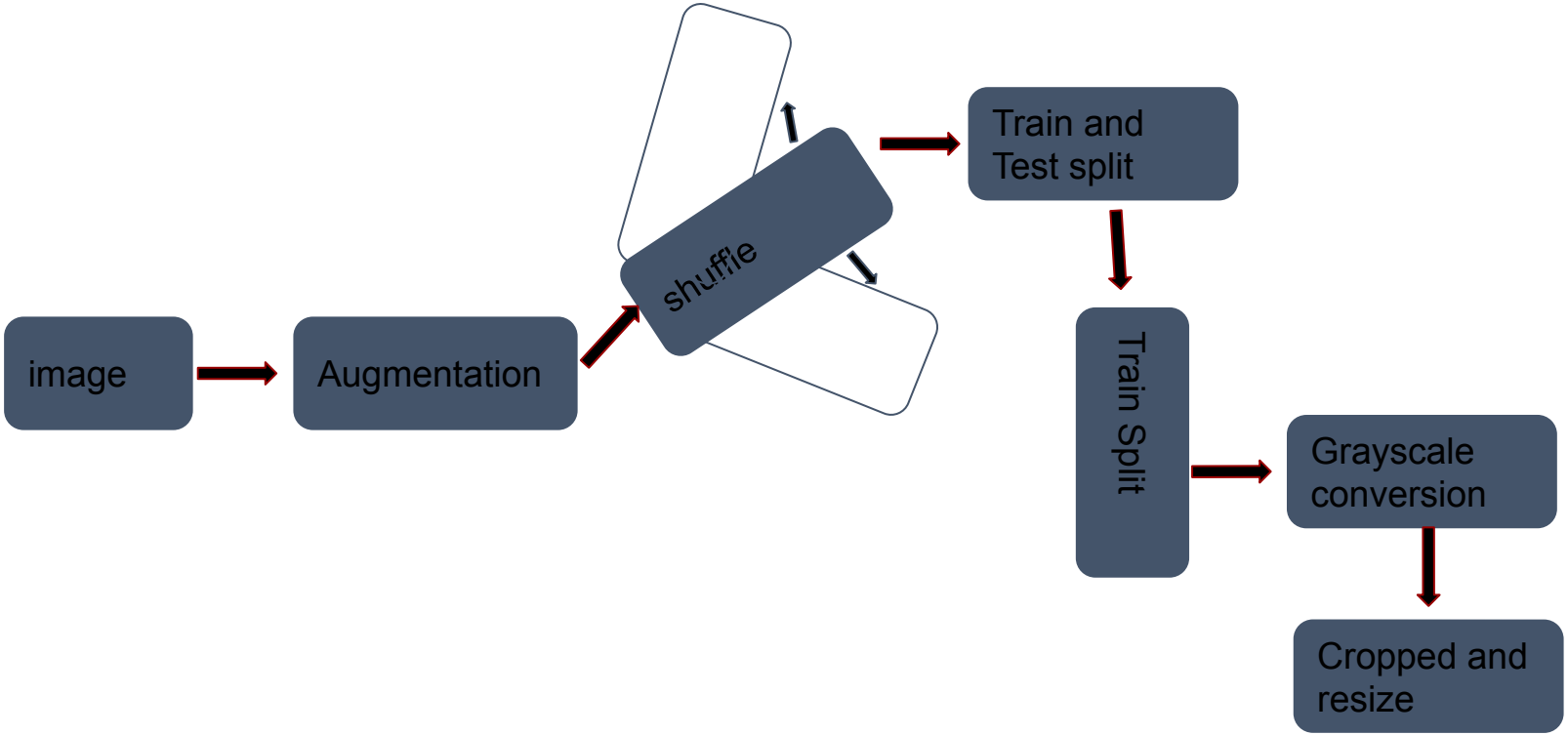


Fig 5: Custom Positive Dataset Preprocessing Pipeline

ii) Public Positive Dataset

Except Augmentation all the preprocessing step is same as that of Custom Positive Dataset preprocessing.

iii) Negative Samples

All the Negative samples were converted to gray scale and cropped to same size as that of positive training samples

4) Code Snippets for Positive Dataset Preprocessing

i) Augmentation

```
## augumentation
import albumentations as A
```

```
transform=A.Compose([A.HorizontalFlip(0.5),
A.Rotate(limit=35),
A.RandomBrightnessContrast(p=0.6),
A.ZoomBlur(max_factor=1.15,p=0.25)])
```

Each image is augmented to further increasing the training samples size as well as to add

ii) Conversion of RGB to Grayscale

```
## convert to Grayscale Image
def bgr2gray(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Convert image from RGB to Grayscale

iii) Shuffling the custom positive samples

```
##shuffling the custom postive samples
np.random.shuffle(custom_dire)
```

Before the train and test split of the dataset images are randomly shuffled for fair evaluation.

iv) Cropping

```
haar_cascade = cv2.CascadeClassifier('../haarcascade_frontalface_alt.xml')

faces_rect = haar_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=9)
for i,(x,y,w,h) in enumerate(faces_rect):
    crop=img[y:y+h,x:x+w,:]
    crop=cv2.resize(crop,(96,96))
```

pre trained haar cascade is used to **cropp the** faces from each sample positive images. Then each image is resized to (96,96) size. **Only train datasets are cropped not the test**

5) Code Snippet of test files

i) Testing custom cascade

src > test.py > ...

```python
import cv2

## loading haar cascade classifier
haar_cascade = cv2.CascadeClassifier('../cascade.xml')

## Detection of face from live camera

### test on camera
cap=cv2.VideoCapture(0)
while cap.isOpened(0):
    sucess,img=cap.read()
    if not sucess:
        break
    gray_img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces_rect = haar_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=9)
    for (x, y, w, h) in faces_rect:
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
    cv2.imshow('Detected Faces',img)
    if cv2.waitKey(1)==27:
        break
    cap.release()
    cv2.destroyAllWindows()
```

Test the custom cascade using webcam.

ii) Getting total number of faces within images folder

```python
### Getting the total number of faces within images folder

def test(classifier,dire):
    face_count=0
    start_time=time()
    img_detected=0
    for img_dire in dire:
        img=cv2.imread(img_dire)
        #img_name=img_dire.split('/')[-1].split('.')[0]
        gray_img=bgr2gray(img)
        faces_rect = classifier.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=9)
        if len(faces_rect)!=0:
            img_detected+=1
        for (x, y, w, h) in faces_rect:
            #print('ok')
            face_count+=1
    elapsed_time=time()-start_time
    return face_count,elapsed_time,img_detected

[15] ✓ 0.6s
```

Take classifier and image directory as parameter and return number of face detected, elapsed_time, and len(img) in which face was detected.

iii) Calculating the Accuracy of cascade classifier

```python
##Calculating the Accuracy of the cascade classifier.


def get_acc(g_t,pred): ## if cust_dataset g_t =number of faces detected by pretrained model.
                       #for public dataset, g_t=len(images) in a folder
    error=np.absolute(g_t-pred)
    error_per=(error/g_t)*100
    acc_per=(100-error_per)
    return acc_per

[9] ✓ 0.6s
```

Take g_t number of faces in images directory and pred number of faces by cascade to return accuracy.

iii) Calculating the precision time

```
## Calculating the precision time

def time_per_face(face_count,elapsed_time):
    return (elapsed_time/face_count)

def time_per_image(num_image,elapsed_time):
    return (elapsed_time/num_image)
```

Functions to calculate inference time.

iv) Calculating the minimum detectable size

```
def minFace(img_dires):
    min_siz=[]
    for img_dire in img_dires:
        img=cv2.imread(img_dire)
        gray_img=bgr2gray(img)
        test_arr=[1024,512,256,128,64,32,16,4,2,1][::-1]
        for siz in test_arr:
            faces_rect = haar_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=9)
            test_faces_rect=haar_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=9,minSize=(siz,siz)) ## size of object less tha minSize are ignored.
            if len(faces_rect)!=0 and len(test_faces_rect)==0:
                min_siz.append(siz)

    return min(min_siz)
```

function to calculate the minimum detectable size of image.

- Parameter: img dires list
- return min_siz

v) Calculating Maximum detectable size.

```python
def maxFace(img_dires):
    max_siz=[]
    for img_dire in img_dires:
        img=cv2.imread(img_dire)
        gray_img=bgr2gray(img)
        test_arr=[1024,512,256,128,64,32,16,4,2,1]
        for siz in test_arr:
            faces_rect = haar_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=9)
            test_faces_rect=haar_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=9,maxSize=(siz,siz)) ## size of object more than minSize are ignored.
            if len(faces_rect)!=0 and len(test_faces_rect)==0:
                max_siz.append(siz)

    return max(max_siz)

```

function to calculate the maximum detectable size of image.

- Parameter: img dires list
- return max_siz

6) Test Evaluation & Conclusion

Test conclusion	Custom Dataset	Celeba Dataset	Prof. Dataset
Minimum detectable size	(64,64)	(64,64)	— — —
Maximum detectable size	(512,512)		— — —
Accuracy	30 %	73.41%	— — —
Inference time per Face(sec)	0.132	0.0026	— — —
Inference time per Image(sec)	0.125	0.0026	— — —

fig : test evaluation table

7) Experimental Analysis: Custom Dataset vs Public Dataset

1. Accuracy Result on test custom dataset is 32% compared to accuracy result on test dataset which is around 73 %. Some reasons for this are:

    a. Large percentage of our testing custom dataset are complex: Different ranges of illumination, intra-class variation, Point view variation, and so on.

    b. Custom training dataset was much smaller compared to public training dataset.

8) Conclusion

1. Advantage

    a. Does not require a lot of resources like other object detection like Yolo,SSD,RCNN and so on.

2. Disadvantage

    a. Current SOTA object detection like Yolo method provide better accurate results.

9) Team Member Contribution:

Team UNO	Team Members
Contribution- Haar Cascade classifier training, evaluation & testing.	Akash Ghimire (12194814) – Team Leader
Contribution- Data-set collection and report writing.	Bijay Pariyar (12194945)- Team Member
Contribution- GitHub related work and management.	Keshav Adhikari (12194874)- Team Member
Contribution- Environment setup and research.	Pradip Thapa (12194940)- Facilitator