VIOLA JONES FACE DETECTION

RAKHIMOV RUSTAM | MUNKHTULGA IRMUUN | KHAITOV SARDOR

❖OBJECTIVES

- To train, test and evaluate the performance of a HAAR cascade for face detection.
- Prepare an appropriate dataset.
- Understand GUI trainer configurations and useful fixes to errors.
- Understand the python code and how to configure it for best results.

❖CONTRIBUTIONS

- ✓ Rustam – Adjusting GUI settings to train the model.
- ✓ Rustam – Writing and adjusting python code to make the best of the available model.
- ✓ Irmuun – Helped Gather datasets and evaluate the model.
- ✓ Sardorbek – tried training and evaluating the model.

❖TRAINING THE MODEL USING GUI

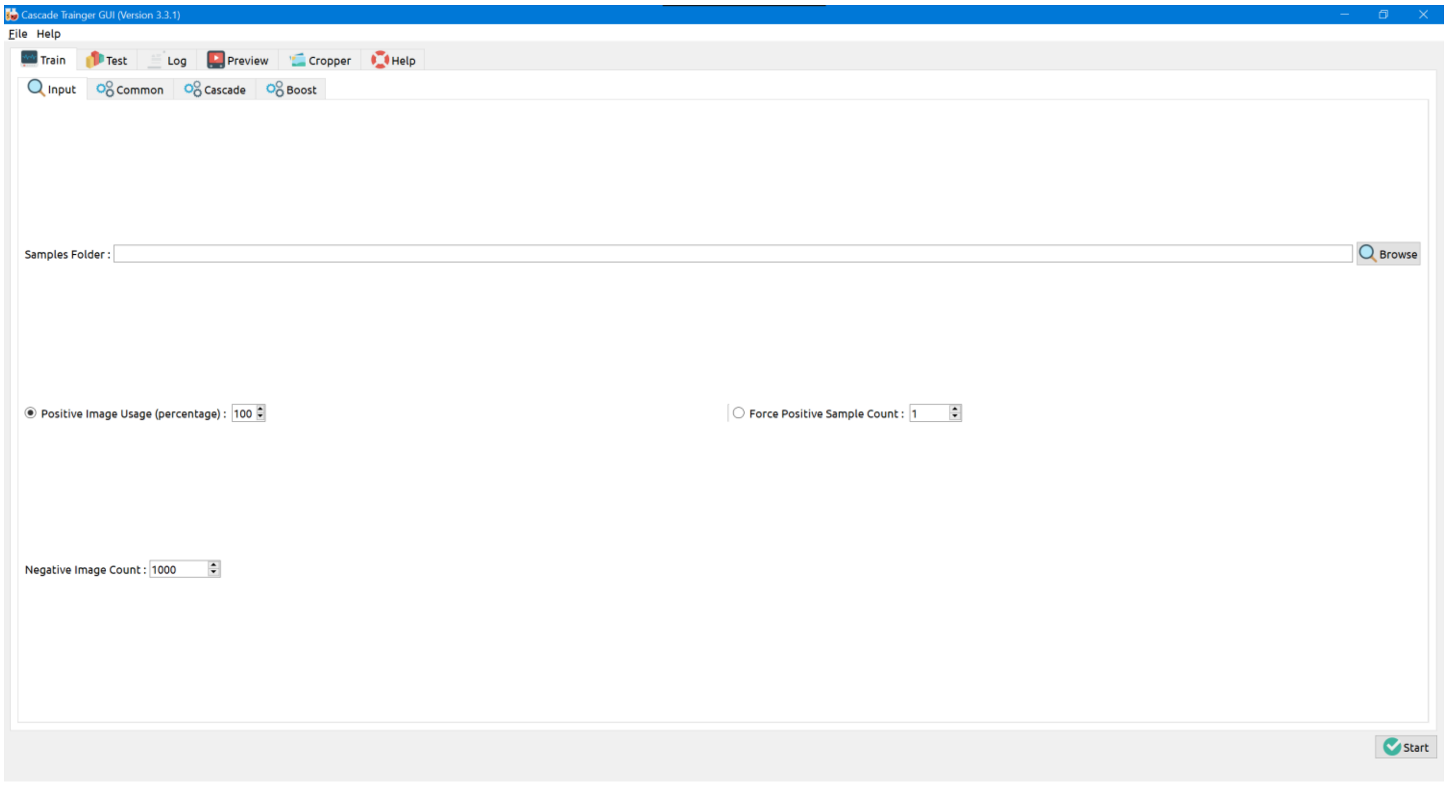
This section will go over what was explained in the demo file to clarify the use of GUI settings.

As can be seen in the image below, there are 3 options before training the model. We fill focus on the 2 located to the left side of the screen as they help quickly, albeit not as efficiently train the model.

The first option, Positive Image Usage decides how much of the p (positive) images are used when training the mode, 100 being all of them and a low number indicating a lower bar.

The second option, Negative Image Count, tells the program how many images in the n (negative) folder have to be used.

Force Positive Sample count basically forces the app to use a certain amount of positive images, regardless of percentile.

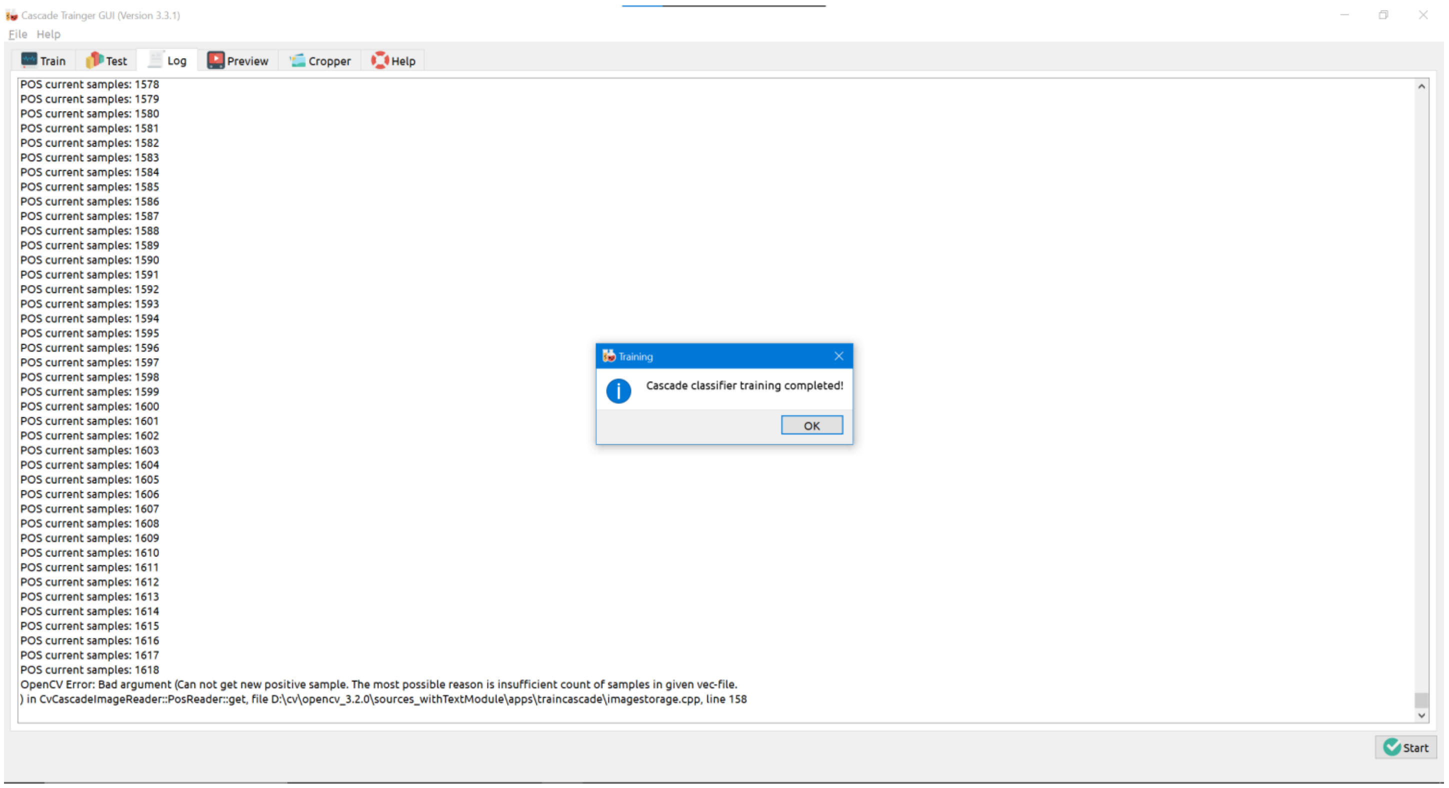

Cascade Trainger GUI (Version 3.3.1)
File Help
Train Test Log Preview Cropper Help
Input Common Cascade Boost
Browse
Start

Samples Folder :

[● Positive Image Usage (percentage) : 100]

○ Force Positive Sample Count : 1

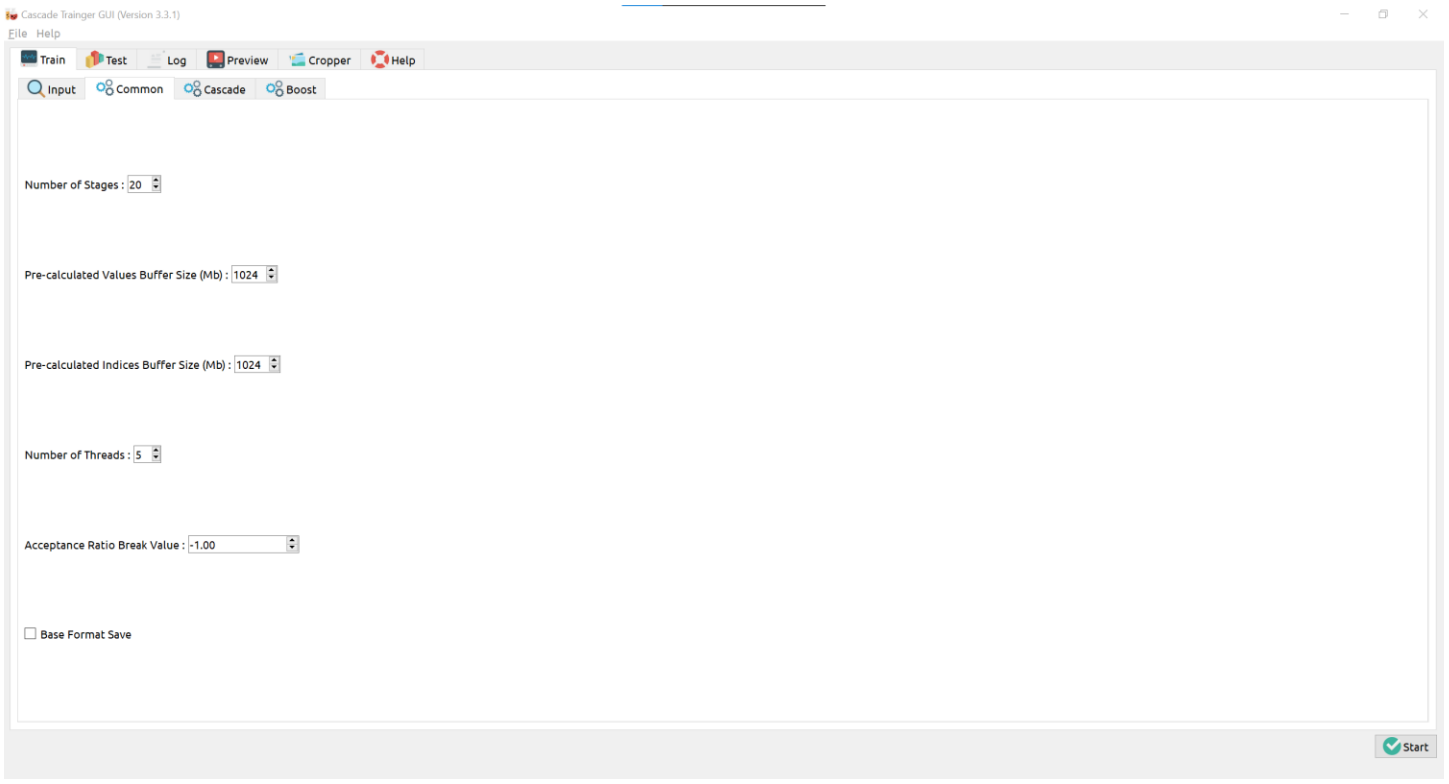
Negative Image Count : 1000


Cascade Trainger GUI (Version 3.3.1)
File Help
Train
Test
Log
Preview
Cropper
Help
Training
Cascade classifier training completed!
OK
Start

```
POS current samples: 1578
POS current samples: 1579
POS current samples: 1580
POS current samples: 1581
POS current samples: 1582
POS current samples: 1583
POS current samples: 1584
POS current samples: 1585
POS current samples: 1586
POS current samples: 1587
POS current samples: 1588
POS current samples: 1589
POS current samples: 1590
POS current samples: 1591
POS current samples: 1592
POS current samples: 1593
POS current samples: 1594
POS current samples: 1595
POS current samples: 1596
POS current samples: 1597
POS current samples: 1598
POS current samples: 1599
POS current samples: 1600
POS current samples: 1601
POS current samples: 1602
POS current samples: 1603
POS current samples: 1604
POS current samples: 1605
POS current samples: 1606
POS current samples: 1607
POS current samples: 1608
POS current samples: 1609
POS current samples: 1610
POS current samples: 1611
POS current samples: 1612
POS current samples: 1613
POS current samples: 1614
POS current samples: 1615
POS current samples: 1616
POS current samples: 1617
POS current samples: 1618
OpenCV Error: Bad argument (Can not get new positive sample. The most possible reason is insufficient count of samples in given vec-file.
) in CvCascadeImageReader::PosReader::get, file D:\cv\opencv_3.2.0\sources_withTextModule\apps\traincascade\imagestorage.cpp, line 158
```

As you can see in the image above, the training ended with an error. The error states cannot get new positive samples, i.e. insufficient positive samples. A fix we have found for this error is to slightly reduce the Positive Image Usage percentile. It is possible that this error is caused when the program is not able to use some images.

Otherwise error, False Alarm can be dealt with by decreasing the negative images used.



Number of Stages : 20

Pre-calculated Values Buffer Size (Mb) : 1024

Pre-calculated Indices Buffer Size (Mb) : 1024

Number of Threads : 5

Acceptance Ratio Break Value : -1.00

☐ Base Format Save

The image above has quiet a few options, however, we only modified some.

When training a large dataset we have decided that it is best to start with a few stages to test for errors and after fixing said errors to increase the stages until a desired model is ready.

As the pc has 16gb ram and good specs, we used 5000 for both Buffer size values and indices to speed-up the training process.

Number of threads and Break Value were not modified at any point.

Once the model has been trained, we use the code below to test the model via camera.

```
import cv2
import time

classifier_path = r'C:\Users\rrakh\Desktop\tester\classifier\cascade.xml'
face_cascade = cv2.CascadeClassifier(classifier_path)

# Initialize the camera
cap = cv2.VideoCapture(0)

# Initialize variables to track time and size metrics
total_time = 0
total_frames = 0
min_face_size = float('inf')
max_face_size = 0

while True:
    ret, frame = cap.read()

    if not ret:
        break

    # Convert the frame to grayscale for face detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Record the start time
    start_time = time.time()

    # Perform face detection
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.01, minNeighbors=5,
minSize=(100, 100))

    # Calculate the time taken for face detection
    end_time = time.time()
    elapsed_time = end_time - start_time

    # Update metrics
    total_time += elapsed_time
    total_frames += 1

    # Update min and max face size metrics
    for (x, y, w, h) in faces:
        face_size = w * h
        min_face_size = min(min_face_size, face_size)
        max_face_size = max(max_face_size, face_size)

    # Draw rectangles around detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the video feed with detected faces
    cv2.imshow('Face Detection', frame)

    # Break the loop if the 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Calculate the average time for face detection
average_time = total_time / total_frames
```

```
# Release the camera and close the OpenCV windows
cap.release()
cv2.destroyAllWindows()

print(f"Average time for face detection: {average_time} seconds")
print(f"Minimum detectable face size: {min_face_size}")
print(f"Maximum detectable face size: {max_face_size}")
```

In the code above, the most important line is

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.01, minNeighbors=5, minSize=(100, 100))
```

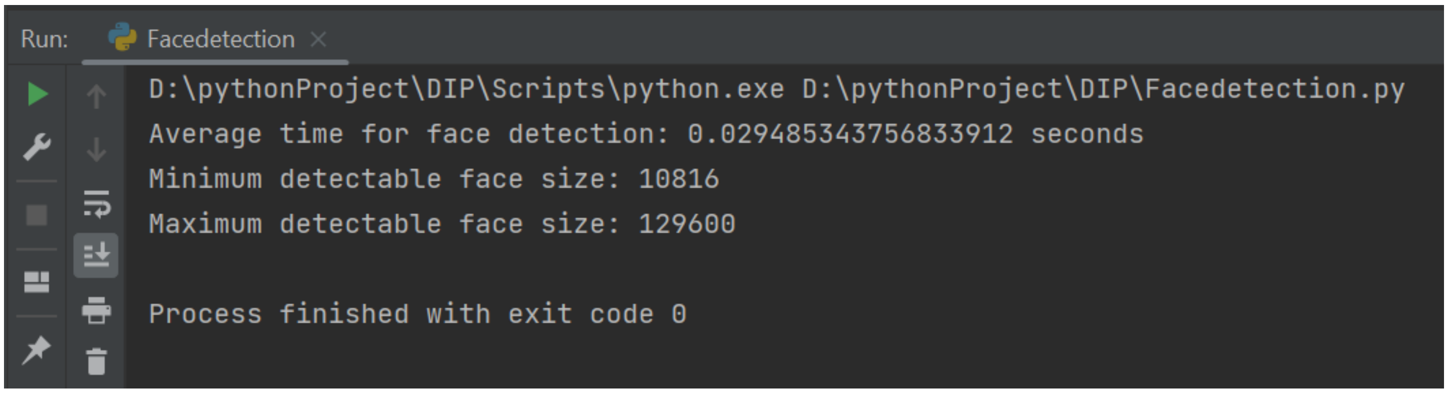
Scale factor decides how in-depth the model works and how much the image is resized, minNeighbors is used to determine how string the model is and minSize is the minimum object size.

The output once the "q" is pressed or the program is manually stopped will contain information on min/max size as well as how much time it takes for the model to detect a face.

❖ Accuracy Analysis

- Viewpoint variation – little to no effect
- Deformation Occlusion – little effect
- Illumination conditions – little to no effect
- Cluttered or textured Background – little effect
- Intra-class variation – little to no effect

The analysis below can be seen in the demo file for further reference.



```
D:\pythonProject\DIP\Scripts\python.exe D:\pythonProject\DIP\Facedetection.py
Average time for face detection: 0.029485343756833912 seconds
Minimum detectable face size: 10816
Maximum detectable face size: 129600

Process finished with exit code 0
```