

|  |
| --- |
| 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 – Adjusting python code to make the best of the available 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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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.

A screen shot of a computer

Description automatically generated