به نام خدا

آزمایشگاه برنامه نویسی پایتون

پروژه نهایی آزمایشگاه پایتون

گروه نهم

اعضای گروه:

مرتضی علیخانی

سروش عاشوری صفت

کیخسرو خسروانی

علیرضا شیرخدایی

تیر ماه 1400

مقدمه ای بر گزارش پروژه:

فایل پیوست شده همراه این گزارش شامل پوشه ای به نام flask_web است که در آن تمامی محتویات پروژه که کد های پایتون و مجموعه ای از عکس ها برای تست و پردازش کد و فایل داکر شده برای build کردن آن در داکر و همچنین دو ویدیو که یکی اجرای نشان دادن اینکه کد ها را در داکر آورده ایم و دیگری تست کردن کد و خروجی گرفتن از آن در محیط داکر است آورده شده است

کد ها کامنت گذاری شده اند و توضیحات کامل تر آن ها در این گزارش کار که در ادامه خواهیم دید آمده است.

هم چنین یک git برای این پروژه ساخته ایم و فایل ها را در آنجا هم قرار داده ایم به این آدرس
https://github.com/AlirezaShirkhodaee/docker-main

با تشکر

اعضای گروه نهم

مرتضی علیخانی

سروش عاشوری صفت

کیخسرو خسروانی

علیرضا شیرخدایی

شرح پروژه:

به صورت خلاصه هدف از این پروژه ساخت یک سرور است که روی داکر و ip local یا global ایجاد شود عکس های داخل یک آدرس را لیست کند و اگر از او عکسی خواستیم ببیند که آیا در لیستش هست یا نه.

اگر بود سریال شده عکس را خروجی بدهد و اگر نبود بگوید که عکس موجود نیست.

هم چنین در حالت امتیازی اگر عکس وجود داشت یک face detection رو عکس انجام دهد و تشخیص بدهد آیا صورت انسان هم در عکس هست یا خیر.

برای این کار دو کد زده ایم با نام های client و api

Client.py:

در کد client اینکه کدام عکس را می خواهیم از سرور بپرسیم و آیا face detection می خواهیم انجام شود یا نه و اینکه فایل دریافتی از سرور را کجا ذخیره کند و اینکه ip و port سرور کدام است را مشخص می کنیم سپس در خواست را در غالب post به سرور می زنیم و با توجه به جوابی که سرور بر می گرداند اگر تصویر نبود پیام : Image not found in the server

اگر بود ولی به مشکل خورد((اتصال درست نداشتن در docker hub)) بیام روبرو را می دهد : There is an error in face detection

در غیر این صورت تعداد تصویر ها را به صورت Number of face detected is خروجی می دهد و تصویر را از سرور دانلود کرده و در آدرسی که گفته بودیم ذخیره می کند.

در زیر کد client را مشاهده می کنیم:

```python
import requests
import json
import argparse
import codecs

parser = argparse.ArgumentParser()
parser.add_argument('--image', default='pic10.jpeg', type=str)
parser.add_argument('--face_detection', default=False, action='store_true')
parser.add_argument('--output_file', default='output.png', type=str)
parser.add_argument('--server_ip', default='localhost', type=str)
parser.add_argument('--server_port', default=5000, type=int)
args = parser.parse_args()

data = {'image': args.image}
if args.face_detection:
    res = requests.post(f'http://{args.server_ip}:{str(args.server_port)}/face_detection_api', data=json.dumps(data))
    res = json.loads(res.text)
    if res['message'] == 'Image Not Found':
        print('Image not found in the server.')
    elif res['face detection'] == "Error In Face Detection":
        print("There is an error in face detection.")
    else:
        face_dict = json.loads(res['face detection'])
        print(f"Number of face detected is {face_dict['faces_count']}")
else:
    res = requests.post(f'http://{args.server_ip}:{str(args.server_port)}/image_processing_api', data=json.dumps(data))
    if res.text == 'Image Not Found':
        print('Image not found in the server.')
    else:
        res = res.content
        with open(args.output_file, "wb") as f:
            f.write(codecs.decode(res, 'base64'))
            print(f'Image saved as {args.output_file}')
```

حال به بررسی هر بخش این کد می پردازیم:

```
parser = argparse.ArgumentParser()
parser.add_argument('--image', default='pic10.jpeg', type=str)
parser.add_argument('--face_detection', default=False, action='store_true')
parser.add_argument('--output_file', default='output.png', type=str)
parser.add_argument('--server_ip', default='localhost', type=str)
parser.add_argument('--server_port', default=5000, type=int)
args = parser.parse_args()
```

در ابتدا پارامتر هایی را از کاربر می گیرد که به ترتیب زده شده در بالا : نام عکس ، آیا می خواهیم تشخیص چهره هم انجام شود یا خیر ، اگر فایلی خواست از سرور دانلود کند آن را کجا ذخیره کند ، ip سروری که می خواهد به آن request بدهد ، port آن سرور.

```
data = {'image': args.image}
if args.face_detection:
    res = requests.post(f'http://{args.server_ip}:{str(args.server_port)}/face_detection_api',
data=json.dumps(data))
    res = json.loads(res.text)
```

حال آن ها را با نام data قرار می دهد.

سپس وارد قسمت ارسال request می شود

اگر می خواهیم که روی آن face detection انجام شود data را در قالب json و به صورت post request به آن سرور ip و آن port و به api مقابل ارسال می کند /face_detection_api

سپس محتویات باز گشته از سرور در قالب json را در res می ریزد.

```
if res['message'] == 'Image Not Found':
    print('Image not found in the server.')
  elif res['face detection'] == "Error In Face Detection":
    print("There is an error in face detection.")
  else:
    face_dict = json.loads(res['face detection'])
    print(f"Number of face detected is {face_dict['faces_count']}")
```

اگر محتوی بیام داخل آن image Not Found بود متن Image not found in the server. را به کاربر نشان می دهد.

در غیر این صورت ((حالتی که تصویر بود)) اگر محتوی بیام Error In Face Detection بود پیام there is an error in face detection را نشان می دهد.

و در غیر این صورت محتویات res['face detection'] را در دیکشنری face_dict ریخته و پیام

Number of face detected is (#) را نمایش می دهد.

```
else:
  res = requests.post(f'http://{args.server_ip}:{str(args.server_port)}/image_processing_api',
data=json.dumps(data))
  if res.text == 'Image Not Found':
    print('Image not found in the server.')
  else:
    res = res.content
    with open(args.output_file, "wb") as f:
      f.write(codecs.decode(res, 'base64'))
      print(f'Image saved as {args.output_file}')
```

اگر هم arg.face_detection مقدار false داشت ((کاربر نمی خواست پردازش صورت انجام شود))

data را در قالب json و به صورت post request به آن سرور ip و آن port و به api مقابل ارسال می کند /image_processing_api

حال اگر باسخ داده شده از سرور عبارت Image Not Found بود پیام image not found in the server را نشان می دهد و در غیر این صورت محتویات res که سریال شده عکس در خواستی از سرور بود که سرور در res برای او فرستاد را در فایلی که در ابتدا کاربر گفته بود خروجی را در آن ذخیره کند قرار می دهد و پیام image saved as {address} را نشان می دهد.

api.py :

این کد به طور خلاصه سرور است .

که ابتدا روی ip و port دلخواه ما بالا می آید و هر وقت client یک request را به یک api در آن ip و port فرستاد بردازش های لازم را انجام می دهد و خروجی را در قالب json به client بر می گرداند.

بس کل مسؤلیت لیست کردن عکس ها و گشتن برای عکس و سریال کردن آن و face detection در اینجا انجام می شود.

در زیر کد api آمده است.

```python
from flask import Flask, render_template, request
import json
import base64
from glob import glob
import os
import cv2

server_ip = os.getenv("server_ip", default='0.0.0.0')
server_port = int(os.getenv("server_port", default='5000'))
volume = os.getenv("volume_address", default='Volume')
cascPath = os.getenv("cascade", default='haarcascade_frontalface_default.xml')
debug = int(os.getenv("dubug", default='0'))

faceCascade = cv2.CascadeClassifier(cascPath)

image_formats = ['.png', '.jpg', '.jpeg', '.gif']
image_list = []
for image_format in image_formats:
    image_list.extend(glob(volume + os.sep + '*' + image_format))

app = Flask(__name__)
app.config["DEBUG"] = (debug == 1)


def serialize_image(image):
    with open(image, "rb") as imageFile:
        str = base64.b64encode(imageFile.read())
    return str


def face_detection(image):
    try:
        face_dict = os.popen(f'curl -X POST -F "file=@{image}" http://localhost:8080/').read()
    except:
        face_dict = "Error In Face Detection"
    return face_dict
```

```python
def face_detection(image):
    try:
        face_dict = os.popen(f'curl -X POST -F "file=@{image}" http://localhost:8080/').read()
    except:
        face_dict = "Error In Face Detection"
    return face_dict


def face_detector_tool(image):
    image_address = volume + os.sep + image
    img = cv2.imread(image_address)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE
    )

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

    filename = image[0:image.rfind('.')] + "_face_detected" + image[image.rfind('.'):]
    cv2.imwrite(volume + os.sep + filename, img)
    image_list.append(volume + os.sep + filename)
    return filename
```

```python
@app.route('/image_processing', methods=['GET', 'POST'])
def home():
    post_request = False
    image_exists = False
    image = ''
    if request.method == 'POST':
        post_request = True
        FD_option = request.form['FaceDetection']
        if FD_option == 'yes':
            FD_option = True
        else:
            FD_option = False
        image = request.form.get('image')
        image_address = volume + os.sep + image
        if image_address in image_list:
            image_exists = True
            if FD_option:
                image = face_detector_tool(image)
        else:
            image_exists = False
    return render_template("index.html", post_request=post_request, image_exists=image_exists, image=image)


@app.route('/image_processing_api', methods=['POST'])
def rest_api_image_processing():
    data = json.loads(request.data)
    image = volume + os.sep + data['image']
    if image in image_list:
        response = serialize_image(image)
    else:
        response = 'Image Not Found'
    return response
```

```python
@app.route('/face_detection_api', methods=['POST'])
def rest_api_face_detection():
    data = json.loads(request.data)
    response = {}
    image = volume + os.sep + data['image']
    if image in image_list:
        response['message'] = 'Image Found'
        response['face detection'] = face_detection(image)
    else:
        response['message'] = 'Image Not Found'
        response['face detection'] = None
    return json.dumps(response)


app.run(port=server_port, host=server_ip)
```

حال به بررسی هر تابع جداگانه می بردازیم:

```
from flask import Flask, render_template, request
import json
import base64
from glob import glob
import os
import cv2

server_ip = os.getenv("server_ip", default='0.0.0.0')
server_port = int(os.getenv("server_port", default='5000'))
volume = os.getenv("volume_address", default='Volume')
cascPath = os.getenv("cascade", default='haarcascade_frontalface_default.xml')
debug = int(os.getenv("dubug", default='0'))

faceCascade = cv2.CascadeClassifier(cascPath)
```

در ابتدا کتابخانه های لازم را فراخوانی می کنیم و برای ابراتور ip و port مورد نظرمان را در خواست می کنیم و همچنین آدرس بوشه volume که عکس ها در آن هست را می دهیم و یک مدل بیش ساخته xml که برای تشخیص چهره هست را فراخوانی می کنیم و در casspath قرار می دهیم و سپس با کمک opencv 2 و مدلcascade face detection ocv آن مدل را می خوانیم و نام آن را facecascade می گذاریم

((این فایل با نام haarcascade_frontalface_default.xml جزو فایل های ضمیمه شده قرار دارد))

```
image_formats = ['.png', '.jpg', '.jpeg', '.gif']
image_list = []
for image_format in image_formats:
   image_list.extend(glob(volume + os.sep + '*' + image_format))


app = Flask(__name__)
app.config["DEBUG"] = (debug == 1)
```

سپس بسوند تمام فرمت های تصویر را در یک آرایه قرار دادیم و سپس تمام تصاویر داخل بوشه volume را در image_list لیست می کنیم

((بخش debug را صرفا در مراحل کد زنی برای بیدا کردن باگ ها قرار دادیم))

```
def serialize_image(image):
   with open(image, "rb") as imageFile:
      str = base64.b64encode(imageFile.read())
   return str
```

این تابع عکس را گرفته و سریال شده عکس را باز می گرداند.

```
def face_detection(image):
    try:
        face_dict = os.popen(f'curl -X POST -F "file=@{image}" http://localhost:8080/').read()
    except:
        face_dict = "Error In Face Detection"
    return face_dict
```

این تابع آدرس عکس را می گیرد و یک post request به یک image آماده ای از docker ((از docker hub گرفته ایم و کار آن پیدا کردن تعداد صورت ها در عکس هست)) که موازی این image روی local ip بالا آورده ایم می فرستد که اگر مشکل در اتصال داکر ها باشد پیام error in face detection می دهد و در غیر این صورت یک دیکشنری محتوی تعداد صورت های در عکس و مختصات آن ها می دهد.

((این دیکشنری آماده ضمیمه شده است و نام آن بوشه face_detection است))

```
def face_detector_tool(image):
    image_address = volume + os.sep + image
    img = cv2.imread(image_address)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE
    )

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

    filename = image[0:image.rfind('.')] + "_face_detected" + image[image.rfind('.'):]
    cv2.imwrite(volume + os.sep + filename, img)
    image_list.append(volume + os.sep + filename)
    return filename
```

در این تابع ما تصویر را ابتدا با cv2 می خوانیم و سپس سیاه و سفید کرده و به کمک مدل xml ای که وارد کرده بودیم ((faceCascade)) مختصات صورت ها را در عکس بیدا می کنیم و سپس دور هر کدام یک مستطیل رسم می کنیم و عکس جدید را به لیست عکس ها اضافه می کنیم و اسم عکس جدید را به عنوان خروجی می دهیم.

```
@app.route('/image_processing', methods=['GET', 'POST'])
def home():
    post_request = False
    image_exists = False
    image = ''
    if request.method == 'POST':
        post_request = True
        FD_option = request.form['FaceDetection']
        if FD_option == 'yes':
            FD_option = True
        else:
            FD_option = False
        image = request.form.get('image')
        image_address = volume + os.sep + image
        if image_address in image_list:
            image_exists = True
            if FD_option:
                image = face_detector_tool(image)
        else:
            image_exists = False
    return render_template("index.html", post_request=post_request, image_exists=image_exists,
image=image)
```

مطابق خط اول این کد هر زمانی که api /image_processing صدا زده شد این قسمت کد اجرا می شود.در واقع ما این فراخوانی را قبل تر در کد طرف client دیدیم.

بس زمانی که client درخواست اجرای این api را کرد این کد اجرا می شود.

در اینجا به وسیله مرورگر وقتی url : /image_processing را روی ip و port این داکر بخوانیم (get request)

چنین تصویری می آید:

وقتی نام عکس را نوشتیم و انجام شدن یا نشدن تشخیص چهره را مشخص کردیم و دکمه show را بزنیم چون نوع request اولیه get بود وارد if نمی شود و مستقیم خط آخر را اجرا می کند و آن یعنی دوباره خود تابع به خودش درخواست می دهد ولی این بار دخواست post است در نتیجه وارد if می شود :

در ابتدا چک می کند که آیا ما می خواهیم face detection هم انجام شود یا خیر و با توجه به آن متغیر FD_option را مقدار true یا false می دهد

حال بررسی می کند که آیا image با آن نام در لیست ساخته شده شامل تصاویر وجود دارد یا نه و با آن متغیر image_exists را true یا false می کند.

سپس اگر image_exist مقدار true داشت و FD_option مقدار true بود یعنی تشخیص چهره می خواهیم بس image که در واقع اسم عکس هست را به تابع face_detector_tool که در بالا توضیح دادیم فراخوانی می کند و خروجی که عکس دارای مستطیل دور صورت ها هست را می گیرد.

و در نهایت دوباره خود را call می کند و این روند ادامه می یابد هر دفعه متغیر ها به وسیله کاربر مقدار می گیرد و متغیر های ورودی کاربر در call بعدی وارد می شوند.

```
@app.route('/image_processing_api', methods=['POST'])
def rest_api_image_processing():
    data = json.loads(request.data)
    image = volume + os.sep + data['image']
    if image in image_list:
        response = serialize_image(image)
    else:
        response = 'Image Not Found'
    return response
```

مطابق خط اول این کد هر زمانی که api /image_processing_api صدا زده شد این قسمت کد اجرا می شود.در واقع ما این فراخوانی را قبل تر در کد طرف client دیدیم.

بس زمانی که client درخواست اجرای این api را کرد این کد اجرا می شود.

این فراخوانی در client زمانی بود که صرفا می خواستیم که بدانیم آیا عکس در سرور هست یا نه

در اینجا از request دریافت شده قسمت data را استخراج کرده و از آن اسم عکس را خارج می کند و سپس چک می کند که آیا آن تصویر در لیست تصاویر هست یا نه

اگر بود به عنوان باسخ مقدار سریال شده عکس را بر می گرداند و اگر نبود جمله Image Not Found را بر می گرداند.

```
@app.route('/face_detection_api', methods=['POST'])
def rest_api_face_detection():
    data = json.loads(request.data)
    response = {}
    image = volume + os.sep + data['image']
    if image in image_list:
        response['message'] = 'Image Found'
        response['face detection'] = face_detection(image)
    else:
        response['message'] = 'Image Not Found'
        response['face detection'] = None
    return json.dumps(response)
```

مطابق خط اول این کد هر زمانی که api /face_detection_api صدا زده شد این قسمت کد اجرا می شود.در واقع ما این فراخوانی را قبل تر در کد طرف client دیدیم.

بس زمانی که client درخواست اجرای این api را کرد این کد اجرا می شود.

این کد زمانی اجرا می شود که client بخواهد هم بودن عکس در فایل ها چک شود و هم وجود صورت در عکس

مانند قسمت قبل از data اسم عکس را استخراج می کند و دنبال آن اسم در لیست اسامی می گردد.

اگر بود بیام Image Found را می فرستد و هم چنین تابع face_detection را فراخوانی می کند که خروجی آن دیکشنری صورت های در عکس هست و آن را به client می دهد.

و اگر عکس وجود نداشت بیام Image Not Found را به client ارسال می کند.

```
app.run(port=server_port, host=server_ip)
```

این آخر هم نشان دهنده ران شدن این کد روی server_port و server_ip است.

اجرا کردن کد روی Docker:

مطالب بالا توضیحات و کامنت گذاری خط به خط کدها و نحوه عملکرد آن است.

حال تنها کار باقی مانده اجرا کردن این کد روی داکر است.

در مورد ساخت و build کردن داکر توضیحاتی نیست که نیاز باشد گزارش کنیم ولی مرحله به مرحله بالا آورن داکر و اجرای کد روی آن را فیلم گرفتیم و با صدا توضیح داده ایم و نام آن ها PART1 و PART2 و docker است که ضمیمه گزارش شده است.

در زیر به طور خیلی خلاصه آن را بیان کرده ایم ولی توضیحات دقیق تر و کامل تر را در فیلم ها داده ایم

ابتدا نوشتن Dockerfile برای بیلد کردن dockerimage سبس نوشتن فایل .yml برای بیلد کردن ساده تر با پلاگین docker compose

بعد از آن نوشتن فایل requirements.txt برای دیپندنسی های پایتون

در نهایت اجرا کردن دستور docker-compose up *.yml که این دستور خود داکر را اتوماتیک build می کند.

اگر بعد از اول می خواهید بیلد کنید اخر این دستور –build می گزاریم

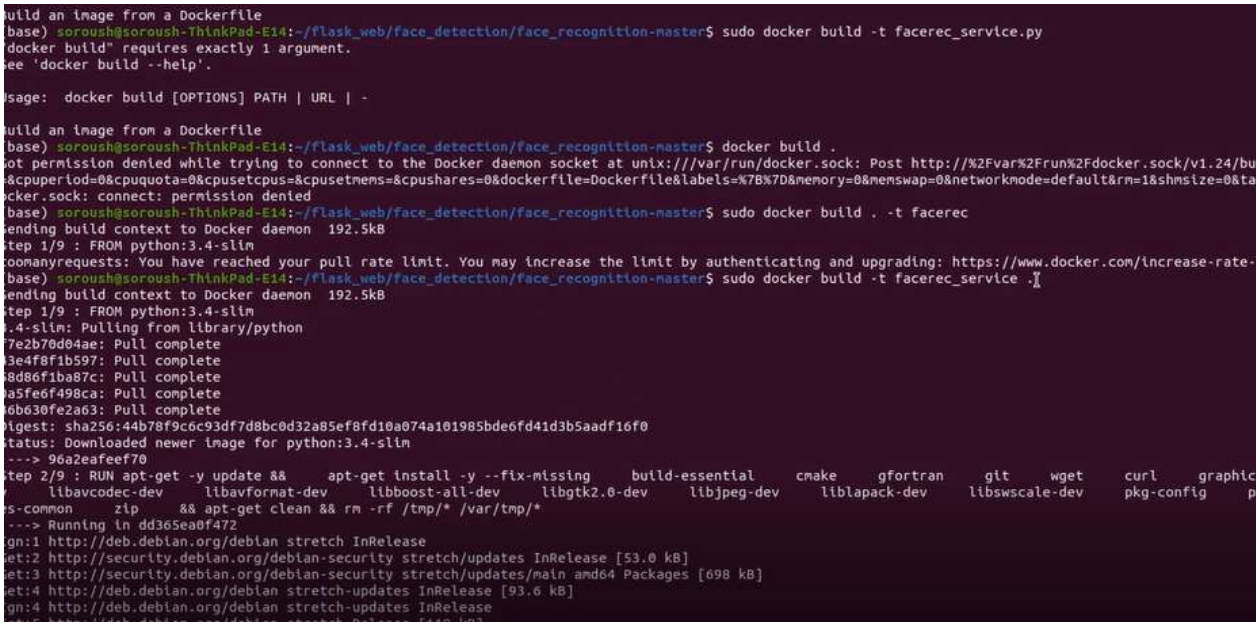
برای ارتباط دو تا داکر با هم باید توجه کنیم که هنگام ساخت آن ها هر دو را روی bridge اتصال دهیم در این صورت صرفا دانستن ip داکر دیگر برای برقراری ارتباط کافی است.

توضیحات تکمیلی در ویدیو docker.mkv ضمیمه شده است

که طبق این دستور پیدا می شود

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' container_name_or_id
```

در زیر قسمتی از عکس کد زده شده برای build کردن را مشاهده می کنید.



در آن دو فایل ابتدا داکر را بالا آورده و کد را روی آن قرار دادیم و image ساختیم و سپس به ازای ورودی های مختلف ((وجود عکس – نبود عکس _ تشخیص تصویر)) آن ها را در ویدیو ها تست کرده ایم.

در زیر تصاویری از خروجی محیط داکر به ازای ورودی های مختلف آورده شده است.

تصویر زیر محیط شماتیک html طراحی شده برای کاربر می باشد که به کمک تیک تایین می کند که آیا face detection می خواهد یا خیر و در زیر آن هم جا برای دادن اسم ورودی عکس است

خروجی هم بعد از زدن دکمه show در زیر آن مانند همین عکس نوشته می شود.

شکل زیر برای کار با terminal توسط کاربر است

```
(base) soroush@soroush-ThinkPad-E14:~/flask_web$ python client.py --face_detection --image pic12.jpeg
Number of face detected is 4
(base) soroush@soroush-ThinkPad-E14:~/flask_web$ python client.py --face_detection --image pic13.jpeg
Image not found in the server.
(base) soroush@soroush-ThinkPad-E14:~/flask_web$
```

که مثلا دو مثال با انجام face detection آورده ایم.

که در عکس اول که عکس زیر است 4 تصویر تشخیص داد

و در عکس دوم چون pic13 نداریم خروجی Image Not Found in the server را داده است.

مورد امتیازی دیگری که علاوه بر تشخیص وجود عکس در تصویر انجام دادیم کشیدن حاشیه برای صورت ها است

یعنی اگر در عکس صورت بود یک عکس دیگر که همان عکس اصلی با حاشیه مستطیل روی صورت ها هست هم اضافه می کند.

که مثلا این مدل opencv دو تا از عکس ها را تشخیص داده است. البته با افزایش سایز پنجره ممکن است بتواند هر 4 تا را تشخیص دهد ولی ما در اینجا با پارامتر های default مدل آن را اجرا کرده ایم

برای انجام دادن face detection یک داکر دیگر موازی داکری که کد خودمان در آن اجرا می شود بالا آوردیم و با اتصال bridge این دو داکر را به هم وصل کردیم توضیحات کامل تر آن داکری که بالا آوردیم را در قسمت توضیح کد face detection داده ایم

```
def face_detection(image):
    try:
        face_dict = os.popen(f'curl -X POST -F "file=@{image}" http://localhost:8080/').read()
    except:
        face_dict = "Error In Face Detection"
    return face_dict
```