פרויקט בחישוב מקבילי ומבוזר

נושא הפרויקט:

סיווג אובייקטים מהיר

שם הסטודנטית:

רננה קייקוב

ת"ז:

212378939

הפרויקט בוצע בהנחיית:

גב' תמר כהן

תוכן עניינים

הרקע לפרויקט

הפרויקט הוא בתחום הבינה המלאכותית machine learning, בנושא סיווג תמונה – classification.

בפרויקט זה השתמשתי בשלושה מודלים מאומנים החוקרים תמונה,

שלושת המודלים עוסקים בעיקר בזיהוי רחפנים.

המודלים הם:

- מודל המבוסס רשת vgg16 המאומן לסיווג ל- 5 מחלקות:
  Airplane, bird, drone, helicopter, other.
- מודל המבוסס רשת vgg16 המאומן לסיווג 2 מחלקות:
  Yes_drone, no_drone.
- מודל המבוסס רשת resnet50 המאומן לסיווג 5 מחלקות:

  Airplane, bird, drone, helicopter, other.

הבעיה שאיתה התמודדתי הייתה הזמן הרב שבו הפריים נחקר ע"י כל מודל ומודל והחזרת התשובה.

ע"כ רציתי לצמצם את זמן הניתוח ובעצם לפתור את הבעיה ע"י מקביליות, שלושת המודלים יחקרו את הפריים במקביל בשלוש מעבדים ובכך יקצרו את זמן הניתוח.

פעולה זו משמעותית וחשובה כיוון שבעיות הסיווג מתבצעות בעיקר בזמן אמת, במיוחד בנושא שאני לקחתי- זיהוי רחפן. כלומר המערכת הכללית עובדת כך שהיא מקבלת סרטון ממצלמה וחותכת לפריימים בכל רגע , בכל סרטון מתבצע זיהוי אובייקטים והשלב הבא הוא סיווג האובייקטים ע"י מודלי סיווג, ובשלב זה רציתי להתערב ולשלב את הפרויקט הנוכחי. האובייקט מסווג ע"י כמה מודלים כדי להגדיל את סיכויי הצלחת הזיהוי הנכון, ע"פ תשובתם של כמה מודלים מקבלים ניתוח נכון יותר. המערכת הכללית מטרתה להחזיר תשובה ולהתריע בזמן אמת, והיא מקבלת בכל שניה לפחות פריים אחד לניתוח, כך שנוצר מצב בעייתי שהוחזרה תשובה זמן רב מדי לאחר הצילום האמיתי של הפריים. לפיכך פעולה זו כה משמעותית.

תהליך המחקר

כדי לבצע את הפרויקט חקרתי כיצד ניתן לחסוך בזמן הסיווג, וכן חקרתי על הספריות שבעזרתן ניתן לבצע זאת, למדתי קצת על מקביליות, כיצד להפעיל tread , מהו מנעול, ועד.

מטרות

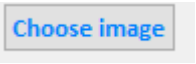
חיסכון בזמן תהליך הסיווג ע"י כמה מודלים.

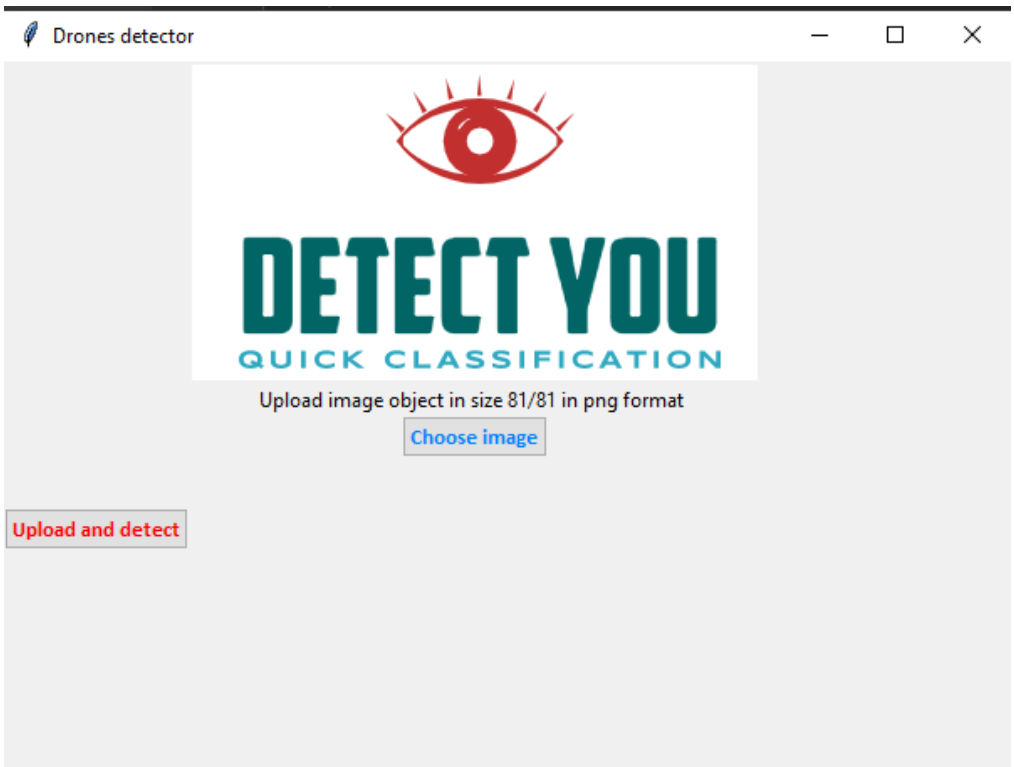
לימוד בכתיבת אלגוריתם ממוקבל.

לימוד בכתיבת אלגוריתם מבוזר תקשורת שרת- לקוח.

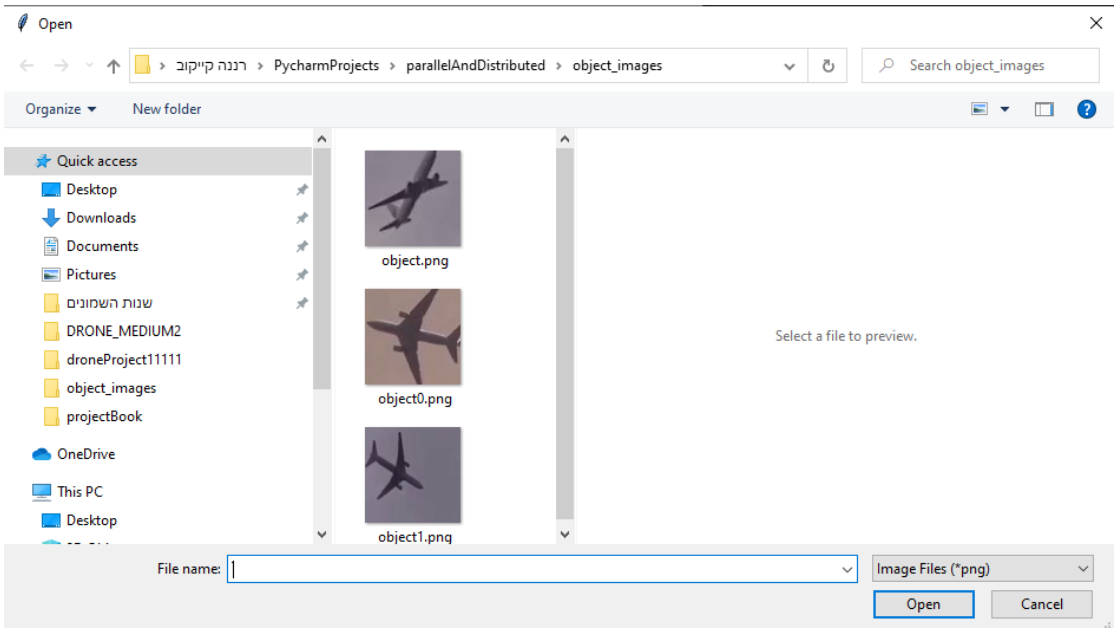
מסכים

מסך ראשי: המשתמש בוחר תמונת אובייקט להעלאה ע"י לחיצה על הכפתור "Choose image" :



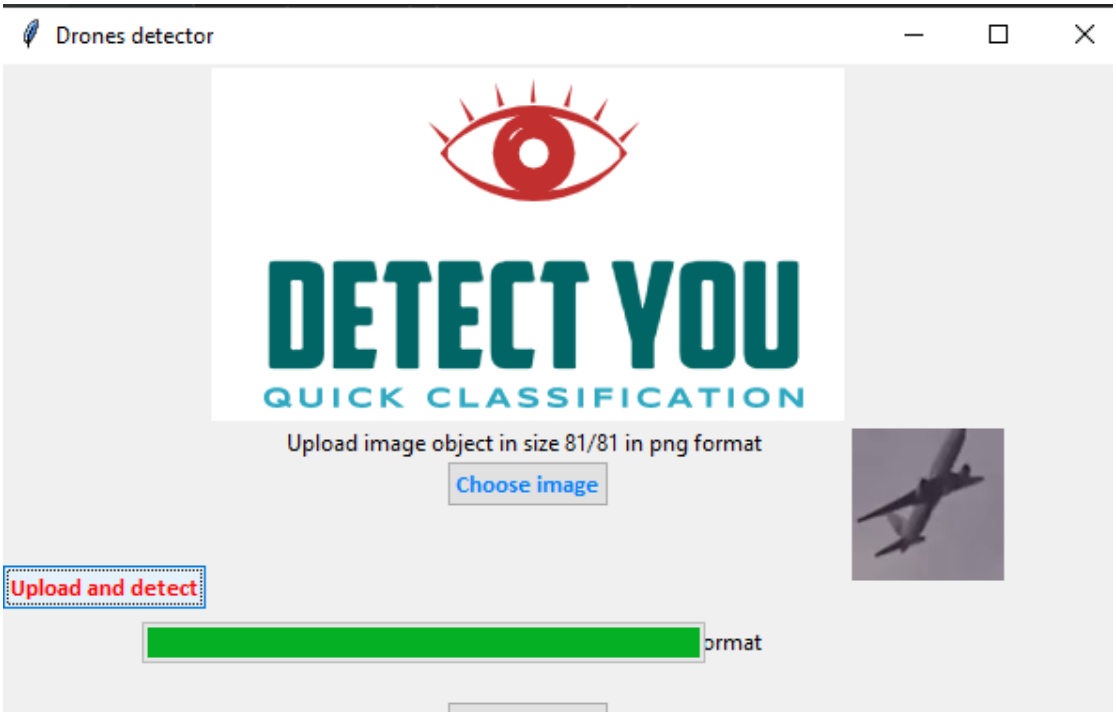


לאחר הלחיצה נפתח למשתמש חלונית עם אפשרות לבחירת קובץ:

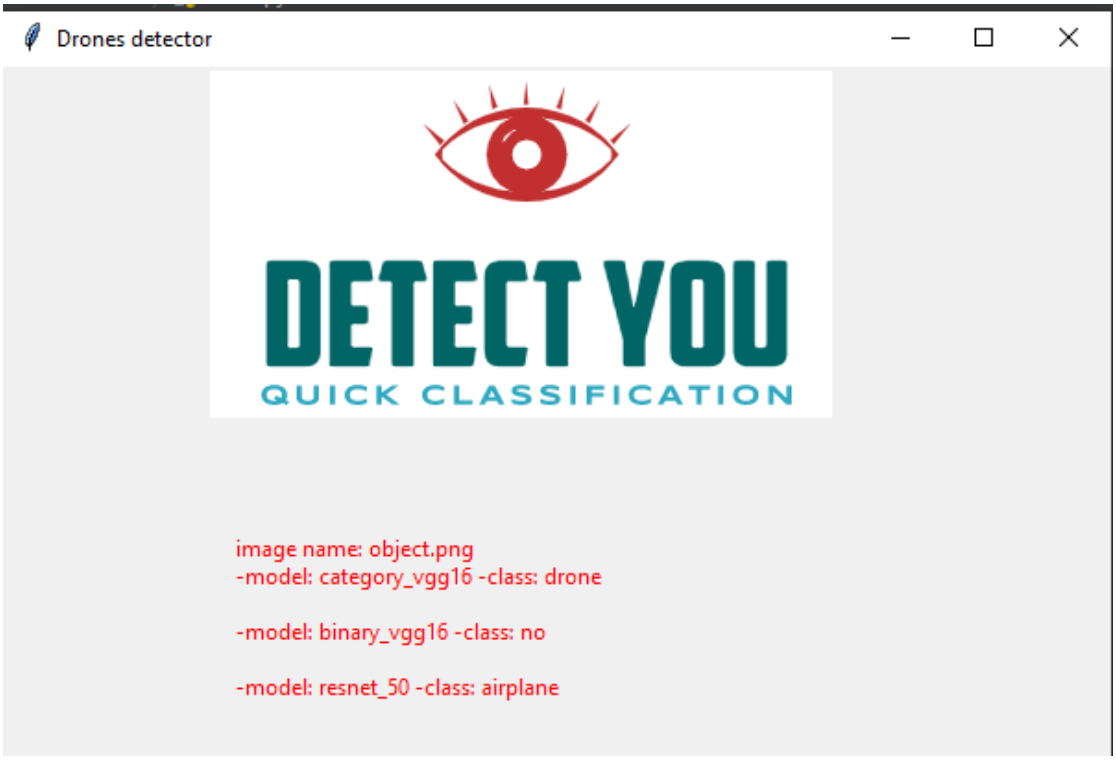


המשתמש בוחר את תמונת האובייקט הרצויה ולוחץ על הכפתור "Upload and detect":

לאחר ההעלאה תמונת האובייקט מופיעה על המסך וכן ציר התקדמות:



לבסוף מוצגות תוצאות הניתוח של שלושת המודלים על האובייקט כך:



תיאור האלגוריתם הראשי

האלגוריתם המקבילי מתחלק לשלושה חלקים:

- Threads: בכל tread תופעל פונקציית הסיווג classification() וכן ישלח לכל thread פרמטר שהוא שם המודל הרצוי להפעלת הסיווג, הפונקציה תסווג לכל tread ע"י מודל אחר.

- Classification: הסיווג עצמו, מתבצע ע"י ספריות של machine learning.

- Results: התוצאות נכתבות לתוך קובץ שלבסוף נקרא ומוחזר לצד ה- client.

כך במקום שהסיווג יתבצע אחד אחרי השני ויתבזבז זמן יקר וקריטי, מבצעים מקביליות כך שכל tread מבצע במקביל סיווג אחר.

קוד האלגוריתם

מקבילי

ראשית יצרתי פונקציית סיווג המקבלת אובייקט וסוג מודל לסיווג, מסווגת, וכותבת לקובץ את התשובה, והשליחה לפונקציה מתבצעת שלוש פעמים פעם לכל מודל וזאת ללא מקביליות, כך:

יצרתי מילון גלובלי שבו שיכנתי לכל סוג מודל את קובץ המודל שלו:

```
models={'category_vgg16':'model_vgg_categorical_s81.h5',
        'binary_vgg16':'model_vgg_s81.h5',
        'resnet_50':'pred_drone_5_classes_restnet_50_3.h5'}
```

הפונקציה הראשית:

```
def classification(path_img,size,model_type):
    classes = ['airplane', 'bird', 'drone', 'helicopter', 'other']
    if model_type == 'binary_vgg16':
        classes = ['yes', 'no']
    model=models[model_type]
    saved_model = load_model(model)
    img = image.load_img(path_img, target_size=(size, size))
    img = np.asarray(img)
    img = np.expand_dims(img, axis=0)
    output = saved_model.predict(img)
    i = np.argmax(output)
    txt="-model: "+model_type+" -class: "+classes[i]+"\n"
    write_to_file(txt)
```

פירוט:

יצירת רשימת מחלקות בהתאם למחלקות המודל, הצבה המשתנה model את קובץ המודל המתאים בהתאם לסוג המודל שהתקבל, טעינת המודל וטעינת התמונה ע"י ספרית keras (ספריה נוחה לשימוש במודלי בינה מלאכותית) והכנת התמונה למודל ע"י ספרית numpy.

קבלת תוצאת הסיווג, וכתיבת התוצאה לתוך קובץ.

הרצה:

```
# run classification
path_img='images/object.png'
size=81
classification(path_img,size,'category_vgg16')
classification(path_img,size,'binary_vgg16')
classification(path_img,size,'resnet_50')
```

```
Runtime in the program is:3.843581438064575
```

לאחר מכן החלטתי להפעיל את התהליך במקביליות, ע"כ שינתי מעט את הפונקציה כך:

הוספת פונקציית treads שיוצרת 3 treads ומפעילה ע"י כל אחד את פונקציית classification():

```
def detect_by_threads(file_path):
    t = []
    args = ['resnet_50', 'binary_vgg16', 'category_vgg16']
    for i in range(3):
        t.append(Thread(target=classification, args=(file_path, 81, args[i])))
        t[i].start()
    for i in range(3):
        t[i].join()
```

הפונקציה מקבלת את נתיב האובייקט לסיווג, היא יוצרת מערך ל- treads וכן מערך ארגומנטים לשליחה לפונקציה שבו מכניסים את שמות שלושת המודלים. הפונקציה עוברת בלולאה בגודל 3, ובכל פעם יוצרת tread חדש, ומכניסה אותו למערך, ומיד מפעילה אותו (t[i].start()), ולאחר מכן עוברת שוב בלולאה בגודל 3 ועושה פעולת join לכל tread, פעולה זו גורמת שתתבצע המתנה לכל ה- treads שהופעלו עד שיסתיימו.

כמו כן, כיוון שמתבצעת כתיבה לקובץ, יש צורך במנעול, וזאת כיוון שהקובץ הוא אובייקט משותף לכל ה- threads, וכשיש אובייקט משותף שלא קוראים ממנו אלא כותבים אליו נוצרת בעיה, וזאת כיוון שיכולות לצאת תוצאות שגויות כיוון שכל ה- threads כותבים במקביל ויכול להיות ש- thread אחד יתחיל לכתוב ובאמצע ימשיך לכתוב באותו מקום ה- tread השני, ע"כ יש צורך ביצירת מנעול שישמור על הקטע הקריטי (הכתיבה לקובץ) , כך שכל פעם ש thread ירצה לכתוב לקובץ הוא יצטרך להמתין עד שה thread האחר יסיים וישחרר את המנעול.

ביצוע:

הספרייה הנדרשת:

```
from threading import Lock
```

יצירת המנעול:

```
mutex=Lock()
```

ושינוי הפונקציה העיקרית כך:

```python
def classification(path_img,size,model_type):
    classes = ['airplane', 'bird', 'drone', 'helicopter', 'other']
    if model_type == 'binary_vgg16':
        classes = ['yes', 'no']
    model=models[model_type]
    saved_model = load_model(model)
    img = image.load_img(path_img, target_size=(size, size))
    img = np.asarray(img)
    img = np.expand_dims(img, axis=0)
    output = saved_model.predict(img)
    i = np.argmax(output)
    # mutex for write in file:
    txt="-model: "+model_type+" -class: "+classes[i]+"\n"
    global mutex
    mutex.acquire()
    write_to_file(txt)
    mutex.release()
```

הפיכת המנעול לגלובלי, כדי שכל ה threads יכירו אותו, נעילה לפני הקטע הקריטי (write_to_file()), ושחרור לאחריו.

כך מתבצעת הכתיבה לקובץ ללא כל דריסה.

תוצאות:

```
Runtime in the program is:3.410719871520996
```

מבוזר

קשר בין שרת ללקוח- התבצע ע"י sockets:

הקשר בפרויקט שלי בין השרת ללקוח מתבצע בחלק העברת התמונה לסרבר ע"י המרה לביטים,

ובחזרה לקלינט החזרת התוצאות.

Client

השתמשתי במסך ה- console של Python, ע"י ספריית tkinter:

יבוא הספריות הנדרשות:

```
import socket
from tkinter import *
from tkinter.ttk import *
from tkinter.filedialog import askopenfile
import os
```

אופן הפעולה:

פונקציה היוצרת את המסך לקלינט:

```
def showScreen():
    # shoe the logo
    img = PhotoImage(file='logo.png')
    mylabel = Label(
        ws,
        image=img
    )
    mylabel.grid(row=0, column=1)

    # the label that say to the client what to do
    chooseImgLbl = Label(
        ws,
        text='Upload image object in size 81/81 in png format '
    )
    chooseImgLbl.grid(row=4, column=1, padx=10)

    # botton 1 "choose image" that starts the function open_file()
    chooseImgBtn = Button(
        ws,
        text='Choose image',
        style='W.TButton',
        command=lambda: open_file()
    )
    chooseImgBtn.grid(row=5, column=1)

    # botton 2 "Upload and detect" that starts the function upload_files()
    upldBtn = Button(
        ws,
        text='Upload and detect',
        style='TButton',
        command=upload_files
    )
    upldBtn.grid(row=6, columnspan=1, pady=30)
```

הפונקציה open_file():

הפונקציה מופעלת ע"י הלחצן הראשון.

```
def open_file():
    file_path = askopenfile(mode='r', filetypes=[('Image Files', '*png')])
    if file_path is not None:
        global img_path
        img_path = file_path.name
        global filesize
        filesize = os.path.getsize(img_path)
        pass
```

הפונקציה פותחת קובץ png ושומרת את נתיב הקובץ וגודל הקובץ במשתנים גלובליים.

הפונקציה upload_file():

הפונקציה מופעלת ע"י הלחצן השני.

זוהי הפונקציה המרכזית בצד ה-client הפותחת סוקט לתקשור עם צד הסרבר ומעבירה לו דרכו את התמונה:

לפני כן:

הצבת כתובת ה-ip של הסרבר כדי שנדע עם מי לתקשר, וכן בחירת -port ערוץ שבו הם יתקשרו (חלק זה נכתב גם בצד הלקוח וגם בצד השרת כמובן):

```
HOST = '192.168.8.999'  # The server's hostname or IP address
PORT = 65432            # The port used by the server
```

כמו כן הגדרה (רק בצד הלקוח) חוצץ וכן גודל buffer שדרכו תעבור התמונה לסרבר:

```
SEPARATOR = "<SEPARATOR>"
BUFFER_SIZE = 10000 # send 10000 bytes each time step
```

הפונקציה:

```python
def upload_files():
    # show the image
    img = PhotoImage(file=img_path)
    mylabel = Label(
        ws,
        image=img
    )
    mylabel.grid(row=3, column=3)

    # progress
    pb1 = Progressbar(
        ws,
        orient=HORIZONTAL,
        length=300,
        mode='determinate'
    )
    pb1.grid(row=7, columnspan=1, pady=20)


    filename = img_path
    filesize = os.path.getsize(filename)

    # create the client socket
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    s.connect((HOST, PORT))

    s.sendall(f"{filename}{SEPARATOR}{filesize}".encode())

    with open(filename, "rb") as f:
        bytes_read = f.read(BUFFER_SIZE)
        s.sendall(bytes_read)
```

```python
    # start the progress pb1
    for i in range(9):
        ws.update_idletasks()
        pb1['value'] += 20
    pb1.destroy()

    data = s.recv(BUFFER_SIZE)
    Label(ws, text=data, foreground='red').grid(row=4, columnspan=3, pady=10)

    s.close()
```

```
filename = img_path
filesize = os.path.getsize(filename)
```

פירוט:

יצירת socket והתחברות לשרת דרך הכתובת והפורט:

```
# create the client socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((HOST, PORT))
```

שליחת נתונים לצד הסרבר: שליחת שם הקובץ וגודלו, זאת ע"י קידוד ה- string:

```
s.sendall(f"{filename}{SEPARATOR}{filesize}".encode())
```

שליחת קובץ התמונה עצמו ע"י פתיחת הקובץ בצורה בינארית ושליחת הקובץ בביטים:

```
with open(filename, "rb") as f:
    bytes_read = f.read(BUFFER_SIZE)
    s.sendall(bytes_read)
```

קבלת תשובה מהסרבר:

```
data = s.recv(BUFFER_SIZE)
```

הצגה במסך וסגירת ה- socket:

```
Label(ws, text=data, foreground='red').grid(row=4, columnspan=3, pady=10)

s.close()
```

server

```python
if __name__ == '__main__':
    while True:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.bind((HOST, PORT))
            s.listen()
            client_socket, address = s.accept()
            with client_socket:
                received = client_socket.recv(BUFFER_SIZE).decode()
                filename, filesize = received.split(SEPARATOR)
                filename = os.path.basename(filename)
                filesize = int(filesize)

                with open(filename, "wb") as f:
                    bytes_read = client_socket.recv(BUFFER_SIZE)
                    f.write(bytes_read)
                txt="\n"+"\n"+"image name: "+filename
                write_to_file(txt)
                detect_by_threads(filename)

                res = get_data()
                client_socket.sendall(res.encode())
                client_socket.close()
                s.close()
```

פירוט:

יצירת server socket, ה- socket מאזין לקלינט, וקבלת ה client socket ע"י הפקודה accept.

קבלת הנתונים שנשלחו מצד הקלינט- שם קובץ וגודל, והמרה חוזרת מהקידוד,

פתיחת קובץ לכתיבה באופן בינארי, קבלת נתוני הקובץ מצד הקלינט וכתיבתו מחדש בצד הסרבר,

כתיבת שם התמונה בקובץ התוצאה,

הפעלת הפונקציה המרכזית- זיהוי במקביליות- detect_by_threads(),

טעינת התוצאה ע"י הפונקציה get_data() הטוענת את הנתונים שבקובץ התוצאה, ושליחת התוצאה בצורה מקודדת לצד הקלינט.

סגירת socket הקלינט ו- socket הסרבר.

מבני נתונים בהם משתמשים בפרויקט

- מילון- לשימוש לאחסון נתיבי קבצי המודל כך ששמות המודלים משמשים כאינדקסים.
- רשימה- רשימה של threads.

מסקנות

ישנו הבדל קטן אומנם בין ההרצות, וזאת כייוון שהיעילות המקביליות עולה יותר ככל שיש יותר תהליכונים וניצול, אך למרות זאת עדיין המקבול עוזר כיוון שאומנם לכל אובייקט ישנו הבדל קטן אך בשימוש לסרטון לדוג ההבדל יהיה יותר מוחשי.