UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică si Calculatoare

Catedra de Calculatoare

Reordonarea obiectelor bazată pe o imagine de referință. Comunicarea Bluetooth dintre o aplicație Android cu placa de dezvoltare Raspberry Pi 4

Student : Podaru Bogdan-Mihai

Grupa : 30238

Profesor indrumător : Lișman Dragoș-Florin

Data : 03-01-2020

Cuprins

1. Rezumat

Studiile arată că oamenii inteleg si lucrează mai usor cu obiectele din jurul lor daca acestea prezinta o anumite ordine , spre exemplu : obiectele aşezate in anumite pozitii tind sa fie distinse mai usor, obiectele rotite si privite din mai multe unghiuri ofera mai multe detalii, un ansamblu sau un sistem in care participă ca si componente mai multe obiecte nu ar putea exista daca intre aceste componente nu ar exista o anumite ordine. In aceasta lucrare s-a propus o soluție pentru ordonarea unor obiecte. Obiectivele principale ale proiectului sunt stabilirea relatiei de ordine intre obiecte, bazată pe o imagine de referință, precum si comunicarea de date cu ajutorul modulului bluetooth integrat al Raspberry Pi 4 si o aplicatie Android dezvoltata in Kotlin. Pentru detecția obiectelor s-a folosit Deep Learning cu limbaj țintă Python iar pentru detectarea de similaritati intre poze s-a folosit tehnica de Template Matching. Rezultatele experimentale au aratat că proiectul funcționeaza doar in conditii de iluminare bune cu o rata de peste 90 %, iar viteza de execuție depinde de hardware-ul folosit (folosind Raspberry Pi 4 media timpului de procesare este de 40 secunde). Aplicația prezinta multe dezvoltari posibile ulterioare in funcție de domeniu.

2. Introducere

Proiectul s-a dezvoltat in jurul domeniilor procesarii de imagini si al inteligenței artificiale , doua dintre cele mai utilizate si populare domenii ale computer science din ziua de azi.

Deep learning este un subset al inteligenței artificiale care foloseste algoritmi pentru invațare autonoma, algoritmi care poti fi imbunatățiți de hardware-ul gazda in funcție de informatia din mediu. O ramura a acestui domeniu este ANN (Artificial neural network) care este inspirata din retelele neuronale biologice din care este constituit creierul uman. O mare problemă al acestui domeniu il constituie comportamentul retelelor neuronale la aparitia de date noi. Printre aplicatii ale deep learning enumera: asistenți la parcare, controlul traficului, recunoașterea de fețe in aeroporturi.

Procesarea de imagini se refera la aplicarea diferitelor tipuri de filtre asupra imaginilor in scopul extragerii de informații sau facilitării extragerii de informații in functțe de aplicatie. Printre metode ale procesarii de imagini enumerăm : clasificarea de imagini, feature extraction, recunoaștere de forme , pattern matching, augmentare de imagini cu aplicatii in medicina , securitate sau alte domenii de importanță mare. In continuare prin **scena** se va face referire la un grup de obiecte la diferite locații, prin **imagine referință** se va face referire la imaginea in care obiectele sunt in pozițiile initiale, poziții care stabilesc ordinea inițiala a obiectelor, prin **imagine test** se va face referire la imaginea care se va compara cu cea de referința pentru a stabili verdictul.

2.1. Obiective

Problema abordată in acest proiect este urmatoarea : determinarea unei masuri prin care se poate stabili daca anumite obiecte si-au modificat poziția inițiala relativa la celelate obiecte, mai precis daca obiectele si-au pastrat ordinea unele față de celelalte pe o direcție de vizualizare , spre exemplu : in scena se afla un pet de apa si o carte situată in dreapta petului de apa, problema aici stabilește dacă petul a rams in stânga cărții sau si-a schimbat poziția si este acum in dreapta acesteia). Obiectivele principale care reies din aceata problemă este stabilirea unei conexiuni, respectiv a unei transmisii intre o aplicatie smartphone si o placa de dezvoltare, determinarea unor algoritmi de procesare imagini si

detectţe de obiecte, precum si determinarea unei modalitati de ordonare a obiectelor detectate , rezultate care vor fi afisate intr-o formă intuitiva.

Per ansamblu se fotografiaza o scena cu o aplicaţie Android de doua ori (in prima poza : scena este in ordine, in a doua poza cateva obiecte sunt repozitionate). Aceste poze sunt transmise catre o placa de dezvoltare Raspberry Pi 4 prin modulul de Bluetooth integrat al placii, după care sunt procesate (se utilizeaza Deep Learning pentru detecţia de obiecte din scenă şi Template Matching pentru determinarea similaritatilor dintre poze) si se stabileste daca obiectele din a doua poza, numită şi poză de test şi-au păstrat sau nu poziţia relativă la celelate obiecte , regula stabilită din prima poză, numită si poză de referinţă.

2.2. Descrierea soluţiei si avantajele ei

Cum aplicaţiile Android sunt la indemana multor persoane si activitaţile de zi cu zi primesc suport din partea smartphone-ului , iar o placa de dezvoltare Raspberry Pi 4 este mai permisiva ca pretţ decat un sistem mai complex de calcul precum un laptop , soluţia propusă este una ieftina si la indemana multor oameni. Majoritatea telefoanelor conţin modulul de bluetooth integrat precum şi unul de camera.

In ceea ce urmează capitolele completeaza raportul prin detalii legate de noţiunile teoretice (capitolul 3 prezinta metode, termeni tehnici, scheme bloc) care vor servi drept bază pentru implementarea propriu zisa descria in capitolul 4 unde sunt prezentati algoritmii de procesare de imagini folositi, implementarea retelei neuronale pentru detectarea obiectelor, structurile de date folsite, limbajele tina utilizate, mediile de dezvoltare in care s-au construit, hardware-ul utilizat precum si conexiunile software intre modulele software create. Totodata in capitolul 4 se dezolva strategia de proiectare a solutie bazata pe elementele amintite mai sus. In capitolul 5 s-au descris rezultatele experimentale pe , conditiile de lucru si de functionare, s-au realizat statistici, s-au adus explicatii suplimentare referitoare la limitarile si disfunctionalitatile proiectului. In final in capitolul 6 au fost trasate concluziile intregului proces de creare al proiectului.

3. Fundamentare teoretica

3.1. Privire de ansamblu

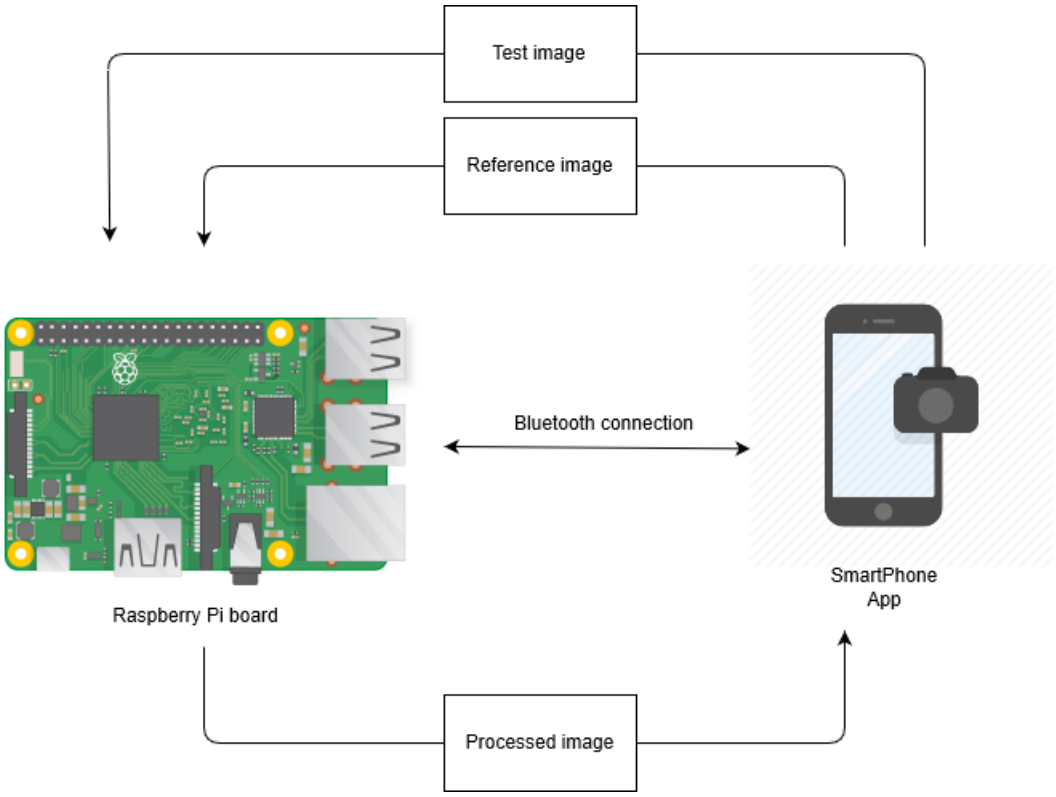


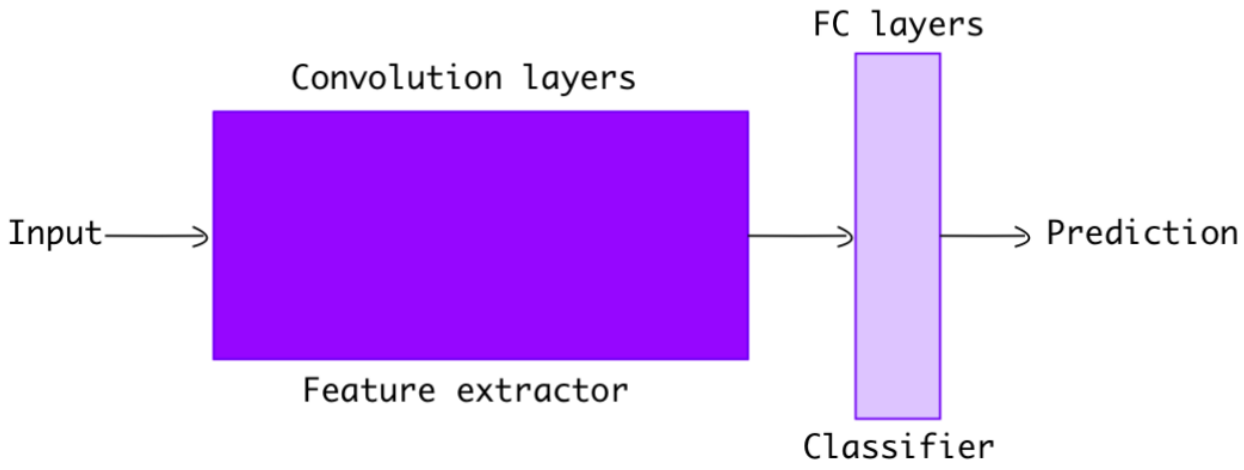
Figura 1 ilustreaza schema bloc a sistemului

Figura 1. prezinta schema de principiu a proiectului. Dupa cum se poate observa exista 2 parti : partea de procesare pe raspberry pi si partea de aplicatie android. Cele 2 parti comunica prin intermediul unor socket-uri deoarece raspberry foloseste ca limbaj nativ Python pe cand aplicatie a fost scrisa in Kotlin.

Figura sugereaza un proces in care aplicatia trimite 2 imagini (una de test si una de referinta) si primeste o alta drept rezultat. Aceasta conexiune se realizeaza prin Bluetooth folosind protocolul RFCOMM. Rezultatul s-a obtinut prin rularea unui program de detectie de imagini si de template matching. Tot acest proces este descris in subsectiunile care urmeaza :

3.2. Modele in Keras

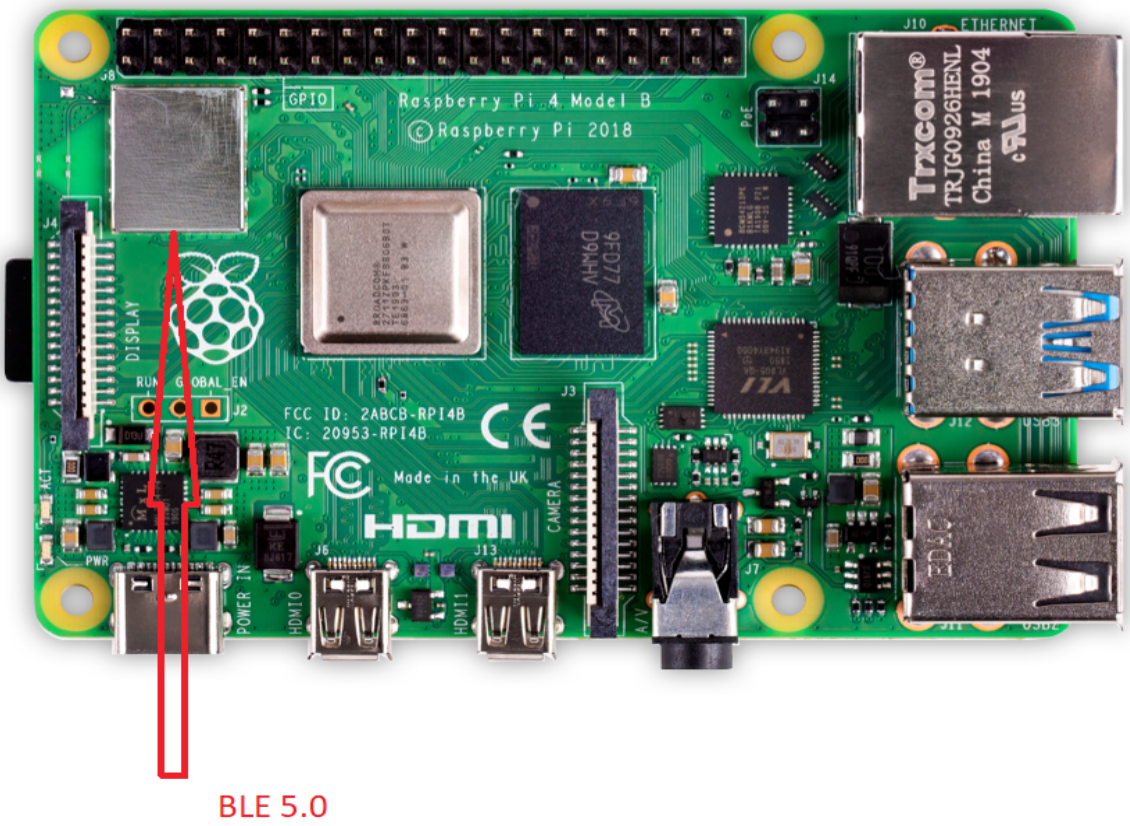
Tensorflow este o librarie folosita pentru machine learning, iar Keras este tot o librarie care ruleaza peste Tensorflow pentru ai face interfata mai simpla, astfel utilizatorii se vor putea concentra mai mult pe proiectare. Keras contine 10 modele preantrenate pe datele de la ImageNet pentru clasificare. ImageNet e o colectie de imagini care contine 1000 de categorii de imagini. Pentru o privire simplificata se considera urmatoarea schema :



Layerele de convolutie au rolul de a extrage atribute din imaginea de intrare. Cele FC(fully connected) au rolul de clasifica imaginea intr-o anumita categorie. Layerele de convolutie cauta anumite atribute in datele de intrare, daca se regasesc acestea vor produce o activare mare. Convolutia in termenii procesarii de imagini se refera la extragerea unei matrici de dimensiune K x K centrata intr-o locatie (x,y) spre exemplu un pixel si inmultirea acesteia cu un layer kernel. Elementele matricei rezultat sunt adunate element cu element .

3.3. Bluetooth sockets

Placa contine un modul bleutooth integrat de tip Low Energy (LE) . Comunicarea cu aplicatia android se realizeaza prin acest modul prin intermediul a doua socket-uri : unul rulat pe raspberry pi care asteapta conexiuni, iar altul rulat pe aplicatia smartphone care incearca sa se conecteze. Astfel are loc schimbul de date intre cele 2 parti amintite mai sus.



Aplicatiile Client-Server sunt construite pe baza socket-urilor, reprezentand un punct mare de interes. In cazul acestui proiect Raspberry Pi 4 va lua rolul de Server iar aplicatia Android de Client. Protocolul RFCOMM este un protocol de transfer pentru transmiterea seriala de date. Profilul serial Bluetooth se bazeaza pe acest protocol. Aceste este similar cu TCP in cazul aplicatiilor Client-Server

over Internet. Pozele vor fi trimise astfel in mod serial, dupa o transformare a acestora in siruri de octeti proveniti din reprezentarile lor ca bitmap.

3.4. Template matching

Aceasta este o tehnica a procesarii de imagini care functioneaza pe urmatorul principiu :

- bucati dintr-o imagine de referinta sunt extrase (numite si templates) si salvate
- se ia un template si se parcurge cu el de sus in jos si de la stanga la dreapata o imagine test, comparand la fiecare pas zona suprapusa
- procesul continua pana la prima detectie sau pana la finalul imaginii de test
- rezultatul acestui proces il reprezinta gasirea locatiei zonei suprapuse

In cadrul proiectului s-a utilizat OpenCV Template Matching, care implementeaza mai multe metode de template matching dintre care s-a ales folosirea TM_CCORR_NORMED.

Pentru suprapunere este nevoie de o masura a similaritatii. Pentru aceasta se poate utiliza metoda Normalized Cross-Corelation **[23]** pentru ca ofera rezultate invariante la lumina si ale caror valori sunt in intervalul [-1,1] .

4. Proiectare si implementare

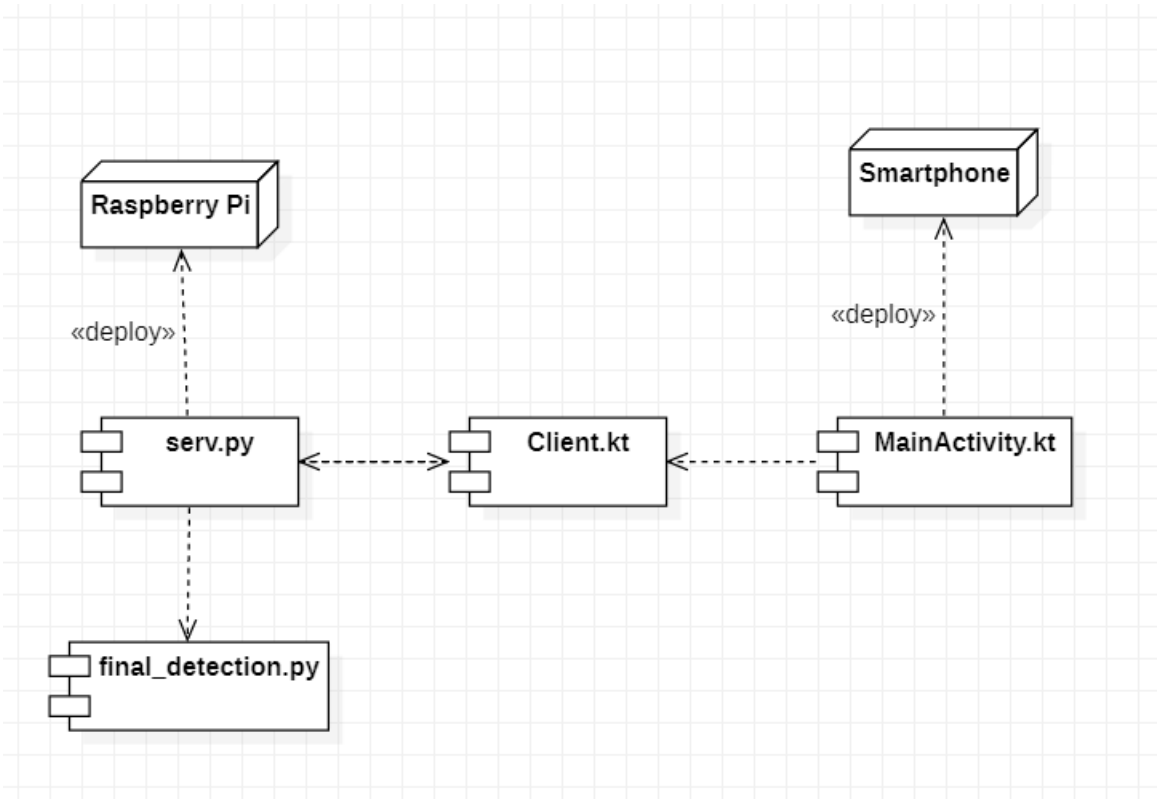
4.1. Proiectare

Facand din nou referire la Figura 1, ma voi raporta de data aceasta la modularizarea si structura interna a fiecarei parti (partea de aplicatie Android si partea de procesare pe server Raspberry).

a) Partea de aplicatie : Intrucat aplicatia contine si interfata grafica am separat logica de transmise de date si de receptie de aceasta interfata astfel au rezultat doua module software : MainActivity.kt si Client.kt . Transmisia si receptia de date are loc asincron din momentul inceperii conexiunii intre socket-uri , astfel pentru sincronizarea comunicarii dintre cele 2 parti s-a creat un nou fir de executie reprezentat de Client care asteapta primirea de informatii pentru afisare si trimiterea pozelor .

b) Partea de server : Raspberry Pi joaca rolul unui server care pe de o parte primeste informatii (pozele serializate) : acesta asteapta conexiuni de la Clienti si primirea a 2 poze (imaginea referinta si imaginea test) pe care le salveaza intr-un folder numit PI_INCOMING de unde ulterior le va prelua pentru procesare, iar pe de alta parte proceseaza informatiile primite. Pentru primirea de poze serializate este nevoie de un script python care sa deschida un socket si sa astepte conexiuni. Totodata acest script trebuie sa aiba functionalitatea de a scrie 2 fisiere imagine din datele primite prin bluetooth. Acest script va fi mentionat ca serv.py. Partea de procesare necesita separarea de partea de transmisie deoarece abia dupa ce au fost primite datele poate incepe prelucrarea acestora astfel e nevoie de un nou script python , denumit in proiect final_detection.py . Acest script trebuie sa fie capabil sa foloseasca o retea neuronala preantreanata de detectie a obiectelor si totodata sa folosesca un algoritm de template matching.

Diagrama de deployment corespunzatoare structurii de mai sus este :



4.2. Implementare

Codul sursa se poate gasi la finalul documentatiei in Anexa. Mai sus au fost proiectate modulele necesare functionarii intregului sistem. In acest capitol se vor prezenta detalii legat de implementarea acestor module organizate pe module.

I. Partea de aplicatie Android

a) MainActivity.kt

Acest modul contine definitii de ascultatori de butoane care stabilesc ce trebuie sa se intample la apasarea acestor butoane .Acesti ascultatori vor apela metodele openBluetooth(), connectToRaspberryPi(),sendImage(imageView) unde imageView stabileste care din imagini sa fie trimisa (cea de referinta sau cea de test). In solutia abordata ambele imagini sunt preluate si

sunt trimise simultan. Astfel metoda este apelata la momente foarte apropiate de timp. Metoda dispatchTakePictureIntent creeaza un Intent care acceseaza resursele de STORAGE si CAMERA **[4]** ale telefonului si lanseaza in executie camera pentru preluarea unei imagini.

b) Client.kt

Client va reprezenta celelat capat al conexiunii si dupa cum s-a descris mai sus acesta trebuie sa functioneze ca un Thread. Astfel metoda cea mai utila a acestui modul este metoda run care va avea acces la obiectele de timp InputStream si OutputStream ale socket-ului bluetooth imediat deschis. Aici se asteapta primirea unor siruri de octeti pentru a fi trimisi (imaginile serializate , acele imageView) pentru a fi transmise mai departe catre server.

II. Partea de procesare Raspberry

a) serv.py

Acest modul realizeaza o operatie blocanta si anume asteapta o conexiune din partea unui Client la un socket de bluetooth deschis pe partea de server. Odata ce un client se conecteaza firul executiei contina si ajunge in punctul in care asteapta primirea de date (primirea de poze serializate). Aici un detaliu de implementarea este faptul ca sirul de date vine in urmatoarea forma: A B SIR_DE_OCTETI_IMAGINE, unde A reprezinta numarul de cifre ale numarului ce reprezinta dimensiunea (B) nr-ului de octeti din SIR_DE_OCTETI_IMAGINE. Acest lucru face posibila transmiterea primei imagini de referinta si posibilitatea distinctiei momentului in care s-a terminat transmisia acestei imagini si astfel se poate continua cu transmisia imaginii de test pe acleasi principiu, altfel singura solutie ar fi fost inchiderea si deschiderea socket-ului din nou , solutie nedorita.

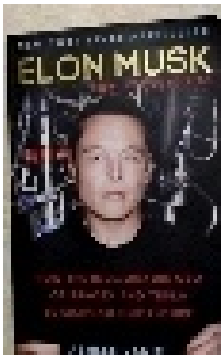
b) final_detection.py

Cel mai important modul al proiectului este final_detection.py. Aici se regasesc 2 sectiuni de cod esentiale si distincte : sectiunnea de detectie obiecte si sectiunea de template matching. Sectiunea de detectie de obiecte preia modelul resnet50 **[12]** antrenat pe baza de date de la ImageNet unde se afla 1000 de categorii de imagini, si il incarca in aplicatie. Din folderul PI_INCOMING preia imaginile de referinta si de test denumite dupa un format (imgr este image reference, iar imgt este image test). Acestea se vor folosi pentru preziceri , adica cu ajutorul metodei predict_on_batch() se vor gasi bounding-boxurile ale obiectelor detectate si se vor salva.

A doua sectiune va parcurge aceste bounding-boxuri **[11]** si cu ajutorul metodei matchTemplate() **[15]** folosing algoritmul TM_CCORR_NORM din libraria cv2 se realizeaza template matching-ul.

Exemplu de decupare dupa template :

Template-urile extrase vor fi :



Acest matching salveaza coordonatele colturilor stanga jos al bounding-boxurilor pentru obiectele in care template matching-ul a avut success. Se realizeaza diferenta intre coordonatele vecine si se salveaza semnele diferentelor. Se realizeaza acesti pasi atata pentru imaginea test cat si pentru cea referinta(cea referinta stabileste template-urile pentru matching pe a doua), si la final se compara semnele coordonatelor bounding-boxurilor pentru cele 2 imagini. Daca se gasesc cel putin 2 semne

diferite inseamna ca cel putin un obiect a fost miscat de pe axa orizonta, iar pozitia sa relativa la toate obiectele s-a schimbat.

5. Rezultate experimentale

Proiectul a fost rulat in conditiile in care sistemul de operare folosit pentru testarea in prima faza a retelei neuronale (inainte de deploy pe placa de dezvoltare Raspberry Pi 4) a fost Windows 10. Pe raspberry sistemul de operare a fost Raspbian Buster Desktop Version 4.19. Mediile de dezvoltare au fost Thonny Python Editor si PyCharm pentru realizarea socket-ului de server, a script-ului de detectie obiecte iar pentru realizarea codului client (pentru aplicatia Android) s-a utilizat Android Studio cu limbaj gazda Kotlin. Alimentarea placutei s-a facut printr-un port USB 3.1 cu voltaj de 5V in conditiile in care placa accepta intre 4.0V si 5.5V. Pentru detectia de obiecte s-a folosit TensorFlow v.2 si Keras 2.5.2.

Testarea s-a facut in urmatorul mod:

5.1. Exemplu de obiecte neordonate

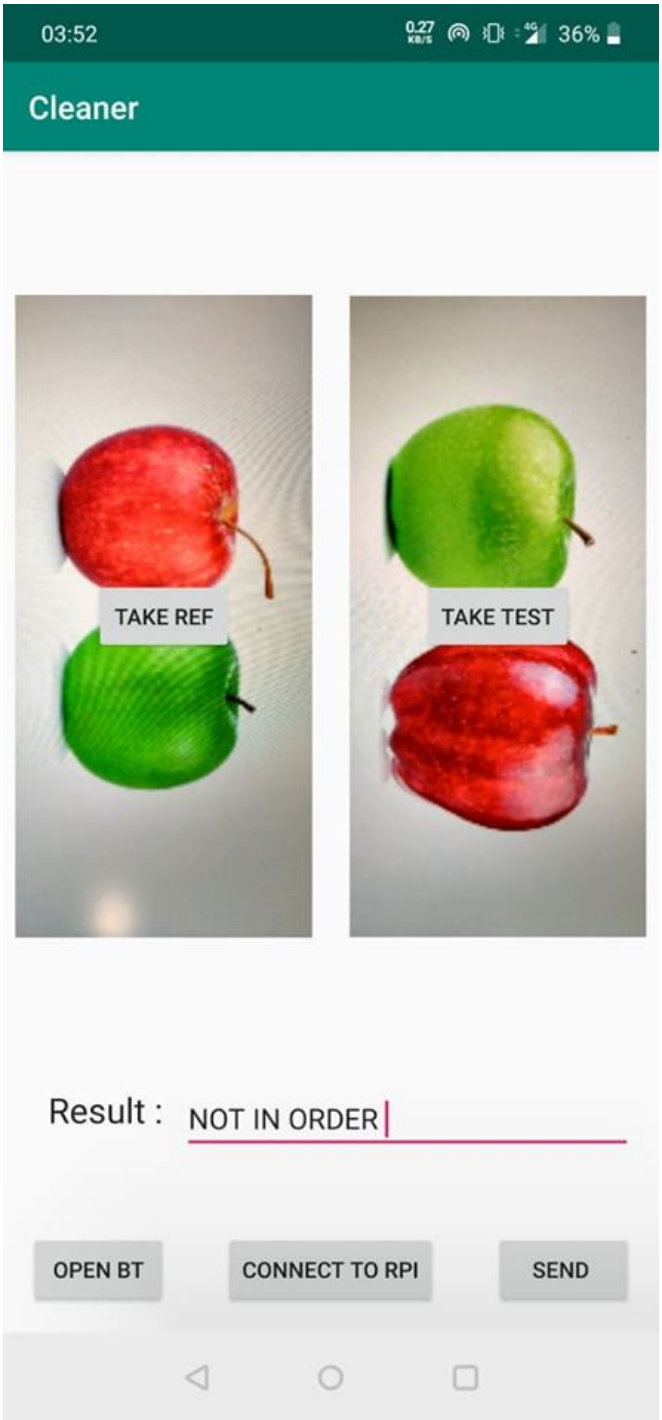
Imaginea de referinta prezinta pozitia marului rosu ca fiind in stanga pozitiei marului verde , iar imaginea de test prezinta pozitiile lor inversate dupa cum urmeaza:

Imagine referinta

Imagine test

Rezultatul in urma procesarii sugereaza faptul ca obiectele sunt in ordine. Se poate observa cu atentie ca merele sunt de diferite forme si de nuante . Acesta este rezultatul metode CV_CCORR_NORMED, utilizata pentru template matching.

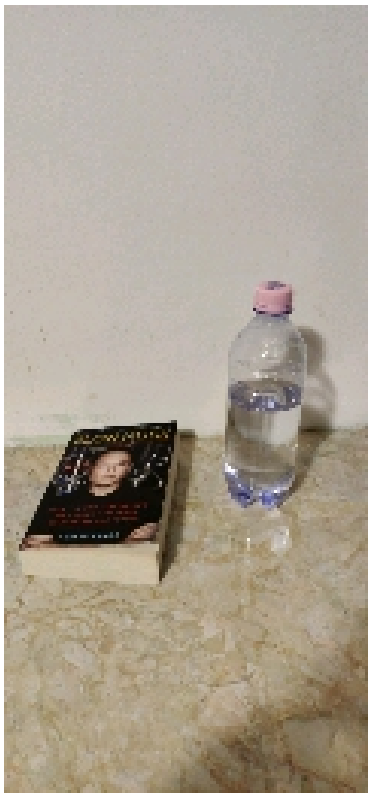
Rezultatul primului exemplu :



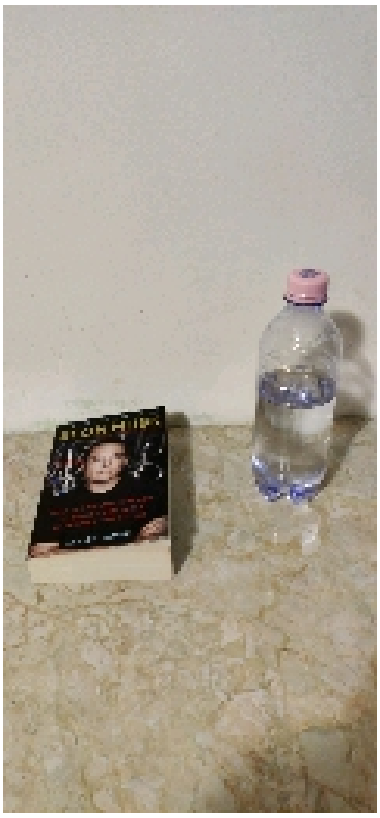
Urmatorul exemplu descrie toti pasii prin care trec cele 2 imagini :

5.2. Exemplu de obiecte ordonate

Pentru urmatorul test s-au folosit drept obiecte o carte si un pet. Imaginea de referinta prezinta pozitia cartii ca fiind in stanga pozitie petului , iar imaginea de test prezinta aceleasi pozitii relative dupa cum urmeaza. Imaginile au fost preluate din folderul de receptie al Raspberry Pi 4, unde pentru a ajunge cu o viteza ridicata au fost comprimate, si astfel au pierdut din calitate.

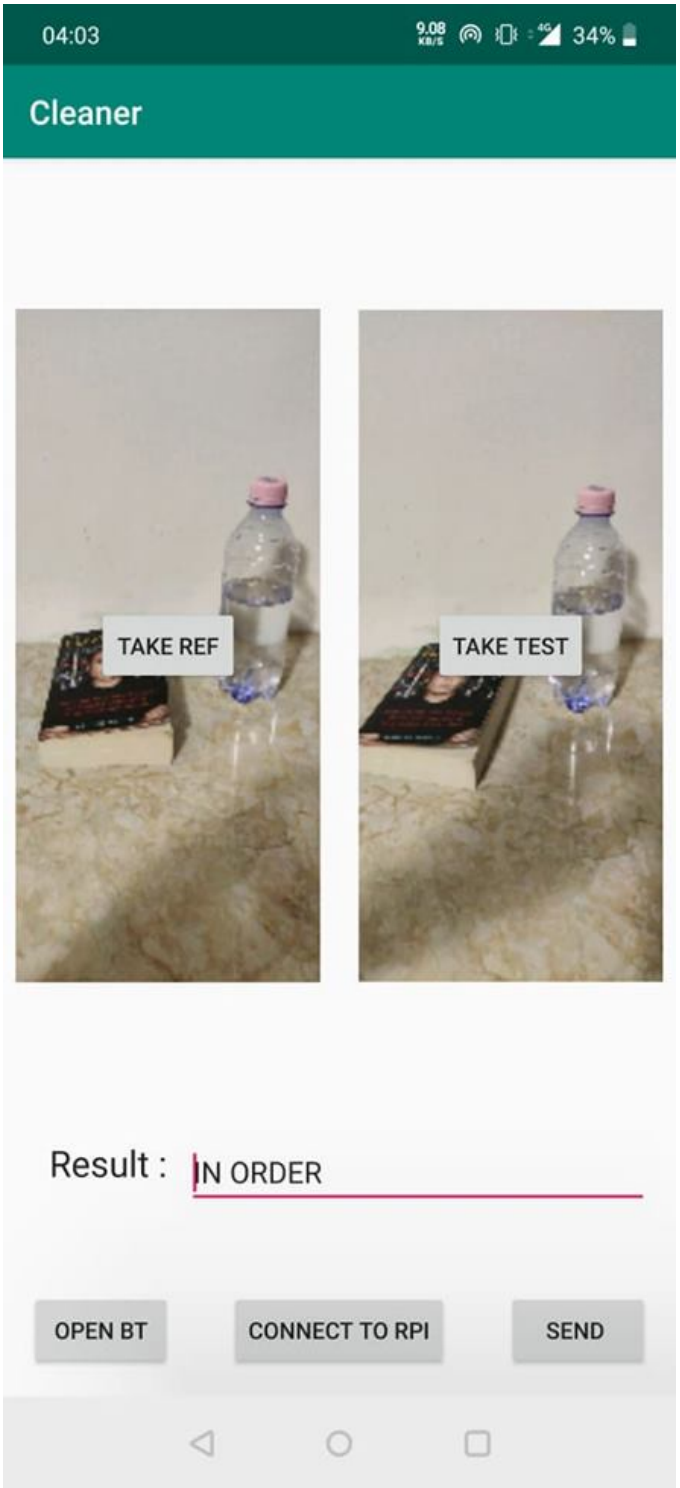


Imagine referinta



Imagine test

Rezultatul obtinut :



6. Concluzie

Obiective proiectului au fost :

1. conexiunea si transmisia intre raspberry pi si aplicatia android
2. detectia de obiecte
3. stabilirea pozitiei obiectelor in scena raportat la o imagine de referinta

In concluzie toate obiectivele au fost realizate cu urmatoarele precizari : detectia de obiecte functineaza doar in conditii de iluminare buna si cu o rata de peste 90% , pozitia obiectelor in scena este comparata doar dupa axa orizontala , astfel solutia propusa nu va distinge obiectele ordonate diferit pe o alta axa.
Contributiile originale au constat in aplicatia android si compararea pozitiilor obiectelor, precum si a transferului serial al imaginilor.

Aplicatia poate fi folosita in multe domenii spre exemplu : organizarea documentelor dintr-un dosar fizic, aranjarea locuintei dupa preferintele fiecaruia, asamblarea unui sistem dupa un template, dezvoltarea controlata plantelor, detectia de disfunctinalitati , etc.

Proiectul poate fi dezvoltat in multe directii dar se propun urmatoarele : ordonarea obiectelor si dupa celelate 2 axe (verticala si de adancime) , antrenarea retelei cu un numar suplimentar de date pentru o acuratete mai buna , realizarea unui kit robotic pentru o reordonare fizica a obiectelor.

Bibliografie

[1] Conexiunea cu Raspberry Pi petru interfata grafica :

https://www.raspberrypi.org/documentation/remote-access/ssh/

[2] Notiuni referitoare la Artificial Neural Networks (ANN) :

https://www.learnopencv.com/neural-networks-a-30000-feet-view-for-beginners/

[3] Concepte de baza referitoare la procesarea de imagini :

https://www.geeksforgeeks.org/digital-image-processing-basics/

[4] Sectiune cod lansare camera in aplicatie :

https://developer.android.com/training/camera/photobasics#kotlin

[5] Conexiunea folosing bluetooth socket (folosing reflection -> liniile 258,259) :

https://github.com/johnhowe/BlueTerm/blob/master/src/es/pymasde/blueterm/BluetoothSerialService.java

[6] Crearea de bluetooth RFCOMM socket (versiunea Python) :

http://pages.iu.edu/~rwisman/c490/html/pythonandbluetooth.htm

https://people.csail.mit.edu/albert/bluez-intro/x232.html#rfcomm-server.py

[7] Punct de pornire :

http://blog.davidvassallo.me/2014/05/11/android-linux-raspberry-pi-bluetooth-communication/

[8] Tutorial tensorflow :

https://www.youtube.com/watch?v=6g4O5UOH304

[9] Ideea de ordonare dupa bounding box :

https://stackoverflow.com/questions/34495847/detect-locations-of-objects-in-an-image]

[10] Detectie de obiecte pe datasetul COCO folosind Keras :

https://medium.com/object-detection-using-tensorflow-and-coco-pre/object-detection-using-tensorflow-and-coco-pre-trained-models-5d8386019a8

[11] Creare de bounding box in tensorflow :

https://stackoverflow.com/questions/48915003/get-the-bounding-box-coordinates-in-the-tensorflow-object-detection-api-tutorial

[12] O parte din codul de detectie de obiecte folosind reteaua ResNet50 :

https://github.com/fizyr/keras-retinanet/blob/master/examples/ResNet50RetinaNet.ipynb

[13] Similaritati intre imagini folosind homografie

https://pysource.com/2018/07/20/find-similarities-between-two-images-with-opencv-and-python/

[14] Tensorflow object detection explicatii suplimentare :

https://www.freecodecamp.org/news/how-to-deploy-an-object-detection-model-with-tensorflow-serving-d6436e65d1d9/

[15] OpenCV Feature Matching :

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[16] Alternativa testare fara aplicatia din proiect :

https://scribles.net/setting-up-bluetooth-serial-port-profile-on-raspberry-pi/

[17] Probleme bluetooth raspberry pi :

https://raspberrypi.stackexchange.com/questions/51548/raspberry-pi-3-bluetooth-pairing-issue-with-tablet

[18] Bluetooth setup pe raspberry pi :

https://learn.adafruit.com/install-bluez-on-the-raspberry-pi/installation

[19] Ghid pentru utilizare modele pre-antrenate folosind Keras :

https://towardsdatascience.com/step-by-step-guide-to-using-pretrained-models-in-keras-c9097b647b29

[20] Keras ResNet50 documentatie :

https://keras.rstudio.com/reference/application_resnet50.html

[21] OpenCV Template Matching

https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html

[22] Kotlin tutorial :

https://youtu.be/F9UC9DY-vIU

[23] Template matching teorie :

https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html

Anexe

Anexa A

Codul sursa complet

final_detection.py (fisierul de detectie si template matching)

```
import keras
from keras_retinanet.utils.image import read_image_bgr, preprocess_image, resize_image
from keras_retinanet.utils.visualization import draw_box, draw_caption
from keras_retinanet.utils.colors import label_color
from keras_retinanet.utils.gpu import setup_gpu
from keras_retinanet.models import load_model
# import miscellaneous modules
import matplotlib.pyplot as plt
import cv2
import os
import numpy as np
import time
gpu = 0
setup_gpu(gpu)
model_path = "/home/pi/py_projects/obj_det/keras-retinanet/snapshots/resnet50_coco_best_v2.1.0.h5"
model = load_model(model_path, backbone_name='resnet50')
labels_to_names = {0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus', 6: 'train', 7:
'truck', 8: 'boat', 9: 'traffic light', 10: 'fire hydrant', 11: 'stop sign', 12: 'parking meter', 13: 'bench', 14:
'bird', 15: 'cat', 16: 'dog', 17: 'horse', 18: 'sheep', 19: 'cow', 20: 'elephant', 21: 'bear', 22: 'zebra', 23:
'giraffe', 24: 'backpack', 25: 'umbrella', 26: 'handbag', 27: 'tie', 28: 'suitcase', 29: 'frisbee', 30: 'skis', 31:
'snowboard', 32: 'sports ball', 33: 'kite', 34: 'baseball bat', 35: 'baseball glove', 36: 'skateboard', 37:
'surfboard', 38: 'tennis racket', 39: 'bottle', 40: 'wine glass', 41: 'cup', 42: 'fork', 43: 'knife', 44: 'spoon',
45: 'bowl', 46: 'banana', 47: 'apple', 48: 'sandwich', 49: 'orange', 50: 'broccoli', 51: 'carrot', 52: 'hot dog',
```

```
53: 'pizza', 54: 'donut', 55: 'cake', 56: 'chair', 57: 'couch', 58: 'potted plant', 59: 'bed', 60: 'dining table',
61: 'toilet', 62: 'tv', 63: 'laptop', 64: 'mouse', 65: 'remote', 66: 'keyboard', 67: 'cell phone', 68:
'microwave', 69: 'oven', 70: 'toaster', 71: 'sink', 72: 'refrigerator', 73: 'book', 74: 'clock', 75: 'vase', 76:
'scissors', 77: 'teddy bear', 78: 'hair drier', 79: 'toothbrush'}
img1 = read_image_bgr("/home/pi/py_projects/PI_Incoming/imgr.jpg")
img2 = read_image_bgr("/home/pi/py_projects/PI_Incoming/imgt.jpg")
draw = img1.copy()
draw2 = img2.copy()
draw = cv2.cvtColor(draw, cv2.COLOR_BGR2RGB)
draw2 = cv2.cvtColor(draw2, cv2.COLOR_BGR2RGB)

# preprocess image for network
img1 = preprocess_image(img1)
img1, scale1 = resize_image(img1)

img2 = preprocess_image(img2)
img2, scale2 = resize_image(img2)

# process image
start = time.time()
boxes1, scores1, labels1 = model.predict_on_batch(np.expand_dims(img1, axis=0))
print("processing time: ", time.time() - start)

# correct for image scale
boxes1 /= scale1

img1t = cv2.imread("/home/pi/py_projects/PI_Incoming/imgr.jpg")
img2t = cv2.imread("/home/pi/py_projects/PI_Incoming/imgt.jpg")
# visualize detections
count = 0
```

```
templates = []
found1 = []
for box, score, label in zip(boxes1[0], scores1[0], labels1[0]):
    count = count + 1
    if score < 0.7:
        break
    color = label_color(label)
    b = box.astype(int)
    x1 = b[0]
    y1 = b[1]
    x2 = b[2]
    y2 = b[3]
    crop_img=img1t[y1:y2,x1:x2]
    cv2.imwrite("cropped_"+labels_to_names[label]+str(count)+".jpg",crop_img)
    gray_img = cv2.cvtColor(img1t, cv2.COLOR_BGR2GRAY)
    template = cv2.imread("cropped_"+labels_to_names[label]+str(count)+".jpg",
cv2.IMREAD_GRAYSCALE)
    templates.append(template)
    w, h = template.shape[::-1]
    result = cv2.matchTemplate(gray_img, template, cv2.TM_CCORR_NORMED  )
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
    top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    img_copy = img1t.copy()
    cv2.rectangle(img_copy,top_left,bottom_right, (0, 255, 0), 3)
    found1.append((top_left[0], top_left[1]))

found2 = []
for template in templates:
    gray_img = cv2.cvtColor(img2t, cv2.COLOR_BGR2GRAY)
```

```
    result = cv2.matchTemplate(gray_img, template, cv2.TM_CCORR_NORMED  )
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
    top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    img_copy = img2t.copy()
    cv2.rectangle(img_copy, top_left, bottom_right, (0, 255, 0), 3)
    found2.append((top_left[0], top_left[1]))



def relative_order(list):
    l = []
    for i in range(0,len(list)-1):
        l.append(np.sign(list[i]-list[i+1]))
    return l



def check_signs(signs1, signs2):
    for i in range(0, len(signs1)):
        if signs1[i] != signs2[i]:
            return 0
    return 1

xcoords1 = [x for (x, y) in found1]
xcoords2 = [x for (x, y) in found2]
signs1 = relative_order(xcoords1)
signs2 = relative_order(xcoords2)
result = check_signs(signs1, signs2)
with open("/home/pi/py_projects/result.txt","w+") as file:
    file.write("IN ORDER" if result==1 else "NOT IN ORDER")
print("IN ORDER" if result==1 else "NOT IN ORDER")
```

serv.py (script pentru transmisie si receptie de imagini)

```
import os
import sys
from bluetooth import *

print("Server running...")
print("Turning on Bluetooth")
hostMACAddress = 'DC:A6:32:54:8E:B7' # The MAC address of a Bluetooth adapter on the server.
The server might have multiple Bluetooth adapters.
backlog = 1
size = 1024
s = BluetoothSocket(RFCOMM)
s.bind((hostMACAddress, PORT_ANY))
s.listen(backlog)
try:
    print("Conecting socket client...")
    client, clientInfo = s.accept()
    print("Connected!")
    with open("/home/pi/py_projects/PI_Incoming/imgr.jpg","wb+") as file:
        imSizeNum = int.from_bytes(client.recv(1),"big")
        imSize = 0
        for i in range(imSizeNum):
            imSize = imSize* 10 + int.from_bytes(client.recv(1),"big")
        for i in range(imSize):
            data = client.recv(1)
            file.write(data)
    with open("/home/pi/py_projects/PI_Incoming/imgt.jpg","wb+") as file:
        imSizeNum = int.from_bytes(client.recv(1),"big")
```

```
        imSize = 0
        for i in range(imSizeNum):
            imSize = imSize* 10 + int.from_bytes(client.recv(1),"big")
        for i in range(imSize):
            data = client.recv(1)
            file.write(data)

    os.system("./main.sh")
    with open("/home/pi/py_projects/result.txt","r") as file:
        client.send(file.read())
except:
    print("Closing socket")
    client.close()
    s.close()
```

MainActivity.kt (Clientul din relatia Client-Server)

```
class MainActivity : AppCompatActivity() {

    var bluetoothAdapter :BluetoothAdapter? = null // local device Bluetooth adapter for exec tasks
    var clientSocket : BluetoothSocket? = null
    var raspberryDevice : BluetoothDevice? = null
    val REQUEST_ENABLE_BLUETOOTH = 1 //open BT request code
    var client : Client? = null
    var openBTFlag = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
//editText.setBackgroundColor(Color.RED)
//editText.setTextColor(Color.WHITE)

open.setOnClickListener {
    openBluetooth()
    Toast.makeText(this,"Bluetooth is open",Toast.LENGTH_LONG).show()

}
connect.setOnClickListener {
    connectToRaspberryPi()
    Toast.makeText(this,"Connected to raspberry pi",Toast.LENGTH_LONG).show()
}
send.setOnClickListener {
    try {
        //sendImage()
        client!!.message(sendImage(imageView),"reference")
        Toast.makeText(this,"Image has been sent",Toast.LENGTH_LONG).show()
        client!!.message(sendImage(imageView5),"test")
        Toast.makeText(this,"Image has been sent",Toast.LENGTH_LONG).show()
    }
    catch (e:Exception){
        //disconnect()
    }
    finally {
        //disconnect()
    }
}
val REQUEST_IMAGE_CAPTURE_REFERENCE = 2
val REQUEST_IMAGE_CAPTURE_TEST = 3
takePhoto.setOnClickListener {
```

```
            dispatchTakePictureIntent(REQUEST_IMAGE_CAPTURE_REFERENCE)
        }

        takePhoto2.setOnClickListener {
            dispatchTakePictureIntent(REQUEST_IMAGE_CAPTURE_TEST)
        }
    }
    private fun connectToRaspberryPi() {
        raspberryDevice = bluetoothAdapter!!.getRemoteDevice("DC:A6:32:54:8E:B7")
        System.out.println(raspberryDevice!!.name)
        if (raspberryDevice != null) {
            client = Client(raspberryDevice!!,editText)
            client!!.start()
        }else{
            println("No such device")
        }
    }
    // !! throw exception if null otherwise specifies not nullable
    private fun openBluetooth() {
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter() // get device's local bluetooth
        if(!bluetoothAdapter!!.isEnabled){
            val enableBluetoothIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE) //launch
new BT open request
            startActivityForResult(enableBluetoothIntent, REQUEST_ENABLE_BLUETOOTH)
        }
        openBTFlag = 1
    }
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if(requestCode == REQUEST_ENABLE_BLUETOOTH){
```

```
        if(resultCode == Activity.RESULT_OK){
            if(bluetoothAdapter!!.isEnabled){
                println("BT connected")
            }
        }
    }
    if (requestCode == 2 && resultCode == RESULT_OK) {
        val imageBitmap = data!!.extras!!.get("data") as Bitmap
        imageView.setImageBitmap(imageBitmap)
    }
    if (requestCode == 3 && resultCode == RESULT_OK) {
        val imageBitmap = data!!.extras!!.get("data") as Bitmap
        imageView5.setImageBitmap(imageBitmap)
    }
}
private fun disconnect() {
    clientSocket!!.inputStream.close()
    clientSocket!!.outputStream.close()
    clientSocket!!.close()
}
private fun dispatchTakePictureIntent(request_code : Int) {
    Intent(MediaStore.ACTION_IMAGE_CAPTURE).also { takePictureIntent ->
        takePictureIntent.resolveActivity(packageManager)?.also {
            startActivityForResult(takePictureIntent, request_code)
        }
    }
}
private fun getNum(num:Int) : Int{
    var tmp = num
    var nr = 0
```

```
        while( tmp != 0 ){
            nr++
            tmp= tmp / 10
        }
        return nr
    }
    private fun sendImage(imageView: ImageView) : ByteArray{
        var baos = ByteArrayOutputStream()
        var bitmap = imageView.drawable.toBitmap()
        bitmap.compress(Bitmap.CompressFormat.JPEG,100,baos)
        val size = baos.toByteArray().size
        println("Size of image :" + baos.toByteArray().size)
        var list = ArrayList<Byte>()
        list.add(getNum(size).toByte())
        var tmp = size
        var tmpList = ArrayList<Byte>()
        while(tmp!=0){
            tmpList.add((tmp%10).toByte())
            tmp/=10
        }
        list.addAll(tmpList.reversed())
        println(list)
        list.addAll(baos.toByteArray().toList())
        return list.toByteArray()
    }
    fun setText(text:ByteArray?) = editText.setText(text.toString(), TextView.BufferType.EDITABLE)
}
```

Client.kt (lucrul cu bluetooth socket-ul pe partea de client)

```
class Client(device: BluetoothDevice,editText: EditText): Thread() {

    var MY_UUID = device.uuids[0].uuid
    private var byteArray: ByteArray? = null
    private var byteArray2: ByteArray? = null
    val paramTypes = arrayOf<Class<*>>(Integer.TYPE)
    val m = device.javaClass.getMethod("createRfcommSocket", *paramTypes)
    private var socket = m.invoke(device, 1) as BluetoothSocket
    private var returnedValue : ByteArray? = null
    private val editText = editText

    override fun run() {

        Log.i("client", "Connecting")
        socket.connect()

        Log.i("client", "Sending")
        val outputStream = this.socket.outputStream
        val inputStream = this.socket.inputStream
        try {
            while(true){
                if(byteArray != null)
                    break
            }
            outputStream.write(byteArray)
            outputStream.flush()

            while(true){
                if(byteArray2 != null)
                    break
```

```
        }
        outputStream.write(byteArray2)
        outputStream.flush()
        while(!currentThread().isInterrupted()) {
            try {
                val bytesAvailable = inputStream.available()
                if (bytesAvailable > 0) {
                    val packetBytes = ByteArray(bytesAvailable)
                    inputStream.read(packetBytes)
                    editText.setText(String(packetBytes))
                }
            }catch (e:Exception){
            }
        }
        Log.i("client", "Sent")
    } catch(e: Exception) {
        Log.e("client", "Cannot send", e)
    } finally {
        outputStream.close()
        inputStream.close()
        socket.close()
    }
}
fun message(msg : ByteArray, which:String){
    if(which.equals("reference"))
        byteArray = msg
    else if(which.equals("test"))
        byteArray2 = msg
}
fun getReturnedValue(): ByteArray? {
```

```
        return returnedValue
    }
}
```