Programska aplikacija za detekciju kršenja mjera socijalnog distanciranja

Gubec, Sandra

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Sandra Gubec

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Sandra Gubec

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svom mentoru doc. dr. sc. Tomislavu Stipančiću na stručnom vođenju, pristupačnosti, povjerenju te pruženoj pomoći tijekom pisanja ovog rada.

Posebno se zahvaljujem svojoj obitelji na razumijevanju, strpljenju i neizmjernoj podršci.

Također zahvaljujem svojim prijateljima i kolegama koji su mi tijekom preddiplomskog dijela studija bili velika pomoć i podrška.

Sandra Gubec

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Sandra Gubec** JMBAG: **0035217506**

Naslov rada na hrvatskom jeziku: **Programska aplikacija za detekciju kršenja mjera socijalnog distanciranja**

Naslov rada na engleskom jeziku: **Software application for detection of violations of social distancing measures**

Opis zadatka:

Računalni modeli temeljeni na algoritmima umjetne inteligencije omogućavaju automatiziranje različitih procesa koji koriste podatke, uključujući strojno: prepoznavanje, klasifikaciju, predviđanje, učenje i sl. Metode prepoznavanja su primjenjive u različitim domenama ljudskog djelovanja uključujući analizu ljudskog djelovanja.
U radu je potrebno razviti računalni model neuronske mreže koji će biti korišten kao detektor kršenja mjera za socijalno distanciranje u doba COVID-19 virusne pandemije. Softversko rješenje je potrebno temeljiti na Python programskom jeziku te OpenCV i ImageAI bibliotekama za računalni vid. Dobiveno softversko rješenje je potrebno eksperimentalno evaluirati uključivši ljudske subjekte. U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
9. 5. 2022.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Računalni vid je područje umjetne inteligencije koje nastoji omogućiti računalima prepoznavanje i razlikovanje objekata svojstveno ljudskom vidu. Zahvaljujući brojnim istraživanjima, razvoju novih algoritama te velikoj količini vizualnih informacija od strane pametnih telefona, sigurnosnih kamera i drugih uređaja, računalni vid se uz pomoć dubokog učenja i konvolucijskih neuronskih mreža u današnjici sve više primjenjuje u zadacima koji zahtijevaju prepoznavanje. Iz prethodno navedenih razloga, računalni vid je uz strojno učenje i umjetne neuronske mreže temelj programske aplikacije razvijene u ovome radu. Programska aplikacija za detekciju kršenja mjera socijalnog distanciranja izrađena je u programskom jeziku Python uz odgovarajuće biblioteke i module. Dobiveno softversko rješenje eksperimentalno je evaluirano uključujući ljudske subjekte.

Ključne riječi: socijalno distanciranje, Python, umjetna inteligencija, neuronske mreže, konvolucijske neuronske mreže, strojno učenje, računalni vid

SUMMARY

Computer vision is a field od artificial intelligence that seeks to enable computers to recognize and distinguish objects characteristic of human vision. Thanks to numerous researches, the development of new algotithms and the large amount of visual information from smartphones, security cameras and other devices, computer vision with the help of deep learning and convolutional neural networks is nowadays increasingly applied in tasks that require recognition. For the aforementioned reasons, computer vision, along with machine learning and artificial neural networks, is the basis of the software application developed in this paper. The software application for detecting violations of social distancing measures was created in the Python programming language with appropriate libraries and modules. The obtained software solution was experimentally evaluated including human subjects.

Key words: social distancing, Python, artificial intelligence, neural networks, convolutional neural networks, machine learning, computer vision

1. UVOD

1.1. COVID-19

Krajem 2019. godine svijet je preplavila vijest o novom virusu s kojim se čovjek nikada do tada nije susreo. Riječ je o virusu SARS-CoV-2 koji uzrokuje sada već svima poznatu bolest zvanu COVID-19, odnosno korona virus. Budući da se dokazalo kako se virus brzo i lako širi među ljudima, iako otkriven u Kini, ubrzo je cijeli svijet bio obuhvaćen pandemijom korona virusa. Nakon proglašenja globalne pandemije od strane Svjetske zdravstvene organizacije, u svijetu je zavladala karantena koja je imala brojne negativne posljedice ne samo na ljude, već i na zdravstvo, ekonomiju i razvoj. Ograničeno kretanje, zabrana okupljanja, zatvaranje brojnih trgovina i poduzeća i socijalno distanciranje samo su neke od mjera donesenih sa svrhom sprječavanja širenja korona virusa.

COVID-19 prenosi se kapljičnim putem, uglavnom kašljanjem i kihanjem, no moguća je zaraza i indirektnim putem preko kontaminiranih ruku oboljele osobe. Stoga su uvedene brojne preventivne mjere poput redovitog pranja ruku, izbjegavanja kontakta s oboljelom osobom, nošenja maske u zatvorenim prostorima, izolacija te samoizolacija. Virus uzrokuje simptome slične gripi poput povišene tjelesne temperature, kašlja, bolova u mišićima, umora te otežanog disanja. Starije osobe i osobe s kroničnim bolestima sklonije su razvijanju teže kliničke slike koja često zahtijeva i bolničko liječenje [1].

1.1.1. Socijalno distanciranje

Jedna od ključnih mjera u sprječavanju širenja korona virusa jest socijalno distanciranje. Pojam socijalnog distanciranja podrazumijeva izbjegavanje bliskog kontakta od najmanje 2 metra u zatvorenom prostoru i 1,5 metar na otvorenom prostoru. Može se odnositi i na mjere većih razmjera poput zabrane održavanja javnih događanja i okupljanja u zatvorenim prostorima, ograničenja broja ljudi, prelazak na *online* nastavu, mogućnost fleksibilnog radnog vremena ili rad od kuće [2]. Neki od konkretnih dokaza uspješnosti mjere socijalnog distanciranja vidljivi su u povijesti tijekom pandemije španjolske gripe 1918. godine. Istraživanje PNAS-a pokazalo je da su gradovi koji su u ranoj fazi pandemije poduzeli mjere poput zatvaranja škola i zabrane javnih okupljanja imali znatno niže stope smrtnosti. Isti rezultati pokazali su se i na primjeru korona virusa u dva kineska grada. Grad Guangzhou, koji je uveo mjere kontrole bolesti u ranoj fazi izbijanja, imao je znatno niži broj hospitalizacija, nego grad Wuhan koji je mjere uveo tek mjesec dana nakon izbijanja korona

virusa [3]. Iz navedenih primjera može se zaključiti kako mjere socijalnog distanciranja imaju veliku ulogu ne samo u sprječavanju širenja zaraze već posljedično i u smanjenju broja oboljelih s težom kliničkom slikom, smanjenju hospitalizacija te na samom kraju i smanjenju smrtnosti. Jedna od važnijih stavki pri uvođenju socijalnog distanciranja jest pravovremena reakcija koja je ključ uspješnosti te mjere.

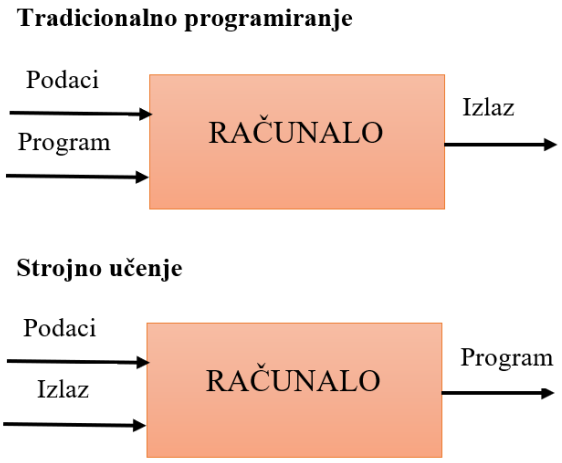
1.2. Umjetna inteligencija

Umjetna inteligencija je dio računalne znanosti koja je usmjerena na razvoj vještine računala da obavljaju zadaće za koje je potreban neki oblik inteligencije, odnosno da se samostalno snalaze u novim situacijama, uče nove koncepte, donose zaključke, razumiju i raspoznaju jezik i prizore [4]. Idealna karakteristika umjetne inteligencije je njezina sposobnost racionalizacije i poduzimanja radnji koje imaju najbolje izglede za postizanje određenog cilja. Kako tehnologija napreduje, prethodna mjerila koja su definirala umjetnu inteligenciju postaju zastarjela. Na primjer, strojevi koji izračunavaju osnovne funkcije ili prepoznaju tekst pomoću optičkog prepoznavanja znakova više se ne smatraju utjelovljenjem umjetne inteligencije budući da se takve funkcije u današnjici smatraju inherentnim funkcijama računala. Umjetna inteligencija se kontinuirano razvija kako bi pogodovala različitim industrijama. Značajnu ulogu zauzima u automatizaciji, proizvodnoj industriji, financijama, zdravstvu i obrazovanju. Može se podijeliti u dvije kategorije: slaba i jaka umjetna inteligencija. Slaba umjetna inteligencija podrazumijeva sustav dizajniran za obavljanje jednog određenog posla. Strojevi mogu reagirati i obavljati određene zadatke, no ne mogu samostalno misliti. Jaka umjetna inteligencija obuhvaća sustave koji mogu samostalno razmišljati i djelovati poput ljudi [5]. Budući da se radi o složenim i kompliciranim sustavima koji su programirani za rješavanje problema bez ljudske pomoći, ne postoje primjeri iz svakodnevnog života, no poanta cijele ideje može se prepoznati u sustavima poput autonomnih vozila. Budući da je umjetna inteligencija vrlo značajna u izradi programske aplikacije, detaljnije će se razraditi u narednim poglavljima.

2. TEORIJSKA OSNOVA RADA

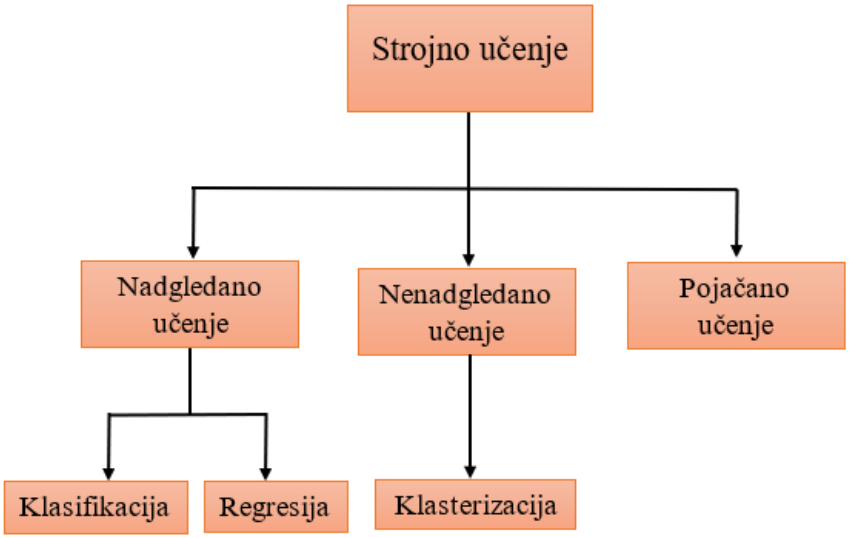
2.1. Strojno učenje

Budući da računalo samo po sebi nema načina da uči, odnosno gradi znanje na temelju pohranjenih podataka, potrebno je razviti pogodne računalne metode koje će omogućiti učenje [6]. Strojno učenje je dio računalne znanosti koji računalima daje sposobnost da uče bez da su eksplicitno programirana. Ono što razlikuje strojno učenje od tradicionalnog programiranja jest činjenica da je kod tradicionalnog programiranja potrebno ručno formulirati, odnosno kodirati pravila, dok kod strojnog učenja algoritam automatski formulira pravila iz podataka [7]. Razlika tih dvaju pojmova vidljiva je i na slici 1.



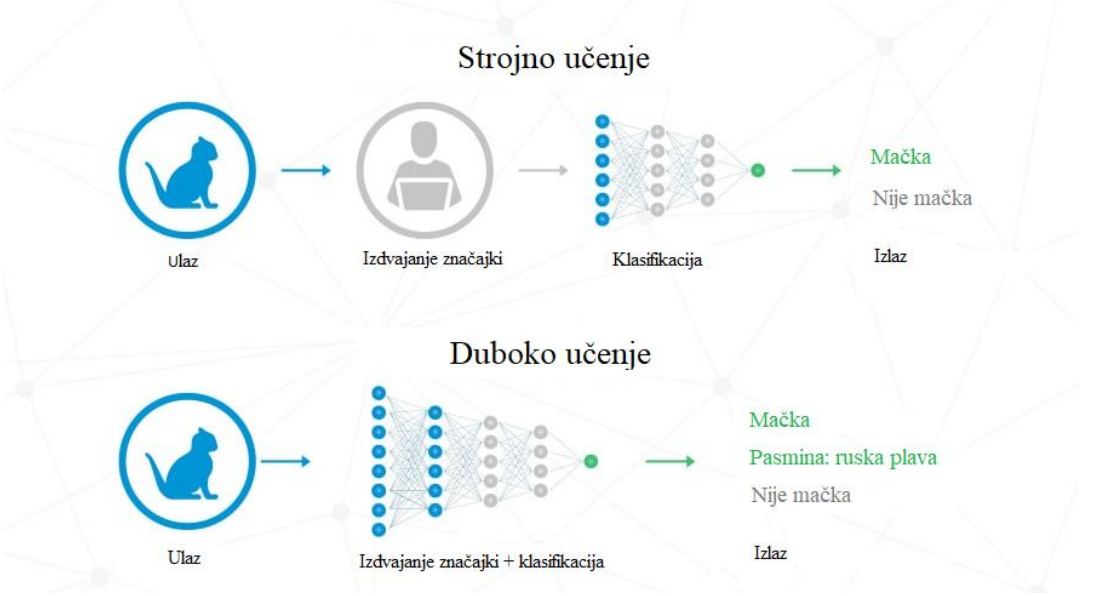
Slika 1. Razlika tradicionalnog programiranja i strojnog učenja [7]

Svaki algoritam za strojno učenje ima tri glavna dijela: reprezentacija, evaluacija i optimizacija. Reprezentacija je prostor hipoteza koji uzima u obzir činjenicu da se modeli izražavaju formalnim jezikom koji može kodirati neke modele lakše nego druge. Primjer reprezentacije mogu biti neuronske mreže. Evaluacija je način na koji se prosuđuje, odnosno, daje prednost jednom modelu u odnosu na drugi, a optimizacija je način na koji se pretražuje prostor hipoteza kako bi se pronašao najbolji model [8]. Strojno učenje može se podijeliti u nekoliko kategorija prikazanih na slici 2.



Slika 2. Podjela strojnog učenja [7]

Često se uz pojam strojnog učenja povezuje duboko učenje. U praktičnom smislu, duboko učenje je samo podskup strojnog učenja te funkcionira na sličan način. Razlika je u tome što osnovni modeli strojnog učenja postaju postupno bolji u obavljanju svojih specifičnih funkcija kako preuzimaju nove podatke, no i dalje im je potrebna određena ljudska intervencija, dok s modelom dubokog učenja algoritam može odrediti je li predviđanje točno ili ne pomoću vlastite neuronske mreže. Dakle, duboko učenje je osmišljeno za kontinuiranu analizu podataka s logičkom strukturom sličnom načinu na koji bi čovjek izvukao zaključke. Da bi se to moglo ostvariti, koristi se umjetna neuronska mreža. Ona je nadahnuta biološkom mrežom neurona u ljudskom mozgu što dovodi do sustava učenja koji je mnogo sposobniji od standardnih modela strojnog učenja [10]. Na slici 3. je prikazana razlika strojnog i dubokog učenja.



Slika 3. Razlika strojnog i dubokog učenja [11]

2.1.1. Nadgledano učenje

Nadgledano učenje temelji se na nadzoru. To znači da se u tehnici nadgledanog učenja treniraju strojevi pomoću označenog skupa podataka, a na temelju treninga stroj predviđa izlaz. Najprije se trenira s ulazom i odgovarajućim izlazom, a potom se od stroja traži da predvidi izlaz pomoću skupa testnih podataka. Neke stvarne primjene nadgledanog učenja jesu procjena rizika, otkrivanje prijevara, filtriranje neželjene pošte. Nadgledano učenje može se kategorizirati u dvije kategorije: klasifikacija i regresija. Klasifikacijski algoritmi predviđaju klase prisutne u skupu podataka, a izlazna varijabla je primjerice „da“ ili „ne“. Regresijski algoritmi koriste se za rješavanje problema u kojima postoji linearan odnos između ulaznih i izlaznih varijabli. Oni se koriste za predviđanje kontinuiranih izlaznih varijabli kao što su tržišni trendovi i vremenska prognoza. Budući da nadgledano učenje radi s označenim skupom podataka, postoji jasna predodžba o klasama objekata. Korisno je u predviđanju rezultata na temelju prethodnog iskustva. Neki od nedostataka jesu da nije pogodno za složenije zadatke te je moguća kriva prosudba izlaza ukoliko se testni podaci razlikuju od podataka korištenih u treningu [9].

2.1.2. Nenadgledano učenje

Kod nenadgledanog učenja nema potrebe za nadzorom što znači da se stroj trenira pomoću neoznačenog skupa podataka te predviđa izlaz bez ikakvog nadzora. Glavni cilj algoritma za nenadgledano učenje je grupiranje ili kategoriziranje nerazvrstanog skupa podataka prema sličnostima, uzorcima i razlikama. Strojevi dobivaju upute da pronađu skrivene uzorke iz ulaznog skupa podataka. Potkategorija nenadgledanog učenja je klasterizacija. Tehnika

klasteriranja se koristi kada se žele pronaći inherentne grupe iz podataka. To je način grupiranja objekata u klaster tako da objekti s najviše sličnosti ostanu u jednoj skupini i imaju manje ili nimalo sličnosti s objektima drugih skupina. Jedan od primjera algoritma klasteriranja je K-means. Prednost algoritma nenadgledanog učenja je da se može koristiti za kompliciranije zadatke u usporedbi s nadgledanim učenjem, a glavni nedostatak je da izlaz može biti dosta neprecizan budući da se radi o neoznačenom skupu podataka [9].

2.1.3. Pojačano učenje

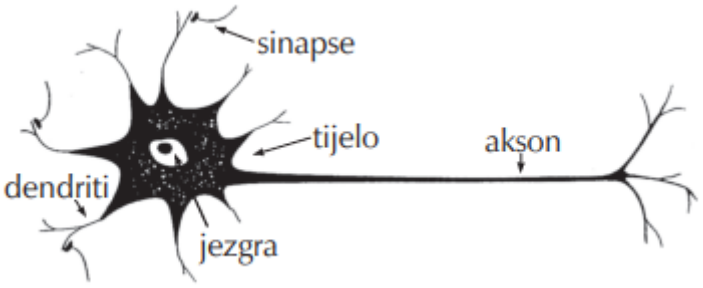
Pojačano učenje radi na procesu temeljenom na povratnim informacijama u kojem agent automatski istražuje svoje okruženje poduzimanjem radnji, učenjem iz iskustva i poboljšavanjem svoje izvedbe. Agent biva nagrađen na svaku dobru radnju i kažnjen za svaku lošu pa je cilj za potkrijepljeno učenje maksimizirati nagrade. U pojačanom učenju nema označenih podataka, a agenti uče samo iz vlastitih iskustava. Kategorije pojačanog učenja jesu pozitivno i negativno pojačano učenje. Pozitivno pojačano učenje određuje povećanje tendencije da će se traženo ponašanje ponoviti dodavanjem nečega. Pojačava snagu ponašanja agenta i pozitivno utječe na njega. Negativno pojačano učenje djeluje suprotno, odnosno povećava tendenciju da će se specifično ponašanje ponoviti izbjegavanjem negativnog stanja. Prednosti pojačanog učenja jesu rješavanje složenih problema iz stvarnog svijeta te model učenja sličan učenju ljudskih bića. Nedostaci su da zahtijeva puno podataka i proračuna te da može dovesti do preopterećenja stanja što može oslabiti rezultate [9].

2.2. Neuronske mreže

Izraz umjetna neuronska mreža izveden je iz bioloških neuronskih mreža koje razvijaju strukturu ljudskog mozga. Slično ljudskom mozgu koji ima međusobno povezane neurone, umjetne neuronske mreže također imaju neurone koji su međusobno povezani u različitim slojevima mreže te se nazivaju čvorovima. Umjetna neuronska mreža pokušava oponašati mrežu neurona koja čini ljudski mozak s namjerom da računala imaju mogućnosti razumjeti i donositi odluke poput čovjeka [12].

2.2.1. Biološka i umjetna neuronska mreža

Budući da su umjetne neuronske mreže inspirirane biološkim neuronima, najprije će se objasniti osnovni pojmovi vezani uz neurone i njihov način funkcioniranja. Na slici 4. je prikazana građa neurona.

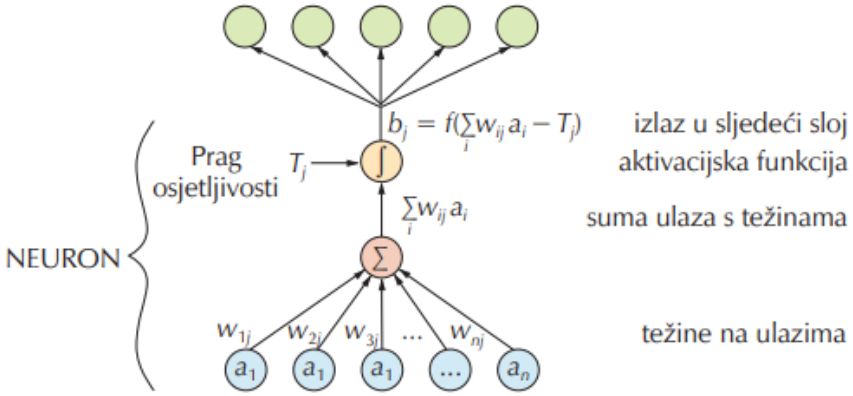


Slika 4. Građa biološkog neurona [13]

Osnovni dijelovi svakog neurona jesu:

- tijelo stanice – sadrži jezgru (nukleus) s informacijama o nasljednim značajkama
- dendriti – kraće niti oko stanice koje prenose signale (impulse) s drugih neurona
- aksoni – duge i tanke niti koje prenose signal do drugih neurona.

Sinapse su funkcionalne jedinice između završetka aksona prethodnog neurona i dendrita sljedećeg neurona koje oslobađaju neurotransmitere pri čemu se odvija elektrokemijska reakcija. Impuls se prenosi preko sinapsi s jednog na drugi neuron. Dendriti pojačavaju ili prigušuju impuls, sumiraju se u jezgri tijela te se putem aksona i sinapsi prenose na druge neurone. Prvi model umjetnog neutrona napravljen je 1943. godine te se zvao perceptron koji je prikazan na slici 5. Po uzoru na biološki neuron, obrađivao je signale putem sinaptičke i somatske operacije [13].



Slika 5. Prikaz perceptrona [13]

Svaka neuronska mreža sastoji se od ulaznog, skrivenog i izlaznog sloja. Ulazni sloj poprima vrijednosti ulaznih veličina. Sinaptička operacija predstavlja množenje svakog ulaznog signala s težinskim koeficijentom, w_i. Tako otežani ulazni signali se zbrajaju, a njihov zbroj uspoređuje se s pragom osjetljivosti neurona, T_j. Težinski faktori analogni su dendritima biološkog neurona. Skriveni sloj zbraja otežane ulaze pomoću neke funkcije sumiranja i tako stvara vlastitu internu aktivaciju. Ako je zbroj otežanih signala veći od praga osjetljivosti

neurona, nelinearna aktivacijska funkcija f generira izlazni signal neurona iznosa b_j. Prijenosna funkcija može biti diskontinuirana skokomična funkcija ili neka kontinuirana funkcija poput sigmoide ili tangens – hiperbolna funkcija [13]. U tablici 1. dana je usporedba biološkog i umjetnog neurona.

Tablica 1. Usporedba biološkog i umjetnog neurona [13]

Biološki neuron	**Umjetni neuron**
Prima ulazni signal putem dendrita	Prima ulaze *i* koji su određeni težinskim koeficijentima (*w*)
Obrada signala u somi	Obrada ulaza, unutarnji prag – bias (*b*)
Pretvara obrađeni ulaz u izlaz putem aksona	Pretvara ulaze u izlaz (prijenosna funkcija)
Šalje informacije putem sinapsi do svih neurona s kojima je neuron povezan	Šalje informaciju prema izlazu i sljedećim neuronima

2.2.2. Učenje umjetnih neuronskih mreža

Pod učenjem podrazumijeva se iterativni postupak optimiranja vrijednosti težinskih faktora na temelju pogreške između modelom proračunate vrijednosti i stvarne vrijednosti mjerene veličine. Učenje, odnosno podešavanje težinskih faktora odvija se prema pravilu širenja unatrag ili algoritma unatražne propagacije izlazne pogreške. Prilikom jednog prolaza informacije kroz neuronsku mrežu, generira se vrijednost koja se potom uspoređuje sa stvarnom vrijednošću. Na temelju razlike stvarne i izračunate vrijednosti, korigiraju se težinski faktori. Korekcijom težinskih faktora neuronska mreža uči predviđati stvarne vrijednosti te se smanjuje razlika stvarnih i predviđenih vrijednosti izlaznih veličina. Kriterij pogreške govori o kvaliteti i robusnosti mreže. Provjerom mreže na novom skupu podataka sprječava se „pretreniranje“ mreže prilikom učenja. Pretreniranje se javlja u slučaju kada mreža s visokom točnošću opisuje vladanje podataka na skupu podataka na kojem je razvijana, a izvan tog skupa pokazuje lošije rezultate. Iz tog razloga postoji tehnika ranog zaustavljanja s namjerom da se učenje neuronske mreže zaustavi u trenutku kada pogreška provjere počne rasti [13].

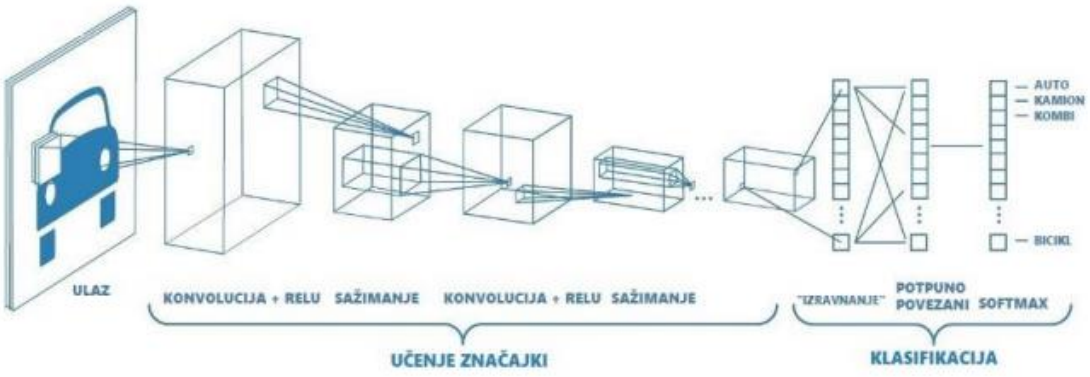
2.2.3. Podjela neuronskih mreža

Budući da postoji velik broj neuronskih mreža, teško je ih podijeliti u kategorije. Sa strukturnog gledišta neuronske mreže mogu se podijeliti na statičke unaprijedne i dinamičke ovisno o modelu neurona od kojeg su građene te po načinu prostiranja signala kroz mrežu. Kod identificiranja i vođenja nelinearnih dinamičkih procesa najčešće se rabe višeslojne statičke neuronske mreže. Od dinamičkih neuronskih mreža uglavnom se koriste višeslojne

neuronske mreže s elementima zadrške [13]. Jedna od značajnijih vrsta neuronske mreže je konvolucijska neuronska mreža.

2.2.3.1. Konvolucijske neuronske mreže

Konvolucijska neuronska mreža je mrežna arhitektura za duboko učenje te uči izravno iz podataka, eliminirajući potrebu za ručnim izdvajanjem značajki. Vrlo je značajna kod računalnog vida, odnosno za pronalaženje uzoraka na slikama za prepoznavanje objekata, lica i prizora. Poput ostalih neuronskih mreža, sastoji se od ulaznog, mnogo skrivenih i izlaznog sloja. Navedeni slojevni izvode operacije koje mijenjaju podatke s namjerom učenja značajki specifičnih za podatke. Tri najčešća sloja jesu konvolucijski slojevi, slojevi sažimanja i slojevi aktivacije. Konvolucijski slojevi (engl. *convolutional layers*) stavljaju ulazne slike kroz skup konvolucijskih filtara od kojih svaki aktivira određene značajke slika. Slojevi sažimanja (engl. *pooling layers*) pojednostavljuju izlaz izvođenjem nelinearnog smanjivanja uzorkovanja, smanjujući broj parametara koje mreža mora naučiti. Aktivacijski ili ReLU (engl. *rectified linear unit*) slojevi omogućuju brži i učinkovitiji trening mapiranjem negativnih vrijednosti na nulu i održavanjem pozitivnih vrijednosti. Naziva se tako budući da se samo aktivirane značajke prenose u sljedeći sloj [14]. Primjer konvolucijske neuronske mreže prikazan je na slici 6.



Slika 6. Primjer konvolucijske mreže [14]

2.2.4. Primjena neuronskih mreža

U primjeni strojnog učenja neuronske mreže često se koriste za klasifikaciju: prepoznavanje slika, govora, prevođenje, analiza društvenih mreža, inteligentno internetsko pretraživanje, ciljani marketing i slično [13]. Još neke primjene jesu u meteorologiji, predviđanje vremena, u industriji, kontrola kvalitete i detekcija neispravnih proizvoda, u poslovnim i investicijskim primjenama, procjene rizika od bankrota, predviđanje vrijednosti dioničkih indeksa. Budući da neuronske mreže imaju puno prednosti, njihova će primjena kroz vrijeme sve više rasti.

2.3. Računalni vid

Računalni vid je polje umjetne inteligencije koje omogućuje računalima i sustavima da izvedu značajne informacije iz digitalnih slika, videa i drugih vizualnih inputa te poduzimaju radnje ili daju preporuke na temelju tih informacija. Računalni vid funkcionira poput ljudskog, no ljudski vid ima prednost jer s lakoćom razlikuje objekte, prepoznaje koliko su udaljeni, kreću li se i slično. Računalni vid trenira strojeve za obavljanje navedenih funkcija, ali mora to učiniti u puno kraćem vremenu s kamerama, podacima i algoritmima umjesto s ljudskim okom [15].

2.3.1. Način rada

Računalni vid zahtjeva puno podataka kako bi se neprestano provodile analize podataka dok se ne uoče razlike i prepoznaju slike. Na primjer, da bi računalo bilo u mogućnosti prepoznati automobil, potrebno je unijeti puno slika automobila i predmeta povezanih s njima kako bi računalo naučilo razlike i prepoznalo ga. Da bi se to postiglo koriste se dvije tehnologije: duboko učenje i konvolucijska neuronska mreža. Duboko učenje koristi algoritamske modele koji omogućuju računalu da uči o kontekstu vizualnih podataka. Pomoću algoritma stroj sam uči bez potrebe da ga netko programira kako bi prepoznao sliku. Konvolucijske neuronske mreže pomažu dubokom učenju na način da razlažu slike na piksele kojima se daju oznake. Zatim koristi te oznake za izvođenje konvolucija (matematička operacija na dvije funkcije za proizvodnju treće funkcije) i daje predviđanja o tome što „vidi“. Neuronska mreža pokreće konvolucije i provjerava točnost svojih predviđanja u nizu ponavljanja sve dok se predviđanja ne počnu ostvarivati. Taj proces je gledanje slika, odnosno prepoznavanje na način sličan ljudima. Poput čovjeka, konvolucijska neuronska mreža prvo razaznaje oštre rubove i jednostavne oblike, a zatim popunjava informacije dok izvodi iteracije svojih predviđanja [15].

2.3.2. Primjena i primjeri računalnog vida

Zahvaljujući brojnim istraživanjima o računalnom vidu, njegova primjena sve je veća. Ključni pokretač je velika količina vizualnih informacija koje dolaze od pametnih telefona, sigurnosnih sustava, prometnih kamera i drugih uređaja. Računalni vid koristi se kod [15]:

- klasifikacije – u mogućnosti je predvidjeti kojoj klasi pripada određeni objekt; primjer toga je upotreba računalnog vida kod društvenih mreža; upotreba za automatsku identifikaciju korisnika te izdvajanje neprimjerenih slika
- detekcije objekata – u mogućnosti je koristiti klasifikaciju za identifikaciju klase određene slike, a potom detektira njihov izgled na slici ili videu; upotreba kod

otkrivanja oštećenja na pokretnoj traci ili prepoznavanje strojeva koji zahtijevaju održavanje

- praćenje objekata – često se upotrebljava sa slikama snimljenim u nizu ili primjerice kod autonomnih vozila jer je potrebno ne samo klasificirati i detektirati objekte poput pješaka, drugih automobila i cestovne infrastrukture, već ih moraju i pratiti u pokretu kako bi se izbjegli sudari te poštivali prometni znakovi
- dohvaćanje slike temeljeno na sadržaju – upotrebljava se za pregledavanje, pretraživanje i dohvaćanje slika iz velikih pohrana podataka; koristi se za sustave upravljanja digitalnom imovinom i imaju mogućnost povećanja točnosti pretraživanja.

3. PROGRAMSKA APLIKACIJA

3.1. Programski jezik Python

Python je interpretirani, interaktivni te objektno orijentirani programski jezik koji uključuje module, iznimke, dinamičko povezivanje te dinamičke tipove podataka visoke razine i klase. Podržava mnogo vrsta programiranja izvan objektnog poput proceduralnog i funkcionalnog programiranja. Osim što posjeduje veliku snagu s jasnom sintaksom, ima sučelja za mnoge sistemske pozive i biblioteke te je proširiv u C ili C++. Može se koristiti kao programski jezik za aplikacije koje trebaju programabilno sučelje te je prenosiv pa radi na mnogim varijantama Unixa uključujući Linux i MacOS te Windows [16].

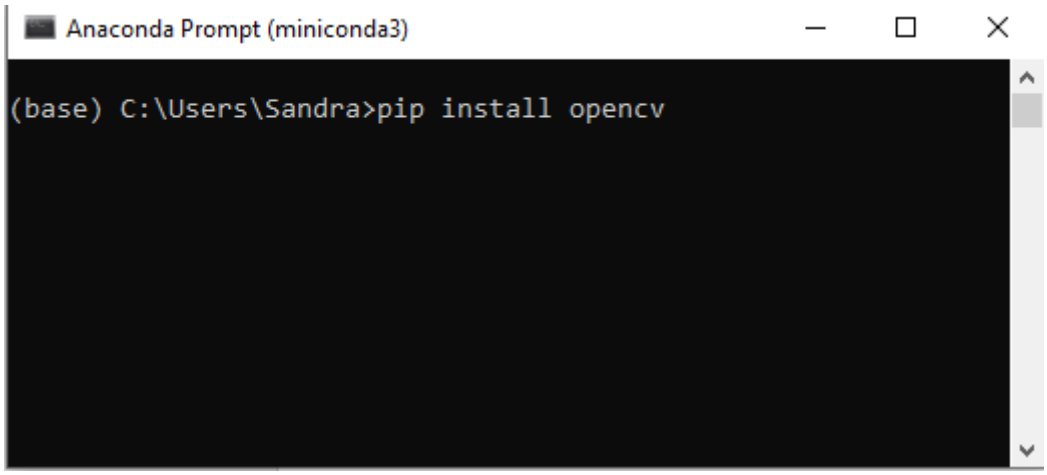
Budući da Python nudi jednostavnost, fleksibilnost, dosljednost te pristup bibliotekama i okvirima za umjetnu inteligenciju i strojno učenje, smatra se najprikladnijim za strojno učenje i projekte temeljene na umjetnoj inteligenciji. Softverske biblioteke jesu unaprijed napisani kodovi koji se koriste kod rješavanja uobičajenih programskih zadataka. Neke od njih bit će korištene u ovome radu, a spomenute su u kasnijim poglavljima [17].

3.2. Jupyter notebook

U radu je korišten Jupyter Notebook za pisanje Python koda. Jupyter Notebook je web-aplikacija otvorenog koda koja omogućuje stvaranje i dijeljenje dokumenata koji sadrže računalni kod, jednadžbe, vizualizacije i tekst. Dokumenti napisani u aplikaciji sadrže opis analiza i rezultate te su istovremeno izvršni dokumenti koji se mogu pokrenuti za analizu podataka. Osim pisanja i uređivanja, ima mogućnost izvođenja koda po dijelovima [18].

3.3. Python biblioteke

Pythonova standardna biblioteka vrlo je opsežna te nudi širok raspon mogućnosti. Biblioteke sadrže ugrađene module koji omogućuju pristup funkcionalnostima sustava koji inače ne bi bili dostupni programerima kao i module koji su napisani u Pythonu te omogućuju standardizirana rješenja za probleme koji se svakodnevno javljaju u programiranju [19]. Programi za instalaciju Pythona obično uključuju cijelu standardnu biblioteku, a moguće ih je i posebno instalirati na način da se u terminal upiše „pip install“ te u nastavku ime potrebnog paketa kao što je prikazano na slici 7. U nastavku će biti navedene korištene Pythonove biblioteke.



Slika 7. Instalacija paketa OpenCV pomoću opcije pip install

3.3.1. OpenCV

OpenCV (engl. *Open Source Computer Vision*) je biblioteka softvera za računalni vid i strojno učenje otvorenog koda. OpenCV je napravljen s namjerom da osigura zajedničku infrastrukturu za aplikacije računalnog vida i ubrza korištenje strojne percepcije u komercijalnim proizvodima. Biblioteka ima više od 2500 optimiziranih algoritama što uključuje opsežan skup klasičnih i najsuvremenijih algoritama računalnog vida i strojnog učenja. Ti se algoritmi mogu koristiti za otkrivanje i prepoznavanje lica, identifikaciju objekata, klasifikaciju ljudskih radnji u videozapisima, praćenje pokreta kamere i pokretnih objekata i slično. Ima C++, Python, Java i MATLAB sučelja te podržava Windows, Linux, Android i MacOS [20].

3.3.2. Imutils

Imutils je paket temeljen na OpenCV-u te može na jednostavniji način pozivati OpenCV sučelje. Neke od funkcija Imutilsa jesu [21]:

- prijevod slike – može izravno odrediti prevedene piksele bez konstruiranja matrice prijevoda
- skaliranje slike – omjer izvorne slike se automatski održava, a mogu se specificirati samo širina, težina i visina
- rotacija slike – rotaciju slike prate dva parametra, prvi su sami podaci slike, a drugi kut rotacije koja je u smjeru suprotnom od kazaljke na satu
- ekstrakcija rubova – odnosi se na proces konstruiranja topološkog skeleta objekata na slici.

3.3.3. NumPy

NumPy je biblioteka Pythona koja sadrži višedimenzionalne objekte u nizovima i paket integrirajućih alata za implementaciju Pythona. U osnovi je kombinacija C-a i Pythona koji se upotrebljava za tradicionalno MATLAB programiranje gdje se podaci u obliku brojeva tretiraju poput nizova za višedimenzionalne funkcije i operacije preuređivanja. NumPy znatno olakšava rad s velikim nizovima i matricama korištenjem različitih matematičkih funkcija. Također pruža funkcije koje omogućuju izvođenje osnovnih, ali i naprednih matematičkih i statističkih funkcija na višedimenzionalnim nizovima i matricama s malo redaka koda. Iako je NumPy brži u usporedbi s Python listama, glavni nedostatak je da se samo jedna vrsta podataka može pohraniti u svaki stupac [22].

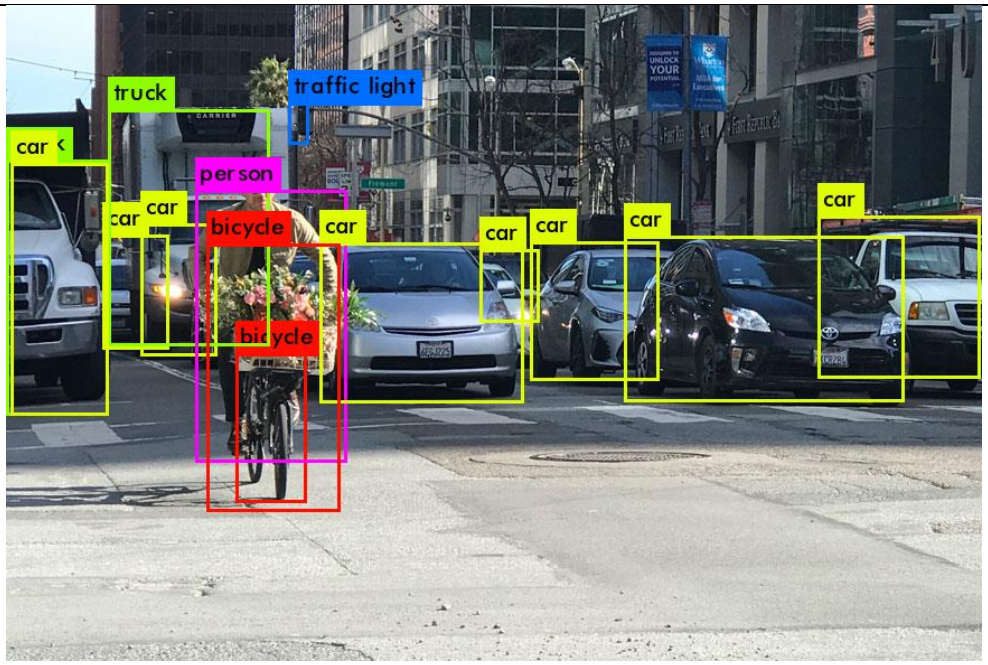
3.3.4. SciPy

SciPy je softver otvorenog koda za matematiku, znanost i inženjerstvo. Biblioteka SciPy je napravljena za rad s NumPy-om gdje pruža algoritme za optimizaciju, integraciju, interpolaciju, rješava probleme svojstvenih vrijednosti, algebarske jednadžbe, diferencijalne jednadžbe, statistiku i brojne druge probleme. SciPy proširuje NumPy na način da pruža dodatne alate za računalstvo nizova te pruža specijalizirane strukture podataka. Obuhvaća visoko optimizirane implementacije napisane u jezicima poput C i C++ [23].

3.4. Osnovne značajke programske aplikacije

3.4.1. Detekcija objekata

Za detekciju objekata korišten je YOLOv3. YOLOv3 (engl. *You Only Look Once, version 3*) je algoritam za otkrivanje objekata u stvarnom vremenu koji identificira određene objekte na videozapisima ili slikama. YOLO algoritam strojnog učenja koristi značajke konvolucijske neuronske mreže koja se temelji na klasifikatorima koji mogu obraditi ulazne slike kao strukturirane nizove podataka i prepoznati uzorke među njima. Omogućuje modelu da sagleda širu sliku tijekom testiranja pa su na taj način predviđanja utemeljena na cijeloj slici. YOLO, poput i drugih algoritama konvolucijske neuronske mreže, „boduje“ regije slike na temelju njihove sličnosti s unaprijed definiranim klasama. Regije s visokim bodovanjem označavaju se kao pozitivne detekcije one klase s kojom se najbliže identificiraju [24]. Iz rezultata detekcije u ovome je radu korištena samo klasa ljudi budući da se ostale klase objekata zanemaruju. Na slici 8. prikazana je primjena YOLO algoritma u prometu.



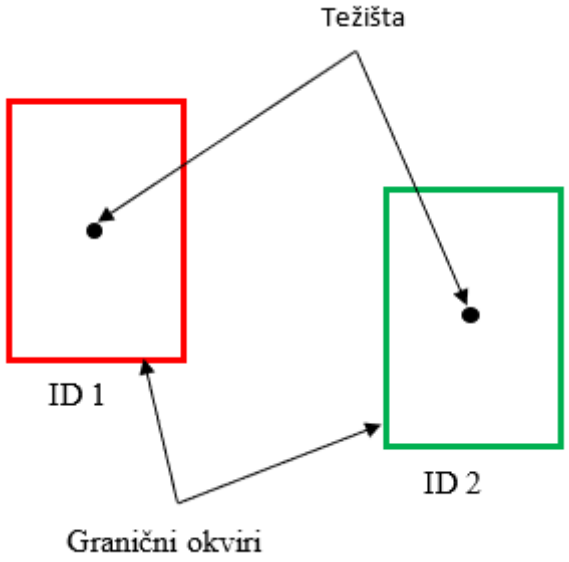
Slika 8. Primjena YOLOv3 u prometu [25]

3.4.2. Praćenje objekata

Budući da se radi o detekciji kršenja socijalnog distanciranja, potrebno je praćenje objekata, odnosno ljudi, kako bi se ustanovilo tko se pridržava, a tko krši navedenu mjeru. Kako bi se to realiziralo, svakom prepoznatom objektu (čovjeku) dodjeljuje se jedinstveni ID (engl. *identity document*) te se oko njega stvara granični okvir. Unutar graničnih okvira postoje težišta koja će u kasnijim koracima predstavljati pojedinog čovjeka. U radu će se koristiti videozapis ljudi koji se kreću u prostoru pa je stoga potrebno da granični okvir dodijeljen određenom ID-u prati upravo taj ID. Drugim riječima, potrebno je prepoznati razliku između jedno te istog čovjeka u pokretu i drugog čovjeka. Za to će se koristiti euklidska udaljenost koja će se mjeriti između svih težišta u pojedinom trenutku. Formula po kojoj se ona računa prikazana je na slici 9. gdje $\boldsymbol{d}$ označava udaljenost između dva čovjeka, pozicija jednog čovjeka označena je koordinatama $\boldsymbol{x_1, y_1}$, a pozicija drugog čovjeka označena je koordinatama $\boldsymbol{x_2, y_2}$. Ona udaljenost koja je zabilježena kao najmanja, predstavlja pomak jednog ID-a, odnosno čovjeka, dok su preostale udaljenosti od drugih ljudi. Granični okvir će poprimiti crvenu boju ukoliko je mjera prekršena, a zelenu ukoliko je mjera ispoštovana što je vidljivo na slici 10.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

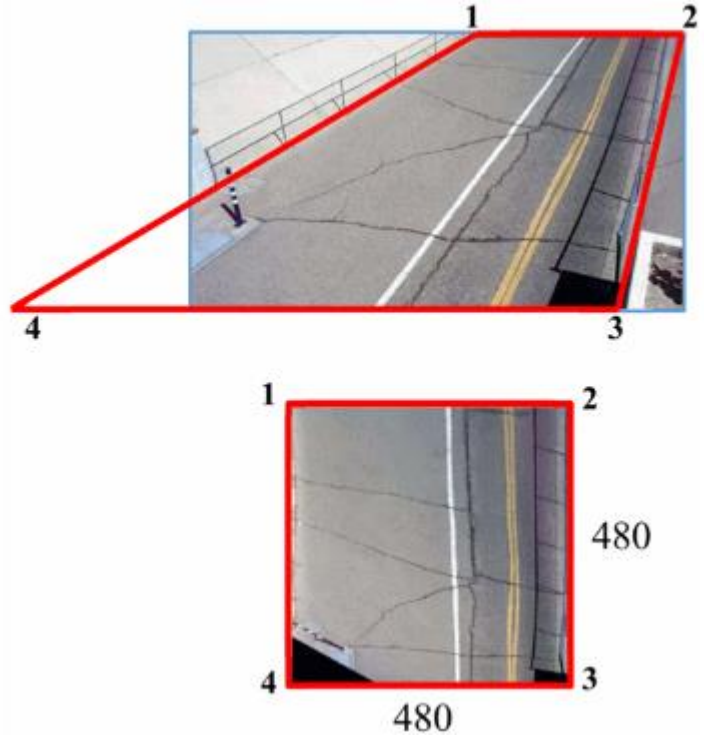
Slika 9. Formula za izračun euklidske udaljenosti [26]



Slika 10. Prikaz graničnih okvira i težišta

3.4.3. Mjerenje udaljenosti

Kako bi se izmjerila udaljenost između ljudi koristi se formula za izračun euklidske udaljenosti. Područje interesa slike fokusirane na ljude u prolazu transformira se u 2D pogled odozgo prema dolje, odnosno u tlocrt. U OpenCV-u transformacija perspektive je jednostavna metoda kalibracije kamere koja uključuje odabir četiri točke u perspektivnom prikazu i njihovo mapiranje u kutove pravokutnika u 2D prikazu slike što je vidljivo iz slike 11. Samim time pretpostavlja se da svaka osoba stoji u istoj razini ravnine. Stvarna udaljenost između ljudi odgovara broju piksela u tlocrtu. Dakle, lokacija pojedinog graničnog okvira u perspektivnom prikazu transformira se u tlocrtni pogled. Za svakog čovjeka položaj u tlocrtnom pogledu procjenjuje se na temelju težišta. Udaljenost između dvoje ljudi može se izračunati iz tlocrta, a udaljenost se skalira pomoću faktora skaliranja procijenjenog iz kalibracije pogleda kamere [26]. Kao što je prethodno navedeno, pomoću koordinata svakog čovjeka, izračunava se njihova međusobna euklidska udaljenost.



Slika 11. Transformacija u 2D pogled [26]

3.5. Izrada programske aplikacije

3.5.1. Postavljanje vrijednosti varijabli

Na samom početku izrade programske aplikacije potrebno je postaviti vrijednosti određenih varijabli. Varijabla MIN_CONF označava minimalnu pouzdanost pri detekciji objekata, NMS_THRESH označava klasu algoritma za odabir jednog entiteta od mnogih entiteta koji se preklapaju (npr. granični okviri) dok posljednja varijabla MIN_DISTANCE označava minimalnu sigurnu udaljenost između dvije osobe. Njezina vrijednost postavlja se na 50 piksela. Navedeno je prikazano na slici 12.

```
# inicijalizacija minimalne vjerojatnosti za filtriranje slabih detekcija
# i praga za primjenu metode Non-Maximum Suppression
MIN_CONF = 0.3
NMS_THRESH = 0.3

# definiranje minimalne sigurne udaljenosti (u pikselima) na kojoj
# dvije osobe mogu međusobno biti
MIN_DISTANCE = 50
```

Slika 12. Prikaz postavljanja varijabli

3.5.2. Kreiranje funkcije za detekciju ljudi

3.5.2.1. Učitavanje potrebnih biblioteka

Kao što je prethodno u radu spomenuto, potrebno je učitati odgovarajuće biblioteke i module koji će omogućiti izradu programske aplikacije. Na slici 13. prikazano je učitavanje NumPy-a i OpenCV-a (cv2 jest ime starije verzije OpenCV-a), a njihove karakteristike navedene su u poglavlju 3.3.

```
# učitavanje potrebnih biblioteka
import numpy as np
import cv2
```

Slika 13. Prikaz učitanih biblioteka Pythona

3.5.2.2. Definiranje funkcije

Definira se funkcija detect_people koja obuhvaća četiri parametra:

- frame – označava okvir iz video datoteke ili izravno iz web kamere
- net – prethodno inicijalizirani i trenirani YOLO model otkrivanja objekata
- ln – nazivi izlaznog sloja
- personIdx – predstavlja klasu ljudi budući da se drugi objekti neće razmatrati.

Nakon toga dohvaćaju se dimenzije okvira za potrebe skaliranja ((H,W) =frame.shape [:2]) te se inicijalizira lista rezultata koju vraća funkcija. Rezultati se sastoje od vjerojatnosti predviđanja osobe, koordinata graničnog okvira za detekciju i težišta objekta. Potom se kreira blobFromImage funkcija koju omogućuje OpenCV, a koristi se za olakšavanje pretprocesiranja slike za klasifikaciju dubokog učenja. Navedena funkcija obavlja srednje oduzimanje, skaliranje i opcionalno zamjenu kanala [27]. U ovom konkretnom primjeru, 1/255.0 predstavlja faktor skaliranja, 416x416 označava željenu veličinu, swapRB=True označava zamjenu R i B kanala na slici budući da OpenCV pretpostavlja da su slike u BGR redoslijedu (blue, green, red). Na kraju se inicijaliziraju popisi koji će potom sadržavati granične okvire, težišta objekata i pouzdanosti otkrivanja objekata. Navedeno je prikazano na slici 14.

```
def detect_people(frame, net, ln, personIdx=0):
# dohvaćanje dimenzija okvira i inicijalizacija liste rezultata
(H, W) = frame.shape[:2]
results = []

# kreiranje blob funkcije iz ulaznog okvira te prosljeđivanje
# YOLO detektoru objekata što rezultira graničnim okvirima
# i pridruženim vjerojatnostima
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
 swapRB=True, crop=False)
net.setInput(blob)
layerOutputs = net.forward(ln)

# inicijalizacija popisa otkrivenih graničnih okvira, težišta
# i pouzdranosti
boxes = []
centroids = []
confidences = []
```

Slika 14. Prikaz kreiranja funkcije za detekciju ljudi

Idući korak je izvođenje petlje kroz svaki od izlaza sloja i detekcije te izdvajanje ID klase i pouzdanosti, odnosno vjerojatnosti trenutne detekcije što je prikazano na slici 15.

```
# petlja preko svakog izlaza sloja
for output in layerOutputs:
 # petlja preko svake detekcije
 for detection in output:
   # izdvajanje ID klase i pouzdanosti (vjerojatnosti)
   # trenutne detekcije objekta
scores = detection[5:]
classID = np.argmax(scores)
confidence = scores[classID]
```

Slika 15. Izvođenje petlje i izdvajanje ID klase i vjerojatnosti

Nakon toga slijedi provjera je li trenutna detekcija osoba i zadovoljava li se uvjet minimalne pouzdanosti. Pretpostavljajući da je tako, izračunavaju se koordinate graničnog okvira te se traži težište. Detekcija se skalira, odnosno množi s prethodno definiranim dimenzijama okvira. Koristeći koordinate graničnog okvira, izvode se gornje lijeve koordinate za objekt. Na kraju se ažuriraju liste graničnih okvira, težišta i pouzdanosti. Navedeno je prikazano na slici 16.

```
# filtriranje detekcija (1) osiguravajući da je detektirani
# objekt osoba i da je minimalna pouzdanost
# ispunjena (2)
if classID == personIdx and confidence > MIN_CONF:
    # skaliranje koordinata graničnog okvira u odnosu na
    # veličinu slike imajući na umu da YOLO vraća
    # središnje (x,y) koordinate graničnog okvira
    # iza kojih slijedi širina i visina
    # okvira
    box = detection[0:4] * np.array([W, H, W, H])
    (centerX, centerY, width, height) = box.astype("int")

    # korištenje središnjih (x,y) koordinata za pronalazak gornjeg
    # lijevog kuta graničnog okvira
    x = int(centerX - (width / 2))
    y = int(centerY - (height / 2))

    # ažuriranje liste koordinata graničnih okvira,
    # težišta i pouzdranosti
    boxes.append([x, y, int(width), int(height)])
    centroids.append((centerX, centerY))
    confidences.append(float(confidence))
```

Slika 16. Izračun koordinata graničnog okvira

Posljednji korak u kreiranju funkcije za detekciju ljudi je izvođenje funkcije Non-Maximum suppression je potiskivanje slabih graničnih okvira koji se međusobno preklapaju. Ta je metoda ugrađena u OpenCV i rezultira detekcijama Idx-ovima. Pod pretpostavkom da navedena metoda daje barem jednu detekciju, vrši se petlja te se izdvajaju koordinate graničnog okvira. Nova, ažurirana lista rezultata sastoji se od:

- pouzdanosti otkrivanja pojedine osobe
- graničnog okvira pojedine osobe
- težišta pojedine osobe.

Na kraju se vraćaju rezultati pozivnoj funkciji. Navedeno je prikazano na slici 17.

```
# primjena metode non-maximum suppression za suzbijanje slabih i
# preklapajućih graničnih okvira
idxs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONF, NMS_THRESH)

# provjera postoji li barem jedna detekcija
if len(idxs) > 0:
    # petlja kroz indekse koji ostaju
    for i in idxs.flatten():
        # izdvajanje koordinata graničnog okvira
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        # ažuriranje liste rezultata koja se sastoji od vjerojatnosti
        # predvđanja osobe, koordinata graničnog okvira
        # i težišta
        r = (confidences[i], (x, y, x + w, y + h), centroids[i])
        results.append(r)

# vraćanje liste rezultata
return results
```

Slika 17. Funkcija Non-Maximum suppression

3.5.3. Dohvaćanje graničnih okvira te proces predviđanja mjerenjem udaljenosti detektiranih ljudi

3.5.3.1. Učitavanje potrebnih biblioteka

Potrebno je učitati odgovarajuće biblioteke i module koji će omogućiti izradu programske aplikacije. Osim već prethodno navedenih NumPy-a i OpenCV-a, u ovom se slučaju uvode i neke nove biblioteke. To su imutils, koji je detaljnije obrađen u poglavlju 3.3. i modul argparse koji se koristi za raščlambu argumenata naredbenog retka. Također, omogućuje korisniku fleksibilnost i mogućnost ponovne upotrebe napisanog koda [28]. Na slici 18. prikazano je učitavanje svih potrebnih biblioteka.

```
# učitavanje potrebnih biblioteka
from TheLazyCoder import social_distancing_config as config
from TheLazyCoder.detection import detect_people
from scipy.spatial import distance as dist
import numpy as np
import argparse
import imutils
import cv2
import os
```

Slika 18. Prikaz učitanih biblioteka Pythona

3.5.3.2. Definiranje funkcije

Nakon učitavanja potrebnih biblioteka i modula, prosljeđuju se argumenti putem naredbenog retka, odnosno terminala:

- input – put do opcionalne video datoteke; ukoliko nije naveden put to određene datoteke, prema zadanim postavkama koristi se prva web kamera računala
- output – put do izlazne, tj. obrađene video datoteke; ukoliko ovaj argument nije naveden, obrađeni video neće biti izvezen na disk
- display – prema zadanim postavkama prikazat će se aplikacija za socijalno distanciranje na zaslonu dok se obrađuje pojedini okvir.

Navedeno je prikazano na slici 19.

```
# analiza argumenata
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", type=str, default="",
    help="path to (optional) input video file")
ap.add_argument("-o", "--output", type=str, default="",
    help="path to (optional) output video file")
ap.add_argument("-d", "--display", type=int, default=1,
    help="whether or not output frame should be displayed")
args = vars(ap.parse_args())
```

Slika 19. Analiza argumenata

Slijedi učitavanje oznake klase COCO na kojima se trenirao YOLO model. COCO (engl. *Common Objects in Context*) je baza podataka koja omogućuje buduća istraživanja za otkrivanje objekata, segmentaciju instanci, opise slika i lokalizaciju ključnih točaka ljudi [29]. Također, definira se put do datoteka prikazan na slici 20.

```
# učitavanje oznake klase COCO na kojima se trenirao YOLO model
labelsPath = os.path.sep.join([config.MODEL_PATH, "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# put do datoteka YOLO weights i konfiguracije modela
weightsPath = os.path.sep.join([config.MODEL_PATH, "yolov3.weights"])
configPath = os.path.sep.join([config.MODEL_PATH, "yolov3.cfg"])
```

Slika 20. Učitavanje COCO klase i YOLO puteva

Koristeći OpenCV modul duboke neuronske mreže učitava se YOLO mreža u memoriju. Ukoliko je postavljena opcija USE_GPU u konfiguraciji, tada je pozadinski procesor postavljen da bude GPU koji podržava NVIDIA CUDA. Ako GPU ne podržava CUDA-u, potrebno je provjeriti je li opcija konfiguracije postavljena na FALSE kako bi se korišten procesor prebacio na CPU. Cijeli postupak vidljiv je na slici 21. GPU (engl. *Graphics processing unit*) je procesor koji se sastoji od puno manjih i više specijaliziranih jezgri. Radeći zajedno, jezgre daju velike performanse kada se zadatak obrade može podijeliti i obraditi u više jezgri. CPU (engl. *Central processing unit*) je izrađen od milijuna tranzistora te može imati više jezgri za obradu. Neophodan je za sve moderne računalne sustave jer izvršava naredbe i procese potrebne za računalno i operativni sustav [30]. CUDA je računalna platforma i programski model koji je razvila NVIDIA za općenito računarstvo na grafičkim procesorskim jedinicama (GPU) [31].

```
# učitavanje YOLO detektora objekata treniranog na skupu podataka COCO (80 klasa)
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# provjera hoće li se koristiti GPU
if config.USE_GPU:
    # postavljanje CUDA-e kao željenu pozadinu i cilj
    print("[INFO] setting preferable backend and target to CUDA...")
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
```

Slika 21. Učitavanje YOLO-a i provjera GPU-a

Slijedi prikupljanje imena izlaznih slojeva iz YOLO-a koja će biti potrebna za obradu rezultata. Pokreće se videozapis, video datoteka preko input argumenata ili prijenos uživo putem kamere. U početku se inicijalizira pisač izlaznog videozapisa na None što je vidljivo na slici 22. Daljnje postavljanje odvija se u petlji obrade okvira.

```
# određivanje izlaznih naziva slojeva koji se koriste od YOLO-a
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# inicijalizacija videa i pokazivač na izlaznu video datoteku
print("[INFO] accessing video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)
writer = None
```

Slika 22. Pokretanje videozapisa

Pokreće se petlja kroz okvira iz videozapisa. Dimenzije ulaznog videa za testiranje jesu prilično velike pa se stoga mijenja veličina svakog okvira, a da se pri tome zadržava omjer slike. Koristeći funkciju detect_people koja je prethodno implementirana, dohvaćaju se rezultati YOLO detekcije objekata. Potom se inicijalizira skup indeksa koji podrazumijevaju kršenje socijalnog distanciranja. Provodi se kod prikazan na slici 23.

```
# petlja kroz okvire iz videozapisa
while True:
    # čitanje sljedećeg okvira iz datoteke
    (grabbed, frame) = vs.read()

    # ako okvir nije dohvaćen, došlo je do kraja
    # videozapisa
    if not grabbed:
        break

    # promjena veličine okvira i otkrivanje ljudi u njima
    frame = imutils.resize(frame, width=700)
    results = detect_people(frame, net, ln,
        personIdx=LABELS.index("person"))

    # inicijalizacija skupa indeksa koji krše mjeru socijalnog
    # distanciranja
    violate = set()
```

Slika 23. Dohvaćanje rezultata YOLO detekcije objekata

Uz pretpostavku da su u kadru otkrivene najmanje dvije osobe, izračunava se prethodno spomenuta euklidska udaljenost između svih parova težišta. Izvodi se petlja kroz gornji trokut matrice udaljenosti budući da je matrica simetrična. Provjerava se krši li izračunata udaljenost minimalnu dozvoljenu socijalnu distancu. Ukoliko su dvije osobe preblizu, dodaju se u skup za kršenje mjere. Cijeli postupak prikazan je na slici 24.

```
# provjera jesu li otkrivene najmanje dvije osobe
# (potrebno za izračun udaljenosti)
if len(results) >= 2:
    # izdvajanje svih težišta iz rezultata i izračun
    # euklidske udaljenosti između svih parova težišta
    centroids = np.array([r[2] for r in results])
    D = dist.cdist(centroids, centroids, metric="euclidean")

    # petlja kroz gornji trokut matrice udaljenosti
    for i in range(0, D.shape[0]):
        for j in range(i + 1, D.shape[1]):
            # provjera je li udaljenost između bilo koja dva para
            # težišta manja od konfiguriranog broja
            # piksela
            if D[i, j] < config.MIN_DISTANCE:
                # ažuriranje skupa kršenja distanciranja s indeksima
                # parova težišta
                violate.add(i)
                violate.add(j)
```

Slika 24. Provjera udaljenosti

Izdvajaju se granični okvir i koordinate težišta, inicijalizira se boja graničnog okvira na zelenu te se provjerava postoji li indeks u skupu kršenja mjere. Ako postoji, boja se ažurira u crvenu. Svaki granični okvir poprima odgovarajuću boju ovisno o tome krši li se mjera. Ispisuje se informacija o ukupnom broju kršenja mjere što je prikazano na slici 25.

```
# petlja kroz rezultate
for (i, (prob, bbox, centroid)) in enumerate(results):
    # izdvajanje graničnih okvira i koordinata težišta te
    # inicijalizacija boje zabilješke
    (startX, startY, endX, endY) = bbox
    (cX, cY) = centroid
    color = (0, 255, 0)

    # ako par indeksa postoji u skupu kršenja mjere,
    # ažurira se boja
    if i in violate:
        color = (0, 0, 255)

    # crta se (1) granični okvir oko osobe i (2)
    # koordinate težišta osobe
    cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
    cv2.circle(frame, (cX, cY), 5, color, 1)

# zapis ukupnog broja kršenja socijalnog distanciranja
# na izlaznom okviru
text = "Social Distancing Violations: {}".format(len(violate))
cv2.putText(frame, text, (10, frame.shape[0] - 25),
    cv2.FONT_HERSHEY_SIMPLEX, 0.85, (0, 0, 255), 3)
```

Slika 25. Detekcija kršenja mjere socijalnog distanciranja

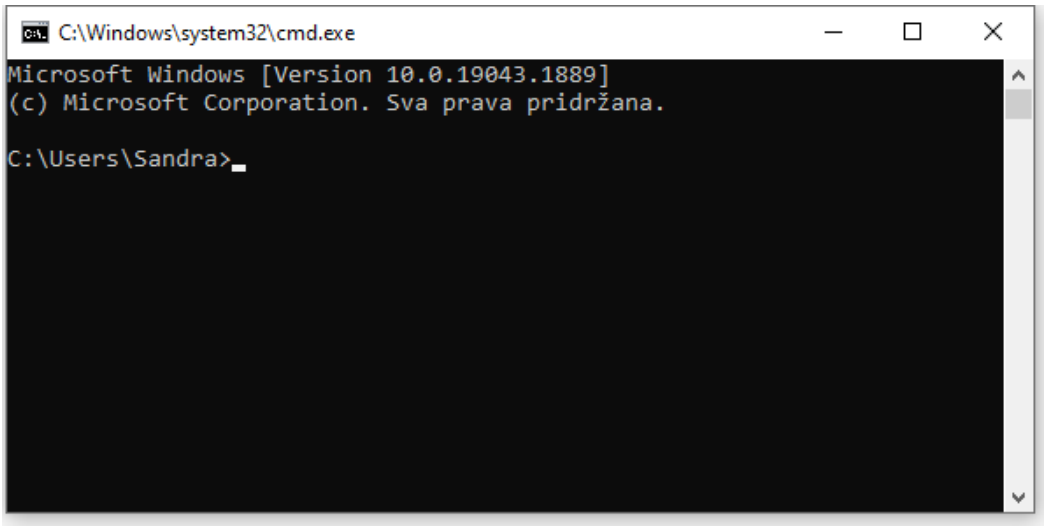
Na samome kraju prikazuje se izlazni okvir na ekranu, ukoliko je to potrebno, inicijalizira se program za snimanje videozapisa te se obrađeni okvir zapisuje na disk prema slici 26.

```
# provjera je li izlazni okvir prikazan na
# zaslonu
if args["display"] > 0:
    # prikaz izlaznog okvira
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # pritiskom na tipku "q" prekida se petlja
    if key == ord("q"):
        break

# ako je isporučen put izlazne video datoteke, a program za snimanje
# videozapisa nije inicijaliziran, to se sada učini
if args["output"] != "" and writer is None:
    # inicijalizacija programa za snimanje videozapisa
    fourcc = cv2.VideoWriter_fourcc(*"MJPG")
    writer = cv2.VideoWriter(args["output"], fourcc, 25,
        (frame.shape[1], frame.shape[0]), True)

# ako program za snimanje videozapisa nije None, zapisuje se okvir
# u izlaznu videodatoteku
if writer is not None:
    writer.write(frame)
```

Slika 26. Inicijalizacija videozapisa

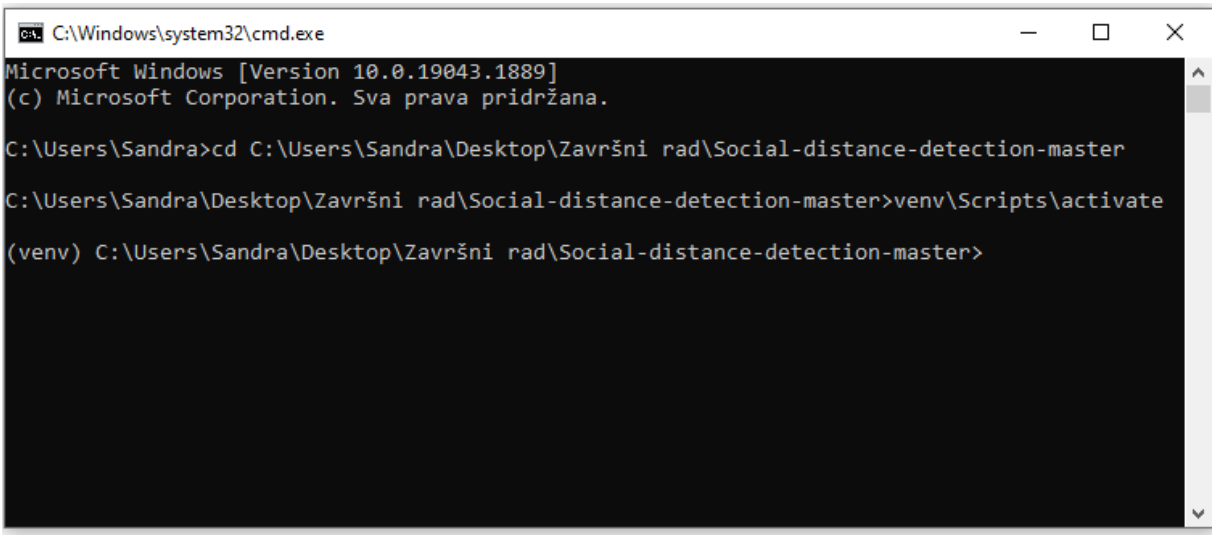
3.6. Izvođenje programske aplikacije

Kako bi se implementirao prethodno objašnjen programski kod potrebno je samo nekoliko koraka. Najprije je potrebno upaliti terminal prikazan na slici 27.



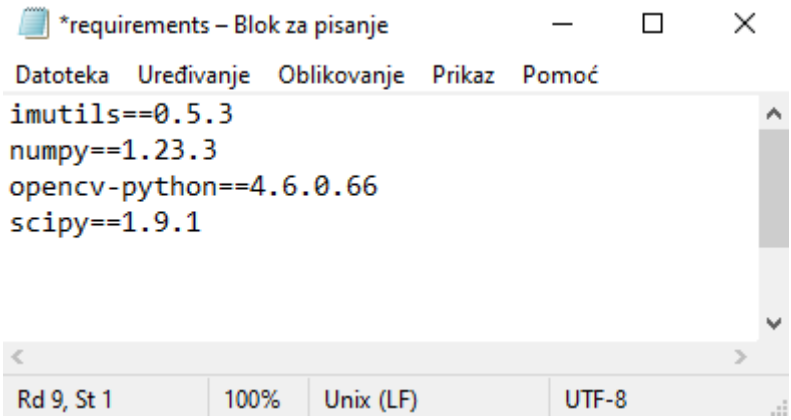
Slika 27. Windows terminal

Potom se upisuje adresa mape u kojoj se nalazi kod i svi potrebni dodaci s predmetkom cd koji označava promjenu direktorija. Upisuje se naredba pip install virtualenv kako bi se kreirala virtualna okolina te joj se daje naziv venv Za aktivaciju navedene virtualne okoline upisuje se naredba venv\Scripts\activate. Na slici 28. vidljiva je aktivna virtualna okolina.

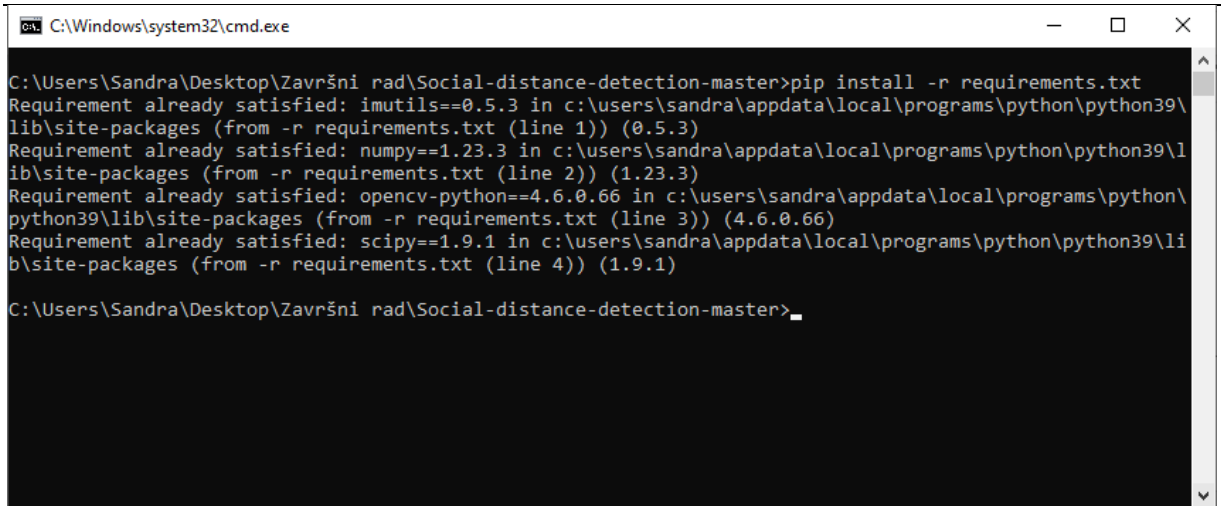


Slika 28. Aktivacija virtualne okoline

Idući korak je instalacija svih potrebnih biblioteka i modula pomoću opcije pip install. Naredba za izvođenje navedenog glasi pip install -r requirements.txt, a datoteka requirements sadrži listu svih biblioteka i njihovih verzija koje su potrebne za izvođenje programske aplikacije. Lista biblioteka prikazana je na slici 29. Uspješna instalacija svih biblioteka prikazana je na slici 30.

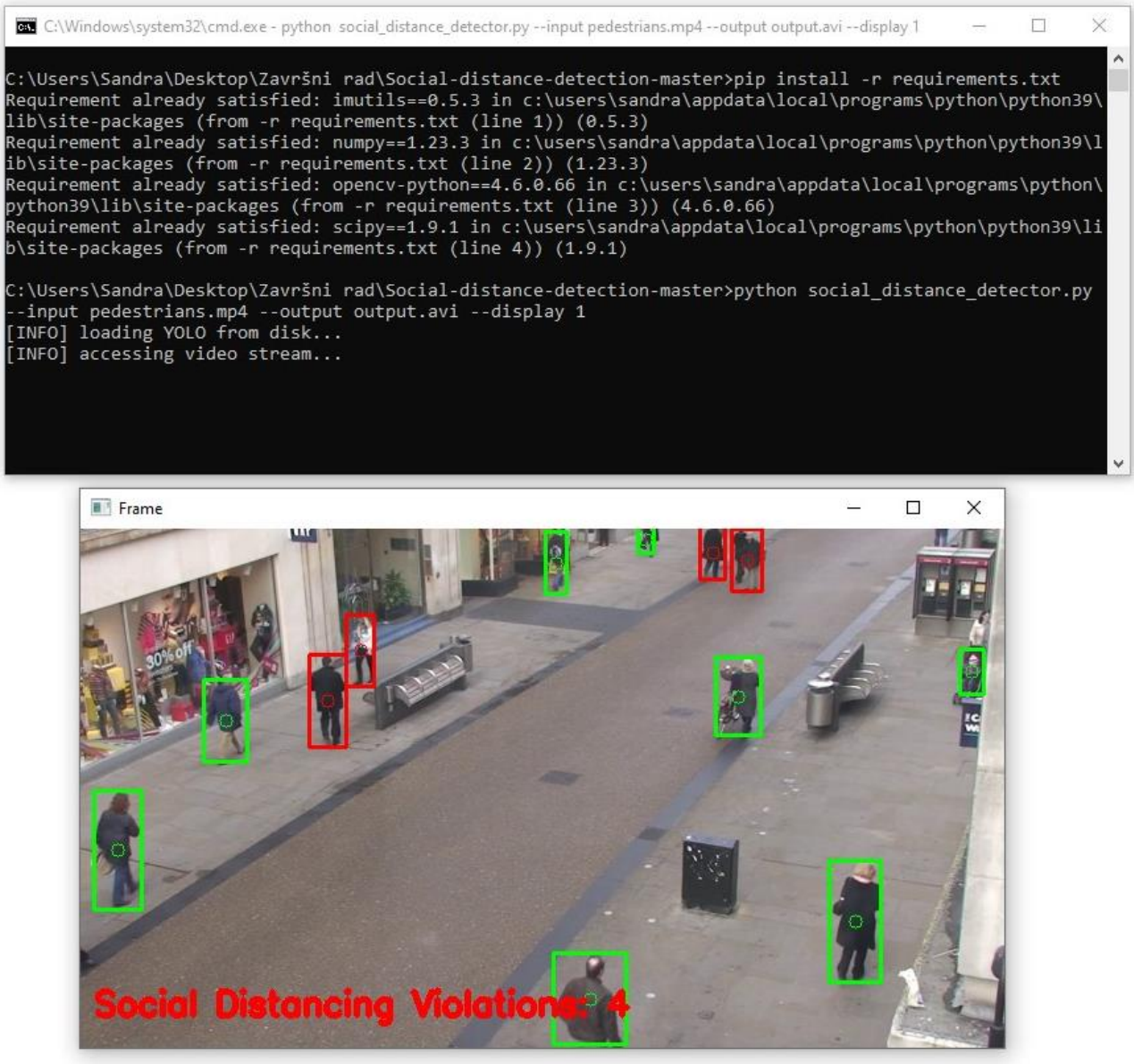


Slika 29. Lista potrebnih biblioteka

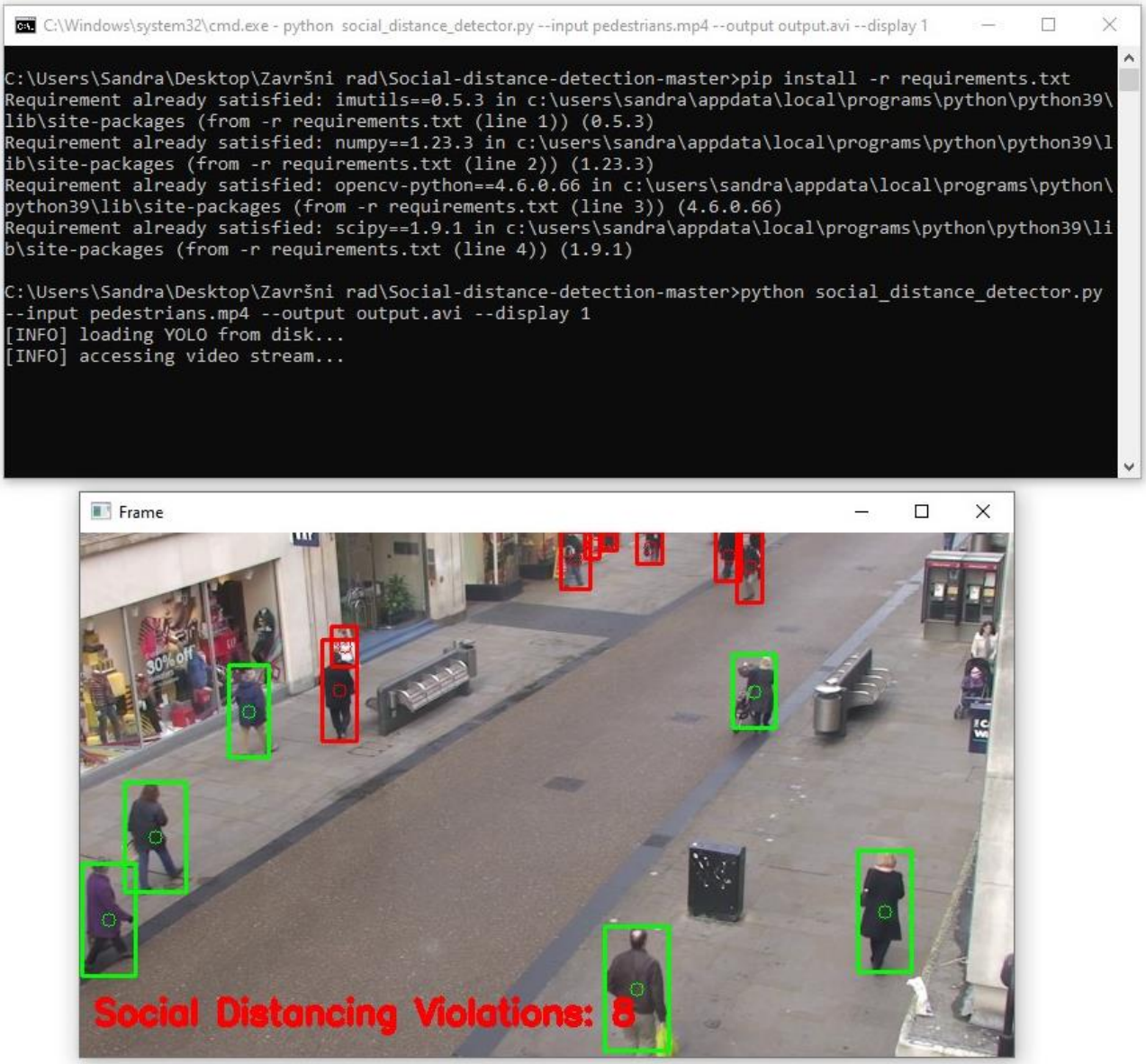


Slika 30. Instalirane biblioteke

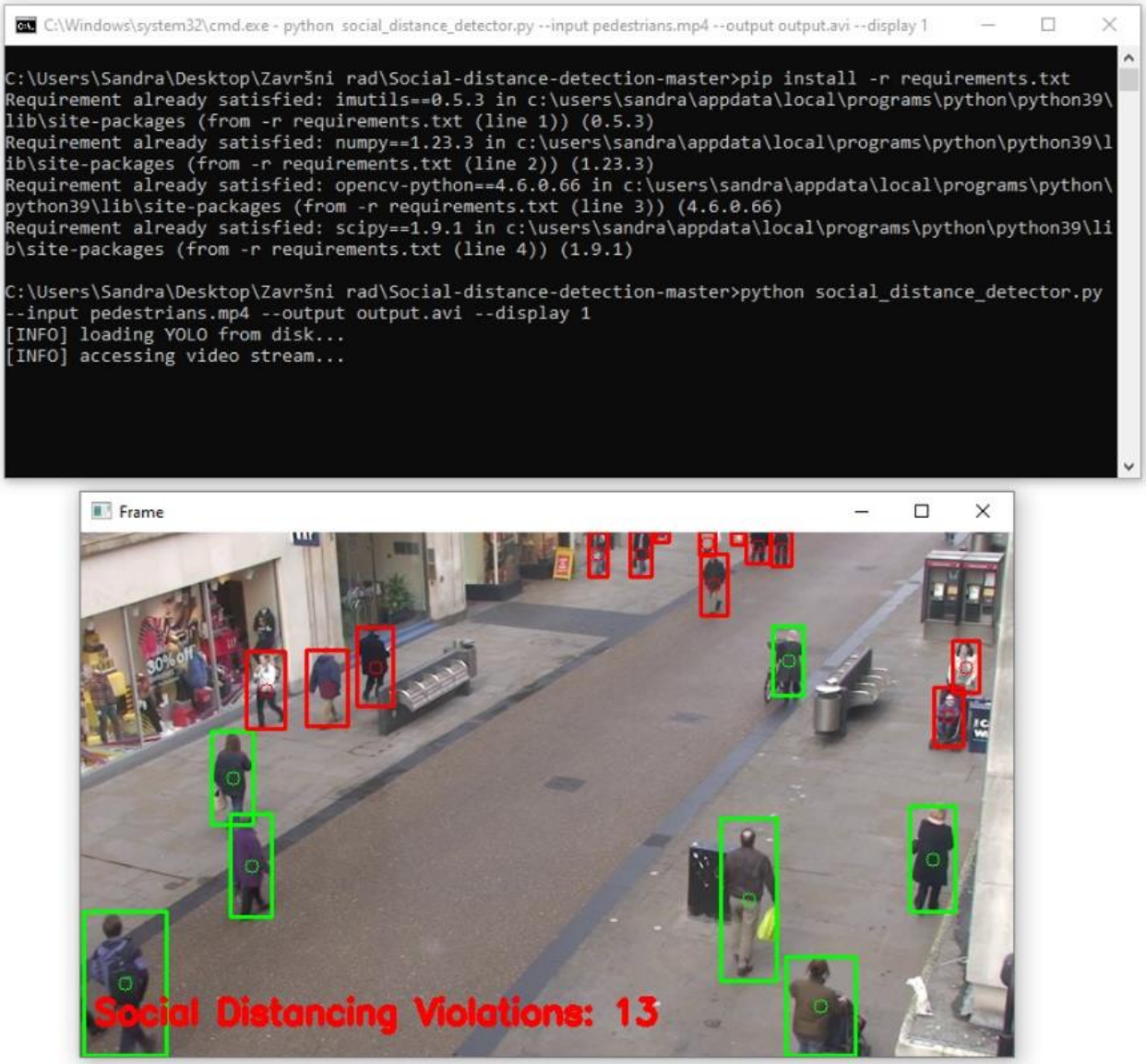
Kada je sve potrebno instalirano, upisuje se naredba python social_distance_detector.py --input pedestrians.mp4 --output output.avi --display 1 nakon čega se učitava YOLO klasa te videozapis koji se nalazi u mapi s kodom i svim potrebnim dodacima za funkcioniranje programske aplikacije. Nakon nekoliko trenutaka pojavljuje se prozor koji prikazuje videozapis ljudi u prolazu. Svaki pojedinac okružen je graničnim okvirom odgovarajuće boje ovisno o tome krši li mjeru socijalnog distanciranja ili ne. U donjem lijevom kutu nalazi se brojač koji prikazuje ukupan broj ljudi koji krše mjeru socijalnog distanciranja. Nekoliko kadrova videozapisa dani su na slikama 31., 32. i 33.



Slika 31. Prikaz kršenja mjera socijalnog distanciranja



Slika 32. Prikaz kršenja mjera socijalnog distanciranja



Slika 33. Prikaz kršenja mjera socijalnog distanciranja

4. KRITIČKA EVALUACIJA

Uzimajući u obzir sve prethodno spomenuto, može se zaključiti kako je programska aplikacija pouzdana. Navedenu tvrdnju potkrjepljuje činjenica kako su na učitanom videozapisu prepoznati i odgovarajućom bojom obilježeni svi pojedinci koji se pojavljuju u određenom vremenskom trenutku. Samim time, transformacija perspektive pomoću OpenCV-a pokazala se kao vrlo efikasna metoda mjerenja udaljenosti između ljudi. Unaprjeđenja programske aplikacije moguća su u nekoliko smjerova. Jedan od njih je upotreba aplikacije u realnom vremenu te korištenje termalne kamere kako bi se istovremeno saznale informacije o tjelesnim temperaturama pojedinaca. Takvo bi poboljšanje bilo vrlo korisno primjerice u zdravstvenim ustanovama gdje je nerijetko velika koncentracija ljudi, a samim time i virusa. Na taj bi se način spriječile potencijalne zaraze te bi se očuvalo zdravlje svakog pojedinca. Još jedno od poboljšanja bi bilo povećanje broja neurona, odnosno slojeva. Ovo bi se moglo riješiti i upotrebom novog modela dubokog učenja, odnosno YOLOv4, koji u odnosu na prethodne verzije ima bolje rezultate u otkrivanju objekata. Zbog većeg broja slojeva neuronske mreže, YOLOv4 pokazuje dobre performanse u pogledu točnosti i preciznosti. Također, povećanje količine i raznolikosti skupa podataka što uključuje više slika ljudi različite građe, rase i starosti moglo bi pridonijeti većoj preciznosti. Uz to, s učinkovitim GPU-om i CPU-om, izvršavanje modela neuronske mreže bit će efikasnije uz manje topline i buke u hardveru. Performanse GPU-a i CPU-a važni su parametri NVIDIA sustava za procjenu sposobnosti izvršavanja algoritama umjetne inteligencije.

5. ZAKLJUČAK

U radu je prikazan tijek razvoja programske aplikacije za detekciju kršenja mjera socijalnog distanciranja uz odgovarajuću teorijsku podlogu. Iako primijenjena na primjeru pandemije korona virusa koja je zahvatila čitav svijet, tema se može primijeniti u prevenciji širenja ostalih bolesti u zdravstvenom sustavu. Rješenje se temelji na primjeni računalnog vida koji uz pomoć dubokog učenja i konvolucijskih neuronskih mreža ima mogućnost prepoznavanja objekata. Kod je pisan u programskom jeziku Python uz pomoć odgovarajućih biblioteka i modula. U sklopu teorijske podloge dane su osnovne informacije o pojmovima poput umjetne inteligencije, strojnog učenja, umjetne neuronske mreže i računalnog vida koji se protežu kroz čitav rad. Osnovne značajke programske aplikacije jesu detekcija objekta, praćenje objekta te mjerenje udaljenosti. Detekcija objekata izvedena je uz pomoć YOLO algoritma za detekciju objekata iz kojeg je izdvojena samo klasa ljudi što znači da će se pratiti samo ljudski subjekti. Mjerenje udaljenosti omogućeno je pomoću OpenCV-a transformacijom pogleda u dvodimenzionalni prikaz te izračunom euklidske udaljenosti između objekata. Izrada programske aplikacije također je podijeljena u nekoliko etapa: postavljanje vrijednosti varijabli, kreiranje funkcije za detekciju ljudi te dohvaćanje graničnih okvira te predviđanja na temelju mjerenja udaljenosti detektiranih ljudi. Nakon postavljanja početnih vrijednosti poput minimalnog dopuštenog razmaka, kreira se funkcija za detekciju ljudi koja uključuje samo klasu ljudi te stvara granične okvire zajedno s težištima. Posljednji korak je primjena funkcije na videozapisu na kojem jesu vidljivi granični okviri oko ljudi u odgovarajućoj boji ovisno o tome krši li se mjera socijalnog distanciranja ili ne. Također, na zaslonu je vidljiv i ukupan broj kršenja mjere.

LITERATURA

[1] https://www.hzjz.hr/sluzba-epidemiologija-zarazne-bolesti/pitanja-i-odgovori-o-bolesti-uzrokovanoj-novim-koronavirusom/ Pristupljeno: 8.8.2022.

[2] https://www.koronavirus.hr/ogranicavanje-okupljanja-i-druge-nuzne-epidemioloske-mjere-i-preporuke/961 Pristupljeno: 9.8.2022.

[3] https://hub.jhu.edu/2020/03/13/what-is-social-distancing/ Pristupljeno: 9.8.2022.

[4] https://www.enciklopedija.hr/natuknica.aspx?ID=63150 Pristupljeno: 10.8.2022.

[5] https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp#toc-applications-of-artificial-intelligence Pristupljeno: 10.8.2022.

[6] B. Dalbelo Bašić, J. Šnajder: Podloge za predavanje iz kolegija „Strojno učenje“, Fakultet elektrotehnike i računarstva, Zagreb, 2015.

[7] T. Stipančić: Podloge za predavanje iz kolegija „Umjetna inteligencija“, Fakultet strojarstva i brodogradnje, Zagreb, 2020.

[8] https://medium.com/@devnag/machine-learning-representation-evaluation-optimization-fc7b26b38fdb Pristupljeno: 11.8.2022.

[9] https://www.javatpoint.com/types-of-machine-learning Pristupljeno: 11.8.2022.

[10] https://www.zendesk.com/blog/machine-learning-and-deep-learning/ Pristupljeno: 11.8.2022

[11] https://www.net-cloud.com/blog/machine-learning-and-deep-learning-101/ Pristupljeno: 11.8.2022

[12] https://www.javatpoint.com/artificial-neural-network Pristupljeno 15.8.2022.

[13] N.Bolf, Osvježimo znanje, Kem. Ind 68 (5-6), 219-220; 2019.

[14] https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html Pristupljeno: 15.8.2022.

[15] https://www.ibm.com/topics/computer-vision Pristupljeno: 15.8.2022.

[16] https://docs.python.org/3/faq/general.html#general-information Pristupljeno: 17.8.2022.

[17] https://steelkiwi.com/blog/python-for-ai-and-machine-learning/ Pristupljeno: 17.8.2022.

[18] https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html Pristupljeno 17.8.2022.

[19] https://docs.python.org/3/library/ Pristupljeno 17.8.2022.

[20] https://opencv.org/about/ Pristupljeno 17.8.2022.

[21] https://developpaper.com/basic-concept-and-usage-of-python-imutils-package/ Pristupljeno 23.8.2022.

[22] https://www.educba.com/what-is-numpy/ Pristupljeno 23.8.2022.

[23] https://scipy.org/ Pristupljeno 23.8.2022.

[24] https://viso.ai/deep-learning/yolov3-overview/#:~:text=YOLOv3%20(You%20Only%20Look%20Once%2C%20Version%203)%20is%20a,network%20to%20detect%20an%20object. Pristupljeno 23.8.2022.

[25] https://zhanghanduo.github.io/post/yolo1/ Pristupljeno 23.8.2022.

[26] https://ieeexplore.ieee.org/document/9243478 Pristupljeno 25.8.2022.

[27] https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/ Pristupljeno 31.8.2022.

[28] https://towardsdatascience.com/a-simple-guide-to-command-line-arguments-with-argparse-6824c30ab1c3 Pristupljeno 5.9.2022

[29] https://medium.com/analytics-vidhya/object-detection-using-yolov3-d48100de2ebb Pristupljeno 5.9.2022.

[30] https://www.intel.com/content/www/us/en/products/docs/processors/cpu-vs-gpu.html Pristupljeno 5.9.2022.

[31] https://developer.nvidia.com/cuda-zone Pristupljeno 5.9.2022.

PRILOG

I. Python kod

1. Postavljanje vrijednosti varijabli i konfiguracije

```
MODEL_PATH = "yolo-coco"

MIN_CONF = 0.3
NMS_THRESH = 0.3

USE_GPU = False

MIN_DISTANCE = 50
```

2. Kreiranje funkcije za detekciju ljudi

```
from .social_distancing_config import NMS_THRESH
from .social_distancing_config import MIN_CONF
import numpy as np
import cv2

def detect_people(frame, net, ln, personIdx=0):
    (H, W) = frame.shape[:2]
    results = []

    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(ln)

    boxes = []
    centroids = []
    confidences = []

    for output in layerOutputs:
        for detection in output:
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            if classID == personIdx and confidence > MIN_CONF:
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))
```

```
                boxes.append([x, y, int(width), int(height)])
                centroids.append((centerX, centerY))
                confidences.append(float(confidence))

    idxs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONF, NMS_THRESH)

    if len(idxs) > 0:
        for i in idxs.flatten():
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])

            r = (confidences[i], (x, y, x + w, y + h), centroids[i])
            results.append(r)

    return results
```

3. Dohvaćanje graničnih okvira te proces predviđanja mjerenjem udaljenosti detektiranih ljudi

```
import social_distancing_config as config
import detect_people
from scipy.spatial import distance as dist
import numpy as np
import argparse
import imutils
import cv2
import os

ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", type=str, default="",
    help="path to (optional) input video file")
ap.add_argument("-o", "--output", type=str, default="",
    help="path to (optional) output video file")
ap.add_argument("-d", "--display", type=int, default=1,
    help="whether or not output frame should be displayed")
args = vars(ap.parse_args())

labelsPath = os.path.sep.join([config.MODEL_PATH, "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

weightsPath = os.path.sep.join([config.MODEL_PATH, "yolov3.weights"])
configPath = os.path.sep.join([config.MODEL_PATH, "yolov3.cfg"])

print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

if config.USE_GPU:
    print("[INFO] setting preferable backend and target to CUDA...")
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
```

```
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

print("[INFO] accessing video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)
writer = None

while True:
    (grabbed, frame) = vs.read()

    if not grabbed:
        break

    frame = imutils.resize(frame, width=700)
    results = detect_people(frame, net, ln,
        personIdx=LABELS.index("person"))

    violate = set()

    if len(results) >= 2:
        centroids = np.array([r[2] for r in results])
        D = dist.cdist(centroids, centroids, metric="euclidean")

        for i in range(0, D.shape[0]):
            for j in range(i + 1, D.shape[1]):
                if D[i, j] < config.MIN_DISTANCE:
                    violate.add(i)
                    violate.add(j)

    for (i, (prob, bbox, centroid)) in enumerate(results):
        (startX, startY, endX, endY) = bbox
        (cX, cY) = centroid
        color = (0, 255, 0)

        if i in violate:
            color = (0, 0, 255)

        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
        cv2.circle(frame, (cX, cY), 5, color, 1)

    text = "Social Distancing Violations: {}".format(len(violate))
    cv2.putText(frame, text, (10, frame.shape[0] - 25),
        cv2.FONT_HERSHEY_SIMPLEX, 0.85, (0, 0, 255), 3)

    if args["display"] > 0:
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF
```

```
        if key == ord("q"):
            break

    if args["output"] != "" and writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 25,
            (frame.shape[1], frame.shape[0]), True)

    if writer is not None:
        writer.write(frame)
```