SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Nikolai Ivan Horvat

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:
Doc. dr. sc. Tomislav Stipančić

Student:
Nikolai Ivan Horvat

Zagreb, 2022.

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

Zahvaljujem se mentoru doc. dr. sc. Tomislavu Stipančiću na iskazanom povjerenju, vodstvu i korisnim savjetima tijekom izrade rada, te prijateljima i kolegama na korisnim savjetima.

Na kraju, posebno se zahvaljujem svojoj obitelji i djevojci na pruženoj ljubavi, povjerenju i podršci tijekom cijelog studija.

Nikolai Ivan Horvat

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-14/22-6/1	
Ur. broj: 15-1703-22-	

DIPLOMSKI ZADATAK

Student: **NIKOLAI IVAN HORVAT** Mat. br.: 0035213869

Naslov rada na hrvatskom jeziku: **Web aplikacija za nadgledanje prostora temeljena na umjetnoj inteligenciji**

Naslov rada na engleskom jeziku: **Web application for spatial monitoring based on artificial intelligence**

Opis zadatka:

Suvremene računalne tehnike temeljene na umjetnoj inteligenciji omogućuju automatizaciju i sinkronizaciju različitih procesa. U radu je potrebno izraditi cjelovito rješenje u vidu web stranice za video nadzor prostora. Web stranica treba predstavljati sučelje između korisnika i sustava preko kojeg će korisnik moći pregledavati snimljeni materija i u kojem će biti prikazan pogled kamere u realnom vremenu. Za web kameru koja nadgleda prostor potrebno je ostvariti funkciju automatskog uključenja funkcije snimanja u dva slučaja, uključujući:

- otkrivanje osobe u prostoru temeljem detektiranog lica,
- otkrivanje bilo kakvih kretnji u prostoru.

Modele za detektiranje lica ili pokreta potrebno je temeljiti na poznatim modelima strojnog učenja koji se mogu koristiti za tu namjenu (npr. Viola Jones ili optical flow algoritam).
Razvijeno rješenje potrebno je eksperimentalno verificirati na opremi dostupnoj u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava.
U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan: 5. svibnja 2022.

Rok predaje rada: 7. srpnja 2022.

Predviđeni datum obrane: 18. srpnja do 22. srpnja 2022.

Zadatak zadao: doc. dr. sc. Tomislav Stipančić

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
A_{i}	px^2	Površina integralne slike
dx	px	Pomak u smjeru *x*-osi
dy	px	Pomak u smjeru *y*-osi
dt	s	Vrijeme između dva uzastopna kadra u videozapisu
S	px^2	Površina dijela integralne slike

SAŽETAK

Računalni vid interdisciplinarno je područje koje obuhvaća mnoge grane poput inženjerstva, računarstva, matematike, ali i biologije, fizike i psihologije. Ono omogućuje računalima da vide, tj. da procesiraju i prepoznaju objekte na slici kao što to mogu ljudi. S obzirom da se dnevno objavljuje i više od četiri milijarde slika, javlja se potreba za razvojem novih i bržih algoritama koji omogućuju napredak računalnom vidu. Uzimajući u obzir današnje trendove razvoja i dizajna, u ovom je radu razvijena web aplikacija za nadzor prostora. Ona predstavlja sučelje između korisnika i sustava u kojem je prikazan pogled kamere u stvarnom vremenu i preko kojeg korisnik može pregledati videozapise u arhivi. Kao okidač spremanja videozapisa na raspolaganju su dva poznata modela strojnog učenja koja se mogu koristiti za praćenje – detekcija lica temeljena na Haar značajkama (algoritam Viola-Jones) i detekcija pokreta optical flow. Na kraju rada provedena je evaluacija aplikacije koja uključuje ljudske subjekte.

Ključne riječi: računalni vid, strojno učenje, umjetna inteligencija, web aplikacija, JavaScript, Python, OpenCV, Viola-Jones, optical flow

SUMMARY

Computer vision is an interdisciplinary field that includes many areas such as engineering, computing, mathematics, but also biology, physics and psychology. It gives computer the ability to see – to process and recognize objects in an image as humans can. Given that more than four billion images are published daily, there is a need for the development of new and faster algorithms that enable computer vision to progress. Considering today's development and design trends, a web application for space monitoring was developed in this paper. It represents an interface between the user and the system in which the camera view is displayed in real time and through which the user can browse the videos in the archive. The trigger to save the videos are two of the famous machine learning models that can be used for tracking - face detection based on Haar features (Viola-Jones algorithm) and optical flow motion detection. At the end of the paper, an evaluation of the web application involving human subjects was carried out.

Keywords: computer vision, machine learning, artificial intelligence, web application, JavaScript, Python, OpenCV, Viola-Jones, optical flow

1. UVOD

Računalni vid područje je umjetne inteligencije (eng. *artificial intelligence* - AI) koje omogućuje računalnim sustavima izlučivanje i interpretaciju informacija koju pruži digitalna slika, videozapis ili neki drugi vizualni podražaj – i djelovanje u skladu s dobivenim informacijama. Dok AI omogućuje računalu da misli, računalni vid omogućuje računalu da vidi, doživljava i razumije. To je interdisciplinarno područje koje ne obuhvaća samo umjetnu inteligenciju, već i grafiku, optiku, digitalnu fotografiju, robotiku, neuroznanost te interakciju čovjeka i računala.

Ljudski se vid ne razlikuje mnogo od računalnog, no ljudi imaju blagu prednost – cjeloživotno učenje konteksta što oči vide poput razlikovanja predmeta, životinja i samih ljudi, interpretacije udaljenosti, mogućnost prepoznavanja kretnje, greške u slici i sl. Računalni vid uvježbava računalo za izvršenje istih funkcija, no to se mora izvršiti mnogo brže pomoću kamera, algoritama i različitih podataka nego što se odvija kod ljudi. [1]

Razvijanjem projekta PLEA, došlo je do potrebe za spremanjem videozapisa interakcije robotskog sučelja i ljudi od strane robota – tj. potrebno je snimiti ono što robot „vidi“.

PLEA je virtualno biće koje živi u MetaVerse kibernetičkom prostoru i može posjećivati različite okoline u realnom svijetu kroz interakcijska sučelja. Interakcijska sučelja mogu biti AR i VR naočale, svjetlo projicirano projektorom, ekran mobitela, računalo, pametni TV i sl. Dvosmjernu neverbalnu komunikaciju s posjetiteljima PLEA temelji na percipiranim emocijama i multimodalnom pristupu gdje analizira izražaje lica, pokrete tijela i glas osobe. PLEA komunicira prikazujući vlastite emocije na licu.

S obzirom da neprekidno spremanje ili spremanje videozapisa u intervalima zauzima previše memorije na tvrdom disku, potrebno je razviti rješenje kod kojeg će spremanje videozapisa krenuti pristupom ljudskog lica u vidno polje kamere koju PLEA koristi kako bi „vidjela“. Drugi prijedlog rješenja je da spremanje videozapisa krene detekcijom pokreta u vidnom polju kamere.

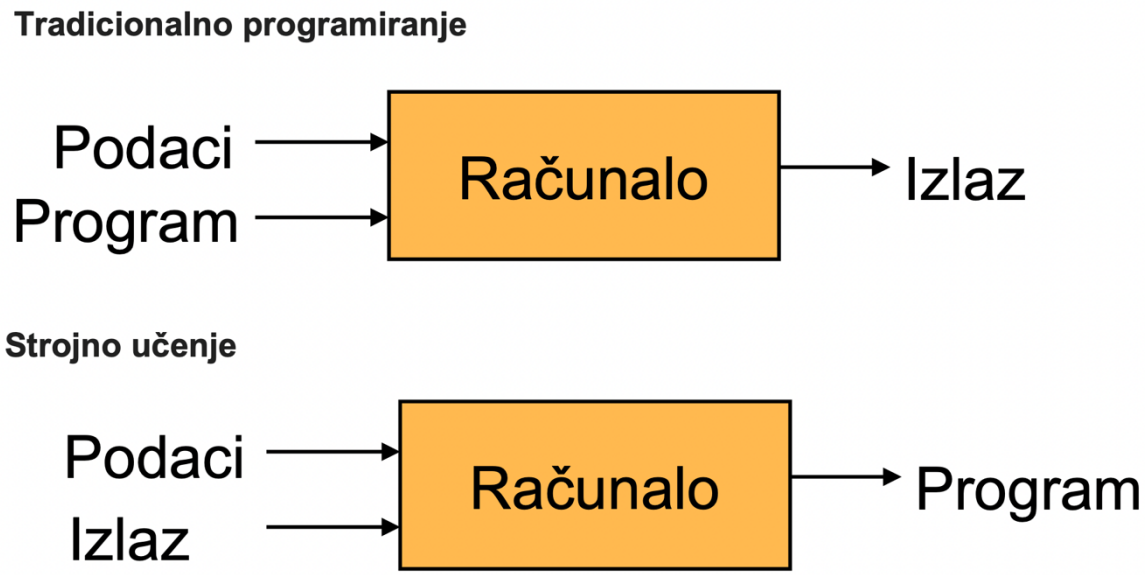
Potrebna oprema za razvoj takvog sustava je računalo i digitalna kamera (u ovom je radu korištena ugrađena web-kamera prijenosnog računala).

2. PREGLED TEHNOLOGIJE

2.1. Strojno učenje

Strojno učenje grana je umjetne inteligencije (eng. *artificial intelligence* – AI) i računalne znanosti koja se usredotočuje na korištenje podataka i algoritama za oponašanje načina na koji uče ljudi, postupno poboljšavajući njegovu točnost. Može se koristiti za detekciju i prepoznavanje lica, prepoznavanje objekata, itd. Također, važna je komponenta rastućeg područja podatkovne znanosti. Korištenjem statističkih metoda, algoritmi se osposobljavaju za izradu klasifikacija ili predviđanja, otkrivajući ključne uvide unutar projekata rudarenja podataka. [2]

Razlika između „tradicionalnog“ programiranja i strojnog učenja (vidljivo u **Slika 1**) je da se kod strojnog učenja računalu daju ulazni podaci i traženi izlazni podaci te nam ono izbaci skup pravila (program) koji daje traženo rješenje, dok kod „tradicionalnog“ programiranja – programer sam definira skup pravila (program) za traženi izlaz, što nije uvijek lako izvedivo, ponekad i nemoguće za određene probleme.



Slika 1: Razlika tradicionalnog programiranja (gore) i strojnog učenja (dolje) [3]

Strojno učenje dijeli se u četiri glavne potkategorije [3]:

- Nadgledano (eng. *supervised learning*) učenje
- Nenadgledano (eng. *unsupervised learning*) učenje
- Polunadgledano (eng. *semi-supervised learning*) učenje
- Ojačano (eng. *reinforcement learning*) učenje

Nadgledano učenje definirano je korištenjem skupova podataka s oznakama (eng. *label*) za učenje algoritama kako bi kasnije računalo samo moglo klasificirati podatke. Ono pomaže u rješavanju raznih problema iz stvarnog svijeta, kao što je razvrstavanje neželjene pošte u zasebnu mapu iz pristigle pošte. Neke metode koje se koriste u nadgledanom učenju uključuju neuronske mreže, linearnu regresiju, logističku regresiju i sl.

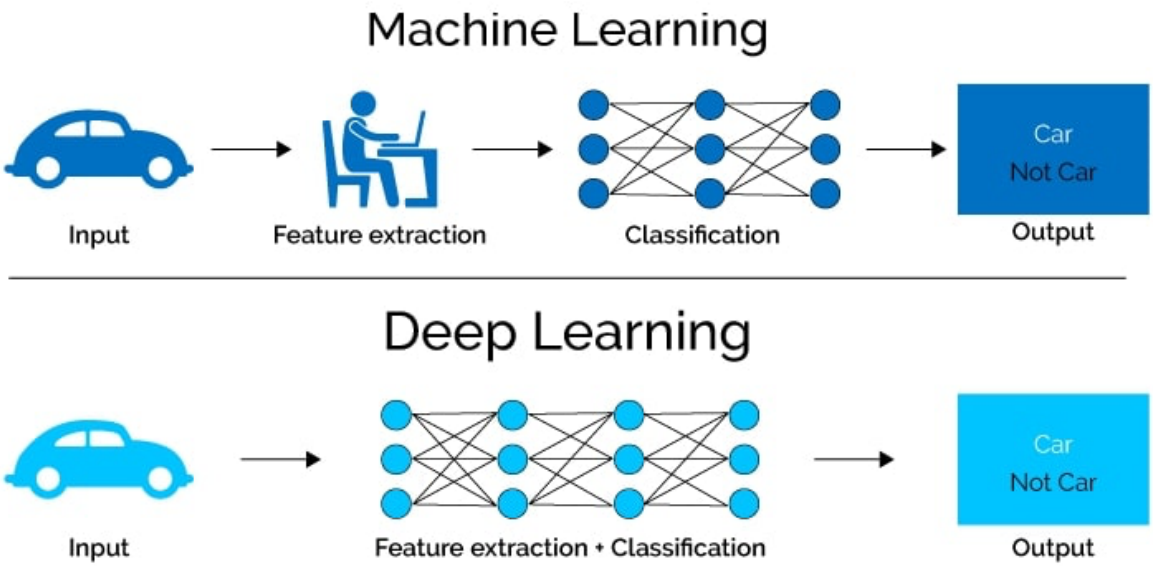
Nenadgledano učenje koristi algoritme strojnog učenja za analizu i grupiranje skupova podataka bez oznake (eng. *unlabeled*). Navedeni algoritmi otkrivaju skrivene obrasce bez potrebe ljudske intervencije. Njegova sposobnost otkrivanja sličnosti i razlika u informacijama, čini ga idealnim rješenjem za istraživačku analizu podataka, strategije unakrsne prodaje, segmentaciju kupaca, prepoznavanje slika i uzoraka. Algoritmi koji se koriste u nenadziranom učenju uključuju neuronske mreže, algoritam k-srednjih vrijednosti, metode vjerojatnog grupiranja, itd.

Polunadgledano učenje nudi kompromis između nadgledanog i nenadgledanog učenja. Tijekom učenja, koristi se manji skup podataka s oznakama (labelima) kako bi vodio klasifikaciju i izdvajanje značajki iz većeg, neoznačenog skupa podataka. Polunadgledano učenje može riješiti problem nedostatka označenih podataka za treniranje algoritma nadgledanog učenja.

Ojačano učenje bihevioralni je model strojnog učenja koji je sličan nadgledanom učenju, ali algoritam nije obučen pomoću uzoraka podataka. Ovaj model uči metodom pokušaja i pogrešaka, tj. uči iz stečenog iskustva. [2]

2.2. Duboko učenje

Prije svega, tradicionalni algoritmi strojnog učenja imaju prilično jednostavnu strukturu, poput linearne regresije ili stabla odlučivanja, dok se duboko učenje – grana strojnog učenja – temelji na umjetnoj neuronskoj mreži. Ova je višeslojna neuronska mreža poput ljudskog mozga, složena i isprepletena. Ti isti algoritmi zahtijevaju mnogo manje ljudske intervencije – značajke se automatski izdvajaju, a algoritam uči iz vlastitih pogrešaka (vidi **Slika 2**). Također, duboko učenje zahtijeva mnogo više podataka od tradicionalnog algoritma za strojno učenje kako bi ispravno funkcionirao. Strojno učenje radi s tisuću točaka podataka, a duboko učenje često s milijunima podataka. [4]



Slika 2: Razlika strojnog i dubokog učenja [4]

Ono pokušava modelirati apstrakciju visoke razine u podacima na temelju skupa algoritama. U tehnikama dubokog učenja postoji izravno učenje iz podataka za sve aspekte modela. Počinje sa značajkama najniže razine, koje predstavljaju prikladan prikaz podataka, a zatim daje apstrakcije više razine za svaki od specifičnih problema u kojima se primjenjuje. Zbog složene višeslojne strukture, sustavu dubokog učenja potreban je veliki skup podataka kako bi se uklonile fluktuacije i napravila visokokvalitetna tumačenja.

Razvoj modela dubokog učenja napredovao je s povećanjem softverske i hardverske mogućnosti. Povećanjem veličine podataka, odnosno razvojem u području velikih podataka (eng. *big data*), konvencionalne tehnike strojnog učenja pokazale su svoje ograničenje, dok su tehnike dubokog učenja dale bolje rezultate. Duboko učenje uvedeno je širom svijeta kao revolucionarna tehnologija zbog toga što je transformiralo tehnike konvencionalnog strojnog učenja (koje rade na starim i tradicionalnim algoritmima) u tehnike čiji procesi i rezultati nalikuju sposobnostima ljudskog mozga. Umjesto rada na algoritmima specifičnim za potreban zadatak, temelji se na učenju reprezentacija podataka. Ovo učenje može biti nadgledano, nenadgledano, polunadgledano i ojačano (što je obrađeno u prethodnom poglavlju). Sam model dubokog učenja sastavljen je od više slojeva nelinearnih procesnih jedinica koje obavljaju transformaciju izdvojenih značajki te svaki sloj kao ulaz uzima izlaz odgovarajućeg prethodnog sloja. Višestruki slojevi, koji omogućavaju apstrakciju visoke razine, čine hijerarhiju koncepata[1]. Postoje modeli dubokog učenja temeljeni na umjetnim neuronskim mrežama koje su slojevito organizirane u duboke generativne modele. [4]

2.3. Neuronske mreže

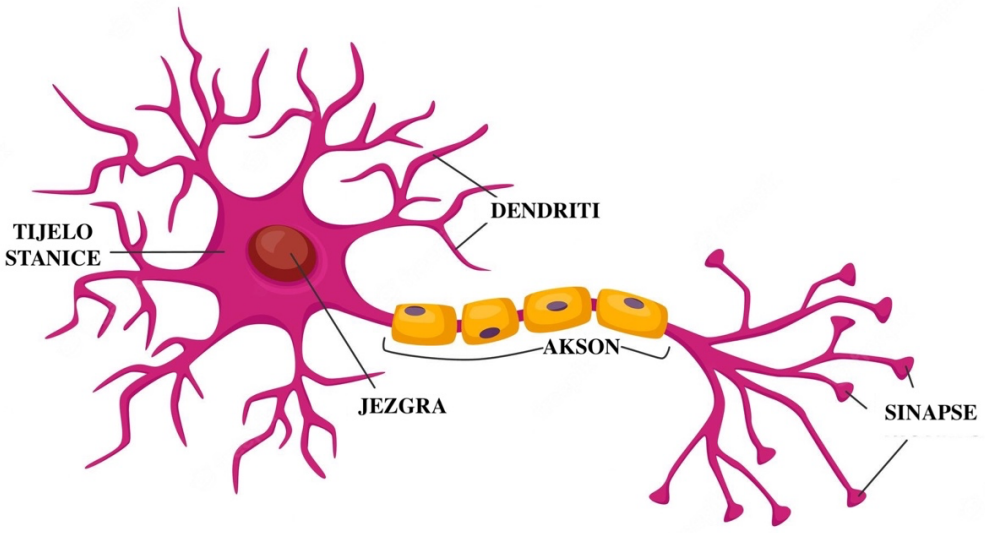
2.3.1. Neuron

Ljudski mozak ima sposobnost učiti nove stvari, prilagođavati se novom i promjenjivom okruženju, tj. ima nevjerojatnu sposobnost analiziranja djelomičnih i nejasnih informacija kao i donošenja vlastitih zaključaka o njima. Na primjer, ljudi su u mogućnosti čitati tuđi rukopis, iako se on može potpuno razlikovati od rukopisa druge osobe. Isto tako, dijete može prepoznati da je oblik lopte i naranče kugla, čak i novorođenče ima sposobnost prepoznavanja majke po dodiru, glasu i mirisu.

Mozak najprimitivnije životinje ima više sposobnosti od najnaprednijeg računala. Njegova funkcija nije samo kontroliranje fizičkih dijelova tijela, već osigurava i složenije aktivnosti poput razmišljanja, vizualizacije, sanjanja ili učenja, odnosno aktivnosti koje se ne mogu opisati fizički. [5]

[1] definira slijed preslikavanja iz skupa koncepata niske razine u više, općenitije koncepte

Da bi se pravilno analizirale umjetne neuronske mreže, najprije je potrebno upoznati biološke neurone – osnovne gradivne dijelove ljudskog mozga – odgovorne za primanje podražaja iz vanjskog svijeta, slanje motoričkih signala mišićima te transformaciju i prijenos električnih signala. Četiri osnovna dijela koja čine neuron su tijelo stanice, skup dendrita, akson i niz sinapsi (**Slika 3**). [6]

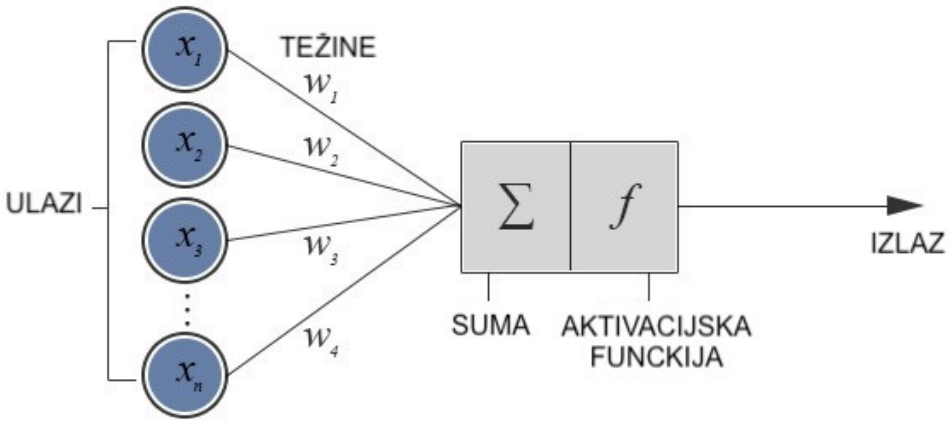


Slika 3: Građa neurona [6]

- Dendriti su mjesto na kojem neuron prima ulaz od drugih stanica i granaju se prema vrhovima.
- Kao izlazna struktura neurona definiran je akson kroz koji se šalje električni signal, tj. akcijski potencijal. Akcijski potencijal kratak je električni signal generiran u aksonu koji označuje neuron „aktivnim“, nakon čega putuje duž aksona i uzrokuje oslobađanje neurotransmitera u sinapsu. Akcijski potencijal i posljedično oslobađanje neurotransmitera omogućuju neuronu komunikaciju s drugim neuronima.
- U tijelu stanice nalaze se jezgra, DNA neurona, kao i sintetizirani proteini namijenjeni transportu kroz akson i dendrite.
- Sinapsa je veza između dendrita dvaju neurona, odnosno neurona i mišićnih stanica. [7]

Međusobna povezanost neurona čini neuronsku mrežu. U ljudskom mozgu postoji oko 10^{11} neurona i desetak tisuća međusobnih veza. Umjetna neuronska mreža (eng. *artificial neural network* - ANN) imitacija je prirodne neuronske mreže u kojoj su umjetni neuroni povezani na sličan način kao u mozgu – gdje neuron prima električne signale od drugih neurona putem dendrita. Kada jačina signala prijeđe određeni prag, neuron – koji je primio signal – se „budi“

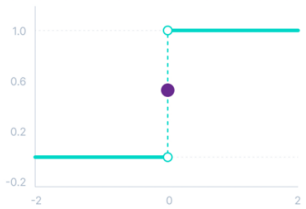
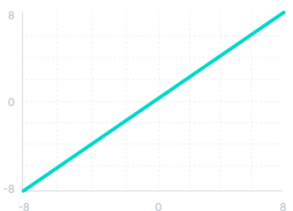
i generira vlastiti signal koji se prenosi na sljedeći neuron preko aksona i sinapse te ih pobuđuje i nastavlja proces. Umjetni neuron pokušava replicirati strukturu i ponašanje prirodnog neurona – sastoji se od ulaza i težina (poput dendrita) te jednog izlaza (poput sinapse). [5]



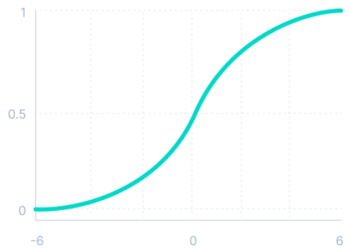
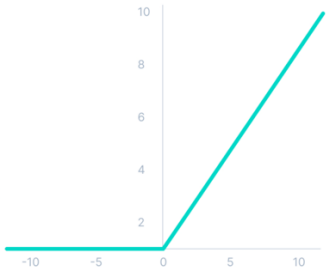
Slika 4: Model umjetnog neurona [5]

Vrijednosti x_1, x_2, ..., x_n ulazi su u neuron. Pristranost (eng. *bias*) se također dodaje neuronu zajedno s ulazima – obično se vrijednost biasa inicijalizira na 1. Vrijednosti w_1, w_2, ..., w_n težinske su vrijednosti (veza sa signalom), a umnožak težina i ulaza daje snagu signala. Ako je zbroj otežanih signala[2] veći od praga osjetljivosti, aktivacijska funkcija generira izlazni signal neurona koji postaje ulaz u sljedeći. Primjeri nekih aktivacijskih funkcija dani su u nastavku u **Tablica 1** [5]:

Tablica 1: Primjeri aktivacijskih funkcija

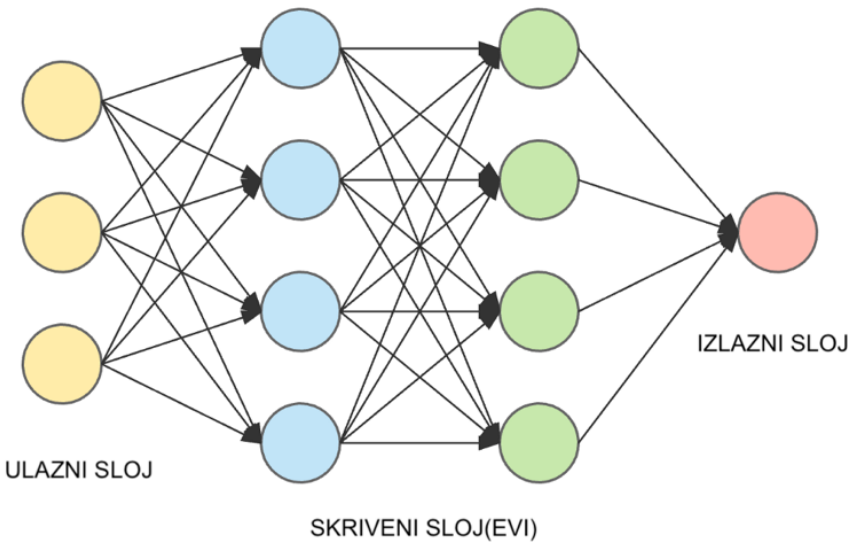
Aktivacijska funkcija	*Jednadžba*	*Grafički prikaz*
Step (odskočna, Heaviside)	$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	
Linearna	$f(x) = x$	

[2] suma svih umnožaka ulaznih signala i težina

Sigmoidalna	$f(x) = \frac{1}{1+e^{-x}}$	
ReLU	$f(x) = \max(0, x)$	

Arhitektura umjetne neuronske mreže sastoji se od (**Slika 5**):

- ulaznog sloja koji prima ulazne vrijednosti
- skrivenog sloja koji čini skup neurona između ulaznog i izlaznog sloja (može biti jedan sloj ili više slojeva)
- izlaznog sloja, koji obično čini samo jedan neuron, čiji se izlaz kreće u intervalu od nula do jedan



Slika 5: Arhitektura umjetne neuronske mreže [5]

Sposobnost obrade pohranjena je u snagama veze između jedinica, odnosno težinama. Ulazna snaga ovisi o vrijednosti težine. Ona može biti pozitivna, negativna ili nula. Negativna težina označava smanjeni ili inhibirani signal, a nulta težina označava nepostojeću vezu između dva neurona. Te se težine prilagođavaju kako bi se postigla potrebna snaga, a postoje i algoritmi za podešavanje težina kako bi se dobio traženi izlaz. Proces prilagođavanja težina naziva se učenje ili treniranje neuronske mreže. [5]

2.3.2. Postupak učenja umjetne neuronske mreže

Najjednostavniji je oblik arhitekture umjetne neuronske mreže perceptron koji se sastoji od jednog neurona s (barem) dva ulaza i jednim izlazom, a aktivacijska funkcija je step funkcija ili linearna funkcija. Perceptroni se koriste za razvrstavanje podataka u dvije odvojene klase, no za složenije primjene koriste se višeslojni perceptroni (eng. *multi-layer perceptron*), koji sadrže jedan ulazni sloj, jedan izlazni sloj i jedan ili više skrivenih slojeva kao što je prikazano u **Slika 5**. Za razliku od perceptrona kojemu je izlaz funkcija ulaza, drugim složenijim modelima neuronskih mreža potrebno je učenje, odnosno treniranje. Učenjem se smatra iterativni postupak optimiranja težinskih vrijednosti s ciljem smanjenja pogreške između vrijednosti izračunate modelom i stvarne vrijednosti te se odvija prema algoritmu unatražne propagacije (eng. *backpropagation*).

Prema backpropagation algoritmu, izlazi skrivenih slojeva propagiraju se u izlazni sloj gdje se računa izlazna vrijednost koja se uspoređuje s traženom (stvarnom) vrijednosti. Na temelju navedene razlike, najčešće računanom metodom najmanjih kvadrata, pogreška se propagira nazad sve do ulaznog sloja. Prolaskom pogreške kroz slojeve, traže se težinske vrijednosti koje najviše utječu na pogrešku u rezultatu te se korigiraju (SGD metoda – stohastic gradient descent). Ciklus propagacije od ulaza do izlaza i nazad naziva se epoha. Neuronska mreža prolazi kroz *N* epoha sve dok pogreška ne bude unutar određene tolerancije. Tada se može reći da je neuronska mreža naučena, odnosno istrenirana.

Nakon što je mreža istrenirana, potrebno je provjeriti ispravan rad novim skupom podataka čime se nastoji spriječiti „pretreniranje“ neuronske mreže – pojavu pri kojoj mreža koristeći dani skup podataka za učenje daje izuzetno točne rezultate – no za podatke iz nekog drugog, sličnog skupa daje pogrešne rezultate. [5]

2.3.3. Podjela neuronskih mreža

Zbog postojanja vrlo velikog broja modela umjetnih neuronskih mreža, one se mogu podijeliti u više kategorija. Dvije osnovne kategorije su algoritam učenja i struktura. [5]

2.3.3.1. Podjela prema algoritmu učenja

Neuronske se mreže prema algoritmu učenja mogu podijeliti na nadgledano učenje, nenadgledano, polunadgledano i ojačano učenje što je već obrađeno u poglavlju 2.1. [3]

2.3.3.2. Podjela prema strukturi

Prema strukturi neuronske mreže, mogu se podijeliti u tri podvrste: unaprijedne (eng. *feed-forward*), povratne (eng. *recurrent*) i konvolucijske (eng. *convolutional*) neuronske mreže.

Kod unaprijedne neuronske mreže neuroni su grupirani u slojeve gdje signali teku od ulaznog sloja do izlaznog sloja jednosmjernim vezama, pri čemu su neuroni jednog sloja povezani s neuronima drugog sloja, ali nisu povezani unutar istog sloja. Najbolji je primjer za unaprijednu neuronsku mrežu višeslojni perceptron.

Za razliku od unaprijedne neuronske mreže, kod povratne je moguće da se izlazi neurona vraćaju istim neuronima ili neuronima u prethodnim slojevima, odnosno signali mogu teći u oba smjera kao što je opisano u poglavlju 2.3.2. [5]

Konvolucijske neuronske mreže bit će detaljnije obrađene kasnije u radu.

2.4. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (eng. *Convolutional Neural Network* – CNN) koriste se za duboko učenje i analogne su tradicionalnim umjetnim neuronskim mrežama po tome što se sastoje od neurona koji vrše samooptimizaciju učenjem. Jedina značajna razlika između konvolucijske i tradicionalne neuronske mreže u tome je što se CNN prvenstveno koriste za pronalaženje i prepoznavanje uzoraka unutar slika. [8]

Cilj je konvolucijskih neuronskih mreža omogućiti strojevima da vide svijet kao ljudi, percipiranje na sličan način, pa čak i da koriste znanje za mnoštvo zadataka kao što su

prepoznavanje slika i videa, analiza i klasifikacija slika, obrada prirodnog jezika, itd. Napredak u području računalnog vida dubokim učenjem konstruiran je i usavršen s vremenom, primarno preko jednog određenog algoritma – konvolucijske neuronske mreže. Njena glavna karakteristika je učenje direktno iz podataka, eliminirajući potrebu za ručnim izdvajanjem značajka.

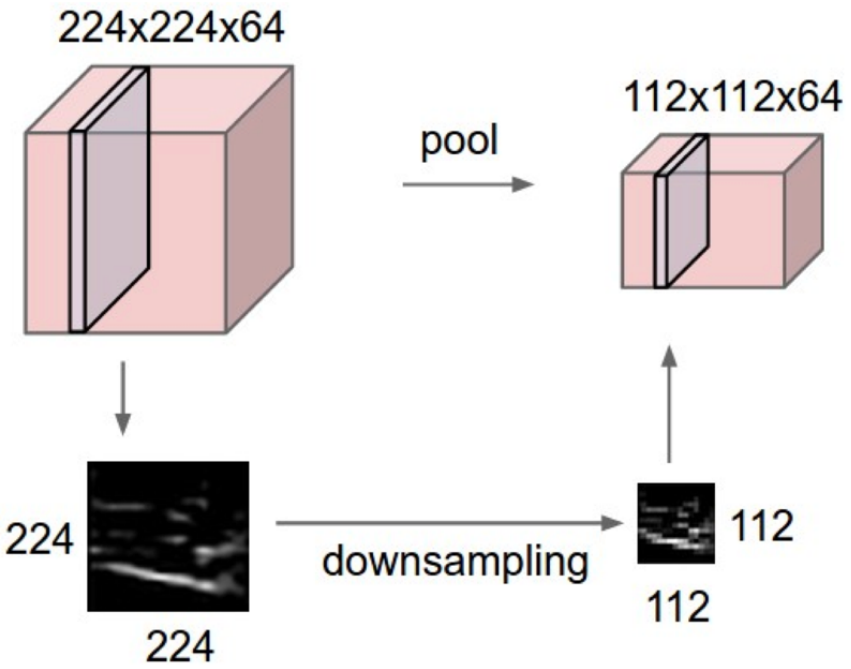
Konvolucijska neuronska mreža može imati desetke ili stotine slojeva – od kojih svaki uči otkrivati različite značajke slike. Filtri se primjenjuju na svaku sliku za treniranje u različitim rezolucijama, a izlaz svake konvoluirane slike koristi se kao ulaz za sljedeći sloj. Filtri mogu započeti kao vrlo jednostavne značajke, kao što su svjetlina i rubovi, no mogu i povećati složenost do značajki koje jedinstveno definiraju objekt. Kao i druge neuronske mreže, CNN se sastoji od ulaznog sloja (eng. *input layer*), izlaznog sloja (eng. *output layer*) i mnogo skrivenih slojeva (eng. *hidden layer*) što je već viđeno u **Slika 5.** [9]

Navedeni slojevi izvode operacije koje mijenjaju podatke s namjerom učenja značajki specifičnih za dane podatke. Četiri najčešća sloja su: konvolucija (eng. *convolution*), aktivacija ili ReLU, sažimanje (eng. *pooling*) i potpuno povezani sloj (eng. *fully-connected layer*).

- Konvolucija stavlja ulazne slike kroz skup konvolucijskih filtara, od kojih svaki aktivira određene značajke slika – jednostavnije rečeno, pridodaju određenu vrijednost pikselima koji im odgovaraju. Na taj način neuronska mreža konvertira podatke filtra s ulaznom slikom kako bi se stvorila tzv. mapa značajki (eng. *feature map*).

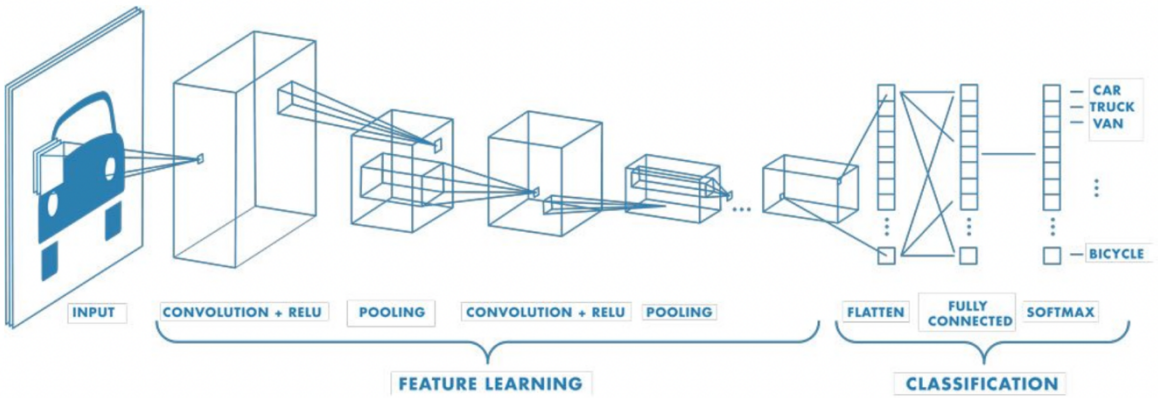
- Rektificirana linearna jedinica (eng. *Rectified linear unit* – ReLU) omogućuje brže i učinkovitije učenje preslikavanjem negativnih vrijednosti na nulu i zadržavanjem pozitivnih vrijednosti. To se ponekad naziva i aktivacijom zbog toga što se samo aktivirane značajke prenose dalje u sljedeći sloj.

- Sažimanje pojednostavljuje izlaz izvođenjem nelinearnog downsamplinga, smanjujući broj parametara koje mreža treba naučiti – odnosno smanjuje razlučivost mape značajki iz konvolucijskih slojeva (**Slika 6**).



Slika 6: Primjer sažimanja (pooling) [3]

Te se operacije (**Slika 7**) ponavljaju u desecima ili stotinama slojeva, pri čemu svaki sloj uči identificirati različite značajke.



Slika 7: Operacije s namjerom učenja značajki [9]

Predzadnji sloj – potpuno povezan sloj (eng. *fully connected layer*) – dolazi nakon skrivenog sloja konvolucijske neuronske mreže, vektora koji sadrži vjerojatnosti za svaku klasu bilo koje klasificirane slike. [9]

2.5. Računalni vid

Računalni vid područje je računalne znanosti koje se usredotočuje na repliciranje dijelova složenosti sustava ljudskog vida i omogućavanje računalima identifikaciju i obradu objekata u slikama i videozapisima na isti način kao i ljudi. Donedavno je računalni vid radio samo u ograničenom kapacitetu. Zahvaljujući nedavnom napretku u umjetnoj inteligenciji i inovacijama u dubokom učenju i neuronskim mrežama, ovo je polje posljednjih godina napravilo velike iskorake i uspjelo je nadmašiti ljude u nekim zadacima otkrivanja i označavanja objekata. Jedan od pokretačkih čimbenika razvoja računalnog vida količina je podataka koje danas ljudi generiraju (dnevno se objavljuje više od četiri milijarde slika).

Repliciranje ljudskog vida pomoću digitalnih slika kroz tri glavne komponente obrade vrši se uzastopnim redoslijedom [10]:

- Pribavljanje slike (eng. *Image acquisition*)
- Obrada slike (eng. *Image processing*)
- Analiza i razumijevanje slike (eng. *Image analysis and understanding*)

Jedna od glavnih prednosti kod ljudi ogleda se u sposobnosti donošenja odluka i davanja smisla onome što se vidi u stvarnom svijetu. Omogućavanje strojevima i računalima ovakvo vizualno razumijevanje pridodalo bi im istu moć, no nije sve tako jednostavno. Neki od problema koji se mogu javiti kod računalnog vida su: percepcija, segmentacija, varijacije pogleda, varijacije osvjetljenja, skaliranje, kretanje, pozadinski šum, višeznačnost, itd.

Sustav za računalni vid sastoji se od jedne ili više kamera, osvjetljenja i računala za obradu slike. Kamera stvara digitalnu sliku stvarnog prostora koji osvjetljava pravilno postavljeno osvjetljenje te se to šalje računalu za obradu (robotski kontroler, stolno računalo, PLC, ...) koje nakon toga procesira dobivenu sliku, tj. daje joj određeno značenje. [10]

2.6. Python

Za razvoj navedenog sustava – odnosno programa – koristit će se programski jezik Python. Python programski je jezik visoke razine i opće namjene. Pythonovoj filozofiji dizajna na prvom je mjestu čitljivost koda uporabom značajnog razmaka – tab-a. Njegov objektno orijentirani pristup pomoć je programerima da napišu jasan i logičan kod za male, ali isto tako i za velike projekte. Podržava više paradigmi programiranja, uključujući strukturirano (posebno proceduralno), objektno orijentirano i funkcionalno programiranje. [11]

Razvijen je krajem 1980-ih, a 1991. godine objavio ga je Guido van Rossum kao nasljednika programskog jezika ABC. Dobio je ime prema satiričnoj seriji Monty Python's Flying Circus, iz 1970-ih godina. Python 2.0, objavljen 2000. godine predstavio je nove značajke. Python 3.0, objavljen 2008. godine, bio je glavna revizija jezika koji nije u potpunosti unatrag kompatibilan, a velik dio Python 2 koda ne radi neizmijenjen na Pythonu 3. S krajem Pythona 2, podržani su samo Python 3.6.x i novije verzije, a starije verzije i dalje podržavaju npr. Windows 7 (i stariji operacijski sustavi koji nisu ograničeni na 64-bitni Windows). [11][12]

Od travnja 2022. Python zauzima prvo mjesto u TIOBE-ovom indeksu najpopularnijih programskih jezika, ispred C-a i Jave, s rastom od 2.88% u odnosu na prethodnu godinu. [13]

Temeljna filozofija jezika sažeta je u dokumentu Zen of Python (PEP 20), koji uključuje aforizme poput [14]:

- Bolje je lijepo nego ružno (eng. *Beautiful is better than ugly*)
- Bolje je eksplicitno nego implicitno (eng. *Explicit is better than implicit*)
- Bolje je jednostavno nego kompleksno (eng. *Simple is better than complex*)
- Bolje je kompleksno nego komplicirano (eng. *Complex is better than complicated*)
- Čitljivost se također računa (eng. *Readability counts*)

Navedena filozofija olakšava programiranje u Pythonu zbog lakog učenja sintakse, dok početnicima pruža ugodno i jednostavno iskustvo učenja novog programskog jezika i programiranja.

Umjesto da svu svoju funkcionalnost ugradi u jezgru programa, Python je dizajniran da bude vrlo proširiv. Upravo ta kompaktna modularnost učinila ga je posebno popularnim kao način dodavanja programabilnih sučelja postojećim aplikacijama. Smišljen je kao lako čitljivi jezik. Za razliku od mnogih drugih jezika, on ne koristi vitičaste zagrade za razgraničenje blokova, a točka zarez nakon izraza nisu obavezni i ima manje sintaktičkih iznimaka i posebnih slova kao C ili Pascal. [15]

Primjer Python programskog koda:

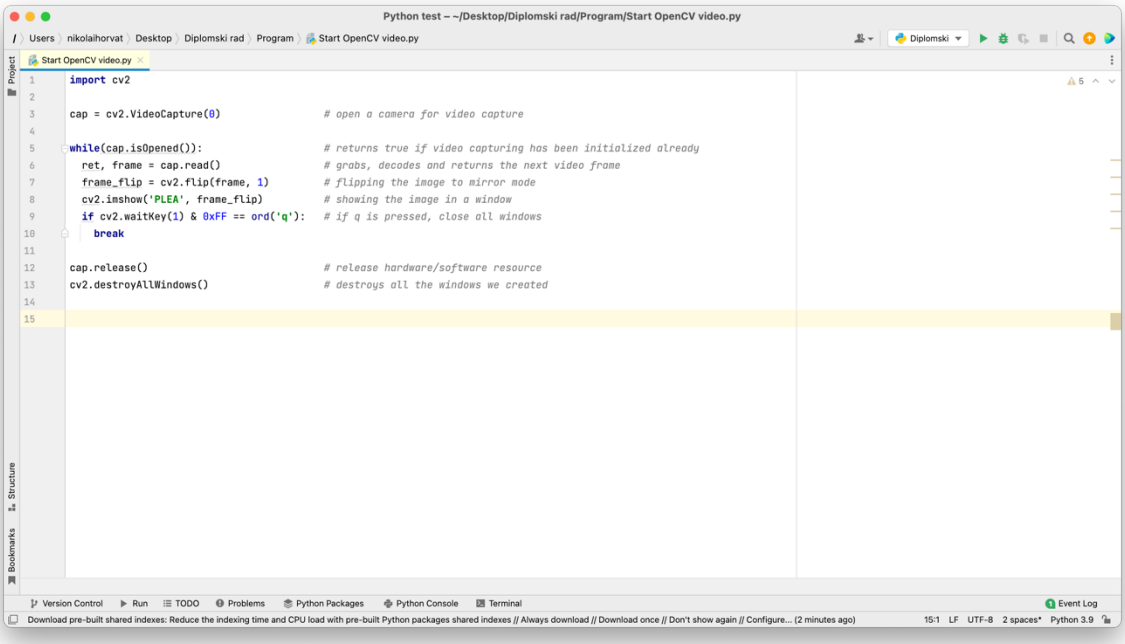
```
import cv2

cap = cv2.VideoCapture(0)

while(cap.isOpened()):
  ret, frame = cap.read()
  frame_flip = cv2.flip(frame, 1)
  cv2.imshow('Camera', frame_flip)
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

Kako je moguće vidjeti, programski jezik Python vrlo je intuitivan i lagan za čitanje te ga mogu shvatiti i početnici – kojima je ovo vrlo vjerojatno prvi programski jezik. Programiranje Python koda može biti obavljeno pomoću mnogo različitih programskih alata, no u ovom će se radu razmatrati programski paket PyCharm.

2.7. PyCharm

PyCharm integrirano je razvojno okruženje (eng. *integrated development environment* – IDE) koje pruža širok raspon osnovnih alata, čvrsto integriranih za stvaranje prikladnog okruženja produktivnog razvoja Pythona, weba i znanosti o podacima (ili podatkovna znanost, eng. *data science*). [16]

Izgled programskog paketa PyCharm dan je ispod u **Slika 8**.



Slika 8: Izgled sučelja programskog paketa PyCharm

PyCharm dostupan je u tri izdanja:

- Community (besplatno i otvorenog koda[3]): za pametan i inteligentan razvoj Pythona, uključujući pomoć kodu, refaktoriranje, vizualno otklanjanje pogrešaka i integraciju kontrole verzija.
- Professional (potrebno plaćanje) : za profesionalni razvoj Pythona, weba i podatkovne znanosti, uključujući pomoć koda, refaktoriranje, vizualno otklanjanje pogrešaka, integraciju kontrole verzija, daljinske konfiguracije, implementaciju, podršku za popularne web okvire, kao što su Django i Flask, podršku baze podataka, znanstvene alate (uključujući podršku za Jupyter notebook).
- Edu (besplatno i otvorenog koda): za učenje programskih jezika i srodnih tehnologija s integriranim obrazovnim alatima.

PyCharm podržava sljedeće verzije Pythona [16]:

- Python 2: verzija 2.7
- Python 3: od verzije 3.6 do verzije 3.11

[3] eng. open-sourced

2.8. OpenCV

OpenCV (*Open Source Computer Vision Library*) biblioteka je računalnog vida i strojnog učenja otvorenog koda. 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, identificiranje objekata, klasificiranje ljudskih radnji u videozapisima, praćenje kretanja kamere, praćenje pokretnih objekata, izdvajanje 3D modela objekata, spajanje slika kako bi se proizvela slika visoke razlučivosti cijele scene, uklanjanje crvenih, itd. Ima C++, Python, Java i MATLAB sučelja i podržava Windows, Linux, Android i Mac OS. Trenutno je u svijetu biblioteka preuzeta već 18 milijuna puta, što od entuzijasta za projekte kućne radinosti do tvrtki, različitih istraživačkih skupina i državnih tijela. [17]

Za detekciju lica razmotriti će se poznati model strojnog učenja: kaskadni klasifikator temeljen na Haar značajkama (Viola-Jones algoritam), a za detekciju pokreta razmotrit će se optical flow algoritam (Farneback optical flow).

2.8.1. Viola – Jones algoritam

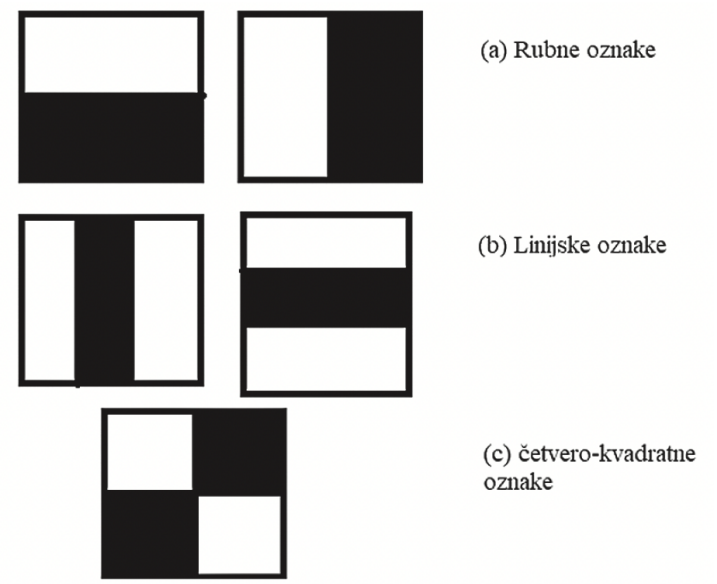
Detekcija lica jedan je od najsloženijih i najizazovnijih problema u području računalnog vida zbog mnoštva varijacija uzrokovanih promjenama u izgledu lica, osvjetljenju i izrazu lica. Primjenom u stvarnom vremenu (eng. *real time*) poput nadzora i biometrijskog skeniranja, ograničenja kamere i varijacije poza čine raspodjelu ljudskih lica u prostoru značajki (eng. *feature space*) složenijom od skeniranja samo prednjeg dijela lica što dodatno komplicira robusne metode detekcije lica. Postoji mnogo metoda i algoritama koji su istraživani godinama, no većina je koncentrirana na detekciju prednjeg dijela lica s zadovoljavajućim osvjetljenjem.

Paul Viola i Michael Jones predstavili su brzu i robusnu metodu za detekciju lica koja je 15 puta brža od bilo koje tehnike (u trenutku objavljivanja algoritma) s preciznošću od 95 % pri 17 sličica u sekundi, tj. fps-a. [18]

2.8.1.1. Haar značajke

Opće je poznato da postoje neke sličnosti između svih ljudskih lica, stoga algoritam traži specifične Haar značajke (značajke pravokutnog oblika umjesto analize piksela, **Slika 9**) lica i

njihovim pronalaskom, algoritam prosljeđuje „kandidata“, tj. sliku u sljedeću fazu. Ovdje kandidat nije cijela slika, već samo kvadratni dio slike veličine 24x24 piksela i provjerava se cijela slika, kvadrat po kvadrat.

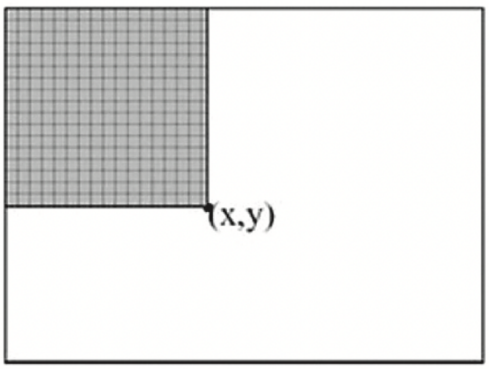


Slika 9: Osnovne Haar značajke [19]

Haar značajke sastoje se od dva ili tri pravokutnika i koriste se kako bi se utvrdila prisutnost odnosno neprisutnost lica. Svaka značajka ima svoju vrijednost i ona se može izračunati pomoću površine svakog pravokutnika koja se dobije iz integralne slike. [18]

2.8.1.2. Integralna slika

Integralna slika definirana je kao zbroj vrijednosti piksela izvorne slike. Vrijednost bilo koje lokacije definirane Kartezijevim koordinatama (*x, y*) zbroj je piksela lijevo i iznad od lokacije, (označeno sivom bojom u **Slika 10**) uzimajući u obzir da je ishodište u gornjem lijevom kutu.

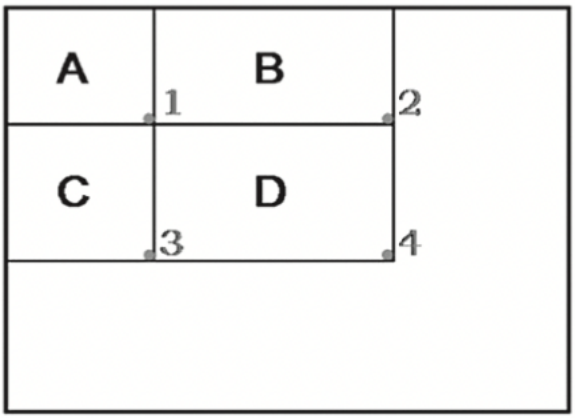


Slika 10: Element integralne slike [19]

Za izračun površine *S* pravokutnika D (**Slika 11**), potrebno je uzeti u obzir ne samo vrijednost **4**, nego i vrijednosti **1**, **2** i **3**. Korištenje tih četiri vrijednosti omogućuje vrlo brz izračun površine prema jednadžbi (1).

$$S = A_4 - A_3 - A_2 + A_1 \tag{1}$$

U jednadžbi (1) dodaje se na kraju vrijednost A_1 – površina elementa A – zbog toga što se ona oduzima u dva navrata, jednom pri oduzimanju vrijednosti A_3 (koja se sastoji od pravokutnika A i C) i jednom pri oduzimanju vrijednosti A_2 (koja se sastoji od pravokutnika A i B). [19]



Slika 11: Prikaz integralne slike po elementima [19]

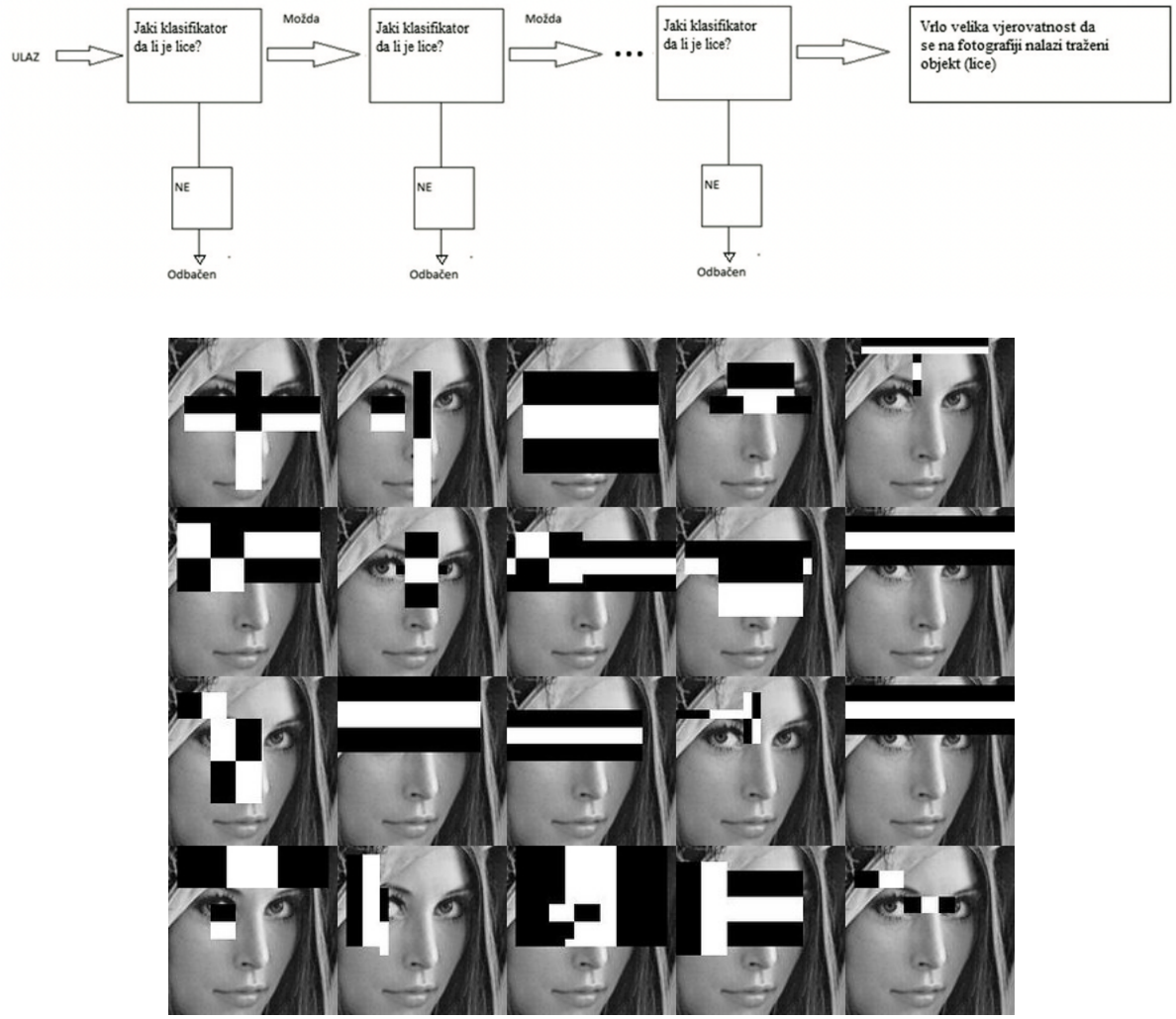
Haar klasifikator značajki množi težinu svakog pravokutnika s njegovom površinom kako bi dobio neki rezultat. Obzirom da je potrebno usporediti pravokutnike, nekoliko klasifikatora čine fazu, odnosno stupanj (eng. *stage*). Tada komparator faza uspoređuje sve rezultate Haar klasifikatora s pragom osjetljivosti, koji je dobiven pomoću AdaBoost algoritma[4]. Nije potrebno da svaka faza ima jednak broj Haar značajki, npr. skup podataka Paula Viole i Michaela Jonesa koristio je dvije značajke u prvoj fazi i deset u drugoj, dok se sve zajedno koristilo 38 faza i preko 6000 značajki. [18]

Iako se koristi uspoređivanje različitih faza, „lažni“ se kandidati još uvijek mogu omaknuti, stoga je potrebno koristiti kaskadni klasifikator.

[4] Obavlja funkciju poboljšanja performansi jednostavnog algoritma strojnog učenja

2.8.1.3. Kaskadni klasifikator

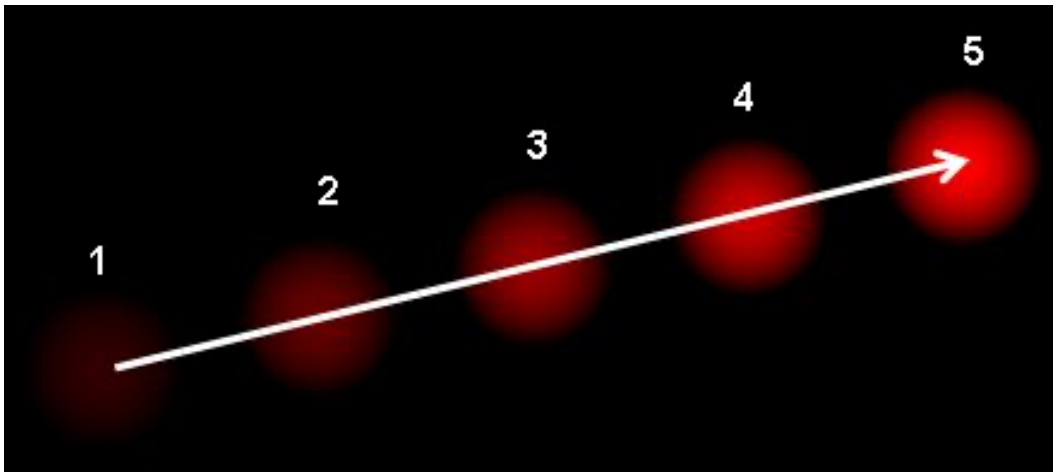
Kaskadni klasifikator eliminira kandidata ako nije prošao prvu fazu, a ako je prošao, šalje ga na sljedeću, složeniju fazu. Dakle, lice je prepoznato ako je fotografija prošla sve faze klasifikatora (**Slika 12**). [18]



Slika 12: Struktura kaskadnog klasifikatora (gore), primjer Haar značajki (dolje) [19]

2.8.2. Optical flow

Optical flow uzorak je prividnog kretanja objekata na slici, između dvije uzastopne slike, nastalog uslijed pomicanja kamere ili samog objekta. Ono čini dvodimenzionalno vektorsko polje u kojem svaki vektor prikazuje kretanje točaka između dvaju kadrova. [20]



Slika 13: Kretanje lopte u 5 uzastopnih slika [20]

Na primjer, u **Slika 13** prikazano je kretanje lopte kroz 5 uzastopnih slika u kojima prikazani vektor označava vektor pomaka same lopte. Optical flow algoritam ima mnoge primjene poput stabilizacije i kompresije videozapisa, no u ovom će se radu primijeniti za samu detekciju pokreta unutar videozapisa.

Optical flow radi koristeći dvije pretpostavke:

- Intenzitet piksela objekta ne mijenja se između uzastopnih kadrova
- Susjedni pikseli imaju slično kretanje

Intenzitet piksela *I* definiran je pomoću tri varijable: *x* i *y* vezane uz koordinate samog piksela uz vrijeme *t*. Prilikom rada sa slikama, nije bilo potrebno koristiti vrijeme – ono je potrebno za određivanje brzine kretanja. Kretanje piksela opisano je pomicanjem za udaljenosti *dx* i *dy* u sljedećem kadru snimljenog nakon *dt* vremena. Obzirom da su pikseli isti i intenzitet se ne mijenja, može se reći:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \quad (2)$$

Aproksimacijom Taylorovog reda na desnoj strani, uklanjaju se uobičajeni članovi i dijele s vremenskim intervalom *dt* da se dobije:

$$I_x u + I_y v + I_t = 0 \tag{3}$$

gdje je:

$$I_x = \frac{\partial I}{\partial x} \tag{4}$$

$$I_y = \frac{\partial f}{\partial y} \tag{5}$$

$$u = \frac{\partial x}{\partial t} \tag{6}$$

$$v = \frac{\partial y}{\partial t} \tag{7}$$

Vrijednost I_x gradijent je piksela duž *x*-osi, I_y gradijent je piksela duž *y*-osi, I_t gradijent je kroz vrijeme, dok su vrijednosti *u* i *v* nepoznanice. Jednadžba (3) se ne može riješiti direktno zbog postojanja dviju nepoznanica, stoga postoji nekoliko metoda za rješavanje – jedna je od njih algoritam Gunnara Farnebacka, dense optical flow koji računa pomake za sve točke, tj. piksele na slici. [21]

Slika 14: Primjer originalne slike i HSV prikaz kretnji slike [21]

2.9. HTML

HyperText Markup Language, ili kratko HTML, osnovni je gradivni blok weba. Web preglednici poput Google Chromea, Mozille Firefox, Safarija i Internet Explorera preuzimaju HTML dokumente s web poslužitelja ili iz lokalne pohrane i pretvaraju jednostavne tekstualne dokumente u multimedijske web stranice. HTML opisuje samu strukturu web stranice i izvorno uključuje informacije o izgledu dokumenta. Njegovi elementi građevni su blokovi stranica – konstrukcija stranice, slike i drugi elementi poput interaktivnih formulara mogu biti ugrađeni u prikazanu stranicu. On pruža sredstva za stvaranje strukturiranih dokumenata, označavajući strukturnu semantiku za tekst, kao što su naslovi, odlomci, razni popisi, hiperveze, citati i dr.

HTML elementi opisani su oznakom koja je smještena unutar kutnih zagrada. Na primjer, oznake <img /> i <input /> izravno uvode sadržaj na stranicu, dok oznake poput <p> okružuju tekst i mogu uključivati druge oznake kao podelemente. Te se oznake (eng. *tag*) ne prikazuju na web stranici, već ih preglednici koriste za tumačenje sadržaja stranice i pravilno prikazivanje.

U strukturu HTML dokumenta mogu se ugraditi i programi napisani u skriptnom jeziku, kao što je JavaScript, koji ima utjecaj na interaktivna sučelja web stranice i određuje njezino ponašanje. Uključivanjem CSS-a (Cascading Style Sheets) u dokument može se detaljnije definirati izgled web stranice. [22]

2.9.1. JavaScript

JavaScript (JS) „lagan“ je i interpretiran programski jezik. Iako je najpoznatiji kao skriptni jezik za web stranice, koriste ga i mnoga druga programska okruženja koja nisu preglednici – Node.js, Apache CouchDB i Adobe Acrobat. Programski kod pokreće se na strani posjetitelja web stranice, što se može iskoristiti za programiranje interaktivne web stranice. JS jednostavan je za učenje i moćan je alat koji se naširoko koristi za upravljanje ponašanjem web stranice. [23]

2.9.2. CSS

Cascading Style Sheets (CSS) jezik je „stilskih“ tablica koji se koristi za opisivanje izgleda i prezentacije dokumenta koji je pisan u označnom jeziku (eng. *Markup language*) kao što je HTML ili XML (uključujući XML dijalekte SVG, MathML ili XHTML).

CSS konstruiran je s ciljem da omogući odvajanje prezentacije i sadržaja, uključujući izgled, boje i fontove. Ono može pružiti veću fleksibilnost i kontrolu u specifikaciji karakteristika prezentacije, kao i mogućnost korištenja jedne CSS datoteke web sjedištima, koja se sastoje od više web stranica što smanjuje složenost i ponavljanje koda.

Sadržaj riječi *Cascade* označava shemu prioriteta primjene stila ako se više od jednog pravila podudara s određenim elementom. [24]

3. IZRADA PROGRAMSKE APLIKACIJE

Kako bi se mogao implementirati, zadatak je podijeljen u dva dijela:

- Izrada Python programskih aplikacija
- Izrada Web stranice

3.1. Izrada Python programskih aplikacija

3.1.1. Aplikacija za detekciju lica

Pomoću programskog paketa PyCharm, programskog jezika Python i biblioteke OpenCV razradit će se programska aplikacija.

Kao i kod svakog projekta, pa tako i ovog, potrebno je na početku datoteke učitati potrebne biblioteke – to su OpenCV s oznakon *cv2*, biblioteku za upravljanje operacijskim sustavom s oznakom *os* i biblioteku *time* s istoimenom oznakom za upravljanje vremenom. Postupak je prikazan programskim kodom danom u nastavku:

```
import cv2, os, time
```

Nakon učitavanja biblioteka, potrebno je inicijalno postaviti potrebne vrijednosti i varijable koje će se kasnije u programu koristiti:

- Postavljanje varijable *cap* kao objekt klase VideoCapture, a u zagradi je postavljen indeks kamere (0 za ugrađenu kameru u laptopu, 1 za dodanu kameru, itd.).

```
cap = cv2.VideoCapture(0)
```

- Definiranje varijable *face_cascade* kao objekt klase CascadeClassifier zadane putanje detektora lica temeljenog na strojnom učenju *casc_path*. Detektor lica temelji se na algoritmu Paula Viole i Michaela Jonesa.

```
casc_path =
os.path.dirname(cv2.__file__)+"/data/haarcascade_frontalface_default.x
ml"
face_cascade = cv2.CascadeClassifier(casc_path)
```

- Postavljanje visine, širine, tipa i imena videozapisa – za ime videozapisa korišten je američki zapis datuma: godina, mjesec, dan – zbog automatskog sortiranja prema imenu datoteke, tj. starosti videozapisa, što s europskim načinom ne bi bio slučaj.

```
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
file_type = '.mp4'
file_name = time.strftime("%Y %m %d - %H %M %S" + file_type)
```

- Postavljanje varijable *writer* koja inicijalizira spremanje videozapisa.

```
fourcc = cv2.VideoWriter_fourcc(*'AVC1')
writer = cv2.VideoWriter(file_name, fourcc, wanted_fps, (width,
height))
```

- Postavljanje dodatnih, pomoćnih varijabli: varijabla *wanted_fps* određuje broj sličica u sekundi, *rectangle_width* zadaje širinu ruba pravokutnika koji će „okružiti“ lice subjekta, *start_timer* sprema trenutno vrijeme koje se koristi za analizu trajanja videozapisa, dok su varijable *counter* i *save_counter* pomoćni brojači koji će pomoći u određivanju uvjeta.

```
wanted_fps = 15
rectangle_width = 1

start_timer = time.time()
counter = 0
save_counter = 0
```

Nakon što su se postavile inicijalne vrijednosti, potrebno je kreirati while petlju zato što je prikaz videozapisa u suštini brz prikaz pojedinačnih slika, od kojih se svaka analizira u petlji na prisutnost lica. Ona se postavlja na sljedeći način:

```
while uvjet:
  naredba1
  naredba2
  naredba3
```

gdje uvjet može biti postavljen kao krajnja vrijednost Booleove logike – točno (eng. *true*) što će se u ovom slučaju koristiti jer je potrebno da se petlja neprestano ponavlja. Otvaranjem petlje potrebno je inicijalizirati mjerač vremena *fps_start* pomoću kojeg će se računati broj sličica u sekundi (fps) s ciljem ispravne kalibracije brzine spremanja videozapisa – vrijednost

wanted_fps postavi se na prikazanu vrijednost fps-a. Također, potrebno je definirati dvije varijable *ret* i *frame* koje su izlaz metode *read* objekta *cap.* Varijabla *ret* daje vrijednost točno ili netočno ovisno o uspješnosti vraćanja slike, a varijabla *frame* sadrži samu sliku.

```
fps_start = time.time()
ret, frame = cap.read()
```

Uspješnim spremanjem slike u varijablu *frame*, potrebno je definirati varijablu *faces.* Metoda *detectMultiScale* unutar objekta *face_cascade* vraća polje svih detektiranih lica, a kao ulaz zahtjeva sliku *frame*, parametar smanjenja slike *scaleFactor*, parametar *minNeighbors* koji označava minimalan zahtijevani broj susjednih elemenata da se zadrži kandidat lica, minimalnu veličinu pravokutnika koji će opisivati lice *minSize* i napomenu *flags* koja iznosi 0 jer se koristila samo u starijim verzijama OpenCV biblioteka.

```
faces = face_cascade.detectMultiScale(
    frame,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    flags=0
    )
```

Svaki element u polju lica *faces* definiran je s četiri vrijednosti: *x* i *y* koji definiraju koordinate unutar slike; *w* i *h* koji definiraju širinu i visinu pravokutnika koji opisuje lice. Koordinate su zadane kao koordinate najbližeg kuta pravokutnika ishodištu. Zbog toga što polje *faces* sadrži više elemenata, potrebna je for petlja koja prolazi po svim elementima te iscrtava pravokutnike oko lica na slici *frame* naredbom *rectangle* koja kao ulaz zahtijeva sliku *frame*, početne i krajnje koordinate, boju i širinu ruba pravokutnika. Nakon iscrtavanja pravokutnika potrebno je prikazati sliku u zasebnom prozoru naredbom *imshow* koja za ulaz treba ime prozora (koje može biti proizvoljno) i sliku *frame*.

```
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 255), rectangle_width)
cv2.imshow('Face detection', frame)
```

Nakon razrade osnovnog dijela programa, tj. detektiranjem lica, potrebno je odrediti okidač za pokretanje spremanja videozapisa. Najprije je potrebno odrediti broj lica, tj. provjeriti sadrži li polje *faces* uopće lica ili ne – što je moguće napraviti naredbom *len* koja vraća broj elemenata unutar polja – i tu je vrijednost potrebno pridružiti varijabli *len_faces.*

```
len_faces = len(faces)
```

Nakon toga, potrebno je odrediti uvjete. Ako slika sadrži lica, tj. broj elemenata u polju *faces* različit je od nule, izvršit će se sljedeći kod:

```
if len_faces != 0:
    start_timer = time.time()
    end_timer = time.time()
    counter = 0
    file_name = time.strftime("%Y %m %d - %H %M %S" + file_type)
    if save_counter == 0:
       save_counter = 1
       writer = cv2.VideoWriter(file_name, fourcc, wanted_fps, (width,
       height))
      duration_start = time.time()
      file_name_check = file_name
    writer.write(frame)
```

Mjerač vremena *start_timer* bit će korišten za izmjeru trajanja videozapisa bez pojave lica, stoga je potrebno postaviti mjerač na trenutno vrijeme ukoliko je lice detektirano. Također, mjerač vremena *end_timer* bit će korišten za izmjeru trajanja videozapisa bez pojave lica, no za razliku od *start_timer*, on će se osvježiti svaki put kada lice nije detektirano – što je moguće vidjeti u programskom kodu ispod odlomka. Brojač *counter*, koji je korišten za oznaku prestanka spremanja, postavlja se na inicijalnu vrijednost, nulu. Pomoću brojača *save_counter* određuje se je li započelo spremanje videozapisa ili nije – ako je, sličica se spremi pomoću metode *write* unutar objekta *writer*, a ako nije, brojač *save_counter* se stavi na vrijednost jedan, ime videozapisa se postavlja na trenutno vrijeme varijablom *file_name*, kao što je opisano ranije. Pokreće se mjerač vremena trajanja *duration_start* te varijabla *file_name_check* poprima vrijednost trenutnog imena videozapisa i ono će biti korišteno pri brisanju nedovoljno dugih videozapisa.

```
if len_faces == 0:
    end_timer = time.time()
```

Svakim izvođenjem petlje lice je prepoznato ili nije, stoga je potrebno odrediti vrijeme koje je prošlo od zadnje pojave lica unutar kadra. To će se odrediti pomoću varijable *elapsed_time*.

```
elapsed_time = round(end_timer - start_timer, 2)
```

Ako je proteklo vrijeme duže od 3 sekunde i video je spreman (što je vidljivo postavljanjem brojača *counter* na nulu), tada se brojaču *counter* pridruži vrijednost jedan, brojač snimanja *save_counter* se postavlja na nulu te se prekida snimanje metodom *release*. Kraj videozapisa označen je varijablom *duration_end* i trajanje *duration* se tada može lako izračunati. Metodom pokušaja i pogrešaka zaključeno je da je, uz postojeće inicijalne parametre, minimalno trajanje „korisnog“ videozapisa barem 6 sekundi. Ako je ono manje, videozapis se odmah briše da dodatno ne zauzima prostor na mediju spremanja te se ispisuje odgovarajuća poruka.

```
if elapsed_time >= 3 and counter == 0:
    counter = 1
    save_counter = 0
    writer.release()
    duration_end = time.time()
    duration = round((duration_end - duration_start), 2)
    if duration < 6:
       os.remove(file_name_check)
       print('File ' + file_name_check + ' is removed. Duration: ' +
       str(duration))
```

Ako je trajanje videozapisa duže ili jednako šest sekundi, datoteka se premješta u arhivu, s obzirom na to da se videozapis izvorno sprema u mapu gdje je i Python datoteka. Destinacija premještanja definirana je varijablom *new_path*, a premješta se pomoću metode *replace* koja kao ulaz zahtjeva trenutnu destinaciju spremanja i novu.

```
  if duration >= 6:
       new_path = "web/Archive/face_tracking/" + file_name_check
       os.replace(file_name_check, new_path)
```

Svakim prolaskom kroz petlju ispisuje se broj sličica u sekundi, no to je u suštini samo trajanje petlje koje se podijeli s jedan. Pritiskom tipke **q**, petlja se prekida i kreće nastavak programa izvan petlje.

```
print("FPS: ", 1.0 / (time.time() - fps_start))
if cv2.waitKey(1) & 0xFF == ord('q'):
       break
```

Izlaskom programa iz petlje, „otpušta“ se veza kamere sa softverom, prekida se spremanje videozapisa te se terminiraju svi prozori.

```
cap.release()
writer.release()
cv2.destroyAllWindows()
```

3.1.2. Aplikacija za detekciju pokreta

Kao i kod aplikacije za detekciju lica (poglavlje 3.1.1.), na početku programa zadatak je učitati potrebne biblioteke koje su jednake onima iz prethodnog poglavlja, no nužno je učitati još jednu biblioteku koja je napravljena s ciljem bolje preglednosti programskog koda.

```
from optical_flow_functions import draw_flow, draw_hsv
```

Nakon učitavanja biblioteka, potrebno je inicijalno postaviti potrebne vrijednosti i varijable koje će se kasnije u programu koristiti:

- Postavljanje varijable *cap* kao objekt klase VideoCapture te je u zagradi postavljen indeks kamere (0 za ugrađenu kameru u laptopu, 1 za dodanu kameru, itd.).

```
cap = cv2.VideoCapture(0)
```

- Potrebno je definirati dvije varijable *ret* i *frame* koje čine isto kao i u prethodnom poglavlju, daju uspješnost vraćanja slike i sadrže samu sliku. Slika sadržana u *frame*, sprema se kao slika u nijansama sive u varijablu *prev_gray* pomoću metode *cvtColor* koja kao ulaz zahtjeva sliku i prostor boja (eng. *color-space*)

```
ret, frame = cap.read()
prev_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

- Ostatak inicijalnog postavljanja varijabli jednak je prethodnom programu, bez dijelova vezanih za detekciju lica s dodanim dijelom gdje se definira varijabla *writer_norm* koja inicijalizira spremanje „čistog“ videozapisa, no potrebno je definirati i ulazne parametre za dense optical flow algoritam. *pyr_scale* parametar je stvaranja piramide slojeva slike, gdje je pri vrijednosti od 0.5 svaki sljedeći sloj dvostruko manji od prethodnog. Varijablom *levels* definira se broj slojeva piramide, a broj jedan označavao bi korištenje samo originalnih slika. *winsize* definira veličinu „prozorčića“ (skup piksela) koji se analiziraju za kretnju – što je veći broj, manja je osjetljivost. Broj iteracija algoritma na svakom sloju piramide definiran je varijablom *iterations. poly_n* definira veličinu susjedstva piksela korištenih za polinomsku ekspanziju, a *poly_sigma* standardnu devijaciju Gaussian filtera s ciljem zaglađivanja derivacija. Napomena *flags* nema, dakle vrijednost je nula.

```
file_name_norm = time.strftime("%Y %m %d - %H %M %S" + " - normal" +
file_type)
writer_norm = cv2.VideoWriter(file_name_norm, fourcc, wanted_fps,
(width, height))

pyr_scale = 0.5
levels = 3
winsize = 5
iterations = 1
poly_n = 7
poly_sigma = 1.5
flags = 0
```

Kao i kod programa za detekciju lica, potrebno je kreirati while petlju u kojoj će se svaka slika s prethodnom analizirati na kretnju prema algoritmu Gunnara Farnebacka. Odmah na početku petlje, potrebno je inicijalizirati mjerač vremena *fps_start* i varijable *ret* i *frame* čije je značenje objašnjeno ranije.

```
fps_start = time.time()
ret, frame = cap.read()
```

Nakon osigurane slike u varijabli *frame,* ona se prebacuje u prostor sive boje i sprema u varijablu *gray.* Nakon toga stvara se objekt *flow* metode *calcOpticalFlowFarneback* koja kao ulaz zahtjeva prethodnu sliku *prev_gray,* trenutnu sliku *gray* i sve ostale već spomenute parametre. Nakon spremanja u *flow,* trenutna slika *gray* postaje prethodna slika *prev_gray* koja će se koristiti u idućoj iteraciji while petlje.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, pyr_scale,
levels, winsize, iterations, poly_n, poly_sigma, flags)
prev_gray = gray
```

Potrebno je stvoriti četiri varijable: tri (*img_bgr, fx_mean, fy_mean*) za vraćene vrijednosti iz metode *draw_flow* i jednu (*img_hsv*) za vraćenu vrijednost iz metode *draw_hsv.* Navedene metode, smještene u posebnoj biblioteci, objašnjene su nakon pregleda cjelokupnog koda:

```
img_bgr, fx_mean, fy_mean = draw_flow(gray, flow)
img_hsv = draw_hsv(flow)
```

Nakon što su vraćene vrijednosti *img_bgr* i *img_hsv*, potrebno ih je prikazati u zasebnom zaslonu zajedno s „čistim“ prikazom pomoću metode *imshow* koja kao ulaz traži ime prozora i sliku.

```
        cv2.imshow('Normal', frame)
        cv2.imshow('Flow', img_bgr)
        cv2.imshow('HSV', img_hsv)
```

Izlazne vrijednosti metode *draw_flow,* također su i srednje vrijednosti *N* najvećih pomaka u smjeru *x* i *y*-osi (detaljnije je objašnjeno u poglavlju 3.1.2.1.) i one će biti korištene pri uvjetu postojanja odnosno nepostojanja pomaka. Metodom pokušaja i pogrešaka pronađena je vrijednost 5 kao optimalna vrijednost detekcije pomaka, zanemarujući pozadinski šum. Ako je srednja vrijednost pomaka u smjeru bilo koje osi veća od 5, pokreće se sljedeći programski kod:

```
if fx_mean >= 5 or fy_mean >= 5:
        start_timer = time.time()
        end_timer = time.time()
        counter = 0
        file_name = time.strftime("%Y %m %d - %H %M %S" + file_type)
        file_name_norm = time.strftime("%Y %m %d - %H %M %S" + " - normal" +
        file_type)
        if save_counter == 0:
                save_counter = 1
                writer = cv2.VideoWriter(file_name, fourcc, wanted_fps, (width,
                height))
                writer_norm = cv2.VideoWriter(file_name_norm, fourcc, wanted_fps,
                (width, height))
                duration_start = time.time()
                file_name_check = file_name
                file_name_check_norm = file_name_norm
        writer.write(img_hsv)
        writer_norm.write(frame)
```

Mjerač vremena *start_timer* bit će korišten za izmjeru trajanja videozapisa bez kretnje, stoga je potrebno postaviti ga na trenutno vrijeme, ako je bilo kakvo kretanje prepoznato.

Također, mjerač vremena *end_timer* bit će korišten za izmjeru trajanja videozapisa bez kretnje, no za razliku od *start_timer*, on će se osvježiti svaki put kad kretanje nije detektirano, što je moguće vidjeti u programskom kodu ispod odlomka. Brojač *counter*, koji je korišten za oznaku prestanka spremanja, postavlja se na inicijalnu vrijednost, nulu. Pomoću brojača *save_counter* određuje se je li započelo spremanje videozapisa ili nije – ako je, sličica HSV prikaza se spremi pomoću metode *write* unutar objekta *writer* i sličica „čistog“ prikaza spremi se pomoću metode *write* unutar objekta *writer_norm*. Ako nije, brojač *save_counter* se stavi na vrijednost jedan, ime videozapisa se postavlja na trenutno vrijeme varijablom *file_name* i *file_name_norm* i kao

što je opisano ranije, pokreće se mjerač vremena trajanja *duration_start* te varijabla *file_name_check* i *file_name_check_norm* poprima vrijednost trenutnog imena videozapisa i ono će biti korišteno kod brisanja nedovoljno dugih videozapisa.

```
if fx_mean < 5 and fy_mean < 5:
        end_timer = time.time()
```

Svakim izvođenjem petlje, kretnja je prepoznata ili nije, zato je potrebno odrediti vrijeme koje je prošlo od zadnje pojave kretanja. To će se odrediti pomoću varijable *elapsed_time*.

```
elapsed_time = round(end_timer - start_timer, 2)
```

Ukoliko je proteklo vrijeme duže od 3 sekunde i video je spreman (što je vidljivo postavljanjem brojača *counter* na nulu), tada se brojaču *counter* pridruži vrijednost jedan, brojač snimanja *save_counter* se postavlja na nulu te se prekida snimanje metodom *release*. Kraj videozapisa označen je varijablom *duration_end* i čije se trajanje *duration* tada može lako izračunati. Metodom pokušaja i pogrešaka zaključeno je da je, uz postojeće inicijalne parametre, minimalno trajanje „korisnog“ videozapisa barem 6 sekundi, kao i u programu za detekciju lica. Ako je ono manje, videozapisi se odmah brišu kako dodatno ne bi zauzimali prostor na mediju spremanja te se ispisuje odgovarajuća poruka.

```
if elapsed_time >= 3 and counter == 0:
  counter = 1
  save_counter = 0
  writer.release()
  writer_norm.release()
  duration_end = time.time()
  duration = round((duration_end - duration_start), 2)
  if duration < 6:
       os.remove(file_name_check)
       os.remove(file_name_check_norm)
       print('File ' + file_name_check + ' is removed. Duration: ' +
       str(duration))
       print('File ' + file_name_check_norm + ' is removed. Duration: ' +
       str(duration))
```

Kada je trajanje videozapisa duže ili jednako šest sekundi, datoteka se premješta u arhivu zbog toga što se videozapis izvorno sprema u mapu gdje je i Python datoteka. Destinacija premještanja definirana je varijablom *new_path* i *new_path_norm*, a premješta se pomoću metode *replace* koja kao ulaz zahtjeva trenutnu destinaciju spremanja i novu.

```
if duration >= 6:
        new_path = "web/Archive/optical_flow/" + file_name_check
        os.replace(file_name_check, new_path)
        new_path_norm = "web/Archive/normal_video/" + file_name_check_norm
        os.replace(file_name_check_norm, new_path_norm)
```

Svakim prolaskom kroz petlju ispisuje se broj sličica u sekundi, no to je u suštini samo trajanje petlje koje se podijeli s jedan. Pritiskom tipke **q**, petlja se prekida i kreće nastavak programa izvan petlje.

```
print("FPS: ", 1.0 / (time.time() - fps_start))
if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Izlaskom programa iz petlje, „otpušta“ se veza kamere sa softverom, prekida se spremanje videozapisa te se terminiraju svi prozori.

```
cap.release()
writer.release()
writer_norm.release()
cv2.destroyAllWindows()
```

Kao što je vidljivo iz priloženog koda, analiza okidača snimanja i postavljanje programskog koda, vrši se gotovo na jednak način u programu za detekciju lica i detekciju pokreta uz minimalne konfiguracijske razlike, no uvelike se razlikuju u korištenim metodama za analizu. U nastavku su objašnjene metode *draw_flow* i *draw_hsv* koje analiziraju i prikazuju kretnju na zaslonu na odgovarajući način.

3.1.2.1. draw_flow

Potrebni ulaz za metodu *draw_flow* trenutna su slika *gray* i objekt *flow*. Također, unaprijed zadani ulaz u funkciju, *step*, predodređen je na vrijednost 20, što je razmak između susjednih točaka za prikaz kretnje pomoću linija.

```
def draw_flow(img, flow, step=20):
        ...
```

Varijable *h* i *w* u sebe spremaju visinu i širinu slike, odnosno vraćaju rezoluciju kamere koja snima. Varijablama *y* i *x* definirano je polje točaka koje će se prikazati na ekranu i iz kojih će

kretati linije za analizu kretnje, dok su u varijable *fx* i *fy* spremljeni pomaci za svaku točku iz mreže točaka – te vrijednosti mogu biti pozitivne ili negativne.

```
h, w = img.shape[:2]
y, x = np.mgrid[step/2:h:step, step/2:w:step].reshape(2,-1).astype(int)
fx, fy = flow[y, x].T
```

Nakon što se dobiju pomaci, koordinate i koordinate umanjene za pomake, spremaju se u polje *lines*. Metoda *vstack* biblioteke numpy transformira horizontalne elemente liste u vertikalne.

```
lines = np.vstack([x, y, x-fx, y-fy]).T.reshape(-1, 2, 2)
```

Kako bi linije bile vidljivije na slici u nijansama sive, one će se iscrtavati u boji za što je potrebno prebaciti varijablu *img_bgr* u prostor BGR (Blue-Green-Red, svodi se na isto kao i RGB) boja istom metodom kao što je ranije objašnjeno. Prebacivanjem u prostor boja, mogu se iscrtati linije metodom *polylines*, koja kao ulazne parametre zahtjeva sliku *img_bgr,* polje linija *lines,* uvjet zatvorene linije – 0 ako je otvorena, i boju. Nakon iscrtavanja linija, potrebno je iscrtati točke u kojima su bilježeni pomaci, tj. točke iz kojih će polaziti linije. To se može napraviti običnom for petljom koja prolazi po koordinatama unutar polja *lines* i metodom *circle* koja crta kružiće veličine 1 piksel što daje točku.

```
img_bgr = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
cv2.polylines(img_bgr, lines, 0, (145, 96, 255))
for (x1, y1), (_x2, _y2) in lines:
    cv2.circle(img_bgr, (x1, y1), 1, (145, 255, 255))
```

Pomaci *fx* i *fy* potrebni su za stvaranje varijabla *fx_mean* i *fy_mean* koje će služiti kao okidač za spremanje videozapisa. Vrijednost *N* broj je elemenata koji će se uzeti u obzir jer je broj pomaka 2304 zbog toga što se gleda svaka točka, a u mnogo točaka ta je vrijednost zanemariva. Da bi se lakše odredilo npr. 20 točaka koje imaju najveću vrijednost, potrebno je sortirati apsolutne cjelobrojne vrijednosti pomaka i tada su najveće vrijednosti spremljene na kraju liste *fx_sort*. Varijabla *fx_last_elements* sadrži *N* zadnjih, odnosno najvećih elemenata i tada se računa njihova srednja vrijednost i sprema se u varijablu *fx_mean* i ona se zaokružuje na jednu decimalu zato što nije potrebno više od toga. Isti se taj postupak ponovi kako bi se dobila varijabla *fy_mean*.

```
N = 20
fx_sort = np.int32(np.sort(abs(fx)))
fx_last_elements = fx_sort[-N:]
fx_mean = sum(fx_last_elements) / N
```

```
fx_mean = round(fx_mean, 1)

fy_sort = np.int32(np.sort(abs(fy)))
fy_last_elements = fy_sort[-N:]
fy_mean = sum(fy_last_elements) / N
fy_mean = round(fy_mean, 1)
```

Na kraju metode, potrebno je odrediti što ona vraća: sliku s iscrtanim točkama i linijama pokreta *img_bgr* i srednje vrijednosti pomaka *fx_mean* te *fy_mean*.

```
return img_bgr, fx_mean, fy_mean
```

3.1.2.2. draw_hsv

Potrebni ulaz za metodu *draw_hsv* samo je objekt *flow*.

```
def draw_hsv(flow):
        ...
```

Varijable *h* i *w* u sebe spremaju visinu i širinu slike, tj. rezoluciju, a u varijable *fx* i *fy* se sprema vrijednost pomaka za svaki piksel – za razliku od metode *draw_flow* gdje se uzima samo pomak u predodređenim točkama.

```
h, w = flow.shape[:2]
fx, fy = flow[:,:,0], flow[:,:,1]
```

Pomaci će biti prikazani u HSV prostoru boja (Hue-Saturation-Value) – nijansa, zasićenje i vrijednost gdje je nijansa definirana kutem od 0° do 360°, a zasićenje i vrijednost brojčanom vrijednošću od 0 do 255. Kut boje definiran je kao vrijednost arcus tangensa, između pomaka *fx* i *fy* uvećanog za π, da se dobije ispravna vrijednost kuta, tj. boje i za dobivanje varijable *hue* potrebno je kut iz radijana pretvoriti u stupnjeve. Dužina linija *length* (dobije se pomoću Pitagorinog poučka) potrebna je da bi se dobila vrijednost boje *val* koja uzima minimalnu vrijednost od dvije opcije – četverostruku vrijednost duljine (za dovoljnu jačinu boje) ili maksimalnu vrijednost prikaza 255. Zasićenje boje *sat* za sve je vrijednosti predodređeno na 255.

```
ang = np.arctan2(fy, fx) + np.pi
length = np.sqrt(fx*fx+fy*fy)

hue = ang*(180/np.pi/2)
sat = 255
val = np.minimum(length*4, 255)
```

Određivanjem potrebnih vrijednosti boja za svaki piksel, potrebno ih je pridružiti dvodimenzionalnom polju u tri slojeva: nakon inicijalizacije praznog polja prvom je sloju potrebno pridružiti vrijednosti zasićenja *hue*, drugom je potrebno pridružiti vrijednosti zasićenja *sat* koja su uvijek 255 i na kraju je potrebno trećem sloju pridružiti vrijednost boje *val*. Kako bi slika bila ispravno prikazana na zaslonu računala, potrebno je spremiti sliku u BGR prostoru boja u varijablu *bgr* koja će biti vraćeni argument iz metode *draw_hsv.*

```
hsv = np.zeros((h, w, 3), np.uint8)
hsv[..., 0] = hue
hsv[..., 1] = sat
hsv[..., 2] = val
bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

return bgr
```

3.2. Izrada web stranice

Web stranica bit će izrađena pomoću HTML jezika, a dizajn definiran pomoću CSS-a. Glavne funkcionalnosti web stranica proizlaze iz funkcija definiranih pomoću JavaScript programskog jezika, uz malu pomoć PHP-a[5]. U nastavku slijedi objašnjenje JavaScript funkcija koje su korištene prilikom izrade web stranice.[6]

Da bi se mogao prikazivati video s kamere u realnom vremenu, potrebno je stvoriti html video element s id-om „videoElement“. Nakon toga u JavaScriptu potrebno je definirati varijablu *video* koja se spaja s navedenim video elementom.

```
var video = document.querySelector("#videoElement");
```

Upit za dopuštenje korištenja kamere definira se pomoću metode *navigator.mediaDevices.getUserMedia.* Ako se dopusti i ako se videozapis može uspješno pročitati, kreće prijenos videozapisa u realnom vremenu pomoću metode *srcObject.*

```
if (navigator.mediaDevices.getUserMedia) {
```

[5] Jedan od skriptnih programskih jezika na strani servera
[6] Uzrok razlikama u načinu pisanja programskog koda je pisanje na preporučen način pisanja za svaku tehnologiju

```
            navigator.mediaDevices
                  .getUserMedia({ video: true })
                  .then(function (stream) { video.srcObject = stream; })
```

Kada se videozapis ne može pročitati, u konzolu se ispisuje odgovarajuća poruka.

```
        .catch(function (err0r) {
            console.log("Something went wrong!");
        });
```

Za pregled videozapisa iz arhive, potrebno je stvoriti jedan novi video element i tri padajuća izbornika: jedan za čisti prikaz, jedan za videozapise s detektiranim licem i jedan za videozapise s detektiranim pokretom. S obzirom da je za to potrebno čitanje datoteka na strani web servera, za tu će se zadaću pobrinuti PHP skripta koju je potrebno staviti u svaku od tri mape arhive videozapisa: *normal_video, face_tracking* i *optical_flow.*

PHP skripta ima sljedeći oblik:

```
<?php

      . . .

?>
```

Unutar nje, definirana je varijabla *dir_path* kao putanja trenutne mape. Postavljen je uvjet ako je mapa s putanjom *dir_path* (koja je spremljena u varijablu *files)* uistinu mapa. U tom slučaju pokreće se while petlja koja se prekida kada više nema datoteka za čitanje. Unutar petlje postavljena su dva uvjeta: ako datoteka nije bez imena, čita se; ako datoteka sadrži „.mp4“ ispisuje se i čita u JavaScript metodi.

```
$dir_path = ".";
if(is_dir($dir_path))
{
  $files = opendir($dir_path);
  {
    if($files)
    {
      while(($file_name = readdir($files)) !== FALSE)
      {
        if($file_name != '.' && $file_name != '..')
        {
          $options = $options."<option>$file_name</option>";

        if (str_contains($file_name, '.mp4'))
```

```
    {
      echo $file_name."\n"; }
```

U JavaScriptu je potrebno definirati funkciju koja će čitati imena .mp4 datoteka koja pronađe prethodna skripta. Funkciji će se pridružiti ime *dataFetcher* kojoj je ulaz potrebna putanja mape iz koje je potrebno čitati. U njoj je potrebno definirati varijablu *data* koja će sadržavati imena videozapisa te listu *data_array* kojoj su elementi imena. Nakon toga, potrebno je otkinuti od liste zadnje ime videozapisa jer je ono prazno. Za kraj, potrebno je definirati da metoda prilikom njenog poziva vraća navedenu listu s imenima videozapisa.

```
async function dataFetcher(givenURL) {
    var URL = givenURL;
    const response = await fetch(URL);
    var data = await response.text();
    var dataArray = data.split("\n");
    dataArray.splice(-1);

    return dataArray;
}
```

S obzirom na to da postoje tri mape unutar arhive, za svaku je potrebno napraviti metodu koja će od liste videozapisa napraviti opcije unutar padajućeg izbornika. Metode su međusobno jednake, samo se razlikuju po putanjama i id-jevima – u nastavku će se objasniti samo jedna. Na početku metode *getNormalVideo* definira se putanja *URL* mape iz koje je potrebno dohvatiti imena videozapisa pomoću već definirane metode *dataFetcher*. Nakon dohvaćanja imena, pristupa se HTML elementu select preko id-a i sprema se u listu *select,* koja se odmah prazni, kako ne bi došlo do gomilanja naziva istih videozapisa. For petljom stvaraju se opcije odabira *element* kojima se pridružuje ime i putanja videozapisa i one se na kraju doda u listu *select.*

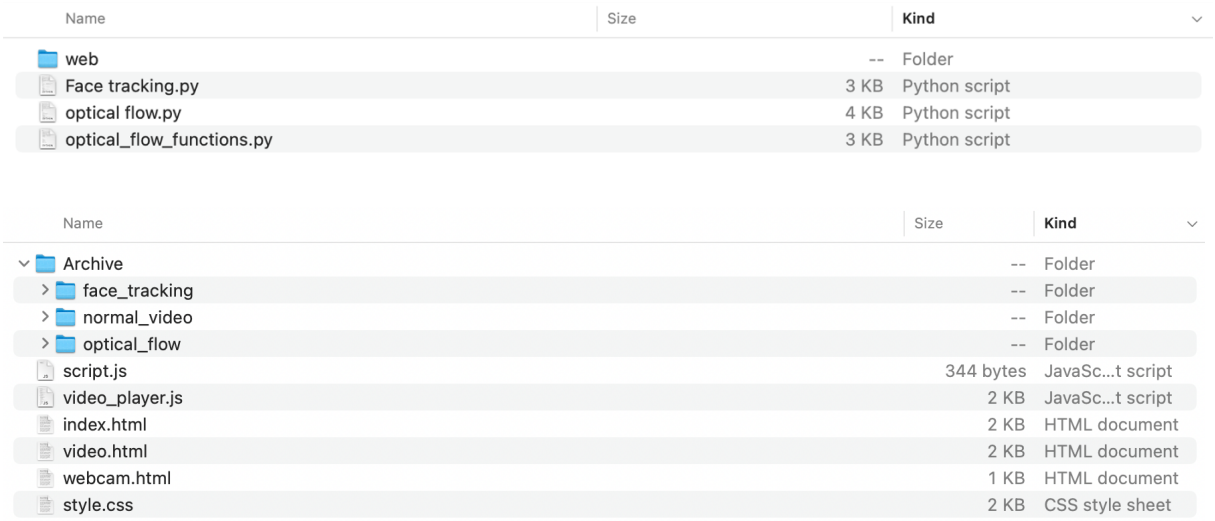
```
async function getNormalVideo() {
    var URL = "/Archive/normal_video/";
    var dataArray = await dataFetcher(URL);

    var select = document.getElementById("selectNormalVideo");
    select.innerHTML = "";

    for (var i = 0; i < dataArray.length; i++) {
        var option = dataArray[i];
        var element = document.createElement("option");
        element.textContent = option;
        element.value = URL + option;
        select.appendChild(element);
    }
```

```
}
```

Na kraju, potrebno je definirati metodu koja će od svih videozapisa video playeru dati onaj na koji se kliknulo. U ovoj se metodi varijabli *selected* pridružuje putanja videozapisa koja se kasnije pridružuje objektu *videoElement* kao izvor videozapisa. Nakon toga, on će se prikazati u prozoru na web stranici.

```
function selectedVideo(id) {
    var option = document.getElementById(id);
    var selected = option.value;

    var videoElement = document.getElementById("video_player");
    videoElement.src = selected;
}
```

Da bi se videozapisi mogli prikazivati unutar web sučelja, potrebno je sve potrebne .py datoteke spremiti u istu mapu u kojoj se nalazi i mapa *web* koja sadrži sve potrebne datoteke razvijene web aplikacije (**Slika 15**).



Slika 15: Prikaz glavne mape (gore) i prikaz mape web stranice (dolje)

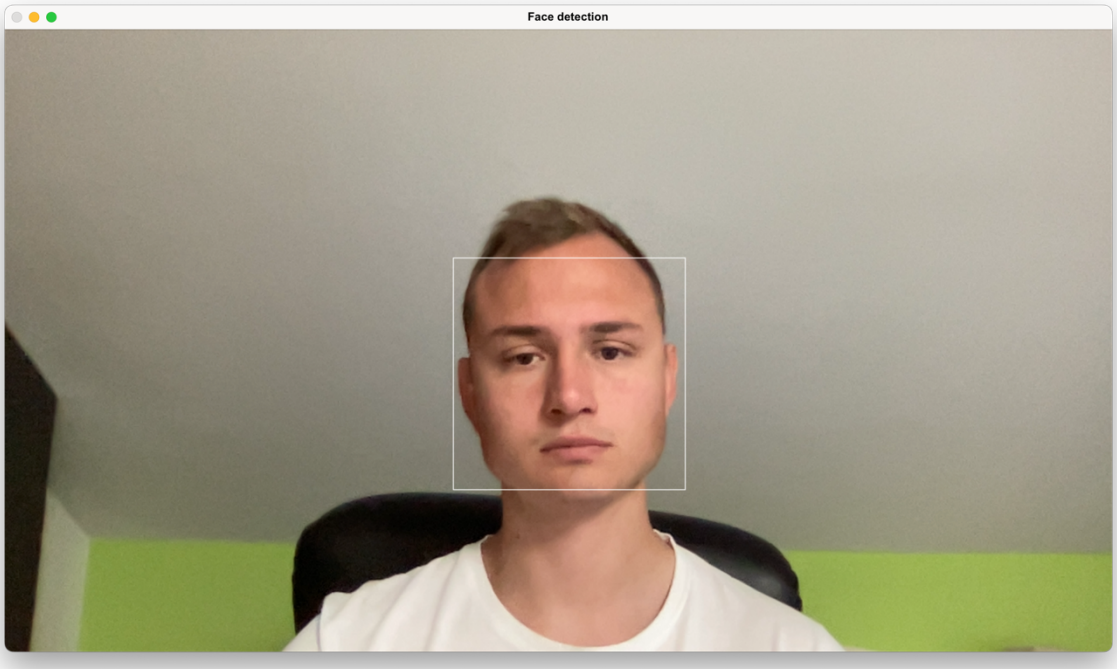
Prikazi web aplikacije i programskih aplikacija vidljivi su u poglavlju 4. EKSPERIMENTALNI REZULTATI.

4. EKSPERIMENTALNI REZULTATI

U ovom će poglavlju biti prikazani rezultati objašnjenih web i programskih aplikacija. Prvo će biti prikazani zasloni koje prikazuju programi, a na kraju će biti prikazana web aplikacija koja objedinjuje prikaz kamere u realnom vremenu s arhivom videozapisa nastalih uslijed detekcije lica ili pokreta.

4.1. Detekcija lica

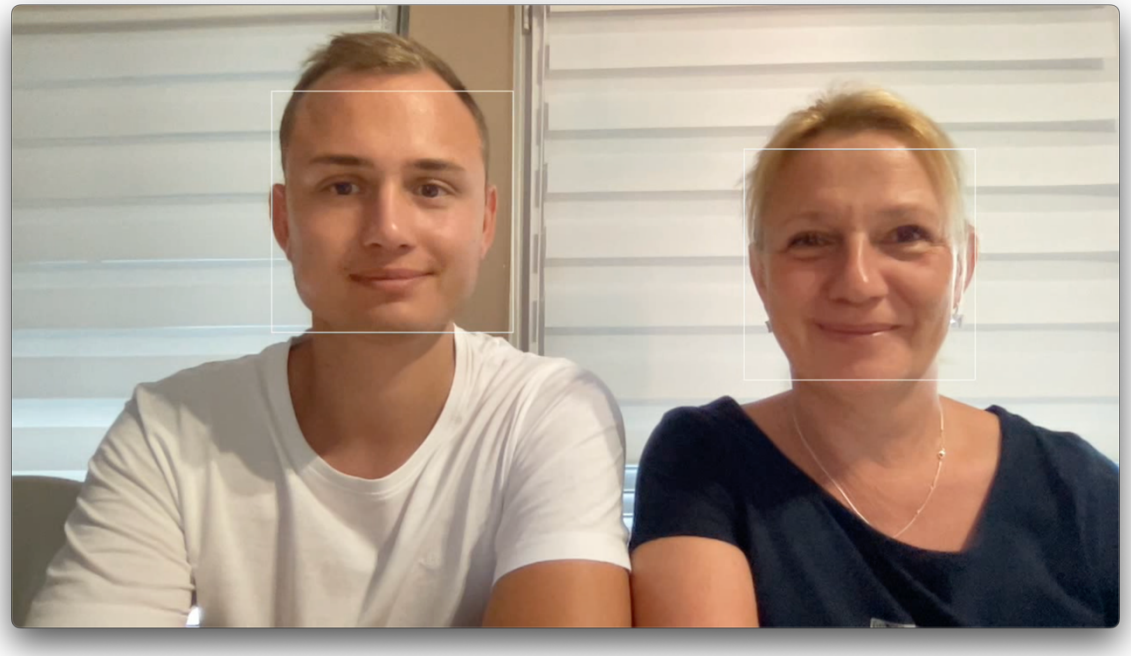
Pokretanjem programske aplikacije pokreće se detekcija lica i oko lica iscrtat će se bijeli kvadrat. U **Slika 16** prikazan je isječak iz prozora *Face detection* u kojem se prikazuje pogled u realnom vremenu s detektiranim licem.



Slika 16: Prikaz detektiranog lica u programu

Nakon što je lice prepoznato, kreće spremanje videozapisa i ako je njegovo trajanje duže od 6 sekundi spremi se u arhivu, a ako nije, videozapis se briše.

Na **Slika 17** priložen je prikaz dvaju detektiranih lica prilikom testiranja ispravnog rada programa uslijed pojave višestrukog broja lica.

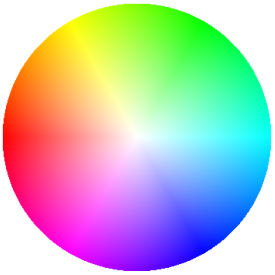


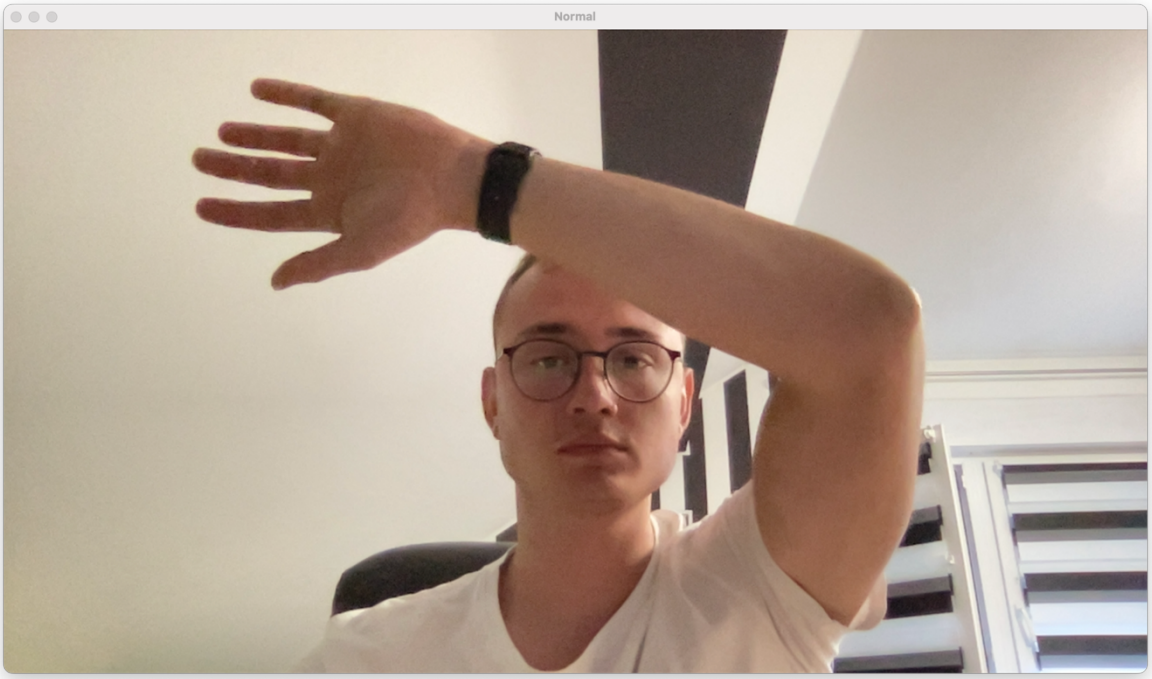
Slika 17: Prikaz dvaju detektiranih lica

4.2. Detekcija pokreta

U programu za detekciju pokreta definirana su tri prozora koja prikazuju: „normalan“ videozapis, videozapis gdje je pokret prikazan linijama i videozapis u kojem je pokret prikazan u HSV prostoru boja. Jednak prikaz kao u programu, vidljiv je i u videozapisu koji se može pregledati u sklopu web stranice. U **Slika 18** prikazan je normalan prikaz iz kamere, dok je u **Slika 19** prikazan pokret pomoću linija.

Prikaz u HSV prostoru boja ovisi o strani prema kojoj se objekt kreće. Ako se objekt kreće ulijevo dominira crvena boja, ako se objekt kreće udesno dominira plavo-zelene boja (ili laički rečeno - tirkizna), ako se objekt kreće prema gore dominira žuto-zelena boja i ako se objekt kreće prema dolje dominira ljubičasto-plava boja. Zaključeno je da se boje mijenjaju prema sljedećem dijagramu boja:

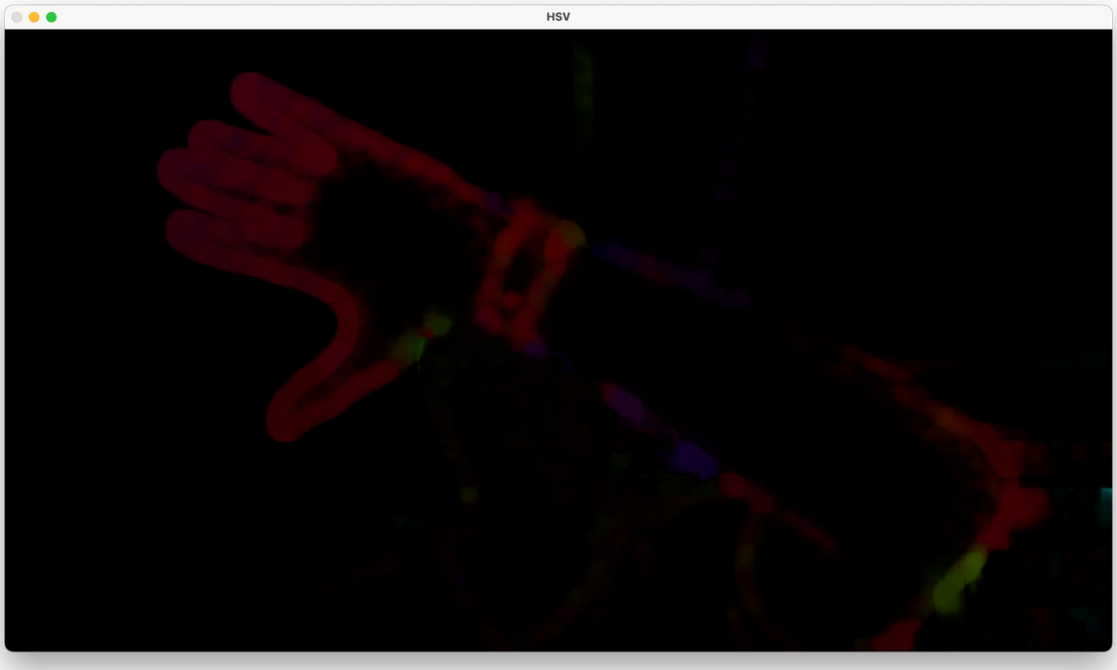




Slika 18: Normalan prikaz kretnje s kamere

Kao što se može vidjeti u slici ispod, duljina linije koja je iscrtana ekvivalent je brzini kretanja, a smjer linije poklapa se sa smjerom kretnje u određenoj točki.

Slika 19: Prikaz kretnje ruke pomoću linija



Slika 20: Pokret ruke ulijevo u HSV prostoru boja

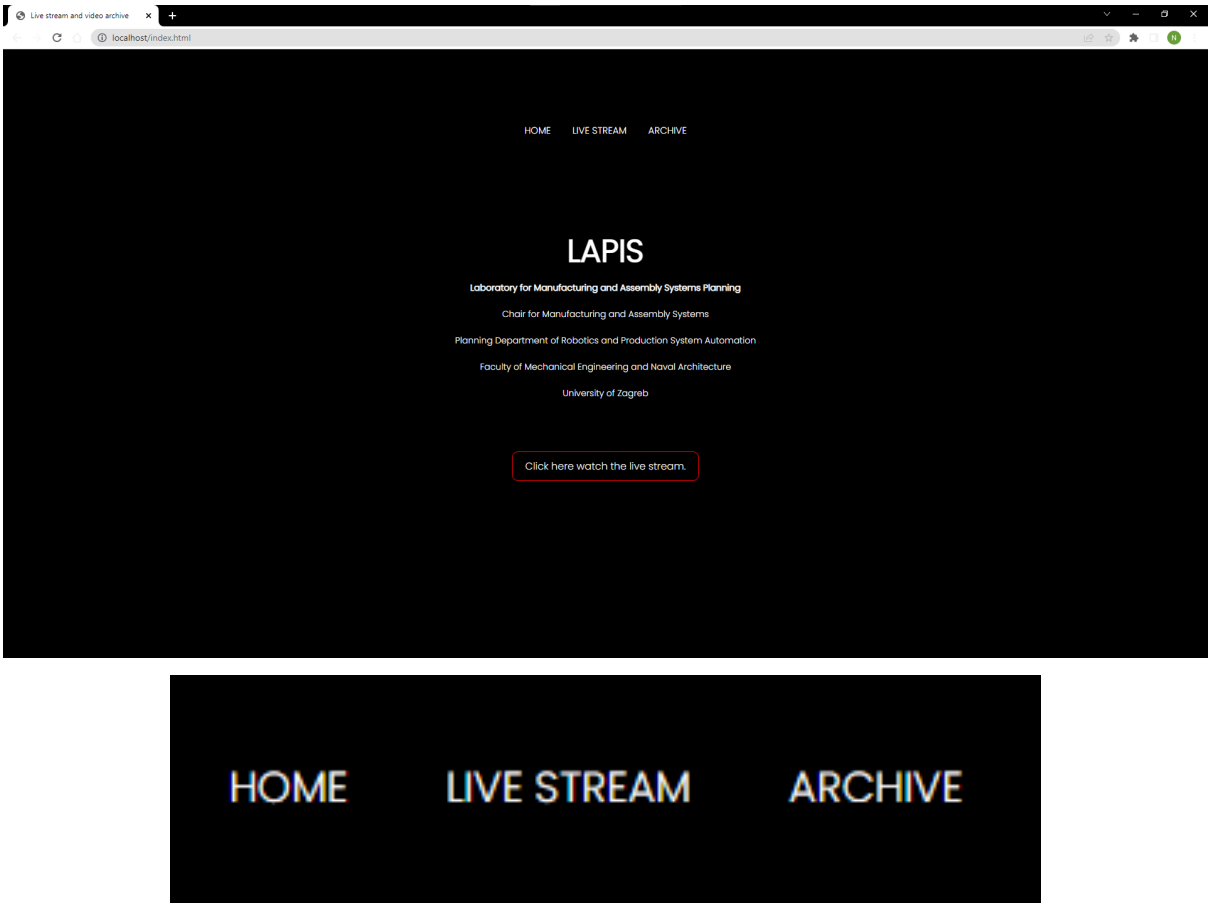
Kao što je već ranije spomenuto, boja koja dominira unutar prikaza može otkriti smjer kretanja. U slici ispod (**Slika 21**) prikazan je pokret prilikom testiranja realnih kretnji glave, lica i pogleda prema sustavu – prema bojama može se zaključiti da se crvena glava kretala ulijevo, a tirkizna udesno.

Slika 21: Prikaz pokreta više objekata

4.3. Web aplikacija

Web aplikacija smještena je na lokalnom sjedištu te je podijeljena u tri dijela: početni dio HOME, prikaz u stvarnom vremenu LIVE STREAM i arhiva videozapisa ARCHIVE. U nastavku je prikazana web aplikacija sa svim njezinim mogućnostima.

Početna stranica web aplikacije prikazana je u **Slika 22** i sadrži naziv fakulteta, katedre i zavoda u sklopu kojeg se nalazi Laboratorij za projektiranje izradbenih i montažnih sustava, odnosno LAPIS.

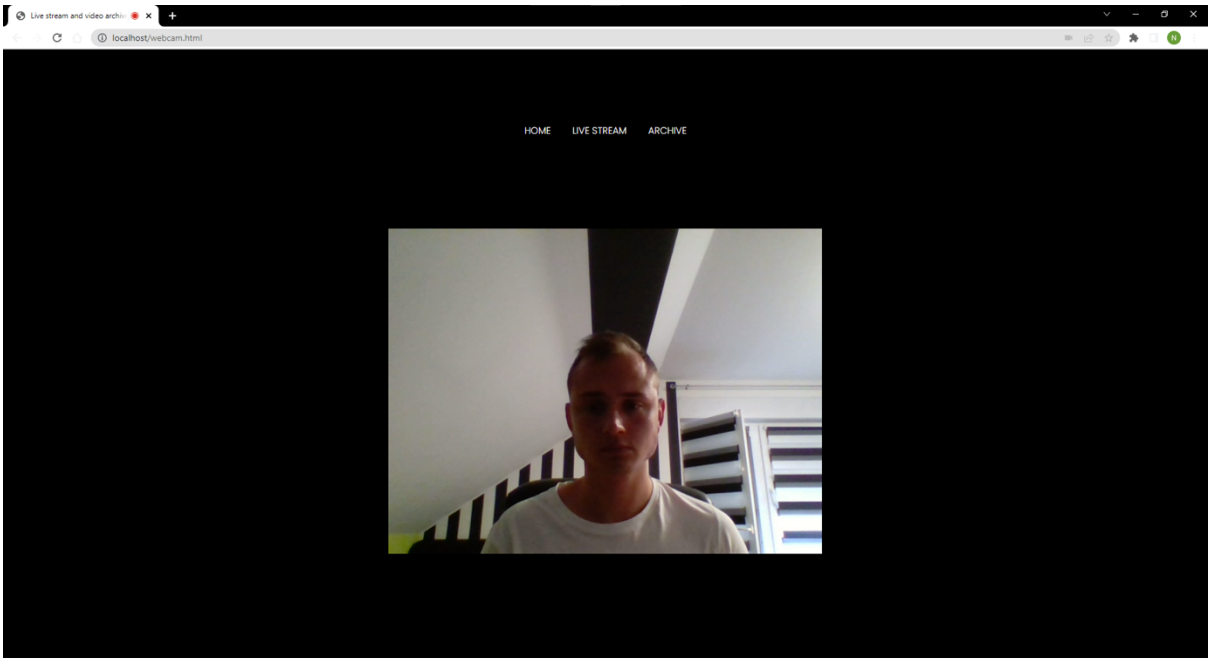


Slika 22: Početna stranica web aplikacije (gore) i navigacijske tipke (dolje)

Kao što se može vidjeti, sadrži tri tipke na vrhu i jednu veliku na dnu stranice. Gornje tri, navigacijske su tipke koje vode do početne strane, prikaza u stvarnom vremenu i arhive videozapisa. Svaki od navedena tri dijela sadrži ih, osim velike crno-crvene tipke koja vodi do prikaza u stvarnom vremenu. Prolaskom miša preko najveće tipke pokreće se intuitivna

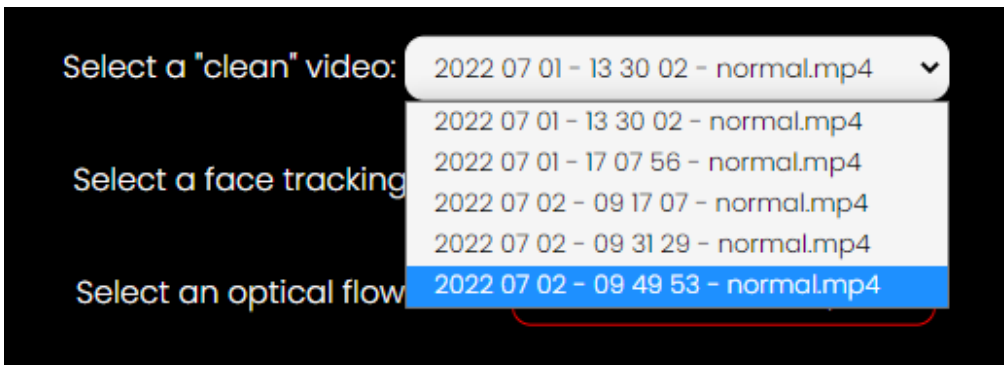
animacija – s ciljem da korisnik zna da je odabrana baš ta tipka – i tada ona postaje bijele boje s crnim slovima (**Slika 23**).

Pritiskom na navedenu tipku ili onu s imenom LIVE STREAM, otvara se stranica koja prikazuje prijenos slike u stvarnom vremenu. U tom je okviru prikazan „čisti“ prijenos, bez posebnih prikaza uslijed detekcije lica i pokreta, s ciljem postizanja estetike. Dakle, videozapisi detekcije lica i detekcije pokreta, smješteni su u arhivu.



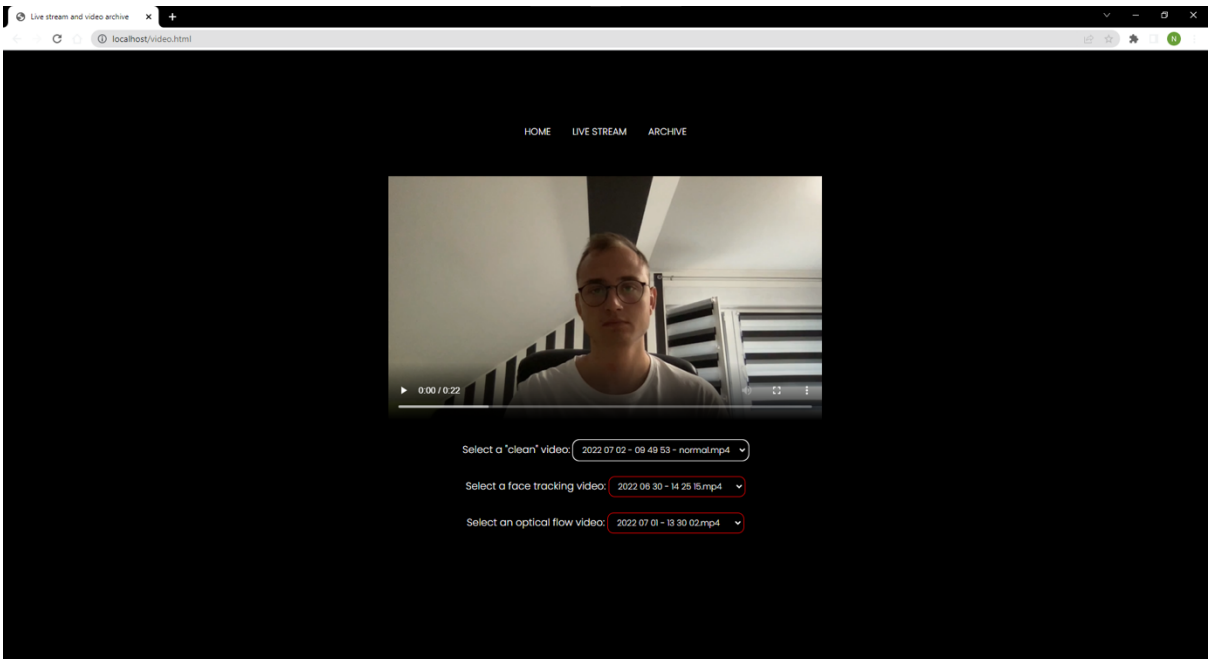
Slika 23: Prikaz prijenosa slike u stvarnom vremenu

Pritiskom na tipku ARCHIVE, korisniku je prikazana stranica arhive videozapisa. Na stranici se nalaze tri padajuća izbornika: jedan za odabir „čistog“ videozapisa, jedan za odabir videozapisa uslijed detekcije lica i jedan za odabir videozapisa uslijed detekcije pokreta. Pritiskom na padajući izbornik i odabirom videozapisa, rub mu postaje bijele boje (**Slika 24**).



Slika 24: Padajući izbornik s učitanim videozapisima na web sjedištu

Nakon odabira videozapisa, on se prikaže unutar web stranice arhive te se pokreće pritiskom na tipku „play“. Ako korisnik poželi pogledati neki drugi videozapis, samo je potrebno unutar odgovarajućeg padajućeg izbornika odabrati novi videozapis i on će se isto tako prikazati kao i onaj prvi.



Slika 25: Prikaz videozapisa unutar stranice arhive

5. ZAKLJUČAK

U današnje vrijeme kamere za snimanje imaju sve veću rezoluciju, stoga im i videozapisi zauzimaju više memorije. Obzirom na stanje, u protekle dvije godine, nedostatka, pa i poskupljenja hardvera, nužno je da datoteke spremljene na računalu zauzimaju što manje mjesta. Unutar ovog rada prikazana je sva potrebna tehnologija i izvedba sustava koji nadzire prostor u stvarnom vremenu i prikazuje videozapise iz arhive – koji su pokrenuti ulaskom ljudskog lica ili pojavom pokreta unutar vidnog polja kamere – čiji su algoritmi prepoznavanja temeljeni na strojnom učenju.

Kako bi potrošnja slobodnog mjesta bila optimalna, beskorisni (čitaj: prekratki) videozapisi automatski se brišu nakon analize trajanja. Također, uslijed sve većeg postotka programa koji se pokreću unutar web preglednika, razvijena je web aplikacija za nadzor kamere u stvarnom vremenu te pregled spremljenih videozapisa u arhivi. Sve izrađene aplikacije, razvijene su s ciljem maksimalne kompatibilnosti na različitim sustavima.

S obzirom na to da se Python aplikacije pokreću ručno, poboljšanje ovog rada bila bi integracija izrađenih aplikacija u samu web aplikaciju – koje se pokreću pritiskom na tipku. Također, kako bi slika bila „jasnija“ može se koristiti kvalitetnija kamera s većom rezolucijom – koja u konačnici zahtjeva mnogo jaču procesorsku jedinicu unutar računala – jer što više piksela slika sadrži, to duže treba vremena za njenu analizu.

LITERATURA

[1] IBM: *Computer Vision,* Pristupljeno 20. ožujka 2022., https://www.ibm.com/topics/computer-vision

[2] IBM: *Machine learning,* Pristupljeno 15. travnja 2022., https://www.ibm.com/cloud/learn/machine-learning

[3] Stipančić, T.: *Vizijski sustavi: Uvod u strojno učenje*, Podloge za predavanja, Fakultet strojarstva i brodogradnje, 2022.

[4] Levity: *Deep Learning vs. Machine Learning,* Pristupljeno 20. travnja 2022., https://levity.ai/blog/difference-machine-learning-deep-learning

[5] Kuldeep, S.: *An introduction to artificial neural network,* Jain University, India, 2016.

[6] Dalbelo Bašić, B., Čupić, M., Šnajder, J.: *Umjetne neuronske mreže,* Fakultet elektrotehnike i računarstva, Zagreb, 2008.

[7] Woodruff, A.: *What is a neuron?*, Queensland Brain Institute, Pristupljeno 23. lipnja 2022., https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron

[8] O'Shea, K., Nash, R.: *An introduction to convolutional neural networks,* Aberystwyth University, UK, 2015.

[9] Levity: *Deep Learning vs. Machine Learning,* Pristupljeno 20. travnja 2022., https://levity.ai/blog/difference-machine-learning-deep-learning

[10] Cybiant: *An introduction to computer vision,* Pristupljeno 25. travnja 2022. https://www.cybiant.com/resources/an-introduction-to-computer-vision

[11] Kuhlman, D.: *A Python Book: Beginning Python, Advanced Python and Python Exercises*, Poglavlje 1.1, 2012.

[12] Python Software Foundation: *Python Developers Guide,* Pristupljeno 20. ožujka 2022., https://devguide.python.org

[13] TIOBE: *TIOBE index,* Pristupljeno 03. travnja 2022., https://www.tiobe.com/tiobe-index

[14] Python Software Foundation: *PEP 20 -- The Zen of Python*, Pristupljeno 04. travnja 2022., https://www.python.org/dev/peps/pep-0020

[15] Python Software Foundation: *Is Python a good language for beginning programmers?* Pristupljeno 04. travnja 2022., https://docs.python.org/3/faq

[16] PyCharm: *Get Started,* Pristupljeno 05. travnja 2022., https://www.jetbrains.com/help/pycharm/quick-start-guide.html

[17] OpenCV: *About*, Pristupljeno 10. travnja 2022., https://opencv.org/about

[18] Dabhi, M. K., Pancholi, B. K.: *Face detection system based on Viola – Jones algorithm,* University Baroda, India, 2016.

[19] Hrga, M.: *Računalni vid,* Veleučilište u Šibeniku, 2018.

[20] OpenCV: *Optical Flow,* Pristupljeno 26. lipnja 2022., https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html

[21] GeeksForGeeks: *The Gunnar Farneback optical flow*, Pristupljeno 26. Lipnja 2022., https://www.geeksforgeeks.org/opencv-the-gunnar-farneback-optical-flow

[22] Mozilla developer network: *HTML*, Pristupljeno 27. Lipnja 2022., https://developer.mozilla.org/en-US/docs/Web/HTML

[23] Mozilla developer network: *JavaScript*, Pristupljeno 27. Lipnja 2022., https://developer.mozilla.org/en-US/docs/Web/JavaScript

[24] Mozilla developer network: *CSS*, Pristupljeno 27. Lipnja 2022., https://developer.mozilla.org/en-US/docs/Web/CSS

PRILOG

- Python programski kod
    - Face tracking.py
    - optical flow.py
    - optical_flow_functions.py
- JavaScript i PHP programski kod
    - index.php
    - script.js
    - video_player.js

Face tracking.py

```
import cv2, os, time

# --------------------------- Initial settings --------------------------- #

cap = cv2.VideoCapture(0)
casc_path =
os.path.dirname(cv2.__file__)+"/data/haarcascade_frontalface_default.xml"
face_cascade = cv2.CascadeClassifier(casc_path)

wanted_fps = 10
rectangle_width = 1

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
file_type = '.mp4'
file_name = time.strftime("%Y %m %d - %H %M %S" + file_type)

fourcc = cv2.VideoWriter_fourcc(*'AVC1')
writer = cv2.VideoWriter(file_name, fourcc, wanted_fps, (width, height))

start_timer = time.time()
counter = 0
save_counter = 0

while True:

  fps_start = time.time()
  ret, frame = cap.read()

  faces = face_cascade.detectMultiScale(
    frame,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    flags=0
    )

  for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 255), rectangle_width)

  cv2.imshow('Face tracking', frame)

  len_faces = len(faces)

  if len_faces != 0:
    start_timer = time.time()
    end_timer = time.time()
    counter = 0
    file_name = time.strftime("%Y %m %d - %H %M %S" + file_type)
```

```
    if save_counter == 0:
      save_counter = 1
      writer = cv2.VideoWriter(file_name, fourcc, wanted_fps, (width, height))
      duration_start = time.time()
      file_name_check = file_name
    writer.write(frame)

  if len_faces == 0:
    end_timer = time.time()

  elapsed_time = round(end_timer - start_timer, 2)

  if elapsed_time >= 3 and counter == 0:
    print('stopped')
    counter = 1
    save_counter = 0
    writer.release()
    duration_end = time.time()
    duration = round((duration_end - duration_start), 2)
    if duration < 6:
      os.remove(file_name_check)
      print('File ' + file_name_check + ' is removed. Duration: ' + str(duration))
    if duration >= 6:
      new_path = "Archive/face_tracking/" + file_name_check
      os.replace(file_name_check, new_path)


  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

  print("FPS: ", 1.0 / (time.time() - fps_start))

cap.release()
writer.release()
cv2.destroyAllWindows()
```

optical flow.py

```
import cv2, time, os
from optical_flow_functions import draw_flow, draw_hsv


# ---------------------------- Initial settings ---------------------------- #

cap = cv2.VideoCapture(0)

ret, frame = cap.read()
prev_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

wanted_fps = 10

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
file_type = '.mp4'
file_name = time.strftime("%Y %m %d - %H %M %S" + file_type)
file_name_norm = time.strftime("%Y %m %d - %H %M %S" + " - normal" + file_type)

fourcc = cv2.VideoWriter_fourcc(*'AVC1')

writer = cv2.VideoWriter(file_name, fourcc, wanted_fps, (width, height))
writer_norm = cv2.VideoWriter(file_name_norm, fourcc, wanted_fps, (width, height))

start_timer = time.time()
counter = 0
save_counter = 0


# ---------------------------- Optical flow parameters ---------------------------- #
pyr_scale = 0.5
levels = 3
winsize = 15
iterations = 1
poly_n = 7
poly_sigma = 1.5
flags = 0
while True:

    fps_start = time.time()

    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, pyr_scale, levels,
winsize, iterations, poly_n, poly_sigma, flags)

    prev_gray = gray
```

```
    img_bgr, fx_mean, fy_mean = draw_flow(gray, flow)
    img_hsv = draw_hsv(flow)

    cv2.imshow('Normal', frame)
    cv2.imshow('Flow', img_bgr)
    cv2.imshow('HSV', img_hsv)

    if fx_mean >= 5 or fy_mean >= 5:
        start_timer = time.time()
        end_timer = time.time()
        counter = 0
        file_name = time.strftime("%Y %m %d - %H %M %S" + file_type)
        file_name_norm = time.strftime("%Y %m %d - %H %M %S" + " - normal" +
file_type)
        writer.write(img_hsv)
        writer_norm.write(frame)
        if save_counter == 0:
            save_counter = 1
            writer = cv2.VideoWriter(file_name, fourcc, wanted_fps, (width,
height))
            writer_norm = cv2.VideoWriter(file_name_norm, fourcc, wanted_fps,
(width, height))
            duration_start = time.time()
            file_name_check = file_name
            file_name_check_norm = file_name_norm

    if fx_mean < 5 and fy_mean < 5:
        end_timer = time.time()

    elapsed_time = round(end_timer - start_timer, 2)

    if elapsed_time >= 3 and counter == 0:
        print('stopped')
        counter = 1
        save_counter = 0
        writer.release()
        writer_norm.release()
        duration_end = time.time()
        duration = round((duration_end - duration_start), 2)
        if duration < 6:
            os.remove(file_name_check)
            os.remove(file_name_check_norm)
            print('File ' + file_name_check + ' is removed. Duration: ' +
str(duration))
            print('File ' + file_name_check_norm + ' is removed. Duration: ' +
str(duration))
        if duration >= 6:
            new_path = "Archive/optical_flow/" + file_name_check
            os.replace(file_name_check, new_path)
            new_path_norm = "Archive/normal_video/" + file_name_check_norm
            os.replace(file_name_check_norm, new_path_norm)
```

```
    print("FPS: ", 1.0 / (time.time() - fps_start))
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    print("FPS: ", 1.0 / (time.time() - fps_start))

cap.release()
writer.release()
writer_norm.release()
cv2.destroyAllWindows()
```

optical_flow_functions.py

```
import numpy as np
import cv2, time


def draw_flow(img, flow, step=20):
    '''
    Draws flow lines.
    '''

    h, w = img.shape[:2]
    y, x = np.mgrid[step/2:h:step, step/2:w:step].reshape(2,-1).astype(int)
    fx, fy = flow[y, x].T

    lines = np.vstack([x, y, x-fx, y-fy]).T.reshape(-1, 2, 2)
    lines = np.int32(lines + 0.5)

    img_bgr = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
    cv2.polylines(img_bgr, lines, 0, (145, 96, 255))

    for (x1, y1), (_x2, _y2) in lines:
        cv2.circle(img_bgr, (x1, y1), 1, (145, 255, 255))

    N = 20
    fx_sort = np.int32(np.sort(abs(fx)))
    fx_last_elements = fx_sort[-N:]
    fx_mean = sum(fx_last_elements) / N
    fx_mean = round(fx_mean, 1)

    fy_sort = np.int32(np.sort(abs(fy)))
    fy_last_elements = fy_sort[-N:]
    fy_mean = sum(fy_last_elements) / N
    fy_mean = round(fy_mean, 2)

    return img_bgr, fx_mean, fy_mean



def draw_hsv(flow):
    '''
    Draws flow in color.
    '''

    h, w = flow.shape[:2]
    fx, fy = flow[:,:,0], flow[:,:,1]

    ang = np.arctan2(fy, fx) + np.pi
    length = np.sqrt(fx*fx+fy*fy)
```

```
    hue = ang*(180/np.pi/2)
    sat = 255
    val = np.minimum(length*4, 255)

    hsv = np.zeros((h, w, 3), np.uint8)
    hsv[..., 0] = hue
    hsv[..., 1] = sat
    hsv[..., 2] = val
    bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

    return bgr
```

index.php

```php
<?php

$dir_path = ".";
$options = "";
if(is_dir($dir_path))
{
  $files = opendir($dir_path);
  {
    if($files)
    {
      while(($file_name = readdir($files)) !== FALSE)
      {
        if($file_name != '.' && $file_name != '..')
        {
          $options = $options."<option>$file_name</option>";

          if (str_contains($file_name, '.mp4'))
          {
            echo $file_name."\n";
          }
        }
      }
    }
  }
}

?>
```

script.js

```javascript
var video = document.querySelector("#videoElement");

if (navigator.mediaDevices.getUserMedia) {
    navigator.mediaDevices
        .getUserMedia({ video: true })
        .then(function (stream) {
            video.srcObject = stream;
        })
        .catch(function (err0r) {
            console.log("Something went wrong!");
        });
}
```

video_player.js

```
async function dataFetcher(givenURL) {
    var URL = givenURL;

    const response = await fetch(URL);
    var data = await response.text();

    var dataArray = data.split("\n");
    dataArray.splice(-1);

    return dataArray;
}

async function getNormalVideo() {
    var URL = "/Archive/normal_video/";
    var dataArray = await dataFetcher(URL);

    console.log(dataArray);

    var select = document.getElementById("selectNormalVideo");
    select.innerHTML = "";

    for (var i = 0; i < dataArray.length; i++) {
        var option = dataArray[i];
        var element = document.createElement("option");
        element.textContent = option;
        element.value = URL + option;
        select.appendChild(element);
    }
}

async function getFaceVideo() {
    var URL = "/Archive/face_tracking/";
    var dataArray = await dataFetcher(URL);

    console.log(dataArray);

    var select = document.getElementById("selectFaceVideo");
    select.innerHTML = "";

    for (var i = 0; i < dataArray.length; i++) {
        var option = dataArray[i];
        var element = document.createElement("option");
        element.textContent = option;
        element.value = URL + option;
        select.appendChild(element);
    }
}
```

```
async function getOpticalVideo() {
    var URL = "/Archive/optical_flow/";
    var dataArray = await dataFetcher(URL);

    console.log(dataArray);

    var select = document.getElementById("selectOpticalVideo");
    select.innerHTML = "";

    for (var i = 0; i < dataArray.length; i++) {
        var option = dataArray[i];
        var element = document.createElement("option");
        element.textContent = option;
        element.value = URL + option;
        select.appendChild(element);
    }
}

function selectedVideo(id) {
    var option = document.getElementById(id);
    var selected = option.value;
    console.log(selected);

    var videoElement = document.getElementById("video_player");
    videoElement.src = selected;
}

function init() {
    getNormalVideo();
    getFaceVideo();
    getOpticalVideo();
}

init();
```