# Primjena umjetne inteligencije za korekciju položaja oka u video komunikaciji

**Črnčec, Patrik**

**Master's thesis / Diplomski rad**

**2022**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:211:318904

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ORGANIZACIJE I INFORMATIKE**
**VARAŽDIN**

**Patrik Črnčec**

# PRIMJENA UMJETNE INTELIGENCIJE ZA KOREKCIJU POLOŽAJA OKA U VIDEO KOMUNIKACIJI

**DIPLOMSKI RAD**

**Varaždin, 2022.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Patrik Črnčec**

**Matični broj: 45770/17–R**

**Studij: Baze podataka i baze znanja**

# PRIMJENA UMJETNE INTELIGENCIJE ZA KOREKCIJU POLOŽAJA OKA U VIDEO KOMUNIKACIJI

**DIPLOMSKI RAD**

**Mentor:**

Doc. dr. sc. Boris Tomaš

**Varaždin, lipanj 2022.**

*Patrik Črnčec*

**Izjava o izvornosti**

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

---

**Sažetak**

Rad se sastoji od dva dijela: teorijskog i praktičnog. U teorijskom dijelu su proanalizirana i objašnjena dosadašnja rješenja, dana je teorijska podloga za tehnologiju i metode korištene u radu te je programsko rješenje (proizvod) opisano i dokumentirano. Praktični dio rada se sastoji od samog programskog proizvoda koji zapravo i predstavlja glavni fokus ovog rada. Glavna zadana funkcionalnost programskog proizvoda je mogućnost korekcije položaja očiju prilikom video komunikacije koja će biti implementirana uz pomoć umjetne inteligencije i strojnog učenja. S obzirom na to da često kod video poziva sudionici ne gledaju direktno u video kameru koja se u pravilu nalazi iznad ekrana, već gledaju sebe ili sugovornika, javlja se problem neprirodne komunikacije u kojoj nema međusobnog kontakta očima. Zbog toga, cilj praktičnog dijela je uz pomoć umjetne inteligencije napraviti obradu videa u stvarnom vremenu kako bi se riješio problem nedostatka kontakta očima prilikom video komunikacije. Također, prilikom obrade videa je potrebno riješiti izazove poput zadržavanja zadovoljavajućih performansi i konzistencije, sprječavanje pojava neprirodnih efekata i artefakta te gladak i prirodan prijelaz između mogućih stanja korekcije položaja očiju.

**Ključne riječi**: umjetna inteligencija; strojno učenje; prividna stvarnost; obrada videa; multimedija; računalni vid

# Sadržaj

# 1. Uvod

Video komunikacija posljednjih godina postaje sve prisutnija zbog svojih prednosti kao što su fleksibilnost, nepostojanje troškova prijevoza, mogućnost upotrebe kolaboracijskih alata i slično. Međutim, video komunikacija zbog svojih mana nije idealan oblik komunikacije, a jedna takva mana u odnosu na prirodnu komunikaciju je i nedostatak kontakta očima. Taj problem se javlja zbog toga što sudionici video poziva najčešće ne gledaju direktno u kameru koja se obično nalazi iznad ili ispod ekrana, već u neki drugi dio ekrana, npr. gledaju sugovornika ili sebe. U slučaju da bi sudionici razgovora gledali direktno u kameru, takav problem ne bi postojao, međutim, kako danas još nisu komercijalno razvijene kamere koje se nalaze ispod samog ekrana uređaja, problem nedostatka kontakta očiju u video komunikaciji je moguće jedino riješiti na neki alternativni način. Ovaj rad upravo nastoji riješiti takvu problematiku, a cilj je da se problem riješi uz pomoć primjene umjetne inteligencije, odnosno strojnog učenja.

Ovaj se diplomski rad sastoji od osam poglavlja, gdje su najprije postavljene hipoteze i ciljevi ovog rada. U idućem poglavlju su opisani neki od osnovnih pojmova koji se često pojavljuju u istraživanjima o sustavima za korekciju položaja očiju. Nakon toga je analizirana relevantna literatura o najnovijim istraživanjima u području korekcije položaja očiju u video komunikaciji pomoću umjetne inteligencije te je za svaku od njih opisana korištena metoda, njezine prednosti i mane te rezultati koji su bili postignuti takvom metodom. Također su analizirane i prikazane aplikacije koje imaju mogućnost korekcije položaja očiju. Zatim slijedi pregled i opis korištenih tehnologija i algoritama na temelju kojih je napravljen model strojnog učenja koji je nakon toga predstavljen te objašnjen i dokumentiran. Na kraju tog poglavlja slijedi analiza postignutih rezultata, evaluacija implementiranog modela te prikaz primjene sustava. Nadalje, u sljedećem poglavlju se razmatra budući rad o temi korekcije položaja oka u video komunikaciji, dok je na kraju rada dani konačan zaključak.

# 2. Hipoteze i ciljevi

Hipoteza ovog rada je:

- **H1**: Primjenom algoritama umjetne inteligencije moguće je u stvarnom vremenu korigirati položaj očiju u video komunikaciji kako bi se na taj način prividno ostvario kontakt očima.

Ciljevi ovog rada su:

- **C1**: Razviti model strojnog učenja koji će omogućiti korekciju položaja očiju u stvarnom vremenu
- **C2**: Model mora prepoznati situacije u kojima ne mora korigirati položaj očiju korisnika te prijelaz u takvo stanje mora izgledati prirodno
- **C3**: Primjena razvijenog modela strojnog učenja u video komunikaciji ne smije znatno narušiti performanse komunikacije
- **C4**: Potrebno je omogućiti primjenu razvijenog modela u aplikacijama za video komunikaciju

# 3. Opis osnovnih pojmova

U nastavku će se ukratko objasniti nekoliko važnih osnovnih pojmova koji se često vežu uz pojam umjetne inteligencije, a koji se često primjenjuju i u metodama za korekciju položaja očiju.

## 3.1. Strojno učenje

Strojno učenje je skupina algoritama iz područja umjetne inteligencije čija je glavna zadaća generalizacija, a nju ostvaruje uz pomoć učenja iz danih podataka. Ono se uspješno primjenjuje u rješavanju problema iz raznoraznih područja poput primjerice klasifikacije, regresije, grupiranja, detekcije anomalija, računalnog vida i procesiranja prirodnog jezika. S obzirom na to da se radi o velikom skupu algoritama, strojno učenje se najčešće dijeli na nadzirano učenje, nenadzirano učenje i potporno učenje. Svaka takva grupa algoritama ima svoje područje primjene te svoj princip rada [1].

Zadaci strojnog učenja se opisuju kroz način procesiranja neke instance podatka koji predstavlja primjer iz domene problema kojeg je potrebno riješiti, a sastoji se od svojih obilježja koji su najčešće reprezentirani uz pomoć vektora. Primjerice, obilježja mogu biti cijene nekretnina, vrijeme trajanja filma ili pak recimo vrijednosti pixela na slici. Prilikom primjene algoritma strojnog učenja potrebno je napraviti njegovu evaluaciju kako bi se znalo koja je točnost treniranog modela. Točnost modela pobliže opisuje udio podataka za koje je model točno predvidio izlaz [1].

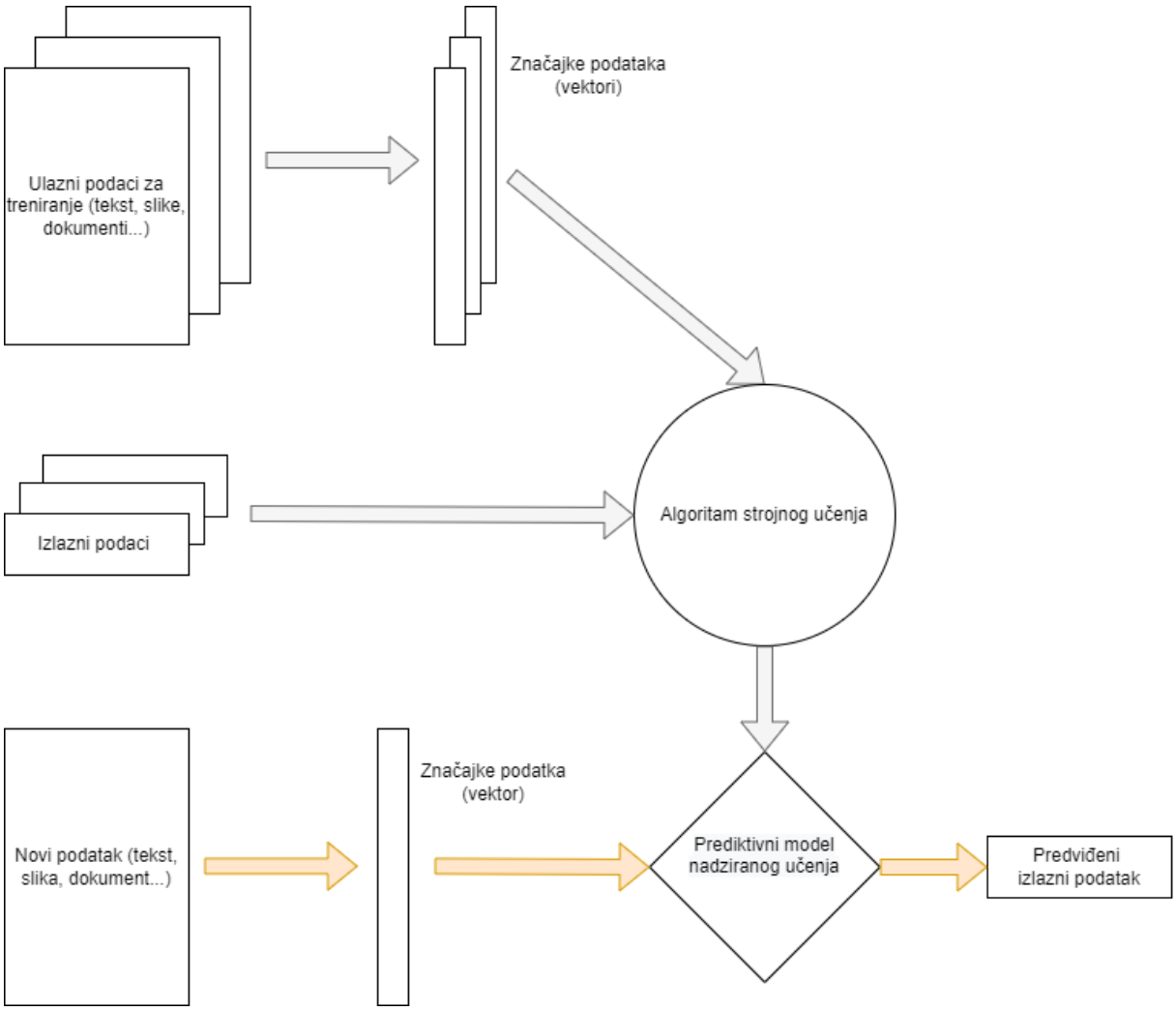
Kako se za korekciju položaja očiju uz pomoć umjetne inteligencije koristi isključivo metode, odnosno algoritmi iz područja nadziranog učenja, u narednom potpoglavlju su opisani osnovni principi rada takvih algoritama.

### 3.1.1. Nadzirano učenje

Nadzirano učenje je područje strojnog učenja čiji algoritmi imaju cilj naučiti preslikavanje ulaznog podatka u željeni izlazni podatak. Zbog toga, u nadziranom učenju svaka instanca podatka sadrži ulazni podatak $x$ i izlazni podatak $y$. Konkretnije, model ima zadaću prilikom svojeg učenja predvidjeti izlazni podatak na temelju danih značajki ulaznog podatka. S obzirom na to da model zna koje su stvarne vrijednosti danog izlaznog podatka te koje su predviđene vrijednosti izlaznog podatka, na temelju njihove razlike se određuje greška modela. Postoje mnogi načini za izračun greške modela poput primjerice srednje kvadratne pogreške, no neovisno o metodi izračuna greške, cilj modela nadziranog učenja je da se ona minimizira kako bi na taj način on što bolje generalizirao svoja predviđanja [1].

Učenje značajki podataka se provodi uz pomoć skupa podataka za treniranje. Nakon treniranja se radi evaluacija nad posebnim podacima za testiranje kako bi se na taj način provjerilo da li je trenirani model dovoljno dobro naučio generalizirati, što ovisno o zadanom problemu uključuje klasifikaciju ili regresiju. Takvi skupovi podataka najčešće nastaju prilikom faze pre-

procesiranja podataka gdje se čitav skup podataka razdvaja na zasebne skupove podataka za treniranje i testiranje. Zadaća klasifikacije je predviđanje zadanih klasa koje su reprezentirane uz pomoć diskretnih vrijednosti, a primjer može biti klasifikacija spola osobe (muško ili žensko). Za razliku od klasifikacije, zadaća regresije je predviđanje realnog broja kao što je to na primjer cijena nekretnine. Na slici je prikazan osnovni princip prikaz rada nadziranog učenja [1].




Slika 1: Princip rada nadziranog učenja (Prema: [2])

## 3.2. Neuronske mreže

Neuronske mreže su skup modela nadziranog učenja čije su strukture i način rada inspirirane neuronskim mrežama koje se nalaze u ljudskim mozgovima, pa se zbog toga ponekad još nazivaju i umjetne neuronske mreže. Naime, osnovna jedinica ljudskog mozga je neuron koji su međusobno povezani u neuronsku mrežu. Neuroni su optimizirani da primaju informacije od drugih neurona, na jedinstven način procesiraju informacije te rezultat šalju drugim ćelijama. Ovisno o učestalosti korištenja, svaka nadolazeća konekcija u neuron se dinamički pojačava ili oslabljuje čime se ostvaruje sposobnost ljudi da uče nove koncepte. Jačina svake konekcije određuje važnost utjecaja ulaza na rezultat izlaza iz neurona. Takav princip rada neuronske

mreže u ljudskome mozgu se može translatirati i reprezentirati u model neuronske mreže na računalu [3].

Princip rada umjetnih neurona je da primaju neki broj ulaza $x_1, x_2, \ldots, x_n$ te je svaki ulaz pomnožen s određenom težinom $w_1, w_2, \ldots, w_n$ te se na kraju umnošci zbroje. Osim toga, često rezultat sadrži i dodatni parametar - konstantu $b$ koja pomaže funkciji u prilagođavanju rezultata. Prema tome, matematička funkcija pojedinog neurona se formalno može zapisati kao:

$$y = f(\Sigma_i^n w_i \cdot x_i)$$

ili kao

$$y = f(\Sigma_i^n w_i \cdot x_i + b)$$

, gdje je $f$ aktivacijska/transformacijska funkcija koja će se kasnije objasniti. No, obično se zbog bržeg izračuna umnožaka ulazni i izlazni podaci zapisuju u vektorskome obliku. Na primjer ulazni podaci se zapisuju u vektor na sljedeći način:

$$x = [x_1 x_2 \ldots x_n]$$

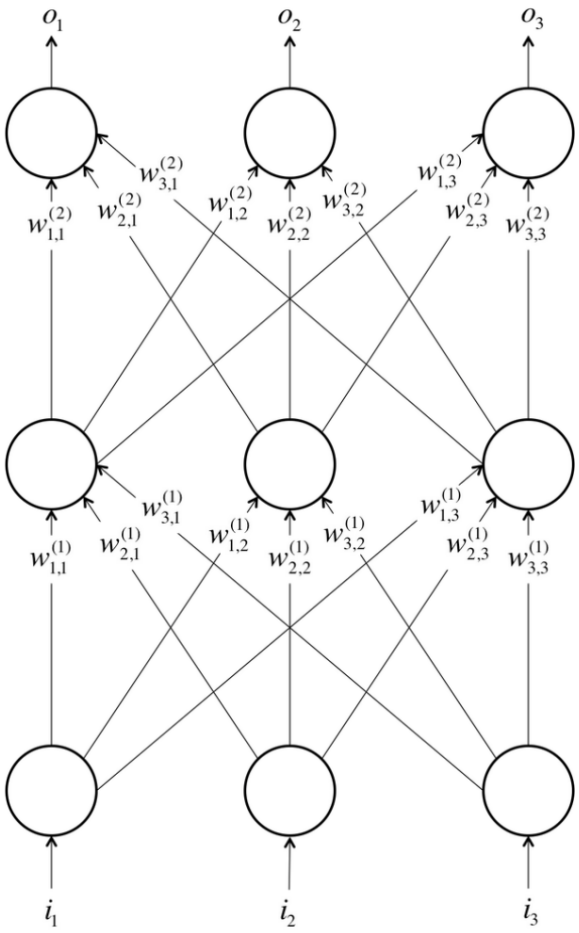
pa se onda izlazni neuroni izračunavaju pomoću funkcije

$$y = f(x \cdot w + b)$$

[3].

Za rješavanje složenijih problema se ne koristi samo jedan neuron već puno veća skupina neurona povezanih u jednu cjelinu koja se naziva neuronska mreža. Ljudski se mozak također sastoji od više neurona koji su strukturirani u slojeve. Primjerice, cerebralni korteks – struktura koja je odgovorna za većinu ljudske inteligencije se sastoji od šest slojeva. Za svrhu raspoznavanja vizualnih objekata, mreža funkcionira na način da prvi sloj prima podatke od očiju koji se zatim na svakom sloju slijedno procesiraju te na zadnjem, šestom sloju se omogućuje raspoznavanje objekata. Taj princip povezanosti neurona u neuronsku mrežu koja je sačinjena od slojeva se može implementirati i na računalu čime se dobivaju umjetne neuronske mreže (dalje će se koristiti samo pojam neuronske mreže). Neuronska mreža se sastoji od ulaznog, skrivenog i izlaznog sloja, s time da je čest slučaj da umjesto jednog skrivenog sloja postoje više skrivenih slojeva koji mogu biti strukturirani u različite arhitekture te je onda u tom slučaju riječ u dubokim neuronskim mrežama i dubokom (strojnom) učenju [3].

Na sljedećoj slici se nalazi prikaz strukture jednostavne neuronske mreže s jednim skrivenim slojem i tri neurona na svakom sloju. Vidljivo je da je izlaz svakog neurona na pojedinom sloju povezan sa svim ulazima neurona na sljedećem sloju. U nastavku će se objasniti dva glavna koraka kod neuronskih mreža: širenje unaprijed i širenje unatrag.




Slika 2: Prikaz jednostavne strukture neuronske mreže (Izvor: [3])

### 3.2.1. Širenje unaprijed

Neuroni se sastoje od svojih težina koji se množe s izlazima neurona iz prethodnog sloja ili pak s ulaznim podacima ako se radi o ulaznom sloju. Na početku, težine neurona su najčešće inicijalizirane kao nasumični brojevi iz neke definirane distribucije, npr. normalne distribucije. Za tu operaciju se koristi ranije opisana formula:

$$y = f(x \cdot w + b)$$

Nakon množenja se primjenjuje aktivacijska, odnosno transformacijska funkcija, kako bi na taj način postigla nelinearnost koja omogućuje modelu veću prilagodbu podacima i samim time bolju preciznost. Postoje razne aktivacijske funkcije poput primjerice sigmoidne funkcije:

$$f(z) = \frac{1}{1 + e^{-z}}$$

tangens hiperbolni:

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

ReLU (eng. *rectified linear unit*):

$$f(z) = max(0, z)$$

softmax:

$$f(z) = \frac{e^{z_i}}{\Sigma_j e^{z_j}}$$

Odabir aktivacijske funkcije ovisi o sloju gdje se neuron nalazi, ali i o zadanom cilju koji se želi postići. Na primjer, softmax funkcija se najčešće koristi u izlaznom sloju i to samo ako je cilj klasifikacija podataka jer rezultat takve funkcije je distribucija vjerojatnosti svih klasa za pojedini ulazni podatak. Klasa koja ima najveću vjerojatnost se smatra predviđenom klasom za dani ulazni podatak. Sigmoidna funkcija kao izlaz daje vrijednost između 0 i 1. U slučaju da je vrijednost *z* jako mala, onda će rezultat sigmoidne funkcije biti blizu 0, dok će u suprotnome slučaju biti blizu 1. Slično vrijedi i za tangens hiperbolni, samo što je kod te aktivacijske funkcije domena rezultata između -1 i 1. ReLU aktivacijska funkcija se često koristi kod skrivenih slojeva u zadacima vezanim uz računalni vid te ona sve negativne rezultate preslikava u nulu. Nakon izračuna rezultata izlaznog sloja, odnosno izračuna predviđene vrijednosti, određuje se greška predviđanja uz pomoć ranije definirane funkcije greške (npr. srednja kvadratna pogreška), čime se određuje greška između predviđene i stvarne izlazne vrijednosti. Izračunom greške se završava korak širenja unaprijed [3].

### 3.2.2. Širenje unatrag

Cilj koraka širenje unatrag je modificirati težine neurona kako bi se uz pomoć novodobivenih težina minimalizirala greška modela, odnosno povećala preciznost predviđanja izlaznih podataka. Kako bi se težine mogle pravilno modificirati, koristi se iterativni algoritam naziva gradijentni spust. Gradijentni spust izračunava gradijente, odnosno parcijalne derivacije funkcije greške po varijabli *w* koja predstavlja težinu pojedinog neurona. Princip rada gradijentnog spusta je da se uz pomoć dobivenih gradijenata izračunavaju nove težine te se one ažuriraju u smjeru suprotnom širenju unaprijed, tj. od izlaznog sloja prema ulaznom. Prilikom ažuriranja težina, koristi se hiperparametar – stopa učenja koja definira stopu ažuriranja težina. Što je definirana stopa učenja veća, to će gradijentni spust u svakoj iteraciji više promijeniti težine. Zapravo se radi o svojevrsnoj težini koja definira stopu promjena težina neurona. Kod definiranja stope učenja je potrebno obratiti pozornost jer prevelikom definiranom stopom je moguće da će gradijentni spust preskočiti minimum funkcije greške, tj. neće konvergirati. U drugom slučaju, moguće je definirati premalu stopu učenju što opet predstavlja problem jer premala stopa učenja zbog sitnih promjena težina neurona prouzrokuje sporo učenje modela [3].

## 3.3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže su vrsta neuronskih mreži koje se koriste za procesiranje podataka koji sadrže rešetkastu strukturu. Primjer podataka za koje se može reći da imaju rešetkastu strukturu su npr. podaci vremenskih serija koji imaju jednodimenzionalnu rešetkastu strukturu i slike koje imaju dvodimenzionalnu rešetkastu strukturu sačinjenih od pixela.

Konvolucijske neuronske mreže posljednjih godina imaju veliki uspjeh i primjenu nad takvim podacima. Naziv dolazi od matematičke operacije konvolucije koja se označava znakom $*$, a radi se o specijalnoj vrsti linearnog operatora koji je definiran jednadžbom:

$$s(t) = \int x(a)w(t-a)da$$

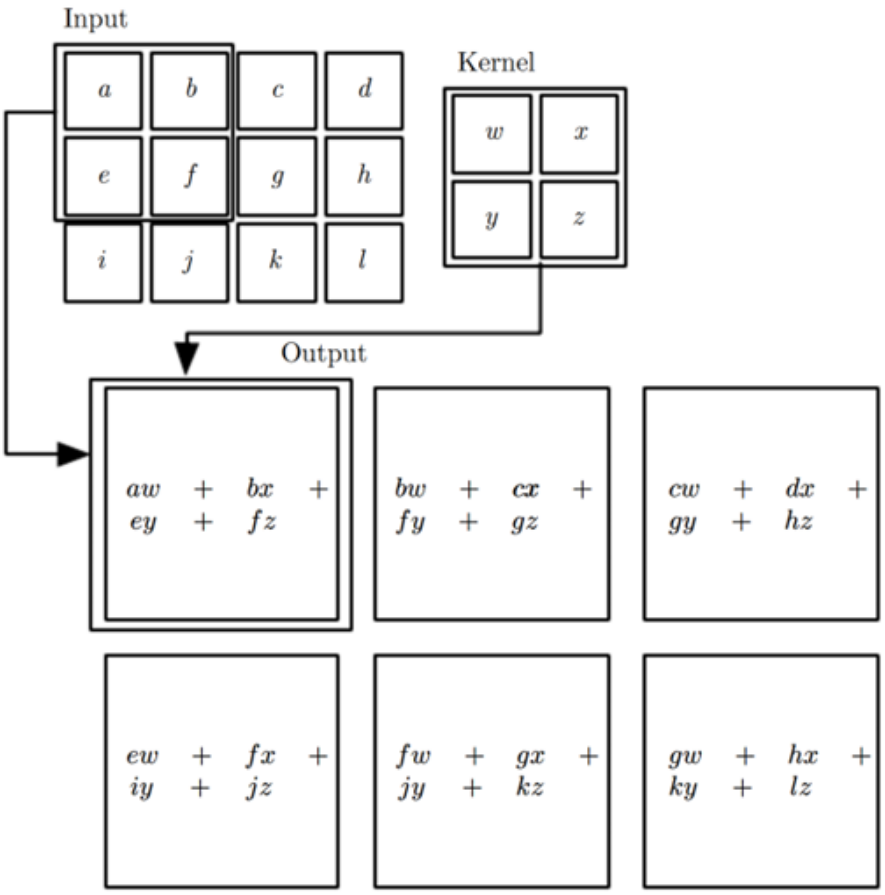
ili pak koristeći linearni operator konvolucije $*$:

$$s(t) = (x * w)(t)$$

gdje je $x$ ulazni podatak, a $w$ filter/jezgra (eng. *kernel*), dok se rezultat operatora konvolucije se naziva mapa značajki (eng. *feature map*). Ulazni podatak je najčešće višedimenzionalno polje podataka (npr. slike), dok je jezgra višedimenzionalno polje parametara koje se tijekom učenja uče, odnosno ažuriraju. Konvolucije se obično koriste nad više osi odjednom, pa se tako za primjerice dvodimenzionalnu sliku $I$ koja predstavlja ulazni podatak, koristi dvodimenzionalna jezgra $K$ na sljedeći način:

$$S(i,j) = (I \cdot K)(i,j) = \Sigma_m \Sigma_n I(m,n)K(i-m, j-n)$$

Na slici 3 se nalazi primjer izračuna pomoću konvolucije nad dvodimenzionalnim poljem. Princip računanja konvolucijom je da se svaki $i, j$ element jezgre pomnoži s $i, j$ elementom ulaznog polja te se na kraju umnošci zbroje te taj rezultat predstavlja prvi element mape značajki. Takav postupak se ponavlja na način da se u novom koraku uzme sljedeća $i \ x \ j$ elementa ulaznog podatka, a završava kad su izračunate sve potrebne mape značajki. Može se zaključiti da su konvolucijske neuronske mreže one neuronske mreže koje u barem jednom svom sloju umjesto matričnih množenja koriste operator konvolucije [1].




Slika 3: Primjer računanja konvolucije nad dvodimenzionalnim poljem (Izvor: [1])

Čest je slučaj da se u jednome sloju konvolucijske neuronske mreže koristi više jezgri kako bi se time omogućilo raspoznavanje više različitih značajki, kao što je primjerice raspoznavanje vertikalnih, horizontalnih i dijagonalnih rubova. Međutim, treba obratiti pozornost na broj jezgri na konvolucijskom sloju jer veći broj jezgri dovodi do smanjenja performansi modela, tj. do sporijeg treniranja i predviđanja. Kako bi se smanjila kompleksnost modela i smanjila prostorna dimenzija mape značajki koristi se parametar skoka koji omogućava veći pomak jezgre nad ulaznim podatkom prilikom konvolucije, što znači da će operator konvolucije preskočiti jedan dio ulaznih podataka te time rezultirati manjom mapom značajki. Drugi način za smanjivanje prostorne dimenzije je korištenje sloja sažimanja nakon konvolucijskog sloja, što se zapravo i u praksi češće koristi. Jedan od mogućih slojeva sažimanja je maksimalni sloj sažimanja koji iz promatranog dijela mape značajki uzima samo najveći (maksimalni) element po svojoj vrijednosti. Osim toga, postoji parametar za određivanje načina ispunjavanja slike. Ispunjavanje slike se koristi u slučajevima kada jezgra ne stane na dio slike pa je tu sliku moguće popuniti na jedan od dva načina: popunjavanjem preostalog dijela nulama i izostavljanje dijela slike koji ne može stati na jezgru [4].

Prilikom konstruiranja konvolucijske neuronske mreže potrebno je obratiti pozornost na dimenziju mape značajki na pojedinom konvolucijskom sloju jer dimenzija mape značajki jednog sloja utječe na dimenzije preostalih mapi značajki, pa i na konačan rezultat. Izlaznu dimenziju mape značajki je moguće odrediti putem jednadžbe:

$$d_{izlaz} = \frac{d_{ulaz} - k + 2p}{s} + 1$$

gdje je $d_{ulaz}$ ulazna prostorna dimenzija mape značajki ili ulaznog podatka, $k$ predstavlja dimenzije jezgre, $p$ broj popunjavanja, a $s$ broj skokova [4].

## 3.4. Generativne suparničke mreže

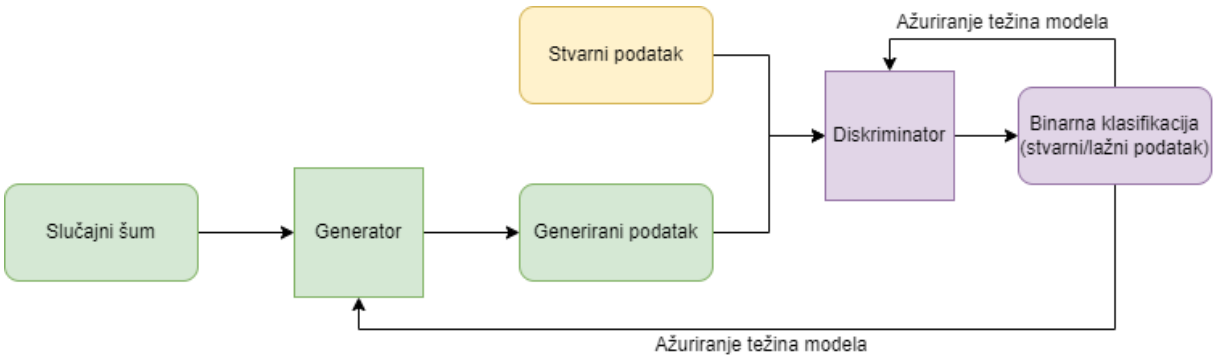
Generativne suparničke mreže pripadaju skupini generativnih modela, što znači da takve mreže omogućuju generiranje novih uzoraka iz dane distribucije. One se sastoje od dva modela neuronske mreže: generatora $G(z)$ i diskriminatora $D(x)$ koji se međusobno natječu. Svrha generatora je generiranje novih podataka, dok je zadaća diskriminatora pronalazak razlike između stvarnih podataka i lažnih podataka stvorenih od strane generatora. Ulaz u generator je nasumični šum na temelju kojeg se može generirati novi podatak, dok je izlaz novi generirani podatak, npr. slika. Ulazi u diskriminator su stvarni podatak i lažni generirani podatak dobiven generatorom, a izlaz je skalar između 0 i 1 koji predstavlja vjerojatnost postojanja stvarnog podatka. Opisani koncept rada generativnih suparničkih mreža je vizualiziran na slici 4. Krajnji cilj generativne suparničke mreže je postići točku ravnoteže, a to je slučaj kada je generator u stanju generirati podatke nalik stvarnim, dok diskriminator kao svoj rezultat izbacuje vjerojatnost od 0.5. U takvom slučaju postigao se cilj generiranja novih podataka nalik stvarnim te se za inferenciju i predviđanje dalje može koristiti samo model generatora [5].

Točku ravnoteže je moguće postići na način da se treniranjem diskriminatora $D$ maksimizira vjerojatnost dodjele ispravne oznake (stvarni ili lažni podatak) oboje stvarnim podacima za treniranje i generiranim lažnim podacima. Za razliku od diskriminatora, tijekom treniranja generator $G$ mora simultano minimizirati grešku $log(1 - D(G(z)))$ [6].

Primjena generativnih suparničkih mreži je ogromna, a neka od njih mogu biti:

- generiranje slika visoke rezolucije na temelju ulazne slike
- kreiranje nove umjetnosti (slika, glazba, poezija...)
- translacija slike na sliku – translacija slike u neku novu domenu, npr. dan u noć, ljeto u zimu [7]

Također, primjena generativnih suparničkih mreža je moguća i na problemu kojim se bavi ovaj rad – korekcija položaja oka (pogleda) u video komunikaciji, a o tome će biti više riječi u sljedećem poglavlju u kojem će se razmotriti najnovija istraživanja na temu ovog rada.




Slika 4: Koncept rada generativnih suparničkih mreža (Prema: [7])

# 4. Pregled literature

Problem korekcije položaja očiju u video komunikaciji se pokušava riješiti već dugi niz godina. U ranijim istraživanjima su predložene metode koje koriste dodatnu opremu poput primjerice senzora dubine i stereo kamere [8].

Međutim, većina ljudi koja koristi video komunikacijske alate nema takvu opremu već ima samo klasičnu RGB kameru koja se nalazi na mobitelima ili računalima. Zbog toga metode za korekciju položaja očiju koje zahtijevaju dodatnu opremu nije moguće primijeniti u stvarnim svakodnevnim situacijama, već je potrebno primijeniti one metode koje ne zahtijevaju dodatnu opremu ni poseban angažman korisnika gdje bi korisnik trebao podesiti ulazne parametre za izvršenje metode.

Prema tome, alternativa hardverskim rješenjima za problem korekcije položaja očiju je primjena isključivo neke vrste softverskog rješenja. S obzirom na to da današnje metode umjetne inteligencije i strojnog učenja omogućuju obradu slika u stvarnom vremenu, u narednim potpoglavljima će se razmotriti najnovija istraživanja na temu korekcije položaja oka koje u svojim istraživanjima koriste metode strojnog učenja.

## 4.1. Korekcija pogleda primjenom stabla odlučivanja

Jedno od prvih rješenja korekcije položaja očiju koje ne uključuje korištenje dodatne opreme poput dubinskih senzora je bilo primjenom strojnog učenja i metode stabla odlučivanja. Najprije se predviđa lokacija očiju i njihovih 6 ključnih točaka, a nakon toga se uz pomoć strojnog učenja obrađuju locirane oči. Obrada se izvršava na način da se za svaki pixel očiju vrši podudaranje sa određenim pixelima u podacima za treniranje. Podudaranje takvih pixela ovisi o susjednim pixelima i lokaciji pixela s obzirom na predviđene ključne točke očiju, a određuje se uz pomoć modela stabla odlučivanja. Prolaskom pojedinog pixela kroz stablo odlučivanja, primjenjuju se dva testa. Prvi test je test izgleda te on uspoređuje vrijednosti dvaju pixela. Drugi test je lokacijski test koji ovisi o ključnim točkama očiju i zadanoj granici te se uspoređuje da li je razlika u koordinatama, odnosno lokaciji između pixela i ključne točke veća od zadane granice. Rezultat primjene stabla odlučivanja su dvodimenzionalni vektori uz pomoć kojih se određuju novi pixeli koji čine izlaznu sliku, tj. sliku s korigiranim položajem očiju [9].

Iako ovaj sustav nudi rješenje obrade slika u stvarnom vremenu za potrebe korekcije položaja očiju bez upotrebe dodatne opreme, postoje mnogi nedostaci zbog kojih se on ne može kvalitetno primijeniti u današnjim aplikacijama. Glavni nedostatak ovog sustava je što on omogućuje samo vertikalnu korekciju, a to ujedno pretpostavlja i da se kamera nalazi na gornjem centru ekrana. Također je potrebno predefinirati udaljenost kamere od korisnika što dodatno otežava korištenje ovog sustava.

## 4.2. Sustav Look at Me!

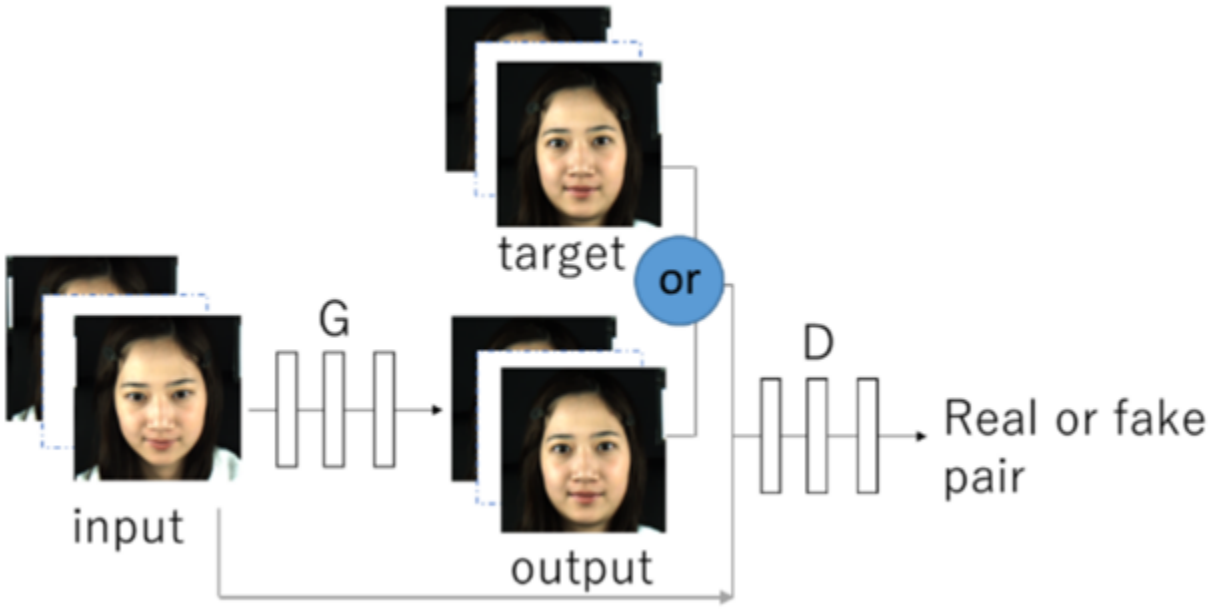
Look at Me! je sustav korekcije pogleda koji kao cilj ima zadržavanje kontakta očiju sudionika video konferencije u stvarnom vremenu. Princip rada ovog sustava je da se preusmjeri pogled lokalnog sudionika razgovora s udaljenog sudionika koji je vidljiv na ekranu računala na kameru lokalnog sudionika. Na taj način se postiže prividni kontakt očiju te se takve modificirane slike šalju udaljenom sugovorniku [10].

Sustav radi na način da se prvo procijene kutovi rotacije očnih jabučica s obzirom na pozicije kamere i očiju sugovornika. Nakon toga se prepoznaju, segmentiraju i pretprocesiraju oči korisnika te se zatim iskrivljuju uz pomoć konvolucijske neuronske mreže što rezultira korekcijom smjera pogleda očiju. Slike s korigiranim očima se na kraju zamjenjuju sa originalnim slikama očiju korisnika. Neuronska mreža koja je korištena u ovom sustavu je ekstenzija, odnosno modifikacija tadašnjeg najsuvremenijeg (engl. *state of the art*) modela DeepWarp. Ulazi u model su slika osobe snimana kamerom, procijenjeni kutovi rotacije očnih jabučica i anchor mape dobivene na temelju ključnih točaka lica, dok je izlaz slika osobe sa korigiranim položajem očiju [10].

Pretpostavka i glavno ograničenje ovog sustava je da smije postojati samo jedan udaljeni sudionik razgovora na ekranu prilikom korekcije pogleda (položaja očiju) lokalnog sudionika. U suprotnome slučaju, sustav ne može prepoznati kojeg se odaljenog sugovornika gleda, pa se korigira pogled prve detektirane osobe. Sljedeći nedostatak je mogućnost pojave izobličenja koje se javljaju u slučaju prikrivanja očiju, npr. naočalama ili kosom. Nadalje, ovaj model na ulazu zahtjeva kut rotacije očnih jabučica, a takav zahtjev predstavlja problem u stvarnim situacijama jer često nije moguće automatski saznati specifikacije uređaja. Posljednji nedostatak je pojavljivanje vizualnih anomalija u slučajevima kada je pogled daleko od centra [10].

## 4.3. Korekcija pogleda primjenom rekurzivno-uvjetnih generativnih suparničkih mreža

Uz pomoć rekurzivno-uvjetnih generativnih suparničkih mreža moguće je automatski generirati video koji sadržava korigirani pogled. Takva metoda omogućava da prilikom generiranja u obzir uzme i odnose između slika u videu, što znači da takva metoda analizira značajke vremenske serije. Cilj ovakvog modela je konvertirati video gdje osoba ima lice i pogled prema dolje u video u kojem osoba gleda ravno u kameru. To se postiže tako što se kao ulaz u generator proslijedi video prije konverzije, odnosno video u kojem osoba ne gleda u kameru te on prilikom učenja generira konvertirani video. Zadaća diskriminatora je odrediti da li je generirani video od strane generatora stvaran ili ne te istu takvu odluku donijeti i za stvaran video. Autori ove metode su također koristili i CLSTM (engl. *convolutional long short-term memory*) model koji omogućuje da se prilikom generiranja svake slike u obzir uzimaju i značajke iz prethodnih slika u videu, čime se dobiva na konzistenciji samih slika [11].



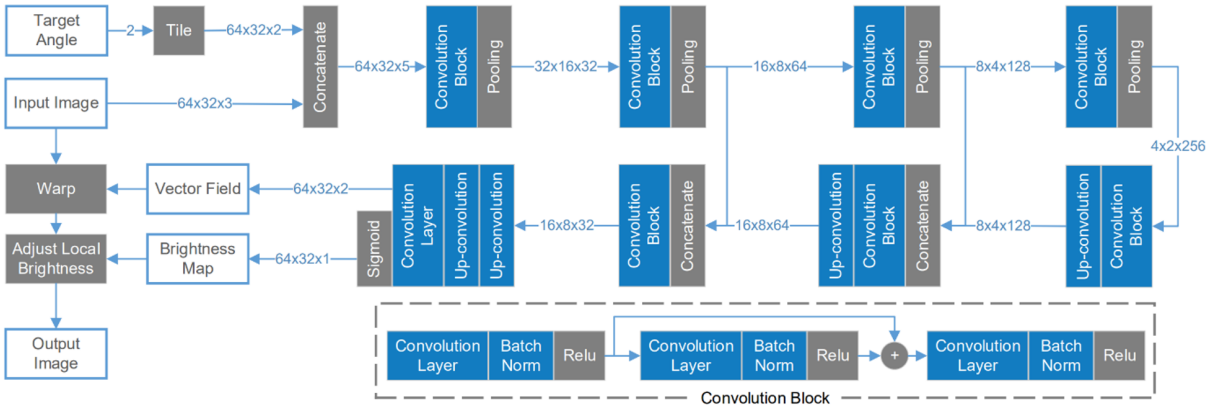

Slika 5: Model korekcije pogleda očiju uz pomoć rekurzivno-uvjetnih generativnih suparničkih mreža (Izvor: [11])

Kao manu autori navode nedostatak preciznosti generiranih videa, a uzrok toga je nedovoljno velika struktura modela. Osim toga, s obzirom na to da su autori samo evaluirali i prikazali rezultate konverzije videa s jednog smjera pogleda u drugi, pretpostavka je da bi dani model imao probleme s generaliziranjem drugih smjerova pogleda [11].

## 4.4. Intelov sustav korekcije pogleda

Sustav korekcije položaja očiju koji su razvili znanstvenici iz Intela omogućuje korekciju pogleda prema centru ekrana u stvarnom vremenu, a da pritom ne zahtijeva kut rotacije ili preusmjeravanja kao što je to slučaj kod sustava opisanog u poglavlju 4.2. Razvijeni model osim korekcije položaja očiju omogućava i implicitno predviđanje smjera ulaznog pogleda, odnosno predviđanje koordinata pogleda prije same korekcije. Korekcija položaja očiju se ostvaruje na prirodan način zahvaljujući razvijenim kontrolnim mehanizmima. Oni omogućuju kontrolu jačine korekcije, sprječava neprirodne jezive poglede koji nastaju prilikom pretjerane korekcije položaja očiju te osiguravaju konzistenciju u stvarnom vremenu [8].

Ovaj sustav prvo koristi model detekcije ključnih točaka lica kako bi se na taj način mogle locirati oči. Pronađene oči na slici se zatim izrežu te se one onda dalje prosljeđuju u konvolucijsku neuronsku mrežu koja je zadužena za korekciju pogleda. Izlaz modela se sastoji od dviju komponenti, od kojih izlaz jedne komponente sadrži dva kanala, dok druga komponenta sadrži preostali jedan kanal. Dva kanala služe za predikciju horizontalnih i vertikalnih komponenata vektorskog polja koje se koristi za iskrivljenje ulazne slike, odnosno očiju. Treći kanal se koristi za podešavanje svjetline koja poboljšava pojavu bjeloočnica nakon iskrivljavanja pogleda. Cijelu arhitekturu modela je moguće vidjeti na slici 6 [8].




Slika 6: Arhitektura Intelovog modela za korekciju pogleda (Izvor: [8])

Model je treniran na dvosmjeran način kako bi se omogućilo obrnuto preslikavanje. U prvom smjeru su ulazi slika i ciljani kut korekcije pogleda, a cilj je minimizirati grešku korekcije koja je zadana kao srednja kvadratna pogreška između kreirane korigirane slike i stvarne ciljane slike. U drugom smjeru su ulazi kreirana korigirana slika i ulazni kut za preusmjeravanje pogleda natrag u originalno stanje. Za treniranje i evaluaciju modela su korišteni stvarni i sintetički podaci. Stvarni podaci su uglavnom korišteni za evaluaciju, dok su sintetički, odnosno generirani podaci za treniranje modela. Podaci se sastoje od parova slika, konkretnije od slika gdje subjekt gleda direktno u kameru te slika gdje subjekt gleda u neku nasumičnu točku na ekranu [8].

Od svih razmatranih sustava za korekciju položaja očiju u video komunikaciji, ovaj sustav nudi najbolje rješenje. Razlog tome je što on ne zahtijeva ulazne parametre poput kuta rotacije ili preusmjeravanja, pa je onda i korištenje takvog sustava fleksibilno i jednostavno. Osim toga, dodatna bitna prednost ovog sustava je što uključuje i kontrolni mehanizam za kontrolu korekcije položaja očiju i pritom sprječava nepoželjne neprirodne poglede.

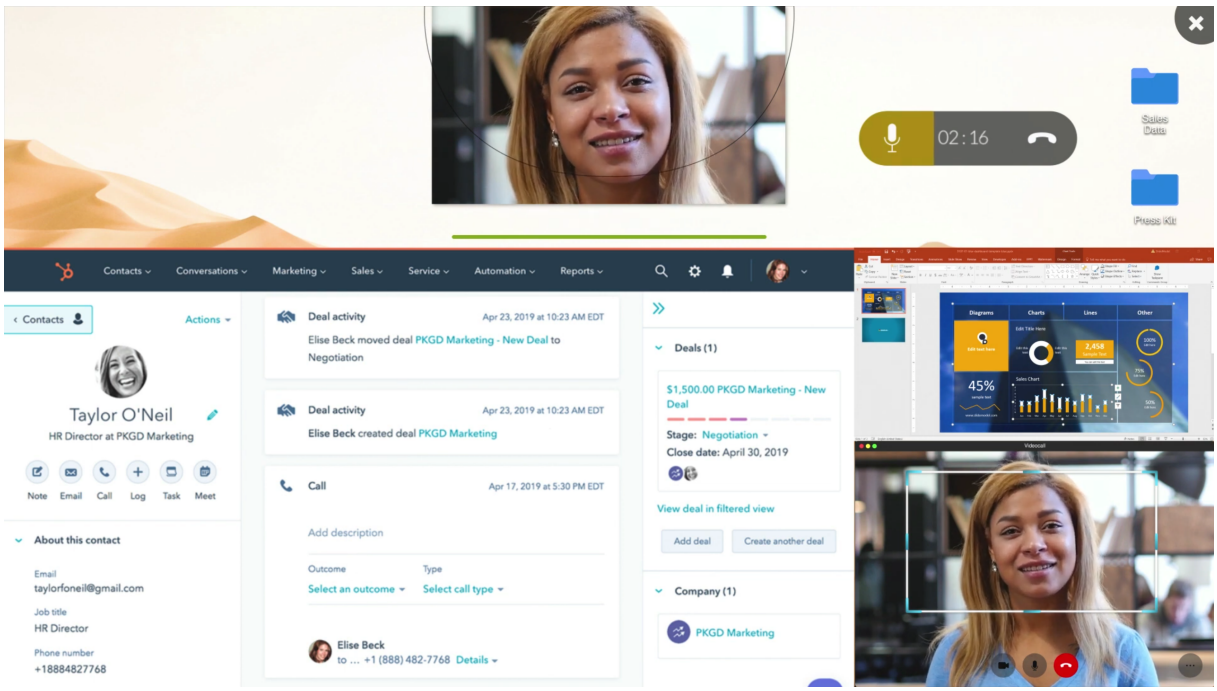
# 5. Alternativna aplikacijska rješenja

Iako postoje brojna istraživanja na temu analize pogleda i korekcije položaja očiju, danas još ne postoji besplatna aplikacija ili programsko rješenje koje bi osiguralo virtualni kontakt očima, neovisno o korištenom video komunikacijskom alatu. U nastavku će se razmotriti neka od najpopularnijih programskih rješenja koja omogućuju virtualni kontakt očiju.

## 5.1. NUIA Full Focus

NUIA Full Focus je komercijalni sustav koji olakšava održavanje kontakta očima tijekom video konferencije. Ovaj sustav ne nastoji korigirati položaj očiju primjenom umjetne inteligencije već taj problem rješava na način da se uz pomoć dodatnog uređaja za praćenje oka prati pogled korisnika. Na temelju pogleda ovaj sustav duplicira dio ekrana kojeg korisnik gleda te takav dupliciran sadržaj postavlja na vrh ekrana gdje se obično nalazi kamera čime se dobiva efekt kontakta očima [12].

Na slici 4 je moguće vidjeti prikaz rada NUIA Full Focus sustava. U prikazanoj situaciji, korisnik aplikacije gleda u donji desni dio ekrana u kojem se nalazi otvoreni prozor video konferencijskog alata te se taj dio ekrana duplicira u gornji središnji dio ekrana.




Slika 7: Prikaz rada NUIA Full Focus sustava (Izvor: [12])

Ovaj princip rješavanja problema kontakta očima ima neke prednosti, ali i brojne mane. Prednosti ovog sustava su što je podržan od strane popularnih video konferencijskih alata te s obzirom na to da nema prividne korekcije položaja očiju, kontakt očima izgleda potpuno prirodno. Međutim, ovaj sustav je podržan samo na Windows operacijskom sustavu, što znači da ne postoji podrška za mobilne uređaje i preostale operacijske sustave. Osim toga, postoje

i neki hardverska ograničenja poput veličine monitora te potreba za korištenjem uređaja za praćenje oka [12]. Nadalje, korisničko iskustvo takvog sustava može biti nepoželjno iz razloga što korisnici vjerojatno ne žele konstantno dupliciranje dijela ekrana zbog toga što takav pristup ograničava korištenje zaslona.

## 5.2. FaceTime

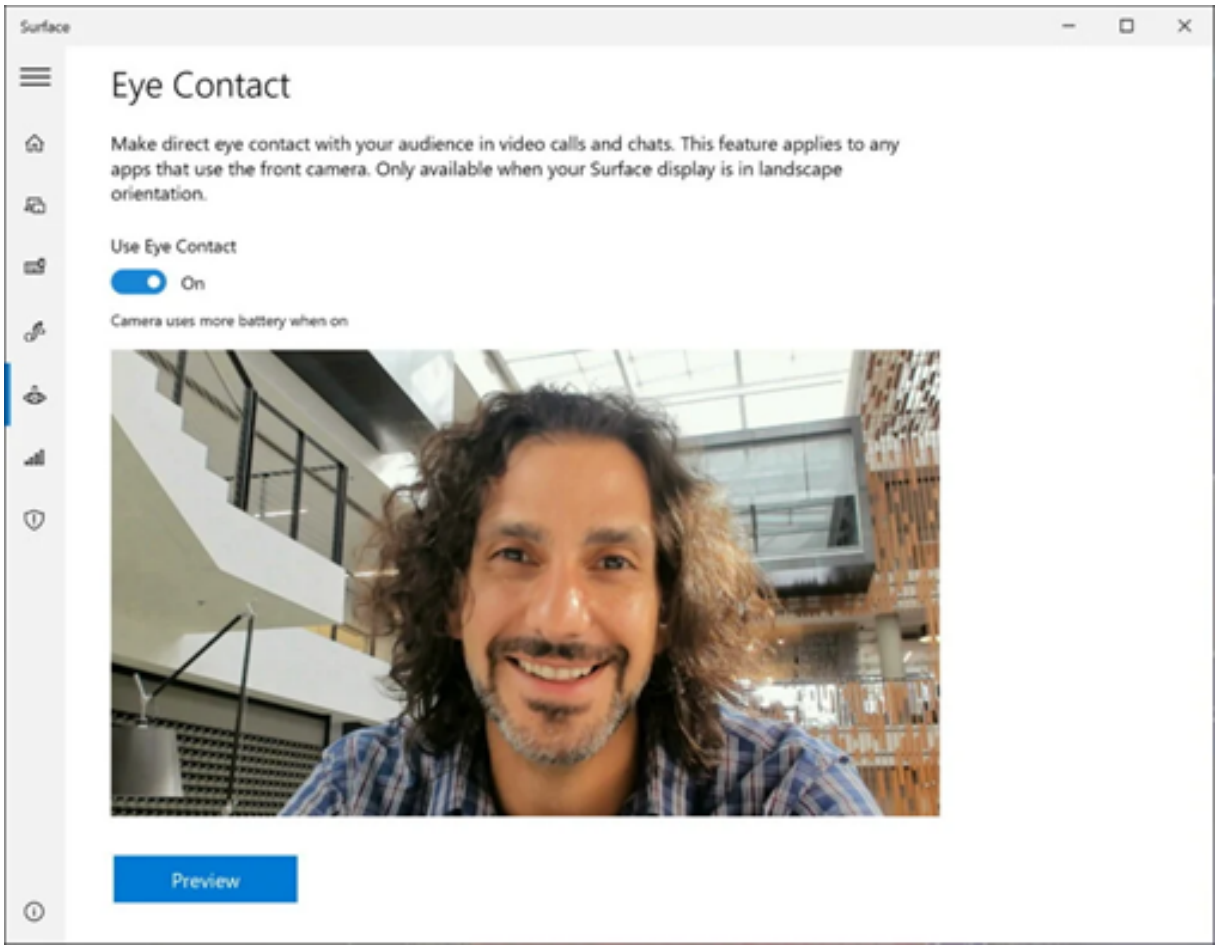
FaceTime je aplikacija za video komunikaciju razvijen od strane Apple-a koji je dostupan za Apple-ove uređaje poput iPhone-a, iPad-a i Apple Watch-a. Ova aplikacija ima opciju korekcije položaja očiju u svrhu ostvarenja kontakta očima, a dostupna je za generaciju iPhone-a 11 te novije uz uvjet korištenja iOS 14 operacijskog sustava ili novijeg. Nisu poznati detalji implementacije ove opcije, međutim pretpostavka je da sustav koristi Apple-ov ARKit. Također je dokazano da sustav koristi neku od metoda iskrivljenja slika, a već su u četvrtom poglavlju objašnjene neke od najnovijih metoda umjetne inteligencije za korekciju položaja očiju koje koriste tehnike iskrivljenja slike, pa je zbog toga pretpostavka da Apple-ov sustav koristi nešto slično tim metodama [13]. Iako Apple-ov sustav za ostvarenje kontakta očima radi prilično dobro, veliki je nedostatak što je taj sustav dostupan samo za novije iPhone uređaje te ga je moguće koristiti samo u FaceTime aplikaciji. Ispod se nalazi prikaz rada ovog sustava, gdje se s lijeve strane nalazi slika u slučaju kad se ne primjenjuje sustav korekcije položaja očiju, a s desne strane se nalazi slučaj kad se takav sustav primjenjuje. U oba slučaja korisnik gleda u ekran koji se nalazi ispod kamere.

Slika 8: Prikaz rada FaceTime sustava (Izvor: [13])

## 5.3. Microsoft Eye Contact

Slično kao i FaceTime, Microsoft na specifičnim uređajima nudi mogućnost korekcije položaja očiju. Jedini uređaj koji trenutno podržava takvu mogućnost je Surface Pro X, a razlog

tome je što taj uređaj sadržava ARM procesor koji u sebi sadržava poseban AI čip koji dodatno ubrzava matematičke operacije korištene u umjetnoj inteligenciji [14]. Naravno, mogućnost podrške Microsoft Eye Contacta samo za taj uređaj je glavna mana, no za očekivati je da će kroz godine sve veći broj uređaja koristiti procesore koje sadrže takav čip, pa će onda i podrška ovog sustava korekcije položaja očiju biti veća. Prednost u odnosu na sustav koji je razvio Apple je taj što je Microsoftov sustav podržan od strane mnogih drugih video komunikacijskih alata, dok Apple-ov sustav korekcije položaja očiju je moguće koristiti samo u FaceTime aplikaciji. Microsoft također nije ponudio detalje implementacije svog sustava, no poznato je što ovaj sustav koristi neku od metoda strojnog učenja. Na sljedećoj slici se nalazi prikaz primjene Microsoftovog sustava korekcije položaja očiju iz koje se teško može zaključiti da se zapravo radi o prividnoj stvarnosti.




Slika 9: Prikaz rada Microsoft Eye Contacta (Izvor: [14])

# 6. Opis implementacije i rada sustava korekcije smjera pogleda očiju

Nakon razmatranja svih dosadašnjih sustava za korekciju položaja očiju u video komunikaciji, sustav predložen od strane znanstvenika iz Intela nudi najprihvatljivije praktično rješenje. Za razliku od drugih istraživanja, u tom istraživanju je navedeno da predloženi sustav ne zahtijeva ulazne parametre poput kuta rotacije ili preusmjeravanja, dok ostala promatrana istraživanja opisana u četvrtom poglavlju imaju ograničenja poput potrebe primjerice korištenja ulaznog kuta rotacije očnih jabučica ili udaljenosti od ekrana. Takva ograničenja otežavaju praktičnu upotrebu, a ako se još uzme u obzir da sustav predložen od strane znanstvenika iz Intela nudi dobre performanse i na tipičnim današnjim procesorima te producira kvalitetne rezultate korekcije pogleda, čini se smislenim da implementacija sustava za korekciju položaja oka bude vođena upravo tim istraživanjem.

## 6.1. Korištene tehnologije i algoritmi

S obzirom na to da će se implementacija sustava korekcije pogleda bazirati na istraživanju opisanom u poglavlju 4.4., kao temeljne tehnologije će se koristiti strojno učenje i konvolucijske neuronske mreže. Iako je željeni model strojnog učenja moguće implementirati u raznim programskim jezicima, u praksi se danas najčešće koristi programski jezik Python iz razloga što on osim što je jednostavan, također nudi i širok raspon biblioteka koje značajno mogu olakšati i ubrzati sam proces implementacije. Osim toga, biblioteke koje podržavaju i omogućuju implementaciju modela strojnog učenja su napisane u mnogo bržim programskim jezicima poput C++-a, pa mana Pythona – slabe performanse ne predstavlja zapreku. Jedna od tih biblioteka je TensorFlow koja je odabrana biblioteka za implementaciju modela te njegovo treniranje i inferenciju. Nadalje, s obzirom na to da je cilj ovog rada korekcija položaja očiju u stvarnom vremenu na temelju slika dobivenih iz nekog uređaja za snimanje videa (npr. web kamera), potrebno je onda u stvarnom vremenu čitati i procesirati slike iz tog video ulaza. U ovom radu je fokus stavljen na desktop verziju sustava, pa je dovoljno omogućiti čitanje ulaznih slika iz web kamere ili već neke druge kamere koja je priključena na računalo ili laptop. Kako bi se to ostvarilo koristi se biblioteka OpenCV, koja će se također koristiti i za manipulaciju slikama. Za kraj, kako bi se omogućio prikaz rezultata korekcije pogleda u željenom video konferencijskom alatu potrebno je koristiti virtualnu kameru. Virtualnu kameru je u Pythonu moguće na jednostavan način koristiti pomoću biblioteke pyvirtualcam.

## 6.2. Opis arhitekture modela strojnog učenja i arhitekture sustava

Arhitektura modela strojnog učenja je ista onoj arhitekturi koju su predstavili znanstvenici iz Intela (poglavlje 4.4.), a vidljiva je na slici 6. U nastavku će se detaljnije opisati arhitektura tog modela, a nakon toga i arhitektura čitavog sustava korekcije pogleda.

### 6.2.1. Opis modela strojnog učenja

Postoje dva ulaza u model: ulazna slika i ciljani kut korekcije pogleda. Na ulaznoj slici se nalazi slika jednog oka čiji je položaj potrebno korigirati, dok je ciljani kut vektor smjera pogleda reprezentiran u Kartezijevom koordinatnom sustavu. Model nije treniran da samo korigira položaj oka u centralnom smjeru kako bi se na taj način dobio privid kontakta očima, već je treniran da korigira oko u različitim smjerovima, pa zbog toga ciljani kut vektora smjera varira ovisno o izlaznoj slici, odnosno smjeru pogleda oka izlazne slike. Razlog treniranja modela na različitim smjerovima pogleda je poboljšanje robusnosti modela te mogućnost modifikacije ciljanog vektora smjera. Naime, prilikom inferencije je potrebno postaviti vektor smjera pogleda na nulu jer je to centar na Kartezijevom koordinatnom sustavu, no postoji mogućnost da na nekim uređajima sustav neće korigirati smjer pogleda prema željenom centru, pa s obzirom da model ima mogućnost korekcije pogleda prema bilo kojem smjeru, nije problem prije inferencije pomaknuti ciljani vektor smjera na onaj smjer koji daje bolje rezultate. Osim toga, mogućnost modifikacije ciljanog vektora smjera pogleda otvara mogućnosti i za nova proširenja u sustavu korekcije pogleda.

U prvom koraku modela, prostorne dimenzije ciljanog vektora smjera pogleda se prošire na način da one budu jednake dimenziji ulazne slike, a to je dimenzija 64x32. Nakon toga se vektor smjera i slika spoje u jednu komponentu koja dalje ulazi u model. Takva komponenta ima dimenzije 64x32x5 jer se vektor smjera i slika spajaju po zadnjoj, trećoj dimenziji. Takva komponenta ulazi u prvi konvolucijski blok. Konvolucijski blok je skup slojeva sačinjen od tri konvolucijskih slojeva, tri serijskih normalizacija i tri ReLU aktivacijskih funkcija. Ovi slojevi su slijedno povezani na način da prvo slijedi konvolucijski sloj, pa serijska normalizacija te na kraju ReLU aktivacijska funkcija te je takva slijedna povezanost ponovljena tri puta. Također, postoji još jedna dodatna veza iz izlaza prvog ReLU sloja u izlaz iz drugog ReLU sloja te se one prije ulaza u treći slijedni blok spajaju međusobnim zbrajanjem.

Još je važno za napomenuti da se u konvolucijskom bloku umjesto konvolucijskih slojeva koriste dubinsko-separabilni konvolucijski slojevi. Takvi slojevi koriste posebnu vrstu konvolucije, gdje umjesto izračuna konvolucije preko svih kanala u jednom koraku, konvolucija se računa u dva koraka. U prvom koraku se primjenjuje samo jedna konvolucijska jezgra za svaki ulazni kanal na slici. U drugom koraku se koristi konvolucijska jezgra prostorne dimenzije 1x1, dok je dubinska, odnosno treća dimenzija jezgre jednaka trećoj dimenziji slike. Rezultat primjene ovih koraka je značajno smanjenje broja potrebnih matematičkih operacija, odnosno brže performanse, no time se i istodobno smanjuje broj parametara neuronske mreže što može rezultirati lošijim rezultatima prilikom treniranja mreže [15].

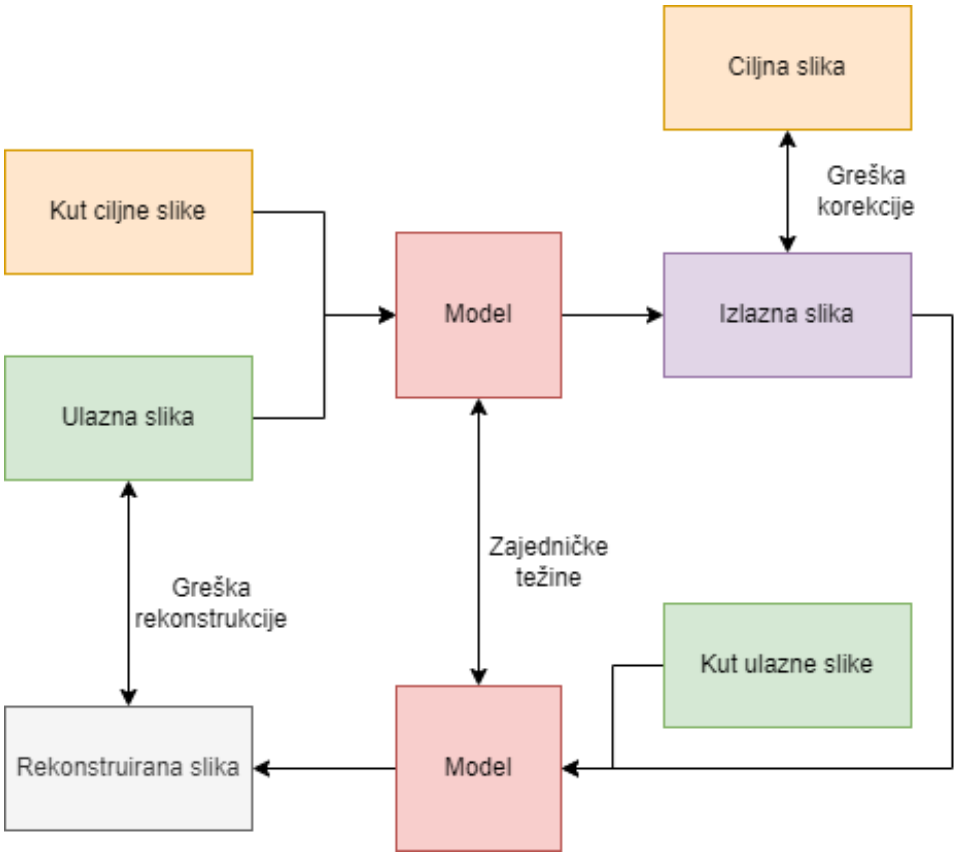
Nakon konvolucijskog bloka slijedi maksimalni sloj sažimanja. Izlaz iz prvog konvolucijskog bloka daje podatak dimenzije 32x16x32 te se prosljeđuje u novi konvolucijski blok. Ova dimenzija je nastala jer su se prve dvije dimenzije prepolovile zbog primjene sloja sažimanja, dok je treća dimenzija rezultat upotrebe 32 jezgre pri konvolucijskom bloku. U novom konvolucijskom bloku se opet prepolove prve dvije dimenzije dok se treća dimenzija podupla. Nakon četvrtog konvolucijskog bloka dobije se podatak dimenzije 4x2x256 te se nakon toga uz pomoć slojeva transponirane konvolucije kreće u proširivanje prve dvije dimenzije i smanjenje treće

sve dok se kao rezultat ne dobije izvorna dimenzija slike. Osim toga, model dodatno koristi neslijedne veze na sličan način kao što je to opisano u konvolucijskom bloku, a vidljivo na slici 6. Takvim vezama se omogućuje lakši oporavak detalja mapi značajki koji su bili izgubljeni prilikom primjene slojeva sažimanja. Posljednji konvolucijski sloj producira podatak iste dimenzije kao i originalna slika, što znači da mapa značajki ima samo tri kanala. Prva dva kanala finalne mape značajki služe za predikciju horizontalnih i vertikalnih komponenata vektorskog polja na temelju kojih se iskrivljuje ulazna slika. Posljednji kanal se koristi za podešavanje svjetline slike. Nakon podešavanja svjetline slike dobije se konačna izlazna slika.

Za treniranje opisanog modela se koristi dvosmjerni princip treniranja. Naime, kao što je već i ranije navedeno, u prvom smjeru su ulazi u model ulazna slika (oko) i ciljani kut pogleda izlazne slike, dok je izlaz iz modela slika s korigiranim pogledom iste rezolucije kao i ulazna slika. U ovom koraku je cilj minimalizirati grešku korekcije $L_k$ koja je zadana kao srednja kvadratna pogreška između stvarne slike s korigiranim pogledom i slike koja je producirao model. U drugom smjeru, ulazi u model su slika s korigiranim pogledom koje je producirao model u prvom smjeru te kut pogleda ulazne slike kako bi se na taj način ponovno korigirao pogled, no ovog puta na kut pogleda izvorne slike. Drugim riječima, u drugom smjeru je cilj kreirati rekonstruiranu ulaznu sliku na temelju korigirane slike nastale u prvom smjeru. Kako bi se to moglo postići, cilj je minimalizirati grešku rekonstrukcije $L_r$ koja je isto zadana kao srednja kvadratna pogreška. Prema tome, postoje dvije različite greške koje model mora minimalizirati – greška korekcije $L_k$ i greška rekonstrukcije $L_r$. Međutim, model koristi samo jednu funkciju greške $L_u$ koja je zadana kao:

$$L_u = 0.8L_k + 0.2L_r$$

što znači da će veći utjecaj na ukupnu grešku imati greška korekcije $L_k$. Na slici ispod se nalazi prikaz koncepta dvosmjernog treniranja.

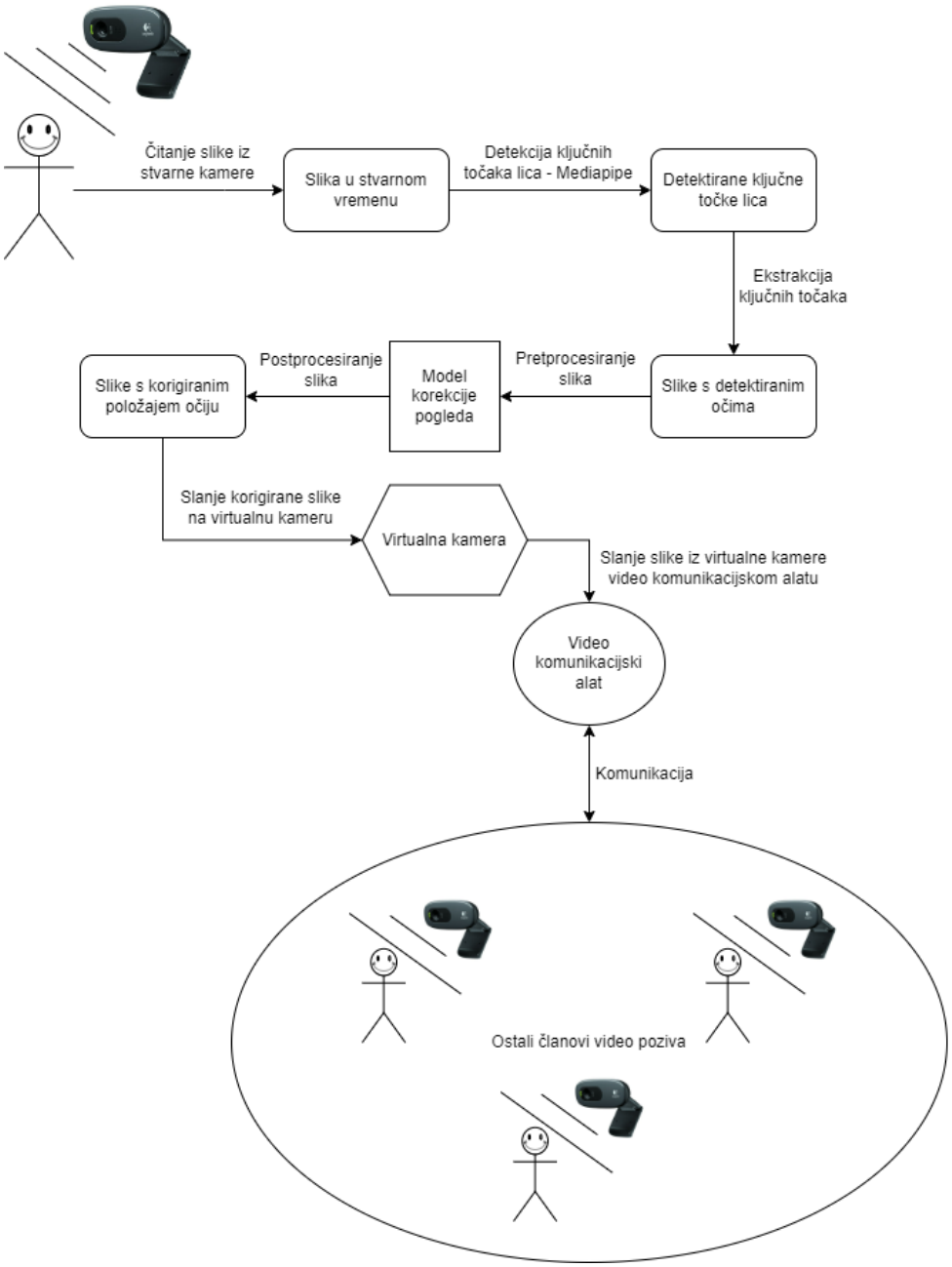



Slika 10: Koncept rada dvosmjernog treniranja modela (Prema: [8])

### 6.2.2. Opis arhitekture sustava

Implementirani sustav korekcije položaja očiju za vrijeme svog izvršavanja čita slike koje snima video kamera priključena na računalo/laptop. Takve slike je potrebno u stvarnom vremenu obraditi te rezultat obrade poslati na virtualnu kameru. Svrha virtualne kamere je da se na jednostavan način omogući slanje obrađene slike na video komunikacijski alat koji prepoznaje virtualnu kameru te da onda takvu obrađenu sliku mogu vidjeti ostali članovi video poziva.

Slike se obrađuju na način da se prvo uz pomoć Python biblioteke Mediapipe prepoznaju ključne točke lica koje se nalazi na slici. S obzirom na to da ključne točke lica sadrže podatke o ključnim točkama očiju, na jednostavan je način moguće iz njih ekstrahirati podatke o lokacijama očiju na slici. Zatim se detektirane oči izrežu od ostatka slike, pretprocesiraju se te se one onda dalje prosljeđuju u model strojnog učenja, odnosno konvolucijsku neuronsku mrežu. Rezultat modela su slike očiju s korigiranim pogledom koje je potrebno još postprocesirati i zamijeniti s nekorigiranim, originalnim očima kako bi se na taj način kao rezultat dobila slika osobe s korigiranim pogledom. Za kraj obrade je takvu sliku potrebno proslijediti virtualnoj kameri koja će nju prikazivati ostalim članovima na odabranom alatu za video komunikaciju.




Slika 11: Arhitektura sustava

## 6.3. Priprema podataka

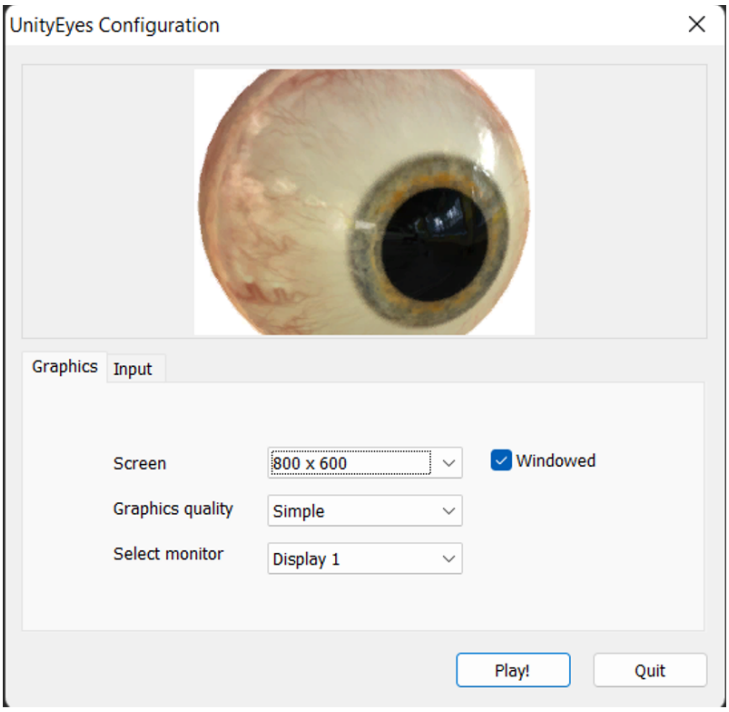
Za treniranje modela se koriste sintetički podaci generirani od strane UnityEyes platforme [16].

UnityEyes omogućuje jednostavno i nasumično generiranje sintetičkih očiju te preusmjeravanje pogleda sintetičke osobe u željenom smjeru. Razlog korištenja sintetičkih podataka, a ne stvarnih je što se na taj način može generirati proizvoljan broj podataka sa savršenim podacima o smjeru pogleda te slikama očiju različitih karakteristika. No postoji i mana korištenja sintetičkih podataka, a to je naravno sama činjenica što su sintetički, odnosno postoji vidljiva razlika između takvih slika i prirodnih. Međutim, ono što je najvažnije kako bi model bio u stanju

minimalizirati svoju grešku je da sintetičke oči generirane od strane UnityEyes-a sadrže iste ili barem slične karakteristike stvarnim očima. Neke od bitnih karakteristika očiju mogu primjerice biti boja očiju, zatim veličina, oblik i boja bjeloočnice, omjer širine i visine očiju, postojanje i gustoća trepavica, itd... S obzirom da UnityEyes nudi pravilne karakteristike očiju, model bi trebao biti u stanju minimalizirati svoju grešku na temelju sintetičkih slika (očiju). Drugim riječima, razlika u kvaliteti karakteristika očiju između takvih sintetičkih slika i prirodnih slika nije prevelika, a kao kompromis se dobije signifikantno veći broj slika sa savršenim oznakama što definitivno može značajno olakšati treniranje modela.

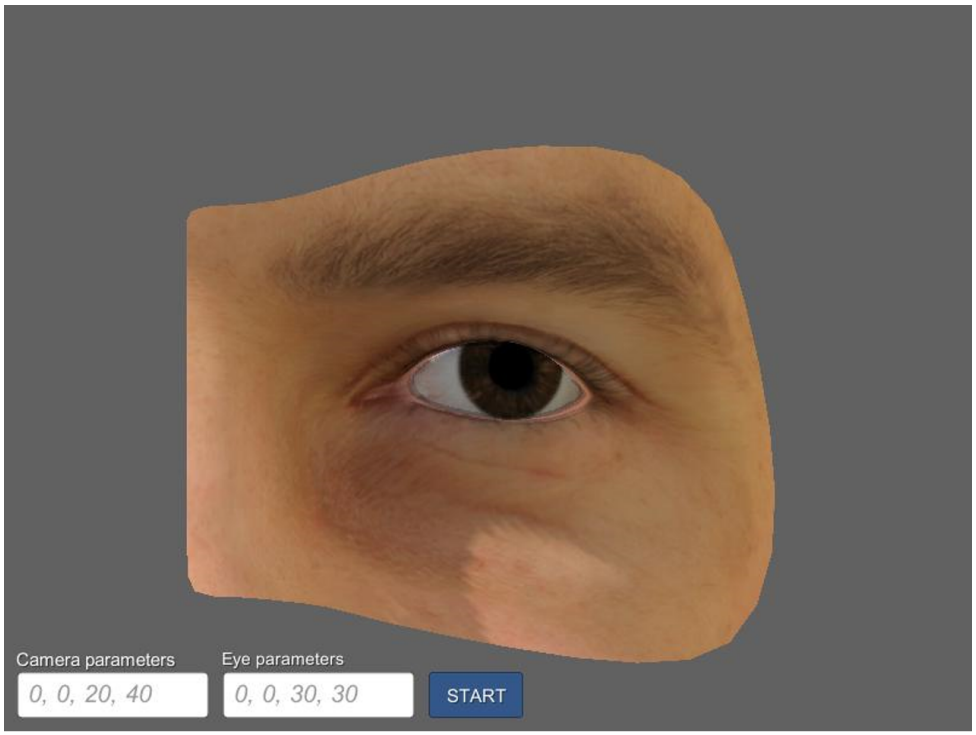
Kao kontrast sintetičkim podacima, postoje javno dostupni skupovi podataka kao što su Columbia Gaze Data Set, MPIIGaze i ShanghaiTechGaze koji sadrže prirodne slike očiju i podatke o smjeru pogleda. Mana takvih skupova podataka je što ih ima brojčano puno manje u odnosu na sintetičke. Na primjer Columbia Gaze Data Set sadrži 56 različitih subjekata od kojih svaki subjekt sadrži 105 jedinstvenih slika, što rezultira s ukupno samo 5 880 slika. Za usporedbu, za potrebe treniranja modela se generiralo 3200 različitih subjekata od kojih svaki sadrži 40 jedinstvenih slika, što rezultira s ukupno 128 000 slika, odnosno preko 21 puta većim skupom podataka. Dodatna prednost korištenja sintetičkih podataka je što se može generirati raznovrsnija distribucija smjerova pogleda, dok kod recimo skupa podataka Columbia Gaze Data Set postoji samo predefiniran skup smjerova pogleda s jednakom varijancom između svakog subjekta.

Nakon pokretanja UnityEyes programa otvara se sučelje na kojemu je moguće odabrati rezoluciju generiranih slika, kvalitetu grafike i zaslon na kojemu će se otvoriti UnityEyes. Odabrana je rezolucija 800x600 jer je to i više nego dovoljna rezolucija za treniranje modela čije su ulazne slike rezolucije 64x32, dok će generirane oči imati veću rezoluciju, što znači da će se svakako prilikom pretprocesiranja smanjiti rezolucija slike. Moguće je odabrati šest različitih razina grafičkih kvaliteta, od Fastest (najbrže) do Fantastic (fantastične). Međutim, za svrhu treniranja modela će se koristi samo dvije razine: Simple i Good (jednostavna i dobra kvaliteta grafike). Razlog tome je što najbolje kvalitete grafike rezultiraju s generiranjem jasno vidljivih pozadinskih refleksija unutar područja očiju koje ipak izgledaju nedovoljno realno te se prilikom generiranja novih očiju uvijek pojavljuje ista pozadinska refleksija što može dovesti do problema s nemogućnošću generaliziranja u modelu. Klikom na gumb Play! se pokreće program. Početno sučelje UnityEyes-a je vidljivo na slici ispod.




Slika 12: Početno sučelje UnityEyes-a

UnityEyes sadrži dva moda: interaktivni i automatski. U interaktivnom modu je moguće mišem i tipkovnicom upravljati scenom, dok je u automatskom modu moguće namjestiti parametre pozicije kamere i smjera pogleda te se pritiskom na gumb *START* pokreće automatsko generiranje sintetičkih podataka na temelju zadanih parametara. Kako bi se u interaktivnom modu kontrolirao smjer pogleda, potrebno je držati i pomicati srednju tipku miša. Za kontrolu kamere se koristi lijeva tipka miša. Nadalje, pritiskom tipke *R* se stvara novi nasumični subjekt s istim postavkama kamere i pogleda kao i prethodna slika. Kako bi se spremila slika potrebno je pritisnuti tipku *S*. Na sljedećoj slici se nalazi prikaz glavnog sučelja UnityEyes-a na kojemu je također vidljiv i primjer generirane sintetičke slike oka.




Slika 13: Glavno sučelje UnityEyes-a

Prilikom pritiska tipke *S* se u mapu *imgs* generiraju dva podatka: .jpg sintetička slika oka nalik onoj na slici 13 i .json datoteka koja sadrži razne podatke o karakteristikama generirane sintetičke slike. Obje generirane datoteke su istog naziva što olakšava učitavanje podataka. Karakteristike, odnosno atributi sintetičkih očiju koje sadrži generirana .json datoteka su: ključne točke očiju, ključne točke mesnatog izraštaja na oku (eng. *caruncle*), ključne točke šarenice, vektor pogleda, veličina i boja šarenice, veličina zjenice, detaljni podaci o pozadinskoj teksturi osvjetljenja, detalji PCA koeficijenata, tekstura kože i kut rotacije od kamere prema subjektu. Međutim, za potrebe pripreme podataka i treniranja modela, jedina potrebna karakteristika je ona o vektoru pogleda i to samo prve dvije komponente vektora koje su normalizirane između -1 i 1. U nastavku se nalazi struktura .json datoteke koja je generirana uz sliku 13.

```
{
  "interior_margin_2d": [
    "(337.1404, 284.2441, 9.1628)",
    "(333.5033, 293.1083, 9.1920)", ...
    ],
  "caruncle_2d": [
    "(319.7133, 271.5341, 9.2320)",
    "(319.3554, 279.8067, 9.2566)", ...
    ],
  "iris_2d": [
    "(376.0167, 301.2666, 8.8176)",
    "(376.8077, 309.8537, 8.8220)", ...
    ],
  "eye_details": {
```

```
    "look_vec": "(0.2334, 0.0157, -0.9723, 0.0000)",
    "pupil_size": "-0.3242266",
    "iris_size": "1",
    "iris_texture": "eyeball_brown"
    },
  "lighting_details": {
    "skybox_texture": "bergen_2k",
    "skybox_exposure": "1.191213",
    "skybox_rotation": "231",
    "ambient_intensity": "1.103478",
    "light_rotation": "(38.4, 242.9, 0.0)",
    "light_intensity": "0.9918349"
    },
  "eye_region_details": {
    "pca_shape_coeffs": [
      "0.02848689",
      "0.01165411", ...
      ],
    "primary_skin_texture": "m05_color"
    },
  "head_pose": "(0.0000, 180.0000, 0.0000)"
  }
```

### 6.3.1. Generiranje sintetičkih podataka

S obzirom na to da UnityEyes ne nudi API te u svom automatskom modu osim smjera pogleda i kamere ne nudi mogućnost kontrole randomizacije karakteristika subjekta, za svrhu kreiranja skupa podataka je potrebno napraviti skriptu koja će unutar interaktivnog moda nasumično pomicati miš kako bi se na taj način mijenjao smjer pogleda. Problem je u tome što se u automatskom modu svaki puta prilikom generiranja nove slike randomiziraju sve moguće karakteristike. Naime, cilj je generirati 40 različitih pogleda po svakom nasumičnom subjektu, a to nije moguće napraviti u automatskome modu jer će se svakom novom slikom uvijek generirati novi subjekt, pa će zbog toga uvijek postojati maksimalno jedan pogled po subjektu s jedinstvenim karakteristikama. Jedini način na koji je moguće iskoristiti automatski mod je da se nakon generiranja podataka grupiraju podaci koji se nalaze u .json datoteci prema zadanim atributima. Međutim, tu nastaje problem jer većina karakteristika je definirano kao kontinuirana slučajna varijabla, a ne diskretna, pa takvo grupiranje onda nema smisla raditi. Postoji opcija i da se podaci grupiraju prema samo diskretnim varijablama (karakteristikama), no onda takvi podaci nisu savršeni jer nisu najispravnije grupirani te zbog toga značajno otežavaju treniranje modela. U nastavku se nalazi dio skripte u Pythonu koji nasumično pomiče miš i pritom generira podatke, a nakon toga slijedi i opis rada skripte.

```
NO_CONTACT_PATH = os.path.join(os.getcwd(), 'imgs/no_contact')
```

```
CONTACT_PATH = os.path.join(os.getcwd(), 'imgs/contact')
SRC_PATH = 'C:/Users/pcrncec/dipl/UnityEyes/imgs'

CENTER_OFFSET_X = 190
CENTER_OFFSET_Y = 120
X_CENTER, Y_CENTER = get_xy_center()
NORMAL_STD_X = 110
NORMAL_STD_Y = 80

NUM_SAMPLES_PER_SUBJECT = 40
NUM_SUBJECTS = 3200

all_contact_imgs = os.listdir(CONTACT_PATH)
all_subject_id = [int(x.split('-')[0]) for x in all_contact_imgs]
if len(all_subject_id) > 0:
    start_subject_id = max(all_subject_id) + 1
else:
    start_subject_id = 0

for subject_id in range(start_subject_id, NUM_SUBJECTS + start_subject_id):
    click(X_CENTER, Y_CENTER)
    save_image()
    time.sleep(0.15)

    src_base_filename = os.listdir(SRC_PATH)[0].split('.')[0]
    src_img_path = os.path.join(SRC_PATH, src_base_filename + '.jpg')
    src_json_path = os.path.join(SRC_PATH, src_base_filename + '.json')

    dst_img_filename = f'{subject_id}-0.jpg'
    dst_json_filename = f'{subject_id}-0.json'
    dst_img_path = os.path.join(CONTACT_PATH, dst_img_filename)
    dst_json_path = os.path.join(CONTACT_PATH, dst_json_filename)
    shutil.move(src_img_path, dst_img_path)
    shutil.move(src_json_path, dst_json_path)

    for sample_id in range(NUM_SAMPLES_PER_SUBJECT-1):
        lower_bound_x = X_CENTER - CENTER_OFFSET_X
        lower_bound_y = Y_CENTER - CENTER_OFFSET_Y
        upper_bound_x = X_CENTER + CENTER_OFFSET_X
        upper_bound_y = Y_CENTER + CENTER_OFFSET_Y
        random_x = generate_random_point(X_CENTER, NORMAL_DST_STD_X, lower_bound_x,
            upper_bound_x)
        random_y = generate_random_point(Y_CENTER, NORMAL_DST_STD_Y, lower_bound_y,
            upper_bound_y)
        click(random_x, random_y)
        save_image()
        time.sleep(0.15)

        src_base_filename = os.listdir(SRC_PATH)[0].split('.')[0]
        src_img_path = os.path.join(SRC_PATH, src_base_filename + '.jpg')
        src_json_path = os.path.join(SRC_PATH, src_base_filename + '.json')

        dst_img_filename = f'{subject_id}-{sample_id+1}.jpg'
```

```
        dst_json_filename = f'{subject_id}-{sample_id+1}.json'
        dst_img_path = os.path.join(NO_CONTACT_PATH, dst_img_filename)
        dst_json_path = os.path.join(NO_CONTACT_PATH, dst_json_filename)
        shutil.move(src_img_path, dst_img_path)
        shutil.move(src_json_path, dst_json_path)
    randomize_person()
```

Na početku se u varijable postavljaju putanje do direktorija gdje se izvorno generiraju podaci i putanje do direktorija kamo će se pohranjivati svi podaci. Zatim je potrebno postaviti konstante prema kojima će se odrediti mogući rasponi pomaka miša po širini i visini ekrana, a to je spremljeno u varijablama CENTER_OFFSET_X i CENTER_OFFSET_Y koje predstavljaju maksimalni mogući pomak miša od centra UnityEyes-a u pixelima. Uz pomoć funkcije *get_xy_center* se dohvaćaju koordinate sredine UnityEyes prozora.

Konstante NORMAL_STD_X i NORMAL_STD_Y predstavljaju standardne devijacije pomaka miša po širini i visini ekrana. Ove standardne devijacije se koriste prilikom generiranja nasumičnih lokacija miša uz pomoć normalne distribucije. Posljednje konstante su NUM_SAMPLES_PER_SUBJECT i NUM_SUBJECTS koje označavaju broj uzoraka slika koje se generiraju po pojedinom subjektu i ukupni broj generiranih subjekata. Skripta radi na način da se za svaki subjekt najprije klikne na sredinu UnityEyes prozora kako bi se na taj način smjer pogleda pomaknuo na sredinu, a to zapravo označava sliku na kojoj postoji kontakt očima te se onda takve slike i .json datoteke spremaju u poseban direktorij naziva *contact*. Nakon toga se generira preostalih 39 uzoraka pogleda pojedinog subjekta na temelju nasumičnog vektora smjera koji je generiran od strane normalne distribucije te se takvi podaci onda spremaju u direktorij naziva *no_contact*. Kako bi se olakšalo učitavanje podataka, postoji jasno definirana struktura naziva generiranih datoteka. Naziv pojedine datoteke se sastoji od broja, odnosno ID-a subjekta koji se nalazi na slici te broja uzorka subjekta. Razlog takve strukture je taj što prilikom učitavanja podataka je potrebno učitavati parove slika istog subjekta, a kako u .json datoteci ne postoji podatak koji definira broj ili naziv subjekta, najjednostavnije rješenje takvog problema je da se ID subjekta zapiše u naziv datoteke.

## 6.4. Detalji implementacije

U ovome potpoglavlju će se uz programski kod objasniti najvažniji dijelovi implementiranog sustava korekcije položaja očiju. Najvažniji dijelovi koje će biti objašnjeni su: učitavanje podataka, implementacija modela, treniranje modela i glavna skripta koja je zaslužna za učitavanje videa sa kamere, predikciju rezultata korekcije pogleda te vizualizaciju na virtualnoj kameri.

### 6.4.1. Učitavanje podataka

Za učitavanje podataka se koristi klasa naziva DataLoader. Za inicijalizaciju objekta ove klase se koriste parametri contact_path, no_contact_path i batch_size. Contact_path i no_contact_path predstavljaju putanje do slika i .json datoteka. U putanji sadržanoj u varijabli

contact_path se nalaze svi podaci koji sadrže poglede u kojima je naizgled ostvareni kontakt očima, dok se u putanji no_contact_path nalaze preostali podaci. Varijabla batch_size predstavlja broj podataka koji istovremeno kao serijski skup ulazi u model prilikom treniranja.

Prilikom inicijalizacije objekta se inicijaliziraju i varijable: all_imgs_filenames, num_images, all_img_filename_pairs, num_img_pairs, num_batch_steps i all_imgs. Ove varijable se kasnije koriste pri generiranju serije podataka i treniranju modela. Varijabla all_imgs_filenames sadrži listu svih naziva slika, num_images sadrži ukupan broj slika, all_img_filename_pairs sadrži sve moguće parove čiji su elementi nazivi ulaznih i izlaznih slika na temelju kojih se vrši treniranje, num_img_pairs predstavlja ukupan broj tih parova, num_batch_steps predstavlja ukupan broj koraka po epohi, gdje korak predstavlja jedan prolazak serije podataka kroz model, te za kraj varijabla all_imgs predstavlja rječnik čiji su ključevi nazivi slika, dok su vrijednosti objekti slika zapisanih u zasebnom polju. Važno je za napomenuti da varijabla all_imgs ovisno o veličini skupa podataka zauzima puno radne memorije (nekoliko GB). Postoji i alternativni način učitavanja podataka koji ne bi koristio varijablu all_imgs te bi se time smanjila potrošnja radne memorije, no u tom slučaju bi se podaci morali pretprocesirati prilikom samog treniranja, a to bi značajno usporilo proces treniranja što je lošiji kompromis u odnosu na brže treniranje i zauzimanje radne memorije. Programski kod za inicijalizaciju klase DataLoader se nalazi ispod.

```
class DataLoader:
    def __init__(self, contact_path, no_contact_path, batch_size):
        self.CONTACT_PATH = contact_path
        self.NO_CONTACT_PATH = no_contact_path
        self.BATCH_SIZE = batch_size
        self.all_img_filenames = self.get_all_img_filenames(self.NO_CONTACT_PATH) +
            self.get_all_img_filenames(self.CONTACT_PATH)
        self.num_imgs = len(self.all_img_filenames)
        self.all_img_filename_pairs = self.generate_all_img_pairs()
        self.num_img_pairs = len(self.all_img_filename_pairs)
        self.num_batch_steps = self.num_img_pairs // self.BATCH_SIZE
        self.all_imgs = self.preprocess_all_imgs()
```

Prilikom pretprocesiranja podataka, koristi se tehnika augmentacije podataka koja omogućuje nasumično generiranje novih podataka na temelju danih postavki. Konkretnije, za svaku pretprocesiranu sliku postoji vjerojatnost da ona bude augmentirana nekom od metoda ili pak kombinacijom više metoda augmentacije. Metode koje se koriste za augmentaciju su: Gaussov šum, Gaussovo zaglađivanje i podešavanje svjetline i kontrasta slike, a implementirane su u datoteci *image_augmentation.py* i klasi ImageAugmentation. Prilikom inicijalizacije objekta klase ImageAugmentation, nasumično se postavljaju parametri augmentacijskih metoda te se nakon toga za svaku metodu augmentacije na temelju zadanih vjerojatnosti određuje da li će se pojedina metoda nad slikom primijeniti ili ne. Gaussov šum i podešavanje svjetline i kontrasta slike imaju za svaku sliku 20% vjerojatnosti da se primjene, dok Gaussovo zaglađivanje ima 10% vjerojatnosti. Za svaki subjekt se inicijalizira zasebni objekt ImageAugmentation klase kako bi na taj način svaki subjekt imao vlastite postavke augmentacije, a da pritom uzorci unutar istog subjekta nemaju različite postavke. U nastavku se nalazi čitav programski kod klase ImageAugmentation, a sadrži inicijalizacijsku metodu, privatne metode koje primjenjuju svaku vrstu augmentacije i metoda *augment_image* koja na temelju zadane slike i zadanih postavki objekta

primjenjuje pojedinu vrstu augmentacije.

```
import cv2
import numpy as np
import random

class ImageAugmentation:
    def __init__(self):
        # initializing random gaussian noise parameters
        gaussian_noise_var = random.uniform(0.005, 0.01)
        sigma = gaussian_noise_var ** 0.5
        self.noise = np.random.normal(0, sigma, (64, 32, 3))
        self.noise = self.noise.reshape((64, 32, 3))

        # initializing random gaussian blur parameters
        self.blur_ksize = random.choice([3, 5, 7])
        self.blur_sigma = random.uniform(0, 10)

        # initializing random brightness & contrast parameters
        self.brightness = random.uniform(-0.2, 0.1)
        self.contrast = random.uniform(0.8, 1.5)
        if self.contrast > 1.2 and self.brightness < 0:
            self.brightness = abs(self.brightness)
        self.brightness += float((1 - self.contrast) / 2)

        # setting random options of augmentation
        self.should_apply_noise = np.random.choice([True, False], p=[0.2, 0.8])
        self.should_apply_blur = np.random.choice([True, False], p=[0.1, 0.9])
        self.should_adjust_cb = np.random.choice([True, False], p=[0.2, 0.8])

    def _apply_gaussian_noise(self, image):
        return image + self.noise

    def _apply_gaussian_blur(self, image):
        return cv2.GaussianBlur(image, (self.blur_ksize, self.blur_ksize), self.
            blur_sigma)

    def _adjust_contrast_and_brightness(self, image):
        return cv2.addWeighted(image, self.contrast, image, 0, self.brightness)

    def augment_image(self, image):
        if self.should_adjust_cb:
            image = self._adjust_contrast_and_brightness(image)
        if self.should_apply_noise:
            image = self._apply_gaussian_noise(image)
        if self.should_apply_blur:
            image = self._apply_gaussian_blur(image)
        return image
```

Za kraj vrijedi opisati metodu *generator* koja se koristi prilikom treniranja i validacije modela. Svrha te metode je da na temelju veličine serije (varijabla batch_size) vraća skup, odnosno seriju parova podatka, gdje par predstavlja ulaznu sliku i vektor pogleda ulazne slike te izlaznu sliku i vektor pogleda izlazne slike. Na temelju podataka koje vraća metoda genera-

tor model vrši svoje treniranje. Metoda *generator* funkcionira na način da se za svaku zadanu epohu generira nasumični poredak slijeda generiranja, odnosno dohvaćanja podataka, a takvo randomizirano vraćanje podataka iz metode *generator* omogućuje kvalitetnije treniranje modela. U svakom se koraku iz varijable all_img_filename_pairs čita traženi skup podataka te se na temelju dobivenih naziva slika iz varijable all_imgs dohvaćaju odgovarajući podaci koje metoda, tj. generator vraća u svakom svom koraku izvršavanja.

```
def generator(self, num_epochs):
    for epoch in range(num_epochs):
        rand_steps = random.sample(range(self.num_batch_steps), self.num_batch_steps
            )
        random.shuffle(self.all_img_filename_pairs)
        for batch in rand_steps:
            batch_output_angles = np.empty((self.BATCH_SIZE, 2))
            batch_input_angles = np.empty((self.BATCH_SIZE, 2))
            batch_input_imgs = np.empty((self.BATCH_SIZE, 64, 32, 3))
            batch_output_imgs = np.empty((self.BATCH_SIZE, 64, 32, 3))

            batch_img_filenames_pairs = self.all_img_filename_pairs[
                                            batch * self.BATCH_SIZE:(batch + 1)
                                                * self.BATCH_SIZE]
            for i, batch_img_filenames_pair in enumerate(batch_img_filenames_pairs):
                input_img_filename, output_img_filename = batch_img_filenames_pair

                input_img_array, (input_angle_x, input_angle_y) = self.all_imgs[
                    input_img_filename]
                batch_input_imgs[i] = input_img_array
                batch_input_angles[i] = np.array([input_angle_x, input_angle_y])

                output_img_array, (target_angle_x, target_angle_y) = self.all_imgs[
                    output_img_filename]
                batch_output_imgs[i] = output_img_array
                batch_output_angles[i] = np.array([target_angle_x, target_angle_y])
            yield [batch_input_imgs, batch_input_angles], [batch_output_imgs,
                batch_output_angles]
```

### 6.4.2. Implementacija modela strojnog učenja

Za implementaciju modela strojnog učenja se koriste biblioteke TensorFlow i Keras. Keras je biblioteka otvorenog koda koja pruža sučelje za Python programski jezik za svrhu implementacije neuronskih mreža te djeluje kao sučelje za TensorFlow [17].

Prvi korak je kreirati klasu u kojoj će se implementirati konvolucijski blok koji se više puta koristi u modelu. Za tu svrhu se koristi Kerasova mogućnost kreiranja prilagođenog sloja na način da se uz pomoć principa nasljeđivanja naslijedi klasa Layer. Prilikom inicijalizacije objekta klase se inicijaliziraju objekti svih slojeva koji se koriste u konvolucijskom bloku, a to su: dubinsko-separabilni konvolucijski slojevi (SeparableConv2D), serijska normalizacija (BatchNormalization) i maksimalno sažimanje (MaxPooling2D). Nakon toga se implementira metoda call koja se poziva prilikom treniranja i inferencije modela. Ona kao svoj argument prima ulazne

podatke u konvolucijski blok te je u njoj definiran slijed pozivanja objekata, odnosno slojeva koji su definirani prilikom inicijalizacije. Drugim riječima, u toj metodi se zapravo implementira sama arhitektura konvolucijskog bloka. Programski kod za konvolucijski blok je sljedeći:

```
from keras.layers import SeparableConv2D, BatchNormalization, ReLU, Layer, Add,
    MaxPooling2D

class ConvBlock(Layer):
    def __init__(self, filters, kernel_size=3, use_max_pooling=False):
        super(ConvBlock, self).__init__()
        self.filters = filters
        self.kernel_size = kernel_size
        self.use_max_pooling = use_max_pooling
        self.conv1 = SeparableConv2D(self.filters, self.kernel_size, padding='same')
        self.bn1 = BatchNormalization()

        self.conv2 = SeparableConv2D(self.filters, self.kernel_size, padding='same')
        self.bn2 = BatchNormalization()

        self.conv3 = SeparableConv2D(self.filters, self.kernel_size, padding='same')
        self.bn3 = BatchNormalization()

        if self.use_max_pooling:
            self.max_pooling = MaxPooling2D()

    def call(self, inputs, *args, **kwargs):
        block_1 = self.conv1(inputs)
        block_1 = self.bn1(block_1)
        block_1 = ReLU()(block_1)

        block_2 = self.conv2(block_1)
        block_2 = self.bn2(block_2)
        block_2 = ReLU()(block_2)

        block_12 = Add()([block_1, block_2])

        block_3 = self.conv3(block_12)
        block_3 = self.bn3(block_3)
        block_3 = ReLU()(block_3)

        if self.use_max_pooling:
            block_3 = self.max_pooling(block_3)

        return block_3
```

Nakon definiranja konvolucijskog bloka potrebno je još definirati prilagođene slojeve koji će vršiti iskrivljenje slike i podesiti svjetlinu slike. Zapravo se radi o posljednjim koracima koji se izvršavaju unutar modela strojnog učenja. Sloj zadužen za iskrivljenje slike je jednostavan za implementirati jer TensorFlow već nudi funkciju koja to omogućuje, a to je funkcija *dense_image_warp* koja kao ulaz očekuje sliku i vektorsko polje. Podešavanje svjetline je

moguće implementirati na način da se nad slikom primjeni matematička funkcija:

$$S_{izlaz} = S_{ulaz}(x, y, c) \cdot (1 - M(x, y)) + M(x, y)$$

gdje je $S_{izlaz}$ izlazna slika nakon primjene funkcije, $S_{uzlaz}$ ulazna slika nad kojom se želi podesiti svjetlina, a $M$ mapa svjetline koja utječe na jačinu podešavanja svjetline te se sastoji od vrijednosti u rasponu od 0 do 1, što je osigurano primjenom sigmoidne funkcije u prethodnom sloju. U nastavku se nalazi programski kod koji implementira ta dva sloja [18].

```
from tensorflow_addons.image import dense_image_warp
from keras.layers import Layer
import tensorflow as tf

class DenseImageWarper(Layer):
    def __init__(self):
        super(DenseImageWarper, self).__init__()

    def call(self, inputs, *args, **kwargs):
        input_img, vector_field = inputs
        return dense_image_warp(input_img, vector_field)

class BrightnessAdjuster(Layer):
    def __init__(self):
        super(BrightnessAdjuster, self).__init__()

    def call(self, inputs, *args, **kwargs):
        warped_img, brightness_map = inputs
        one = tf.constant(1, dtype=tf.float32)
        x = tf.subtract(one, brightness_map)
        x = tf.multiply(warped_img, x)
        x = tf.add(x, brightness_map)
        return x
```

Preostali slojevi koji se nalaze u arhitekturi modela koji je opisan u poglavlju 4.4. su već implementirani u biblioteci Keras. Prema tome, upotrebom Kerasovih slojeva te prethodno opisanih prilagođenih slojeva je moguće implementirati konačni model na slični način kao što je implementirani konvolucijski blok, a sljedeći arhitekturu na slici 6. Programski kod koji implementira model strojnog učenja se nalazi ispod.

```
import tensorflow as tf
from keras.models import Model
from keras.layers import Input, concatenate, Conv2DTranspose, Conv2D, Activation
from conv_block import ConvBlock
from custom_layers import DenseImageWarper, BrightnessAdjuster

class ECCNet(Model):
    def __init__(self):
        super(ECCNet, self).__init__()
        self.conv_block_1 = ConvBlock(32, use_max_pooling=True)
        self.conv_block_2 = ConvBlock(64, use_max_pooling=True)
        self.conv_block_3 = ConvBlock(128, use_max_pooling=True)
        self.conv_block_4 = ConvBlock(256, use_max_pooling=True)
```

```
        self.conv_block_5 = ConvBlock(128)
        self.up_conv_1 = Conv2DTranspose(128, 3, padding='same', strides=2)
        self.conv_block_6 = ConvBlock(64)
        self.up_conv_2 = Conv2DTranspose(64, 3, padding='same', strides=2)
        self.conv_block_7 = ConvBlock(32)

        self.up_conv_3 = Conv2DTranspose(32, 2, padding='same', strides=2)
        self.up_conv_4 = Conv2DTranspose(32, 2, padding='same', strides=2)
        self.final_conv = Conv2D(3, 3, padding='same')
        self.sigmoid = Activation('sigmoid')

        self.dense_image_warp = DenseImageWarper()
        self.brightness_adjuster = BrightnessAdjuster()

    def call(self, inputs):
        input_img, target_angle = inputs
        target_angle_tile = tf.reshape(target_angle, (-1, 1, 1, 2))
        target_angle_tile = tf.tile(target_angle_tile, (1, 64, 32, 1))
        img_angle_concat = concatenate([input_img, target_angle_tile])
        out_conv_1 = self.conv_block_1(img_angle_concat)

        out_conv_2 = self.conv_block_2(out_conv_1)
        out_conv_3 = self.conv_block_3(out_conv_2)
        out_conv_4 = self.conv_block_4(out_conv_3)

        out_conv_5 = self.conv_block_5(out_conv_4)
        out_up_conv_1 = self.up_conv_1(out_conv_5)
        out_concat_1 = concatenate([out_up_conv_1, out_conv_3])
        out_conv_6 = self.conv_block_6(out_concat_1)
        out_up_conv_2 = self.up_conv_2(out_conv_6)
        out_concat_2 = concatenate([out_up_conv_2, out_conv_2])
        out_conv_7 = self.conv_block_7(out_concat_2)

        final_layer = self.up_conv_3(out_conv_7)
        final_layer = self.up_conv_4(final_layer)
        final_layer = self.final_conv(final_layer)

        vector_field = final_layer[..., 0:2]
        brightness_map = self.sigmoid(final_layer[..., 2:3])
        warped_img = self.dense_image_warp([input_img, vector_field])
        output_img = self.brightness_adjuster([warped_img, brightness_map])
        return output_img

    def create_model(self, input_img_dim=(64, 32, 3)):
        input_img = Input(input_img_dim)
        target_angle = Input(2)
        model = Model(inputs=[input_img, target_angle], outputs=self([input_img,
            target_angle]), name="ECCNet")
        return model
```

### 6.4.3. Treniranje modela strojnog učenja

Kao i implementacija modela, za treniranje modela se koriste bilblioteke TensorFlow i Keras. Keras već nudi metodu *fit* uz pomoć koje se omogućuje treniranje prosljeđivanjem potrebnih parametara. Međutim, s obzirom na to da je potrebno model trenirati na dvosmjeran način, metodu *fit* nije moguće koristiti već je potrebno implementirati vlastito prilagođeno treniranje. Kod prilagođenog treniranja je potrebno implementirati funkciju koja će u neuronskoj mreži vršiti algoritme širenje unaprijed i unatrag. Za tu potrebu, koristi se TensorFlowova klasa GradientTape koja implementira automatsku diferencijaciju te klasa željenog optimizatora. Optimizator je zapravo algoritam koji izvršava gradijentni spust. Odabrani je optimizator Adam koji je nadogradnja gradijentnog spusta. Prilikom inicijalizacije optimizatora, važno je podesiti stopu učenja modela. Testirane su različite stope učenja, a kao najbolja se pokazala stopa učenja od 0.005 u prvih 50 epoha te stopa učenja od 0.0001 u preostalim epohama. Funkcija koja kod treniranja implementira korake širenja unaprijed i unatrag se nalazi ispod. Vidljivo je i da je iznad funkcije definirani dekorator *@tf.function*. On omogućava treniranje u graf modu, dok bi se izostavljanjem dekoratora treniranje vršilo u eager modu. Graf mod je specijaliziran način rada koji umjesto standardnog izvršavanja operacije po operaciju koristi TensorFlow-ove grafove. Takav način rada omogućuje TensorFlowu da radi značajno brže i paralelno [19].

```
@tf.function
def train_step(input_imgs, input_angles, output_imgs, output_angles):
    with tf.GradientTape() as tape:
        pred_output_imgs = model([input_imgs, output_angles], training=True)
        correction_loss = loss_fn(output_imgs, pred_output_imgs)
        pred_input_imgs = model([pred_output_imgs, input_angles], training=True)
        reconstruction_loss = loss_fn(input_imgs, pred_input_imgs)
        total_train_loss = total_loss_fn(correction_loss, reconstruction_loss)
    grads = tape.gradient(total_train_loss, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
    return total_train_loss
```

### 6.4.4. Glavna skripta

Glavna skripta je datoteka čijim se pokretanjem pokreće sustava korekcije pogleda. Zadaća ove skripte je u stvarnom vremenu čitati ulazne podatke (slike) iz dostupne kamere, pa takve slike proslijediti modelu detektora očiju, zatim detektirane oči proslijediti modelu koji korigira pogled i na kraju na sliku s korigiranim pogledom proslijediti virtualnoj kameri. Osim toga, postoje međukoraci poput pretprocesiranja slike očiju prije nego što one budu proslijeđene modelu korekcije pogleda te postprocesiranje korigirane slike očiju kako bi se uspješno mogle zamijeniti oči izvornog pogled a sa očima korigiranog pogleda. Također, s obzirom na to da biblioteka Mediapipe i njezin model detektora ključnih točaka lica može kao rezultat izbaciti negativne koordinate očiju, potrebno je čitavo vrijeme provjeravati da li su dobivene koordinate valjane te da li su uopće na slici detektirane oči. U nastavku će se objasniti neki od najvažnijih dijelova glavne skripte.

Na početku skripte potrebno je kreirati objekt modela korekcije pogleda te učitati najbolje

težine modela metodom *load_weights*. U varijabli WEIGHTS_NAME je zapisan naziv željene težine koje su bile pohranjene prilikom treniranja modela.

```
WEIGHTS_NAME = 'model_weights-835-0.0015.hdf5'
weights_path = os.path.join(os.getcwd(), 'model_checkpoints', WEIGHTS_NAME)
model = ECCNet().create_model()
model.load_weights(weights_path)
```

Prije nego li se krene u bilo kakvu detekciju potrebno je kreirati objekt virtualne kamere te podesiti širinu, visinu i broj slika u sekundi (fps):

```
with pyvirtualcam.Camera(width=640, height=480, fps=30) as cam
```

Zatim je moguće uz pomoć biblioteke OpenCV kreirati objekt koji će omogućiti čitanje slika dobivene iz kamere:

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

Sljedeći korak je kreirati Mediapipe objekt detektora ključnih točaka lica sa definiranim parametrima koji uključuju maksimalni broj detektiranih lica i minimalnu razinu samopovjerenja modela u svoje rezultate detekcije:

```
with mp_face_mesh.FaceMesh(
                max_num_faces=1,
                refine_landmarks=True,
                min_detection_confidence=0.5,
                min_tracking_confidence=0.5
        ) as face_mesh .
```

Nakon toga je metodom *read* objekta VideoCapture moguće čitati slike dobivene iz kamere:

```
ret, input_frame = cap.read().
```

Za detekciju ključnih točaka lica i ključnih točaka očiju se koriste sljedeće naredbe:

```
results = face_mesh.process(rgb_frame)
if results.multi_face_landmarks:
    mesh_points = np.array([np.multiply([p.x, p.y], [img_w, img_h]).astype(int)
                            for p in results.multi_face_landmarks[0].landmark])
    l_min_x, l_min_y = min(mesh_points[LEFT_EYE][:, 0]), min(mesh_points[LEFT_EYE
        ][:, 1])
    l_max_x, l_max_y = max(mesh_points[LEFT_EYE][:, 0]), max(mesh_points[LEFT_EYE
        ][:, 1])
    r_min_x, r_min_y = min(mesh_points[RIGHT_EYE][:, 0]), min(mesh_points[RIGHT_EYE
        ][:, 1])
    r_max_x, r_max_y = max(mesh_points[RIGHT_EYE][:, 0]), max(mesh_points[RIGHT_EYE
        ][:, 1])
```

Varijabla results sadrži koordinate ključnih točaka lica koje su dobivene pozivanjem metode process i prosljeđivanjem objekta slike. Nakon toga se provjerava da li postoji barem jedna detektirana ključna točka lica te ako postoji, na temelju predefiniranih indeksa ključnih točaka očiju (LEFT_EYE i RIGHT_EYE) se dohvaćaju njezine koordinate. Dobivene su koordinate od najmanje i najveće ključne točke po širini i dužini slike na oboje lijevom i desnom oku. Nadalje, na temelju dobivenih koordinata se iz slike dobivene iz kamere izrežu oči koje se zatim prosljeđuju modelu koji je zadužen za korekciju položaja očiju (korekciju pogleda). Kako model prima samo sliku jednog oka, potrebno je svako oko zasebno proslijediti u model i to na način da se slika lijevog oka zrcali, dok slika desnog oka ostane onakva kakva je. Osim slika očiju, u model je potrebno proslijediti vektor smjera pogleda. S obzirom na to je cilj korigirati oči prema centru kako bi se na taj način ostvario prividni kontakt očima, elementi vektora smjera pogleda će biti jednaki nulama. Slike očiju u model korekcije pogleda je moguće proslijediti na način da se jednostavno pozove objekt modela, a kao rezultat će se vratiti izlazna slika, tj. slika s korigiranim pogledom. Primjerice kako bi se dobila korigirana slika desnog oka, koriste se sljedeće naredbe:

```
l_cropped_eye = cv2.flip(l_cropped_eye, 1)
batch_target_angle = np.zeros((1, 2))
l_img_array = utils.preprocess_image(l_cropped_eye)
l_img_batch = np.expand_dims(l_img_array, axis=0)
r_img_array = utils.preprocess_image(r_cropped_eye)
r_img_batch = np.expand_dims(r_img_array, axis=0)
l_pred = model([l_img_batch, batch_target_angle])[0]
```

Za kraj, kako bi se slika s korigiranim očima proslijedila virtualnoj kameri kojeg koriste alati za video komunikaciju, koristi se objekt virtualne kamere i metoda *send*:

```
virtual_cam.send(corrected_frame).
```

## 6.5. Rezultati i evaluacija modela

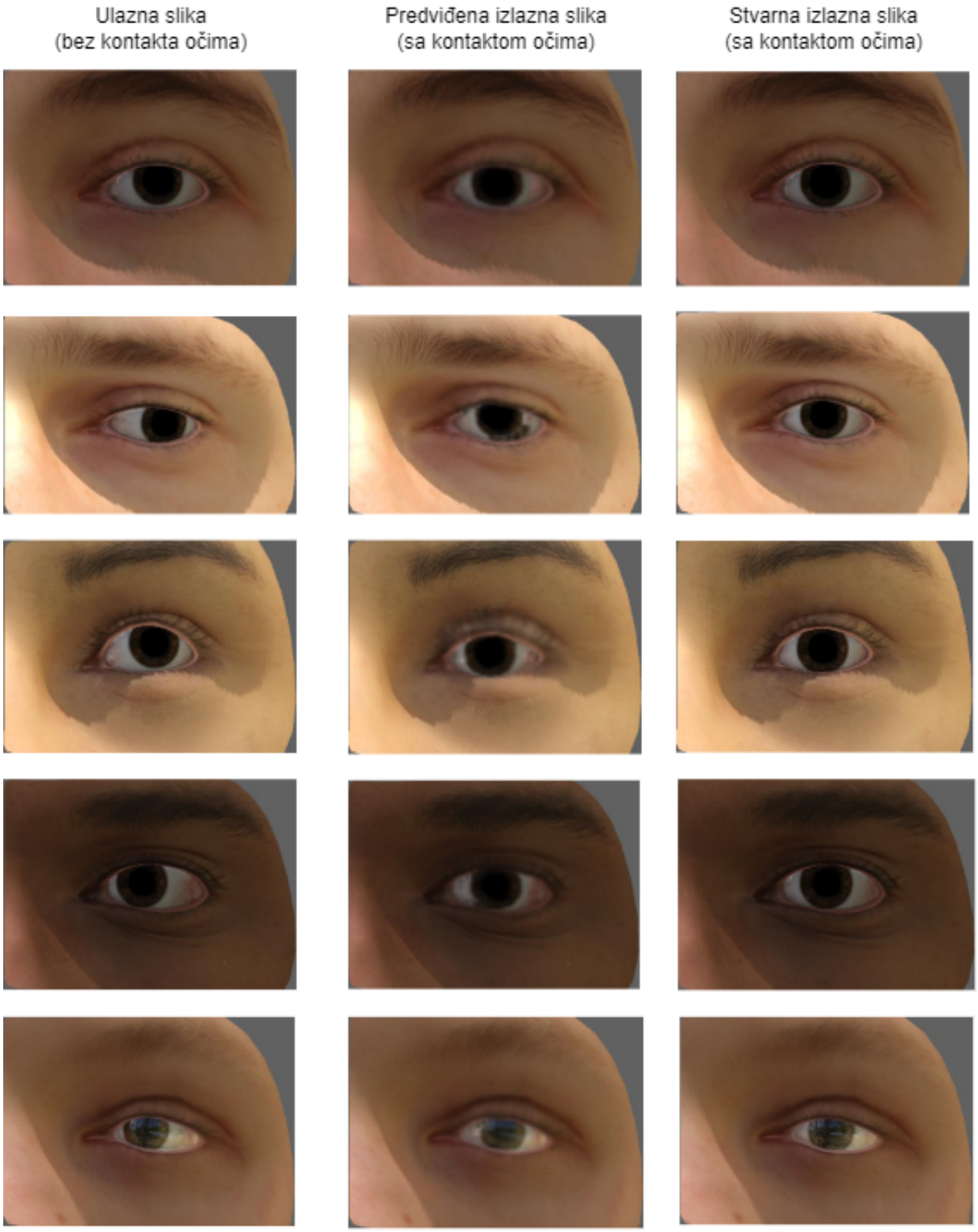
Model za korekciju pogleda je bio treniran na 900 epoha, s time da je prvih 700 epoha trenirao na manjem skupu podataka od nekoliko tisuća slika. Najbolje rezultate je model imao na epohi 843. Na toj epohi je model imao najmanju validacijsku grešku, što znači da je u toj epohi model najbolje naučio generalizirati svoja predviđanja (korekciju pogleda). Nakon toga se validacijska greška nije smanjivala, već se polako povećavala, dok se greška treniranja smanjivala. U takvim slučajevima kada greška treniranja pada a validacijska greška raste se radi o prekomjernom prilagođavanju podacima (eng. *overfitting*). To znači da model prestaje dobro generalizirati, tj. kroz svoje treniranje se previše oslanja na podatke za treniranje te postoji mogućnost da će imati puno bolje rezultate nad podacima za treniranje u odnosu na podatke za validaciju. Na 843. epohi je model imao grešku treniranja 0.0017, dok je validacijska greška iznosila 0.0023.

Što se tiče podjele podataka, 90% podataka se koristilo za treniranja, a 10% za validaciju modela. Vrijedi i napomenuti da iako je za treniranje bilo odvojeno 115 200 slika na kojima se nalazi 2880 različitih subjekata, model je dvosmjerno trenirao na svim kombinacijama po-

gleda pojedinog subjekta. Drugim riječima, za svaki subjekt su se generirali svi mogući parovi ulaznih i izlaznih slika. Zbog toga, model je trenirao na čak $\binom{40}{2} * 2880 * 2 = 4492800$ različitih parova slika. Takvo treniranje je na grafičkoj kartici NVIDIA GeForce GTX 1650 SUPER trajalo oko 3 tjedna.

Upotreba sustava korekcije pogleda ne utječe znatno na performanse video komunikacije. Konkretnije, za testiranje se koristio laptop s procesorom Intel Core i5-1135G7 te se zbog primjene sustava korekcije pogleda broj slika u sekundi (fps) smanjio u prosjeku za maksimalno 5. Stoga, dobivene performanse nisu uopće predstavljale problem da se omogući obrada videa u stvarnom vremenu te razlika u performansama između primjene sustava korekcije pogleda i bez njegove primjene je najčešće bila neprimjetna.

Na slici ispod se nalaze dobiveni rezultati nad podacima za validaciju. Vidljivo je da neki podaci imaju lošije rezultate kao što je recimo subjekt u drugom redu, dok neki podaci bolje kao recimo subjekt u prvom redu. Rezultat predikcije korekcije pogleda mnogo ovisi i o izvornom smjeru pogleda ulazne slike. Naime, što je smjer pogleda bliži smjeru kod kojeg postoji kontakt očima, to su rezultati bolji i sadrže manje nepoželjnih artefakata. Primjer nepoželjnih artefakata se nalazi na subjektu u drugom redu kod kojeg u donjem dijelu oka postoji vidljivi artefakt. Iz toga se može zaključiti da još postoji prostora za poboljšanje modela, no on je definitivno naučio koje su potrebne karakteristike očiju na temelju kojeg korigira smjer pogleda. Vrijedi još napomenuti da eventualna vidljiva zamućenja na predviđenim izlaznim slikama postoje ponajviše zbog povećanja rezolucija slika oka u odnosu na originalnu predviđenu izlaznu sliku oka koja je znatno manjih dimenzija (64x32). U stvarnoj primjeni takva zamućenja nisu vidljiva, što je i vidljivo na slici 15.

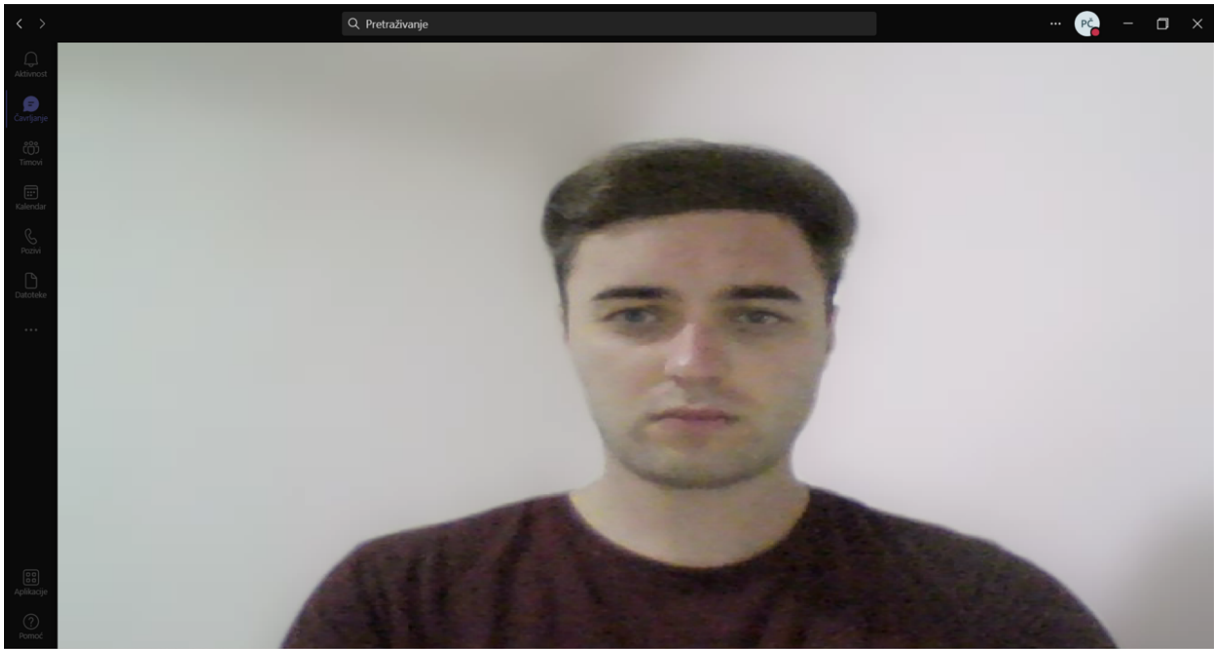



Slika 14: Rezultati modela nad validacijskim skupom podataka

## 6.6. Korištenje sustava

Preduvjet za korištenje sustava korekcije pogleda je da se instalira program OBS koji je besplatno dostupan na adresi: https://obsproject.com/. Razlog ovog preduvjeta je što OBS

nudi instalaciju virtualne kamere, a tu virtualnu kameru koristi biblioteka pyvirtualcam. Prilikom korištenja sustava nije potrebno pokretati OBS, već je dovoljno samo instalirati virtualnu kameru. Sljedeći preduvjet je instalirati sve potrebne biblioteke. Instalaciju svih biblioteka je moguće izvršiti naredbom u terminalu: pip install -r requirements.txt. Za kraj, potrebno je pokrenuti glavni skriptu main.py naredbom: python main.py. Za vrijeme kad je skripta main.py pokrenuta, na virtualnoj kameri naziva OBS Virtual Camera će se prikazivati slike dobivene kamerom s uključenom korekcijom pogleda. Na sljedećoj slici je primjer prikaza rada sustava korekcije pogleda na video komunikacijskom alatu Microsoft Teams.




Slika 15: Primjer rada korekcije pogleda na MS Teams

# 7. Budući rad

Iako su postignuti svi zadani ciljevi, postoji još mnogo prostora za poboljšanje implementiranog sustava. Kao prvo, postoji opcija da se za treniranje koriste i drugi podaci, ili pak isti podaci, ali modificirani od strane generativne suparničke mreže čime bi se stvorili realističniji podaci kao što je to već prikazano u istraživanju koje je analizirano u poglavlju 4.4. Drugo, postoji opcija primjene i nekih drugih modela strojnog učenja ili pak nadogradnja već postojećeg. Novi modeli bi stvorili mogućnost poboljšanja već postojećih performansi te precizniju korekciju pogleda. Treće, zbog ograničenja biblioteke *pyvirtualcam*, trenutni sustav korekcije pogleda je samo moguće pokrenuti na Windows operacijskom sustavu, pa bi bilo dobro da se koriste i drugi pristupi korištenja virtualne kamere kako bi ovaj sustav korekcije pogleda bio dostupan i na Linuxu te macOS-u. Nadalje, za buduća treniranja modela bi bilo bolje da se za validaciju umjesto sintetičkih podataka koristi nekih od prirodnih skupa podataka poput primjerice Columbia Gaze Data Seta. Finalno, trenutno je implementirani sustav koji radi samo na stolnim računalima i laptopima. S obzirom na to da danas mnogo neprofesionalnih video razgovora se vrši i preko pametnih mobilnih uređaja, bilo bi korisno proširiti sustav i na takve uređaje. Međutim, mobilni uređaji u prosjeku imaju značajno slabije performanse u odnosu na računala što može onemogućiti ili barem smanjiti praktičnost sustava korekcije pogleda.

Prema tome, u budućnosti ima smisla da se ovaj sustav i dalje razvija u cilju da korekcija pogleda u svrhu ostvarenja prividnog kontakta očima bude lako dostupna na svim uređajima koji podržavaju video komunikaciju te da rezultati korekcije pogleda budu što prirodniji.

# 8. Zaključak

U ovom radu je najprije bila definirana hipoteza vezana uz mogućnost primjene umjetne inteligencije u svrhu postizanja prividne stvarnosti prirodnog kontakta u video komunikaciji. Ova hipoteza je ostvarena i potvrđena kroz ranije prikazane i opisane rezultate, a također su postignuti i svi definirani ciljevi. Za ostvarenje ciljeva rada se koristila tehnologija strojnog učenja i konvolucijskih neuronskih mreža. Na razmotrenim relevantnim istraživanjima je dokazano da primjena strojnog učenja na problematiku korekcije pogleda daje najbolje i najpraktičnije rezultate, dok je analizom razmotrenih istraživanja utvrđeno da će se za daljnji razvoj sustava korekcije pogleda koristiti predložene metode i model dan u istraživanju od strane Intelovih znanstvenika. Osim analize dosadašnjih relevantnih istraživanja na temu korekcije položaja oka, analizirana su i alternativna aplikacijska rješenja. Zaključak takve analize je da za vrijeme pisanja ovog rada ne postoji javno dostupno i besplatno programsko rješenje koje bi omogućilo ostvarenje prividnog kontakta očima, neovisno o korištenom uređaju i alatu za video komunikaciju. Zbog toga, fokus ovog rada je bio da se omogući korekcija pogleda u svrhu ostvarenja prirodnog kontakta te da takvo rješenje bude primjenjivo na svakom video komunikacijskom alatu, što je i ostvareno uz pomoć primjene virtualne kamere. Međutim, postoji još mnogo prostora za unaprjeđenje implementiranog sustava. Osim mogućnosti da se unaprijedi kvaliteta i performanse korekcije pogleda, bilo bi od koristi da se omogući primjena ovog sustava i na mobilne uređaje čime bi domena primjene sustava korekcije pogleda bila kompletna.

# Popis literature


[1] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*. MIT Press, 2016., http://www.deeplearningbook.org.

[2] G. Khanzode, *Machine Learning*, [Slika] (30.04.2015.). [Na internetu]. [pristupano 30.05.2022.]. adresa: https://www.slideshare.net/GirishKhanzode/supervised-learning-52218215.

[3] N. Buduma i N. Locascio, *Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms*. O'Reilly Media, 2017., ISBN: 9781491925614. adresa: https://books.google.hr/books?id=EZFfrgEACAAJ.

[4] Prabhu, „Understanding of Convolutional Neural Network (CNN) — Deep Learning," (04.03.2018.). [Na internetu]. [pristupano 16.08.2022.]. adresa: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[5] K. Ganguly, *Learning Generative Adversarial Networks: Next-Generation Deep Learning Simplified*. Packt Publishing, 2017., ISBN: 1788396413.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza i dr., *Generative Adversarial Networks*, 2014. DOI: 10.48550/ARXIV.1406.2661. adresa: https://arxiv.org/abs/1406.2661.

[7] J. Brownlee, „A Gentle Introduction to Generative Adversarial Networks (GANs)," (17.06.2019.). [Na internetu]. [pristupano 17.08.2022.]. adresa: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/.

[8] F. Isikdogan, T. Gerasimow i G. Michael, „Eye contact correction using deep neural networks," *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020., str. 3318–3326.

[9] D. Kononenko i V. Lempitsky, „Learning to look up: Realtime monocular gaze correction using machine learning," lipanj 2015., str. 4667–4675.

[10] C.-F. Hsu, Y.-S. Wang, C.-L. Lei i K.-T. Chen, „Look at Me! Correcting Eye Gaze in Live Video Communication," *ACM Trans. Multimedia Comput. Commun. Appl.*, sv. 15, br. 2, lipanj 2019. adresa: https://doi.org/10.1145/3311784.

[11] K. Otsu, M. Seo, T. Kitajima i Y.-W. Chen, „Automatic Generation of Eye Gaze Corrected Video Using Recursive Conditional Generative Adversarial Networks," *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, 2020., str. 674–677.

[12] 4tiitoo, *Eye Contact in Video Conference*, [Na internetu]. [pristupano 30.05.2022.]. adresa: https://www.4tiitoo.com/nuia-full-focus-for-eye-contact-in-video-calls.

[13] B. Lovejoy, *FaceTime eye contact correction in iOS 13 uses ARKit*, [Na internetu]. [pristupano 30.05.2022.]. adresa: https://9to5mac.com/2019/07/03/facetime-eye-contact-correction-in-ios-13-uses-arkit/.

[14] Microsoft, *Make a more personal connection with Eye Contact, now generally available*, [Na internetu]. [pristupano 30.05.2022.]. adresa: https://blogs.windows.com/devices/2020/08/20/make-a-more-personal-connection-with-eye-contact-now-generally-available/.

[15] F. Chollet, „Xception: Deep Learning with Depthwise Separable Convolutions," *CoRR*, sv. abs/1610.02357, 2016. arXiv: 1610.02357. adresa: http://arxiv.org/abs/1610.02357.

[16] E. Wood, T. Baltrušaitis, L.-P. Morency, P. Robinson i A. Bulling, „Learning an Appearance-Based Gaze Estimator from One Million Synthesised Images," *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research  Applications*, 2016., str. 131–138.

[17] Wikipedia, *Keras*, (bez dat.). [Na internetu]. [pristupano 26.08.2022.]. adresa: https://en.wikipedia.org/wiki/Keras.

[18] D. Kononenko i V. Lempitsky, „Learning to look up: Realtime monocular gaze correction using machine learning," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015., str. 4667–4675. DOI: 10.1109/CVPR.2015.7299098.

[19] TensorFlow, *Introduction to graphs and tf.function*, (bez dat.). [Na internetu]. [pristupano 26.08.2022.]. adresa: https://www.tensorflow.org/guide/intro_to_graphs.

# Popis slika