Implementacija i evaluacija odabranih modela strojnog učenja u mobilnom okruženju: Od teorije do prakse

Vukosav, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Marko Vukosav

IMPLEMENTACIJA I EVALUACIJA ODABRANIH MODELA STROJNOG UČENJA U MOBILNOM OKRUŽENJU: OD TEORIJE DO PRAKSE

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marko Vukosav

Matični broj: 0016139927

Studij: Baze podataka i baze znanja

IMPLEMENTACIJA I EVALUACIJA ODABRANIH MODELA STROJNOG UČENJA U MOBILNOM OKRUŽENJU: OD TEORIJE DO PRAKSE

DIPLOMSKI RAD

Mentor :

Doc. dr. sc. Bogdan Okreša Đurić

Varaždin, rujan 2024.

Marko Vukosav

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

Teorijski dio rada pruža pregled modela strojnog učenja u kontekstu mobilnih tehnologija i njihovu primjenjivost u modernim mobilnim uređajima. Osnova praktičnog dijela rada je razvoj vlastitog modela strojnog učenja za specifičnu primjenu u mobilnoj aplikaciji, kao što je prepoznavanje objekata ili obrada govora te njegova usporedba s postojećim modelima strojnog učenja sličnih značajki. Testiranjem i evaluacijom modela, provodi se usporedba s odabranim postojećim gotovim modelima kako bi se ocijenila efikasnost i primjenjivost razvijenog modela u stvarnim mobilnim okruženjima.

Ključne riječi: Strojno učenje, modeli, mobilno okruženje, prepoznavanje lica, razvoj modela strojnog učenja

Sadržaj

1. Uvod

Rad se bavi razvojem modela za prepoznavanje lica pomoću strojnog učenja te implementacijom mobilne aplikacije u svrhu prijave prisutnosti studenata na kolegijima. Uvođenje tehnologije prepoznavanja lica u obrazovne institucije može unaprijediti postojeće metode praćenja prisutnosti, koje su često neefikasne, sklone greškama i raznim prevarama. Tradicionalne metode, poput popisivanja na papir, često odvlače pažnju studenata od predavanja, jer su prisiljeni pratiti listu kako ne bi propustili potpis ili ometati druge ukoliko zaborave pribor za potpisivanje. To smanjuje koncentraciju i ometa tijek predavanja. Automatizirani sustavi, s druge strane, nude brže, preciznije i sigurnije rješenje. Cilj ovog rada je razvoj, implementacija i evaluacija takvog sustava unutar mobilne aplikacije, s ciljem poboljšanja točnosti i učinkovitosti prijave prisutnosti u obrazovnim ustanovama. Rad će također obuhvatiti analizu performansi sustava kako bi se osigurala njegova primjenjivost u stvarnim uvjetima korištenja.

2. Osnove i tehnike strojnog učenja

U ovom poglavlju detaljno će se proći osnovni pojmovi i tehnike strojnog učenja. Razmotrit će se kako različiti algoritmi strojnog učenja omogućuju detekciju lica, ekstrakciju značajki i klasifikaciju, koji su neophodni koraci za izgradnju modela. Poseban fokus će biti na konvolucijske neuronske mreže(engl. Convolution neural networks (CNN)) gdje će se detaljno razmotriti sama arhitektura CNN-ova i na predtrenirane modele poput modela grupa vizualne geometrije sa 16 slojeva (engl. Visual Geometry Group with 16 layers (VGG16)) i modela inverznih rezidualnih blokova i linearnih uskih grla (engl. Inverted Residuals and Linear Bottlenecks, MobileNetV2) te metrike za evaluaciju performansi modela.

2.1. Strojno učenje

Strojno učenje je grana umjetne inteligencije koja računalima pruža sposobnost da uče iz prošlih iskustava, slično kao što to čine ljudi.[1] Naime, umjesto da se računalnim sustavima unaprijed programiraju specifične upute za rješavanje problema, strojno učenje omogućuje računalima da sama prepoznaju obrasce i donose odluke na temelju prikupljenih podataka, što im omogućuje fleksibilnost i sposobnost za prilagodbu različitim zadacima. Hijerarhijski odnos između umjetne inteligencije, strojnog učenja i dubokog učenja prikazan je Vennovim dijagramom na slici 1.

Slika 1: Venn-ov dijagram, preuzeto iz [2]

Temeljna ideja strojnog učenja je da računalni sustavi mogu poboljšati svoje performanse analizirajući i učeći iz velikih skupova podataka. Ti podaci mogu biti skupljeni iz prirode, od strane ljudi, ili stvoreni od drugih algoritama. Uz pomoć tih informacija tvori se osnova za stvaranje modela koji onda omogućuje sustavima da rješavaju probleme i donose odluke u stvarnom svijetu. Strojno učenje se može definirati kao proces rješavanja problema kroz pri-

kupljanje podataka i izgradnju modela na temelju tih podataka[3]. Dakle, cilj strojnog učenja je stvoriti učinkovite i precizne prediktivne algoritme koji su sposobni rješavanju nekog problema. Kvaliteta podataka nad kojim model trenira je izuzetno bitna jer kvalitetniji podaci modelu omogućuju učinkovitije učenje, može se reći kako je strojno učenje usko povezano s analizom podataka i statistikom.

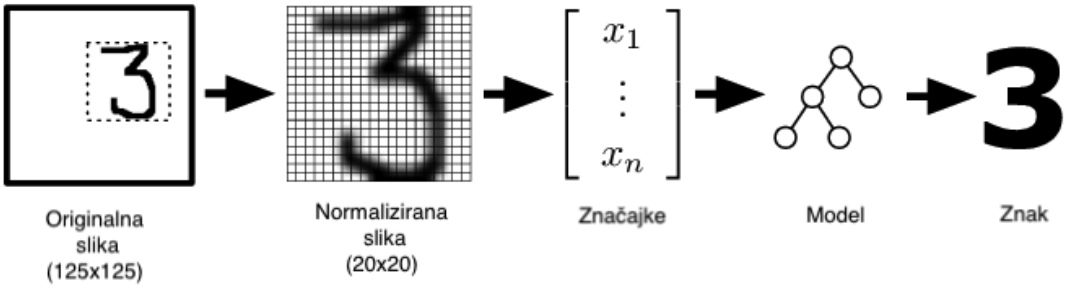
2.1.1. Primjena strojnog učenja

Danas primjena strojnog učenja se može pronaći svugdje. Ona omogućuje automatizaciju složenih zadatke, na način da prepoznaje obrasce u podacima preko značajki pomoću čega se i donose odluke. Primjeri strojnog učenja se kreću od svakodnevnih aktivnosti do posebnih istraživanja.

Jedan od primjera strojnog učenja je i prepoznavanje rukopisa. Najosnovniji primjer je pretvorba prezentacija u pdf format nad kojim se može pretraživati, gdje se koristi tehnologija poput optičkog prepoznavanja znakova (OCR), gdje računalni sustavi prepoznaju i konvertiraju slike ili skenirane dokumente u strojno čitljiv tekst. Isto tako, primjer prepoznavanja rukopisa se najčešće može vidjeti u bankarskim sustavima za automatsko čitanje čekova, kreiranje digitalnih skripti i materijala te u mnogim drugim područjima gdje je potrebno brzo i precizno prepoznavanje pisanog teksta.

Proces prepoznavanja rukopisa [4] (slika 2) započinje skeniranjem rukom pisanog znaka, čime se dobiva digitalizirana slika rukopisa. Ta slika zatim prolazi kroz nekoliko faza obrade kako bi se pripremila za analizu od strane modela strojnog učenja. Prva faza obuhvaća normalizaciju, pri čemu se svaki piksel slike skalira tako da njegove vrijednosti budu unutar raspona od 0 do 1. Normalizacijom se uklanjaju razni šumovi i nepravilnosti, kao što su nejednake veličine, debljina linija, ili stil fonta, kako bi se fokus stavio na bitne značajke samih znakova. Slika se obično skalira na veličinu 20x20 piksela kako bi se smanjila kompleksnost podataka te postigla brža obrada i manja potrošnja memorije.

Nakon normalizacije, ključni dijelovi slike koji opisuju karakteristike pisanog znaka, odnosno značajke, koriste se kao ulaz za model strojnog učenja. Značajke sadrže informacije kao što su razni oblici, rubovi ili obrasci unutar slike, koji su važni za razlikovanje različitih znakova. Model, koji je prethodno treniran na velikom skupu pisanih uzoraka i koristi naučene obrasce iz tih značajki za analizu novih podataka. Na temelju toga model daje procjenu vjerojatnosti za svaku moguću klasu (npr. koji broj je prikazan). Na temelju tih vjerojatnosti model donosi zaključak o tome koji je znak najvjerojatnije prikazan, odnosno koja klasa daje najvišu vjerojatnost.

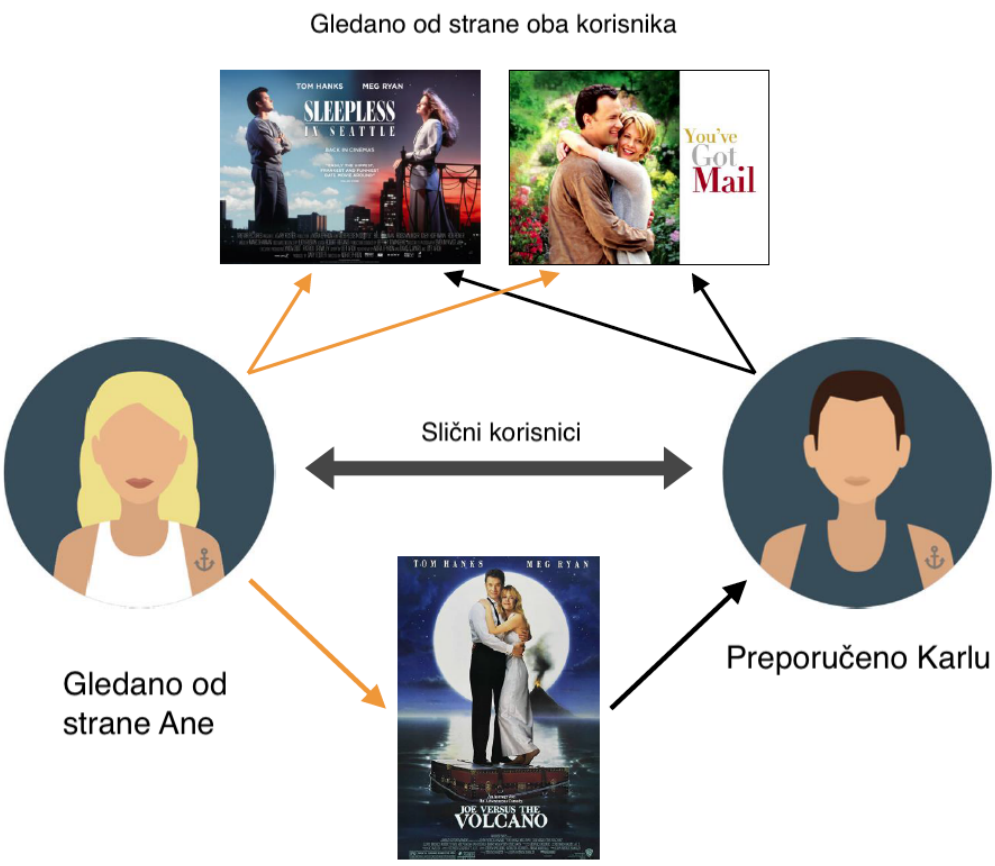


Slika 2: Proces klasificiranja, prema [5]

U financijskom svijetu, strojno učenje ima primjenu pri detekciji prevara. Možda čak najidealnije mjesto za treniranje modela jer se svakodnevno kreiraju milijuni transakcija, omogućujući modelu da uči na vrlo velikom skupu podataka. Model može vrlo dobro prepoznati sumnjive aktivnosti i neobične transakcije, čime se smanjuje rizik od prevara i poboljšava sigurnost korisnika.

Jedna od najvidljivijih primjena strojnog učenja su sustavi za preporuku, koji se koriste na platformama poput Netflixa, Amazona ili YouTubea.[6] Takvi sustavi su napravljeni sa sposobnošću analiziranja uzorka ponašanja nekog korisnika te s obzirom na povijest pretraga i preference personalizira se preporuka filmova ili serija. Na taj način, korisnici dobivaju sadržaj koji je najrelevantniji za njih. Pomoću takvog sustava korisnici dobivaju sadržaj koji je njima personaliziran, što za samu platformu znači zadržavanje korisnika i njihove pretplate. Na primjeru prikazanom na slici 3, vidljive su dvije osobe Ana i Karlo koji su korisnici neke platforme za filmove. Ana i Karlo su gledali dva ista filma: "Sleepless in Seattle" i "You've Got Mail". Na temelju te informacije, algoritam prepoznaje da su Ana i Karlo slični jer dijele slične interese. Ana je gledala film "Joe Versus the Volcano," ali Karlo još nije. Budući da su oboje identificirani kao slični, algoritam preporučuje taj film Karlu. Dakle, ako jedna osoba preferira određeni sadržaj, a druga osoba sa sličnim preferencama također voli takav sadržaj, sustav će biti u stanju preporučiti sadržaj osobi koji je druga osoba gledala s vjerojatnošću da će se taj sadržaj svidjeti toj osobi na temelju prethodnog iskustva.

Takav način pristupa se zove kolaborativno filtriranje gdje se ne zahtijeva detaljno poznavanje samog sadržaja, već se isključivo oslanja na podatke o ponašanju i njihovoj sličnosti. [7]



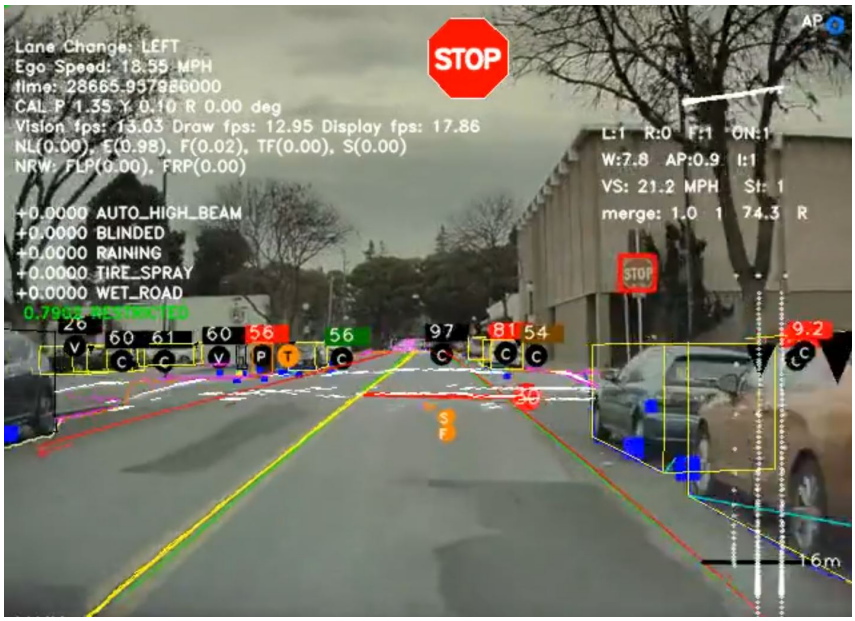
Slika 3: Primjer kolaborativnog filtriranja, prema [8]

Sljedeće područje primjene je računalni vid. Strojno učenje u ovoj primjeni kao zadatak mora moći interpretirati i razumijeti ono što "vidi". Najčešći primjer računalnog vida je u autonomnim vozilima gdje sustav pomoću raznih kamera, lidara, radara i senzora omogućuje vozilu da "vidi" i donosi odluke u stvarnom vremenu. Podaci koji se prikupljaju prolaze kroz velik broj procesa kao što je detekcije objekata, segmentacije scene i praćenja objekata, što na kraju vozilima omogućuje prepoznavanje drugih automobila, pješaka, prometnih znakova i prepreka na cesti.

Detekcija objekata je prvi korak u ovom procesu, a obavlja se uz pomoć modela treniranih sa dubokim konvolucijskim neuronskim mrežama (CNN). CNN modeli analiziraju slike koje kamere snimaju i identificiraju objekte poput automobila, pješaka, biciklista, prometnih znakova i semafora. Kako bi modeli bili uspješni, jako je bitno trenirati na velikim količinama podataka, gdje su slike ručno anotirane kako bi algoritam naučio prepoznavati različite kategorije objekata odnosno raditi generalizaciju.[9]

Nakon detekcije objekta, potrebno je određivanje pozicije objekta i da li postoji povezanost s okolinom. Taj proces se zove segmentacija scene jer se kroz sliku prolazi u segmentima. Slika 4 prikazuje kako autonomno vozilo identificira i kategorizira različite objekte na cesti, poput drugih vozila, bicikala i prometnih znakova, označavajući ih raznim oznakama i okvirima.

Nakon obrade vizualnih informacija, sustav donosi odluke poput planiranja putanje, kontrolu brzine i reakcije na promjene u prometu. Primjer prikazan na slici 4 je i prepoznavanje prometnih znakova, poput znaka "STOP". Ovaj znak signalizira vozilu da se mora zaustaviti, računalni vid omogućuje sustavu da prepozna znak, te da pravilno postupi prema znaku, u ovom slučaju da se automobil mora zaustaviti. Na sličan način radi se i sa semaforima, naime sustav kod Tesle uz pomoć kamera i GPS lokacija detektira gdje su mogući semafori i kamera detektira koja boja se nalazi na semaforu, prilikom svake detekcije vozilo uspori kako bi što točnije i preciznije prepoznalo svijetlo, te ukoliko se utvrdi da je svjetlo na semaforu zeleno tada vozilo usporeno nastavlja kroz raskrižje. Ukoliko prepozna žuto ili crveno svijetlo tada vozilo usporava do potpunog zastoja. Korisnik preko papučice za ubrzanje ima mogućnost nadglasiti naredbu.[9]



Slika 4: Tesla znak stop, preuzeto iz [10]

Ovi primjeri pokazuju kako strojno učenje ima različitu primjenu, od svakodnevnih aktivnosti poput preporuka sadržaja do primjena u industriji i medicini. S daljnjim razvojem tehnologije i dostupnošću sve većih količina podataka, primjena strojnog učenja bi mogla rasti.[11]

2.1.2. Učenja u strojnom učenju

Strojno učenje se sastoji od nekoliko glavnih vrsta, koja se razlikuju po načinu na koji model uči i vrsti podataka koji koristi.

Nadzirano učenje (engl. Supervised Learning): Nadzirano učenje je najraširenija vrsta strojnog učenja, gdje model uči iz označenih podataka. Svaki podatak se mapira na izlaznu vrijednost koja se naziva klasa. Na taj način se omogućuje modelu da prepozna obrasce na podacima i prepozna klasu kojoj pripada. Na primjer, u medicini, nadzirano učenje može se koristiti za klasifikaciju rendgen snimaka, kako bi se identificirale razne bolesti ili promjene. Model se trenira na skupu označenih slika, gdje svaka slika ima pridruženu dijagnozu odnosno klasu, a zatim se koriste naučeni obrasci odnosno značajke za identifikaciju bolesti na novim, neoznačenim slikama. [3]

Nenadzirano učenje (engl. Unsupervised Learning): Nenadzirano učenje koristi neoznačene podatke, što znači da model nema unaprijed definirane klase za primjere na kojima uči. Cilj ovakvog učenja je prepoznati skrivene značajke u podacima.[3] Na primjer, u analizi društvenih mreža, nenadzirano učenje može se koristiti za grupiranje korisnika na temelju njihovih zajedničkih aktivnosti, kao što je na instagramu interakcija s postovima, lajkovi ili komentari. Takvi podaci se koriste kako bi se identificirale grupe sa zajedničkim interesima unutar mreže. Model može automatski grupirati korisnike sličnih interesa ili ponašanja, poput onih koji često gledaju akcijske serije ili one koji preferiraju luksuzne marke. Ovo grupiranje može pomoći u ciljanju skupine za različite marketinške kampanje.

Poticano učenje (engl. Reinforcement Learning): Poticano učenje temelji se na sustavu nagrada i kazni, gdje model uči kroz interakciju s okolinom. Model donosi odluke koji utječe na okolinu, a povratne informacije o tim odlukama koriste se za optimizaciju ponašanja modela. Ovo učenje se koristi u igri, gdje algoritmi uče kako najbolje igrati neke igru. Na primjer, algoritmi podržanog učenja su korišteni za razvoj sustava kako bi svladali igre poput šaha.[12] Ova vrsta učenja se koristi i u autonomnim vozilima, gdje model uči kako se sigurno i učinkovito kretati kroz složena raskrižja.[3]

Različite vrste učenja omogućuju modelima da rješavaju širok spektar problema, prilagođavajući se specifičnim potrebama različitih primjena, od jednostavnih klasifikacija do složenih odluka u stvarnom vremenu.

2.2. Tehnike strojnog učenja

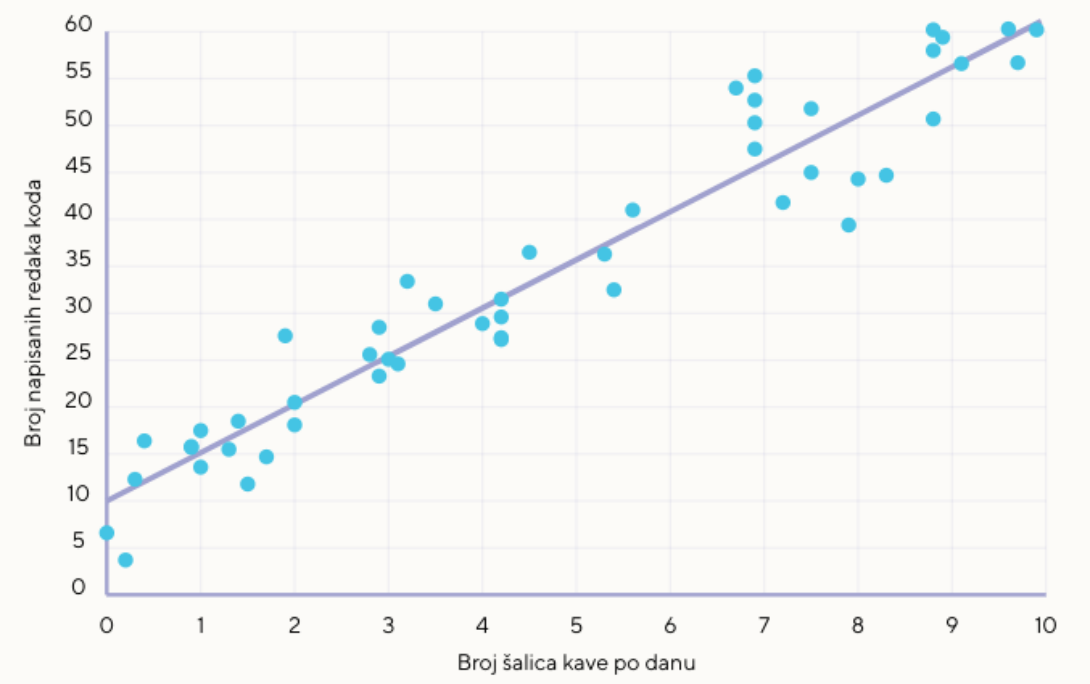
U ovom podpoglavlju će se obraditi različite tehnike strojnog učenja koje se koriste za treniranje modela kako bi mogli prepoznati obrasce, donositi odluke i predviđati ishodi na temelju podataka.

Različite metode se primjenjuju u strojnom učenju, od jednostavnih kao što su regresija i klasifikacija, do kompleksnijih kao što su neuronske mreže i duboko učenje. Svakoj tehnici su pridružene prednosti i koriste se prema vrsti podataka i problemu koji se treba riješiti. Ovdje će se opisati tehnike kao što su regresija, klasifikacija, stabla odlučivanja, neuronske mreže i generativni modeli.

2.2.1. Regresija

Regresija je metoda pomoću koje se određuje odnos između zavisne varijable (cilja) i jedne ili više nezavisnih varijabli (značajki) u strojnom učenju. [13] Regresija predviđa stvaran broj kao izlaznu varijablu. Na primjer, linearna regresija može se koristiti za predviđanje cijena nekretnina na temelju značajki kao što su lokacija, površina i starost zgrade. U ovoj metodi, linearna kombinacija značajki se koristi za procjenu vrijednosti cilja gdje svaka karakteristika ima težinsku vrijednost koja određuje njen utjecaj na krajnji rezultat.

Jedna od glavnih prednosti linearne regresije je njena jednostavnost. Koeficijenti koje model nauči mogu pružiti uvid u to kako svaka značajka utječe na izlaz. Na primjer odnos broja šalica kave koju popije neki zaposlenik i broj linija koda koji napiše u istom danu. Slika 5 prikazuje jednostavnu linearnu regresiju koja prikazuje odnos između broja šalica kave koje zaposlenik dnevno popije i broja linija koda koje napiše u istom danu. Svaka točka na grafu predstavlja jednog zaposlenika, a linija koja prolazi kroz točke prikazuje predviđeni ishod. Iz grafa je vidljivo da svaka dodatna šalica kave poveća broj linija koda za 5. Odsječak na liniji govori da i zaposlenici koji ne piju kavu ipak napišu oko 10 redaka koda dnevno. [14]



Slika 5: Linearna regresija na primjeru odnosa između broja šalica kave i broja linija koda, preuzeto iz [14]

Nedostatak linearne regresije je što model ne može dobro uhvatiti složene, nelinearne odnose u podacima, te je jako osjetljiva na iznimke u podacima. [14]

Unatoč tim nedostacima regresija je zbog svoje jednostavnosti i brzine treniranja, jedan od najčešće korištenih modela u strojnom učenju. Neki od najčešćih primjera primjene metoda regresije je kod predviđanja cijena nekretnina gdje je cilj predvidjeti kontinuiranu vrijednost na temelju ulaznih podataka.

2.2.2. Klasifikacija

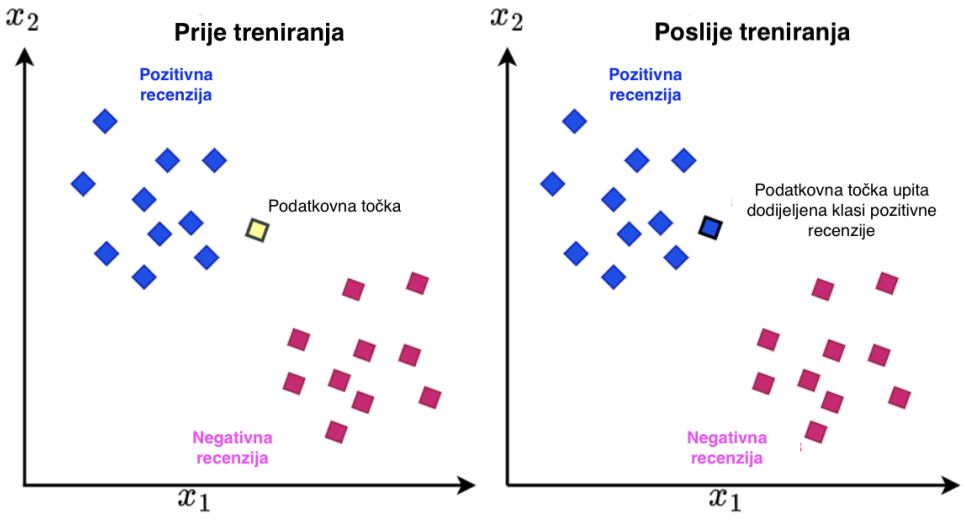
Klasifikacija je tehnika strojnog učenja koja se koristi za svrstavanje podataka u različite kategorije ili klase.[15] U klasifikaciji, model uči iz skupa označenih podataka, gdje su podaci unaprijed kategorizirani, te na temelju tih primjera model uči prepoznati obrasce koji omogućuju ispravnu kategorizaciju novih, neoznačenih podataka. Najčešći primjer klasifikacije je spam e-pošte. [15]

Jedna od jednostavnijih metoda za klasifikaciju je klasifikator najbližih susjeda (KNN). KNN spada u algoritme "lijenog učenja", gdje se podaci iz treninga samo pohranjuju i čekaju dolazak novih podataka za testiranje. Ova metoda klasificira nove podatke na temelju njihovih najbližih susjeda iz skupa za treniranje. KNN je potrebno manje vremena za treniranje, ali više vremena za predviđanje. Kad god se pojavi novi podatak, KNN algoritam pronalazi "k" najbližih susjeda iz postojećeg skupa podataka i klasificira novi podatak prema klasi kojoj pripada većina tih susjeda. Na primjer, ako postoji skup podataka prikazanih kao točke na grafu, gdje su različite boje točaka povezane s različitim klasama (npr. zelena i plava), KNN algoritam izračunava udaljenost između novog podatka i postojećih podataka u skupu za treniranje. Klasifikacija se

izvodi prema klasi kojoj pripada većina najbližih susjeda (npr. ako su najbliži susjedi većinom zeleni, novi podatak će biti klasificiran kao zelena klasa). [16]

Udaljenost između podataka može se mjeriti na različite načine, a najčešća metoda je euklidska udaljenost, koja predstavlja standardnu geometrijsku udaljenost između dvije točke u prostoru.[17] KNN se najčešće primjenjuje u sustavima za preporuke, poput onih koji predviđaju korisničke preferencije na temelju sličnih korisnika. Na primjer, ako korisnik voli disco glazbu iz 80-ih, a drugi slični korisnici uživaju u nekoj novoj pjesmi, KNN algoritam može preporučiti tu pjesmu jer predviđa da će se i tom korisniku svidjeti. [17]

Na slici 6 je prikazan primjer primjene KNN algoritma za klasifikaciju recenzija. Prije treninga (lijevi graf), novi podatak, označen kao "podatkovna točka", nije klasificiran, no nakon treninga (desni graf), algoritam je taj podatak klasificirao kao pozitivnu recenziju.



Slika 6: klasifikator najbližih susjeda primjer, preuzeta od [17]

To je postignuto tako što je KNN algoritam identificirao "k" najbližih susjeda u skupu za treniranje, koristeći euklidsku udaljenost. Euklidska udaljenost između dvije točke p(x1, x2, ..., xn) i q(y1, y2, ..., yn) računa se pomoću formule

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

Nakon izračuna udaljenosti između novog podatka i svih postojećih podataka u skupu za treniranje, KNN algoritam identificira "k" najbližih susjeda. Klasifikacija se provodi na temelju klase kojoj pripada većina tih susjeda. U primjeru recenzija najbliži susjedi nove točke uglavnom pripadaju klasi pozitivnih recenzija, te je zbog toga novi podatak klasificiran kao pozitivna recenzija. Prednost KNN-a je u njegovoj jednostavnosti gdje su podaci jasno grupirani u različite kategorije. No, nedostatak ove metode je što može postati spor i neučinkovit kada se koristi na vrlo velikim skupovima podataka, jer za svaki novi podatak koji treba klasificirati, potrebno je izračunati udaljenost do svih podataka u skupu za treniranje, neovisno o tome jesu

li oni već klasificirani. Ovi podaci u skupu za treniranje imaju poznate oznake (klase), a novi podatak se uspoređuje s njima kako bi se pronašli najbliži susjedi, na temelju kojih se donosi odluka o njegovoj klasifikaciji.

Naivni Bayesov klasifikator je algoritam za klasifikaciju, koji pretpostavlja da jedan uzrok (npr. je li recenzija pozitivna ili negativna) izravno utječe na niz učinaka (npr. prisutnost određenih riječi u recenziji). Ključna pretpostavka je da su svi učinci međusobno neovisni kada je poznat uzrok.[18] Matematički zapis pretpostavke:

$$P(\text{Uzrok}, \text{Riječ}_1, \ldots, \text{Riječ}_n) = P(\text{Uzrok}) \prod_i P(\text{Riječ}_i \mid \text{Uzrok}).$$

Ova pretpostavka o neovisnosti naziva se 'naivnom' jer zanemaruje stvarne veze između učinaka. Naivni Bayesov model pretpostavlja da je svaki učinak (kao što je prisutnost određene riječi u recenziji) neovisan o ostalim učincima, što u ponekad ne mora biti slučaj.

Na primjer, ako recenzija sadrži riječi "odličan", "preporučujem", i "najbolje", naivni Bayes bi izračunao vjerojatnost da je recenzija pozitivna, uzimajući u obzir svaku od tih riječi zasebno, bez obzira na druge riječi u tekstu.

Na kraju, koristeći Bayesov teorem, moguće je procijeniti vjerojatnost da je recenzija pozitivna s obzirom na prisutnost tih riječi:

$$P(\text{Pozitivna recenzija} \mid \text{riječi}) = \alpha P(\text{Pozitivna recenzija}) \prod_j P(\text{riječ}_j \mid \text{Pozitivna recenzija}).$$

Razlika između klasifikacije i regresije je u izlaznoj varijabli gdje regresija predviđa kontinuiranu vrijednost (kao što je cijena ili temperatura), dok klasifikacija dodjeljuje podatke jednoj od više mogućih kategorija (klasa). Na primjer, klasifikacija može odrediti je li određena poruka e-pošte spam ili ne-spam, dok bi regresija mogla predvidjeti koliko je vjerojatno da će korisnik kliknuti na oglas na temelju povijesnih podataka.

2.2.3. Stablo odlučivanja

Stablo odlučivanja je [18] prikaz funkcije koja mapira vektor vrijednosti atributa na jednu izlaznu vrijednost odnosno odluku. Stablo odlučivanja dolazi do svoje odluke izvođenjem niza testova, počevši od korijena i prateći odgovarajuću granu sve dok se ne dosegne list. Svaki unutarnji čvor u stablu odgovara testu vrijednosti jednog od ulaznih atributa. Grane iz čvora označene su s vrijednostima atributa, a listovi određuju koja vrijednost će biti vraćena funkcijom.

Proces izgradnje stabla započinje testiranjem na atributima u korijenskom čvoru, a zatim se podaci granaju u unutarnje čvorove na temelju rezultata testa, sve dok se ne dođe do završnih čvorova ili listova. Ovi listovi mogu predviđati kategorijsku varijablu (klasifikacijsko stablo) ili kontinuiranu varijablu (regresijsko stablo). Stabla odlučivanja mogu biti vrlo jednostavno za rad s različitim vrstama podataka, no mogu sadržavati problem pretreniranosti, gdje model

postaje prekompliciran i prilagođava se previše na jedan skup podataka za treniranje, što smanjuje njegovu sposobnost generalizacije. Na slici 7 nalazi se stablo odlučivanja koje prikazuje proces donošenja odluke o tome je li osoba fit ili ne. Odluke ovise o atributima poput dobi osobe, učestalosti konzumacije pizze i jutarnje vježbe. Stablo započinje pitanjem "Je li osoba mlađa od 30 godina?" i na temelju odgovora grana se na daljnje atribute, dok se ne dođe do konačnog ishoda "fit" ili "nije fit".

Slika 7: Stablo odlučivanja, preuzeto sa [19]

2.2.4. Neuronske mreže i duboko učenje

Cilj neuronskih mreža je oponašati ljudski način učenja i donošenja odluka, koristeći ulazne podatke za generiranje odgovarajućih izlaza. Ove mreže se koriste za velik broj zadataka kao što je prepoznavanje uzoraka, klasifikacija i regresija, te su dobre u područjima poput prepoznavanja slika, govora i analize teksta. Neuronska mreža se sastoji od više međusobno povezanih neurona, organiziranih u slojeve: ulazni sloj, jedan ili više skrivenih slojeva, te izlazni sloj.[20]

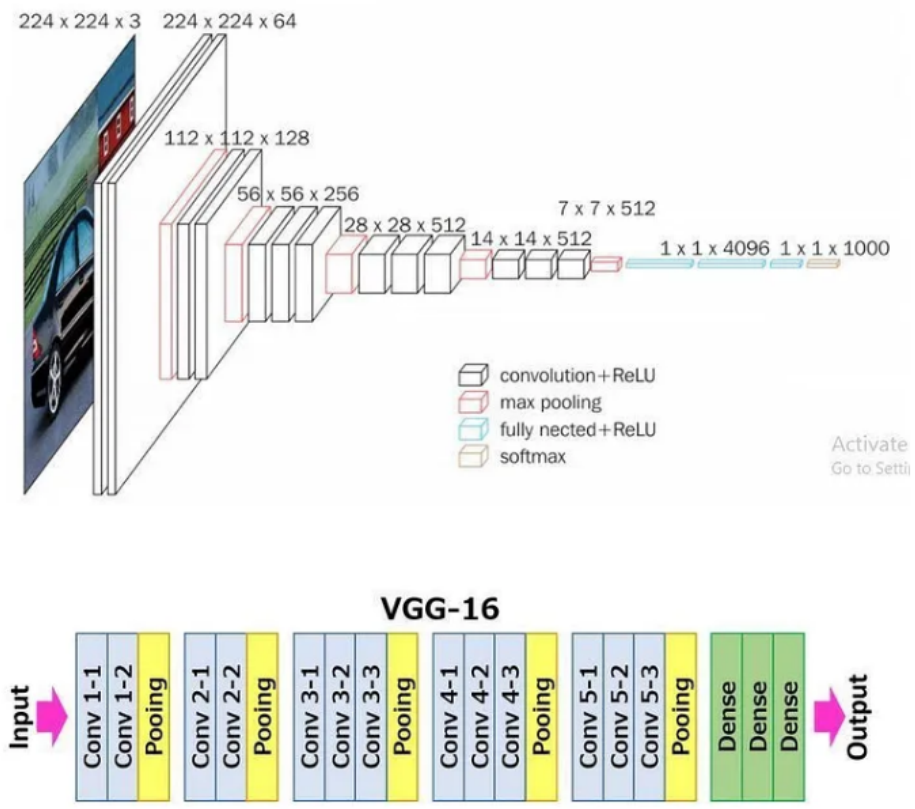
Arhitektura neuronske mreže definira se brojem slojeva i brojem neurona u svakom sloju. Ulazni sloj prima sirove podatke iz vanjskog okruženja, dok skriveni slojevi obrađuju te podatke, omogućujući mreži da uči složenije i apstraktnije obrasce. Izlazni sloj generira predikciju ili klasifikaciju na temelju obrađenih informacija.

Duboko učenje (Deep Learning) je naprednija forma neuronskih mreža koja se oslanja na korištenje velikog broja skrivenih slojeva za obradu složenih podataka. Duboke mreže koriste velik broj slojeva, omogućujući mreži da uči apstraktne i složene podatke. Glavna prednost dubokih mreža je njihova sposobnost automatskog učenja prikaza iz sirovih podataka, bez potrebe za ručnim generiranjem značajki. No nedostatci su što im je potrebna velika količina podataka i računalnih resursa, te uz kriva podešavanja nastaju problemi poput pretreniranosti. [21]

Proces učenja u neuronskoj mreži kreće se kroz metodu unazad, gdje mreža uči prilagođavati težine na temelju grešaka između predviđenih i stvarnih vrijednosti. Arhitektura neuronske mreže može se prilagoditi tijekom procesa učenja kako bi se postigli bolji rezultati. Glavna mana neuronskih mreža je što često djeluju kao “crne kutije", odnosno teško je razumjeti zašto je mreža donijela određenu odluku. [21]

Napredne tehnike poput rekurentnih neuronskih mreža (RNNs) i konvolucijskih neuronskih mreža (CNNs) omogućuju obradu složenih podataka i slika. CNN-ovi, na primjer, koriste se za prepoznavanje objekata na slikama, pri čemu slojevi konvolucije omogućuju mreži da prepoznaje značajke kao što su rubovi, teksture i oblici.

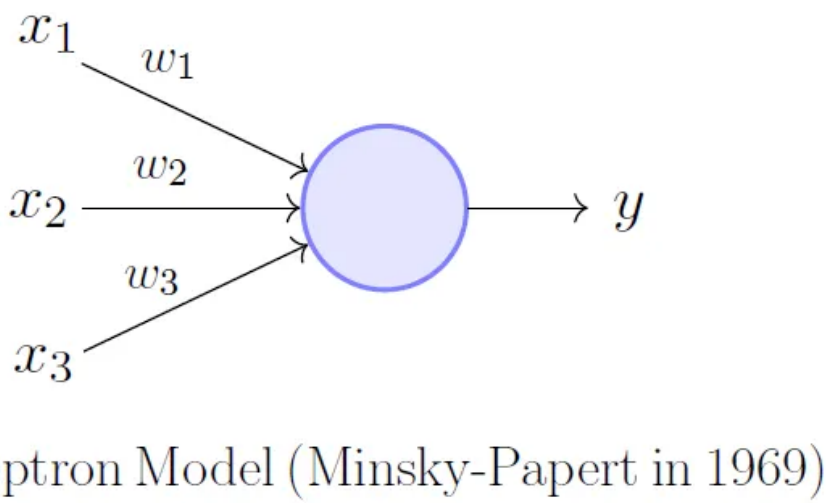
Jedan od najpoznatijih CNN modela je VGG16.[22]. Naime, VGG16 je duboka konvolucijska mreža sa 16 slojeva, koja koristi male 3x3 konvolucijske filtere za postizanje visoke točnosti u zadacima prepoznavanja slika. VGG16 je najčešće korišteni model u sustavima računalnog vida, zbog svoje jednostavne, ali učinkovite arhitekture koja omogućuje preciznu obradu slika s relativno malom potrebom za teškim resursima. [23]



Slika 8: VGG16 Arhitektura, preuzeto iz [23]

Izrada neuronskih mreža izvodi se kroz nekoliko koraka. Definiranje arhitekture mreže (broj slojeva i neurona u svakom sloju), odabir aktivacijskih funkcija, te postavljanje inicijalnih težinskih vrijednosti. Treniranje mreže podrazumijeva prilagođavanje težinskih vrijednosti na temelju greške između stvarnih i predviđenih izlaza, dok se validacija koristi za optimizaciju hiperparametara i osiguranje da mreža generalizira dobro na nove podatke. [21]

Jedan od temeljnih modela neuronskih mreža je perceptron, najjednostavniji oblik mreže, koji se sastoji od jednog ulaznog sloja i izlaznog čvora.[24] Perceptron je temelj svake složenije neuronske mreže i koristi se za binarnu klasifikaciju podataka.[21] Novi modeli danas koriste složenije arhitekture koje uključuju puno više slojeva i tehnika za treniranje kako bi imale što bolji rezultati.

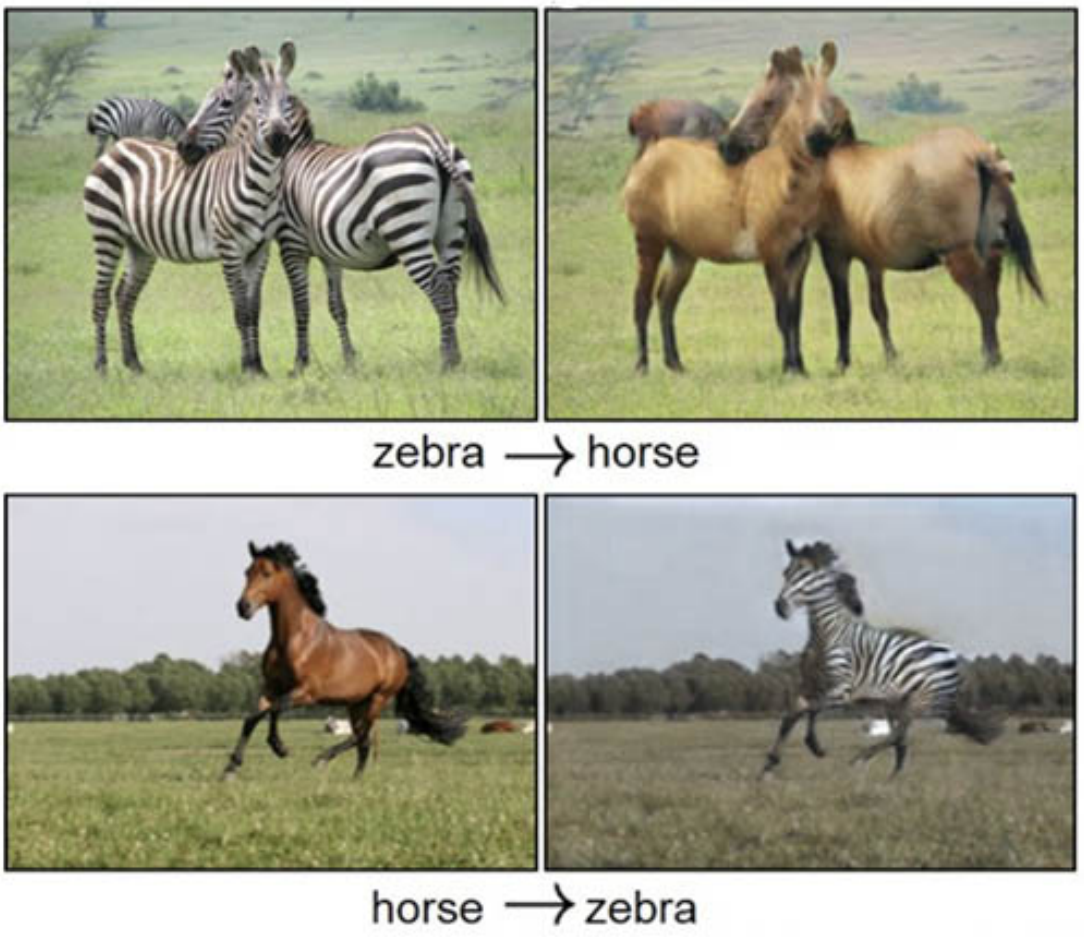


Slika 9: Perceptron model, preuzeto iz [25]

2.2.5. Generativni modeli

Generativni modeli su tehnika strojnog učenja koja se koristi za modeliranje distribucije podataka kako bi se generirali novi podaci koji su slični onima iz skupa za treniranje. Ovi modeli uče obrasce i strukture u podacima, omogućujući im stvaranje novih uzoraka.[24] Generativna suparnička mreža (engl. Generative adversarial networks (GAN))[26] se sastoje od dvije mreže - generatora i diskriminatora koje se natječu jedna protiv druge. Generator pokušava stvoriti lažne podatke koji su dovoljno uvjerljivi da prevari diskriminatora, dok diskriminator pokušava razlikovati stvarne od lažnih podataka.

Generativni modeli se primjenjuju u računalnom vidu, medicini i mnogim različitim područjima. Na primjer, GAN-ovi se koriste za generiranje realističnih slika, kao što su slika lica ili pejzaži, koji ne postoje u stvarnosti, ali izgledaju uvjerljivo. Slika 10 prikazuje primjer rezultata nakon korištenja GAN-a.



Slika 10: Primjer rezultata generativne suparničke mreže, preuzeto iz [26]

2.3. Metrike strojnog učenja

U kontekstu strojnog učenja, metrike su jedan od glavnih kriterija za odabir algoritma strojnog učenja. Pomoću metrika moguće je kreirati evaluaciju performansi modela i procjenu njihove točnosti i učinkovitosti. Jedan od osnovnih kriterija za odabir algoritama je točnost algoritma na neviđene podatke odnosno pogreška generalizacije. [27] Uz osnovne metrike koristi se niz različitih metrika koji se koriste ovisno o vrsti zadatka koji model treba obaviti, poput klasifikacije, regresije ili klasterizacije.

Za klasifikacijske zadatke, najčešće korištene metrike su točnost (engl. accuracy), preciznost (engl. precision), odziv (engl. recall), F1 rezultat i matrica konfuzije.[27] Točnost mjeri postotak ispravno klasificiranih instanci u odnosu na ukupan broj instanci. Preciznost govori koliko od predviđenih pozitivnih slučajeva stvarno pripada pozitivnoj klasi, dok odziv mjeri koliko od stvarnih pozitivnih slučajeva model ispravno identificira. F1 rezultat je harmonijska sredina između preciznosti i odziva, a korisna je kada postoji neuravnoteženost u klasama. Matrica konfuzije prikazuje mjeru kako su stvarne klase podijeljene među predviđenim klasama. [28]

Za regresijske zadatke, metrike koje se koriste su srednja kvadratna pogreška (engl. Mean Squared Error - MSE), srednja apsolutna pogreška (engl. Mean Absolute Error - MAE) i koeficijent determinacije (R^2 rezultat). MSE mjeri koliko su u prosjeku predviđene vrijednosti udaljene od stvarnih vrijednosti, s tim da više "kažnjava" velike pogreške. Drugim riječima, ako je razlika između predviđene i stvarne vrijednosti velika, MSE će je značajno uvećati jer kvadrira pogreške, što znači da velike pogreške imaju puno veći utjecaj na rezultat nego male. MAE mjeri prosječnu apsolutnu pogrešku, što je korisno za razumijevanje prosječne veličine pogrešaka bez obzira na njihov smjer. R^2 rezultat pokazuje koliko dobro predviđene vrijednosti objašnjavaju varijaciju stvarnih vrijednosti.[29]

Za modele strojnog učenja koji uključuju klasifikaciju, regresiju, generativne modele ili duboke neuronske mreže, metrika gubitka (loss) je također važna metrika. Gubitak predstavlja mjeru koliko su predviđanja modela daleko od stvarnih vrijednosti, te se ova metrika pokuša minimalizirati te dostići optimalan broj nula.[29]

2.3.1. Točnost (engl. Accuracy)

Točnost (Accuracy) je jedna od najjednostavnija i najčešće korištenih metrika u klasifikacijskim zadacima. Ona mjeri postotak ispravno klasificiranih instanci u odnosu na ukupan broj instanci. [28]

Formula:

$$\text{Točnost} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.1}$$

Gdje su:

- **Pravi pozitivni slučajevi** (engl. *True Positives (TP)*) – Model ispravno klasificira pozitivne primjere. Na primjer, model ispravno prepoznaje poruku koja je spam kao "spam".

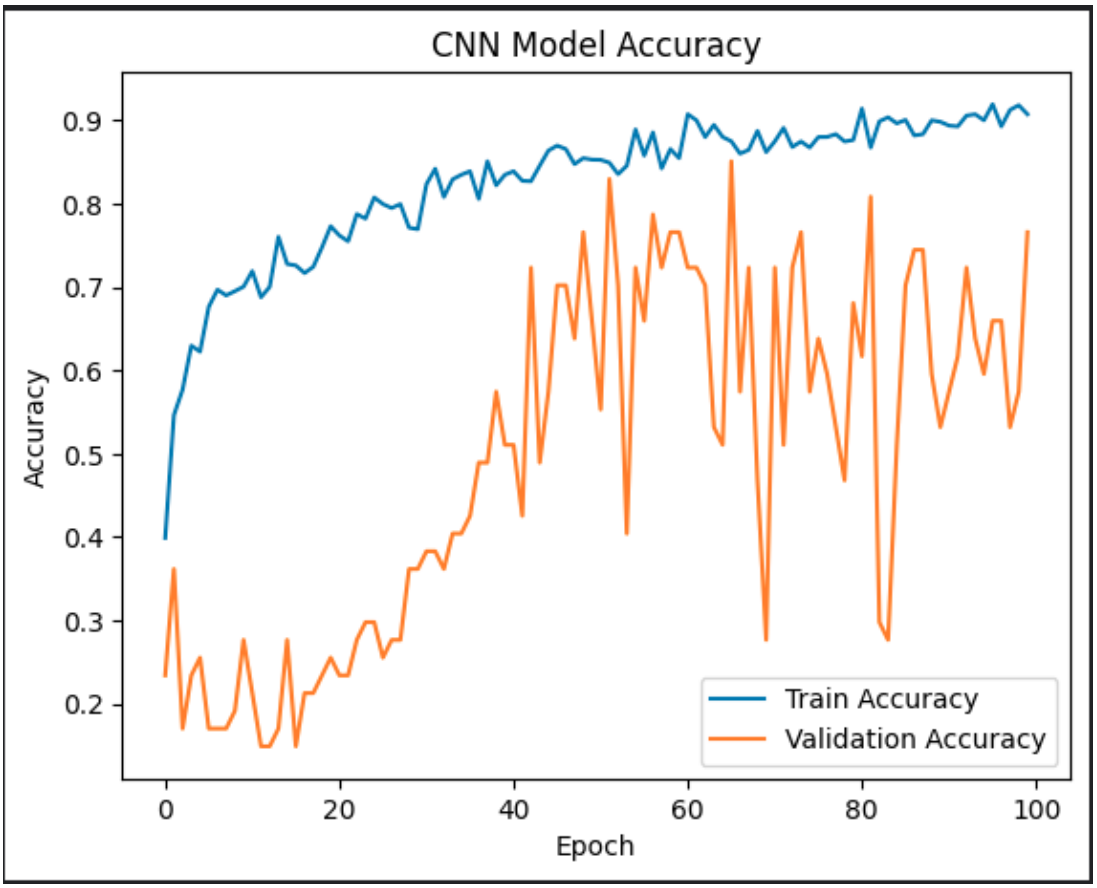
- **Pravi negativni slučajevi** (engl. *True Negatives (TN)*) – Model ispravno klasificira negativne primjere. Na primjer, model ispravno prepoznaje poruku koja nije spam kao "nije spam".
- **Lažno pozitivni slučajevi** (engl. *False Positives (FP)*) – Model pogrešno klasificira negativne primjere kao pozitivne. Na primjer, model pogrešno prepoznaje poruku koja nije spam kao "spam". Ovo se često naziva "false alarm" jer signalizira problem koji ne postoji.
- **Lažno negativni slučajevi** (engl. *False Negatives (FN)*) – Model pogrešno klasificira pozitivne primjere kao negativne. Na primjer, model pogrešno prepoznaje poruku koja je spam kao "nije spam". Ovo može biti opasno jer stvarni problem (spam) nije otkriven.

Na primjer, ako model klasificira 100 e-poruka, od kojih je 90 ispravno klasificirano (80 ispravnih kao "nije spam" i 10 ispravnih kao "spam"), točnost modela bi bila 0.9.

Točnost računa se kao omjer ispravno klasificiranih primjera (TP + TN) u odnosu na ukupan broj primjera. Dakle, za ovaj primjer:

$$\text{Točnost} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{80 + 10}{100} = 0.9 \tag{2.2}$$

Nedostatak ove metrike je što njena vrijednost može varirati u odnosu na zastupljenost neke klase, te za zastupljenije podatke neke klase na kojoj je više trenirano može imati veću točnost u odnosu na manje zastupljene podatke za drugu klasu. Na primjer, ako je klasa "nije spam" dominantna, model može imati visoku točnost čak i ako ne prepoznaje spam poruke pravilno, jer dominira klasa "nije spam".



Slika 11: Primjer grafa točnosti

2.3.2. Preciznost (engl. Precision)

Preciznost (Precision) govori koliko od predviđenih pozitivnih slučajeva zaista pripada pozitivnoj klasi[28]. Računa se kao:

$$\text{Preciznost} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.3}$$

Na primjer, ako model za detekciju bolesti ispravno prepozna 80 bolesnih pacijenata (TP), ali isto tako označi 20 zdravih pacijenata kao bolesne (FP), preciznost modela bi bila 80%. Preciznost je korisna kada se želi minimizirati lažno predstavljanje.Prednost preciznosti je da pruži informacije o pouzdanosti pozitivnih predikcija. Nedostatak je što ne uzima u obzir lažno negativne slučajeve, one slučajeve koji su trebali biti označeno kao pozitivni ali nisu.

2.3.3. Odziv (engl. Recall)

Odziv (Recall), mjeri sposobnost modela da ispravno identificira sve pozitivne slučajeve u skupu podataka. [28] Izračunava se kao:

$$\text{Odziv} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.4}$$

Na primjer, ako model ispravno identificira 70 od 100 stvarno pozitivnih slučajeva, što znači da je propustio 30 (lažno negativni), odziv modela bi bio 70%. Prednost odziva je što nam govori koliko je model uspješan u hvatanju svih stvarno pozitivnih slučajeva, dok nedostatak je što može doći do većeg broja lažno pozitivnih slučajeva (FP).

2.3.4. F1 rezultat

F1 rezultat je harmonijska sredina preciznosti i odziva, koja uzima u obzir i lažno pozitivne i lažno negativne slučajeve. Računa se pomoću formule:

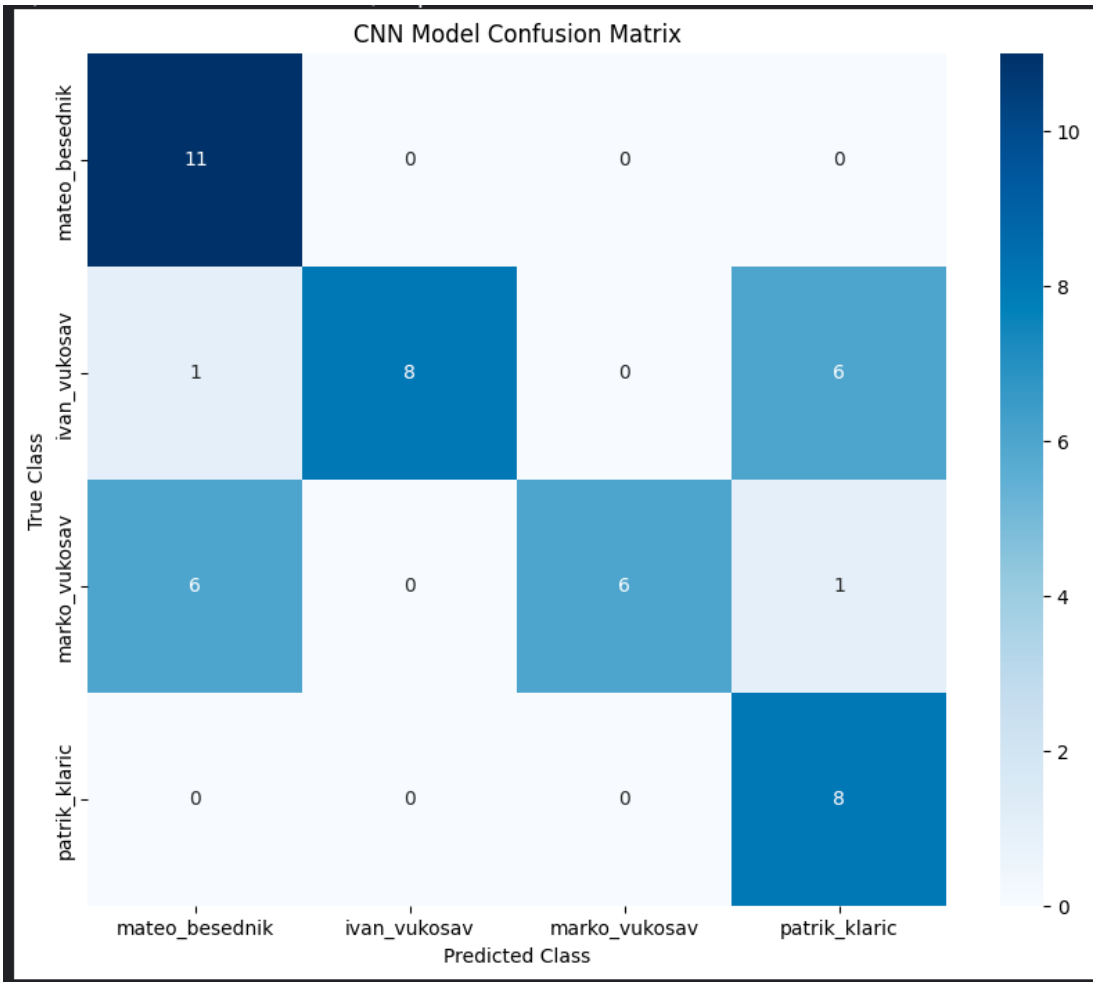
$$\text{F1 rezultat} = 2 \times \frac{\text{Preciznost} \times \text{Odziv}}{\text{Preciznost} + \text{Odziv}} \tag{2.5}$$

Na primjer, ako model ima preciznost od 80% i odziv od 70%, F1 rezultat bi bio 74.7%. F1 rezultat je koristan kada postoji neuravnoteženost između klasa kako bi se održala ravnoteža između preciznosti i odziva. Glavna prednost F1 rezultata je to što uklanja nedostatke preciznost i odziva na način što se može otkriti koliko broj lažno negativnih i lažno pozitivnih uzoraka ima ulogu. Nedostatak je što se ova mjera teže interpretira u usporedbi s mjerama poput preciznosti ili odziva. [30]

2.3.5. Matrica konfuzije (engl. Confusion matrix)

Matrica konfuzije je matrica koja prikazuje performanse modela strojnog učenja na testnom setu podataka. Naime, prikazuje broj ispravnih i pogrešnih predikcija modela, razvrstanih po stvarnim i predviđenim klasama. Svaka instanca u skupu podataka može biti svrstana u jednu od četiri kategorije: pravi pozitivni (TP), pravi negativni (TN), lažno pozitivni (FP) i lažno negativni (FN). Na primjer, ako model za detekciju bolesti ima 50 TP, 10 FP, 30 FN i 110 TN, matrica konfuzije prikazuje matricu performansi modela gdje su prikazani uspjesi i pogreške modela. [31]

Prednost matrice konfuzije što je moguće vidjeti koja klasa pruža najbolje rezultate, a s kojom ima poteškoća, te isto tako koje klase se kose sa drugim klasama i pružaju lažno pozitivne ili lažno negativne slučajeve. Nedostatak je što može postati teška za interpretaciju kada postoji veliki broj klasa.



Slika 12: Primjer matrice konfuzije

2.3.6. Metrika gubitka (engl. Loss)

Metrika gubitka, poznata i kao funkcija gubitka, je metrika koja pruža uvid o tome kako model vrši predikciju na pravim podacima. Manja vrijednost obično znači da je model bolji. Ona kvantificira razliku između predikcija modela i stvarnih vrijednosti (ciljeva). Cilj modela je minimizirati vrijednost funkcije gubitka tijekom treniranja, čime model postaje precizniji u svojim predikcijama. [32]

Postoji nekoliko vrsta funkcija gubitka koje se koriste u ovisnosti o zadatku. Jedna od korištenijih je srednja kvadratna greška (engl. Mean Squared Error (MSE)). MSE izračunava prosjek kvadrata razlika između predikcija modela i stvarnih vrijednosti. [33]

Formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{2.6}$$

- n: Ukupan broj podataka.
- Y_i: Stvarna vrijednost za i-ti podatak.
- $\hat{Y}_i$: Predviđena vrijednost za i-ti podatak.

- $\left|Y_i - \hat{Y}_i\right|$: Apsolutna pogreška, tj. razlika između stvarne i predviđene vrijednosti.

MSE se najčešće koristi kod modela koji kao zadatak imaju predvidjeti kontinuirane vrijednosti, kao što su cijene nekretnina, temperature, ili slične regresijske problemi.[33] Na primjer u modelu za predviđanje cijena kuća, MSE bi se koristio za mjerenje koliko se modelove predikcije cijena razlikuju od stvarnih cijena.

Sljedeća funkcija gubitka je srednja apsolutna greška (engl. Mean Absolute Error (MAE)). MAE mjeri prosječnu apsolutnu razliku između predikcija modela i stvarnih vrijednosti. Za razliku od MSE-a, MAE ne kvadrira greške, što ga čini manje osjetljivim na ekstremne vrijednosti. [33]

Formula:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{2.7}$$

- n: Ukupan broj podataka ili primjera.
- y_i: Stvarna vrijednost za i-ti podatak.
- x_i: Predviđena vrijednost za i-ti podatak.
- $|y_i - x_i|$: Apsolutna pogreška, tj. razlika između stvarne i predviđene vrijednosti.

MAE se najčešće koristi za regresijske zadatke kada su ekstremne vrijednosti manje važne ili kada je cilj smanjiti prosječnu apsolutnu grešku. Na primjer u nekom modelu koji predviđa vrijeme putovanja, MAE bi se koristio za mjerenje prosječnog odstupanja od stvarnog vremena putovanja.

Sljedeća funkcija gubitka je kategorijska unakrsna entropija (engl. Categorical Cross-Entropy). To je funkcija gubitka koja se koristi za klasifikaciju kada postoji više klasa, gdje model predviđa vjerojatnost da podatak pripada jednoj od tih klasa. [34]

Formula:

$$J(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{2.8}$$

- N - ukupan broj klasa,
- y_i - stvarna oznaka za i-tu klasu (one-hot kodiranje, obično 0 ili 1),
- $\hat{y}_i$ - predviđena vjerojatnost za i-tu klasu,
- $\mathbf{w}$ - težine modela koje se optimiziraju.

One-hot kodiranje (engl. One-Hot Encoding) je tehnika koja pretvara kategorijske podatke, poput imena klasa, u binarne vektore. U tim vektorima samo jedan element ima vrijednost 1, dok su svi ostali elementi 0. Ova tehnika se često koristi u funkciji gubitka kategorijska unakrsna entropija (engl. Categorical Cross-Entropy). [35]

Ova funkcija gubitka koristi se u zadacima klasifikacije s više klasa, kao što su prepoznavanje objekata na slikama ili klasifikacija lica.

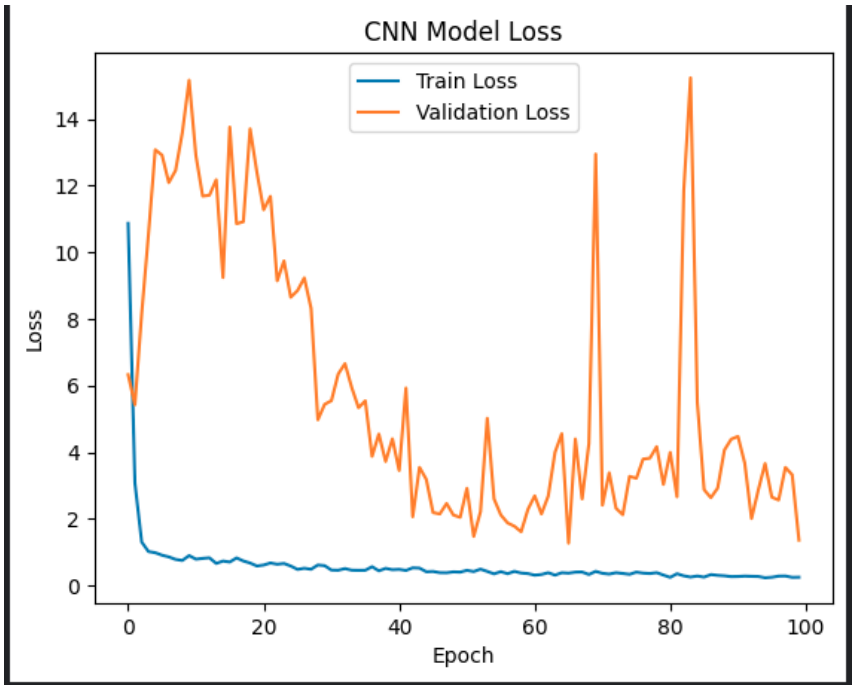
Sljedeća funkcija gubitka je rijetka kategorijska unakrsna entropija (engl. Sparse Categorical Cross-Entropy), koja se razlikuje od kategorijske unakrsne entropije po tome što koristi oznake cijelih brojeva umjesto one-hot kodiranih vektora. Te oznake izravno predstavljaju vrijednosti klasa. [34]

Formula:

$$J(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{2.9}$$

- N - ukupan broj klasa,
- y_i - stvarna oznaka za i-tu klasu (one-hot kodiranje, obično 0 ili 1),
- $\hat{y}_i$ - predviđena vjerojatnost za i-tu klasu,
- $\mathbf{w}$ - težine modela koje se optimiziraju.

Ova funkcija gubitka koristi se u zadacima klasifikacije s više klasa kada su oznake kao cijeli brojevi, što manje koristi memoriju i ima brži rad s velikim brojem klasa. Na primjer, u modelu za prepoznavanje lica, oznake su cijeli brojevi koji predstavljaju različite osobe u bazi podataka. Svaka osoba ima svoj jedinstven broj ili se prilikom učitavanja seta podataka oni pretvore u niz cijelih brojeva.



Slika 13: Primjer gubitka korištenjem rijetke kategorijske unakrsne entropije

Funkcija gubitka koristi se ovisno o vrsti zadatka koji model rješava. Za regresijske zadatke, gdje je cilj predviđati kontinuirane vrijednosti, najčešće se koriste srednja kvadratna

greška (MSE) i srednja apsolutna greška (MAE). Za klasifikacijske zadatke s više klasa koriste se rijetka kategorijska unakrsna entropije ili kategorijska unakrsna entropija, ovisno o formatu stvarnih oznaka. Ako su oznake one-hot kodirani vektori, koristi se kategorijska unakrsna entropija, dok se za cijele brojeve koristi rijetka kategorijska unakrsna entropija.

3. Strojno učenje u prepoznavanju lica

Prepoznavanje lica smatra se kao jedna od najkompleksnijih primjena strojnog učenja. Prepoznavanje lica omogućuje automatsko identificiranje ili verifikaciju osobe analizom karakterističnih crta lica na slikama ili u video. S razvojem strojnog učenja, posebno dubokog učenja i neuronskih mreža, prepoznavanje lica postalo je jako precizno pri velikom broju podataka i dobrim računalnim resursima.

Proces prepoznavanja lica sastoji se od nekoliko ključnih koraka: detekcija lica, ekstrakcija značajki, i klasifikacija ili verifikacija [36]. Svaki od ovih koraka može se implementirati korištenjem različitih algoritama i modela. U nastavku ovog poglavlja detaljno će se istražiti kako strojno učenje omogućuje prepoznavanje lica, koje tehnologije i modeli se koriste, te koji su izazovi i etička pitanja.

3.1. Uvod u prepoznavanje lica

Prepoznavanje lica omogućuje identifikaciju ili verifikaciju pojedinca analizom različitih fizičkih karakteristika, s naglaskom na specifične crte lica. Lice čovjeka je jedinstveno i složeno je od različitih elemenata, kao što su oči, nos, usta, i konture glave. Zbog te jedinstvenosti i raznih promjena u izrazu lica, promjena u osvjetljenju, prisutnosti sjena i kuta pod kojim je lice snimljeno, se smatra kao najkompleksnija primjena strojnog učenja.[37]

Slika lica sadrži puno informacija, od boje i teksture kože, do raspodjele svjetla i sjena, do raznih detalja koji oblikuju konture lica i slično. Svaka mala promjena ovih značajki može utjecati na preciznost prepoznavanja. Na primjer, boja kože može varirati ovisno o osvjetljenju, dok sjene i refleksije mogu stvoriti dodatne probleme u detekciji i analizi lica. Slike lica snimljene iz različitih kutova mogu prikazati lice iz perspektiva koje se međusobno značajno razlikuju, što može otežati identifikaciju istog lica u različitim situacijama.

Zbog takvih promjena i poteškoća u prepoznavanju lica potrebni su napredniji algoritmi sa sposobnošću izdvajanja relevantnih značajki koje će se pouzdano koristiti za prepoznavanje lica. Ovi algoritmi moraju biti dovoljno robusni da se mogu prilagoditi raznim promjenama ali opet i dovoljno precizni da razlikuju različita ali i slična lica. Jedni od takvih algoritama su konvolucijske neuronske mreže (CNN), koje imaju sposobnost izdvajanja relevantnih značajki i učiti prepoznati te značajke lica iz velikih skupova podataka te omogućuju visoku točnost i pouzdanost u različitim uvjetima.[37]

3.2. Proces prepoznavanja lica

Prepoznavanje lica se sastoji od nekoliko faza [37]: detekcije lica, ekstrakcije značajki, i klasifikacije lica. Svaka od ovih faza ima bitnu ulogu koje osigurava preciznost i pouzdanost sustava za prepoznavanje lica.

3.2.1. Detekcija lica

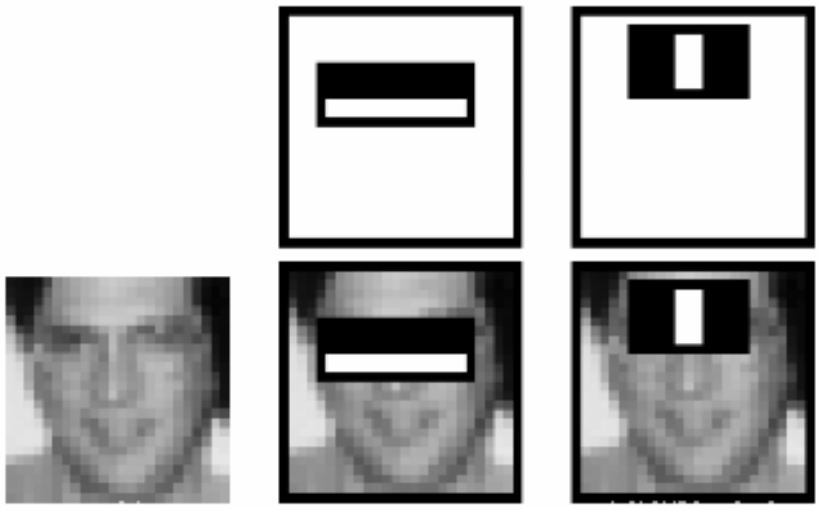
Detekcija lica je prvi i najvažniji korak u procesu prepoznavanja lica. Cilj ove faze je prepoznati i locirati lice na slici ili u videu. Ovaj korak je najvažniji jer bez pravilne detekcije lica, daljnji koraci se ne mogu provesti.

Klasične metode detekcije lica, kao što su Haar Cascades i Histogram Oriented Gradients (HOG), koriste ručno izrađene značajke i klasifikatore za identifikaciju lica.

S razvojem dubokog učenja, konvolucijske neuronske mreže (CNN) postale su dominantna metoda za prepoznavanje lica. CNN-ovi koriste više slojeva za učenje hijerarhijskih značajki iz slikovnih podataka, što im omogućuje prepoznavanje lica u različitim uvjetima, uključujući varijacije u osvjetljenju, kutovima, pa čak i linijama lica.[38]

Haar Cascades je metoda za detekciju objekata, osobito lica, koja je uvedena u radu Paula Viola i Michaela Jonesa.[39] Ova metoda koristi strojno učenje za kreiranje kaskade klasifikatora koji se temelje na jednostavnim značajkama, koje se zovu Haarove značajke. Haarove značajke su pravokutni uzorci koji se primjenjuju na slike kako bi se identificirale razlike u osvjetljenju između različitih područja slike.[39]

Na primjer, jedno od svojstava lica je da su oči često tamnije od okolnog područja (čelo, obrazi). Slika 14 prikazuje dvije različite Haarove značajke. Crno-bijeli pravokutnici na vrhu slike predstavljaju uzorke koji se primjenjuju na sliku lica ispod. Prva značajka detektira razliku u osvjetljenju između očiju i nosa, dok druga značajka detektira razliku u položaju između očiju i nosa. Svaka značajka se izračunava oduzimanjem zbroja piksela ispod bijelog pravokutnika od zbroja piksela ispod crnog pravokutnika. Na primjer, u prvoj značajki, zbroj piksela u crnom pravokutniku (oči) bit će oduzet od zbroja piksela u bijelom pravokutniku (nos/obrazi). Ako su oči tamnije od nosa, rezultat toga će označavati prisutstvo lica.



Slika 14: Detekcija lica, preuzeto sa [40]

Nakon što su značajke izračunate, slijedi selekcija značajki. Zbog mogućnosti velikog broja značajki one se validiraju Adaboost algoritmom koji ima mogućnost razlikovanja lica od ne lica. Nakon selekcije, te značajke grupiraju se u kaskadne klasifikatore, odnosno niz jednos-

tavnijih klasifikatora koji se primjenjuju na dio koji sadrži lice. Zatim se Kaskadnim pristupom eliminiraju područja slike koja vjerojatno ne sadrže lice (primjena jednostavnijih klasifikatora), dok se složeniji klasifikatori primjenjuju samo na ona područja koja su prošla kroz prethodne faze selekcije i imaju veću vjerojatnost za sadržavanje lica. [41]

3.2.2. Ekstrakcija značajki

Nakon što je lice detektirano, slijedi faza ekstrakcije značajki. U ovoj fazi fazi se izdvajaju relevantne informacija s lica koje će se koristiti za identifikaciju. U ovoj fazi sustav detektira i analizira različite karakteristike lica, kao što su udaljenost između očiju, oblik nosa, širina usta, te ostale linije koje čine jedan jedinstven identifikator. Uz pomoć ovog procesa model ima mogućnost razlikovanja jednog lica od drugog.

U tradicionalnim metodama, poput HOG (eng. Histogram of Oriented Gradients) i SIFT (eng. Scale-Invariant Feature Transform), značajke su se ručno definirale na način da su algoritmi bili osmišljeni da identificiraju karakteristike kao što su rubovi, oblici i konture lica, dakle nije postojala automatizacija. S dubokim učenjem, ekstrakcija značajki se automatizirala i postala je preciznija korištenjem konvolucijskim neuronskim mrežama (CNN-ovima). CNN-ovi automatski uče značajke iz sirovih podataka, gdje prvi slojevi mreže prepoznaju jednostavne oblike i rubove, dok kasniji slojevi identificiraju složenije strukture, poput očiju, nosa i usta. Ovaj postupak omogućuje sustavu prepoznavanje lica i u uvjetima kao što je različito osvjetljenje, kutovi, i prepreke ispred lica. [42]

3.2.3. Klasifikacija lica

Klasifikacija lica je zadnja faza u procesu prepoznavanja lica. Nakon što su značajke izdvojene, sustav koristi te informacije za identifikaciju ili verifikaciju identiteta osobe. U ovoj fazi, lice se uspoređuje s bazom poznatih lica kako bi se odredilo podudaranje ili se koristi za kategorizaciju, na primjer, određivanje izraza lica. Različiti algoritmi mogu se koristiti za klasifikaciju, uključujući Support Vector Machines(SVM), k-Nearest Neighbors (k-NN), ili naprednije metode poput potpuno povezanih slojeva u dubokim neuronskim mrežama. CNN-ovi su jako dobri u ovoj fazi jer mogu učiti i komoleksnije razlike između lica čak i kada se radi o sličnim licima. [43]. Klasifikacija može imati poteškoća u prepoznavanju zbog različitih razloga, kao što je starenje, promjena frizuri ili nošenju predmeta. Modeli trenirani na velikom skupu podataka takve poteškoće mogu smanjiti.

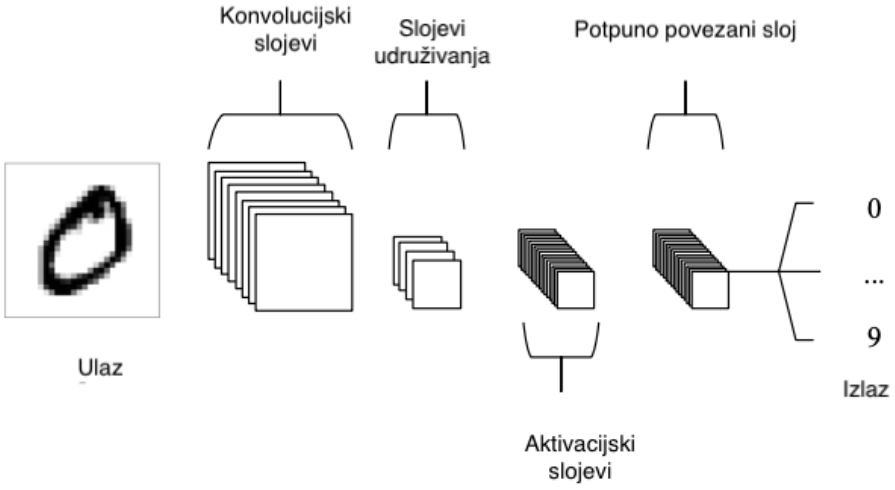
3.3. Tehnologije i modeli u prepoznavanju lica

Razvoj prepoznavanja lica se najviše oslanja na tehnologije i modele strojnog učenja, koji omogućuju visoku preciznost i učinkovitost u različitim uvjetima kao što je starenje. Među najvažnijim tehnologijama koje se koriste u ovom području za prepoznavanje lica su konvolucijske neuronske mreže (CNN), predtrenirani modeli, i tehnika transfernog učenja.

3.3.1. Konvolucijske neuronske mreže (CNN)

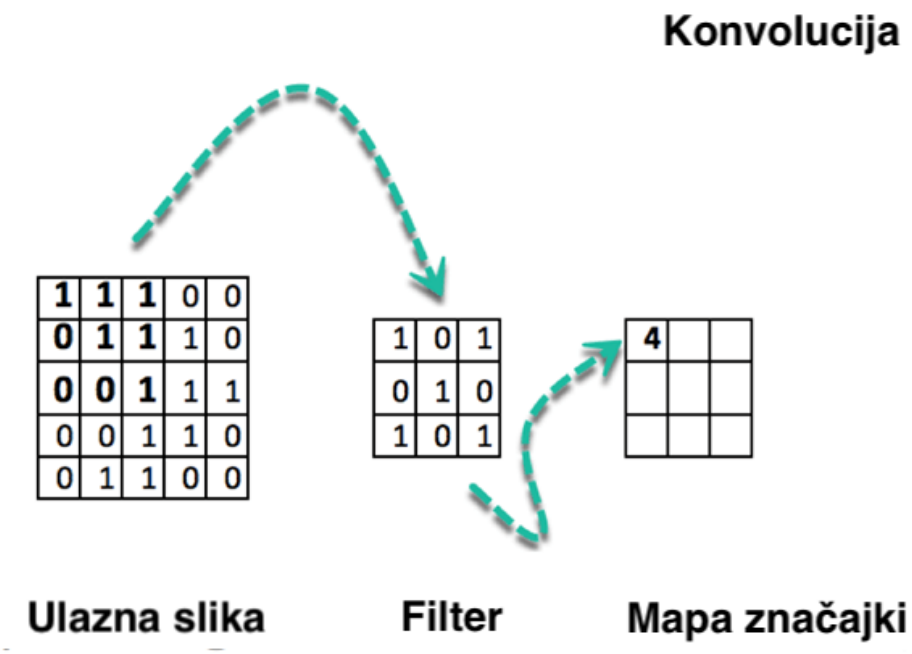
Konvolucijske neuronske mreže (CNN) su jedna od tehnologija u prepoznavanju lica. Osnovni princip rada CNN-a je korištenje konvolucijskih slojeva koji analiziraju slike na različitim razinama kako bi prepoznali sve složenije obrasce i značajke koje bi bilo teško uočiti ručnim metodama. Prvi korak u ovom procesu je ulazni sloj, koji prima sirove pikselne vrijednosti slike. Ove vrijednosti zatim prolaze kroz niz skrivenih slojeva, od kojih svaki izvodi određene operacije nad tim podacima. [44]

Najvažniji skriveni slojevi u CNN-u su konvolucijski slojevi, slojevi za udruživanje i aktivacijski slojevi.[44] Slojevi su prikazani na slici 15.



Slika 15: Slojevi konvolucijske neuronske mreže, preuzeto iz [44]

Konvolucijski slojevi koriste filter koji "klizi" preko slike i izvodi matematičke operacije nad manjim dijelovima slike. Rezultat tih operacija je nova slika, nazvana karta značajki. Karta značajki se sastoji od određenih uzoraka ili oblika na slici. S više različitih filtera, konvolucijski slojevi mogu prepoznati različite značajke na slici. Svrha konvolucijskog sloja je izdvajanje značajki na slici koje će se kasnije koristiti za prepoznavanje lica s novom neviđenom slikom. Slika 16 prikazuje proces konvolucije u kojem se filter primjenjuje na ulaznu sliku kako bi se generirala nova mapa značajki (feature map), koja se sastoji od istaknutih karakteristika slike.



Slika 16: Proces konvolucije, preuzet sa [45]

Slojevi udruživanja smanjuju veličinu mape značajki tako što dijele mapu na manje dijelove i iz svakog biraju najvišu vrijednost. To smanjuje računalnu složenost i čini model otpornijim na manje promjene u slici.[44]

Aktivacijski slojevi pomažu mreži da uči složenije veze između ulaza i izlaza. ReLU je najčešća aktivacijska funkcija koja negativne vrijednosti postavlja na nulu, dok pozitivne ostavlja nepromijenjene. Ova funkcionalnost omogućuje mreži da efikasno uči složene uzorke jer ubrzava izračune, smanjuje problem nestajanja gradijenta te omogućuje modeliranje nelinearnih odnosa u podacima. [44]

Na kraju, izlaz iz svih skrivenih slojeva šalje se potpuno povezanom sloju, koji kombinira sve prikupljene informacije i donosi konačnu odluku, tj. predikciju. U ovom sloju svaki neuron je povezan sa svim neuronima iz prethodnog sloja, omogućujući mreži da napravi točnu procjenu na temelju svih prikupljenih značajki slike. Kroz ovaj proces CNN-ovi mogu prepoznati lice na slikama.

3.3.2. Predtrenirani modeli

predtrenirani modeli su modeli koji su već trenirani na velikim skupovima podataka, kao što su ImageNet, i mogu se ponovno koristiti za specifične zadatke, kao što je prepoznavanje lica. Korištenje predtreniranih modela značajno smanjuje vrijeme i računalne resurse potrebne za treniranje modela od nule.

Jedan od najpoznatijih predtreniranih modela u prepoznavanju lica je VGG16, duboka konvolucijska mreža sa 16 slojeva. VGG16 je treniran na milijunima slika, te je često korišten u ovakvim sustavima zbog svoje jednostavne, ali učinkovite arhitekture.[23] Svaki predtrenirani

model se može s dodatnim slojevima primijeniti specifičnom zadatku što smanjuje vrijeme i resurse za treniranje.

3.3.3. Transferno učenje (engl. Transfer learning)

Transferno učenje je tehnika u strojnom učenju koja omogućuje modelima da primijene znanje stečeno u jednom zadatku na slične zadatke. U kontekstu prepoznavanja lica, transferno učenje omogućuje primjenu predtreniranih modela na nove skupove podataka s relativno malo dodatnog treniranja. Ova tehnika je korisna kada je dostupna količina podataka ograničena ili kada je vrijeme za treniranje modela kritično.

Primjer transfernog učenja može uključivati korištenje predtreniranog modela poput VGG16 kao baze, a zatim dodavanje dodatnih slojeva ili prilagodbu postojećih slojeva kako bi se model specijalizirao za određeni skup podataka lica. Ovaj pristup smanjuje potrebu za velikim količinama podataka za treniranje a dobiva se model koji može postići vrlo visoku točnost.

3.4. Izazovi i etička pitanja u prepoznavanju lica

Prepoznavanje lica je moćna tehnologija, ali sa sobom donosi neka važna etička pitanja. Tehnički gledano, jedan od najvećih izazova je sposobnost sustava da prepozna lica u različitim uvjetima. Lice može izgledati različito zbog promjena u osvjetljenju, kutu snimanja ili izraza lica, što može otežati točno prepoznavanje. Također, modeli koji se koriste za prepoznavanje lica mogu biti previše prilagođeni jednoj klasi ako nisu trenirani na dovoljno različitim skupovima podataka.

Osim tehničkih izazova, prepoznavanje lica može imati problema i s privatnošću. Naime, moguće je prepoznavanje pojedinaca u javnim prostorima bez njihova znanja, što se može zloupotrijebiti. S razvitkom ove tehnologije postoji sve veća zabrinutost oko ugrožavanja privatnosti i slobode ljudi. S pravne strane, još uvijek ne postoji jedinstveni okvir koji regulira upotrebu prepoznavanja lica. Iako neke zemlje već postavljaju ograničenja, kao što je Velika Britanija, kako bi se ublažila zabrinutost, u mnogim dijelovima svijeta pravila nisu uopće definirana ili nisu dovoljno jasno definirana.[46] Europska unija je postavila određene zaštitne mehanizme kroz GDPR, koji regulira upotrebu biometrijskih podataka, no razvoj zakonodavstva još je u tijeku kako bi se pratili ubrzani tehnološki napreci. [47]

Primjena tehnologije prepoznavanja lica nije ograničena samo na javne prostore, već je prisutna i na digitalnim platformama kao što su Google Photos i Facebook. Ove platforme koriste napredne algoritme za prepoznavanje lica na fotografijama, automatski predlažući korisnicima osobe koje bi mogli označiti na temelju tih prepoznatih lica.[48] Iako ove funkcionalnosti mogu poboljšati korisničko iskustvo, otvara se pitanje točnosti identifikacije i zaštite privatnosti korisnika. Korisnici možda nisu uvijek svjesni da su njihova lica analizirana i pohranjena na taj način, što može stvoriti osjećaj nesigurnosti i narušiti privatnost. Pitanje nije samo u prepoznavanju lica u javnim prostorima, već i u načinima na koje se tehnologija koristi na privatnim

fotografijama i društvenim mrežama.

Društveno gledano, primjena prepoznavanja lica može promijeniti način na koji ljudi doživljavaju svoju privatnost i sigurnost. Zbog stalnog nadzora, ljudi bi mogli postati nesigurni ili oprezniji u svojim ponašanjima. Također, tehnologija prepoznavanja lica može povećati društvenu nejednakost, osobito ako sustavi nisu precizni, što može dovesti do netočnih prepoznavanja, pogrešaka i diskriminacije, posebno prema određenim etničkim skupinama.

Zbog tih rizika, EU zahtijeva da svaka primjena ove tehnologije bude proporcionalna, nužna i transparentna, no pravni okviri za nadzor prepoznavanja lica još su u fazi razvoja.[47]

4. Proces treniranja i evaluacija

Proces treniranja i evaluacije modela strojnog učenja sastoji se od nekoliko ključnih koraka koji su bitni za postizanje točnosti i pouzdanosti modela. Prikupljanje podataka, obrada podataka, izgradnja modela za treniranje, evaluacija te implementacija modela.

Slika 17: Proces treniranja modela, preuzeto sa [49]

Prvi korak u ovom procesu je prikupljanje podataka. Prikupljanje velikog broja podataka može bitno utjecat na uspješnost modela jer više podataka omogućuje modelu da bolje prepozna obrasce i bolje klasificira. [49]

Nakon prikupljanja podataka, slijedi obrada i priprema podataka za treniranje. To znači da je podatke potrebno očistiti, ukloniti nepotrebne informacije i uravnotežiti klase odnosno osigurati jednak broj podataka za svaku klasu.

Nakon obrade podataka, oni se dijele na tri skupa: skup za treniranje, validaciju i testiranje. Skup za treniranje koristi se za učenje modela. Model na temelju ovih podataka prepoznaje obrasce i razvija sposobnost klasifikacije ili predviđanja. Validacijski skup služi za podešavanje hiperparametara modela i za sprječavanje pretreniranosti, koja se javlja kada model postane previše prilagođen treniranim podacima i gubi sposobnost generalizacije. Testni skup se koristi za konačnu procjenu performansi modela nakon što je treniranje završeno, kako bi se provjerilo koliko dobro model generalizira na nove, neviđene podatke.[49]

Proces izgradnje modela započinje nakon što su podaci pripremljeni i podijeljeni. Tijekom treniranja, model koristi podatke iz skupa za treniranje kako bi naučio prepoznati značajke i obrasce povezane s različitim klasama ili vrijednostima. Najveći izazov svakog modela je balansiranje između pretreniranosti i podtreniranja.

Pretreniranost se javlja kada model postane previše prilagođen specifičnostima treniranih podataka, što rezultira lošijom generalizacijom na nove podatke, odnosno model ne uči

prepoznavati podatke već pamti već viđene podatke. Dok, podtreniranje nastaje kada model prikupi obrasce u podacima ali se previše prilagodi njima, što rezultira lošim performansama. Kako se ovo ne bi dogodilo moguće je koristiti razne tehnike poput regularizacije, prilagodbe hiperparametara. Posljednja opcija je korištenje drugog modela. [44]

Nakon izgradnje i treniranje modela, model se evaluira korištenjem testnog skupa podataka. Ovaj korak omogućuje procjenu točnosti i pouzdanosti modela nad podacima koji nisu korišteni tijekom treniranja. Evaluacijom je moguće dobiti razne metrike poput točnosti, preciznosti, odziva i F1 rezultat-a, kako bi se procijenilo koliko dobro model radi na stvarnim zadacima.

Kako bi se osigurala dobra generalizacija modela, važno je koristiti podatke za validaciju, koji nisu korišteni tijekom treniranja i testiranja. Ovaj korak osigurava da model može dobro generalizirati na nove podatke u stvarnim situacijama. Nakon treniranja za prilagođavanje novih podataka u model može se koristiti proces finog podešavanja (engl. fine-tuning) koji omogućuje da se već obučeni model prilagodi manjem i specifičnom skupu podataka. Kod ovog procesa se zamrzavaju slojevi kako bi model zadržao prethodno znanje koje je stekao na prvom treniranju i iskoristio za novi skup podataka. Time se štedi vrijeme jer nema potrebe za treniranjem novog modela, a i resursi jer se uglavnom ovaj proces provodi na manjem skupu podataka. Teorijski se svi procesi od sakupljanja podataka, treniranja i same evaluacije čine vrlo jednostavnim, no u sljedećem poglavlju se detaljno prolazi kroz praktični dio gdje su prvi koraci upravo ovi procesi.

5. Razvoj modela strojnog učenja

U ovom poglavlju je opisan cjelokupan proces razvoja modela strojnog učenja, od prikupljanja i obrade podataka do treniranja modela i njegove evaluacije. Proces je podijeljen u nekoliko ključnih faza koje su detaljno razrađene kako bi se osiguralo da je krajnji rezultat bio precizan i pouzdan model za prepoznavanje lica. Faze prikupljanje podataka, obrade podataka i treniranje modela.

5.1. Prikupljanje podataka

Inicijalno su podaci prikupljeni ručno. Pretraživanjem interneta i prikupljanjem slika osoba za koje se željelo provoditi detekciju, prikupljeno je oko 30 slika po osobi, što je predstavljalo prvi izazov zbog malog broja podataka. Ove slike su zatim prebačene na računalo, gdje je provedena njihova inicijalna obrada.

Kako bi se osiguralo jednostavnije imenovanje i organizacija slika, svaka slika je preimenovana pomoću skripte (Isječak koda 1) koja je dodjeljuje numeričke nazive slikama (linija 12) i spremala ih u odgovarajući direktorij (linija 15). Preimenovanje je kasnije poslužilo radi lakšeg provođenja testiranja na skupu za testiranje. Ova skripta obrađuje slike iz zadanog direktorija, imenuje ih prema broju, i zamjenjuje izvorne datoteke novim, pravilno imenovanim slikama.

```
1  import os
2  import cv2
3  input_dir = "original"
4  files = os.listdir(input_dir)
5  files.sort()

6  for count, file in enumerate(files, start=1):
7      img_path = os.path.join(input_dir, file)
8      img = cv2.imread(img_path)

9      if img is None:
10         print(f"Warning: Could not read image {img_path}")
11         continue

12     new_file_name = f"{count}.jpg"
13     new_file_path = os.path.join(input_dir, new_file_name)
14     cv2.imwrite(new_file_path, img)
15     os.remove(img_path)
16     print(f"Renamed {file} to {new_file_name}")
```

Isječak koda 1: Skripta za preimenovanje slika

5.2. Obrada podataka

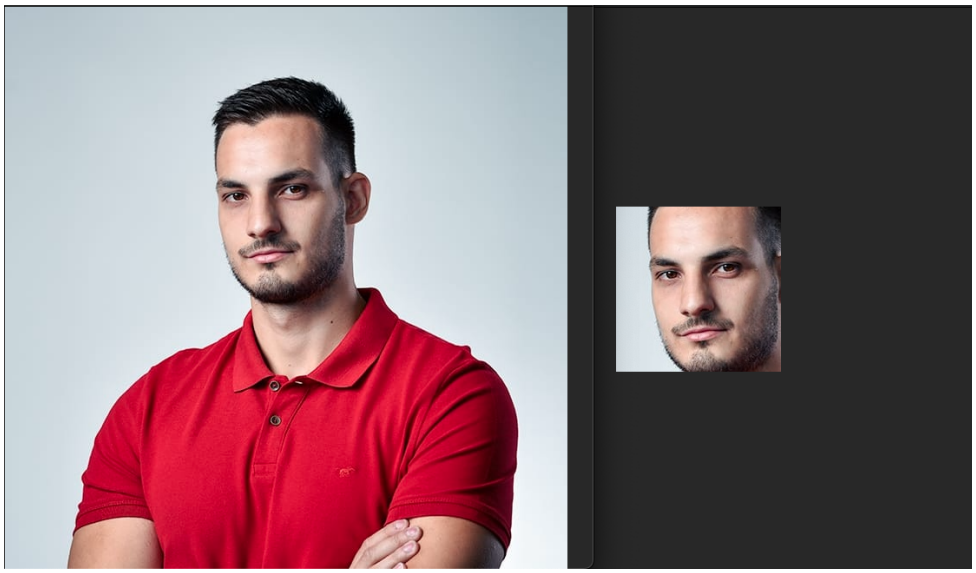
Originalne slike sadržavale su puno šuma i nepoželjnih elemenata u pozadini, što bi moglo ometati proces treniranja modela. Kako bi se smanjila mogućnost da model nauči nebitne detalje iz pozadine, bilo je potrebno ukloniti šumove i izolirati lica s fotografija. Ovaj korak proveden je pomoću metode za detekciju lica Haar cascade. Skripta je automatski detektirala lica na slikama, izrezala ih i spremila u poseban direktorij, eliminirajući tako većinu šuma.

Za ovaj proces koristila se sljedeća isječak koda 2.

```
1  import os
2  import cv2
3  path = "mv_test_new"
4  images_to_crop_path = "images/mv_test_unmirrored"
5  os.makedirs(path, exist_ok=True)
6  files = os.listdir(images_to_crop_path)
7  print(f"Files to process: {len(files)}")
8  count = 1
9  face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
10 for file in files:
11     img_path = os.path.join(images_to_crop_path, file)
12     img = cv2.imread(img_path)
13     if img is None:
14         print(f"Warning: Could not read image {img_path}")
15         continue  # skip to the next file if the image is not read successfully
16     try:
17         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
18         faces = face_cascade.detectMultiScale(gray, 1.3, 4)
19         if len(faces) == 0:
20             print(f"No faces found in image {file}")
21         else:
22             print(f"Found {len(faces)} face(s) in image {file}")
23             for (x, y, w, h) in faces:
24                 roi_color = img[y:y + h, x:x + w]
25                 cv2.imwrite(os.path.join(path, f"{count}.jpg"), roi_color)
26                 count += 1
27     except Exception as e:
28         print(f"An error occurred while processing {file}: {e}")
29 print(f"Processed {count-1} faces out of {len(files)} images.")
```

Isječak koda 2: Skripta za ekstrakciju lica

Skripta prvo provjerava sve slike unutar zadanog direktorija i zatim koristi Haar Cascades metodu za detekciju lica (linija 18). Za svaku detektiranu sliku, skripta provjerava može li se učitati slika, a zatim koristi algoritam za detekciju lica. Ako se lica detektiraju, skripta izreže dio slike koji sadrži lice te ga sprema u novi direktorij (linija 24 i 25). Time se osigurava da će treniranje modela biti fokusirano na lica, a ne na nepotrebne elemente iz pozadine. Sljedeća slika prikazuje primjer nakon pokretanja skripte.



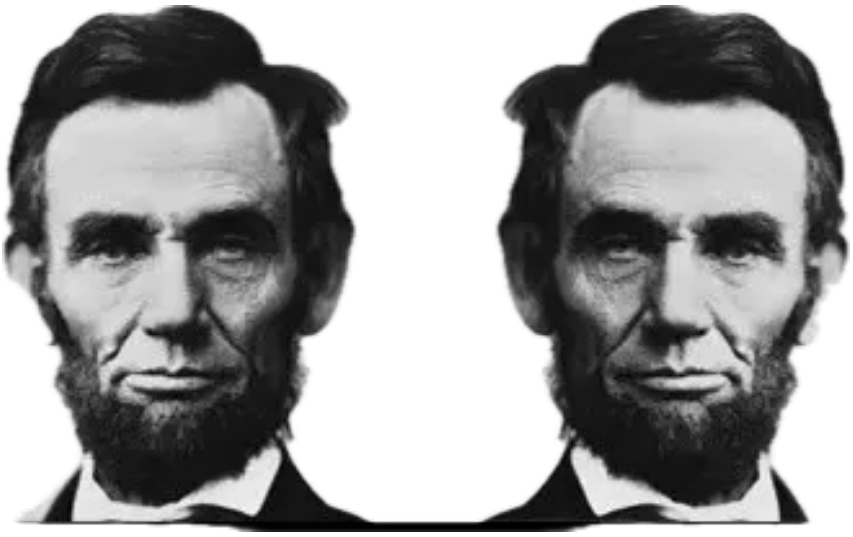
Slika 18: Prepoznavanje lica

Dodatni problem pojavio se kod slika koje su bile okrenute po horizontalnoj osi, što bi potencijalno moglo zbuniti model tijekom treniranja. Takve slike su se identificirane kroz pregled svih slika jedne osobe i radila usporedba sa pravom slikom. Kako bi se izbjegla ova konfuzija i osiguralo da su sve slike konzistentne, identificirane slike su ručno okrenute pomoću skripte. Skripta (Isječak koda 3) koristi OpenCV biblioteku za čitanje svake slike iz direktorija (linija 8), okretanje slike horizontalno (linija 12) te spremanje slike u novi direktorij (linija 14). Na ovaj način, sve slike su bile ujedinjene prije nego što su bile proslijeđene modelu na treniranje.

```
1  import os
2  import cv2
3  unmirrored_dir = "images/mv_test_unmirrored"
4  mirrored_dir = "images/mv_test"
5  os.makedirs(unmirrored_dir, exist_ok=True)
6  for file in os.listdir(mirrored_dir):
7      img_path = os.path.join(mirrored_dir, file)
8      img = cv2.imread(img_path)
9      if img is None:
10         print(f"Warning: Could not read image {img_path}")
11         continue

12     img_unmirrored = cv2.flip(img, 1)
13     output_path = os.path.join(unmirrored_dir, file)
14     cv2.imwrite(output_path, img_unmirrored)
15     print(f"Saved unmirrored image to {output_path}")
```

Isječak koda 3: Skripta za okretanje slike po horizontalnoj osi

Skripta je automatski procesuirala sve slike u zadanom direktoriju, osiguravajući da su sve okrenute slike ispravno okrenute i spremljene u novi direktorij. Na slici 19 moguće je vidjeti primjer prije i nakon korištenja skripte. Time je minimizirana mogućnost zbunjivanja modela i potencijalno poboljšana točnost prepoznavanja.



Slika 19: Primjer zrcaljenja Lincoln, preuzeto sa [50]

Nakon što su podaci bili obrađeni, potrebno je bilo razvrstati slike prema osobama. Taj proces se radio ručno na način da se za svaku osobu kreirao direktorij te svakom direktoriju pridružila slika te osobe. Kasnije je taj proces isto automatiziran.

5.3. Proces treniranja

Nakon što su svi podaci bili spremni pristupilo se fazi treniranja modela. Korištene su različite metode i pristupi kako bi se osiguralo da model može generalizirati i točno prepoznavati lica u različitim uvjetima. Modeli su se trenirali preko platforme Kaggle. Kaggle je online platforma za strojno učenje i ona je omogućila razvoj modela korištenjem resursa poput grafičkih kartica čime je proces treniranja bio ubrzan, dok proces treniranja na lokalnoj mašini zahtjeva dodatna podešavanja okruženja koje je bilo dugotrajno zbog različitih ovisnosti jedne biblioteke za drugom odnosno verzije.

Prije detaljne obrade implementacije modela, biti će objašnjeni početni dio svakog modela koji je ujedno i isti kroz neke preinake poput veličine serije (batch). Za početak, proces treniranja uključuje korake učitavanje podataka, podjela podataka na skupove, obrada podataka, i pretvorba podataka.

Prvi korak je učitavanje podataka.U isječku koda 4 funkcija "load_dataset()" se koristi za učitavanje slika iz zadanog direktorija i njihovo klasificiranje. Ova funkcija pregledava sve slike u direktoriju, provjerava njihovu valjanost kako bi se izbjegle oštećene ili neispravne datoteke, te svakoj slici dodjeljuje odgovarajuću klasu na temelju imena direktorija (linija 11). Cilj ove funkcije je ekstrakcija ispravnih slika kako bi se mogle koristiti pri procesu treniranja.

```
def load_dataset(dataset_path, label):
    image_paths = []
    labels = []
    for file in os.listdir(dataset_path):
        if file.endswith(('.jpg', '.png', '.jpeg')):
            file_path = os.path.join(dataset_path, file)
            try:
                img = Image.open(file_path)
                img.verify()
                image_paths.append(file_path)
                labels.append(str(label))
            except (UnidentifiedImageError, IOError):
                print(f"Skipping corrupted image: {file_path}")
                continue
    return np.array(image_paths), np.array(labels)
```

Isječak koda 4: Funkcija za učitavanje podataka

Nakon učitavanja, slijedi podjela podataka na skupove za treniranje, validaciju i testiranje (isječak koda 5). Podaci su podijeljeni tako da se 70% koristi za treniranje modela(linija 1), a preostalih 30% je rezervirano za dodatnu podjelu podataka na skup za validaciju i testiranje od kojih svaki sadrži 15% izvornih podataka (linija 2). Skup za treniranje pomaže modelu naučiti zadatak, validacijski skup služi za prilagodbu modela tijekom treniranja, dok se testni skup koristi za konačnu provjeru njegove točnosti.

```
X_train, X_temp, Y_train, Y_temp = train_test_split(all_image_paths, all_labels,
↪  test_size=0.3, random_state=42)
X_val, X_test, Y_val, Y_test = train_test_split(X_temp, Y_temp, test_size=0.5,
↪  random_state=42)
```

Isječak koda 5: Skripta za podjelu podataka na skupove

Obrada podataka (isječak koda 6) provodi se pomoću "ImageDataGenerator" klase, koja omogućuje povećanje podataka, na način da se koriste različite tehnike kao što su rotacija, horizontalno pomicanje, uvećanje (zooming) i promjena svjetline te se time generiraju dodatne varijacije originalnih slika. Uz pomoć ovoga se pomaže modelu da za iste slike dobije razne inačice koje onda pomažu u generaliziranju. Ovaj korak smanjuje rizik od pretreniranosti i poboljšava sposobnost modela da prepozna lica u različitim uvjetima.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    pd.DataFrame({'filename': X_train, 'class': Y_train}),
    x_col='filename',
    y_col='class',
    target_size=(128, 128),
    batch_size=50,
    class_mode='sparse'
)

validation_generator = val_test_datagen.flow_from_dataframe(
    pd.DataFrame({'filename': X_val, 'class': Y_val}),
    x_col='filename',
    y_col='class',
    target_size=(128, 128),
    batch_size=50,
    class_mode='sparse'
)
```

Isječak koda 6: Skripta za obradu podataka

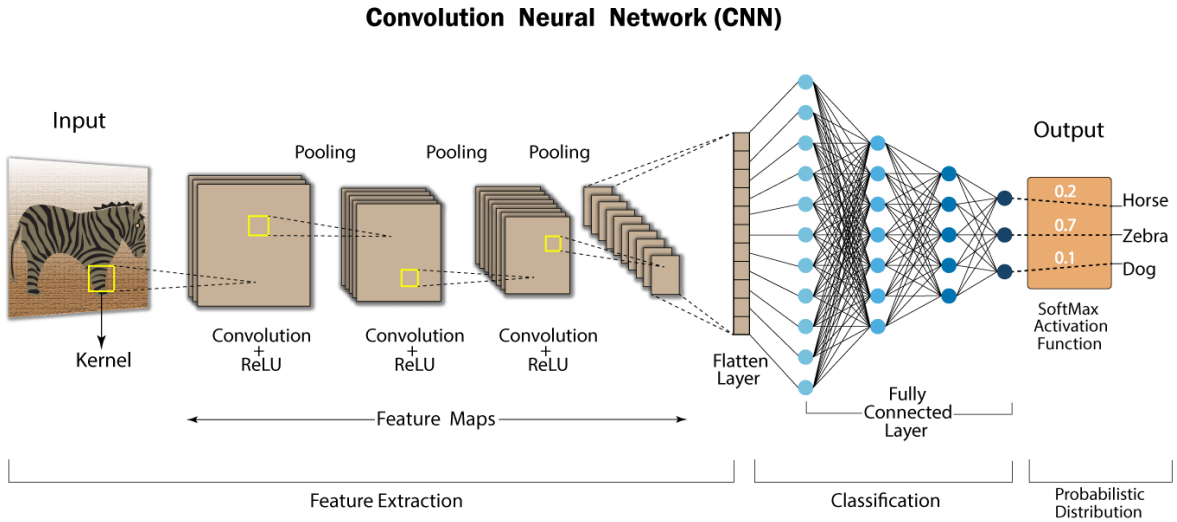
Konačno, podaci se pretvaraju u tf.data.Dataset format pomoću funkcije `generator_to_tfdata()`. Ova funkcija pretvara generatore podataka u tf.data.Dataset. Korištenje tf.data.Dataset omogućava modelu da lakše upravlja velikim skupovima podataka u okruženju s ograničenim resursima jer optimizira performanse zbog paralelne obrade. Ovo formatiranje je bilo potrebno radi funkcije repeat() koja pomaže generatorima podataka da stalno pružaju podatke tijekom cijelog procesa treniranja modela. Isječak koda 7.

```
def generator_to_tfdata(generator):
    def gen():
        while True:
            for x, y in generator:
                yield x, y
    return tf.data.Dataset.from_generator(gen, output_signature=(
        tf.TensorSpec(shape=(None, 128, 128, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None,), dtype=tf.float32),
    ))

train_ds = generator_to_tfdata(train_generator).repeat()
val_ds = generator_to_tfdata(validation_generator).repeat()
test_ds = generator_to_tfdata(test_generator)
```

Isječak koda 7: Skripta za generiranje podataka

5.3.1. Vlastita implementacija konvolucijske neuronske mreže (CNN)

Prvi model razvijen je koristeći vlastitu implementaciju konvolucijske neuronske mreže (CNN). Kako bi model mogao automatski učiti i prepoznavati različite osobe potrebno je definirati slojeve za izdvajanje značajki koji služe za prepoznavanje osnovnih oblika i uzoraka kao što su rubovi, teksture i različite složenije oblike. Nakon što su dodatno obrađeni podatci, potrebno je definirati vlastiti model sa slojevima. Kako bi to osigurali kreirala se funkcija`create_cnn_model` koja vraća CNN model sa slojevima. Slojevi su se kreirali prema različitim primjerima iz prakse prateći pristup koji je prikazan na slici 20, te se radile preinake u ovisnosti o rezultatu modela kako bi se pronašao najbolji.



Slika 20: CNN slojevi preuzeto sa [51]

Za omogućivanje dodavanja slojeva u model jedan iza drugoga koristila se funkcija Sequential(). Isječak koda 8.

```
def create_cnn_model(input_shape, num_classes):
    model = Sequential()
```

Isječak koda 8: Skripta za omogućivanje dodavanja slojeva

Nakon što je omogućeno dodavanje slojeva, u prvi blok slojeva dodaju se konvolucijski slojevi, normalizacija, pooling, i dropout. Isječak koda 9.

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape,
↪  kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
```

Isječak koda 9: Prvi blok slojeva CNN

Conv2D(32, (3, 3)): Ovo je prvi konvolucijski sloj koji ima 32 filtera, svaki veličine 3x3 piksela. Ovaj sloj izvlači osnovne značajke poput rubova iz ulazne slike.

activation='relu': ReLU (Rectified Linear Unit) je aktivacijska funkcija koja postavlja sve negativne vrijednosti na nulu.

input_shape=input_shape: ulazni oblik u ovom slučaju 128x128 piksela s 3 kanala za boje.

BatchNormalization(): Normalizira podatke, čime se ubrzava treniranje. Korisno je za sprječavanje modela da slabo uči ili odluta u nepoželjni smjer.

MaxPooling2D(pool_size=(2, 2)): Smanjuje dimenzije karte značajki tako što uzima maksimalnu vrijednost iz svake 2x2 regije. Smanjuje broj parametara i pomaže modelu da se usredotoči na najvažnije značajke. Manja karta značajki znači manje podataka koje treba obraditi u sljedećim slojevima mreže, što smanjuje vrijeme obrade i potrebnu memoriju. Dropout(0.3): Isključuje 30% neurona nasumično tijekom treniranja kako bi se spriječila prenaučenost modela.

Nakon toga slijede drugi, treći i četvrti blok (isječak koda 10) koji su skoro isti kao prvi uz robusnije parametre. Ti blokovi omogućuju modelu da uči na složenijim obrascima s kombinacijom prethodnih značajki. Prvi blok uči jednostavne značajke kao što su rubovi, kutovi i teksture odnosno može prepoznati rubova nosa ili očiju. Drugi blok će uz prethodne značajke proširiti i prepoznati veće dijelove lica čime se i povećava sposobnost modela da bolje generalizira podatke. Generalno više slojeva može pomoći modelu u povećanju robusnosti modela odnosno model može postati otporniji na razne šumove i varijacije u podacima. No, previše slojeva može dovesti model i do prenaučenosti gdje model neće moći dobro generalizirati na nove podatke. U nastavku slijede drugi i treći blok sa različitim parametrima.

```
1  def create_cnn_model(input_shape, num_classes):
2      model = Sequential()
3      model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
       ↪  kernel_initializer='he_normal'))
4      model.add(BatchNormalization())
5      model.add(MaxPooling2D(pool_size=(2, 2)))
6      model.add(Dropout(0.3))

7      model.add(Conv2D(128, (3, 3), activation='relu', padding='same',
       ↪  kernel_initializer='he_normal'))
8      model.add(BatchNormalization())
9      model.add(MaxPooling2D(pool_size=(2, 2)))
10     model.add(Dropout(0.4))

11     model.add(Conv2D(256, (3, 3), activation='relu', padding='same',
       ↪  kernel_initializer='he_normal'))
12     model.add(BatchNormalization())
13     model.add(MaxPooling2D(pool_size=(2, 2)))
14     model.add(Dropout(0.4))
```

Isječak koda 10: Kreiranje CNN modela

Drugi konvolucijski sloj koristi 64 filtera (linija 3), čime povećava sposobnost modela da uči složenije značajke. parametar padding='same'(linija 3) osigurava da izlazna mapa značajki ima iste dimenzije kao ulazna. Treći konvolucijski sloj (linije 7-10) koristi 128 filtera, za još složenije značajke te Dropout(0.4): u ovom sloju, 40% neurona je isključeno kako bi se još više smanjila mogućnost prenaučenosti. Četvrti konvolucijski sloj (linije 11-14) koristi 256 filtera, za još složenije značajke te Dropout(0.4): u ovom sloju, 40% neurona je isključeno.

Nakon što su definirani sakriveni slojevi, potrebno je definirati sloj za potpuno povezivanje i izlaz. 11.

```
1  model.add(Flatten())
2  model.add(Dense(128, activation='relu'))
3  model.add(Dropout(0.5))
4  model.add(Dense(num_classes, activation='softmax'))
5  return model
```

Isječak koda 11: Slojevi za potpuno povezivanje i izlaz

Nakon što konvolucijski slojevi izdvoje značajke iz slike, te značajke trebaju biti preoblikovane u format koji model može koristiti za donošenje odluka. Uz pomoć Flatten() sloj pretvara sve te značajke u vektor čime je omogućeno povezivanje s potpuno povezanim slojevima. Dense sloj s 128 neurona kombinira te značajke kako bi model mogao prepoznati složenije stvari. Kako bi se spriječila predtreniranost nasumično se isključi 50% neurona tijekom treniranja čime se sprječava model da postane previše ovisan o specifičnim značajkama tokom treniranja. Na kraju se koristi Dense sloj koji ima onoliko neurona koliko ima različitih klasa. Taj sloj koristi softmax aktivaciju, koja pretvara izlaz modela u vjerojatnosti za svaku

moguću klasifikaciju, što omogućuje modelu da predvidi kojoj kategoriji pripada slika.

Zatim slijedi kompilacija samog modela čime se definira način na koji će model učiti i vrednovati tokom treniranja. Kada se definira model, potrebno je odabrati parametre poput funkcije gubitaka, optimizator te metričke funkcije kako bi model mogao dobro učiti. Prvo, odabire se funkcija gubitaka (loss function), koja mjeri koliko su predikcije modela točne. Model tijekom treniranja pokušava smanjiti razliku između svojih predikcija i stvarnih rezultata. Najčešće za ovakve zadatke se koristi `sparse_categorical_crossentropy`.

Zatim se bira optimizator, koji služi za podešavanje težina modela. Postoji nekoliko opcija optimizatora poput Stochastic Gradient Descent (SGD), Adam i RMSprop, no prema istraživanju [52] ispostavlja se kako je Adam optimizator najbolji jer kombinira prednosti drugih metoda čime pomaže modelu da brže i stabilnije uči.

Na kraju se postavlja metričke funkcije (metrics) kako bi se pratilo stanje modela tijekom treniranja i moglo prekinuti ukoliko je potrebno. Prema drugim primjerima, a i prema teoriji, najčešće se koristi točnost (accuracy), koja pokazuje koliko je model točno predvidio odgovore. Isječak koda 12.

```
1 cnn_model.compile(optimizer="adam", loss='sparse_categorical_crossentropy',
  ↪  metrics=['accuracy'])
```

Isječak koda 12: Kompilacija modela

Kod treniranja modela potrebno je pozvati funkciju fit(), kojoj se pridruže razni parametri poput podataka za treniranje, podataka za validaciju, broj ponavljanja tijekom kojih će se model trenirat, broj koraka unutar jedne epohe (najčešće na temelju broja uzoraka podijeljen sa veličinom batch-a, tako da se kroz svaku sliku prođe barem jednom), te povratne sprege odnosno akcija koje bi se mogle izvršiti ukoliko model više ne uči, ukoliko dolazi do smanjenja točnosti ili spremanje najboljeg modela. Isječak koda 13.

```
1  # Define callbacks for training
2  reduce_lr_cnn = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=20,
   ↪  min_lr=1e-4)
3  early_stop_cnn = EarlyStopping(monitor='val_loss', patience=60,
   ↪  restore_best_weights=False)
4  checkpoint_cnn = ModelCheckpoint('best_cnn_model.keras', monitor='val_loss',
   ↪  save_best_only=True, mode='min')

5  history_cnn = cnn_model.fit(
6      train_ds,
7      steps_per_epoch=steps_per_epoch,
8      epochs=100,
9      validation_data=val_ds,
10     validation_steps=validation_steps,
11     callbacks=[reduce_lr_cnn, early_stop_cnn, checkpoint_cnn]
12 )
```

Isječak koda 13: Definiranje povratne sprege

Funkcija ReduceLROnPlateau (linija 2) smanjuje stopu učenja kada model zapne u lokalnom minimumu ili kada nema poboljšanja u gubitku na validacijskom skupu tijekom određenog broja epoha. To pomaže modelu da dublje krene učiti odnosno detaljnije ukoliko se smanjuje stopa učenja ili sažetije ukoliko se poveća stopa. U ovom modelu se koristi smanjenje stope učenja ukoliko vrijednost gubitka na validacijskom skupu podataka se ne poboljša odnosno smanji nakon 20 uzastopnih. U tom slučaju stopa učenja se smanjuje za faktor 0.2 te će se ona smanjivati do minimalne vrijednosti, a to je 0.00001.

Funkcija EarlyStopping (linija 3) koristi se za zaustavljanje treniranja modela kada se performanse na validacijskom skupu prestanu poboljšavati nakon određenog broja epoha koji se definiraju u parametru patience. Ako je `restore_best_weights` postavljen na "True" odnosno istina, model će se vratiti na stanje s najboljim validacijskim rezultatima. Ako je postavljen na False odnosno laž, model će zadržati težine iz posljednje epohe treniranja. Postavljanje `restore_best_weights` na "False" se najčešće koristi ukoliko se smatra da model još uvijek ući ili se planira dodatno fino podešavanje kao što je ovdje slučaj. U ovom modelu se koristi zaustavljanje ukoliko vrijednost gubitka na validacijskom skupu podataka se ne poboljša odnosno smanji nakon 60 slijedećih epoha te se ne vraća najboljim validacijskim rezultatima.

Funkcija ModelCheckpoint (linija 4) služi za spremanje modela kada god postigne bolje performanse na validacijskom skupu, čime se osigurava verzija modela s najboljim rezultatom. U ovom slučaju također se prati vrijednost validacijskog gubitka te se kao mod koristi "min" odnosno najmanje jer se za tu vrijednost s želi postići najmanji gubitak odnosno biti što bliži broju 0.

Za treniranje modela smatralo se kako je 100 epoha dovoljno za treniranje u koracima ukupnog broja uzoraka podijeljeno sa veličinom serija (batch). To je broj uzoraka koji se obrađuje u jednom prolazu kroz model tijekom treniranja. Veća veličina serije može ubrzati treniranje, ali zahtijeva više memorije, dok manja veličina serije može usporiti treniranje, ali može modelu poboljšati generalizaciju. Nakon početnog treniranja modela, postupak finog pode-

šavanja započinje odmrzavanjem (unfreezing) posljednjeg sloja baznog modela. Nakon toga potrebno je ponovno model kompajlirati te se ovaj put stopa učenja postavlja na 1e-4, kako bi model dodatno učilo nad podacima a i kako bi se izbjegla destabilizacija modela. Broj epoha smanjen na 80 uz isti broj koraka po epohi. Isječak koda 14.

```
for layer in cnn_model.layers[:-1]:
    layer.trainable = False

cnn_model.compile(optimizer=Adam(learning_rate=1e-4),
↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history_fine = cnn_model.fit(
    train_ds,
    steps_per_epoch=steps_per_epoch,
    epochs=80,
    validation_data=val_ds,
    validation_steps=validation_steps,
    callbacks=[reduce_lr_cnn, early_stop_cnn, checkpoint_cnn]
)
```

Isječak koda 14: Treniranje modela s parametrima

5.3.2. VGG16 implementacija

Drugi pristup uključivao je korištenje unaprijed treniranog modela VGG16, jednog od najpopularnijih modela za obradu slika. Ukoliko skup podataka nad kojim se trenira nije pretjerano velik odnosno do 500-injak podataka, preporučljivo je koristiti unaprijed trenirani model kao što je VGG16 ili MobileNetV2 jer su ti modeli već trenirani na velikom skupu podataka i sadrže veoma dobre značajke. Ova implementacija modela također sadrži fino podešavanje (fine-tuned) na prikupljenim podacima nakon završetka inicijalnog treniranja radi što bolje generalizacije.

VGG16 je prethodno treniran na velikom skupu podataka ImageNet, koji se sastoji od preko 14 milijuna označenih slika koje su raspoređene u više od 20.000 različitih kategorija ili klasa, te je vrlo dobar model na području računalnog vida.

Umjesto korištenja izvorne VGG16 arhitekture s njenim vlastitim klasifikacijskim slojevima, koristi se samo bazni dio modela bez zadnjeg sloja, sloja za klasifikaciju(includeTop=False). Isječak koda 15.

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

)
```

Isječak koda 15: Definiranje VGG16 modela sa dodatnim slojevima

Uz bazni dio modela, dodani su još slojevi kao što je GlobalAveragePooling2D koji zamjenjuje klasično izravnavanje (Flatten) globalnim prosječnim poolingom. Umjesto da jednostavno izravna sve vrijednosti značajki, on uzima prosječnu vrijednost svake značajke preko cijele karte značajki, smanjujući dimenzionalnost i smanjujući broj parametara.

Nakon toga, Dense sloj s 512 neurona omogućuje modelu da nauči složenije obrasce u podacima. Aktivacijska funkcija ReLU uvodi nelinearnost, što pomaže modelu u prepoznavanju i razumijevanju složenijih veza između značajki. Nakon toga sa Dropout-m se nasumično isključuje 50% neurona radi bolje generalizacije na nove podatke, s čime je poželjno bilo koristiti i normalizaciju podataka za samu stabilizaciju modela. Na kraju se koristi Dense sloj koji ima onoliko neurona koliko ima različitih klasifikacija. Taj sloj koristi softmax aktivaciju, koja pretvara izlaz modela u vjerojatnosti za svaku moguću klasifikaciju, što omogućuje modelu da predvidi

kojoj kategoriji pripada slika. Na kraju je stvoren model koji kao bazu koristi prethodno trenirani model VGG16 sa dodatnim izmjenama za bolje rezultate na manjem skupu podataka.

Nakon podešavanje slojeva na modelu, potrebno je bilo isti kompajlirati. Koristili su se isti parametri kao i na prethodnom CNN modelu, gdje se pri tome kao početna stopa učenja postavlja na 0.0001. Kasnije se kroz povratnu spregu ona prilagođava uz uvjet da nakon 3 epohe bez promjene ili odlaska na gore se stopa učenja smanji. Za prijevremeno zaustavljanje se odabralo 10 epoha gdje bi se pratio validacijski gubitak. Isječak koda 16.

```
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
↪  metrics=['accuracy'])

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
↪  min_lr=1e-6)
early_stop = EarlyStopping(monitor='val_loss', patience=10,
↪  restore_best_weights=True)
checkpoint = ModelCheckpoint('best_vgg16_model.h5', monitor='val_loss',
↪  save_best_only=True, mode='min')

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=55,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[reduce_lr, early_stop, checkpoint]
)
```

Isječak koda 16: Kompajliranje VGG16 modela

Za treniranje modela pokazalo se kako je 55 epoha dovoljno za treniranje ovog modela u koracima od broja elemenata koji su se generirali za skup za treniranje. Nakon početnog treniranja modela s osnovnim slojevima VGG16 zamrznutim, postupak finog podešavanja započinje odmrzavanjem posljednjih šest slojeva osnovnog modela (isječak koda 17). Ovi slojevi se sada mogu trenirati, što je omogućilo modelu da se prilagodi specifičnim karakteristikama novog skupa podataka. Nakon toga potrebno je ponovno model kompajlirati te se ovaj put stopa učenja postavlja na 1e-5, kako bi model dodatno učio nad podacima, a i kako bi se izbjegla destabilizacija modela.

```
for layer in base_model.layers[-6:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5),
    loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=55,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[reduce_lr, early_stop, checkpoint]
)
```

Isječak koda 17: Treniranje modela VGG16

Zatim se model nastavlja trenirati koristeći prethodno definirane podatke i callback funkcije kroz 55 epoha.

5.3.3. MobileNetV2

U ovom pristupu, kao bazni model koristi se MobileNetV2, koji je prethodno treniran na velikom skupu podataka poput ImageNeta (isječak koda 18). Kao i kod VGG16, posljednji sloj baznog modela je isključen (includeTop=False), što omogućava prilagodbu modela specifičnim zadacima, u ovom slučaju klasifikaciji s manjim brojem klasa. Na vrh baznog modela dodani su novi slojevi, identični kao i za VGG16 uz drugačije parametre kao što je primjer kod Dense sloja s 64 neurona, koji je manji u odnosu na sloj s 512 neurona u VGG16 modelu iz razloga što je arhitektura VGG16 modela puno složenija i preporučljivo je koristiti veći broj neurona kako bi se zadržala složenost modela.

Završni sloj modela je Dense sloj s brojem neurona koji odgovara broju klasa i koristi softmax aktivaciju kako bi osigurao da izlazi modela predstavljaju vjerojatnosti koje zbrojene daju 1. Nakon definiranja svih slojeva, model je spreman za treniranje.

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224,
    224, 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)  # Further reduced complexity
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

Isječak koda 18: Definiranje MobileNetV2 modela

Nakon što su svi slojevi osnovnog modela zamrznuti kako bi se spriječilo njihovo mije-

njanje tijekom početnog treniranja, model se mora kompajlirati kako bi bio spreman za treniranje Isječak koda 19. Za kompajliranje se koristi optimizator Adam s početnom stopom učenja od 0.0001. Definirane su i povratne sprege za treniranje modela kako bi se optimizirala stopa učenja i izbjegla prenaučenost. ReduceLROnPlateau smanjuje stopu učenja za faktor 0.5 kad se validacijski gubitak ne poboljšava tijekom sljedećih 5 epoha, s minimalnom stopom učenja postavljenom na 1e-6. EarlyStopping je postavljen s pragom od 15 epoha, pri čemu prati validacijski gubitak. Ako se gubitak ne poboljša tijekom tog perioda, treniranje se zaustavlja, a najbolja verzija modela se vraća. To osigurava da model ne trenira duže nego što je potrebno, što pomaže u izbjegavanju prenaučenosti. Koriste se ponderi klase (class weights) kako bi se model bolje prilagodio neravnoteži u podacima što se događa koje nejednolikog broja podataka u klasama. Oni se izračunavaju pomoću funkcije computeClassWeight, koja osigurava da manje zastupljene klase imaju veći utjecaj na treniranje modela.

```
for layer in base_model.layers:
    layer.trainable = False

optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
↪  metrics=['accuracy'])

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5,
↪  min_lr=1e-6)
early_stop = EarlyStopping(monitor='val_loss', patience=15,
↪  restore_best_weights=True)
checkpoint = ModelCheckpoint('best_mobilenetv2_model.keras', monitor='val_loss',
↪  save_best_only=True, mode='min')

from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight('balanced', classes=np.unique(Y_train),
↪  y=Y_train)
class_weights = dict(enumerate(class_weights))

history = model.fit(
    train_ds,
    steps_per_epoch=steps_per_epoch,
    epochs=50,
    validation_data=val_ds,
    validation_steps=validation_steps,
    class_weight=class_weights,
    callbacks=[reduce_lr, early_stop, checkpoint]
)
```

Isječak koda 19: Treniranje MobileNetV2 modela

Nakon početnog treniranja modela s osnovnim slojevima MobileNetV2 zamrznutim, postupak finog podešavanja započinje otključavanjem (unfreezing) posljednjih 5 slojeva osnovnog modela (isječak koda 20). Ovi slojevi se sada mogu trenirati, što je omogućilo modelu da se prilagodi specifičnim karakteristikama novog skupa podataka. Nakon toga potrebno je ponovno model kompajlirati te se ovaj put stopa učenja postavlja na 1e-5, kako bi model dodatno

dublje učilo nad podacima a i kako bi se izbjegla destabilizacija modela.

```
for layer in base_model.layers[-5:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5),
    loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_ds,
    steps_per_epoch=steps_per_epoch,
    epochs=50,
    validation_data=val_ds,
    validation_steps=validation_steps,
    class_weight=class_weights,
    callbacks=[reduce_lr, early_stop, checkpoint]
)
```

Isječak koda 20: Postupak finog podešavanja

6. Evaluacija treniranih modela

Evaluacija razvijenih modela provedena je korištenjem različitih metrika, uključujući točnost, preciznost, odziv i F1-mjeru. Korištena matrica konfuzije za detaljniji uvid u performanse modela, omogućujući identificiranje specifičnih područja gdje je model imao poteškoća. Na temelju ovih rezultata, izvršene su dodatne prilagodbe modela kako bi se postigla što veća točnost i pouzdanost.

Nakon treniranja modela, evaluacija se provodi kroz nekoliko koraka. Prvo, kreiraju se grafovi gubitka (loss) i točnosti (accuracy) za treniranje i validacijske skupove kako bi se prikazale performanse modela tijekom treniranja (isječak koda 21). Uz pomoć ovih grafova moguće je vidjeti da li se kod modela pojavljuje pretreniranosti ili podtreniranost, te se mogu raditi dodatna korekcije na sami model. Za izradu grafova koristila se biblioteka matplotlib kojom je vrlo jednostavno vizualizirati željene vrijednosti. Za svaki istrenirani model su se kreirali grafovi točnosti i gubitka gdje na x osi se nalazilo vrijeme odnosno epohe, a na y osi same vrijednosti validacijske točnosti odnosno gubitaka. U nastavku je prikazan način kreiranja grafova dok će se posebno po modelima prikazati rezultati.

```
import matplotlib.pyplot as plt
# Plot training & validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Isječak koda 21: Kreiranje grafova

Zatim se model evaluirao na testnom skupu podataka kako bi se izmjerila njegova točnost na neviđenim podacima, čime se moglo predočiti samo generaliziranje modela. Isječak koda 22.

```
test_loss, test_accuracy = model.evaluate(test_ds, steps=len(X_test) // batch_size)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

Isječak koda 22: Evaluacija modela na testnom skupu

Za još bolji pregled generalizacije koristila se i matrica konfuzije gdje se po svakoj klasi mogla prikazati klasificiranja na neviđene podatke, koje su mogle biti ispravne ili pogrešne. Pomoću matrice se može identificirati koje klase model najbolje prepoznaje a koje najmanje. Isječak koda 23.

```
conf_matrix = confusion_matrix(Y_test_int, Y_pred_classes)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
↪  xticklabels=class_dict.keys(), yticklabels=class_dict.keys())
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```

Isječak koda 23: Generiranje matrice konfuzije

Nakon toga se generirao klasifikacijski izvještaj gdje su se prikazane metrike poput preciznosti (precision), odzivu (recall), F1 rezultati i točnosti (accuracy) modela po klasama (isječak koda 24). Na kraju metrike na sveobuhvatne performanse modela.

```
class_report = classification_report(Y_test_int, Y_pred_classes,
↪  target_names=class_dict.keys())
print(class_report)

accuracy = accuracy_score(Y_test_int, Y_pred_classes)
precision = precision_score(Y_test_int, Y_pred_classes, average='weighted')
recall = recall_score(Y_test_int, Y_pred_classes, average='weighted')
f1 = f1_score(Y_test_int, Y_pred_classes, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

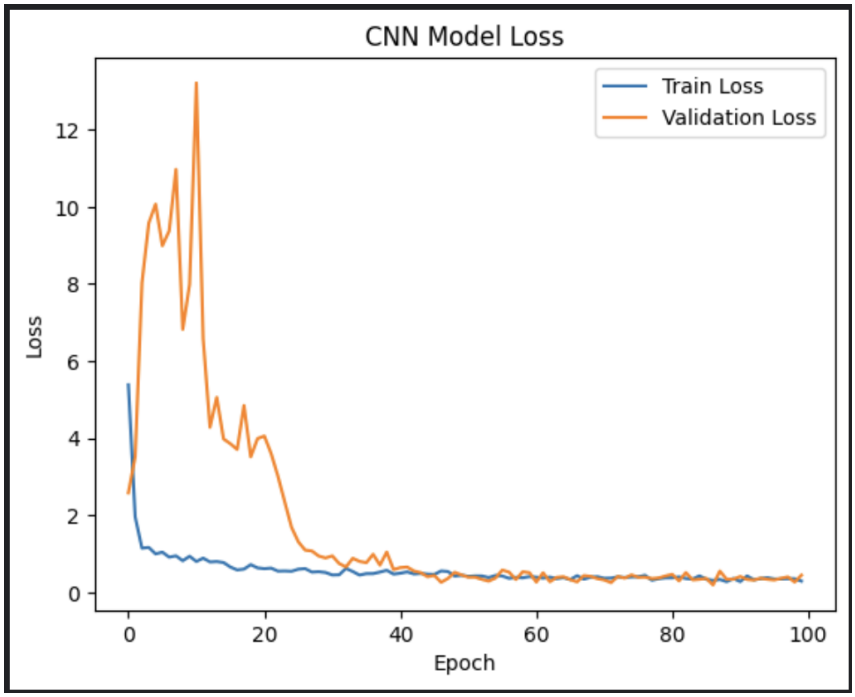
Isječak koda 24: Generiranje klasifikacijskog izvještaja

6.1. Konvolucijska neuronska mreža

Nakon završenog treniranja modela, provedena je detaljna evaluacija korištenjem različitih metrika. Glavni cilj evaluacije bio je provjeriti kako model generalizira na neviđenim podacima i identificirati potencijalne probleme u klasifikaciji specifičnih klasa.

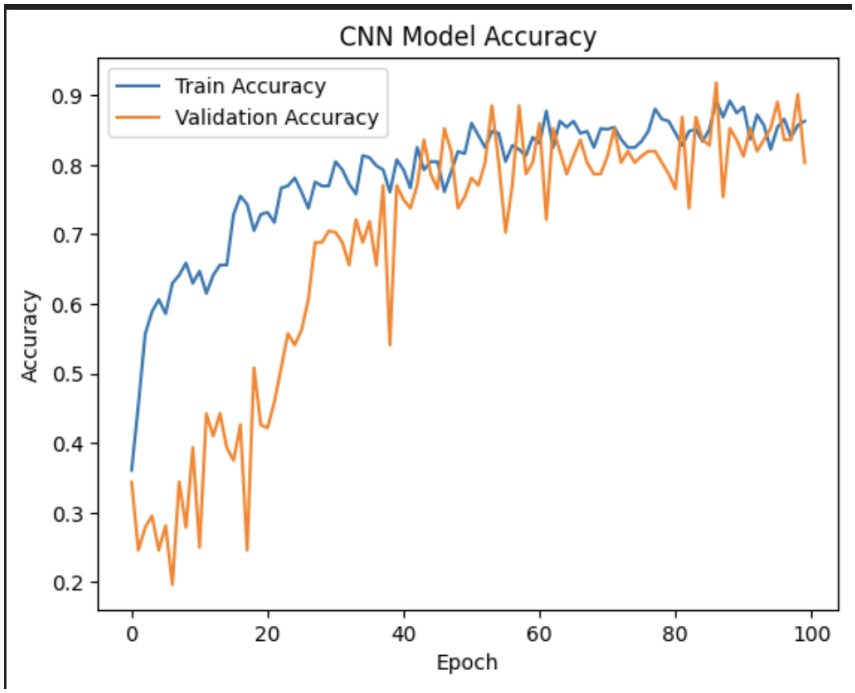
Graf prikazuje gubitak (loss) tijekom treniranja i validacije konvolucijske neuronske mreže kroz epohe. Na početku se događa nagli porast i oscilacije u gubitku na validacijskom skupu podataka, što je očekivano u početcima treniranja jer model pokušava optimizirati svoje težine.

Od oko 18. epohe dolazi do značajnog pad gubitaka, što znači da model počinje bolje učiti značajke iz podataka što znači smanjenje pogrešaka u predikciji. Nakon 35. epohe, graf gubitka se stabilizira, gdje je gubitak na setu treniranje i validacije skoro pa jednak. Stabilizacija gubitaka na niskim vrijednostima znači da je model naučio značajke te da su male šanse za predtreniranošću ili podtreniranošću modela. U teorijskom dijelu nalazi se primjer grafa slika 13 nad kojim je moguće vidjeti primjer pretreniranosti gdje krivulja na validacijskom skupu oscilira i ima poprilično visoki gubitak. Pri završetku treniranja model je bio vrlo blizu optimalne točke i to na oba skupa podataka što prikazuje naredna slika 21.



Slika 21: Graf gubitaka

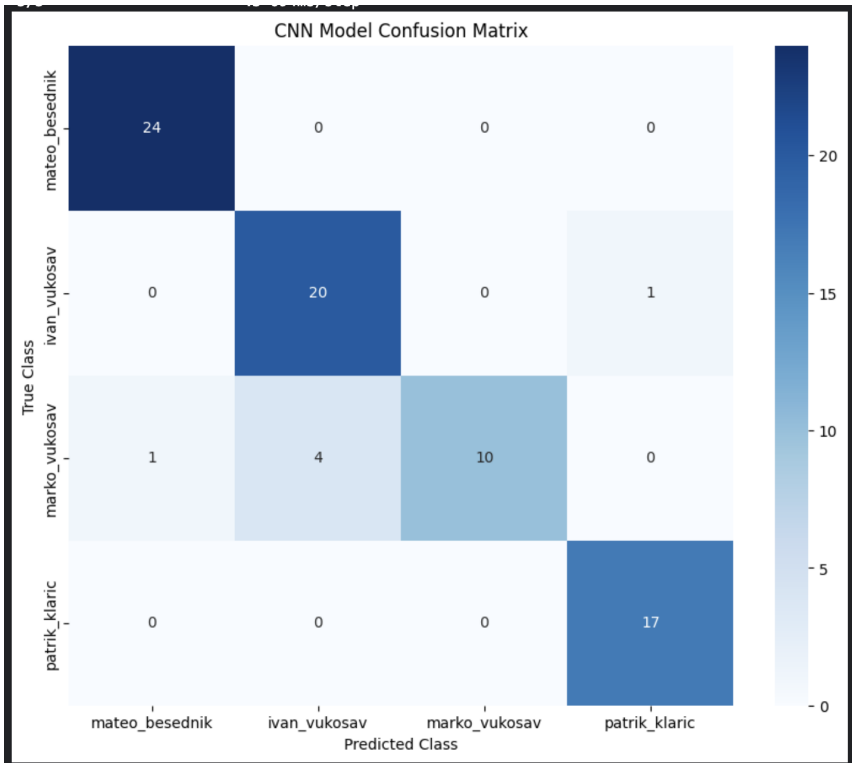
Sljedeći graf slika 22 prikazuje točnosti tijekom treniranja i validacije kroz epohe. Na početku se na validaciji događa nagli pad od 35% točnosti na 25% te nastavlja padati do otprilike 8 epohe gdje se onda događa porast i oscilacije. Kao i kod gubitaka, na početku je ovakvo događanje očekivano, jer se model uči prepoznavati obrasce u podacima. Ove oscilacije tijekom treniranja se mogu događati kad se trenira na manjem skupu podataka, što može govorit da model ima poteškoća s generaliziranjem novih podataka. Kako epohe prolaze, odnosno model se trenira, tako se može vidjeti postepeno povećanje i stabilizacija gdje na kraju model doseže validacijsku točnost preko 85%. Točnost kod treniranja od početka je u kontantnom porastu bez naglih oscilacija, te otprilike nakon 40. epohe validacijska točnost se stapa s treniranjem. Pri završetku treniranja rezultati pokazuju da model ima dobre performanse s visokim razinama točnosti, ali bi naknadno fino podešavanje dodatno smanjilo oscilacije i poboljšalo generalizaciju.



Slika 22: Graf točnosti

Matrica konfuzije prikazuje performanse modela na način da prikazuje ispravne i pogrešne predikcije po klasama. Na matrici je vidljivo koliko je model bio uspješan u predviđanju svake klase. Dijagonalni elementi (tamno plavi kvadrati) predstavljaju ispravne predikcije za svaku klasu, dok ostali elementi pokazuju koliko je puta model pogrešno predvidio klasu.

Na primjer, model je 24 puta točno predvidio klasu `mateo_besednik` i 20 puta klasu `ivan_vukosav`, što govori da model dobro prepoznaje te dvije klase. Za klasu `marko_vukosav`, model je imao poteškoća s točnim predikcijama, jer je model nekoliko puta napravio pogrešnu klasifikaciju te ih je klasificirao pod klasama `ivan_vukosav` i `patrik_klaric`. Sve u svemu, matrica konfuzije je prikazala solidan rezultat.



Slika 23: Matrica konfuzije

Klasifikacijski izvještaj prikazuje detaljne performanse modela po klasama. Rezultati su sljedeći:

Klasa `mateo_besednik` ima visoku preciznost, odziv i F1-mjeru, što znači da model ima vrlo dobre performanse za tu klasu.

Klasa `ivan_vukosav` pokazuje nešto nižu preciznost (0.83) i F1-mjeru (0.89), što govori da model ima malo poteškoća s točnim predikcijama za tu klasu.

Klasa `marko_vukosav` ima 100% preciznost, ali niži odziv (0.67) i F1-mjeru (0.80), što znači da model ima poteškoća u pronalaženju svih pozitivnih primjera za ovu klasu, iako kada ih pronađe, predikcije su točne što je bilo i vidljivo na matrici konfuzije.

Klasa `patrik_klaric` ima jako dobre rezultate u svim metrikama, što govori da model ima visoku pouzdanost za tu klasu.

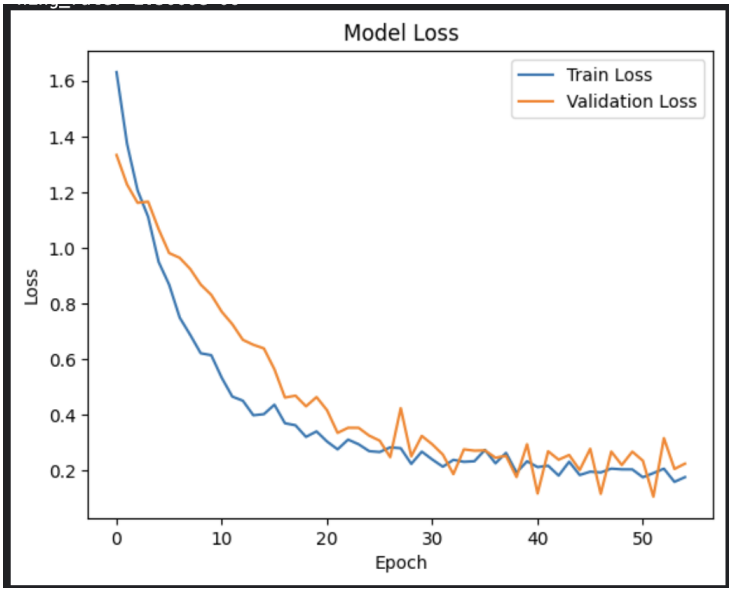
```
                precision    recall  f1-score   support

mateo_besednik       0.96      1.00      0.98        24
  ivan_vukosav       0.83      0.95      0.89        21
 marko_vukosav       1.00      0.67      0.80        15
 patrik_klaric       0.94      1.00      0.97        17

      accuracy                           0.92        77
     macro avg       0.93      0.90      0.91        77
  weighted avg       0.93      0.92      0.92        77

Accuracy: 0.92
Precision: 0.93
Recall: 0.92
F1 Score: 0.92
```

Slika 24: Metrike

Model CNN pokazuje dobre rezultate. Međutim, kod pojedinih klasa postoji mjesta za poboljšanje kako bi se poboljšala generalizacija modela i smanjile krive klasifikacije. Ukupna točnost modela je visoka 93% , što govori da je model uspješno naučio generalizirati većinu podataka iz skupa.

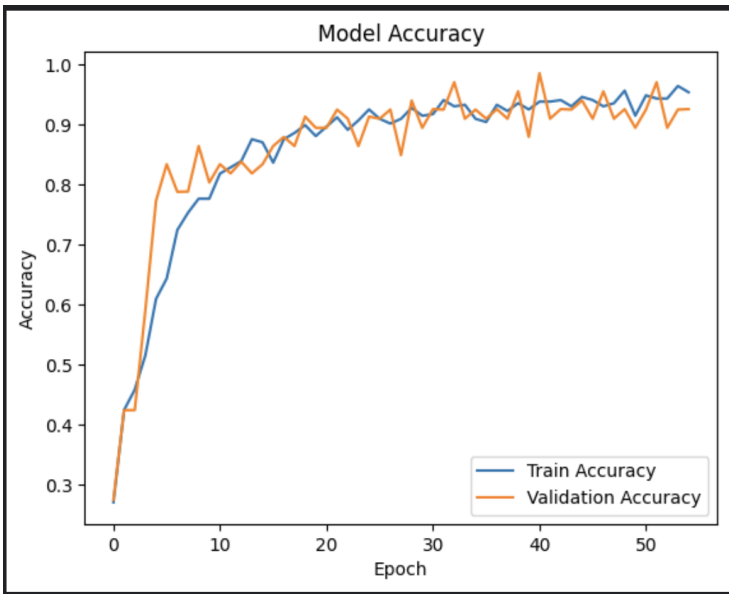
6.2. VGG 16

Graf za VGG16 model prikazuje gubitke (loss) za treniranje i validaciju kroz epohe. Od početka je vidljivo da gubitci opadaju kako epohe napreduju, što ukazuje na to da model od početka prepoznaje uzorke i značajke u podacima. Nakon otprilike 30. epohe, gubici na treniranju i validaciji počinju se stabilizirati, pri čemu su skoro jednaki. Sličan razvoj je i kod CNN modela, gdje su gubici na kraju treniranja stabilni i na niskim vrijednostima.



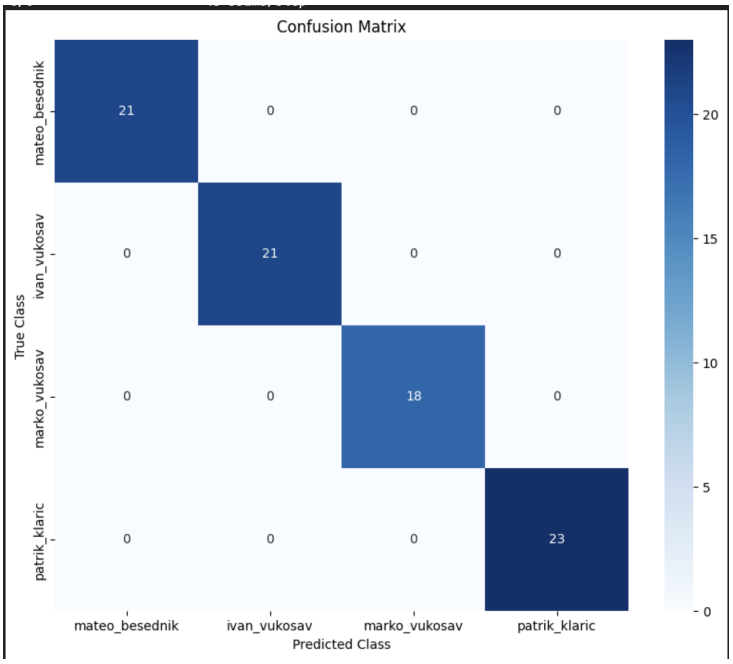
Slika 25: Graf gubitaka

Sljedeći graf prikazuje točnost modela VGG16 tijekom treniranja i validacije kroz epohe. Na početku su prisutne oscilacije između 5 i 10 epohe, no kako epohe napreduju, točnost se stabilizira. Na kraju treniranja točnost na treniranju i validaciji doseže visoke razine, oko 90%-95%, što ukazuje na dobru generalizaciju modela, slično kao i u modelu CNN osim što je ovdje točnost s manje oscilacija.



Slika 26: Graf točnosti

Matrica konfuzije za model VGG16 pokazuje da model vrlo dobro prepoznaje sve klase. Dijagonalni elementi pokazuju točne predikcije za sve klase odnosno model je ispravno klasificirao svih 21 uzorak za klasu `mateo_besednik`, svih 21 za klasu `ivan_vukosav`, 18 za klasu `marko_vukosav`, i svih 23 za klasu `patrik_klaric`. Ova matrica konfuzije ukazuje na to da je model VGG16 precizno treniran i da ispravno klasificirati sve klase.



Slika 27: Matrica konfuzije

Klasifikacijski izvještaj za VGG16 model prikazuje odlične performanse za sve klase. Odnosno, model je uspješno naučio generalizirati klase i točno ih predviđati.

	precision	recall	f1-score	support
mateo_besednik	1.00	1.00	1.00	21
ivan_vukosav	1.00	1.00	1.00	21
marko_vukosav	1.00	1.00	1.00	18
patrik_klaric	1.00	1.00	1.00	23
accuracy			1.00	83
macro avg	1.00	1.00	1.00	83
weighted avg	1.00	1.00	1.00	83

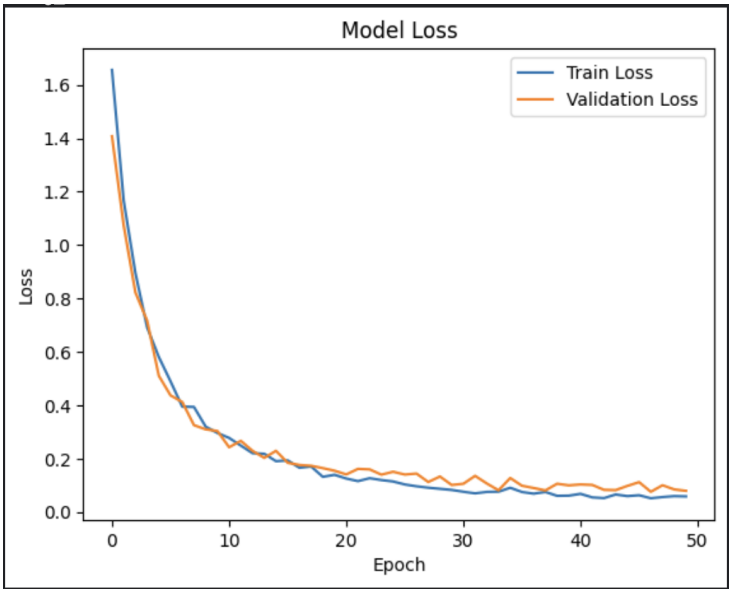
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00

Slika 28: Klasifikacijski izvještaj i metrike

Model VGG16 je rezultatima nakon treniranja prikazao gotovo savršene rezultate. Sve metrike su prikazale 100%. Grafovi gubitaka i točnosti su prikazali odlične rezultate kao i matrica konfuzije koja je sve klase pravilno klasificirala. Performanse VGG16 modela su jako dobre, te prema podatcim boljim od prethodnog CNN modela što je i očekivano s obzirom da je VGG16 model predtrenirani model na milijune različitih podataka.

6.3. MobileNetV2

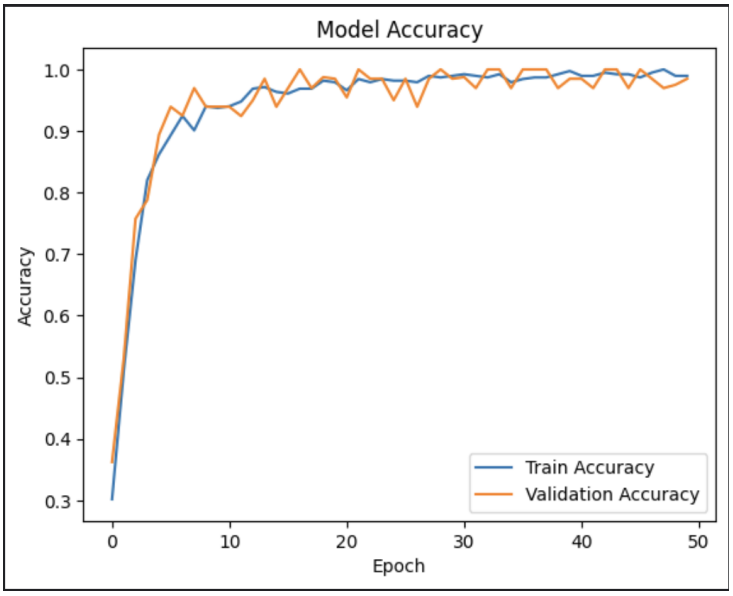
Graf za MobileNetV2 model prikazuje gubitke (loss) za treniranje i validaciju kroz epohe. Kao i VGG16 Od početka je vidljivo da gubitci opadaju kako epohe napreduju. Gubici na treniranju i validaciji od početka su skoro identične vrijednosti te nakon 30. epohe se počinju stabilizirati i do kraja kreću prema niskim vrijednostima.



Slika 29: Graf gubitaka

Sljedeći graf prikazuje točnost modela MobileNetV2 tijekom treniranja i validacije kroz

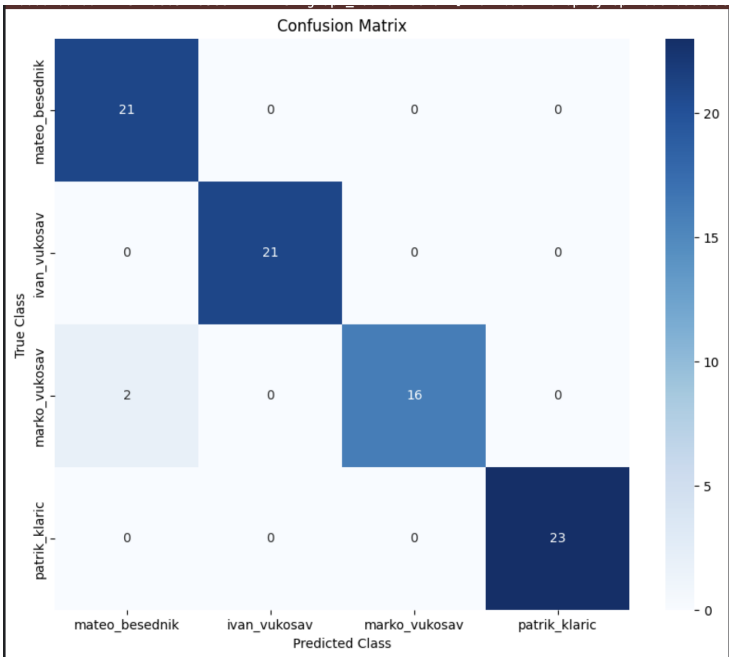
epohe. Kao i u prethodnom grafu, krivulje se konstantno prate prema vrlo visokoj točnosti preko 96%. Već na otprilike 15. epohi točnost se stabilizirala te se nastavila kretati u rangu 93%-98% s malim oscilacijama.



Slika 30: Graf točnosti

Matrica konfuzije za model MobileNetV2 pokazuje da model vrlo dobro prepoznaje sve klase. Dijagonalni elementi pokazuju točne predikcije za sve klase odnosno model je ispravno klasificirao svih 21 uzorak za klasu `mateo_besednik` , svih 21 za klasu `ivan_vukosav`, 18 za klasu `marko_vukosav`, i svih 23 za klasu `patrik_klaric`. Ova matrica konfuzije ukazuje na to da je model VGG16 precizno treniran i da ispravno klasificirati sve klase.

Matrica konfuzije za model MobileNetV2 pokazuje kako je model relativno dobro klasificirao većinu klasa. Na primjer, klasa `mateo_besednik` točno je klasificirana 21 put, dok je klasa `ivan_vukosav` također točno klasificirana 21 put. Klasa `marko_vukosav` ima nekoliko pogrešnih klasifikacija, s dva primjera koja su krivo klasificirana kao `ivan_vukosav`. Pogrešna klasifikacija ukazuje na to da model ima malih poteškoća u razlikovanju tih klasa.



Slika 31: Matrica konfuzije

Klasifikacijski izvještaj za MobileNetV2 model prikazuje dobre performanse za većinu klasa. Model prepoznaje dobro većinu klasa i rezultati su vrlo zadovoljavajući.

	precision	recall	f1-score	support
mateo_besednik	0.91	1.00	0.95	21
ivan_vukosav	1.00	1.00	1.00	21
marko_vukosav	1.00	0.89	0.94	18
patrik_klaric	1.00	1.00	1.00	23
accuracy			0.98	83
macro avg	0.98	0.97	0.97	83
weighted avg	0.98	0.98	0.98	83

Accuracy: 0.98
Precision: 0.98
Recall: 0.98
F1 Score: 0.98

Slika 32: Klasifikacijski izvještaj i metrike

Model MobileNetV2 je, nakon treniranja, pokazao vrlo dobre rezultate. Ukupna točnost modela je 98%, a ostale metrike preciznost, odziv i F1 mjera također pokazuju dobre vrijednosti, što govori da je model u stanju vrlo dobro klasificirati većinu podataka. Grafovi gubitaka i točnosti potvrđuju da je model dobro treniran, s vrlo stabilnim performansama kroz epohe. Matrica konfuzije pokazuje da model točno klasificira većinu klasa, uz minimalne pogreške kod nekoliko klasa, što je znak vrlo dobre generalizacije. Iako rezultati nisu potpuno savršeni kao kod VGG16 modela, MobileNetV2 postiže gotovo jednako dobre performanse uz manji broj parametara i manju složenost.

7. Implementacija servisnog sloja

U ovom poglavlju je detaljno opisana implementaciju servisnog sloja potrebnog za realizaciju sustava za prepoznavanje i klasifikaciju slika. Sustav je osmišljen tako da omogućava korisnicima slanje fotografija putem mobilne aplikacije radi klasifikacije te kasnije pohrane i preuzimanje fotografija. Ovaj servisni sloj povezuje aplikaciju s bazom podataka, omogućava klasifikaciju slika, te osigurava pohranu fotografija na Google Disk radi daljnjeg finog podešavanja modela.

- **Baza podataka**: NoSQL baza podataka MongoDB koristi se za pohranu informacija o korisnicima aplikacije, kolegijima, te prijavama studenata na nastavi.
- **Klasifikacija slika**: Implementirane su funkcionalnosti za klasifikaciju slika koristeći tri različita modela: CNN, VGG16 i MobileNetV2.
- **Pohrana na Google Disk**: Fotografije koje odgovaraju klasifikaciji pohranjuju se na Google Disk putem Google API-ja.
- **API servisi**: Implementirane su CRUD operacije (Create, Read, Update, Delete) za korisnike, kolegije i prisutnost studenata, kao i servisi za fino podešavanje modela te razne operacije nad modelima.

U ovom radu korišten je niz alata i biblioteka kako bi se osigurala funkcionalnost sustava. Svaki od ovih alata ima specifičnu ulogu za implementaciju sustava:

- **Flask**: Web framework za implementaciju servisa.
- **Werkzeug**: Biblioteka za sigurnu autentifikaciju i rukovanje HTTP zahtjevima.
- **google-auth**, **google-auth-oauthlib**, **google-api-python-client**: Koriste se za autentifikaciju korisnika i rad s Google Diskom.
- **TensorFlow**: Za implementaciju i korištenje modela za klasifikaciju slika (CNN, VGG16, MobileNetV2).
- **scikit-learn**: Koristi se za analizu podataka i metričke izračune tijekom evaluacije modela.
- **numpy**: Za numeričke operacije na matricama i vektorima.
- **Flask-Cors**: Omogućava komunikaciju između različitih domena putem API poziva.
- **pymongo**: Za rad s MongoDB bazom podataka.
- **Pillow**: Za obradu slika prije pohrane i klasifikacije.

U nastavku se nalazi najbitniji dijelovi servisnog sloja sa detaljnim objašnjenjima svakog dijela.

7.1. Klasifikacija slika

U ovom poglavlju obrađuje se način na koji model klasificira određenu osobu. Naime, kada korisnik pošalje sliku zajedno za klasom za klasificiranje, tada model provjerava koliko ta slika odgovara unaprijed definiranim klasama. Proces počinje usporedbom značajki (engl. features) koje model izdvaja iz nove slike i značajki koje su već pohranjene za određenu klase.

Na početku učitavanja modela (isječak koda 25), iz njega se izvlače značajke za tada istrenirane klase kako bi se moglo s njima uspoređivati. Za svaku sliku po klasi se izdvajaju njene značajke (linija 6) i pohranjuju kao mapa vrijednosti ključ, vrijednost gdje je vrijednost ključa sama klasifikacija a vrijednosti su značajke svih slika za tu klasifikaciju (linija 8).

```
1 reference_features = {}
2 for class_name, class_path in reference_images_path.items():
3     features = []
4     for img_file in os.listdir(class_path):
5         img_path = os.path.join(class_path, img_file)
6         img_features = extract_features(model, img_path)
7         features.append(img_features)
8     reference_features[class_name] = np.vstack(features)
```

Isječak koda 25: Ekstrakcija značajki za svaku učitanu sliku

Kada korisnik pošalje sliku zajedno sa referencom, pokreće se postupak usporedbe značajki. Iz dobivene slike se izvlače značajke koristeći funkciju za izdvajanje značajki iz slika te izračunavanje sličnosti pomoću kosinusne sličnosti.

```
1  def extract_features(model, img):
2      if isinstance(img, Image.Image):
3          img = img.resize((224, 224)) if 'vgg16' in model_path else img.resize((128,
           ↪  128))
4      else:
5          if 'vgg16' in model_path:
6              img = load_img(img, target_size=(224, 224))
7          else:
8              img = load_img(img, target_size=(128, 128))
9      img_array = img_to_array(img) / 255.0
10     img_array = np.expand_dims(img_array, axis=0)
11     features = model.predict(img_array)
12     return features
```

Isječak koda 26: Funkcija za ekstrakciju značajki

U isječku koda 26 funkcija učitava sliku, prilagođava ju modelu VGG16 i MobileNetV2 su trenirani na veličini 224x224 dok CNN na veličini 128x128 te ju pretvara u niz piksela. Zatim se normalizira (svaka vrijednost piksela se postavlja između 0 i 1 (linija 9 i 10)), nakon čega model predviđa značajke slike (linija 11). Ove značajke su numerički prikaz slike koji omogu-

ćuje modelu da identificira obrasce koji pomažu u prepoznavanju klase slike. Za usporedbu značajki koristi se kosinusna sličnost (isječak koda 27), koja mjeri kut između dva vektora (vektora značajki) te vraća vrijednost sličnosti između 0 i 1. Što je veća vrijednost sličnosti, to je veća vjerojatnost da slika pripada odabranoj klasi.

```
input_features = extract_features(custom_cnn_model, img)
class_features = reference_features.get(class_name, None)
if class_features is None:
    return 0.0  # Return 0 similarity if the class is not found
similarity = cosine_similarity(input_features, class_features)
avg_similarity = np.mean(similarity)
return avg_similarity
```

Isječak koda 27: Funkcija za klasifikaciju

Nakon dobivanja vrijednosti sličnosti, ta vrijednost se uspoređuje sa željenom vrijednosti koja je po inicijalizaciji postavljena na 85%, te ju je moguće podesiti ukoliko se pokaže da model ne može preko te vrijednosti što je i moguće s obzirom da se treniralo na manjem skupu podataka.

Dakle, proces klasifikacije ukratko izgleda:

1. Ekstrakcija značajki: izdvajanje značajki iz slike koju je korisnik poslao. Isječak koda 26

2. Usporedba s referentnim značajkama: značajke ulazne slike uspoređuju se s pohranjenim značajkama svih slika unutar odabrane klase koristeći kosinusnu sličnost. Isječak koda 25

3. Izračun sličnosti: funkcija vraća prosječnu sličnost između ulazne slike i slika referentne klase. Isječak koda 27

Ako je sličnost dovoljno visoka, slika je klasificirana kao da pripada toj klasi.

7.2. Fino podešavanje modela i operacije nad modelima

Kako bi model mogao konstantno učiti i raditi sa novim klasifikacijama potrebno je model nadograditi, odnosno koristiti fino podešavanje. Taj proces radi na sličan način kao što se radio proces treniranja za predtrenirani model, dakle za korišteni model se izvuče bazni model bez zadnje pripadajućeg sloja, po mogućnosti se dodaje dodatni konvolucijski sloj te se poziva funkcija fit() u kojoj su postavljeni željeni parametri za fino podešavanje.

Postupak započinje učitavanjem originalnih i novih skupova podataka, kombiniranjem tih skupova te podjelom na trening, validacijske i testne skupove podataka (isječak koda 28). Koristi se ImageDataGenerator za generiranje slika s augmentacijom kako bi se povećala raznolikost trening skupa.

```
original_image_paths, original_labels, class_dict =
↪  load_all_datasets(original_dataset_path)
        new_image_paths, new_labels, class_dict = load_all_datasets(new_dataset_path,
        ↪  class_dict)
        combined_image_paths = np.concatenate((original_image_paths,
        ↪  new_image_paths), axis=0)
        combined_labels = np.concatenate((original_labels, new_labels),
        ↪  axis=0).astype(str)

        X_train, X_temp, Y_train, Y_temp = train_test_split(combined_image_paths,
        ↪  combined_labels, test_size=0.3, random_state=42,
        ↪  stratify=combined_labels)
        X_val, X_test, Y_val, Y_test = train_test_split(X_temp, Y_temp,
        ↪  test_size=0.5, random_state=42, stratify=Y_temp)

        train_datagen = ImageDataGenerator(
            rescale=1./255,
            rotation_range=20,
            width_shift_range=0.1,
            height_shift_range=0.1,
            shear_range=0.1,
            zoom_range=0.1,
            horizontal_flip=True,
            fill_mode='nearest'
        )
        val_test_datagen = ImageDataGenerator(rescale=1./255)

        train_generator = train_datagen.flow_from_dataframe(
            pd.DataFrame({'filename': X_train, 'class': Y_train}),
            x_col='filename',
            y_col='class',
            target_size=(224, 224),
            batch_size=16,
            class_mode='sparse'
        )

        validation_generator = val_test_datagen.flow_from_dataframe(
            pd.DataFrame({'filename': X_val, 'class': Y_val}),
            x_col='filename',
            y_col='class',
            target_size=(224, 224),
            batch_size=16,
            class_mode='sparse'
        )

        test_generator = val_test_datagen.flow_from_dataframe(
            pd.DataFrame({'filename': X_test, 'class': Y_test}),
            x_col='filename',
            y_col='class',
            target_size=(224, 224),
            batch_size=16,
            class_mode='sparse',
            shuffle=False
        )

        train_ds = generator_to_tfdata(train_generator).repeat()
        val_ds = generator_to_tfdata(validation_generator).repeat()
        test_ds = generator_to_tfdata(test_generator)
```

Zatim se uklanja posljednji sloj i dodaju dodatni slojevi, u ovom slučaju identični koji su se koristili za treniranje VGG16 modela, te kreira novi model. Isječak koda 29.

```
x = custom_model.layers[-6].output
  x = GlobalAveragePooling2D()(x)
  x = Dense(512, activation='relu')(x)
  x = Dropout(0.5)(x)
  x = BatchNormalization()(x)
  predictions = Dense(len(class_dict), activation='softmax')(x)

  model = Model(inputs=custom_model.input, outputs=predictions)
```

Isječak koda 29: Generiranje slojeva za VGG16 model

Zatim se baza modela je zamrzava (isječak koda 30, linija 1), dok se novi slojevi prilagođavaju klasifikaciji novih podataka. Model se ponovno kompajlira i trenira (linije 4-16).

```
for layer in custom_cnn_model.layers[:-6]:
        layer.trainable = False

    # Compile the model with a lower learning rate
    model.compile(optimizer=Adam(learning_rate=1e-3),
    ↪  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    # Define callbacks
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
    ↪  min_lr=1e-4)
    early_stop = EarlyStopping(monitor='val_loss', patience=10,
    ↪  restore_best_weights=True)
    checkpoint = ModelCheckpoint('fine_tuned_vgg16_model.keras', monitor='val_loss',
    ↪  save_best_only=True, mode='min')

    model.fit(
        train_ds,
        steps_per_epoch=len(train_generator),
        epochs=50,
        validation_data=val_ds,
        validation_steps=len(validation_generator),
        callbacks=[reduce_lr, early_stop, checkpoint]
    )
```

Isječak koda 30: Definiranje svih parametara za fino podešavanje na modelu VGG16

Nakon treniranja, model se evaluira na testnom skupu podataka te ukoliko točnost zadovoljava postavljani prag u ovom slučaju to je isti prag koji se koristi za klasifikaciju tada se model sprema na Google Disk i postavlja kao novi model za buduće klasifikacije. Novi model se sprema kao trenutno vrijeme sa rezultatom točnosti i naziv modela. Isječak koda 31.

```
test_loss, test_accuracy = model.evaluate(test_ds, steps=len(test_generator))
print(f'Test Accuracy: {test_accuracy:.2f}')
logger.info(f'Test Accuracy: {test_accuracy:.2f}')

current_time = datetime.now().strftime("%Y%m%d_%H%M%S")
new_model_path =
↪  f'modeli/fine_tuned/fine_tuned_vgg16_model_mv_{current_time}_{test_accuracy:.2f}.h5'
model.save(new_model_path)
threshold = get_threshold()

if test_accuracy >= threshold:
    print(f"Test Accuracy {test_accuracy} meets or exceeds the threshold
    ↪  {threshold}. Proceeding to save model.")
    logger.info(f"Model meets or exceeds the threshold {threshold}. Proceeding to
    ↪  save model.")
    load_reference_features()
    upload_model_to_gdrive(new_model_path, f'modeli/fine_tuned/{new_model_path}')
    model_path = new_model_path
    custom_cnn_model = load_model(new_model_path)
else:
    logger.info(f"Test Accuracy {test_accuracy} does not meet the threshold
    ↪  {threshold}. Model not saved.")
    print(f"Test Accuracy {test_accuracy} does not meet the threshold {threshold}.
    ↪  Model not saved.")
```

Isječak koda 31: Evaluacija testnih podataka

Iz razloga što ova radnja zahtijeva resurse ona se vrši u pozadini kako ne bi blokirala ostale radnje koje su potrebne, jer je za istu napravljena krajnja točka kako bi se moglo pozivati preko mobilne aplikacije.

7.3. Baze podataka i API servisi

U svrhu pohrane slika i modela, te manipulacija nad njima korišten je Google Drive servis sa bazom podataka MongoDb čime je omogućen rad s podacima koji su potrebni za sami sustav, odnosno u ovom slučaju za mobilnu aplikaciju.

7.3.1. Google Drive API za pohranu i dohvat podataka

Google Drive API omogućuje pohranu i dohvaćanje modela i slika koje se koriste za klasifikaciju i treniranje modela. Prilikom dohvaćanja ili pohrane podataka potrebno je odraditi autentifikaciju za Google računom preko Google OAuth2 sustava, koji osigurava pristup koris-

ničkim datotekama. Nakon uspješne autentifikacije, pomoću Google Drive API servisa moguće je preuzimati i pohranjivati datoteke na Google Driveu.

Autentifikacija korisnika se radi preko Google OAuth2, gdje se koristi klijentova tajna datoteka i token. Jednom kada se korisnik prijavi s tokenom za osvježavanje moguće je konstantno osvježavati token koji ističe svaki sat i pomoću tog tokena koristiti servis. Sljedeći isječak koda 32 to prikazuje.

```
def authenticate():
    global creds
    if os.path.exists(TOKEN_FILE):
        creds = Credentials.from_authorized_user_file(TOKEN_FILE, SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
            with open(TOKEN_FILE, 'w') as token:
                token.write(creds.to_json())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(CLIENT_SECRETS_FILE,
            ↪  SCOPES)
            creds = flow.run_local_server(port=8080, host='127.0.0.1')
            with open(TOKEN_FILE, 'w') as token:
                token.write(creds.to_json())
    service = build(API_SERVICE_NAME, API_VERSION, credentials=creds)
    return service
```

Isječak koda 32: Funckija za autentifikaciju korisnika

Ista autentifikacija se koristi kako bi se podatci mogli lokalno preuzeti. Skup slika se preuzima kao zip datoteka koja se onda otpakira lokalno, dok se modeli preuzimaju tek nakon što budu odabrani, a do tada se u aplikaciji prikazuju samo njihove informacije poput naziva i identifikacije. Kod pokretanja servisa inicijalno je postavljeno koji model će se preuzeti i učitati kako bi se mogla izvršavati predikacija ili fino podešavanje.

Sljedeća funkcija pokazuje proces preuzimanja i otpakiranje slika iz kojih se onda izvlače značajke pri učitavanju modela kako bi se mogle koristiti pri klasifikaciji. Isječak koda 33.

```
def download_and_extract_dataset(file_id, destination_path):
    service = authenticate()
    request = service.files().get_media(fileId=file_id)
    file = io.BytesIO()
    downloader = MediaIoBaseDownload(file, request)
    done = False
    while done is False:
        status, done = downloader.next_chunk()
        print("Download %d%%." % int(status.progress() * 100))
    file.seek(0)
    with open(destination_path, 'wb') as f:
        f.write(file.read())
    with zipfile.ZipFile(destination_path, 'r') as zip_ref:
        zip_ref.extractall(os.path.dirname(destination_path))
```

Isječak koda 33: Funkcija za preuzimanje i ekstrakciju podataka

Za preuzimanje modela koristi se funkcija downloadModel koja preko autentifikacije dohvaća pristup prema Google Drive-u i preuzima željeni model u dijelovima (isječak koda 34). Kada je preuzimanje završeno, model se lokalno sprema. Pri završetku preuzimanja poziva se funkcija laodModel iz TensorFlow-a koja onda učitava model. Po završetku učitavanja modela poziva se ista funkcija za izvlačenje značajki.

```
def download_model(file_id, destination_path):
    service = authenticate()
    request = service.files().get_media(fileId=file_id)
    file = io.BytesIO()
    downloader = MediaIoBaseDownload(file, request)
    done = False
    while done is False:
        status, done = downloader.next_chunk()
        print("Download %d%%." % int(status.progress() * 100))
    file.seek(0)
    with open(destination_path, 'wb') as f:
        f.write(file.read())

def load_model_from_gdrive(file_id, model_path):
    download_model(file_id, model_path)
    return load_model(model_path)
```

Isječak koda 34: Funkcija za preuzimanje i čitanje modela sa Google Diska

7.3.2. MongoDB

Za pohranu korisnika, kolegija, te praćenje prijava studenata na nastavi koristi se MongoDB, nerelacijska baza podataka koja radi s nestrukturiranim podacima. Baza je poprilično jednostavna sastoji se od 3 entiteta: korisnik, kolegij i prisustvo.

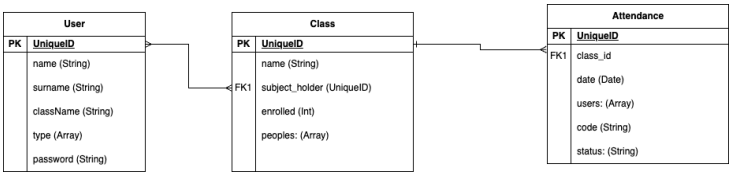
Korisnik (User) se sastoji od osnovnih informacija poput id, ime, prezime, klasifikacijsko

ime (kombinacija ime i prezime), zatim tip osobe koja označava ulogu korisnika u aplikaciji te lozinke koja se sprema kao kriptirana sha256 vrijednost.

Kolegij (Class) se sastoji od id-a, imena, id-a osobe koja je nositelj kolegija, broj osoba koja je upisala kolegij, te lista id-a osobi koja je upisana na kolegij.

Prisustvo (Attendance) se sastoji od id, id kolegija za koji se radi prisutstvo, datum, lista korisnika koje se prijavilo kao prisutno, kod za prijavu prisustva te status prisustva koji označava da li je prijava u tijeku ili ne odnosno da li je prijava otvorena ili zatvorena.

Na slici 33 je prikazan UML dijagram klasa koji prikazuje kako za jedan kolegij može biti više nositelja, dok jedna osoba može biti nositelj na više predmeta, te prisustvo može biti samo za jedan kolegij dok jedan kolegij može imati više prisustva.



Slika 33: UML dijagram klasa

Za svaku klasu kreirane su CRUD operacije, odnosno kreiranje, čitanje, ažuriranje i brisanje podataka. U nastavku je prikazana najzanimljivija operacija a to je kreiranje korisnika. Svaki korisnik se sastoji od osnovnih informacija te dodatno tipa i klasifikacijskog imena koje se koristi kod klasificiranja. Tip korisnika može biti student, profesor te admin. Admin tip znači da korisnik može birati trenutno aktivni model, pokretati finu doradu modela, kao i mijenjati prag točnosti modela. Profesor ima ovlasti pregleda prisustva dok student se može prijaviti na prisustvo. Samo osoba koja je nositelj predmeta ima pravo na pokretanje prisustva. Klasifikacijsko ime mora biti jedinstveno za svaku osobu ali mora se sastojati od njegovo imena i prezimena, u slučaju osoba koje imaju isto ime i prezime tada se za sljedeću osobu dodaje redni broj u sklopu naziva. Kod postavljanje lozinke ona se po kreiranju kriptira metodom pbkdf2:sha256 koja je vrlo sigurna. U isječku koda 35. je prikazana sama implementacija i rezultat u MongoDB.

```python
@faculty_management_bp.route('/add_user', methods=['POST'])
VALID_USER_TYPES = {"ADMIN", "PROFESSOR", "STUDENT"}

def generate_unique_classification(name, surname):
    base_classification = f"{name}_{surname}".lower()
    classification = base_classification
    counter = 1
    while db.users.find_one({"classification": classification}):
        counter += 1
        classification = f"{base_classification}{counter}"
    return classification

def add_user():
    data = request.get_json()
    name = data.get('name')
    surname = data.get('surname')
    user_type = data.get('type', [])
    password = data.get('password')

    if not name or not surname or not user_type or not password:
        return jsonify({"error": "Name, surname, type, and password are required"}),
        ↪  400

    if not all(utype in VALID_USER_TYPES for utype in user_type):
        return jsonify({"error": f"Invalid user type. Valid types are {',
        ↪  '.join(VALID_USER_TYPES)}"}), 400

    if len(user_type) != len(set(user_type)):
        return jsonify({"error": "Duplicate user types are not allowed"}), 400

    classification = generate_unique_classification(name, surname)

    user_data = {
        "name": name,
        "surname": surname,
        "classification": classification,
        "type": list(set(user_type)),  # Ensure unique types
        "password": generate_password_hash(password, method='pbkdf2:sha256')
    }

    result = db.users.insert_one(user_data)
    user_id = str(result.inserted_id)

    return jsonify({
        "message": "User added successfully",
        "user": {
            "_id": user_id,
            "name": name,
            "surname": surname,
            "classification": classification,
            "type": list(set(user_type))
        }
    }), 201
```

Isječak koda 35: Funkcija za dodavanje novog korisnika

Slika 34: Rezultat dodavanja korisnika

7.3.3. Servis za API integraciju

Za implementaciju API servisa koristi se Flask, Python web framework koji omogućuje jednostavno i brzo kreiranje API servisa. Flask podržava modularnost sa Blueprintova, s kojima je moguće kreirati logičku podjelu aplikacije na manje dijelove, u ovom slučaju se podijelilo po funkcionalnosti poput upravljanja korisnicima, treniranja modela ili upravljanja datotekama, u zasebne module. Moduli su kreirani po funkcionalnosti kako bi se ista podijelila te je u tu svrhu kreiran posebni modul za fino podešavanje u ovisnosti o modelu. Zatim modul za upravljanjem prisustvom, moduli za upravljanje funkcionalnostima za korisniea i kolegije, te modul za upravljanje modelima. Većinu modula se već objasnilo kroz prethodna poglavlja te u ovom poglavlju najviše pažnje posvetiti modulu za upravljanje prisustvom.

Modul je odgovoran za upravljanje procesima vezanim uz evidenciju prisutnosti studenata na nastavi. Ovaj modul pruža API rute koje omogućuju profesorima i studentima interakciju s evidencijom prisutnosti kao što su pokretanje i zaustavljanje evidencije, označavanje prisutnosti te pregled statusa prisutnosti.

Ruta za pokretanje evidencije prisutnosti omogućava korisniku (koja je nositelj predmeta) pokretanje nove evidencije prisutnosti za određeni kolegij (isječak koda 36). Kada profesor pošalje zahtjev, sustav generira nasumični kod (linija 1) koji će studenti koristiti za prijavu prisutnosti ukoliko tijekom klasifikacije model ne može prepoznati. Ovaj kod se zatim povezuje s trenutnim kolegijem i evidencijom prisutnosti te se bilježi u bazi podataka (linija 23). Kod je jedinstven za svaku evidenciju.

```
1  def generate_code(length=6):
2      return ''.join(random.choices(string.ascii_uppercase + string.digits, k=length))

3  @attendance_management_bp.route('/start_attendance', methods=['POST'])
4  def start_attendance():
5      data = request.get_json()
6      class_id = data.get('class_id')
7      user_id = data.get('user_id')

8      if not user_id:
9          return jsonify({"error": "Unauthorized"}), 401

10     class_data = db.classes.find_one({"_id": ObjectId(class_id)})
11     if not class_data:
12         return jsonify({"error": "Class not found"}), 404

13     if class_data['subject_holder'] != user_id:
14         return jsonify({"error": "Only the subject holder can start the
           ↪  attendance"}), 403

15     code = generate_code()
16     attendance_check = {
17         "class_id": class_id,
18         "date": datetime.now(),
19         "users": [],
20         "code": code,
21         "status": "open"
22     }

23     db.attendance.insert_one(attendance_check)
24     return jsonify({"code": code}), 200
```

Isječak koda 36: Funkcija za pokretanje prijave prisustva i generiranje koda

Ruta za zaustavljanje evidencije omogućuje profesoru zatvaranje otvorene evidencije prisutnosti (isječak koda 37). Nakon što kornsik želi zatvoriti prijavu, može poslati zahtjev kojim zatvara prijavu za taj dan. Sustav automatski ažurira status evidencije iz "open" u "closed" (linija 10), čime se onemogućuje daljnje prijavljivanje na tu nastavu.

```python
@attendance_management_bp.route('/stop_attendance', methods=['POST'])
def stop_attendance():
    data = request.get_json()
    class_id = data.get('class_id')
    user_id = data.get('user_id')

    if not user_id:
        return jsonify({"error": "Unauthorized"}), 401

    today_start = datetime.combine(datetime.today(), datetime.min.time())
    today_end = today_start + timedelta(days=1)

    result = db.attendance.update_many(
        {"class_id": class_id, "status": "open", "date": {"$gte": today_start,
        ↪  "$lt": today_end}},
        {"$set": {"status": "closed"}}
    )

    if result.matched_count == 0:
        return jsonify({"error": "No open attendance to close"}), 400

    return jsonify({"message": "Attendance closed"}), 200
```

Isječak koda 37: Funkcija za zaustavljanje prijave prisustva

Ruta za označavanje prisutnosti je glavna funkcionalnost za prijavu prisutnosti studenata. Studenti mogu prijaviti svoju prisutnost na dva načina:

Korištenjem koda (isječak koda 38): Studenti nakon što ne uspiju s prepoznavanjem lica kao rezervnu opciju se mogu prijaviti unosom jedinstveni kod koji im je profesor podijelio na početku nastave. Uz kod se šalje slika koju je student prethodno slikao te se ista pohranjuje na Google Disk (linija 7).

```
if code == attendance_check.get('code'):
    img = Image.open(io.BytesIO(image.read()))
    save_dir = os.path.join(new_dataset_path, classification)
    os.makedirs(save_dir, exist_ok=True)
    save_path = os.path.join(save_dir, image.filename)

    # Run saving and uploading in the background
    Thread(target=save_and_upload_image, args=(img, save_path, image.filename,
    ↪  file_id)).start()

    attendance_record["method"] = "code"
    attendance_record["similarity_score"] = similarity_score_pre
    db.attendance.update_one(
        {"_id": attendance_check['_id']},
        {"$push": {"users": attendance_record}}
    )
    return jsonify({"message": "Attendance marked with code. Image is being
    ↪  saved."}), 200
else:
    return jsonify({"error": "Invalid code"}), 200
```

Isječak koda 38: Za provjeru prisustva preko koda

Prepoznavanjem lica (isječak koda 39): Ako student pošalje sliku, sustav koristi prethodno trenirani model za prepoznavanje lica i uspoređuje sliku s referentnim slikama (linija 3). Ako je postotak sličnosti veći od zadanog praga, prisutnost se bilježi automatski (linije 5-15).

```
img = Image.open(io.BytesIO(image.read()))
  clas_img = img.resize((128, 128), Image.LANCZOS)
  similarity_score = compare_with_reference(img, classification)
  threshold = get_threshold()
  if similarity_score > threshold:
      save_dir = os.path.join(new_dataset_path, classification)
      os.makedirs(save_dir, exist_ok=True)
      save_path = os.path.join(save_dir, image.filename)

      Thread(target=save_and_upload_image, args=(img, save_path, image.filename,
      ↪  file_id)).start()

      attendance_record["method"] = "face_recognition"
      attendance_record["similarity_score"] = float(similarity_score)
      db.attendance.update_one(
          {"_id": attendance_check['_id']},
          {"$push": {"users": attendance_record}}
      )
      return jsonify({
          "message": "Attendance marked with face recognition. Image is being
          ↪  saved.",
          "similarity_score": float(similarity_score),
      }), 200
  else:
      return jsonify({
          "error": f"Face recognition failed, try again or use code. Similarity
          ↪  score ({similarity_score:.2f}) did not meet the threshold
          ↪  ({threshold:.2f}). Try again or with code",
          "similarity_score": float(similarity_score)
      }), 200
```

Isječak koda 39: Za provjeru prisustva preko slike lica

Prije same prijave prisustva vrše se razne provjere poput pripada li student tom kolegiju, postoji li otvorena prijava prisustva te da li je student već obavio prijavu prisutnosti. Nadalje postoje još dvije rute, a to je pregled evidencija po kolegiju i korisniku, te status prisustva koja vraća postoji li aktivna evidencija prisustva za taj dan.

8. Implementacija mobilne aplikacije

Cilj ovog rada je izraditi sustav za prepoznavanje lica u svrhu automatizirane prijave prisutnosti studenata na nastavi. Kako bi se taj cilj ostvario, razvijena je mobilna aplikacija koja studentima omogućuje jednostavnu i brzu prijavu putem prepoznavanja lica nazvana Attendio. Ova aplikacija izrađena je za Android platformu korištenjem programskog jezika Kotlin, dok se za razvoj korisničkog sučelja koristio Jetpack Compose.

Slika 35: Mobilna aplikacija

8.1. Tehnologije

Kao osnovu za spajanje mobilne aplikacije s backendom koristila se biblioteka Retrofit, koja omogućava komunikaciju s REST API-jem. Retrofit se koristi za slanje HTTP zahtjeva, kao što su prijava korisnika, slanje slika za prepoznavanje lica, te dohvaćanje podataka o predmetima i prisutnosti studenata.

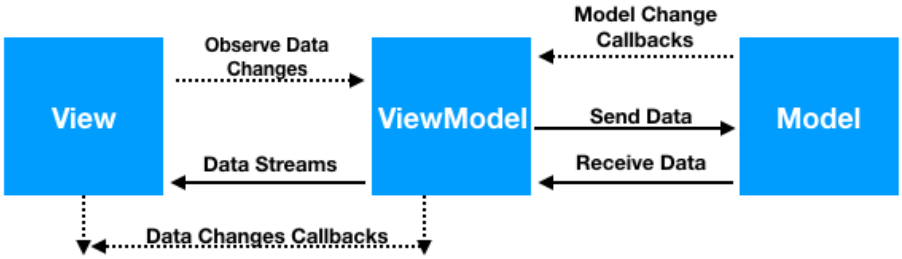
Za detekciju lica u realnom vremenu korištena je biblioteka od Google-a, ML Kit, koja omogućava prepoznavanje lica na slikama. ML Kit koristi algoritme strojnog učenja kako bi mogao prepoznati lica i pozicije u stvarnom vremenu.

Uz ML Kit, za rad s kamerom koristila se biblioteka CameraX, koja omogućuje analizu slika u stvarnom vremenu, te u kombinaciji sa ML kit slike s kamere se automatski šalju na analizu, gdje ih ML Kit analizira i detektira lica.

Na primjer, svaki put kad se kamera aktivira, CameraX šalje slike u realnom vremenu na ML Kit za analizu. ML Kit vraća rezultate o tome postoji li lice na slici i ako postoji onda vraća pozicije lica. Zbog ove kombinacije moguće je automatizirati obradu slika koje su se prije ručno radile uz pomoć Haar-a, tako da se kroz API poziv automatski šalje slika koja je izrezana i obrađena.

8.2. Arhitektura

Aplikacija je razvijena koristeći arhitekturu MVVM (Model-View-ViewModel). Uz pomoć ove arhitekture moguće je odvojiti logiku aplikacije od korisničkog sučelja čime se postiže bolja modularnost, testabilnost i održivost koda.



Slika 36: MVVM arhitektura, preuzeto sa [53]

Model je sloj modela koji upravlja podacima aplikacije. U ovom slučaju, model predstavlja podatke o korisnicima, predmetima, prisutnosti, te rezultatima prepoznavanja lica. Model također definira i logiku aplikacije, te pristup podacima preko API-ja.

Sloj prikaza (View) zadužen je za prikazivanje podataka korisniku. Korištenjem Jetpack Composea, sučelje je izgrađeno na jednostavan i brz način, te je prilagodljivo različitim

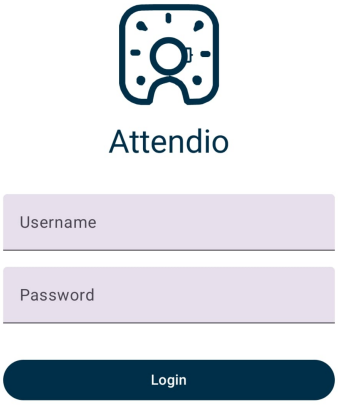
veličinama ekrana.

ViewModel služi kao posrednik između View-a i Model-a, Odnosno tamo se nalazi sva logika kao što su upute za korisničku interakciju i upravljanje stanjima prikaza, učitavanje podataka ili prikazivanje pogrešaka. Na taj način sloj prikaza ima samo zadaću prikazati podatke ne znajući ništa o njima.

Za upravljanje ovisnostima unutar aplikacije, korišten je Hilt, biblioteka za Dependency Injection (DI). Hilt ubrizgava ovisnosti u različite komponente aplikacije, poput ViewModela, API klijenata i smanjuje kreiranja instanci na nepotrebna mjesta.

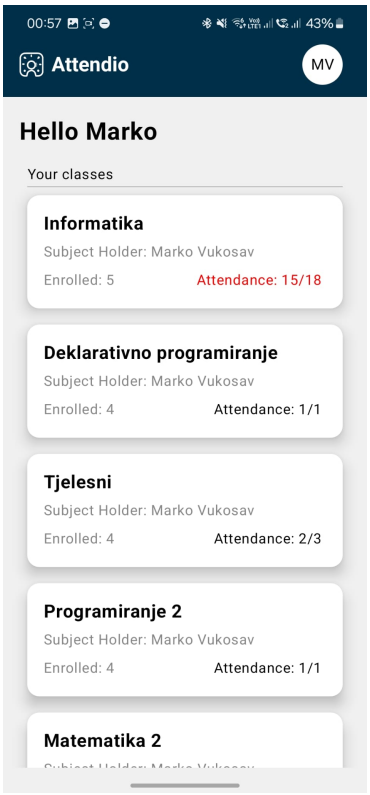
8.3. Funkcionalnosti

Aplikacija je osmišljena kako bi korisnicima pružila jednostavan način ali i interaktivni način za prijavu prisustva. Kada korisnik otvori aplikaciju otvara se ekran za prijavu ili početni ekran ukoliko je korisnik već bio prijavljen.



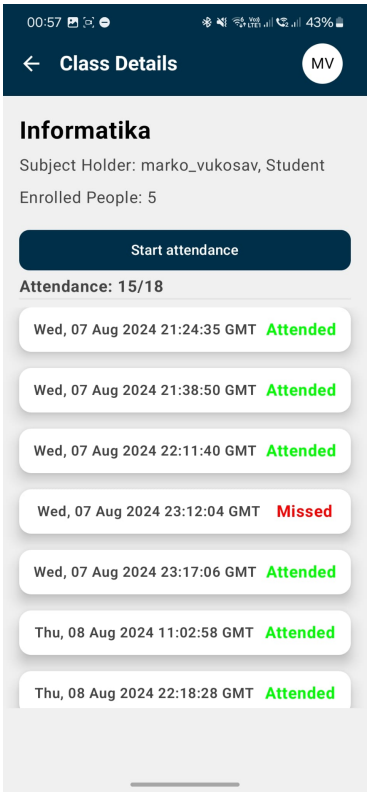
Slika 37: Ekran za prijavu

Korisnici se prijavljuju u sustav koristeći svoju klasifikaciju i lozinku. Nakon uspješne prijave, korisniku se prikazuje početni ekran s popisom predmeta. Početni ekran prikazuje sve predmete na koje je student upisan. Studenti mogu odabrati kolegij kako bi vidjeli više informacija, uključujući prisutnost, i mogućnost prijave prisutnosti ako je otvorena.

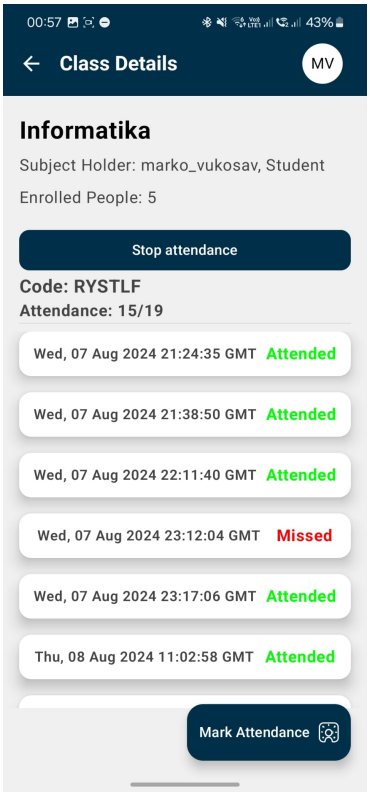


Slika 38: Početni ekran

Ekran detalja kolegija sadrži informacije o odabranom predmetu pregled dosadašnje prisutnosti i mogućnost prijave prisutnosti.

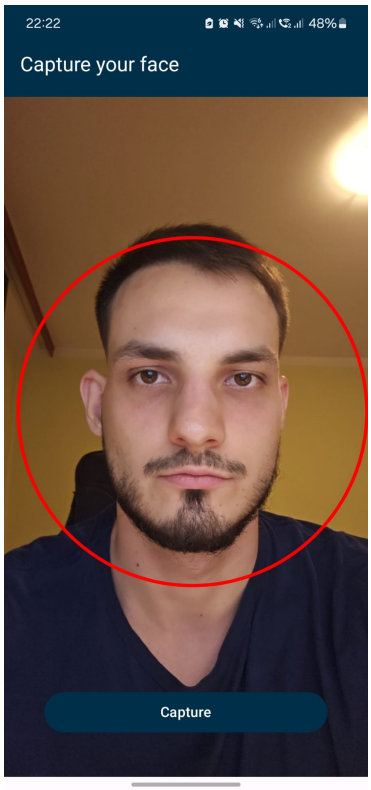


Slika 39: Detalji kolegija

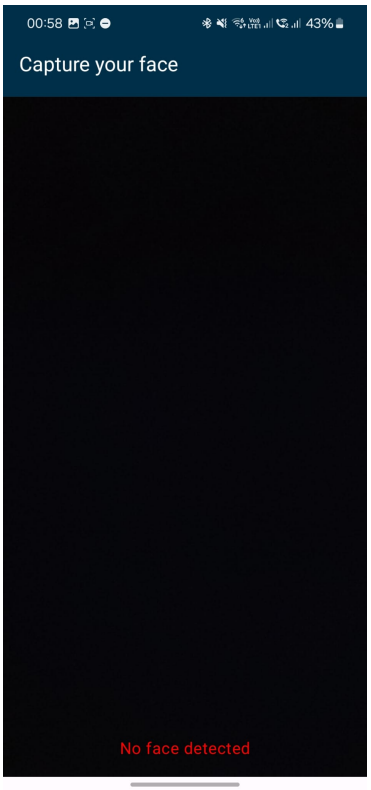


Slika 40: Detalji kolegija s mogućnošću prijave

Ekran za kreiranje slika. Kada je prijava aktivna, studenti mogu koristiti ovaj ekran za kreiranje slike za kasnije prepoznavanje. Ako se lice detektira tada će korisniku crvenom kružnicom biti naznačeno lice i omogućiti će se gumb za kreiranje slika. Ako lice ne bude detektirano, tada će biti prikazana poruka.

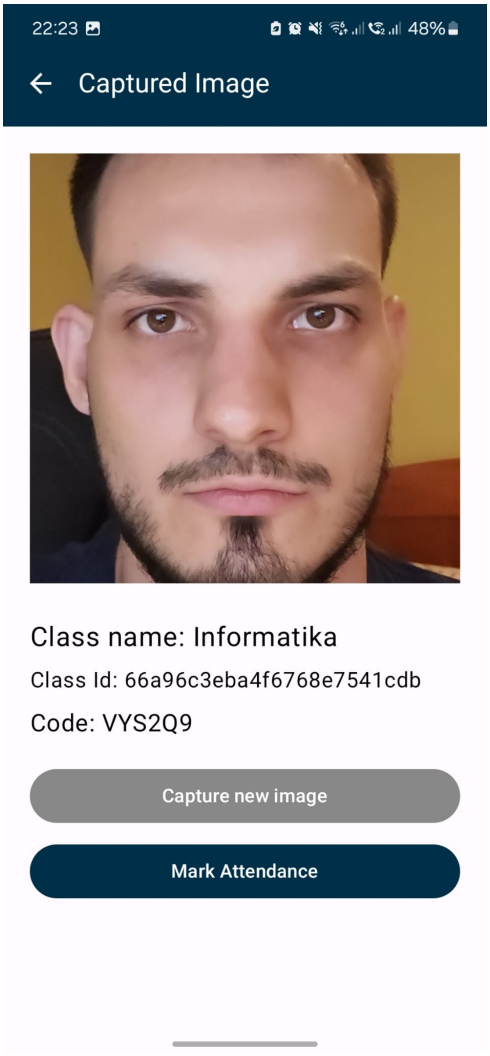


Slika 41: Ekrana za kreiranje slika sa licem



Slika 42: Ekrana za kreiranje slika bez lice

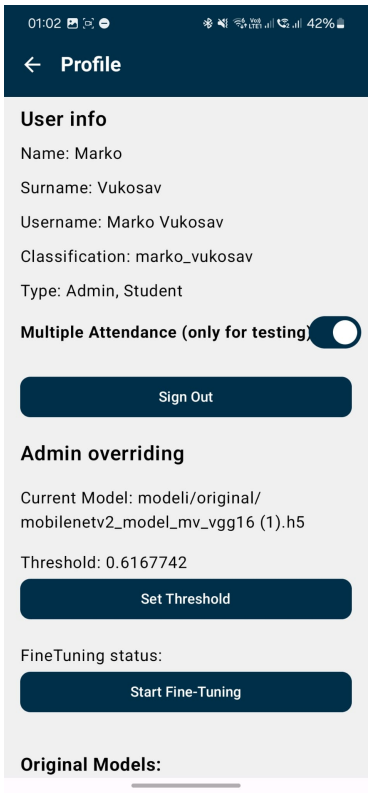
Nakon što je slika kreirana korisnik će se prikazati ekran za prijavu prisutnosti koji prikazuje kreiranu sliku i mogućnost za prijavu ili ponovno slikanje.



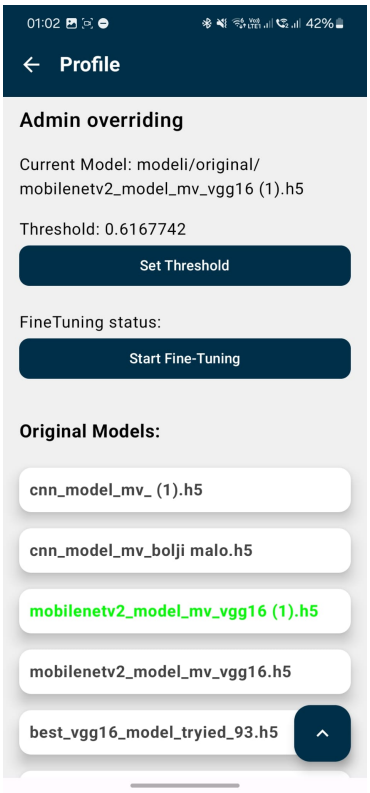
Slika 43: Prikaz ekrana za prijavu prisutnosti

Ako korisnik odabere prijavu tada se šalje ta slika sa korisničkom klasifikacijom i model provjerava da li ta slika pripada toj klasi i vraća rezultat. Ako je on pozitivan, iznad odabranog praga tada se korisnik automatski prijavljuje na prisustvu ako nije tada će mu se ponuditi opcija prijava sa kodom gdje će se poslati zadnja kreirana slika za kasnije fino podešavanje modela. Nakon toga korisnik će biti proslijeđen natrag na ekran za detalje kolegija.

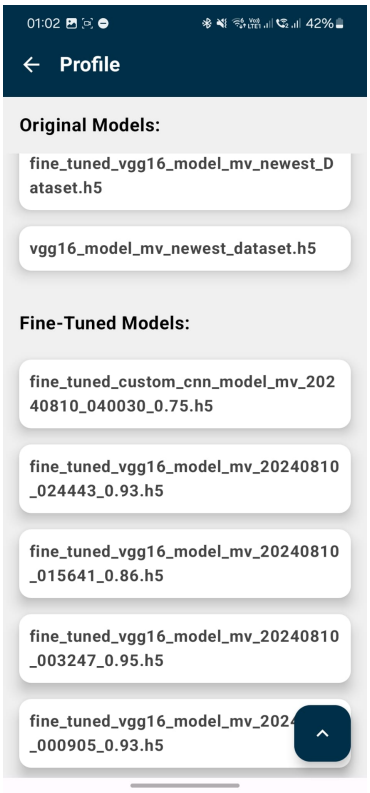
Korisnik svoje detalje može vidjeti na ekranu profil gdje ovisno o tipu korisnika (student, administrator), ekran korisničkog profila omogućava pregled podataka i odjavu. Administratori mogu upravljati dostupnim modelima za prepoznavanje lica, nadogradnjom modela, i prilagodbom praga prepoznavanja za poboljšanje točnosti.



Slika 44: Ekran profil 1



Slika 45: Ekran profil 2



Slika 46: Ekran profil 3

9. Evaluacija modela u praksi

Evaluacija modela provedena je u stvarnim uvjetima prijave prisutnosti putem mobilne aplikacije. Za svaki model (VGG16, CNN i MobileNetV2), se posebno vršila prijava prisutnosti i približno se tražio jednak broj prijava od svake osobe. Cilj evaluacije bio je utvrditi učinkovitost modela u prepoznavanju korisnika i njihovoj prijavi prisutnosti, kao i provjeriti ponašanje sustava u slučajevima kada se korisnik pokušava prijaviti kao druga osoba. Rezultati prijava su podijeljene u 4 kategorije koje su se mogle iskoristiti za računanje samih metrika.

Ispravne pozitivna (TP): Korisnici koje je sustav ispravno prepoznao kao pripadnika određene klase (osobe), kao rezultat u bazi se prikazalo kao prijava putem metode faceRecognition.

Lažne negativna (FN): Korisnici koje sustav nije ispravno prepoznao, unatoč tome što pripadaju toj klasi. U ovom slučaju, prijava se obavila preko koda jer sustav nije uspio prepoznati lice korisnika kao ispravno, iako je korisnik bio prisutan.

Kako bi se utvrdile varijable ispravne negativna (TN) i lažne pozitivna (FP), korisnici su se prijavili kao druga osoba i pokušali kreirati prisutnost kao druge osobe.

Ispravne negativna (TN): Korisnici koji su pokušali lažno prijaviti prisutnost kao druga osoba, ali ih je sustav ispravno identificirao kao neispravne. Prijava je zatim obavljena putem koda jer sustav nije prihvatio njihovo lice.

Lažne pozitivna (FP): Korisnici koji su se prijavili kao druga osoba, a sustav je pogrešno prepoznao njihovo lice kao ispravno i prijavio ih putem metode faceRecognition, iako nisu stvarno pripadali toj klasi.

9.1. CNN evaluacija

Model CNN nakon provedenoga testiranja dao je sljedeće rezultate:

Broj ispravnih pozitivnih prijava (TP): 70/106

Model je uspio ispravno prepoznati korisnike u 70 slučajeva, odnosno uspješno je klasificirao slike za tu osobu.

Broj lažnih negativnih prijava (FN): 36/106 U 36 slučajeva, sustav nije prepoznao korisnike iako su trebali biti prepoznati.

Broj ispravnih negativnih prijava (TN): 22/24 Model je ispravno prepoznao 22 korisnika kao neispravne prijave, odnosno sustav ih nije klasificirao sa dovoljnom točnošću da pripadaju toj klasi.

Broj lažnih pozitivnih prijava (FP): 2/24 U 2 slučaja sustav je pogrešno prepoznao korisnika kao ispravnog, iako je prijava bila lažna, odnosno sustav je krivo klasificirao osobu kao da pripada toj klasi.

Iz tih rezultata mogle su se napraviti metrike poput preciznosti, odziva, točnosti te F1

mjere.

FP = 2 TN = 22 TP = 70 FN = 36

Preciznost = 70 / (70 + 2) = 0.9722 Odziv = 70 / (70 + 36) = 0.6604 F1 mjera = 2 * (0.9722 * 0.6604) / (0.9722 + 0.6604) = 0.7887 Točnost = (70 + 22) / (70 + 22 + 2 + 36) = 0.7925

CNN model ima visoku preciznost od 97.22% u prepoznavanju korisnika, ali povremeno pogriješi prepoznati sve odnosno odziv iznosi 66.04%, s ukupnom točnošću od 79.25% i F1-mjerom od 78.87%.

9.2. VGG16 evaluacija

Model VGG16 nakon provedenoga testiranja dao je sljedeće rezultate:

Broj ispravnih pozitivnih prijava (TP): 55/67 Model je uspio ispravno prepoznati korisnike u 55 slučajeva, odnosno uspješno je klasificirao slike za tu osobu.

Broj lažnih negativnih prijava (FN): 8/67 U 8 slučajeva, sustav nije prepoznao korisnike iako su trebali biti prepoznati.

Broj ispravnih negativnih prijava (TN): 29/30 Model je ispravno prepoznao 29 korisnika kao neispravne prijave, odnosno sustav ih nije klasificirao sa dovoljnom točnošću da pripadaju toj klasi.

Broj lažnih pozitivnih prijava (FP): 1/30 U 1 slučaju sustav je pogrešno prepoznao korisnika kao ispravnog, iako je prijava bila lažna, odnosno sustav je krivo klasificirao osobu kao da pripada toj klasi.

Iz tih rezultata mogle su se napraviti metrike poput preciznosti, odziva, točnosti te F1 mjere.

FP = 1 TN = 29 TP = 55 FN = 8

Preciznost = 55 / (55 + 1) = 0.9821 Odziv = 55 / (55 + 8) = 0.8730 F1 mjera = 2 * (0.9821 * 0.8730) / (0.9821 + 0.8730) = 0.9248 Točnost = (55 + 29) / (55 + 29 + 1 + 8) = 0.9542

VGG16 model ima visoku preciznost od 98.21% u prepoznavanju korisnika, ali povremeno propušta prepoznati sve korisnike, s odzivom od 87.30%. Ukupna točnost modela iznosi 95.42 %, a F1 mjera je 92.48%.

9.3. MobileNetV2 evaluacija

Model MobileNetV2 nakon provedenog testiranja dao je sljedeće rezultate:

Broj ispravnih pozitivnih prijava (TP): 58/66 Model je uspio ispravno prepoznati korisnike u 58 slučajeva, odnosno uspješno je klasificirao slike za tu osobu.

Broj lažnih negativnih prijava (FN): 8/66 U 8 slučajeva, sustav nije prepoznao korisnike iako su trebali biti prepoznati.

Broj ispravnih negativnih prijava (TN): 29/30 Model je ispravno prepoznao 29 korisnika kao neispravne prijave, odnosno sustav ih nije klasificirao sa dovoljnom točnošću da pripadaju toj klasi.

Broj lažnih pozitivnih prijava (FP): 1/30 U 1 slučaju sustav je pogrešno prepoznao korisnika kao ispravnog, iako je prijava bila lažna, odnosno sustav je krivo klasificirao osobu kao da pripada toj klasi.

Iz tih rezultata mogle su se napraviti metrike poput preciznosti, odziva, točnosti te F1 mjere.

FP = 1 TN = 29 TP = 58 FN = 8

Preciznost = 58 / (58 + 1) = 0.9831 Odziv = 58 / (58 + 8) = 0.8788 F1 mjera = 2 * (0.9831 * 0.8788) / (0.9831 + 0.8788) = 0.9276 Točnost = (58 + 29) / (58 + 29 + 1 + 8) = 0.9552

MobileNetV2 model ima visoku preciznost od 98.31% u prepoznavanju korisnika, ali povremeno propušta prepoznati sve korisnike, s odzivom od 87.88%. Ukupna točnost modela iznosi 95.52%, a F1 mjera je 92.76%.

10. Analiza rezultata

Analiza rezultat obavit će se na temelju metrika koje su dobivene tijekom treniranja modela i metrika na stvarnom sustava za svaki model. Sljedeće tablice predstavljaju rezultate evaluacije modela CNN, VGG16 i MobileNetV2 nad dva različita skupa podataka: testnim podacima i podacima iz stvarnog sustava.

Prva tablica prikazuje rezultate evaluacije modela na testnim podacima, koji su obično dobro uravnoteženi i predstavljaju uvjete pod kojima su modeli trenirani. VGG16 pokazuje najbolje rezultate sa stopostotnom metrikama. MobileNetV2 i CNN također pokazuju dobre performanse, ali malo zaostaju za VGG16

	CNN	VGG16	MobileNetV2
Točnost	0.92	1.00	0.98
Preciznost	0.93	1.00	0.98
Odziv	0.92	1.00	0.98
F1 mjera	0.92	1.00	0.98

Tablica 1: Evaluacija modela na testnim podacima

Druga tablica pokazuje rezultate evaluacije na stvarnim podacima. Ovdje se vidi da iako su modeli zadržali visoke performanse, došlo je do smanjenja točnosti i odziva u stvarnom okruženju. CNN ima najniži odziv (0.6604), što govori da model zna krivo prepoznati korisnika u stvarnim uvjetima. VGG16 i MobileNetV2 su imaju bolje rezultate, što govori da mogu bolje generalizirati nad stvarnim podacima.

	CNN	VGG16	MobileNetV2
Preciznost	0.9722	0.9821	0.9833
Odziv	0.6604	0.8730	0.8901
Točnost	0.7925	0.9542	0.9625
F1 mjera	0.7887	0.9248	0.9333

Tablica 2: Evaluacija modela na stvarnom sustavu na temelju različitih metrika

Svi modeli pokazuju vrlo dobre rezultate, s tim da predtrenirani modeli VGG16 i MobileNetV2 imaju bolju sposobnost generalizacije osoba u stvarnim uvjetima dok CNN model se pokazao kao nešto slabiji, ali još uvijek ima solidne rezultate. Predtrenirani modeli su se pokazali očekivano kao bolji jer su ti modeli već prije trenirani na velikom broju podataka i kompleksniji su od CNN modela. CNN model je treniran na manjem skupu podataka i kao takav je pokazao solidne rezultate. Daljnjim finim podešavanjem i treniranje na većim skupovima. podataka postoji mogućnost da model dostigne razinu predtreniranih modela.

11. Zaključak

U ovom radu obrađena je implementacija i evaluacija modela strojnog učenja u svrhu prepoznavanja lica, u primjeni u mobilnim okruženjima. Rad obuhvaća razvoj prilagođenog modela temeljenog na konvolucijskoj neuronskoj mreži (CNN) te usporedbu s predtreniranima modelima kao što su VGG16 i MobileNetV2.

Kroz rad su primijenjeni razni alati kao što je Haar cascade koji je kasnije zamijenio ML Kit za detekciju lica, CameraX za obradu slika, Retrofit za komunikaciju s backend servisima, Google Disk kao pohrana slika i treniranih modela i sličnih alata. Svaki proces se pokušao automatizirati kako bi se mogao predstaviti što stvarniji sustav. Na primjer ručno skupljanje podataka i obrada se automatizirala uz pomoć mobilne aplikacije gdje se detekcija i obrada slike vrši dok se slike spremaju na Google Disk odkuda se kasnije dohvaća za dodatno treniranje modela.

Evaluacija modela izvršena je na temelju metrika poput preciznosti, točnosti, odziva i F1-mjere, gdje se kreirala usporedba između testnih podataka nakon treniranja i stvarnih podataka u stvarnoj primjeni.

Rezultati evaluacije su pokazali kako predtrenirani modeli, zbog svoje kompleksnije arhitekture i treniranosti na velikom broju podataka, pružaju bolje performanse u stvarnim uvjetima. Vlastiti CNN model također pokazuje solidne rezultate, gdje bi uz daljnje usavršavanje putem finog podešavanja i treniranja na većim skupovima podataka postojala mogućnost za napredak. Zaključno korištenjem strojnog učenja u obrazovanju može doprinijeti digitalizaciji i automatizaciji procesa prijave prisustva na nastavi, što može povećati koncentraciju studenata na nastavi, povećati broj dolazaka, te olakšati pratiti stanje svojih dolazaka odnosno ne dolazaka. Uz dovoljno velik skup podataka moguće je iskoristiti neki od predtreniranih modela sa dodanim slojevima i kreirati dovoljno dobar model s dobrim performansama koji bi uspješno izvršavao zadatak prijave prisustva.

Popis literature

[1] A. Group. „Mali rječnik digitalne transformacije.” Pristupljeno: Kolovoz 2024. (2019.), adresa: https://aestus.hr/mali-rjecnik-digitalne-transformacije.

[2] J. Josip, *Sustav za provjeru identiteta osobe*, Pristupljeno: Kolovoz 2024, 2021. adresa: https://urn.nsk.hr/urn:nbn:hr:200:656325.

[3] N. Bolf, *Strojno ucenje*, Pristupljeno: Kolovoz 2024, 2024. adresa: https://hrcak.srce.hr/file/382926.

[4] OpenCV. „OCR of Hand-written Data using kNN.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://docs.opencv.org/4.x/d8/d4b/tutorial_py_knn_opencv.html.

[5] K. Patel, J. Fogarty, J. Landay i B. Harrison, „Investigating statistical machine learning as a tool for software development,” travanj 2008., str. 667–676. DOI: 10.1145/1357054.1357160.

[6] I. 2024 Salesforce. „Machine learning examples.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.tableau.com/learn/articles/machine-learning-examples.

[7] NVIDIA. „What is a Recommendation System?” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.nvidia.com/en-us/glossary/recommendation-system/.

[8] NVIDIA. „Recommendation System Image.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.nvidia.com/en-us/glossary/recommendation-system.

[9] Imgur. „Manual Image on Imgur.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://imgur.com/a/manual-K6m91n3MGTGTw.

[10] S. Alvarez. „Tesla Autopilot’s Traffic Light and Stop Sign Detection Details Leaked.” Pristupljeno: Kolovoz 14, 2024. (2024.), adresa: https://www.teslarati.com/tesla-autopilot-traffic-light-stop-sign-detection-details-leaked/.

[11] Coursera. „10 Machine Learning Examples.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.coursera.org/articles/machine-learning-examples.

[12] Aditya. „Reinforcement Learning in Chess.” Pristupljeno: Kolovoz 2024. (2023.), adresa: https://medium.com/@samgill1256/reinforcement-learning-in-chess-73d97fad96b3.

[13] elementsofai. „Regresija.” Pristupljeno: Kolovoz 2024. (2019.), adresa: https://intuitivetutorial
com/2023/04/07/k-nearest-neighbors-algorithm/.

[14] E. of AI. „Različiti pristupi strojnog učenja.” Pristupljeno: Kolovoz 2024. (2024.), adresa:
https://course.elementsofai.com/hr/4/3.

[15] elementsofai. „Vrste strojnog učenja.” Pristupljeno: Kolovoz 2024. (2019.), adresa: https:
//course.elementsofai.com/hr/4/1.

[16] elementsofai. „Klasifikator najbližih susjeda.” Pristupljeno: Kolovoz 2024. (2019.), adresa:
https://course.elementsofai.com/hr/4/2.

[17] R. S. H. „K-Nearest Neighbors Algorithm.” Pristupljeno: Kolovoz 2024. (2019.), adresa:
https://course.elementsofai.com/hr/4/3.

[18] S. Russell i P. Norvig, *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson,
2020., ISBN: 9780134610993. adresa: http://aima.cs.berkeley.edu/.

[19] K. Kian, „On the analysis of political sentiment in presidential elections in Afghanistan
using social media data,” disertacija, prosinac 2020. DOI: 10.13140/RG.2.2.20642.
91840.

[20] IBM. „What is a neural network?” Pristupljeno: Kolovoz 2024. (2021.), adresa: https:
//www.ibm.com/topics/neural-networks.

[21] elementsofai. „Osnove neuronskih mreža.” Pristupljeno: Kolovoz 2024. (2019.), adresa:
https://course.elementsofai.com/hr/5/1.

[22] R. Thakur. „Beginner's Guide to VGG16 Implementation in Keras.” Pristupljeno: Kolovoz
2024. (2024.), adresa: https://builtin.com/machine-learning/vgg16.

[23] G. Learning. „Everything You Need to Know About VGG16.” Pristupljeno: Kolovoz 2024.
(2024.), adresa: https://medium.com/@mygreatlearning/everything-you-
need-to-know-about-vgg16-7315defb5918.

[24] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Springer Internati-
onal Publishing, 2018. adresa: http://ndl.ethernet.edu.et/bitstream/
123456789/88552/1/2018_Book_NeuralNetworksAndDeepLearning.pdf.

[25] T. D. Science. „Perceptron: The Artificial Neuron.” Pristupljeno: Kolovoz 2024. (2024.),
adresa: https://towardsdatascience.com/perceptron-the-artificial-
neuron-4d8c70d5cc8d.

[26] Amazon. „https://aws.amazon.com/what-is/gan/.” Pristupljeno: Kolovoz 2024. (2024.), adresa:
https://aws.amazon.com/what-is/gan.

[27] J. Šnajder. „Vrednovanje modela.” Pristupljeno: Kolovoz 2024. (2022.), adresa: https:
//www.fer.unizg.hr/_download/repository/SU1-2022-P21-VrednovanjeModela.
pdf.

[28] Google. „Classification: Accuracy, recall, precision, and related metrics.” Pristupljeno:
Kolovoz 2024. (2024.), adresa: https://developers.google.com/machine-
learning/crash-course/classification/accuracy-precision-recall.

[29] Google. „Linear regression.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://developers.google.com/machine-learning/crash-course/linear-regression/loss.

[30] I. Logunova. „A Guide to F1 Score.” Pristupljeno: Kolovoz 2024. (2021.), adresa: https://serokell.io/blog/a-guide-to-f1-score.

[31] Geeks for Geeks. „Understanding the Confusion Matrix in Machine Learning.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.geeksforgeeks.org/confusion-matrix-machine-learning.

[32] M. Sorokin. „Loss functions. Comprehensive Guide to Loss Functions in Various Machine Learning Domains.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://medium.com/@vergotten/loss-functions-comprehensive-guide-to-loss-functions-in-various-machine-learning-domains-1e76f7a9b584.

[33] S. Gupta. „The 7 Most Common Machine Learning Loss Functions.” Pristupljeno: Kolovoz 2024. (2023.), adresa: https://builtin.com/machine-learning/common-loss-functions.

[34] kedarps. „Cross Entropy vs. Sparse Cross Entropy: When to use one over the other.” Pristupljeno: Kolovoz 2024. (2017.), adresa: https://stats.stackexchange.com/questions/326065/cross-entropy-vs-sparse-cross-entropy-when-to-use-one-over-the-other.

[35] Geeks For Geeks. „ml one hot encoding.” Pristupljeno: Kolovoz 2024. (2021.), adresa: https://www.geeksforgeeks.org/ml-one-hot-encoding.

[36] Innovatrics. „Facial Recognition Technology.” Pristupljeno: Kolovoz 2024. (2023.), adresa: https://www.innovatrics.com/facial-recognition-technology.

[37] Oloyede, *review on face recognition systems: recent approaches and challenges.* Pristupljeno: Kolovoz 2024, 2020. adresa: https://doi.org/10.1007/s11042-020-09261-2.

[38] H. Li, *A Convolutional Neural Network Cascade for Face Detection*, Pristupljeno: Kolovoz 2024, 2015. adresa: https://openaccess.thecvf.com/content_cvpr_2015/papers/Li_A_Convolutional_Neural_2015_CVPR_paper.pdf.

[39] M. Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, Pristupljeno: Kolovoz 2024, 2001. adresa: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf.

[40] OpenCV. „Haar Cascade Example.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[41] P. Viola i M. J. Jones, *Robust real-time face detection*, 2004.

[42] E. Academy. „Describe the structure of a CNN, including the role of hidden layers and the fully connected layer.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://hr.eitca.org/artificial-intelligence/eitc-ai-dltf-deep-learning-with-tensorflow/convolutional-neural-networks-in-tensorflow/convolutional-neural-networks-basics/examination-review-convolutional-neural-networks-basics/describe-the-structure-of-a-cnn-including-the-role-of-hidden-layers-and-the-fully-connected-layer/.

[43] S. Adnan, *Facial Feature Extraction For Face Recognition*, Pristupljeno: Kolovoz 2024, 2020. adresa: https://iopscience.iop.org/article/10.1088/1742-6596/1664/1/012050/pdf.

[44] K. He, X. Zhang, S. Ren i J. Sun, *Deep Residual Learning for Image Recognition*, Pristupljeno: Kolovoz 2024, 2015. arXiv: 1511.08458 [cs.CV]. adresa: https://arxiv.org/pdf/1511.08458.

[45] Guru99. „Convolutional Neural Network Tutorial in TensorFlow.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.guru99.com/hr/convnet-tensorflow-image-classification.html.

[46] A. Adria. „Prvi svjetski kodeks o etičkoj upotrebi prepoznavanja lica.” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.asadria.com/prvi-svjetski-kodeks-o-etickoj-upotrebi-prepoznavanja-lica/.

[47] FRA. „Facial recognition technology fundamental rights considerations in the context of law enforcemen.” Pristupljeno: Kolovoz 2024. (2019.), adresa: https://fra.europa.eu/en/publication/2019/facial-recognition-technology-fundamental-rights-considerations-context-law.

[48] K. Hill i R. Mac. „Facebook, Citing Societal Concerns, Plans to Shut Down Facial Recognition System.” Pristupljeno: Kolovoz 2024. (2023.), adresa: https://www.nytimes.com/2021/11/02/technology/facebook-facial-recognition.html.

[49] S. Singh. „Machine learning steps: A complete guide for beginner in ML.” Pristupljeno: Kolovoz 2024. (2022.), adresa: https://www.labellerr.com/blog/machine-learning-steps-a-complete-guide-to-the-ml-process/.

[50] Quora. „Why are Instagram photos backwards?” Pristupljeno: Kolovoz 2024. (2024.), adresa: https://www.quora.com/Why-are-Instagram-photos-backwards.

[51] „basics of cnn in deep learning.” Pristupljeno: Kolovoz 14, 2024. (2024.), adresa: https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/.

[52] A. Oppermann. „Optimization in Deep Learning: AdaGrad, RMSProp, ADAM.” Pristupljeno: Kolovoz 2024. (2021.), adresa: https://artemoppermann.com/optimization-in-deep-learning-adagrad-rmsprop-adam.

[53] M. Brian. „Mastering Android MVVM Architecture: Developers’ Guide.” Pristupljeno: Kolovoz 14, 2024. (2024.), adresa: https://medium.com/@mutebibrian256/mastering-android-mvvm-architecture-developers-guide-3271e4c8908b.

Popis slika

Popis tablica

Popis isječaka koda