SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

Karlo Posavec

MaskValidator - web aplikacija za detekciju maske na licu

Zagreb, 2021.

Ovaj rad izrađen je na Zavodu za robotiku i automatizaciju proizvodnih sustava pod vodstvom doc. dr. sc. Tomislava Stipančića i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2020./2021.

Karlo Posavec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

UVOD

1.1. Umjetna inteligencija

U posljednjih nekoliko desetljeća računala su se počela koristiti za automatizaciju raznih procesa koji koriste podatke kao što su klasifikacija, predviđanje, prepoznavanje i pretraživanje podataka. Računala mogu obavljati te zadatke zahvaljujući umjetnoj inteligenciji.

Umjetna inteligencija (UI, prema engl. akronimu AI, od *artificial intelligence)* je dio računalne znanosti (informatike) koja se bavi razvojem sposobnosti računala da obavljaju zadaće za koje je potreban neki oblik inteligencije, tj. da se mogu snalaziti u novim prilikama, učiti nove koncepte, donositi zaključke, razumjeti prirodni jezik, raspoznavati prizore i dr.[1].

Prema stupnju inteligencije, umjetna inteligencija dijeli se na [2]:

- **slabu umjetnu inteligenciju**- koja je karakteristična za strojeve koji mogu reagirati na specifične situacije, ali ne mogu misliti za sebe.
- **jaku umjetnu inteligenciju** -koja je karakteristična za strojeve koji mogu razmišljati i djelovati kao ljudi. Kako još nema njezinih primjera, njena najbolja reprezentacija je ona iz filmova.

Umjetna inteligencija je razvijena sa zamišlju da računala uče i obavljaju zadatke kao što to rade ljudi tj. na načelu pokušaja i pogrešaka. Značajan impuls razvoju umjetne inteligencije došao je u posljednje vrijeme u obliku razvoja novih i unaprjeđenja starih sofisticiranih algoritama kao što su umjetne neuronske mreže zatim razvojem i pojeftinjenjem računalne snage te na kraju generiranjem ogromne količine podataka prema [3] 90% svjetskih podataka stvoreno je u posljednje dvije godine.

Područja umjetne inteligencije u kojima se trenutno provodi najviše istraživanja i čija se upotreba naveliko proširila izvan akademske zajednice u mnoge kompanije privatnog sektora su: strojno učenje, računalni vid, robotika, umjetne neuronske mreže i duboko učenje .

Trenutno se umjetna inteligencija koristi za automatizaciju procesa koji sadrže veliku količinu podataka, povećavanje efikasnosti procesa i kao pomoć u donošenju odluka na temelju podataka. Nadanja su kako bi ona mogla u budućnosti zamijeniti ljude kod izvođenja repetitivnih zadatka kako bi se oni mogli usredotočiti na misaone probleme i na razvoj novih ideja.

1.2. Motivacija

Svijet se trenutno nalazi uslijed pandemije uzrokovane koronavirusom. Svjetska zdravstvena organizacija nazvala ga je SARS-CoV-2, a bolest koju uzrokuje COVID 19 . Otkriven je u Kini krajem 2019. godine nakon čega se proširio svijetom i u ožujku 2020 doveo do proglašenja globalne pandemije od strane svjetske zdravstvene organizacije. [4]

Nedugo nakon proglašenja pandemije cijeli svijet se našao u nekom obliku karantene. To je dovelo do mnogo neželjenih i štetnih posljedica .Uvođenjem karantene ograničeno je kretanje i nitko osim hitnih službi nije mogao ići na posao što je dovelo zatvaranja mnogih kompanija i obrta pri čemu su najbolje stradala mala i srednje velika poduzeća. Direktne i indirektne posljedice karantene su pad ekonomskog rasta, otpuštanje radnika, ograničavanje slobode kretanja, produljivanje liste čekanja u bolnicama, pogoršavanje mentalnog zdravlja nacije i povećano zaduživanje države.

COVID-19 je bolest koja se prenosi kapljičnim putem. To znači da se prenosi primarno s osobe na osobu preko malih kapljica iz nosa ili usta koje se izbacuju kad oboljela osoba kiše, govori ili kašlje. Te kapljice su relativno teške, ne prenose se na veliku udaljenost i relativno brzo padaju na stvari i površine u okolini zaražene osobe. To dovodi do kontaminiranja tih objekata preko kojih se osobe mogu indirektno zaraziti ako dođu s njima u kontakt. Razdoblje inkubacije tj. vrijeme od izloženosti virusu do početka simptoma traje između 2 i 12 dana.

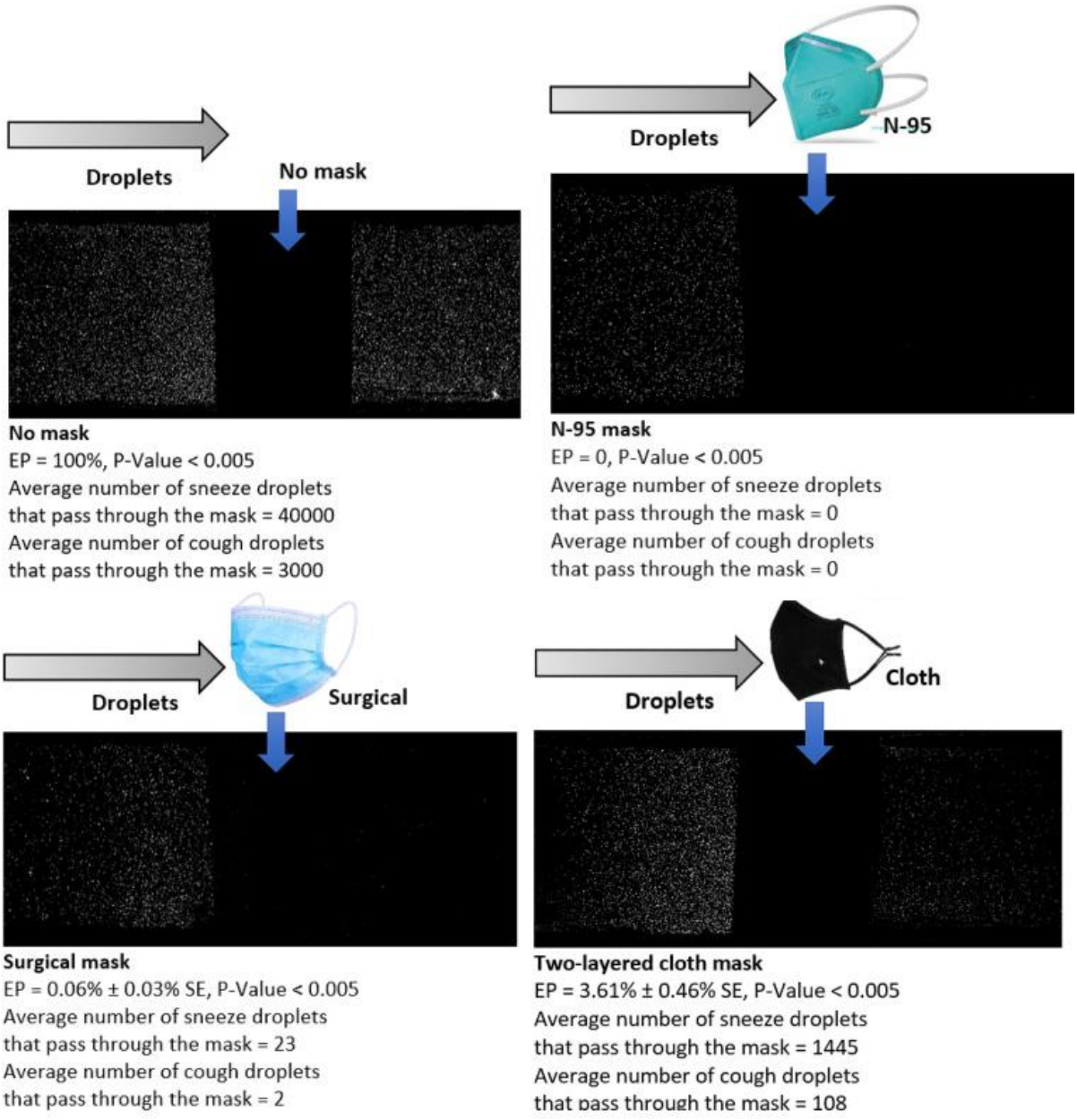
Najčešće se javljaju simptomi slični gripi poput [4]:

- povišene tjelesne temperature
- kašlja
- kratkog daha
- bolova u mišićima
- umora
- gubitka osjeta okusa i mirisa

U težim slučajevima javlja se teška upala pluća, sindrom akutnog otežanog disanja, sepsa i septički šok koji mogu uzrokovati smrt pacijenta. Osobe koje boluju od kroničnih bolesti podložnije su težim oboljenjima.

Preventivne mjere koje se preporučuju su: održavanje razmaka od 1 metar na otvorenom i od 2 metra u zatvorenim prostorijama, redovito pranje ruku, izbjegavanje mjesta gdje se okuplja velik broj ljudi, provjetravanje zatvorenih prostorija i nošenje maske za lice. Maska za lice se preporuča u okolnostima kad se okuplja veći broj ljudi na otvorenom, u liftovima, prilikom

čekanja u redu i u javnom prijevozu. Na slici [**Pogreška! Izvor reference nije pronađen.**.] su p rikazani rezultati dobiveni u istraživanju [5] koji prikazuju koliki postotak kapljica nastalih pri kihanju ili kašljanju prođe kroz pojedini tip maske, a on je na slici označen s EP.



Slika 1.1. Postotak kapljica nastalih pri kihanju ili kašljanju koji prođe kroz pojedini tip maske [5]

Iz slike [Slika 1.1.] se jasno vidi utjecaj nošenja maske kod sprječavanja širenja bolesti koje se prenose kapljično pod koje spada i COVID-19 pa je na temelju toga kao projekt odabran web aplikacija koja će omogućiti prepoznavanje nošenja maske za lice. Ovaj projekt odabran je s ciljem kako bi se ubrzao povratak u normalno i kako bi se pomoglo u prevenciji i usporavanju širenja koronavirusa. Dodatan razlog za provedbu ovog projekta je primjenjivost

ovog rješenja kod novih bolesti i pandemija koje će se širiti kapljično i samim time zahtijevati nošenje maske za lice.

1.3. Strukturalna podjela rada

Rad je strukturalno podijeljen tako da je prvo dana teorijska podloga u kojoj su opisani najvažniji pojmovi, nakon toga je prikazana izrada same web aplikacije te su na kraju prikazani rezultati i izveden je zaključak.

U drugom poglavlju dane su kratke teorijske osnove pojmova koji su potrebni za shvaćanje same web aplikacije. U ovom poglavlju opisani su računalni vid, strojno učenje, duboko učenje, umjetne neuronske mreže te konvolucijske neuronske mreže. Za svaki pojam dana je definicija, osnovni princip i područja primjene.

Treće poglavlje opisuje izradu same web aplikacije. Samo poglavlje započinje prikazom konceptualne razrade zadatka nakon čega je opisan programski jezik Python i njegove biblioteke koje su korištene u ovom radu. Zatim je prikazan izabrani skup podataka(engl. dataset) te na kraju dolazi opis postupka izrade zadatka u Pythonu. Postupak izrade je podijeljen na dva dijela i to tako da prvi dio opisuje postupak izrade modela koji će vršiti pretpostavke o nošenju maski, a drugi dio opisuje izradu grafičkog sučelja te funkcionalnosti povezanih uz njega tj. predviđanje nošenja ili ne nošenja maski u slikama te pomoću web kamere u realnom vremenu .

Eksperimentalni rezultati primjene cjelokupno izrađene web aplikacije MaskValidator prikazani su u četvrtom poglavlju. Ovo poglavlje se sastoji od rezultata dobivenih koristeći konvencionalne alate za evaluaciju neuronskih mreža i rezultata dobivenih korištenjem funkcionalnosti web aplikacije MaskValidator.

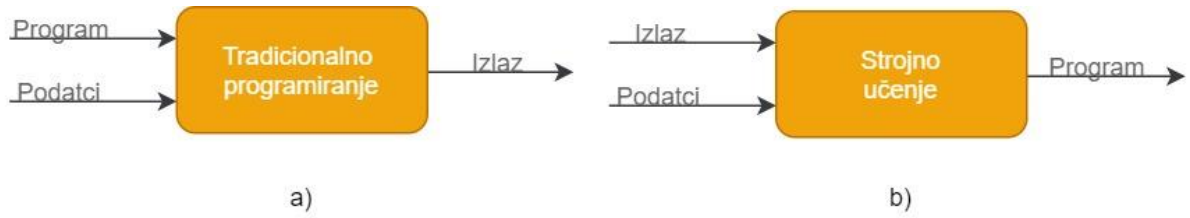
Predzadnje poglavlje je poglavlje rasprave u kojem su navedene trenutne mogućnosti i nedostatci web aplikacije te su navedena moguće proširenja same aplikacije.

Na kraju će iz svega izloženog biti izveden zaključak te u njemu opisana moguća primjena rada.

2. TEORIJSKA PODLOGA

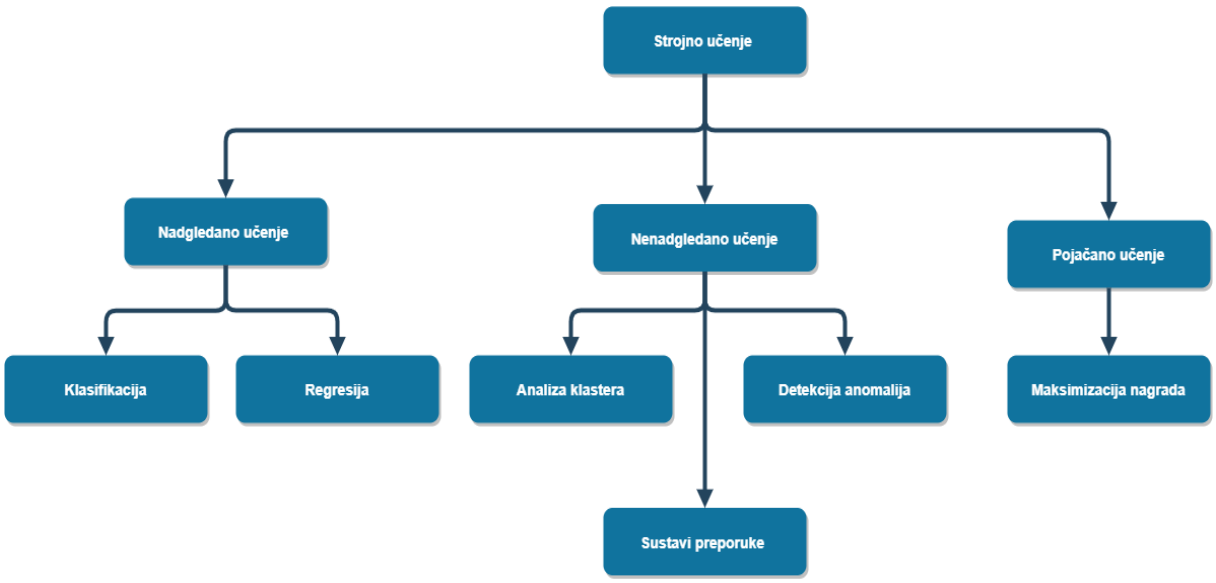
2.1. Strojno učenje

Svijet je pun podataka kao što su slike, glazba, videozapisi i tekstovi. Ti podatci nisu generirani samo od strane ljudi, već ih generiraju i računala, mobiteli i ostali uređaji. Strojno učenje se danas upotrebljava u mnogim primjenama kao što su označavanje ljudi i predmeta na slikama te sustavi preporuke iako to na prvi pogled nije očito. Strojno učenje je grana umjetne inteligencije koja daje računalima sposobnost da uče iz iskustva bez da su za to eksplicitno programirana. Ovu definiciju strojnog učenja osmislio je Arthur Samuel koji se smatra pionirom strojnog učenja. Kako bi se lakše ustanovila razlika između tradicionalnog programiranja i strojnog učenja dana je slika [Slika 2.1.]. Iz slike [Slika 2.1.] vidljivo je da se kod strojnog učenja računalu prezentiraju podatci i željeni izlaz za neki problem te nam ono daje program (skup pravila) koji daje rješenje za taj problem dok bi kod tradicionalnog programiranja ljudi morali sami izraditi taj program što nije uvijek jednostavno ili moguće za neke probleme.



Slika 2.1. Razlika između a) tradicionalnog programiranja i b) strojnog učenja

Strojno učenje prema [3] možemo podijeliti u tri glavne podskupine [Slika 2.2.].



Slika 2.2. Podjela strojnog učenja

Pojačano učenje (engl. *reinforcement learning*) je tip strojnog učenja kod kojeg je cilj sustava da uči iz stečenog iskustva. Ljudi programiraju algoritam tako da definiraju što bi morao biti konačni rezultat, a bez da zadaju najbolji način kako ga postići.

Nenadgledano učenje (engl. *unsupervised learning*) je tip strojnog učenja kod kojeg računalo dobije podatke bez labela (oznaka) i mora samo pronaći određene obrasce u podatcima na temelju određenih zajedničkih karakteristika.

Nadgledano učenje (engl. *supervised learning)* je tip strojnog učenja kod kojeg računalima dajemo podatke s labelama (oznakama) kako bi onda ona kasnije mogla sama dodjeljivati labele (oznake) novim podatcima bez ljudske intervencije. Računalu se inicijalno daje slika automobila ili nečeg drugog što je povezano s određenim zadatkom zatim mu se pomoću labela (oznaka) prikaže što je pojedini objekt i kako želimo da on bude interpretiran. Te na temelju tih primjera računalo može dodijeliti labele (oznake) kad ponovo prepozna te objekte [6].

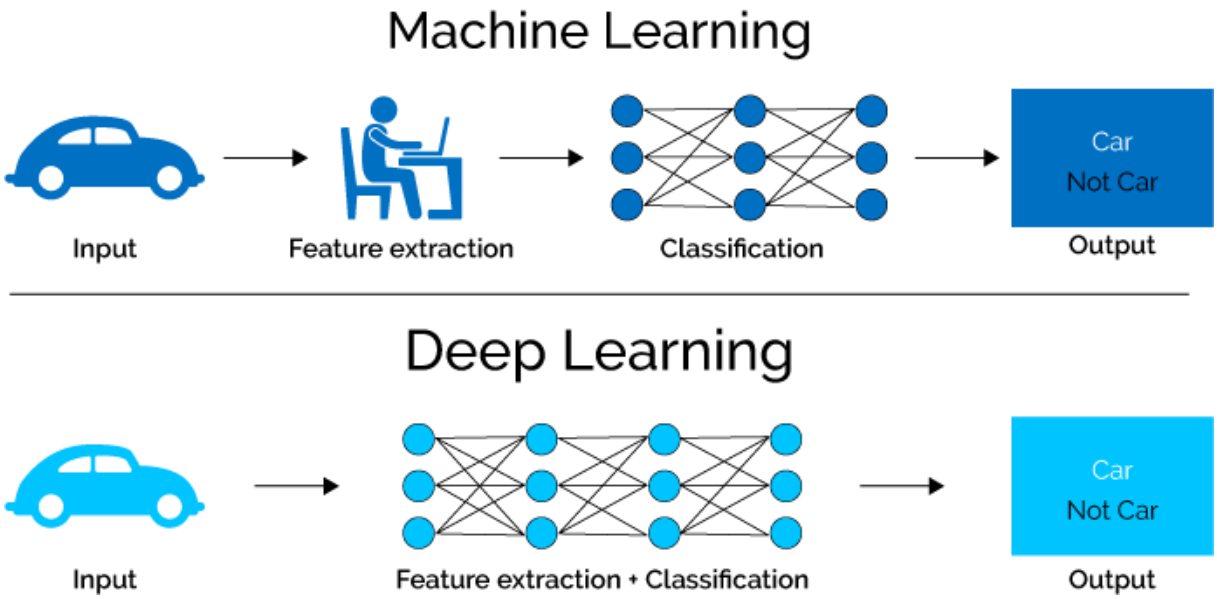
Uobičajeni postupak korištenja strojnog učenja može se opisati kao korištenje podataka kako bi se dobili odgovori na tražena pitanja . Korištenje podataka u tom postupku se naziva trening, a odgovaranje na tražena pitanja se naziva donošenje predviđanja. Trening se odnosi na korištenje podataka kako bi se stvorio model i prilagodili parametri tog modela. Zatim se taj model može koristiti za donošenje predviđanja na dosad neviđenim podatcima i na odgovaranje na ranije spomenuta pitanja. Kako se skuplja sve više i više podataka model se može kroz vrijeme poboljšavati i mogu se primijeniti novi efikasniji algoritmi predviđanja.

2.2. Duboko učenje

Duboko učenje je podskupina strojnog učenja koja oponaša rad ljudskog mozga u obradi podataka i stvaranju uzoraka koji se koriste pri donošenju odluka. Pojam duboko kod dubokog učenja najčešće se odnosi na broj slojeva u umjetnoj neuronskoj mreži zbog toga još jedan naziv koji se koristi za duboko učenje je duboka neuronska mreža [7].

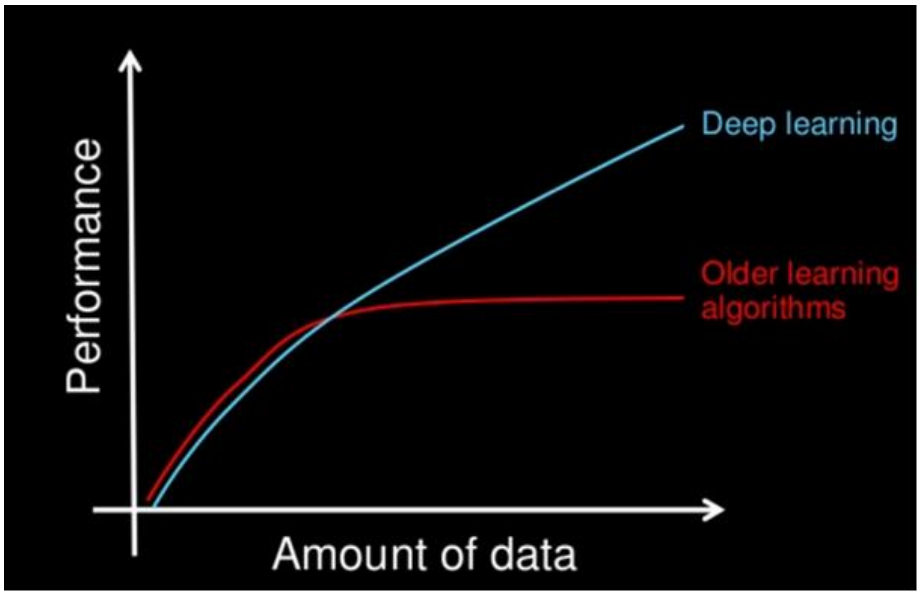
Često se čuje da ljudi miješaju pojmove strojnog učenja i dubokog učenja ili ih koriste naizmjenično zbog toga su u nastavku opisane njihove osnovne razlike. Kod strojeva programiranih strojnim učenjem čovjek mora ispravljati pogreške koje je napravio stroj tako da prilagodi konfiguraciju i time osigura da stroj ne ponavlja te iste pogreške. Međutim, model temeljen na dubokom učenju može samostalno ustvrditi da li je njegov zadatak uspješno dovršen ili ne, koristeći vlastitu umjetnu neuronsku mrežu koja mu omogućuje svojstva

samostalnog učenja i donošenja odluka. To predstavlja osnovnu razliku između spomenutih pojmova koja je još dodatno prikazana na slici [Slika 2.3.].



Slika 2.3. Usporedba dubokog učenja sa strojnim učenjem[8]

Pojmovi koji se često spominju kod dubokog učenja su skalabilnost i učenje značajki. Skalabilnost se odnosi na to da s konstruiranjem velikih neuronskih mreža i njihovim treniranjem na sve više i više podataka, njihova izvedba se nastavlja poboljšavati. Što onda duboko učenje općenito čini različitim od ostalih tehnika strojnog učenja koje dostižu plato u svojoj izvedbi [Slika 2.4.] Uz skalabilnost, još jedna često citirana prednost modela dubokog učenja je njegova sposobnost automatskog izdvajanja značajki iz sirovih podataka, koja se još naziva učenje značajki [7].



Slika 2.4. Usporedba skalabilnosti dubokog učenja i ostalih tehnika strojnog učenja[7]

Iako je duboko učenje prvi put bilo teoretizirano 1980-ih, dva su glavna razloga što je tek nedavno postalo korisno [7] :

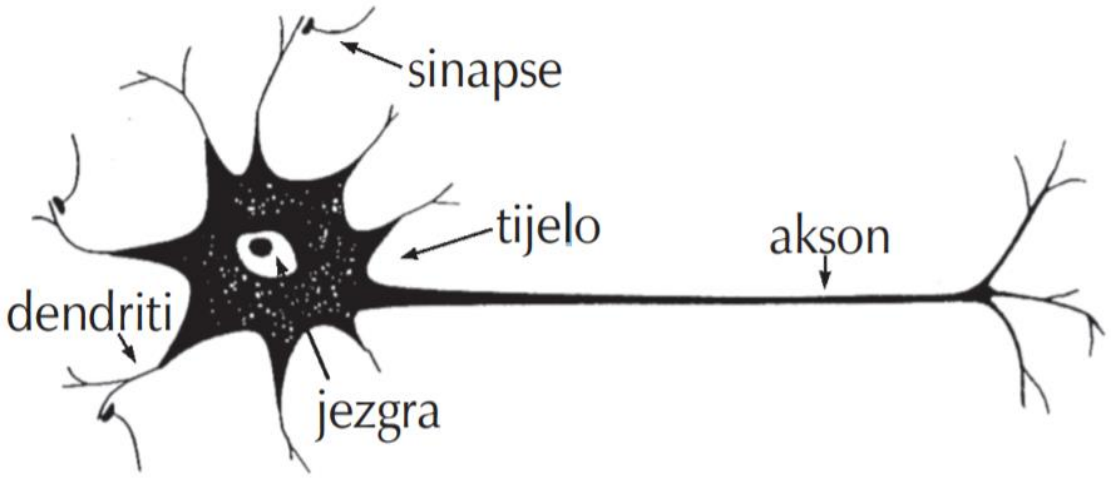
1. Duboko učenje zahtijeva značajnu računalnu snagu koja je tek u posljednjem desetljeću postala šire dostupna kao što su grafičko procesorske jedinice. Grafičko procesorske jedinice (engl. GPU-s) su specijalizirani elektronički sklopovi koji imaju paralelnu arhitekturu koja je učinkovita za duboko učenje.
2. Duboko učenje zahtijeva ogromne količine podataka s labelama (oznakama). Na primjer, za razvoj autonomnog vozila potrebno je nekoliko milijuna slika i tisuće sati videozapisa.

Neke od najpoznatijih primjena kod kojih se danas koristi duboko učenje su: obrada prirodnog jezika, analiza sentimenta, raspoznavanje govora, autonomna vozila i računalni vid.

2.3. Umjetne neuronske mreže

2.3.1. Biološki i umjetni neuron

Kako bi se lakše moglo razumjeti kako rade umjetne neuronske mreže prvo će se prikazati biološki neuron jer je ideja za umjetnu neuronsku mrežu nastala iz pokušaja modeliranja strukture i principa rada ljudskog mozga.



Slika 2.5. Prikaz biološkog neurona [9]

Svaki neuron sastoji se od tri dijela kao što je to prikazano na slici [Slika 2.5.], a to su [9]:

- **tijelo stanice-** koje sadrži jezgru ili nukleus s informacijama o nasljednim značajkama;
- **dendriti-** kraće niti oko stanice, koje prenose signale (impulse) s drugih neurona;
- **aksoni-** duge i tanke niti, koje se granaju u vlakna i prenose signal do drugih neurona

Sinapse predstavljaju funkcionalni spoj između aksona prethodnog neurona i dendrita sljedećeg neurona u kojima se oslobađaju neurotransmiteri koji služe za prijenos signala elektrokemijskom reakcijom.[9]

Umjetni neuron je dizajniran da oponaša osnovne funkcije biološkog neurona, ali kako funkcioniranje i struktura biološkog neurona nije egzaktno utvrđena on u stvarnosti predstavlja njegovo viđenje koje je temeljeno na određenim pretpostavkama. Analogija između rada i funkcija umjetnih i bioloških neurona dana je u tablici [Tablica 2.1.].

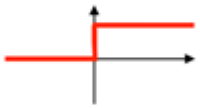
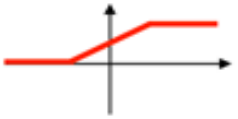
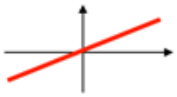
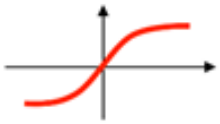
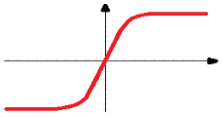
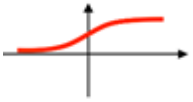
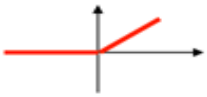
Tablica 2.1. Analogija između biološkog i umjetnog neurona [9]

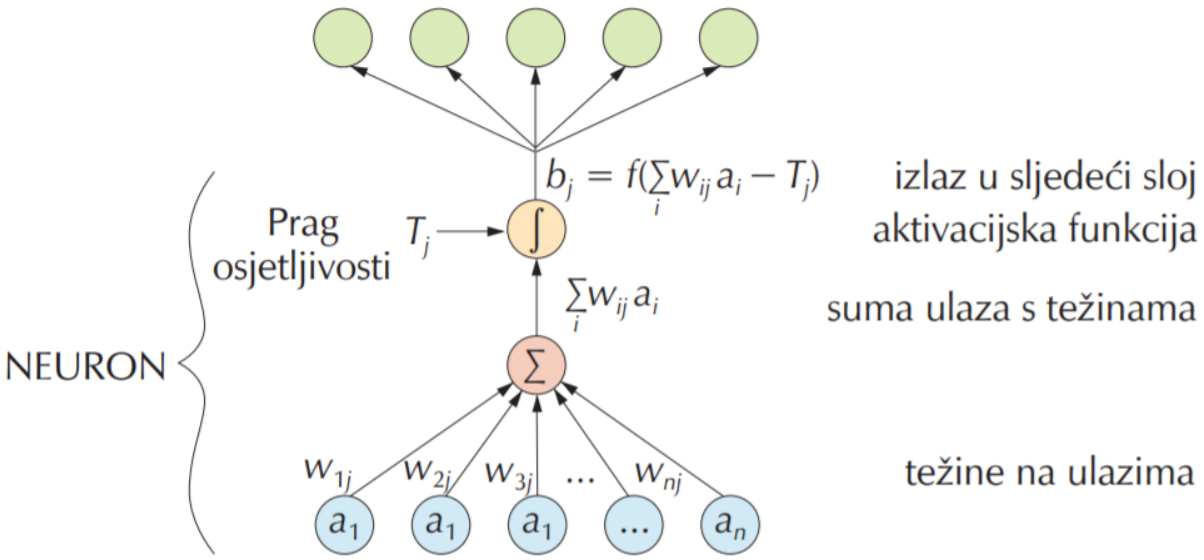
Biološki neuron	Umjetni neuron
Prima ulazni signal putem dendrita (sinaptičke veze)	Prima ulaze (i) koji su određeni težinskim koeficijentima (w)
Obrada signala u somi	Obrada ulaza, unutarnji prag – bias (b)
Pretvara obrađeni ulaz u izlaz putem aksona	Pretvara ulaze u izlaz (prijenosna funkcija)
Šalje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

2.3.2. Arhitektura umjetne neuronske mreže

Model neurona koji djeluje kao i biološki neuron, tj. signale obrađuje putem sinaptičke i somatske operacije naziva se perceptron [Slika 2.6.]. Svaka neuronska mreža sastoji se od ulaznog, skrivenog i izlaznog sloja. Ulazni sloj poprima vrijednosti ulaznih veličina. Sinaptička operacija predstavlja množenje svakog ulaznog signala s težinskim koeficijentom, wi . Tako otežani ulazni signali se zbrajaju, a njihov zbroj uspoređuje se s pragom osjetljivosti neurona, Tj (engl. threshold). Težinski faktori analogni su dendritima biološkog neurona. Skriveni sloj zbraja otežane ulaze pomoću neke funkcije sumiranja i tako da stvara vlastitu internu aktivaciju. Ako je zbroj otežanih signala veći od praga osjetljivosti neurona, nelinearna aktivacijska funkcija f generira izlazni signal neurona iznosa bj [9]. Primjeri najčešće korištenih aktivacijskih funkcija i njihovi matematički zapisi prikazani su u tablici [Tablica 2.2.]

Tablica 2.2. Vrste aktivacijskih funkcija

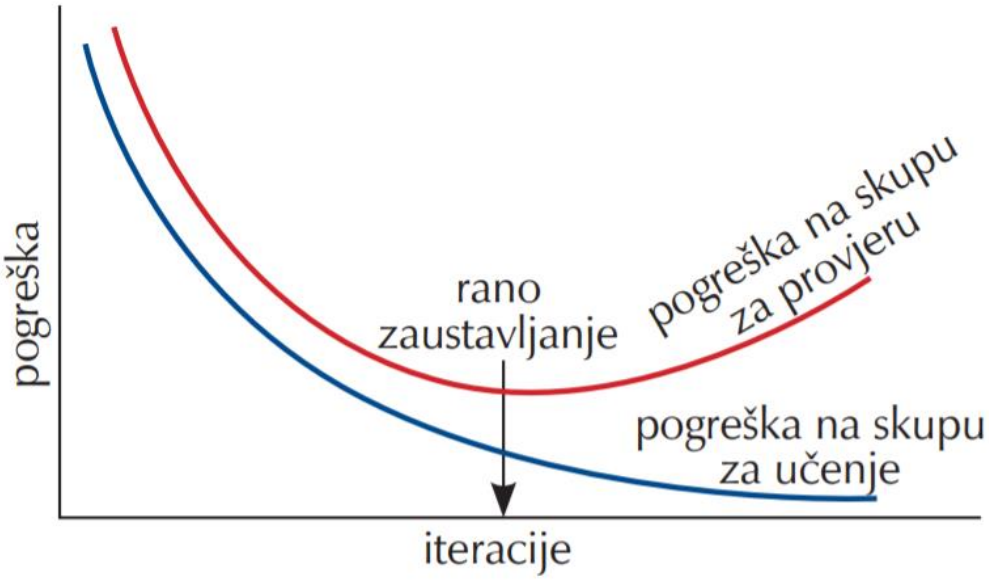
Aktivacijska funkcija	Jednadžba	Primjena	Dijagram
Odskočna (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 1, & z \geq 0, \end{cases}$	Varijante perceptrona (engl. perceptron variant)	
Po dijelovima linearna	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Metode potpornih vektora (engl. support vector machines)	
Linearna	$\phi(z) = z$	linearna regresija (engl. *linear regression)*	
Funkcija tangensa hiperboličnog	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Višeslojne neuronske mreže (engl. *multi-layer neural networks)*	
Softmax	$\phi(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$	Višeslojne neuronske mreže (engl. *multi-layer neural networks)*	
Sigmoidna	$\phi(z) = \frac{1}{1 + e^{-z}}$	Višeslojne neuronske mreže (engl. *multi-layer neural networks)*	
ReLU	$\phi(z) = max(0, z)$	Višeslojne neuronske mreže (engl. *multi-layer neural networks)*	



Slika 2.6. Model umjetnog neurona [9]

2.3.3. *Postupak učenja umjetnih neuronskih mreža*

Učenje (engl. *learning, training*) označava iterativni postupak prilagođavanja (optimiziranja) vrijednosti težinskih faktora na temelju pogreške između izračunate vrijednosti modelom i stvarne vrijednosti izmjerene veličine. Učenje, odnosno podešavanje težinskih faktora odvija se prema jednom od pravila učenja kao što je tzv. pravilo povratnog prostiranja (engl. *back propagation*). Danas na raspolaganju su brojni algoritmi i strukture učenja. Tijekom jednog prolaska informacije kroz neuronsku mrežu generira se vrijednost koja se zatim uspoređuje sa stvarnom vrijednošću. Na temelju te razlike, ispravljaju se težinski faktori. Tim ispravljanjem neuronska mreža uči predviđati stvarne vrijednosti i razlika stvarnih i predviđenih vrijednosti izlaznih veličina se smanjuje. Robusnost (generalizacija) neuronske mreže i njezina kvaliteta mogu se vidjeti iz kriterija pogreške. Provjerom mreže na skupu za provjeru (nov skup podataka), sprječava se "pretreniranje" (engl. *overfitting*) mreže prilikom učenja. Pretreniranje je pojava kod koje mreža s visokom točnošću opisuje ponašanje podataka na skupu podataka na kojem je trenirana, dok na podatcima izvan tog skupa daje slabije rezultate. Na slici [Slika 2.7.] prikazana je ovisnost pogreške učenja (engl. *training error*) i pogreške provjere (engl. *testing error*) o broju iteracija učenja. Iz slike je vidljivo da se obje pogreške smanjuju do točke minimalne vrijednosti pogreške provjere , a nakon te točke pogreška učenja nastavlja se smanjivati, ali pogreška provjere počinje rasti, što indicira pretreniranost mreže. Sa svrhom sprječavanja ove pojave kod uočavanja rasta pogreške provjere učenje neuronskih mreža se zaustavlja [9].



Slika 2.7. Prikaz učenja neuronske mreže s tehnikom ranog zaustavljanja [9]

2.3.4. Podjela umjetnih neuronskih mreža

Zbog velikog broja umjetnih neuronskih mreža postoje razni načini kako se mogu podijeliti. Pa prema tome umjetne neuronske mreže znaju biti podijeljene ovisno o njihovoj dubini, odnosno koliko imaju slojeva između ulaza i izlaza mreže koji se još nazivaju skriveni slojevi. Mogu biti još podijeljene prema broju skrivenih čvorova koje model neuronske mreže sadržava ili po broju ulaza i izlaza koji svaki od čvorova ima. Najjednostavnija inačica umjetne neuronske mreže je unaprijedna (engl. *feedforward*) neuronska mreža. Ovaj tip umjetne neuronske mreže karakterizira prolazak informacija ravno od ulaza prema izlazima. Može i ne mora imati skrivene slojeve. Kompleksnije su povratne (engl. *recurrent*) neuronske mreže. Koriste se kod dubokog učenja i karakterizira ih to da se obrađeni signal vraća nazad na početak modela i smatra se da tako model uči. Još jedan tip umjetnih neuronskih mreža koji se koristi kod dubokog učenja su konvolucijske neuronske mreže. One su korištene u ovom radu pa će u nastavku biti opisane malo detaljnije.

2.3.5. Primjena umjetnih neuronskih mreža

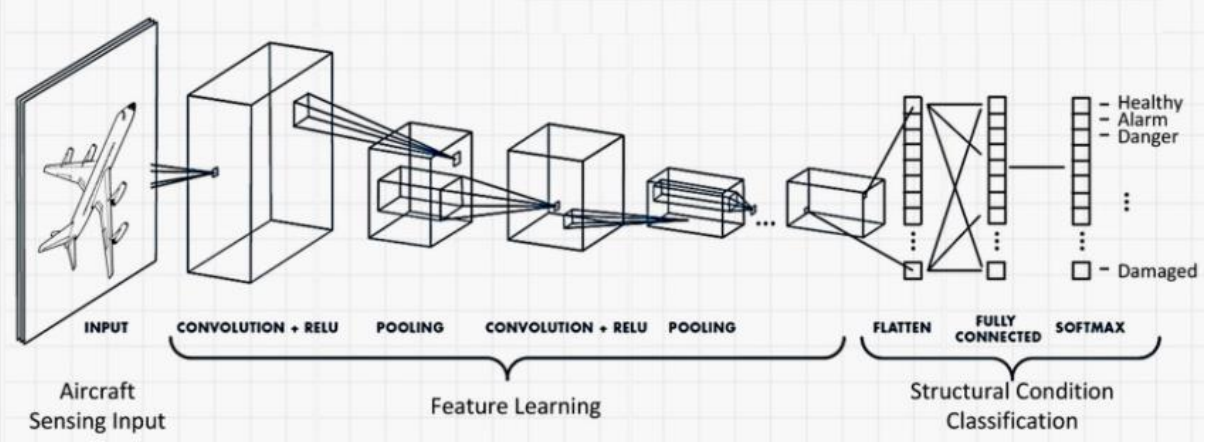
U metodama strojnog učenja neuronske mreže se često primjenjuju za klasifikaciju: prepoznavanje slika, govora, prevođenje, analiza društvenih mreža, inteligentno internetsko pretraživanje i sl. Još neki primjeri njihove upotrebe su autonomna vozila, predviđanje dionica na burzama, procjena rizika koda izdavanja kredita te na kraju procjene prodaje. Važna svojstva zašto se umjetne neuronske mreže koriste i zašto će njihova upotreba nastaviti rasti su ta da rezultati postaju bolji s korištenjem većeg skupa podataka te s upotrebom većih modela (više slojeva neuronskih mreža).

2.4. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (ConvNets ili CNN-s) su neuronske mreže koje se koriste za duboko učenje. Njihova glavna karakteristika je da uče direktno iz podataka, eliminirajući potrebu za ručnim izdvajanjem značajki.

Konvolucijske neuronske mreže se kao i sve ostale neuronske mreže sastoje od ulaznog sloja, skrivenih slojeva i izlaznog sloja. Ono po čemu se konvolucijska neuronska mreža razlikuje od drugih je to da sadrži više različitih skrivenih slojeva. Uobičajeno se koriste četiri vrste skrivenih slojeva: konvolucijski sloj (engl. *convolutional layer*), sloj sažimanja (engl. *pooling layer*), ReLU sloj(engl. *ReLU layer*) i potpuno povezani sloj (engl. *fully connected layer*).

Prikaz uobičajene konvolucijske neuronske mreže dan je na slici [Slika 2.8.].



Slika 2.8. Prikaz uobičajene konvolucijske neuronske mreže [10]

Konvolucijski slojevi (engl. convolutional layers) su slojevi koji daju konvolucijskim neuronskim mrežama njihovo ime. U konvolucijskim slojevima čvorovi primjenjuju svoje filtere na ulaznu sliku. Umjesto da odjednom pogleda cijelu sliku, on je skenira u preklapajućim blokovima piksela. Cilj konvolucijskog sloja je da identificira i izvuče značajke iz slike. Jednostavno rečeno, filtri pridodaju vrijednost pikselima koji im odgovaraju. Što im više odgovaraju to viša je vrijednost koju im filtri pridaju. Tako neuronska mreža konvertira (prepliće) podatke filtra s ulaznom slikom da bi se stvorila „mapa značajki" (engl. *feature map)*.

Slojevi sažimanja (engl. *pooling layers*) su skriveni slojevi koji slijede nakon konvolucijskih slojeva. U sloju sažimanja, sve vrijednosti piksela u pojedinoj mapi značajki zajedno su „sažete". To smanjuje razlučivost mapa značajki iz konvolucijskih slojeva. Manji prikaz slike znači manje parametara i manje proračuna. Sažimanje je korak koji omogućuje otkrivanje objekata bez obzira gdje se na slici nalaze. Drugim riječima, slojevi sažimanja daju fleksibilnost konvolucijskim neuronskim mrežama. Oni zaustavljaju računalo da ne stavlja prevelike težine na određene značajke koje se nalaze na specifičnim mjestima.

ReLU slojevi (engl. *ReLU layers)* su skriveni slojevi koji slijede nakon slojeva sažimanja. „ReLU" označava „ispravljenu linearnu jedinicu" (engl. *rectified linear unit*). Svrha ReLU slojeva je da uvedu nelinearnost gdje linearnost označava odvijanje stvari u točno određenom redu. Oni uvode nelinearnost tako da zamijene negativne vrijednosti piksela s nulama. Jednostavno rečeno ReLU slojevi u osnovi omogućuju računalu da bolje obrađuje složene podatke kao što su slike, a one su nelinearne.

Potpuno povezani slojevi (engl. *fully connected layers*) dolaze na kraju skrivenih slojeva konvolucijske neuronske mreže. U konvolucijskom sloju čvorovi primaju ili dijele informacije samo s dijelom prethodnog sloja. U potpuno povezanom sloju svaki čvor prima izlaz od svakog čvora prethodnog sloja. Potpuno povezani slojevi slični su onima koji se nalaze kod skrivenih

slojeva uobičajene umjetne neuronske mreže. U ovim slojevima se sve značajke izvučene od strane konvolucijske neuronske mreže kombiniraju. To znači da računalo vidi cijelu sliku što može pomoći u ostvarivanju točnih rezultata.[11]

Kod dubokog učenja najčešće se upotrebljavaju konvolucijske neuronske mreže zbog sljedećih razloga:

- eliminiraju potrebu za ručnim (manualnim) izvlačenjem značajki
- rezultiraju visokom točnošću kod prepoznavanja
- mogu biti prenamijenjene za nove zadatke prepoznavanja

Konvolucijske neuronske mreže uobičajeno se koriste za zadatke koji zahtijevaju računalni vid i prepoznavanje objekata kao što su: primjene za prepoznavanje lica, uličnih znakova, tumora i kod autonomnih vozila.

2.5. Računalni vid

Računalni vid je područje umjetne inteligencije koje se bavi omogućavanjem računala da prepoznaju i procesiraju razne objekte sa slika i videa na način kako je to svojstveno ljudima.

Problem računalnog vida izgleda kao jednostavan problem jer ljudi taj isti zadatak rješavaju trivijalno. Gledajući sliku ljudi lako mogu prepoznati i prebrojati osobe na slici, pa čak i odrediti njihova emocionalna stanja. Ipak računalni vid je još uvijek u velikoj mjeri neriješen problem. Razlozi za to su da je ljudsko shvaćanje rada vlastitog vizualnog sustava ograničeno i velika kompleksnost vizualne percepcije u dinamičkom i konstantno mijenjajućem svijetu.

Cilj računalnog vida je preslikavanje mogućnosti ljudskog vida pomoću digitalnih slika koristeći tri glavne procesne komponente u sljedećem redoslijedu [12]:

1. Pribavljanje slike (engl. *Image acquisition)*
2. Obrada slike (engl. *Image processing)*
3. Analiza i razumijevanje slike (engl. *Image analysis and understanding)*

Razlika između računalnog vida i obrade slike je u tome da je cilj obrade slike stvaranje nove slike na temelju postojeće slike za razliku od računalnog vida kod kojeg je cilj razumijevanje sadržaja slike. Obrada slike se može koristiti za procesiranje ulaznih podataka koji se upotrebljavaju kod računalnog vida.

Rad računalnog vida podsjeća na sastavljanje slagalice. Računala sastavljaju slike na isti način kako bi se spajale slagalice. Kod sastavljanja slagalice prisutni su svi dijelovi i moraju se složiti

u sliku. Na sličan način funkcioniraju umjetne neuronske mreže kod računalnog vida. One razlikuju različite dijelove slike, prepoznaju rubove te zatim modeliraju potkomponente. Koristeći filtriranje i nizom radnji kroz duboke slojeve neuronske mreže računala mogu sastaviti sve dijelove slike u cjelinu, slično kao što bi se uradilo i kod slagalice. Za razliku od slagalice kod koje je konačna slika prikazana na vrhu kutije u kojoj su bile slagalice računalo ne dobije konačnu sliku nego je ono trenirano na stotinama ili tisućama sličnih slika koje sadrže određene objekte kako bi moglo prepoznati te objekte [13].

Bitno svojstvo koje ljudi posjeduju je sposobnost donošenje odluka i pridjeljivanje smisla onome što vide u stvarnom svijetu. Omogućivanje računalima i strojevima određenu razinu ovog vizualnog razumijevanja postiže se kroz sljedeće postupke [12]:

- **Klasifikacija objekata** – koja uključuje treniranje modela na skupu podataka određenih objekata i zatim model klasificira nove objekte u jednu ili više kategorija određenih treningom.
- **Identifikacija objekata** – je postupak u kojem će model prepoznati specifične instance objekata

Kao što je ranije u ovom radu spomenuto da su ogromna količina vizualnih podataka koja se dnevno generira, razvoj novih algoritama i veća dostupnost potrebne računalne snage, u posljednje vrijeme doveli do značajnog napretka umjetne inteligencije, a to isto vrijedi i za razvoj računalnog vida. Taj napredak doveo je do upotrebe računalnog vida kod mnogih primjena od kojih su najpoznatije: kontrola kvalitete izrade proizvoda, inspekcija strojeva, automatizacija maloprodaje, prepoznavanje otiska prsta, medicinski računalni vid, navođenje raketnih sustava i virtualna stvarnost. Većina od tih nabrojanih aplikacija ne bi mogla biti moguća bez primjene dubokog učenja.

Izazovi koji se danas javljaju s računalnim vidom su većinom povezani s privatnošću i lažnim sadržajem. Izazovi koji se vežu uz privatnost su nadzor i praćenje koji koriste računalni vid i tako ugrožavaju privatnost građana u mnogim zemljama što je dovelo do ograničavanja njihove upotrebe i strožih zakona za njihovo korištenje. Dok će izazovi s lažnim sadržajem sve više postajati problem jer će sam razvoj tehnologije dovesti do sve težeg uočavanja razlika između nečega što će biti lažno generirano i nečega što se zapravo dogodilo.

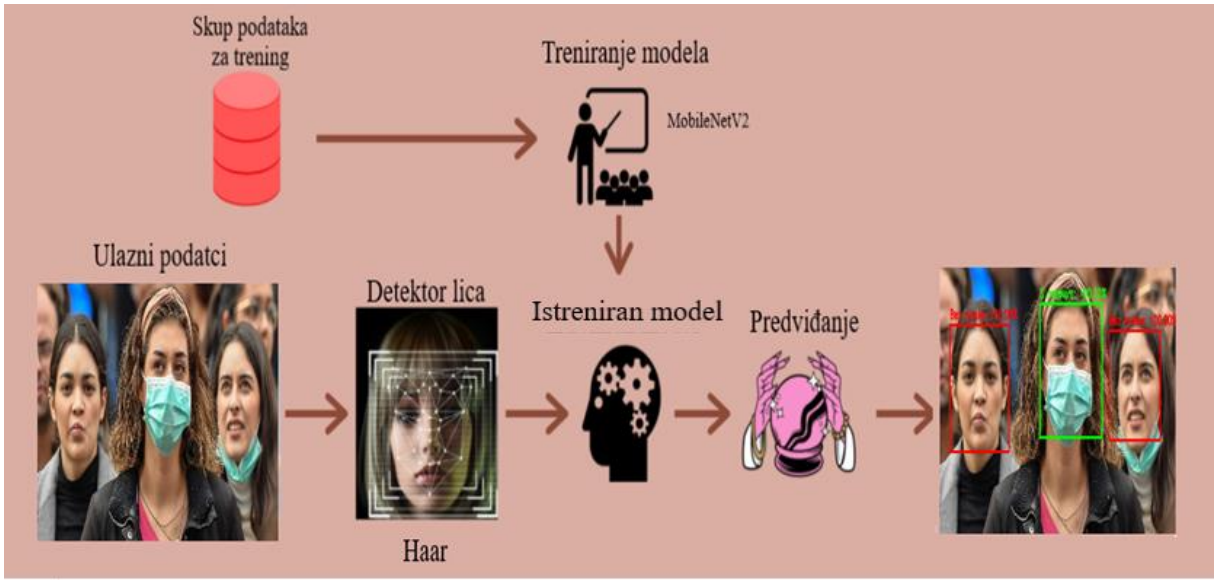
3. IZRADA WEB APLIKACIJE MASKVALIDATOR

3.1. Konceptualna razrada web aplikacije MaskValidator za detekciju maske na licu

Kako bi se mogla implementirati ova aplikacija je podijeljen na dva osnovna koraka svaki sa svojim odgovarajućim dijelovima [Slika 3.1.]:

1. **Treniranje modela-** ovdje se odvija učitavanje odgovarajućeg skupa podataka iz poglavlja [3.5.] koji će se koristiti za treniranje, zatim izgradnja modela na temelju korištenih Python biblioteka iz poglavlja [3.4.] i na kraju spremanje tog modela za danju upotrebu.
2. **Izrada grafičko sučelja web aplikacije MaskValidator i njenih funkcionalnosti-** nakon što je istreniran model dalje se prijelazi na izradu grafičkog sučelja web aplikacije te razvoj funkcionalnosti predviđanja.

Svaki od ovih koraka i njegovih dijelova bit će detaljnije prikazan u nastavku ovog rada.



Slika 3.1. Grafički prikaz odvijanja osnovnih koraka i potkoraka programske aplikacije

3.2. Programski jezik Python

Python je interpreterski, interaktivni, objektno orijentirani programski jezik kojeg je osmislio i napravio Guido van Rossum 1991. godine. Interpreterski programski jezici su jezici kod kojih se izvorni kod izvršava direktno uz pomoć interpretera, tj. kod ovakvih tipova programskih jezika nema potrebe za kompajliranjem prije izvršavanja, tj. prevođenjem u izvršni oblik.

Sadrži module, iznimke, dinamičko povezivanje, dinamičke tipove podataka visoke razine i klase. Podržava više stilova programiranja izvan objektno orijentiranog programiranja, kao što

su proceduralno i funkcionalno programiranje. Python kombinira izvanrednu snagu s vrlo jasnom sintaksom. Ima sučelje za mnoge sistemske pozive i biblioteke, a proširiv je u C ili C++. Također se može koristiti kao produženi jezik za aplikacije kojima je potrebno programibilno sučelje. Python je prenosiv: radi na mnogim verzijama Unix uključujući Linux, MacOS i Windows [14].

Python je odabran za ovaj rad jer je veoma rasprostranjen samim time ima veliku zajednicu, zbog toga što je besplatan i jednostavan, te zbog opsežne količine modula. Programski dio rada je rađen u Pythonu izdanja 3.7 pošto ima besplatne biblioteke potrebne za izradu zadatka. Još jedan razlog zašto je odabran Python je taj da je on trenutno najpopularniji programski jezik za strojno učenje. Razlozi za korištenje programskog jezika Python kod primjene strojnog i dubokog učenja su sljedeći: efikasna i jasna sintaksa, jednostavna integracija s drugim programskim jezicima te najvažnije opsežna podrška za biblioteke otvorenog koda tako da se ne moraju izrađivati vlastiti algoritmi i umjetne neuronske mreže iz temelja.

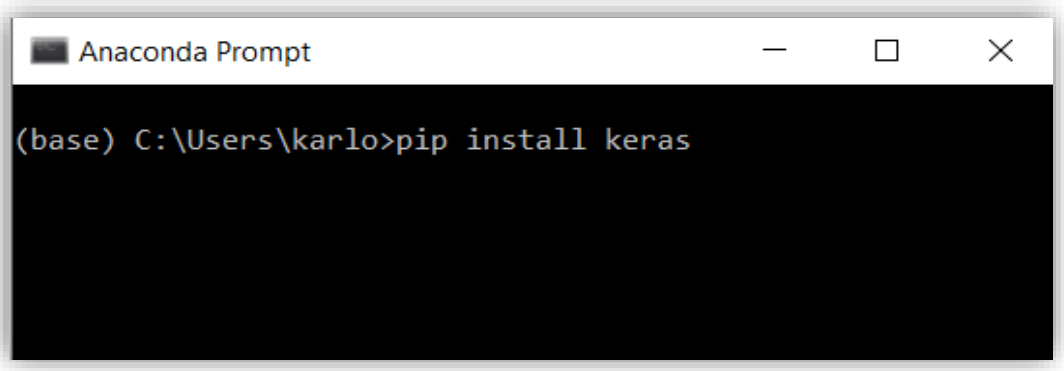
3.3. Jupyter notebook

U ovom radu Python kôd pisan je unutar Jupyter Notebooka. Jupyter Notebook je besplatna interaktivna web-aplikacija otvorenog koda koja omogućuje stvaranje i razmjenu dokumenata koji sadrže računalni kôd koji se može izvoditi uživo te bogate tekstualne elemente (odlomke, jednadžbe, slike, poveznice itd.). Tako izrađeni programi su istovremeno izvršni dokumenti koji se mogu pokrenuti za analizu podataka i čitljivi dokumenti koji sadrže opise analiza i rezultate. Njezine glavne značajke su pisanje , uređivanje i izvođenje koda u internet pregledniku, zatim izvođenje koda odvojeno po dijelovima, mogućnost korištenja LaTeXa i bogati izbor medija za vizualnu reprezentaciju [15].

3.4. Korištene Python biblioteke

Biblioteka je zbirka prethodno kombiniranih kodova koja se može koristiti iterativno za smanjenje vremena potrebnog za programiranje. Biblioteke su posebno korisne kod pristupanja prethodno napisanom kodu koji se često koristi, umjesto da se piše svaki put ispočetka. Ovo prethodno spomenuto je posebno korisno kod strojnog i dubokog učenja da se ne moraju svaki put iz temelja programirati umjetne neuronske mreže ili drugi složeni algoritmi koji se primjenjuju u ovim područjima.

Sve biblioteke u Pythonu se instaliraju na jednak način, a to je tako da se u terminal upiše 'pip install [potrebni paket]' kao što je prikazano na slici [Slika 3.2.].



Slika 3.2. Pip install

3.4.1. TensorFlow

TensorFlow je Python biblioteka otvorenog koda koju je razvio Google. TensorFlow je dio gotovo svake Googleove aplikacije za strojno učenje. Značajke koje opisuju TensorFlow su:

1. Laka vizualizacija svih dijelova računskih grafova
2. Fleksibilnost
3. Lako provođenje treninga
4. Paralelni trening neuronskih mreža
5. Velika zajednica
6. Otvoreni kod

TensorFlow radi kao biblioteka za pisanje novih algoritama koji uključuju veliki broj tenzorskih operacija, pošto se neuronske mreže mogu lako izraziti kao računski grafovi one se mogu implementirati pomoću TensorFlowa kao niz operacija na tenzorima. Tenzori su N-dimenzionalne matrice koje predstavljaju podatke [16].

3.4.2. Keras

Keras je jedna od najkorištenijih biblioteka za strojno učenje u Pythonu, a interno koristi Theano ili TenserFlow. Značajke koje opisuju Keras su:

1. Gladak rad na grafičkom i običnom procesoru
2. Podržava gotovo sve modele neuronskih mreža i ujedno se ti modeli mogu kombinirati tako da tvore kompleksne modele

3. Pošto je modularne naravi iznimno je izražajan, fleksibilan i pogodan za inovativna istraživanja
4. U potpunosti je baziran na Pythonu što ga čini jednostavnim za otklanjanje pogrešaka

Keras također pruža neke od najboljih usluga za kompajliranje modela, obradu skupa podataka i lakši način za izražavanje neuronskih mreža [16].

3.4.3. Matplotlib

Matplotlib je višeplatformska biblioteka za vizualizaciju podatka temeljna na NumPy tipu podatka polja (engl. *array)* i izgrađena tako da radi u okviru šire biblioteke SciPy. Jedno od najvažnijih svojstava Matplotliba je to da dobro radi u nizu operativnih sustava i neovisno o izlaznom formatu. To je dovelo do velikog broja korisnika i samim time velikim interesom za njegovim razvojem od strane razvojnih inženjera [17].

3.4.4. NumPy

NumPy je Python biblioteka koja se koristi za rad s poljima (engl. *array*) koju ostale biblioteke kao što je TensorFlow koriste interno za provedbu niz operacija na tenzorima. Značajke koje opisuju NumPy su:

1. Interaktivnost i jednostavna upotreba
2. Jednostavna implementacija kompleksnih matematičkih izraza i operacija
3. Intuitivnost
4. Otvoreni kod

Uobičajene upotrebe ove biblioteke su kod izražavanja slika, zvučnih valova i drugih binarnih tokova kao polja (engl. *array*) realnih brojeva u N-dimenzionalnom prostoru [16].

3.4.5. OpenCV

OpenCV je vizijska biblioteka računalnog vida i strojnog učenja otvorenoga koda. Biblioteka je izvorno napisana u programskim jezicima C i C++ , ali može se izvoditi na svim platformama. Biblioteka OpenCV je izrađena kako bi pružila zajedničku infrastrukturu za primjene koncentrirane na računalni vid i da ubrza upotrebu percepcije strojeva u komercijalnim proizvodima. Biblioteka se sastoji od više od 2500 optimiziranih algoritama, što obuhvaća sveobuhvatan set klasičnih i najnovijih algoritama za računalni vid i strojno učenje. Ti algoritmi mogu se koristiti za otkrivanje i prepoznavanje lica, identifikaciju objekata, klasificiranje ljudskih kretnji u videozapisima, praćenje objekata u pokretu , izvlačenje 3D modela predmeta, povezivanje niza slika kao bi se dobila slika vrlo visoke razlučivosti cijele scene i mnoge druge

primjene. Ima vrlo veliku zajednicu korisnika koja se sastoji od istraživača i hobista, a njezina upotreba široko je zastupljena i u poslovnom svijetu gdje se uglavnom primjenjuje kod vizijskih primjena u realnom vremenu [18].

3.4.6. Sklearn ili Scikit-learn

Sklearn ili Scikit-learn je Python biblioteka otvorenog koda koja je povezana s Python bibliotekama NumPy i SciPy. Smatra se jednom od najboljih biblioteka za rad sa složenim podatcima. Značajke koje opisuju Sklearn su:

1. Krosvalidacija
2. Algoritmi nenadgledanog učenja
3. Izvlačenje značajki

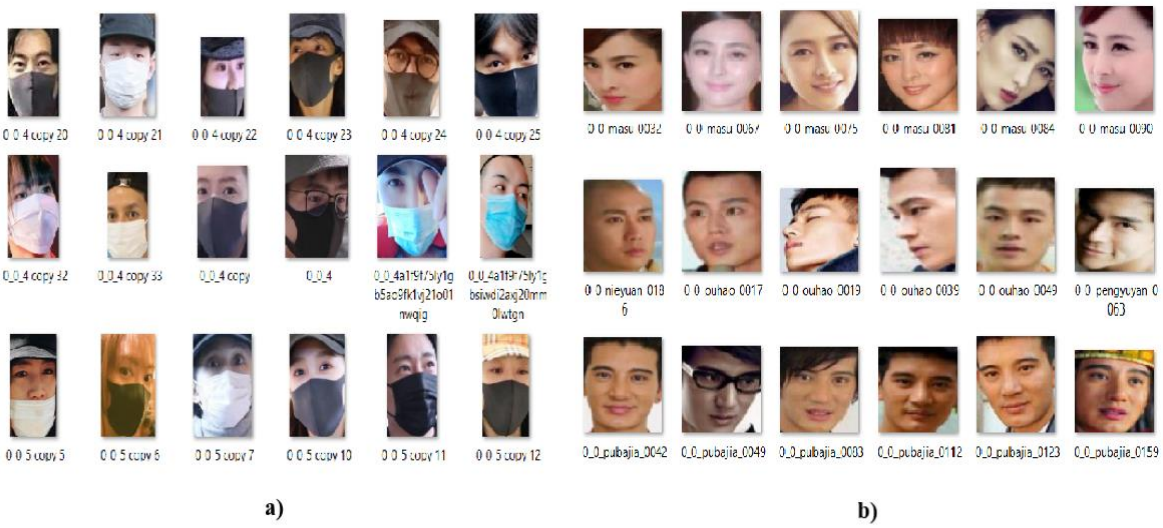
Sklearn sadrži brojne algoritme za primjenu standardnih zadataka strojnog učenja i rudarenje podataka kao što su: smanjenje dimenzionalnosti, klasifikacija, regresija, klasterizacija i odabir modela [16].

3.5. Korišteni skup (set) podataka

U ovom radu će se koristi prikladni skup (set) podatka (engl. *dataset)* [19]. Skup podataka se sastoji od 3833 slika podijeljenih u dvije klase [Slika 3.3.]:

- with_mask : 1915 slika
- without_mask : 1918 slika

Sadržane slike su slike ljudi koji nose i ne nose maske na licu. Slike su sastavljene iz skupova podatka koji se nalaze na Kaggle internet stranici koja je jedan od najpopularnijih stranica s raznim skupovima podataka otvorenog pristupa.



Slika 3.3. Prikaz nekoliko slika iz skupa podatka a) with_mask i b) without_mask

3.6. Prvi osnovni korak rada: Treniranje modela za web aplikaciju MaskValidator

3.6.1. Učitavanje potrebnih biblioteka i modula

Na početku svakog projekta pa tako i ovog moraju se učitati potrebne biblioteke i moduli što je i prikazano na slici [Slika 3.4.].

```
#učitavanje potrebnih biblioteka i modula
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
```

Slika 3.4. Prikaz učitavanja potrebnih biblioteka i modula

Gdje će učitavanje biblioteke tensorflow.keras omogućiti:

- umjetno povećavanje podataka (engl. *data augmentation*)

- učitavanje potrebne MobilnetV2 [20] arhitekture konvolucijskih neuronskih mreža čiji je model prethodno istreniran s težinama ImageNET baze podataka i uz to je model proračunski efikasan i samim time pogodan za primjenu kod ugradbenih (engl. *embedded)* sustava kao što je Raspberry Pi
- izgradnju potpuno povezanih slojeva
- pretprocesiranje i učitavanje slika

Biblioteka sklearn (scikit-learn) će poslužiti za binarizaciju oznaka klasa, podjelu korištenog skupa podataka i prikaz klasifikacijskog izvješća.

Module os omogućuje funkcionalnosti za interakciju s operativnim sustavom i on dolazi u standardnim Python modulima pa ga nije potrebno dodatno instalirati.

3.6.2. Učitavanje i pridjeljivanje skupa podataka

U ovom dijelu su sve slike pročitane i pridjeljenje odgovarajućim listama. Ovdje se izvuku putevi (engl. *paths)* gdje se nalaze slike i na temelju toga im se dodijeli oznaka . Kako je ranije rečeno [3.5.] skup podataka se sastoji od dvije mape with_masks i without_masks to je tako da bi se lako mogle pridijeliti oznake na temelju izvučenog imena mape. Također još se obrađuju slike i mijenja im se veličina na dimenzije 224x224.

Ovaj postupak prikazan je na slici [Slika 3.5.]

```
# dohvaćanje skupa podataka iz direktorija na računalu gdje je spremeljen
imagePaths = list(paths.list_images('Desktop\zavrsni_python\dataset'))
# inicijalizacija liste podataka (slika) i klasa tih slika
print("[INFO] loading images...")
data = []
labels = []
# prolazak kroz putove (engl. paths ) gdje se nalaze slike u petlji
for imagePath in imagePaths:
    # izvlačenje oznaka klasa iz imena mapa
    label = imagePath.split(os.path.sep)[-2]
    # učitavanje ulazne slike (224x224) i njezino procesiranje
    image = load_img(imagePath, target_size=(224, 224))
    # pretvorba u array format
    image = img_to_array(image)
    # skaliranje intenzita piksela u ulaznoj slici u rasponu od [-1,1]
    image = preprocess_input(image)
    # ažuriranje lista podataka i oznaka svaku za sebe
    data.append(image)
    labels.append(label)
# pretvorba podataka i oznaka u tip podatka NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)
```

Slika 3.5. Prikaz učitavanja i pridjeljivanja skupa podataka

3.6.3. Učitavanje i prilagodba prethodno istreniranog MobileNetV2 modela

U ovom dijelu koda učitan je prethodno istrenirani model MobileNetV2 [20] i prilagođena njegova struktura za ovaj problem prema [21]. Prilagodba se sastoji od isključivanja zadnjeg potpunog povezanog sloja i dodavanja nekoliko novih slojeva. Pošto u ovom problemu imamo samo dva izlaz u zadnji sloj su dodana samo dva čvora (engl. *nodes)* .

Ovaj postupak [Slika 3.6.] se još naziva prijenosno učenje (engl. *transfer learning*) gdje se iskoristi prethodno istreniran model za novi problem.

```
# učitvanje MoblieNetV2 arhitekture i osiguravanje da gornji(zadnji) potpuno povezani sloj nije uključen
baseModel = MobileNetV2(weights="imagenet", include_top=False,
                        input_shape=(224, 224, 3))
# konstruiranje head modela  koji će biti stavljeni kao gornji(zadnji) sloj osnovnog (baznog) modela
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
# pretvorba 3D mape značajki u 1D vektor značajki
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

# postavljanje head poptpuno povezani model na osnovni(bazni) model
# on će postati onaj model koji će se zapravo trenirati
model = Model(inputs=baseModel.input, outputs=headModel)
# prolazak kroz sve slojeve u osnovnom (baznom) modelu u petlji
# i zamrzivanje(engl. freeze) tih slojeva kako oni tj. njihove težine
#ne bi bili ažurirani tijekom prvog procesa treninga tj. povratnog prostiranja
for layer in baseModel.layers:
    layer.trainable = False
```

Slika 3.6. Prikaz učitavanja i prilagodbe prethodno istreniranog modela

3.6.4. Podjela i povećavanje podataka

Prvo je izvršeno one hot kodiranje. To je način kako se kodiraju kategorični podatci i ono se izvodi tako da se stvori polje (engl. *array*) s brojevima elemenata koji odgovaraju brojevima kategorija (klasa) koji su sadržani u problemu (u ovom zadatku 2). Sve te vrijednosti su 0, osim jedne koja je 1, a indeks di je ta jedinica odgovara vrijednosti te kategorije u ovom slučaju [1 0] [0 1] . Ovo kodiranje se provodi kao bi model tretirao sve ulaze jednako. Nakon toga se provodi umjetno povećavanje podataka (engl. *data augmentation*) koje značajno povećava raznolikost podataka koji su dostupni za treniranje modela, a bez da se stvarno skupljaju novi podatci. Neke tehnike koje se koriste kod toga su obrezivanje (engl. *cropping)*, rotiranje, smicanje (engl. *shearing)* i vodoravno okretanje (engl. *flipping)* i one se uobičajeno koriste za treniranje velikih neuronskih mreža. Sve ovo je prikazano na slici [Slika 3.7.]

```
# izvođenje one-hot kodiranja na oznakama
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
# podjela podataka u dijelove za trening i testiranje gdje se 80% podataka
# koristi za trening, a preostalih 20% za testiranje (provjeru, validaciju)
(trainX, testX, trainY, testY) = train_test_split(data, labels,
                                                  test_size=0.20, stratify=labels, random_state=42)
# konstruiranje generatora slika za trening koji će se koristiti za
# povećavanje podataka (engl.data augmenatiton)
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")
```

Slika 3.7. Prikaz podjele i povećavanja podataka

3.6.5. Treniranje modela na prethodno obrađenim podatcima

Za treniranje modela prvo je potrebno inicijalizirati parametre potrebne za treniranje neuronske mreže. Zatim se model kompajlira i trenira na prethodno povećanim i obrađenim podatcima [Slika 3.8.].

```
# definiranje hiperparametara za treniranje neuronske mreže tj. inicijalizacija
# broja epoha za trening , veličinu serije (engl. batch size) i inicijalnu stopu
# učenja(engl. learning rate)
initial_LR = 1e-4
EPOCHS = 20
BS = 32

# kompajliranje modela
print("[INFO] compiling model...")
opt = Adam(lr=initial_LR, decay=initial_LR/ EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
              metrics=["accuracy"])
# treniranje ranije definiranog head modela
print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

Slika 3.8. Prikaz treniranja modela na prethodno obrađenim podatcima

U nastavku su prikazane epohe kod odvijanja treninga neuronske mreže na računalu na dvije slike [Slika 3.9.] i [Slika 3.10.].

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/20
95/95 [==============================] - 821s 9s/step - loss: 0.5651 - accuracy: 0.7319 - val_loss: 0.1450 - val_accuracy: 0.9
831
Epoch 2/20
95/95 [==============================] - 661s 7s/step - loss: 0.1697 - accuracy: 0.9547 - val_loss: 0.0788 - val_accuracy: 0.9
883
Epoch 3/20
95/95 [==============================] - 731s 8s/step - loss: 0.1081 - accuracy: 0.9736 - val_loss: 0.0589 - val_accuracy: 0.9
896
Epoch 4/20
95/95 [==============================] - 181s 2s/step - loss: 0.0860 - accuracy: 0.9790 - val_loss: 0.0459 - val_accuracy: 0.9
922
Epoch 5/20
95/95 [==============================] - 127s 1s/step - loss: 0.0721 - accuracy: 0.9820 - val_loss: 0.0410 - val_accuracy: 0.9
896
Epoch 6/20
95/95 [==============================] - 126s 1s/step - loss: 0.0685 - accuracy: 0.9783 - val_loss: 0.0368 - val_accuracy: 0.9
922
Epoch 7/20
95/95 [==============================] - 127s 1s/step - loss: 0.0586 - accuracy: 0.9811 - val_loss: 0.0348 - val_accuracy: 0.9
922
Epoch 8/20
95/95 [==============================] - 132s 1s/step - loss: 0.0442 - accuracy: 0.9866 - val_loss: 0.0336 - val_accuracy: 0.9
909
Epoch 9/20
95/95 [==============================] - 127s 1s/step - loss: 0.0509 - accuracy: 0.9847 - val_loss: 0.0321 - val_accuracy: 0.9
909
Epoch 10/20
95/95 [==============================] - 127s 1s/step - loss: 0.0428 - accuracy: 0.9845 - val_loss: 0.0305 - val_accuracy: 0.9
909
Epoch 11/20
95/95 [==============================] - 126s 1s/step - loss: 0.0402 - accuracy: 0.9861 - val_loss: 0.0306 - val_accuracy: 0.9
922
```

Slika 3.9. Prikaz epoha kod treniranja mreže 1 od 2

```
Epoch 12/20
95/95 [==============================] - 126s 1s/step - loss: 0.0328 - accuracy: 0.9923 - val_loss: 0.0287 - val_accuracy: 0.9
909
Epoch 13/20
95/95 [==============================] - 127s 1s/step - loss: 0.0391 - accuracy: 0.9876 - val_loss: 0.0288 - val_accuracy: 0.9
922
Epoch 14/20
95/95 [==============================] - 127s 1s/step - loss: 0.0336 - accuracy: 0.9892 - val_loss: 0.0316 - val_accuracy: 0.9
909
Epoch 15/20
95/95 [==============================] - 127s 1s/step - loss: 0.0392 - accuracy: 0.9906 - val_loss: 0.0270 - val_accuracy: 0.9
922
Epoch 16/20
95/95 [==============================] - 126s 1s/step - loss: 0.0330 - accuracy: 0.9863 - val_loss: 0.0264 - val_accuracy: 0.9
922
Epoch 17/20
95/95 [==============================] - 126s 1s/step - loss: 0.0267 - accuracy: 0.9926 - val_loss: 0.0263 - val_accuracy: 0.9
922
Epoch 18/20
95/95 [==============================] - 128s 1s/step - loss: 0.0346 - accuracy: 0.9883 - val_loss: 0.0273 - val_accuracy: 0.9
922
Epoch 19/20
95/95 [==============================] - 127s 1s/step - loss: 0.0264 - accuracy: 0.9916 - val_loss: 0.0304 - val_accuracy: 0.9
896
Epoch 20/20
95/95 [==============================] - 128s 1s/step - loss: 0.0367 - accuracy: 0.9866 - val_loss: 0.0284 - val_accuracy: 0.9
909
```

Slika 3.10. Prikaz epoha kod treniranja mreže 2 od 2

Kad se sve ovo odvilo potrebno je još spremiti istrenirani model kako bi se mogao kasnije pozvati i koristiti kod daljnjih primjena [Slika 3.11.].

```
# spremanje istreniranog modela
model.save('mask_recog_model.h5')
```

Slika 3.11. Prikaz spremanja istreniranog modela

Nakon izvršavanja koda prikazanog u ovom poglavlju slijedi evaluacija neuronskih mreža konvencionalnim alatima, a ona će biti prikazana u poglavlju vezanim uz rezultate [4.1].

3.7. Drugi osnovni korak rada: Izrada grafičko sučelja web aplikacije MaskValidator i njenih funkcionalnosti

Kako se korisnik ne bi morao baviti kodom i importiranjem raznih modela i biblioteka da bi mogao koristiti željene funkcionalnosti izrađeno je grafičko sučelje web aplikacije MaskValidator. Grafičko sučelje zamišljeno je tako da omogući korisnicima jasan i intuitivan način korištenja web aplikacije MaskValidator. Samo grafičko sučelje izrađeno je pomoću Streamlita koji je Python biblioteka otvorenog koda koja omogućuje izradu web aplikacija za strojno učenje i duboko učenje. Streamlit je odabran u ovom projektu jer uz minimalno znanje CSS-a i HTML-a mogu se izraditi vizualno impresivne web aplikacije s raznim ugrađenim značajkama te jer je posebno prikladan za projekte povezane s strojnim i dubokim učenjem.

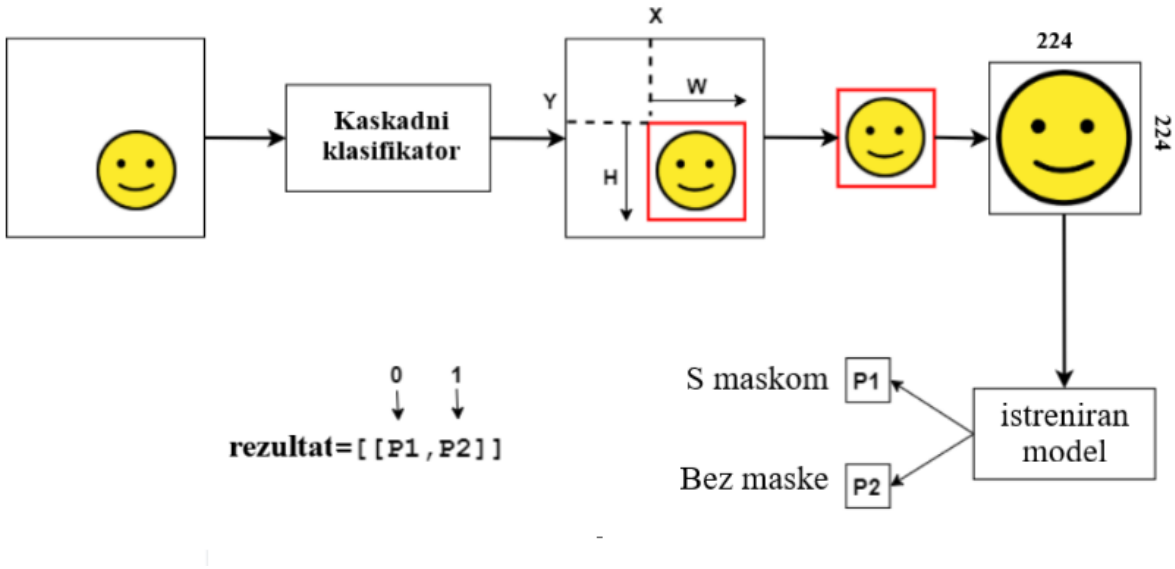
U nastavku ovog poglavlja prvo će biti dan općeniti opis rada funkcionalnosti implementiranih u MaskValidatoru te će nakon toga biti prikazan detaljniji opis izrade njegovog grafičkog sučelja i funkcionalnosti. Samo grafičko sučelje i rad web aplikacije MaskValidator bit će prikazani u poglavlju [4.2.].

Kao što je prikazano na slici [Slika 3.1.] funkcionalnosti MaskValidatora se odnose na primjenu detektora lica na nekom tipu ulaznog podatka i predviđanje nosi li osoba/e masku na licu ili ne na temelju prethodno istreniranog model [3.6.5.] .

Istrenirani model dobiven u [3.6.5.] će se primijeniti na dva načina tj. web aplikacija MaskValidator će imati dva načina rada:

1. Detekcija nošenja ili ne nošenja maske na licu u slikama
2. Detekcija nošenja ili ne nošenja maske na licu u realnom vremenu preko web kamere

Za detekciju značajki lica u obje inačice primjene korišten je kaskadni klasifikator temeljen na Haar značajkama (engl. *Haar feature-based cascade classifier*) preuzet iz [22]. To je algoritam strojnog učenja za detekciju objekata koji se koristi za identifikaciju objekata u slici ili u videozapisu, a temelji se na konceptu značajki predloženih od strane Paula Viole i Micheala Jonesa. U tom konceptu kaskadna funkcija je trenirana na temelju mnogo pozitivnih i negativnih slika. Nakon toga se koristi za detekciju objekata u drugim slikama. Osnovni princip rada obiju inačica primjene je isti i bazira se na upotrebi biblioteke OpenCV, a grafički je prikazan na slici [Slika 3.12.]. U nastavku ovog rada obje inačice bit će detaljno opisane.



Slika 3.12. Grafički prikaz osnovnog principa rada svih inačica primjene

3.7.1. Razrada postupka izrade grafičkog sučelja MaskValidatora i njegovih funkcionalnosti

Kao što je već ranije spomenuto na početku svakog projekta pa tako i ovog moraju se učitati potrebne biblioteke i moduli što je i prikazano na slici [**Pogreška! Izvor reference nije p ronađen.**.].

```
import streamlit as st
#streamlit-webrtc koristi se za prikaz u realnom vremenu koji dolazi iz web kamere
from streamlit_webrtc import webrtc_streamer, VideoTransformerBase , WebRtcMode, ClientSettings
from playsound import playsound
from PIL import Image, ImageEnhance
import numpy as np
import cv2
import os
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model

#  Postavljanje proizvoljnog imena stranice i ikone s promijenjenim izgledom bočne trake
st.set_page_config(page_title='MaskValidator ', page_icon='😷', layout='centered', initial_sidebar_state='expanded')
```

Slika 3.13. Prikaz učitavanja potrebnih biblioteka i modula za detekciju u slikama

Ovdje se tensorflow.kears koristi za pretprocesiranje slike i učitavanje modela, a cv2 tj. OpenCV se koristi za prikaz slike i njene manipulacije. Od ostalih novih biblioteka tu se nalaze streamlit koji služi za izradu samog grafičko sučelja web aplikacije, streamlit_webrtc koji služi za prikaz u realnom vremenu koji dolazi iz web kamere, PIL koji služi za dodatnu obradu slika te playsound modul će služiti za produkciju zvučnog alarma ako maska neće biti detektirana.

Te još ovdje dodana linija kod za proizvoljno ime stranice i ikone koje se pojavljuju u web pregledniku te promijenjen izgled bočne trake na stranici.

U nastavku će biti prikazan i opisan kod koji služi za detekciju maske na licu iz fotografija [Slika 3.14.] . To se postiže definiranjem funkcije (tj. metode) naziva mask_slika_detection koja će se kasnije pozvati. Zatim se unutar nje učitaju model detektora lica i model detektora nošenja maske na licu. Sad nakon što su učitani potrebni modeli mogu se učitati i pretprocesirati slike, a to se radi preko definiranja funkcije gdje je jedini ulazni argument slika na kojoj se želi provesti predviđanje. Nakon toga mora se pretvoriti boja slike u sivu (engl. *grayscale*). Zatim se provodi prolaženje kaskadnog klasifikatora kroz sliku i to tako da kroz različite dijelove slike algoritam prolazi s različitim veličinom za pregled. Nadalje se izvlači regija interesa (engl. *ROI-region of interest*) lica , pretvara iz BGR u RGB kanal, preoblikuje u 224x224 i dalje procesira. Zatim slijedi predviđanje ako je osoba na slici s maskom ili bez maske na temelju vjerojatnosti koju daje istrenirani model i ovisno o predviđanju se dodjeljuje zelena odnosno crvena boja pravokutnika. Na kraju se nacrta pravokutnik oko lica i prikaže predviđena oznaka. Taj postupak je vidljiv ss slike [Slika 3.14.].

```
def mask_slika_detection():
    global RGB_slika

    # učitavanje haarcascade_frontalface_default.xml
    print("[INFO] loading face detector model...")
    faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
    # učitavanje istreniranog modela
    print("[INFO] loading face mask detector model...")
    model = load_model("mask_recog_model.h5")
    frame = cv2.imread("./images/out1.jpg")
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray,
                                         scaleFactor=1.1,
                                         minNeighbors=5,
                                         minSize=(60, 60),
                                         flags=cv2.CASCADE_SCALE_IMAGE)

    for (x, y, w, h) in faces:
        # iz ulaznih podataka izvlači se regija interesa (engl. ROI-region of interest) lica
        # pretvara se iz BGR u RGB kanal, preoblikuje u 224x224 i dalje procesira
        face_frame = frame[y:y + h, x:x + w]
        face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)
        face_frame = cv2.resize(face_frame, (224, 224))
        face_frame = img_to_array(face_frame)
        face_frame = np.expand_dims(face_frame, axis=0)
        face_frame = preprocess_input(face_frame)
        # propuštanje lica kroz istreniran model da se odredi nosi li
        # osoba masku ili ne
        (mask, withoutMask) = model.predict(face_frame)[0]
        label = "S maskom" if mask > withoutMask else "Bez maske"
        color = (0, 255, 0) if label == "S maskom" else (0, 0, 255)
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
        cv2.putText(frame, label, (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 3)
        RGB_slika = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

Slika 3.14. Prikaz pronalaženja lica i određivanja predviđanja u slikama

U sljedećem koraku bit će prikazan i opisan kod koji služi za detekciju maske na licu u realnom vremenu preko web kamere [Slika 3.16.] . To se postiže definiranjem funkcije (tj. metode) naziva mask_webkamera_detection koja će se kasnije pozvati, ali prije toga potrebno je napisati dio koda koje postavlja početne postavke za pravilan rad kamere prikazan na slici [Slika 3.15.].

```
# početne postavke za pravilan rad web kamere
WEBRTC_CLIENT_SETTINGS = ClientSettings(
    rtc_configuration={"iceServers": [{"urls": ["stun:stun.l.google.com:19302"]}]},
    media_stream_constraints={"video": True, "audio": False},
)
```

Slika 3.15. Prikaz postavljanja početnih postavki za pravilan rad web kamere

Nakon toga kod [Slika 3.16.] se u većem dijelu poklapa s kodom koji ima istu funkciju kod detekcije na slikama [Slika 3.14.] pa neće biti pružen dodatni opis za taj dio koda nego će biti samo dodatno opisani oni dijelovi koji se razlikuju. Prva razlika je odmah na početku koda i to je dio koda vezan za definiranje nove klase VideoTransformer s ulaznim argumentom VideoTransformerBase te odmah unutar nje definiranje nove metode(tj. funkcije) transform. Taj kod služi za pravilan rad modula streamlit_webrtc koji onda osigurava pravilan rad web kamere.. Predzadnja razlika je kod za aktivaciju zvučnog alarma koji se aktivira u slučaju ako je detektirana osoba bez maske. Zadnja razlika je kod za prikaz izlaznih podataka na zaslon računala.

```
def mask_webkamera_detection():

    class VideoTransformer(VideoTransformerBase):
        def transform(self, frame):
            frames = frame.to_ndarray(format="bgr24")

            '# učitavanje haarcascade_frontalface_default.xml'
            print("[INFO] loading face detector model...")
            faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
            # učitavanje istreniranog modela
            print("[INFO] loading face mask detector model...")
            model = load_model("mask_recog_model.h5")
            gray = cv2.cvtColor(frames, cv2.COLOR_BGR2GRAY)
            faces = faceCascade.detectMultiScale(gray,
                                                 scaleFactor=1.1,
                                                minNeighbors=5,
                                                minSize=(60, 60),
                                                flags=cv2.CASCADE_SCALE_IMAGE)

            for (x, y, w, h) in faces:
                # iz ulaznih podataka izvlači se regija interesa (engl. ROI-region of interest) lica
                # pretvara se iz BGR U RGB kanal, preoblikuje u 224x224 i dalje procesira
                face_frame = frames[y:y + h, x:x + w]
                face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)
                face_frame = cv2.resize(face_frame, (224, 224))
                face_frame = img_to_array(face_frame)
                face_frame = np.expand_dims(face_frame, axis=0)
                face_frame = preprocess_input(face_frame)
                faces_list.append(face_frame)
                # propuštanje lica kroz istreniran model da se odredi nosi li
                # osoba masku ili ne
                (mask, withoutMask) = model.predict(face_frame)[0]
                label = "S maskom" if mask > withoutMask else "Bez maske"
                color = (0, 255, 0) if label == "S maskom" else (0, 0, 255)
                label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
                cv2.putText(frames, label, (x, y - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)
                cv2.rectangle(frames, (x, y), (x + w, y + h), color, 3)

                # Aktivacija alarma kad je oznaka "Bez maske"
                if mask < withoutMask:
                    path = os.path.abspath("Alarm.wav")
                    playsound(path)
            return frames
```

Slika 3.16. Prikaz pronalaženja lica i određivanja predviđanja u realnom vremenu preko web kamere

Nakon što je definiran i opisan koda za funkcionalnosti web aplikacije MaskValidator u nastavku bit će prikazan i objašnjen kod kojim su definirane pojedine stranice u grafičkom sučelju.

Unutar metode (funkcije) mask_detection definirana su svojstva koja će se prikazivati na svim stranicama od kojih je u kodu prvi određeni glavni naslov. Nakon njega određen je izgled bočne trake u kojoj je dodan logo MaskValidatora te je u bočnu traku dodan padajući izbornik pomoću kojeg će se pristupati pojedinima stranicama koje su definirane kao početna, slika i web kamera i to je sve prikazano na slici [Slika 3.17.]

```
def mask_detection():

    st.markdown('<h1 align="center"> Detekcija maske na licu 😷</h1>', unsafe_allow_html=True)
    # postavljanje izgleda i opcija bočne trake
    activities = ["Početna","Slika", "Web kamera"]
    st.set_option('deprecation.showfileUploaderEncoding', False)
    st.sidebar.image("logo.png")
    st.sidebar.write(" ------ ")
    st.sidebar.markdown("## Popis mogućih opcija")
    choice = st.sidebar.selectbox("Odaberite jednu od ponuđenih opcija:", activities)
```

Slika 3.17. Prikaz svojstava koja se protežu kroz sve stranice web aplikacije

Nakon što su definirana svojstva koja se protežu kroz sve stranice kreće se s definiranjem pojedinačnih stranica u onom redoslijedu kako se pojavljuju u padajućem izborniku pa je prvo opisana početna stranica [Slika 3.18.]. Ona se odabira pomoću if petlje čime se povezuje s padajućim izbornikom te se u nastavku preko ugrađenih funkcija (metoda) biblioteke streamlit st.write i st.image uređuje izgled sadržaja same stranice. Sadržaj same stranice sadrži kratku opis i princip rada detekcije unutar web aplikacije MaskValidator što će se moći ljepše vidjeti u poglavlju [4.2.].

```
# oblikovanje izgleda početne stranice
if choice == "Početna":
    col1, col2, col3 = st.beta_columns([1.5, 6, 1])
    with col1:
        st.write("")
    with col2:
        st.image("logo.png")
    with col3:
        st.write("")
    st.write('')
    st.write(
        'Razvoj ovog projekta bio je potaknut pandemijom koronavirusa naročito njezinim drugim valom  koji je donio
         svakodnevno pojavljivanje velikog broja novozaraženih i  povećanje  stope smrtnosti.')
    st.write(
        ' Uslijed te situacije donesene su razne preventivne mjere kao što su: nošenje maske za lice, redovito pranje ruku
          te održavanje razmaka. Zdravstvene organizacije i vlade posebno su kao mjeru sprječavanja širenja bolesti
          COVID 19 isticale nošenje maske za lice na mjestima gdje ima više ljudi. ')
    st.write(
        'Uzimajući sve prethodno navedeno u obzir razvijena je  web aplikacija MaskValidator kako bi se potaknulo ljude
         na nošenja maski za lice gdje je tako propisano te time  pomoglo u prevenciji i u sprječavanju daljnjeg
         širenja koronavirusa.')
    st.write(
        ' MaskValidator je web aplikacija za detekciju maske na licu s točnošću modela od 99% razvijena pomoću Pythona
          i njegovihbiblioteka. Sama aplikacija je bazirana na konceptima konvolucijskih neuronskih mreža i računalnog
          vida koji omogućuju detekciju maske na licu u dva načina rada: detekcija na fotografijama i detekcija u
          prijenosu uživo tj. preko web kamere.')
    st.markdown('<h3 align="center"> Za odabir željenog načina detekcije odaberite jednu od ponuđenih opcija
                'u padajućem izborniku koji se nalazi na bočnoj traci s lijeve strane</h3>',
                unsafe_allow_html=True)
    st.write('')
    st.write('')
    st.write('## Grafički prikaza rada web aplikacije MaskValidator')
    st.image("prikaz_rada.png")
```

Slika 3.18. Prikaz izgleda sadržaja početne stranice

Zatim se prelazi na definiranje izgleda sadržaja stranice označne oznakom slika [Slika 3.19.]. Kao kod i početne stranice prvo se pomoću petlje if povezuje sama stranica s padajućim izbornikom te se nakon toga definira podnaslov stranice i mogućnost prenošenja slike s lokalnog uređaja na web aplikaciju preko ugrađene funkcionalnosti streamlita st.file_uploader. Nakon toga slijedi unutar dvije if petlje dio koda preko kojeg se omogućuje i poziva ranije definirana funkcionalnost za detekciju maske na licu iz fotografija koje korisnik prenese[Slika 3.14.] sam rad opisane funkcionalnosti na stranici bit će detaljno opisan u poglavlju [4.2.].

```
# oblikovanje izgleda stranice za detekciju iz slike
if choice == 'Slika':
    st.markdown('<h2 align="center">Detekcija na slici </h2>', unsafe_allow_html=True)
    st.markdown("### Ovdje prenesite svoju sliku ⬇")
    image_file = st.file_uploader("Prensite sliku",
                                  type=["png", "jpg", "jpeg", "gif"])  # upload image
    if image_file is not None:
        our_image = Image.open(image_file)  # making compatible to PIL
        im = our_image.save('./images/out1.jpg')
        saved_image = st.image(image_file, caption='', use_column_width=True)
        st.markdown('<h3 align="center">Slika je uspješno prenesena!</h3>', unsafe_allow_html=True)
        if st.button('Obrada'):
            mask_slika_detection()
            st.image(RGB_slika, use_column_width=True)
```

Slika 3.19. Prikaz izgleda i omogućavanje funkcionalnosti na stranici oznake slika

Te na kraju ovog poglavlja slijedi opis definiranja izgleda stranice opisane oznakom web kamera. Kao i kod prve dvije stranice i ona se preko if funkcije poveže s padajućim izbornikom na bočnoj traci te ostatak danog koda [Slika 3.20.] opisuje dodavanje podnaslova i pozivanje funkcionalnosti detekcije maske na licu iz prijenosa u realnom vremenu preko web kamere.

```
# oblikovanje izgleda stranice za detekciju u realnom vremenu tj. preko web kamere
if choice == 'Web kamera':
    st.markdown('<h2 align="center">Detekcija preko web kamere </h2>', unsafe_allow_html=True)
    st.markdown("### Za početak rada prisnite tipku start ⬇")
    mask_webkamera_detection()

mask_detection()
```

Slika 3.20. Prikaz izgleda i omogućivanje funkcionalnosti na stranici oznake web kamera

4. EKSPERIMENTALNI REZULTATI

U ovom poglavlju će biti prikazani dobiveni rezultati. Oni su podijeljeni u dva dijela. Prvi dio će prikazati rezultate dobivene pomoću konvencionalnih alata za evaluaciju neuronskih mreža, a drugi dio će prikazati ispitivanje funkcionalnosti web aplikacije i sam način njezinog korištenja unutar napravljenog grafičko sučelja izrađenog u poglavlju [3.7.].

4.1. Evaluacija istreniranog modela pomoću konvencionalnih alata za evaluaciju neuronske mreže

U poglavlju [3.6.4] prikazano je da je skup podataka podijeljen u dvije kategorije. Prvi i najveći dio skupa podataka koristi se za trening mreže te sadržava 80% ukupne veličine skupa podatka, a drugi dio skupa podataka sadržava ostatak. To je bitno jer tom podjelom je osigurano da se nijedna slika ne pojavljuje više puta i time se riješio mogući problem pristranosti istreniranog modela neuronske mreže. U nastavku na slici [Slika 4.1.] je dan kod pomoću kojeg se dobije na lijep način formatirani prikaz klasifikacijskog izvješća i kod za izradu grafičkog prikaza stope gubitka treninga (engl. *training loss*) i točnosti (engl. *accuracy)*.

```
# izvršavanje predviđanja na skupu za testiranje
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# za svaku sliku u skupu za testiranje treba pronaći indeks oznake
# s pripadajućom najvećom predviđenom vjerojatnošću
predIdxs = np.argmax(predIdxs, axis=1)

# prikaz klasifikacijskog izvješća
print(classification_report(testY.argmax(axis = 1), predIdxs,
    target_names = lb.classes_))


# grafički prikaz stope gubitka (engl. training loss)
# i točnosti (engl.accuracy)
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("graf")
```

Slika 4.1. Izgled dijela koda za prikaz rezultata dobivenih konvencionalnim alatima za evaluaciju neuronskih mreža

Provedbom gornjeg koda dobiven je prikaz klasifikacijskog izvješća na slici [Slika 4.2.] i grafički prikaz stope gubitka treninga (engl. *training loss*) i točnosti (engl. *accuracy)* na slici [Slika 4.3.]

	precision	recall	f1-score	support
with_mask	0.99	0.99	0.99	383
without_mask	0.99	0.99	0.99	384
accuracy			0.99	767
macro avg	0.99	0.99	0.99	767
weighted avg	0.99	0.99	0.99	767

Slika 4.2. Prikaz klasifikacijskog izvješća

Klasifikacijsko izvješće u Pythonu prikazuje preciznost (engl. *precision)*, f1-mjeru(engl. *f1-score*), odziv (engl. *recall*), točnost (engl. *accuracy*) i podršku (engl. *support*).

Postoje četiri načina kako se može vidjeti ako je predviđanje točno ili krivo [23]:

1. **TN/ Točni negativi** – slučaj je negativan i predviđanje je negativno
2. **TP/ Točni pozitivi** – slučaj je pozitivan i predviđanje je pozitivno
3. **LP/ Lažni pozitivi** – slučaj je pozitivan, a predviđanje je negativno
4. **LN/ Lažni negativi** – slučaj je negativan, a predviđanje je pozitivno

Na temelju njih se određuju točnost, preciznost i odziv, a f1-mjera se određuje na temelju dobivenih preciznosti i odziva.

Preciznost (engl. *precision)* ili točnost pozitivnih predviđanja se odnosi na postotak donesenih predviđanja koja su ispala točna. Ona je sposobnost klasifikatora da ne označi slučaj pozitivnim, ako je u stvarnosti negativan. Za svaku klasu je određena kao omjer točnih pozitiva i sume točnih pozitiva i lažnih pozitiva (1).

$$Preciznost = TP/(TP + LP) \quad \textbf{(1)}$$

Odziv (engl. *recall)* ili postotak pozitivnih predviđanja koja su točno određena. Ona je sposobnost klasifikatora da nađe sve pozitivne slučajeve. Za svaku klasu je određena kao omjer točnih pozitiva i sume točnih pozitiva i lažnih negativa (2).

$$Odziv = TP/(TP + LN) \quad \textbf{(2)}$$

F1-mjera (engl. *f1-score)* je težinska srednja vrijednost preciznosti i odziva i to tako da je njena najbolja vrijednost prikazan s 1.0, a najgora s 0.0 (3) . F1-mjera ima većinom manje

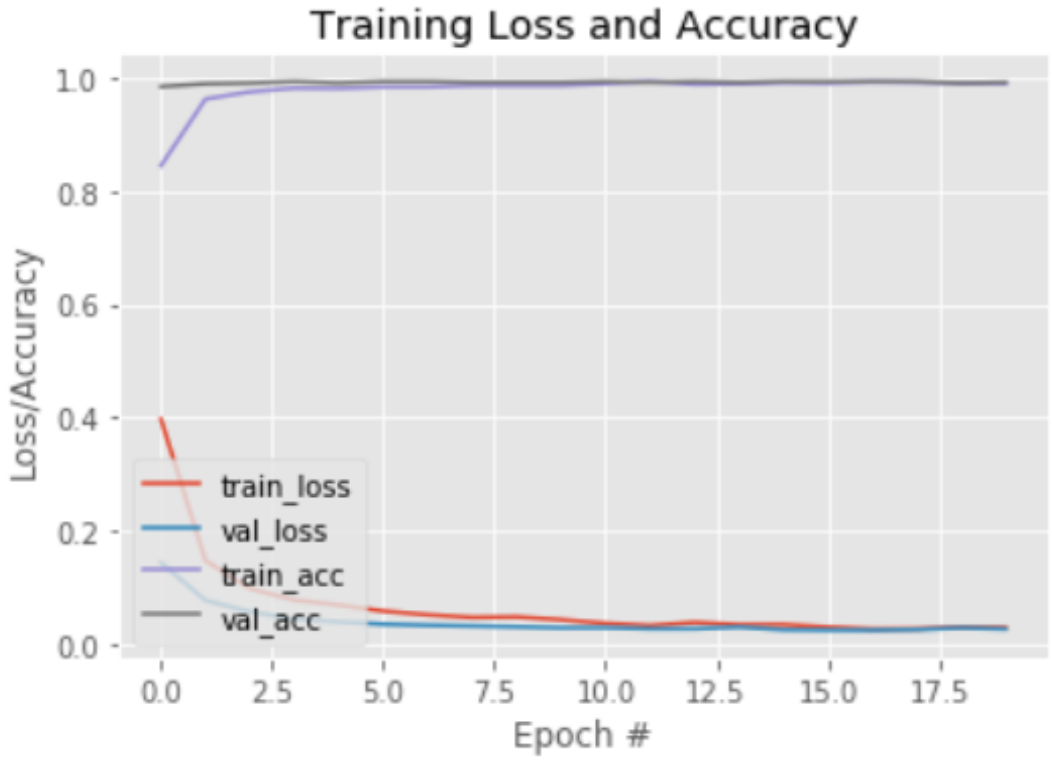
iznose od točnosti jer u svojem računu spaja preciznost i odziv. Kao pravilo f1 se koristi za usporedbu modela klasifikacije, a ne za globalnu točnost.

$$F1\ mjera = 2 * (preciznost * odziv)/(preciznost + odziv) \quad \textbf{(3)}$$

Točnost (engl. *accuracy)* vrijedi samo ako je model ujednačen tj. ako pojedina klasa ne sadrži znatno više primjera (slučaja) od druge. Iz [3.5.] vidi se da je odabrani skup podatak dobro odabran jer obje klase sadrže približno podjednak broj primjera. Za svaku klasu je određena kao suma točnih pozitiva i točnih negativ kroz suma točnih pozitiva, točnih negativa, lažnih pozitiva i lažnih negativa (4).

$$Točnost = (TP + TN)/(TP + TN + LP + LN) \quad \textbf{(4)}$$

Podrška (engl. *support*) je broj stvarnih pojavljivanja pojedine klase u odabranom skupu podataka (engl. *dataset).* Neujednačenosti podrške u podatcima za trening mogu upozoravati na strukturalnu slabost u dobivenim rezultatima klasifikatora i na potrebu za uzimanjem novih primjera u skup podataka. Podrška se ne mijenja između modela nego dijagnosticira proces evaluacije.[23]



Slika 4.3. Grafički prikaz gubitka i točnosti

Na slici [Slika 4.3.] vidimo graf gubitka i točnosti treninga . Na osi apscisa se nalazi broj epoha tj. broj prolazaka neuronske mreže kroz cijeli skup podataka, a na osi ordinata dobivena točnost

(eng. accuracy) i gubitak (engl. *loss)*. Crvena linija predstavlja podatke koji se odnose na gubitak, a dobiveni su od skupa podataka koji se koristio za trening, dok plava linija predstavljaju podatke koji se odnose na gubitak, a dobiveni su od podataka namijenjenih za validaciju (testiranje) mreže. Razlika između gubitka i točnosti je u tome što je funkcija koja opisuje gubitak sa svakom epohom postiže sve manje vrijednosti, a za točnost vrijedi suprotno. Cilj kod treniranja modela neuronske mreže je postići što veću točnost i koda toga dobiti da se točnosti skupa za treniranje i testiranje ne razlikuju previše. Podaci za trening vrlo brzo konvergiraju prema konačnih 99% točnosti i vidljivo je da se kod već druge epohe postiže točnost iznad 90% skupa podataka za treniranje i onda nakon toga počinju konvergirati sa skupom podataka za testiranje prema konačnoj točnosti od čak 99%.

Konfiguracija zadnjeg (izlaznog) sloja neuronske mreže može se promatrati kao okvir na temelju kojeg se predviđa problem, onda se funkciju gubitaka može gledati kao način da se izračuna greška tog okvira zadanog problema. Pošto je u ovom zadatku problem binarne prirode tj. određuje se vjerojatnost pripadnosti jednoj od dviju klasa kao funkcija gubitaka se uzima binarna unakrsna entropija (engl. *binary crossentropy*). Na temelju nje se računaju gubitci pri skupu za treniranje i skupu za testiranje (validaciju) . Cilj je da ti gubitci posebno gubitci skupa za testiranje budu najmanji mogući. Na temelju njih se još gleda ako je mreža pretrenirana (engl. *overfitting*) ili podtrenirana (engl. *underfitting)*.

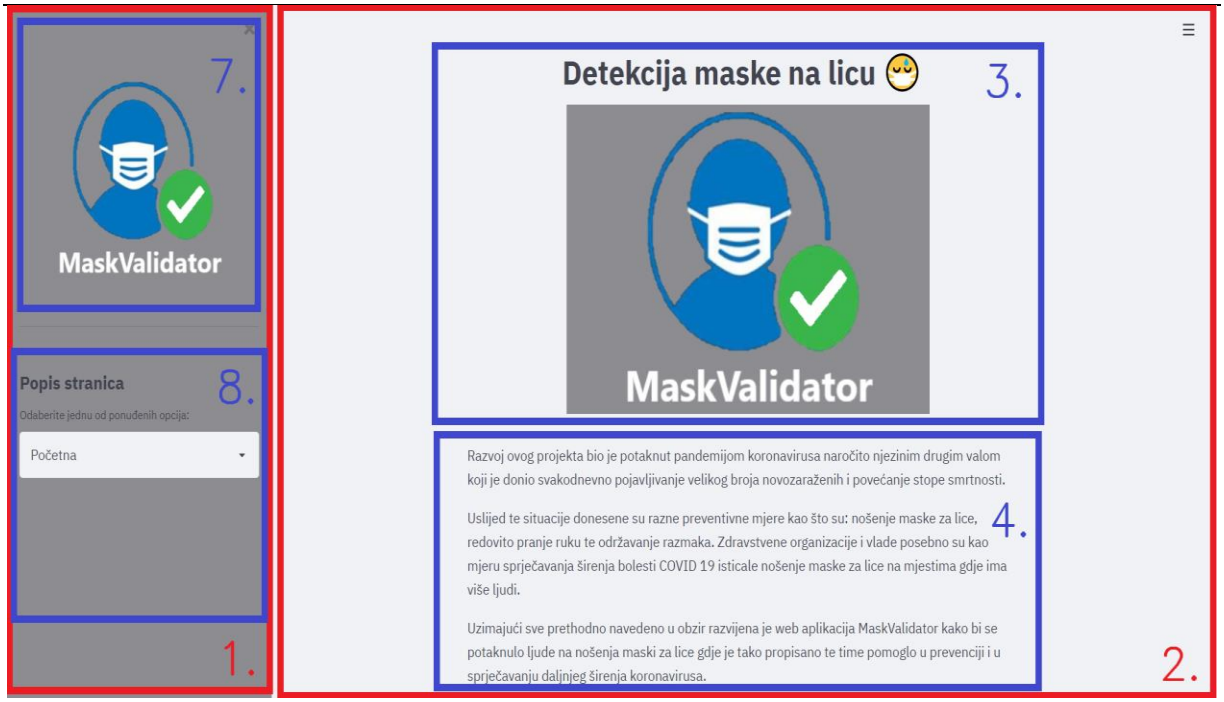
Ako je vrijednost gubitaka skupa za validaciju (testiranje) puno veća od vrijednosti gubitaka skupa za treniranje javlja se pretreniranost, a suprotni slučaj se naziva podtreniranost. Na temelju slike [Slika 4.3.] vidi se da se ove vrijednosti modela koji je istreniran u ovom radu u najvećoj mjeri poklapaju i time ne potpadaju u nijednu od gore spomenutih kategorija neuronskih mreža.

4.2. Ispitivanje funkcionalnosti i opis korištenja web aplikacije MaskValidator

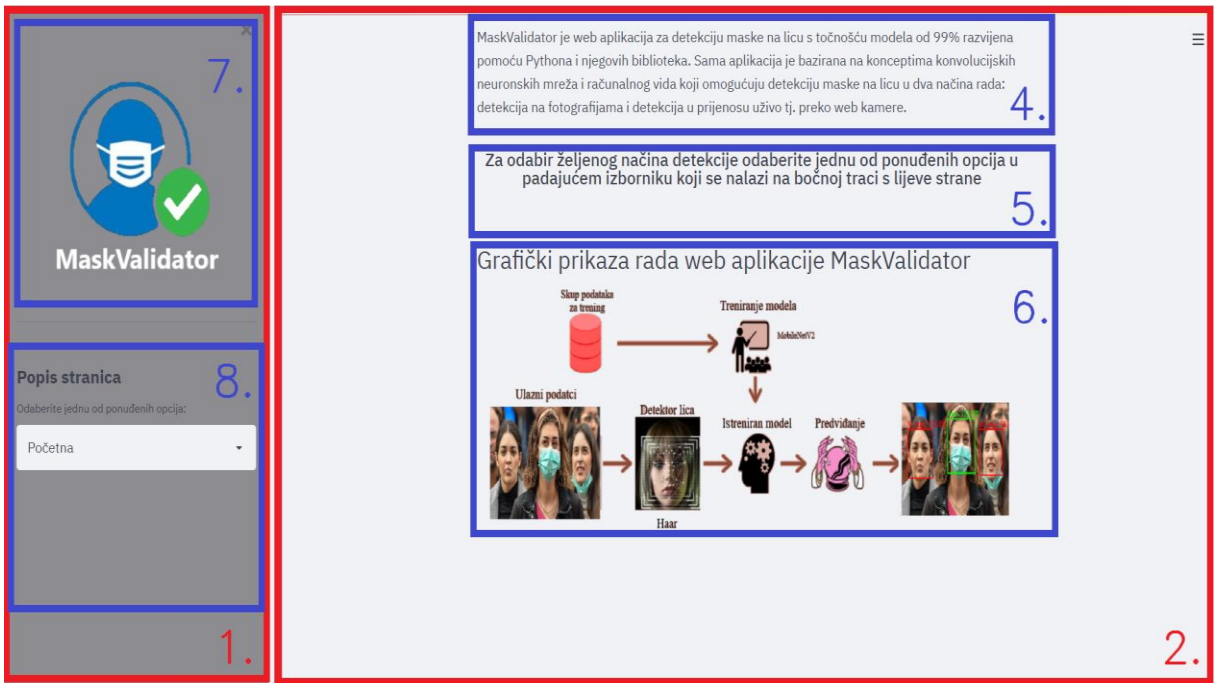
Kao što je već ranije u ovom radu spomenutu MaskValidator je razvijen kao web aplikacija kako bi bio dostupan većem broju korisnika te kako bi se omogućilo korisnicima da bez ikakvog znanja i korištenja programskog koda mogu koristiti željene funkcionalnosti detekcije maske na licu u fotografijama i preko web kamere. Uzimajući u obzir dodatan zahtjev da proces korištenja MaskValidatora bude za korisnika jasan i intuitivan, oblikovano i izrađeno je posebno grafičko sučelje u poglavlju [3.7.]. Da je to stvarno tako bit će prikazano u nastavku kroz opis korištenja same web aplikacije i ispitivanje njenih funkcionalnosti.

Kako bi korisnici mogli pristupiti web aplikaciji preko svoje internetske veze prvo bi je uz sve potrebne datoteke bilo potrebno prenijeti (engl. *upload*) na neki web server čime bi web aplikacija dobila svoju jedinstvenu poveznicu (engl. *link*) koji bi vodio na nju kad bi korisnici taj link unijeli u svoju internetsku tražilicu. Nakon što korisnik ispravno unese točnu poveznicu koja vodi do web aplikacije MaskValidator na zaslonu mu se pojavljuje početna stranica web aplikacije [Slika 4.4.].

Početna stranica kao i preostale dvije stranice sastoji se od dva osnovna dijela označenih crvenim pravokutnicima i brojevima 1 i 2 na slikama [Slika 4.4.] i [Slika 4.5.] , a to su glavni dio stranice (2.) i bočna traka(1.). Prvo što treba uočiti da se početna stranica sastoji od puno sadržaja pa kako bi se on u cijelosti pregledao potrebno se pomaknuti prema dolje pomoću klizača unutar željenog web preglednika i zato je početna stranica prikazana kroz dvije slike gdje prva prikazuje početnu stranicu kod očitavanja, a druga početnu stranicu nakon pomicanja klizača do dna. Glavni dio početne stranice sastoji se od četiri dijela označenih plavim pravokutnicima na slikama [Slika 4.4.] i [Slika 4.5.]. Pravokutnik označen brojem 3. prikazuje naslov i slikovni logo MaskValidatora dok pravokutnik označen brojem 4. označava tekst vezan uz motivaciju za izradu same aplikacije i kratki opis web aplikacije MaskValidator. Pravokutnik označen brojem 5. dodatno upućuje korisnika na mjesto gdje se nalazi padajući izbornik koji služi za navigaciju između stranica te se na dnu same stranice nalazi grafički prikaz rada same aplikacije i on je označen pravokutnikom s brojem 6. .



Slika 4.4. Prikaz početne stranice web aplikacije MaskValidator 1/2

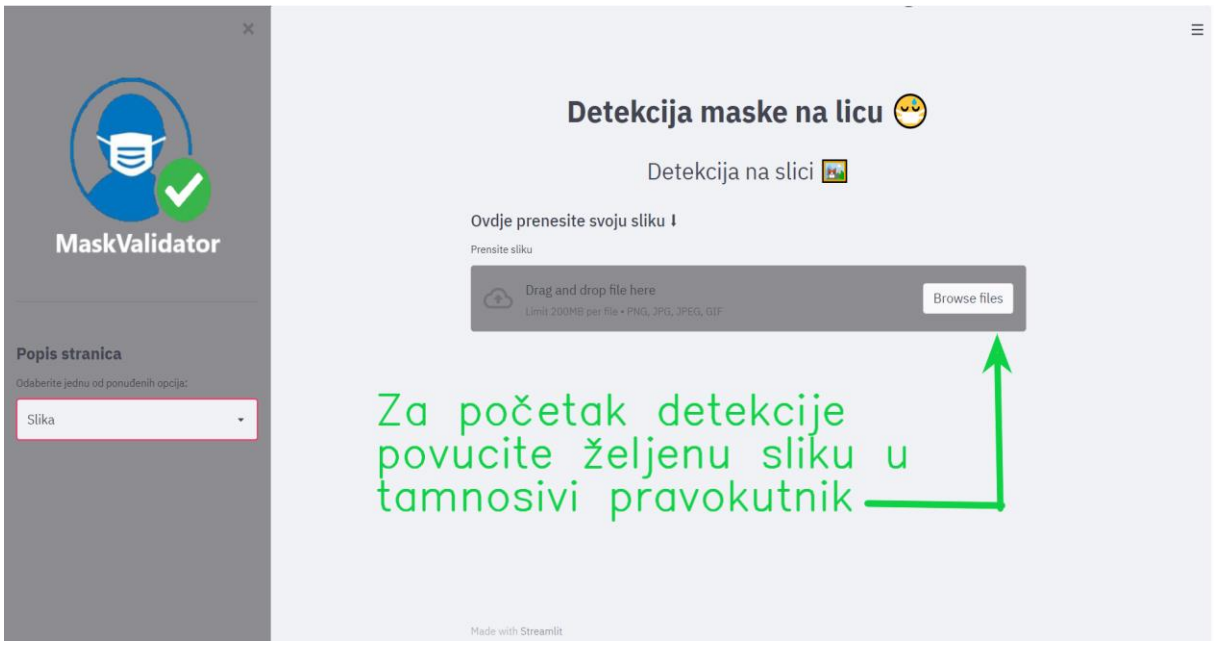


Slika 4.5. Prikaz početne stranice web aplikacije MaskValidator 2/2

Bočna traka je kod svih stranica ista pa će ona biti samo ovdje opisana i sve to isto će vrijediti i za preostale dvije stranice. Sama bočna traka može se podijeliti na dva dijela koji su označeni plavi pravokutnicima pod brojevima 7. i 8. . Gornji dio bočne trake označen brojem 7. sastoji se od slikovnog prikaza loga same aplikacije i crtom je odijeljen od funkcionalnog dijela bočne trake koji će sljedeći biti opisan. Donji dio bočne trake opisan brojem 8. u sebi sadrži uputu koja sugerira gdje se može odabrati željena stranica te ispod nje se nalazi sama funkcionalnost

koja omogućuje navigaciju između stranica tj. opcija koje omogućuju detekcije na različitim podatcima. Sam padajući izbornik se koristi tako da se pritisne na njega nakon čega se on raširi prema dolje kako je prikazano na slici [Slika 4.6.] te se onda pritiskom na jednu od tri ponuđene opcije tj. stranice korisnik prebaci na tu željenu stranicu.

Slika 4.6. Prikaz odabira željene stranice iz padajućeg izbornika

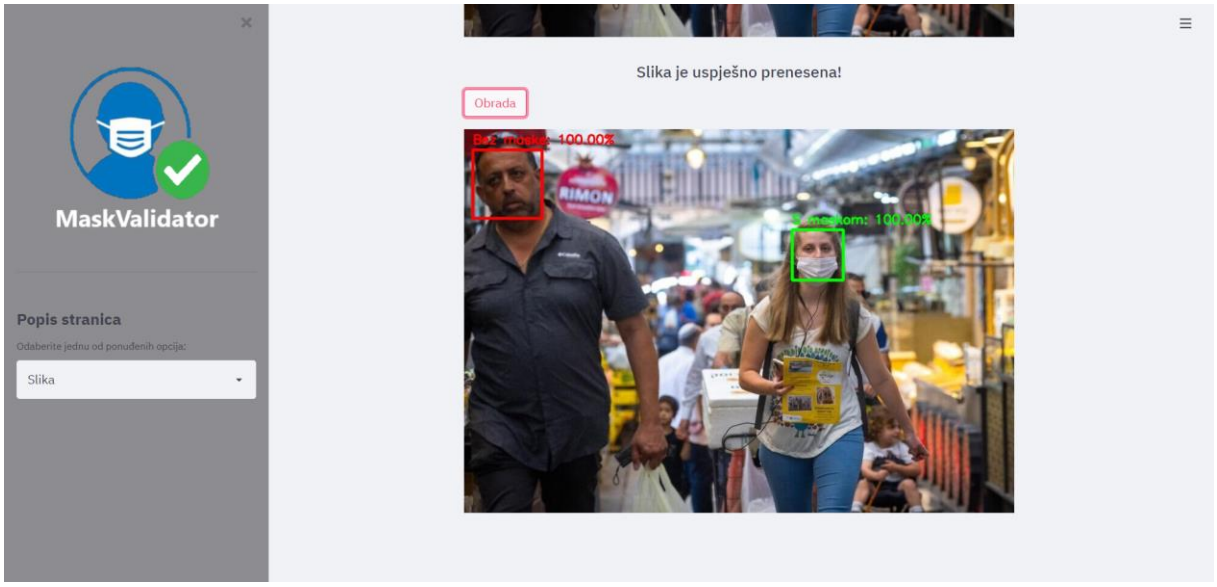


Slika 4.7. Prikaz stranice pod oznakom „Slika“ web aplikacije MaskValidator

Nakon što gore opisanim postupkom za korištenje padajućeg izbornika korisnik odabere stranicu (opciju) pod oznakom „Slika“ prikaže mu se sljedeća stranica [Slika 4.7.]. Ona se sastoji od podnaslova koji govori korisniku da se nalazi na stranici za detekciju maske na licu

na slici. Zatim ispod tog podnaslova je uputa za korištenje same funkcionalnosti te na kraju nalazi se dio kojim se započinje postupak detekcije maske na licu sa slike označen tamnosivom podlogom. Sam postupak korištenja ove funkcionalnosti bit će opisan u nastavku. Kao što i sama uputa na stranici označava korisnik prvo mora prenijeti fotografiju na kojoj želi vršiti detekciju u tamno sivi pravokutnik unutra kojeg su i navedeni svi formati fotografija koje aplikacija podražava, a to su png, jpg, jpeg i gif [Slika 4.7.].

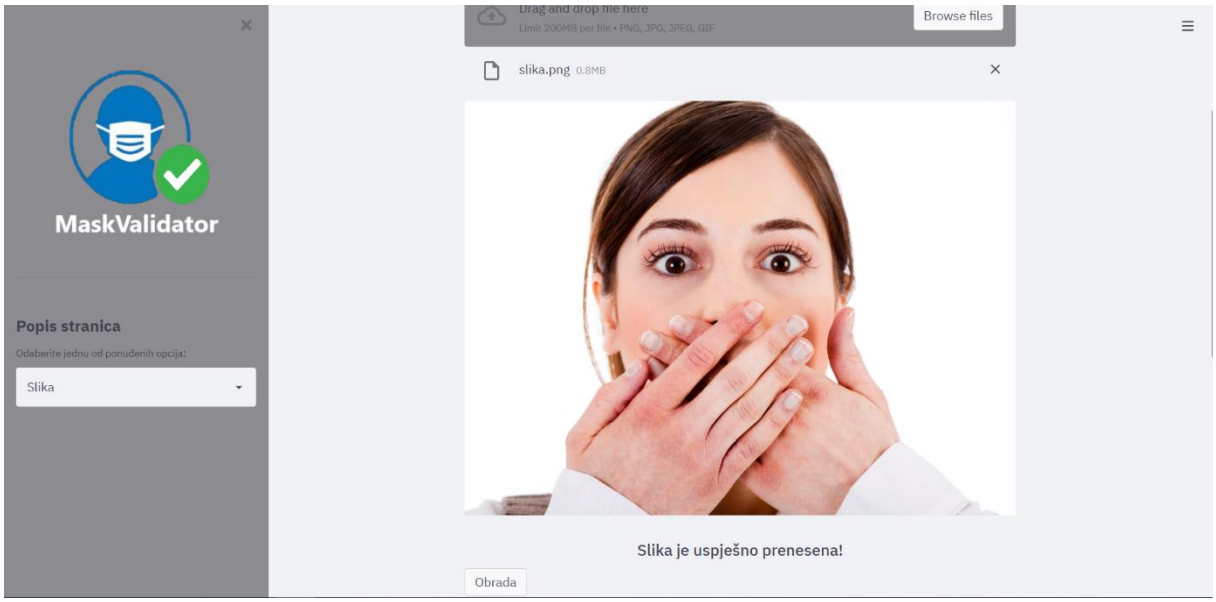
Slika 4.8. Prikaz primjene detekcije na prvoj slici [24]



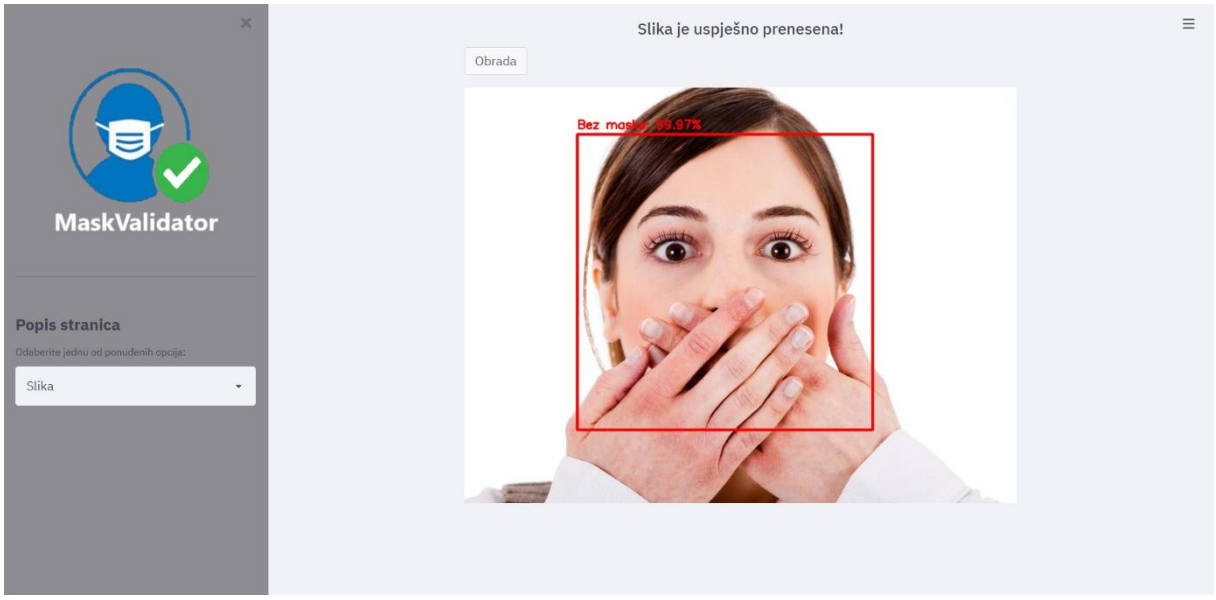
Slika 4.9. Rezultat primjene detekcije na prvoj slici [24]

Nakon što je korisnik prenio svoju sliku u web aplikaciju te ako je ona ispravnog formata ona se učita te se po redu prikaže prvo ime slike i vrsta njene formata, zatim se učita sama slika, nakon toga se ispiše poruka „Slika je uspješno prenesena!“ te se na kraju pojavi gumb unutar

kojeg piše „Obrada“ kao što je i prikazano na slici [Slika 4.8.] . Nakon što se on pojavi i ako je korisnik zadovoljan odabranom slikom korisnik onda pritišće na taj gumb i nakon nekoliko trenutaka prikaže se mu se slika na kojoj su detektirana lica i donijeta predviđanja ako su ta lica su s maskom ili bez. Lica su označena zelenim i crvenima pravokutnicima ovisno o tome ako osoba nosi ili ne nosi masku te je na vrhu tih pravokutnika napisano to predviđanje i postotak vezan uz pojedinačno predviđanje i to je sve vidljivo sa slike [Slika 4.9.] Nakon toga ako korisnik želi provesti detekciju na još nekoj slici samo je prenese ponovo u tamnosivi pravokutnik i ponovi postupak. U nastavku je cijeli postupak izvršen za još jednu sliku te je dan samo slikovni prikaz gotove detekcije vidljiv na slikama [Slika 4.10.] i [Slika 4.11.].



Slika 4.10. Prikaz primjene detekcije na drugoj slici [25]



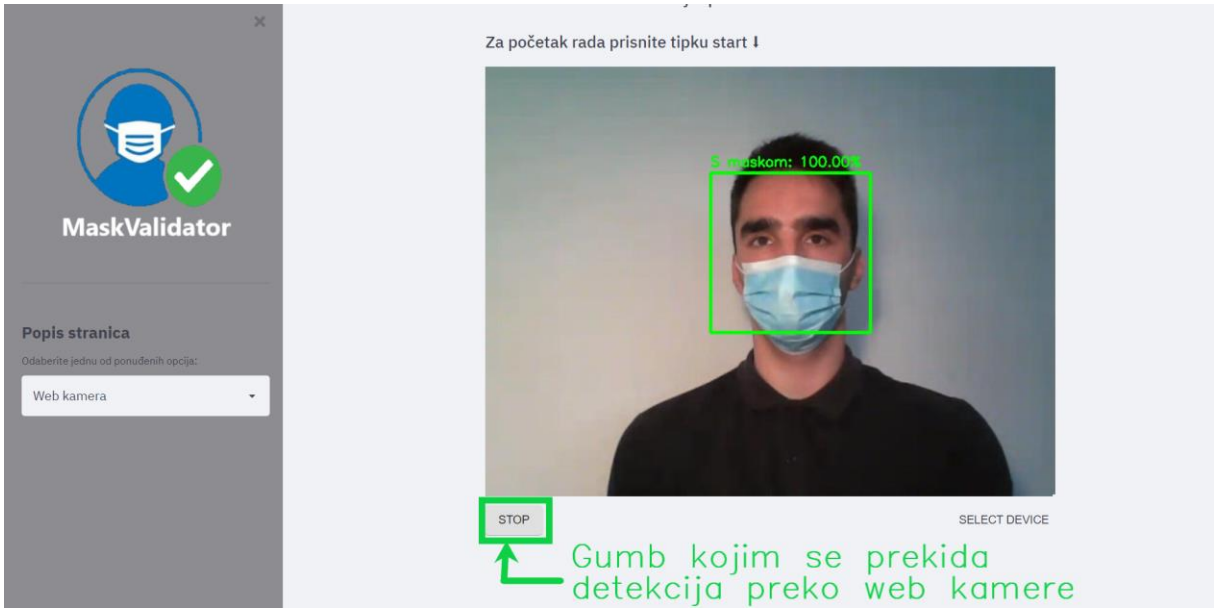
Slika 4.11. Rezultat primjene detekcije na drugoj slici [25]

Te za kraj kad korisnik opisanim postupkom za korištenje padajućeg izbornika odabere stranicu (opciju) pod oznakom „Web kamera“ prikaže mu se sljedeća stranica [Slika 4.12.]. Ona se sastoji od podnaslova koji govori korisniku da se nalazi na stranici za detekciju maske na licu preko web kamere. Ispod tog podnaslova je uputa za korištenje same funkcionalnosti te na kraju nalazi se dio kojim se započinje postupak detekcije maske na licu preko web kamere. Sam postupak korištenja ove funkcionalnosti bit će opisan u nastavku.

Slika 4.12. Prikaz stranice pod oznakom „Web kamera“ web aplikacije MaskValidator

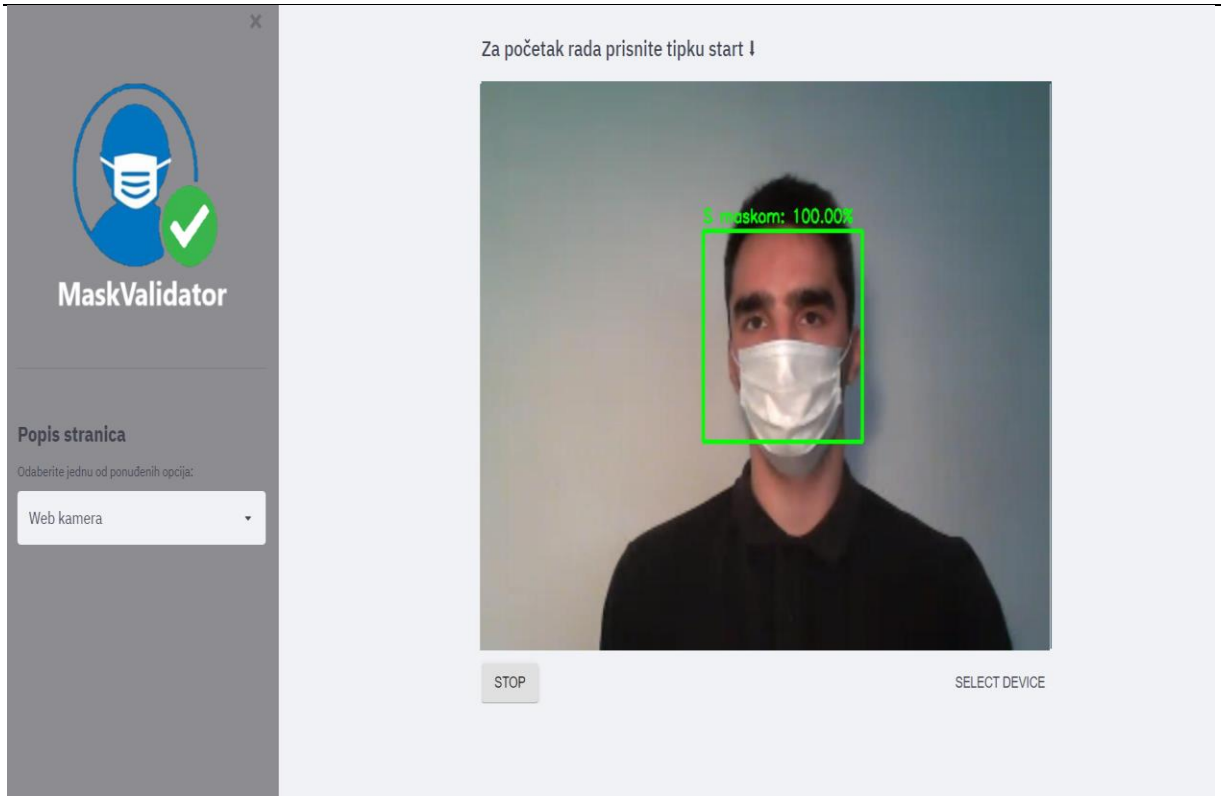
Postupak detekcije kao što i piše u uputi počinje pritiskom na tipku start nakon čega se pojavi obavijest da stranica traži od korisnika dozvolu za upotrebu njegove web kamere što onda on mora odobriti ako želi koristiti željenu funkcionalnost [Slika 4.13.]. Nakon korisnikove dozvole funkcionalnost se nakon nekoliko trenutaka učita te se aktivira web kamere i proširi zaslon na stranici koji prikazuje prijenos iz web kamere u realnom vremenu kao što je vidljivo iz slike [Slika 4.14.]. Sama detekcija maske na licu preko web kamere završava se pritiskom na tipku stop koja se nalazi ispod prozorčića koji prikazuje rezultate detekcije preko web kamere kao što je to prikazano i na slici [Slika 4.14.].

Slika 4.13. Prikaz dozvole potrebne za rad same funkcionalnosti

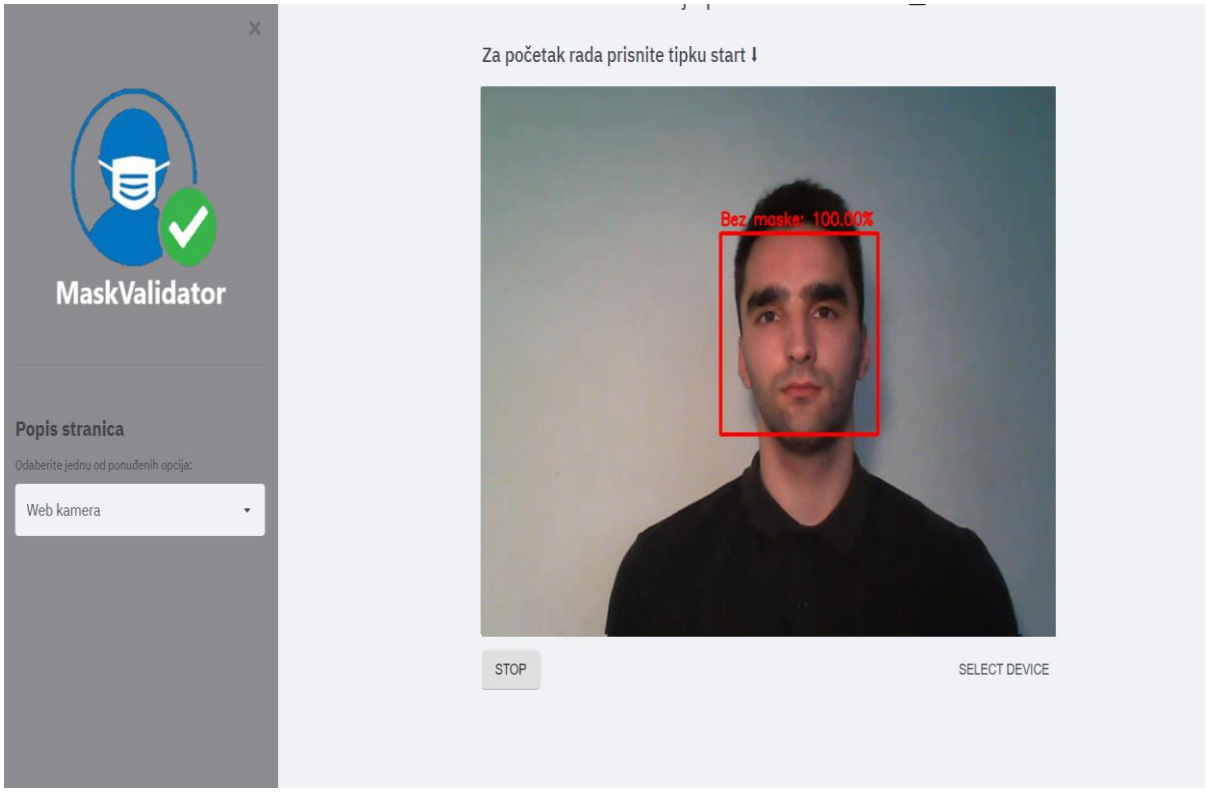


Slika 4.14. Prikaz rezultata detekcije preko web kamere za slučaj nošenja plave maske

U nastavku na slikama [Slika 4.15.] i [Slika 4.16.] bit će prikazani rezultati ove provedbe još kroz dvije slika dobivenih iz daljnje provedbe ove funkcionalnosti MaskValidatora i to u slučajevima kad nema maske i kad je maska prisutna, ali je bijele boje za razliku od prvog slučaju kad je bila plave boje.



Slika 4.15. Prikaz rezultata detekcije preko web kamere za slučaj nošenja bijele maske



Slika 4.16. Prikaz rezultata detekcije preko web kamere za slučaj bez maske

5. RASPRAVA

Iz prethodnog poglavlja [4.2.] vidljivo je da web aplikacija MaskValidator intuitivna za korištenje i da daje vrlo dobra predviđanja u obje svoje inačice primjene. U nastavku ovog poglavlja prokomentirat će se dobiveni rezultati ispitivanja mogućnosti web aplikacije MaskValidator te će na kraju biti dane mogućnosti njenog unaprjeđenja.

Iz slika [Slika 4.8.] i [Slika 4.9.] vidi se da je istrenirani model neuronske mreže dobro označio obje osobe od kojih je jedna s maskom, a druga bez maske što ukazuje na ispravan rad same funkcionalnosti detekcije maske na licu iz slika. Kako bi se taj ispravan rad dodatno utvrdio detekcija je provedena na slici [Slika 4.10.], a na njoj je prikazana osoba koja bi htjela nadmudriti izrađenu funkcionalnost web aplikaciju MaskValidator i to tako da je prekrila dio lica oko usta gdje se uobičajeno nalazi maska s rukama. No može se vidjeti iz slike [Slika 4.11.] da je razvijeni detektor nošenja maski na licu ponovno dobro izdvojio lice osobe i donio dobro predviđanje da osoba na slici ne nosi masku iako su njene ruke stavljene na uobičajenu poziciju maske što je moglo zavarati sustav i time je još jednom potvrđena robusnost same programske aplikacije za detekciju maske na licu . Funkcionalnost detekcije maske za lice preko web kamere razlikuje se od funkcionalnosti detekcije maske za lice iz slike po tome što se odvija u realnom vremenu te preko kamere detektira da li je prisutna osoba ili osobe s maskom ili bez maske uz istovremeno prikazivanje pravokutnika koji obrubljuje lice isto u realnom vremenu te po tome što je ovdje dodano i nova svojstvo koje će biti objašnjene u nastavku. To dodano svojstvo je da ako sustav detektira osobu ili više njih i onda još uz to da ona ili one su bez maske kao što je prikazano na slici [Slika 4.16.] aktivira se zvučni alarm u realnom vremenu kojem je funkcija da upozori osobu da stavi masku. Ispitivanje ove funkcionalnosti MaskValidatora kao što je i ranije spomenuto odvijalo se u realnom vremenu i provedeno je za slučajeve kad nema maske i kad je maska prisutna, a slučaj kad je maska prisutna dodatno je prikazan za dva različita tipa maske koji su se razlikovali po izgledu, materijalu i boji. No vidi se iz slika [Slika 4.14.],[Slika 4.15.] i [Slika 4.16.] da je i ova funkcionalnost za sve slučajeve uspješno donijela točna predviđanja i to s vrlo visokom točnošću. Sve u svemu na temelju dobivenih slika iz ispitivanja funkcionalnosti web aplikacije MaskValidator za detekciju maske na licu vidi se da je ona sposobna za odvijanje preko prenesenih slika, ali i u realnom vremenu preko web kamere te da uz to istovremeno donosi ispravna predviđanja o nošenju maske na licu.

U ovom dijelu poglavlja bit će opisani neki nedostatci same web aplikacije. Nedostatak web aplikacije je to da detektor lica u slučajevima kad su lica previše udaljena od kamere ili ako su pod određenim kutom unutar vidnog polja kamere u kojim ih kamera ne može detektirati tj. ne

može odrediti njihovu prisutnost, a to bi se isto dogodilo ako bi lice bilo previše prekriveno pa ga detektor lica ne bi razaznao dolazi do toga da prisutnost lica nije određena (detektirana) te se posljedično ne mogu donijeti previđanja samog nošenja maske. Rješenje spomenutog problema bit će opisano u nastavku ovog poglavlja u dijelu koji se odnosi na mogućnosti unaprjeđenja same web aplikacije.

Te na kraju ovog poglavlja bit će dana moguća unaprjeđenja web aplikacije MaskValidator koja mogu ići u nekoliko smjerova. Prvi od njih bio bi povećanje broja neurona, skrivenih slojeva i koraka učenja kako bi se dobio bolji model. Zatim povećanje količine i raznolikosti skupa podataka što bi uključivalo još više slika s maskama raznih boja, materijala i tekstura. Još jedan mogući način unaprjeđenja bi bio rješavanje ranije spomenutog problema kad detektor lica ne može pronaći lice i time ne može donijeti ni predviđanje. Taj problem bi se mogao riješiti tako da se pronađe i implementira bolji i pouzdaniji detektora lica od detektora korištenog u ovom radu čime bi se posljedično poboljšao rad i same izrađene programske aplikacije.

6. ZAKLJUČAK

Unutar ovog rada je prikazan tijek razvoja web aplikacije MaskValidator koja služi za detekciju nošenja ili ne nošenja maske na licu. Ova primjena odabrana je zbog mogućnosti njezine primjene u prevenciji i usporavanju daljnjeg širenja bolesti COVID-19 koja je uzrokovala pandemiju u kojoj se cijeli svijet trenutno nalazi te primjenjivosti ovog rješenja kod novih bolesti i pandemija koje će se širiti kapljično i samim time zahtijevati nošenje maske za lice. Samo rješenje temelji se na tehnikama dubokog učenja točnije na primjeni konvolucijskih neuronskih mreža zajedno sa računalnim vidom u Python programskom jeziku. U sklopu teorijske podloge ovog rada dane su osnovne informacije o umjetnoj inteligenciji, strojnom i dubokom učenju, umjetnim i konvolucijskim neuronskim mrežama te računalnom vidu. Na temelju potrebnih Python biblioteka i odabranog skupa podataka treniran i testiran je sam model izrađene neuronske mreže, a pomoću konvencionalnih alata za evaluaciju određeno je da je njegova točnost čak 99%. Nakon toga razvijeno je grafičko sučelje same web aplikacije i unutar njega su ugrađene funkcionalnosti koje na temelju prethodno istreniranog modela omogućavaju detekciju maske na licu u slikama i u realnom vremenu preko web kamere. Ispitivanjem funkcionalnosti web aplikacije MaskValidator moglo se vidjeti da je razvijena web aplikacija vrlo intuitivna za korištenje te se uz to pokazala kao i vrlo robusna u svim svojim inačicama primjene u kojima je davala ispravna predviđanja o nošenju ili ne nošenju maske na licu. Moguće primjene ove aplikacije bile bi u zatvorenim prostorima gdje je obavezno nošenje maske kao što su trgovine, zgrade državnih institucija, škole, fakulteti, autobusni i željeznički kolodvori i mnogi drugi. U tu svrhu moglo bi svaki put kad se detektira osoba bez maske pojaviti umjesto alarma obavijest u obliku zvučnog signala koji bi reproducirao ranije snimljen ljudski glas koji bi služio za podsjećanje osobe da stavi masku na lice.

LITERATURA

[1] umjetna inteligencija. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2020. http://www.enciklopedija.hr/Natuknica.aspx?ID=63150 . Pristupljeno: 25. 5. 2021.

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

[3] SINEF. “Big Data, for better of worse: 90% of world's dana generated over last two years.“ ScienceDaily. ScienceDaily, 22 May 2013. www.sciencedaily.com/releases/2013/05/130522085217.htm .Pristupljeno:25. 5. 2021.

[4] https://www.koronavirus.hr/sto-moram-znati/o-bolesti/najcesca-pitanja-i-odgovori/106 Pristupljeno: 31.5. 2021.

[5] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7757609/ Pristupljeno: 31. 5. 2021.

[6] https://www.ironhack.com/en/data-analytics/what-is-machine-learning Pristupljeno: 7. 6. 2021.

[7] https://machinelearningmastery.com/what-is-deep-learning/ Pristupljeno: 10. 6. 2021.

[8] https://www.xenonstack.com/blog/log-analytics-deep-machine-learning/ Pristupljeno: 10. 6. 2021.

[9] N. Bolf, "OSVJEŽIMO ZNANJE", Kem. Ind. 68 (5-6), 219–220; 2019.

[10] https://medium.com/syntechx/convolutional-neural-network-cnn-in-c-52c9ed47a6ea Pristupljeno: 11. 6. 2021.

[11] https://www.thinkautomation.com/eli5/eli5-what-is-a-convolutional-neural-network/ Pristupljeno: 11. 6. 2021.

[12] https://www.cybiant.com/resources/an-introduction-to-computer-vision/ Pristupljeno: 11. 6. 2021.

[13] https://www.sas.com/en_us/insights/analytics/computer-vision.html Pristupljeno: 11. 6. 2021.

[14] https://docs.python.org/3/faq/general.html#what-is-python Pristupljeno: 12. 6. 2021.

[15] https://jupyter-notebook.readthedocs.io/en/stable/notebook.html Pristupljeno: 12. 6. 2021.

[16] https://www.edureka.co/blog/python-libraries/ Pristupljeno: 12. 6. 2021.

[17] https://jakevdp.github.io/PythonDataScienceHandbook/04.00-introduction-to-matplotlib.html Pristupljeno: 12. 6. 2021.

[18] https://opencv.org/about/ Pristupljeno: 12. 6. 2021.

[19] https://drive.google.com/drive/folders/1XDte2DL2Mf_hw4NsmGst7QtYoU7sMBVG Pristupljeno: 10. 6. 2021.

[20] https://arxiv.org/abs/1801.04381 Pristupljeno: 10. 6. 2021.

[21] https://github.com/balajisrinivas/Face-Mask-Detection Pristupljeno: 10. 6. 2021.

[22] https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt2.xml Pristupljeno: 10. 6. 2021.

[23] https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397 Pristupljeno: 26. 6. 2021.

[24] https://www.timesofisrael.com/health-ministry-weighs-returning-indoor-mask-mandate-amid-school-outbreaks/ Pristupljeno: 26. 6. 2021.

[25] https://bonniedell.wordpress.com/2016/09/20/why-guarding-your-tongue-is-equally-important-as-guarding-your-heart/ Pristupljeno: 26. 6. 2021.

SAŽETAK

Autor: Karlo Posavec

Naslov: MaskValidator - web aplikacija za detekciju maske na licu

Zahvaljujući ogromnoj količini vizualnih podataka koja se dnevno generira, razvojem novih algoritama i većom dostupnošću potrebne računalne snage, u posljednje vrijeme došlo je do značajnog napretka računalnog vida i sukladno tome primjenjuje se kod sve više zadataka koji uključuju prepoznavanje. Uzimajući ovo u obzir u ovom radu je razvijena web aplikacija za prepoznavanje maske na licu nazvana MaskValidator. U radu su objašnjeni potrebni pojmovi vezani za područja koja ova web aplikacija obuhvaća te je također detaljno prikazan razvoj modela umjetne neuronske mreže i grafičkog sučelja same aplikacije. MaskValidator baziran je na programskom jeziku Python te njegovim bibliotekama. Na kraju ovog rada provedena je evaluacija koja je uključivala ljudske subjekte.

Ključne riječi: MaskValidator, maska za lice, računalni vid, Python, konvolucijske neuronske mreže

SUMMARY

Author: Karlo Posavec

Title: MaskValidator – web application for face mask detection

Thanks to the huge amount of visual data generated daily, the development of new algorithms and greater availability of the necessary computing power, in recent times there has been a significant improvement in computer vision and accordingly it is applied to more and more tasks involving recognition . Taking this into account, a web application for face mask detection named MaskValidator was developed. The paper explains the necessary concepts related to the topics covered by this web application and also presents in detail the development of the artificial neural network model and the graphical interface of the application itself. MaskValidator is based on Python programming language and its libraries. At the end of this paper, evaluation was carried out which involved human subjects.

Key words: MaskValidator, face mask, computer vision, Python, convolutional neural networks