Sustav za automatsko mjerenje temperature lica i detekciju maske na licu

Hendija, Karlo

Sveučilište Sjever

Završni rad br. 525/EL/2023

SUSTAV ZA AUTOMATSKO MJERENJE TEMPERATURE LICA I DETEKCIJU MASKE NA LICU

Karlo Hendija

Varaždin, rujan 2023. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za elektrotehniku

STUDIJ preddiplomski stručni studij Elektrotehnika

PRISTUPNIK Karlo Hendija | JMBAG 0336013873

DATUM 23.08.2023. | KOLEGIJ Signali i sustavi

NASLOV RADA Sustav za automatsko mjerenje temperature lica i detekciju maske na licu

NASLOV RADA NA ENGL. JEZIKU System for automatic face temperature measurement and face mask detection

MENTOR Emil Dumić | ZVANJE izv. prof. dr. sc.

ČLANOVI POVJERENSTVA

1. doc. dr. sc. Ladislav Havaš, predsjednik
2. doc. dr. sc. Tomislav Horvat, član
3. izv. prof. dr. sc. Emil Dumić, član
4. doc. dr. sc. Dunja Srpak, zamjenski član
5.

Zadatak završnog rada

BROJ 525/EL/2023

OPIS

U ovom radu bit će napravljen sustav za mjerenje temperature lica i detekciju maske na licu. Različiti sustavi pomažu ljudima s automatiziranim načinom rada kako bi se neke informacije mogle saznati bez prisustva čovjeka. S obzirom na novonastalu situaciju vezanu za COVID-19 pandemiju, korisna informacija je i tjelesna temperatura osobe te da li osoba pravilno nosi masku. Za svaki od ta dva problema postoji više načina rješavanja. Za mjerenje temperatue može se koristiti neki beskontaktni infracrveni senzor. Za detekciju maske na licu prvo se treba detektirati lice na slici, te potom zaključiti da li se na detektiranom licu nalazi maska. Za detekciju lica, može se koristiti npr. Viola-Jones algoritam, SVM+HOG ili neuronska mreža, implementirani unutar OpenCV i TensorFlow programskih platformi. Za detekciju maske može se koristiti neuronska mreža.
U praktičnom dijelu zadatka će se implementirati sustav za mjerenje temperature pomoću infracrvenog senzora. Koristit će se Raspberry Pi i npr. MLX90614 senzor za mjerenje ambijentalne temperature i temperature objekta. Za detekciju maske, koristit će se kamera povezana s Raspberry Pi, ResNet-10 neuronska mreža za detekciju lica, te potom trenirana neuronska mreža MobileNetV2 koja razlikuje lice s maskom od lica bez maske. Dobiveni sustav će se ispitati na nekoliko ispitanika i zaključiti o točnosti rješenja, potencijalnim problemima i mogućim nadogradnjama, kao i o vremenu izvođenja.

ZADATAK URUČEN 23.08.2023. | POTPIS MENTORA Emil Dumić

Sveučilište Sjever

Odjel za elektrotehniku

Završni rad br. 525/EL/2023

SUSTAV ZA AUTOMATSKO MJERENJE TEMPERATURE LICA I DETEKCIJU MASKE NA LICU

Student

Karlo Hendija

Mentor

izv. prof. dr. sc. Emil Dumić

Varaždin, rujan 2023. godine

Sažetak

U ovom radu je napravljen sustav za mjerenje temperature lica te detekciju maske na licu. Također, na jednostavan način kroz teoriju je objašnjeno što je računalni vid, zašto je izrazito važan u današnje vrijeme te opseg njegovog korištenja. Uz praktični dio rada, kroz teoriju su prikazani i objašnjeni razni algoritmi za detekciju lica poput metode potpornih vektora (SVM), Viola-Jones algoritma te histograma orijentiranih gradijenata (HOG). Nadalje je objašnjen proces obrade slike i treniranja sustava za detekciju objekata. Uz to, obrađena je i tema strojnog i dubokog učenja, najčešće korištene arhitekture te osnovne razlike među njima. Za mjerenje temperature lica obrađen je senzor koji se koristi u praktičnom dijelu, način spajanja sa Raspberry Pi računalom te načelo rada sustava za mjerenje temperature. U praktičnom dijelu je implementiran sustav za mjerenje temperature i detekciju lica pomoću ResNet modela te klasifikaciju da li lice ima masku pomoću MobileNetV2 arhitektura neuronskih mreža uz prikazane sve korake sa potrebnim objašnjenjima.

Summary

In this paper, a system for automatic face temperature measurement and face mask detection is implemented. Also, there are simplified theoretical explanation on computer vision, why is it extremely important today and the extent of its use. In addition to the practical part of work, various face detection algorithms such as the support vector machines (SVM), the Viola-Jones algorithm and the histogram of oriented gradients (HOG) are presented and explained through theory. Furthermore, the process of image processing and training of object detection systems is explained. In addition, the topic of machine and deep learning, the most commonly used architectures and the basic differences between them is covered. To measure the temperature of the face, the sensor that was used in the practical part, the method of connecting it to the Raspberry Pi computer and the principle of operation of the temperature measurement system is discussed. In the practical part, a system for measuring the temperature of the face and face detection is implemented using ResNet model and classification if faces wear mask using MobileNetV2 architecture with all the steps shown with the necessary explanation

Popis korištenih kratica

HOG	histogram of oriented gradients	Histogram orijentiranih gradijenata
SVM	Support vector machine	Metoda potpornih vektora
CNN	convolution neural networks	Konvolucijske neuronske mreže
DNN	Deep Neural Networks	Duboke neuronske mreže
DBN	Deep Belief Networks	Duboke mreže vjerovanja
ReLU	Rectified linear unit	Ispravljena linearna jedinica
RBM	Restricted Boltzmann machine	Ograničeni Boltzmannov stroj
ADC	Analog to digital conversion	Analogno – digitalna pretvorba
DSP	Digital Signal Processing	Obrada digitalnog signala
PWM	Pulse Width Modulation	Pulsno-širinska modulacija
SMBus	System Management Bus	Sabirnica za upravljanje sustavom
POR	Power On Reset	Generator resetiranja pri uključenju
IR	Infra Red	Infra-crveno
ResNet	Residual Neural Network	Rezidualna neuronska mreža
Caffe model	Convolutional Architecture for Fast Feature Embedding	Konvolucijska arhitektura za brzo ugrađivanje značajki
mAP	Mean Average Precision	Srednja prosječna preciznost
IoU	Intersection over Union	Omjer presjeka i unije
AP	Average Precision	Prosječna preciznost

Sadržaj

1. Uvod

Razvojem novih tehnologija i mogućnosti, umjetna inteligencija i računalni vid nalaze sve širu primjenu u industriji, medicini i ostalim granama znanosti i djelatnosti, ali i u svakodnevnom životu. Sustavi temeljeni na umjetnoj inteligenciji su sve precizniji, greške su postale zanemarive, a cijene nikad dostupnije zahvaljujući jeftinim računalima i mikroprocesorima poput Arduina i Raspberry Pi-a na kojem se i temelji praktični dio ovog rada. Upravo zahvaljujući navedenim računalima i dostupnosti literature na internetu, svatko uz malo volje i vještine može početi učiti osnove umjetne inteligencije i računalnog vida uz jednostavne projekte za početnike. Uz to, kao veliku pomoć u radu, dostupna nam je i „open source“ biblioteka OpenCV koja nudi razna rješenja za problematiku vezanu uz računalni vid. Da bismo uspješno implementirali neki sustav za detekciju objekata potrebno je biti upoznat sa teorijom koja stoji iza toga. Najčešće korišteni algoritmi za detekciju, razlike među njima, vrste i strukture neuronskih mreža i slično. Kod umjetne inteligencije samo su mašta i znanje granica, naravno uz ograničenja korištenog hardvera. Problematika detekcije objekata na prvu zvuči trivijalno baš zbog ljudskih mogućnosti automatskog detektiranja i prepoznavanja objekata u svakodnevnom životu bez razmišljanja i davanja bilo kakve pažnje tome. Međutim, važno je razumjeti da bi računalo dobilo (donekle) istu percepciju vida za prepoznavanje određenih objekata, potrebna nam je utrenirana duboka neuronska mreža.

2. Računalni vid

Računalni vid (engl. *computer vision*) je grana umjetne inteligencije koja se bavi razvojem algoritama i tehnologija koje omogućuju računalima da interpretiraju i analiziraju vizualne informacije iz digitalnih slika i videozapisa [1]. To podrazumijeva korištenje matematičkih modela i algoritama za prepoznavanje oblika, objekata, lica, gesta, pokreta i drugih vizualnih karakteristika u slikama i videima, te njihovu interpretaciju i obradu na način sličan onome kojim to čini ljudski mozak. Računalni vid se primjenjuje u mnogim područjima, uključujući sigurnost, medicinu, robotiku, automobilsku industriju, zrakoplovstvo, zabavu i druge industrije.

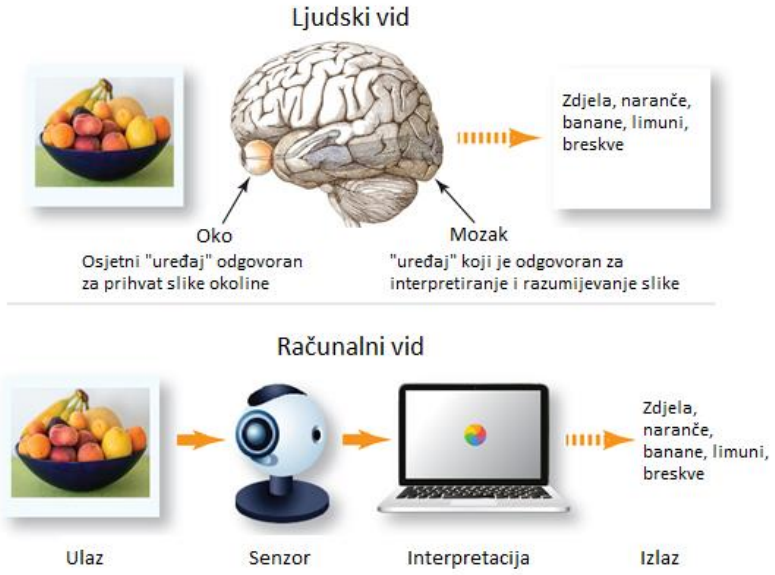
Ljudskom oku je trivijalno detektirati i prepoznati neki objekt, ali da bi isti rezultat postigli računalom, proces je znatno kompliciraniji. Poanta računalnog vida je preslikati mogućnosti ljudskog vida pomoću digitalnih slika sljedećim redoslijedom:

1. dobivanje slike (engl. i*mage Acquisition*),
2. obrada slike (engl. *image processing*),
3. analiza i razumijevanje slike (engl. *image analysis and understanding*).

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

- klasifikacija objekata – uključuje treniranje modela na skupu podataka određenih objekata i zatim model klasificira nove objekte u jednu ili više kategorija određenih treningom,
- identifikacija objekata – postupak u kojem će model prepoznati specifične instance objekata.

Koncept računalnog vida poznat je još od 1950-ih a prva primjena je zabilježena dvadesetak godina kasnije kod razlikovanja ručno pisanog od „tipkanog“ teksta. Danas je računalni vid kao tehnologija široko rasprostranjena u svim granama djelatnosti te omogućuje ogromnu uštedu na vremenu i novcu. Detekcija kvara, otkrivanje uljeza, detekcija zaštitnih maski, otkrivanje bolesti u ranom stadiju, klasifikacija vozila te analiza prometa samo su jedni od primjera korištenja računalnog vida. Na slici 2.1 je prikazana osnovna razlika između ljudskog i računalnog vida.



Slika 2.1. Osnovna razlika ljudskog i računalnog vida (izvor. https://manningbooks.medium.com/how-does-computer-vision-work-bc35b0fb5df5)

Neke od prednosti korištenja računalnog vida:

- pouzdanost,
- jednostavnost i brzina samog procesa,
- mogućnost široke primjene,
- mali troškovi implementacije i održavanja.

3. Algoritmi za detekciju lica

3.1 Viola-Jones algoritam

Viola-Jones algoritam je nazvan po svojim autorima Paulu Violi te Michaelu Jonesu koji su ga osmislili sada već davne 2001. godine [2]. Osnovna namjena ovog algoritma je detekcija lica u realnom vremenu, ali koristan je i kod prepoznavanja bilo kakvih objekata na slici ili videozapisu. Sam algoritam se bazira na strojnom učenju, tj. uz pomoć više „pozitivnih“ i „negativnih“ primjera slika u bazi podataka razvija kaskadnu funkciju. Nakon „uvježbavanja“ poznatih primjera, gotov model se koristi za detekciju lica na drugim slikama. Prilikom procesa detekcije lica algoritam proučava dijelove slike i pokušava detektirati prepoznatljive značajke na tim dijelovima.

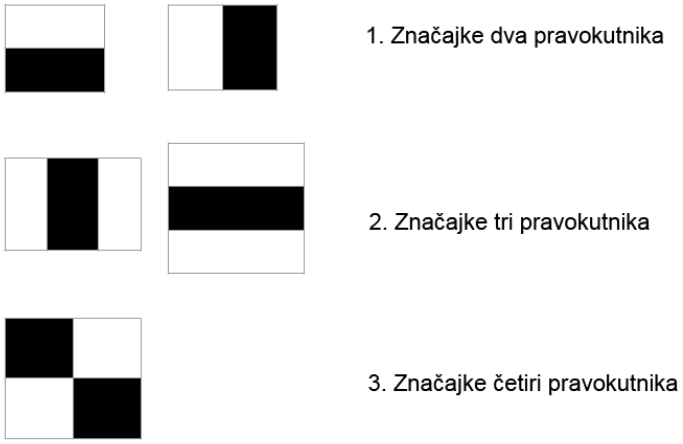
Kod izvođenja Viola-Jones algoritma postoje 4 glavna koraka:

- Haarove značajke,
- integralna slika i izračun Haarovih značajki,
- AdaBoost algoritam,
- kaskadni klasifikator (filter).

3.1.1 Haarove značajke

Da bismo pojednostavili proces, umjesto direktnog rada sa pikselima koristimo rad sa osnovnim značajkama pravokutnog oblika. Koristimo 3 osnovne Haarove značajke (slika 3.1):

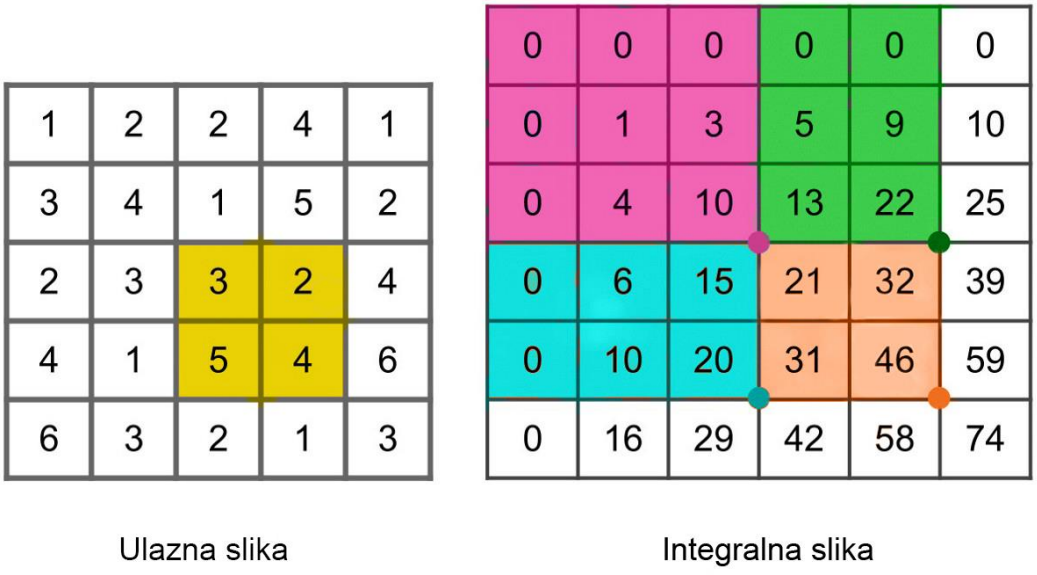
- rubne značajke (engl. *edge features*) - Razlika između zbroja piksela unutar dva pravokutnika. Regije su istog oblika i veličine te su vodoravno i okomito susjedne,
- značajke linija (engl. *line features*) – Zbroj dva vanjska pravokutnika umanjen za zbroj središnjeg pravokutnika,
- značajke četiri pravokutnika (engl. *four rectangle features*) – Razlika između dijagonalnih parova pravokutnika.



Slika 3.1 Osnovne Haarove značajke

3.1.2 Integralna slika i izračun Haarovih značajki

Integralna slika je alternativni prikaz izvorne slike (slika 3.2) ili njezinog dijela u svrhu pojednostavljenja izračuna Haarovih značajki:



Slika 2.2 dobivanje integralne slike

Zbog lakšeg snalaženja na integralnu sliku s lijeve i gornje strane nadodamo piksele s vrijednošću „0“. U sljedećem koraku svaku vrijednost piksela sa izvorne slike prepisujemo u integralnu sliku na istu poziciju na način da toj vrijednosti nadodamo sve vrijednosti piksela koje se nalaze gore i lijevo od navedene pozicije. Prema tome, piksel s vrijednošću „1“ na poziciji (1,1) izvorne slike zadržava istu vrijednost s obzirom da s njemu lijeve i gornje strane nema piksela čije vrijednosti bi mu nadodali. Isto tako, piksel s vrijednošću „5“ na poziciji (3,4) izvorne slike prepisujemo na integralnu sliku na istu poziciju ali na njegovu vrijednost nadodajemo vrijednosti svih piksela koji se nalaze njemu s gornje i lijeve strane. U ovom slučaju:

$$(5)+1+2+2+3+4+1+2+3+3+4+1=31$$

Isti korak ponavljamo za sve piksele izvorne slike. Ovim procesom omogućena je obrada izvorne slike korištenjem kombinacije pravokutnih površina (slika 3.3) umjesto pojedinačnih piksela.

Izračun Haarovih značajki koristeći integralnu sliku:

- osnovna značajka pravokutnika zahtjeva samo 4 ciklusa pretraživanja memorije:

$$A+E-B-D$$

- značajka dva pravokutnika zahtjeva samo 6 ciklusa pretraživanja memorije:

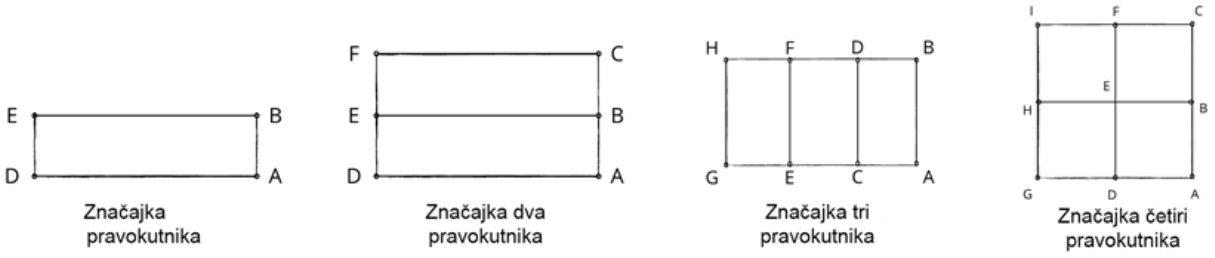
$$A-2B+C-D+2E-F$$

- značajka tri pravokutnika zahtjeva samo 8 ciklusa pretraživanja memorije:

$$A-B-2C+2D+2E-2F-G+H$$

- značajka četiri pravokutnika zahtjeva samo 9 ciklusa pretraživanja memorije:

$$A-2B+C-2D+4E-2F+H-2I+J$$



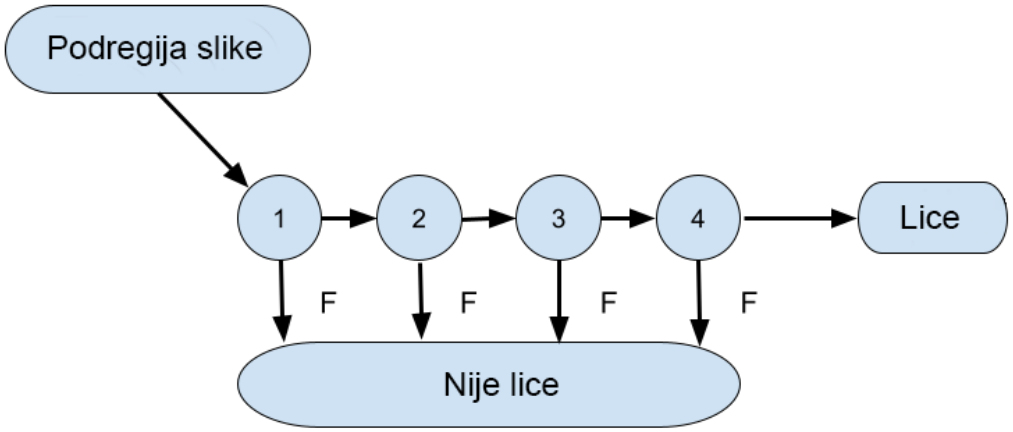
Slika 3.3 Izračun Haarovih značajki

3.1.3 Adaboost (engl. *Adaptive Boosting*)

Namjena Adaboost algoritama je smanjenje broja kombinacija Haarovih značajki na način da istrenira klasifikatore lokaliziranjem najbitnijih Haarovih značajki, te odbacivanjem ostalih manje bitnih značajki. Za to su potrebne dvije baze podataka. Prva se baza podataka sastoji isključivo sa pozitivnim uzorcima, odnosno slikama objekata za detekciju, a druga sa negativnim uzorcima, odnosno slikama bez objekata za detekciju.

3.1.4 Kaskadni klasifikator (filter)

Najbitnije značajke se pohranjuju u binarni klasifikator (slika 3.4). Prolaskom kroz klasifikator pozitivni uzorci šalju se na sljedeću značajku, a negativni se uklanjaju iz daljnje obrade, te se time reducira utrošeno vrijeme na računanje negativnih uzoraka s obzirom da nema potrebe da prolaze kroz sve značajke. Vrijednost praga filtera je moguće promijeniti da bismo podesili „preciznost-odziv“ krivulju. Nizak prag će postići veliki broj lažno pozitivnih (FP) što rezultira manjom preciznošću te manji broj lažno negativnih (FN) što rezultira većom osjetljivošću filtera. U suprotnom slučaju za visoku vrijednost praga filtera postižemo manji broj lažno pozitivnih (FP) čime dobivamo veću preciznost te veći broj lažno negativnih (FN) čime dobivamo manju osjetljivost filtera.



Slika 3.4 Shema kaskadnog klasifikatora

Ovo je ujedno i glavni dio algoritma koji mu omogućuje obradu videa brzinom od 15 sličica u sekundi i omogućuje implementaciju u stvarnom vremenu.

3.2 Histogram orijentiranih gradijenata (HOG)

HOG je deskriptor značajki, a široku upotrebu ima u računalnom vidu za izdvajanje značajki sa slike u svrhu detekcije objekata [3]. Načelna ideja rada HOG deskriptora je da se izgled i oblik objekata detekcije (slika 3.5) unutar slike mogu opisati distribucijom intenziteta gradijenata ili smjerova ruba na način da sliku podijelimo na prostorne ćelije gdje pojedinačne ćelije akumuliraju smjerove gradijenata. Svaka se ćelija sastoji od određenog broja piksela za koje je potrebno izračunati gradijente koji su prethodno grupirani za pojedine ćelije. Takav način spajanja histograma unutar ćelija se naziva HOG deskriptor.

Slika 3.5 Ulazna slika (izvor. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/)

Proces HOG detekcije objekata možemo opisati u sljedećim koracima:

1. predobrada podataka(slike),
2. izračun gradijenata,
3. izračun magnitude i orijentacije,
4. izračun histograma gradijenata u ćeliji 8x8,
5. normalizacija gradijenata u ćeliji 16x16,
6. generiranje značajki detekcije kompletne slike.

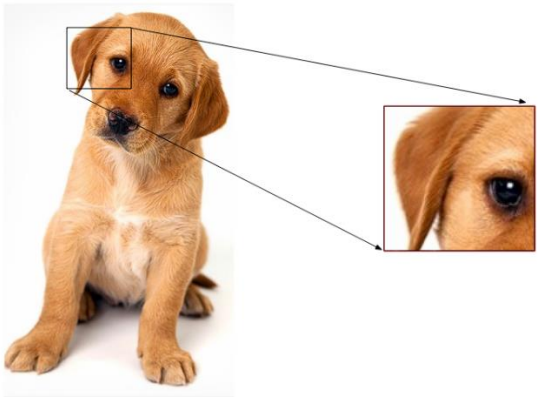
3.2.1 Predobrada slike

Predobrada podataka je ključan korak u bilo kojem procesu koji uključuje strojno učenje. Omjer širine i visine objekta detekcije, odnosno ulazne slike treba biti postavljen na 1:2 omjer (slika 3.6). Idealna veličina slike kod postavljenog omjera je 64x128p zbog kasnije podjele slike na 8x8 i 16x16 dijelove u svrhu izdvajanja ključnih značajki. Ovako postavljena veličina slike (64x128) nam olakšava daljnji postupak.

Slika 3.6 Postavljanje ulazne slike na 1:2 omjer (izvor. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/)

3.2.2 Izračun gradijenata (smjer x i y)

Sljedeći korak (slika 3.7) je izračun gradijenata za svaki piksel na slici. Gradijent je vektor koji opisuje promjenu skalarne funkcije definirane u prostoru. Prvi korak je izabrati mali dio slike i izračunati gradijente na njemu.



Slika 3.7 Izračun gradijenata na izabranom dijelu slike (izvor. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/)

Za odabrani dio slike potrebno je generirati tablicu (slika 3.8) vrijednosti piksela:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Slika 3.8 Tablica vrijednosti piksela

Odabrali smo piksel sa vrijednošću 85. Da bismo izračunali gradijent, odnosno promjenu u x-smjeru potrebno je od vrijednosti piksela sa desne strane oduzeti vrijednost piksela sa lijeve strane. Isto tako, za izračun gradijenta u y-smjeru od vrijednosti piksela s gornje strane oduzmemo vrijednost piksela s donje strane odabranog piksela s vrijednošću 85:

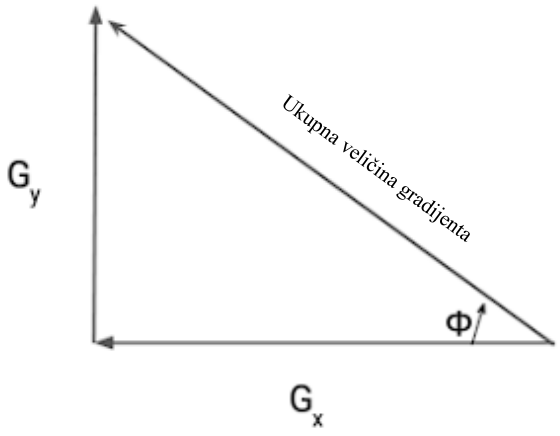
Gradijent (promjena) u X-smjeru (Gx): 89 - 78 = 11

Gradijent (promjena) u Z-smjeru (Gy): 68 - 56 = 8

Ovime smo izračunali gradijente za oba smjera odabranog piksela. Isti proces ponavljamo za svaki piksel na slici. Sljedeći korak je izračun magnitude i orijentacije pomoću dobivenih vrijednosti.

3.2.3 Izračun magnitude i orijentacije gradijenata

Koristeći gradijente dobivene u prethodnom koraku, potrebno je izračunati magnitudu i smjer za vrijednost svakog piksela na slici. Za ovaj korak (slika 3.9) koristimo Pitagorin poučak:



Slika 3.9 Ukupna veličina gradijenta

Gradijenti su zapravo katete ovog pravokutnog trokuta. U prethodnom koraku smo dobili vrijednosti gradijenata Gx = 11 i Gy = 8. Primijenimo li Pitagorin poučak (formula 3.1), možemo izračunati ukupnu veličinu(magnitudu) gradijenta:

$$\text{Ukupna veličina gradijenta} = \sqrt{[(\text{Gx})^2 + (\text{Gy})^2]} \tag{3.1}$$

$$= \sqrt{[(11)^2 + (8)^2]} = 13.6$$

Sljedeći korak je izračun orijentacije (smjera) za taj piksel (formula 3.2). Za kut Φ vrijedi:

$$\tan(\Phi) = \text{Gy} / \text{Gx} \tag{3.2}$$

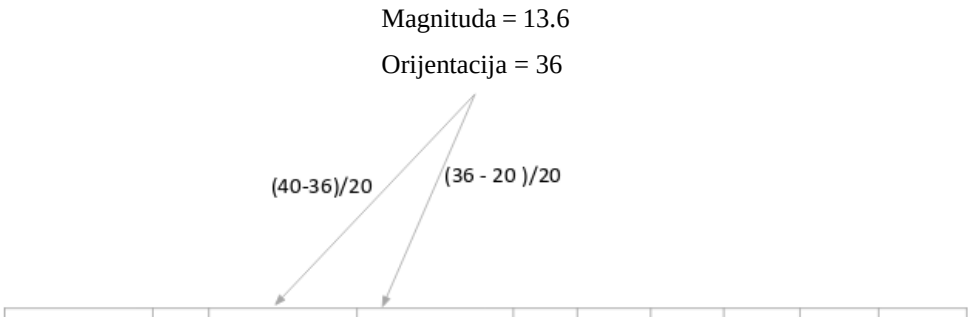
$$\Phi = \text{atan}(\text{Gy} / \text{Gx})$$

Unesemo li vrijednosti u formulu, dobijemo da je vrijednost orijentacije 36°. Istim postupkom za sve piksele sa slike izračunamo ukupnu veličinu gradijenata i orijentaciju (smjer). Slijedi generiranje histograma koristeći dobivene vrijednosti gradijenata i orijentacija.

3.2.4 Kreiranje histograma koristeći dobivene vrijednosti gradijenata i orijentacije

Histogram je zapravo funkcija koja nam pokazuje distribuciju frekvencija skupa kontinuiranih podataka. Postoji više načina za kreiranje histograma, a sljedeći način nam najbolje odgovara za daljnje korake.

Ukupnu veličinu gradijenta podijelimo na dva polja (slika 3.10) koja su najbliža vrijednosti dobivene orijentacije piksela na način da se veća „težina“ nalazi u polju koje je bliže točnoj vrijednosti orijentacije piksela:

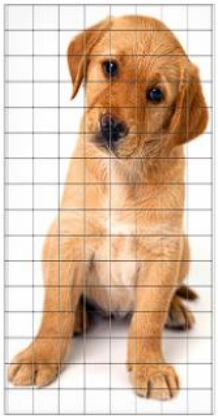


Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

Slika 3.10 tablica za kreiranje histograma

3.2.5 Izračun histograma gradijenata u ćeliji 8x8

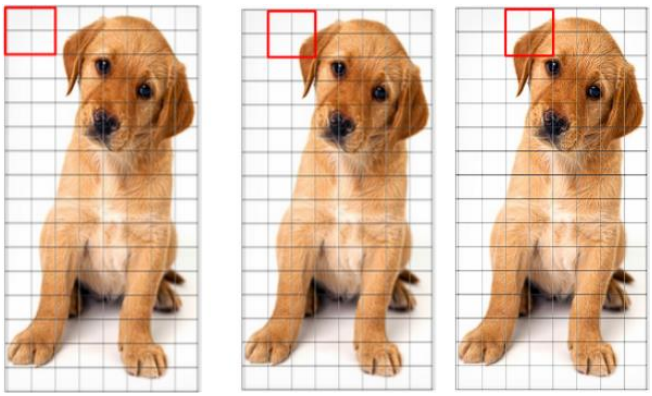
Histogrami kreirani u HOG deskriptoru nisu generirani za cijelu sliku. Umjesto toga, slika je podijeljena u 8x8 ćelije (slika 3.11) te je histogram gradijenata izračunat za svaku ćeliju. Time se dobe značajke za manje dijelove slike koje zapravo predstavljaju cijelu sliku. Isto tako, moguće je promijeniti veličinu ćelije na 16x16 ili 32x32. Ukoliko podijelimo sliku na 8x8 ćelije te generiramo histograme, dobit ćemo 9x1 matricu za svaku ćeliju.



Slika 3.11 Podjela slike na ćelije (izvor. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/)

3.2.6 Normalizacija gradijenata u 16x16 ćeliji

Iako već imamo HOG značajke kreirane za 8x8 ćeliju, gradijenti slike su osjetljivi na osvjetljenje, odnosno problem je u tome što su neki dijelovi slike jako svijetli u odnosu na ostale dijelove. To nije moguće ukloniti, ali možemo reducirati razlike u svjetlini dijelova slike normaliziranjem gradijenata sa 16x16 blokovima/ćelijama [4]. Primjer kreiranja 16x16 blokova (slika 3.12) na već napravljenim 8x8 blokovima:



Slika 3.12 kreiranje 16x16 blokova (izvor. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/)

Jednostavno kombiniramo četiri 8x8 ćelije da bismo napravili 16x16 ćeliju. Već od prije nam je poznato da svaka 8x8 ćelija ima 9x1 matricu za histogram. Tako svaka 16x16 ćelija ima četiri 9x1 matrice ili jednu 36x1 matricu. Da bismo normalizirali matricu potrebno je podijeliti svaku vrijednost sa korijenom zbroja vrijednosti blokova (formula 3.3). Matematički zapisano, za vektor V vrijedi:

$$V = [a1, a2, a3, \dots a36] \tag{3.3}$$

Prema tome, izračunamo (formula 3.4) korijen zbroja blokova:

$$k = \sqrt{(a1)^2 + (a2)^2 + (a3)^2 + \cdots (a36)^2} \tag{3.4}$$

i podijelimo sve vrijednosti vektora V sa vrijednošću k:

$$Normalizirani\ vektor = \left(\frac{a1}{k}, \frac{a2}{k}\ \frac{a3}{k} \dots \frac{a36}{k}\right)$$

Rezultat je normalizirani vektor veličine 36x1

3.2.7 Generiranje značajki detekcije kompletne slike

Nakon kreiranja značajki za 16x16 ćelije slike potrebno je dobiti ukupan broj značajki (slika 3.13) za cijelu sliku.

Slika 3.13 dobivanje ukupnog broja značajki (izvor. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/)

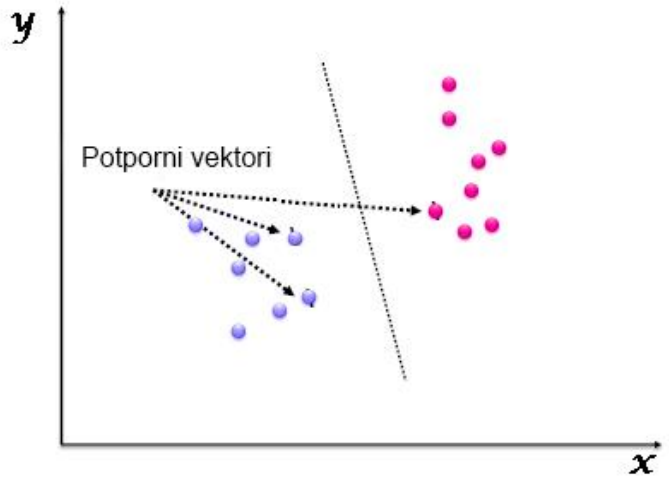
Za sliku 64x128p imamo 105 (7x15) blokova 16x16. Svaki od tih 105 blokova sadrži vektor veličine 36x1 kao značajku. Prema tome, ukupan broj značajki za cijelu sliku je:

$$105 \times 36 \times 1 = 3780$$

Sada generiramo HOG značajke za cijelu sliku i provjeravamo jesmo li na kraju dobili isti broj značajki.

3.3 Support Vector Machine

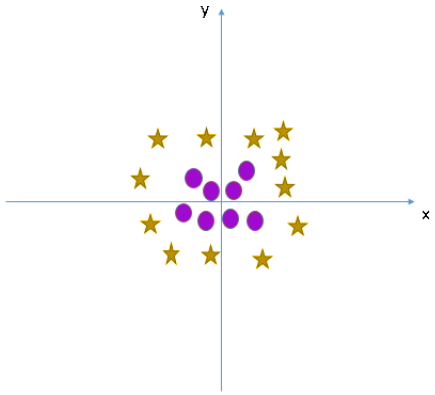
SVM je nadzirani algoritam strojnog učenja koji se koristi za klasifikaciju ili regresiju (u ovom slučaju klasifikaciju) [5]. U SVM algoritmu se svaka podatkovna stavka crta kao točka u n-dimenzionalnom prostoru čije koordinate predstavljaju značajke te stavke. Zatim SVM provodi test klasifikacije kreiranjem hiper-ravnine u 2D ili 3D prostoru na način da se podatkovne stavke (točke) jedne kategorije nalaze s jedne strane, a podatkovne stavke druge kategorije s druge strane hiper-ravnine (slika 3.14). Od mogućih hiper-ravnina, SVM pokušava pronaći onu koja najbolje razdvaja dvije kategorije na način da maksimizira udaljenost od točaka obje kategorije. Tu udaljenost nazivamo marginom, a točke koje leže na marginama zovu se potporni vektori.



Slika 3.14 SVM, razdvajanje podatkovnih stavki različitih kategorija

Da bismo pronašli optimalnu hiper-ravninu, SVM zahtjeva trening ili ulazne podatkovne stavke (točke) koje već sadrže labele (oznake) ispravnih kategorija. Zbog toga kažemo da je SVM nadzirani algoritam strojnog učenja. Najveće prednosti SVM-a su jednostavnost, laka implementacija i upotreba. Također, efektivan je kod treniranja s manjom količinom podataka.

Problem kod SVM-a nastaje kada se podatkovne stavke (točke) ne mogu razdvojiti hiper-ravninom u 2D prostoru (slika 3.15).

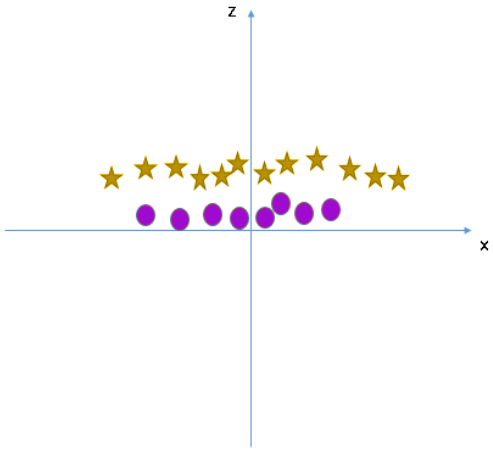


Slika 3.15 Prikaz nemogućnosti razdvajanja podataka u 2D prostoru

Ovaj problem rješavamo uvođenjem treće dimenzije z (slika 3.16), uz postojeće x i y. Uz to dobivamo i novu funkciju:

$$z = x^2 + y^2 \tag{3.5}$$

Prikaz podatkovnih točaka u 3D prostoru na osi x i z:

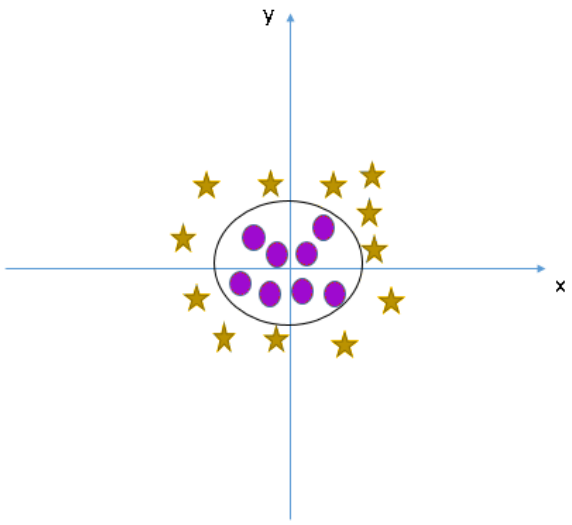


Slika 3.16 prikaz podatkovnih stavki u 3D prostoru

Dolazimo do sljedećih zaključaka:

- sve vrijednosti za „z“ su uvijek pozitivne s obzirom da su rezultat zbroja kvadrata x i y vrijednosti
- u izvornom 2D prostoru ljubičaste podatkovne točke nalaze se bliže ishodištu na x i y osi pa možemo lako zaključiti da je za njih vrijednost „z“ manja i da će se u 3D prikazu nalaziti ispod žutih točaka koje u 3D prostoru poprimaju veću vrijednost „z“

Pod ovim okolnostima, SVM lako pronalazi optimalnu hiper-ravninu (slika 3.17) između ovih dviju kategorija podatkovnih stavaka:



Slika 3.17 optimalna hiper-ravnina razdvajanja podatkovnih stavki

U ovakvim slučajevima, gdje je nemoguće pronaći hiper-ravninu u 2D prostoru, SVM algoritam automatski poziva „kernel funkciju“ koja 2D prostor prikazuje u tri dimenzije i time omogućuje separaciju podatkovnih točaka.

4. Strojno učenje

Strojno učenje je grana umjetne inteligencije koja se bavi oblikovanjem algoritma koji svoju učinkovitost poboljšava na temelju empirijskih podataka, bez eksplicitnog programiranja za svaki pojedinačni zadatak [6]. Strojno učenje je zbog svojih mnogobrojnih mogućnosti primjene jedno od najaktivnijih područja informatičke znanosti. Standardni postupak korištenja strojnog učenja bazira se na treningu, odnosno korištenju poznatih podataka u svrhu stvaranja funkcionalnog modela sa prilagođenim parametrima. Dobiveni model se koristi na novim, do tada nepoznatim podacima u svrhu rješavanja zadane problematike. Obrađivanjem i skupljanjem sve više podataka, model se s vremenom poboljšava te se greške prilikom izvođenja svode na minimum. Osnovna ideja strojnog učenja je da se modeli razvijaju na temelju podataka, a ne na temelju unaprijed programiranog skupa pravila. Dakle, sustavi strojnog učenja mogu prilagođavati svoje modele prema novim podacima koje dobivaju, što im omogućuje da postanu precizniji i sposobniji rješavati složenije zadatke.

Tri osnovna pristupa strojnom učenju su:

- nadzirano učenje - klasifikacija i regresija,
- nenadzirano učenje - grupiranje podataka temeljem sličnosti (engl. *clustering*), estimacija distribucije podataka (engl. *density estimation*), smanjenje dimenzionalnosti, učenje asocijativnih pravila (association rule learning),
- pojačano učenje.

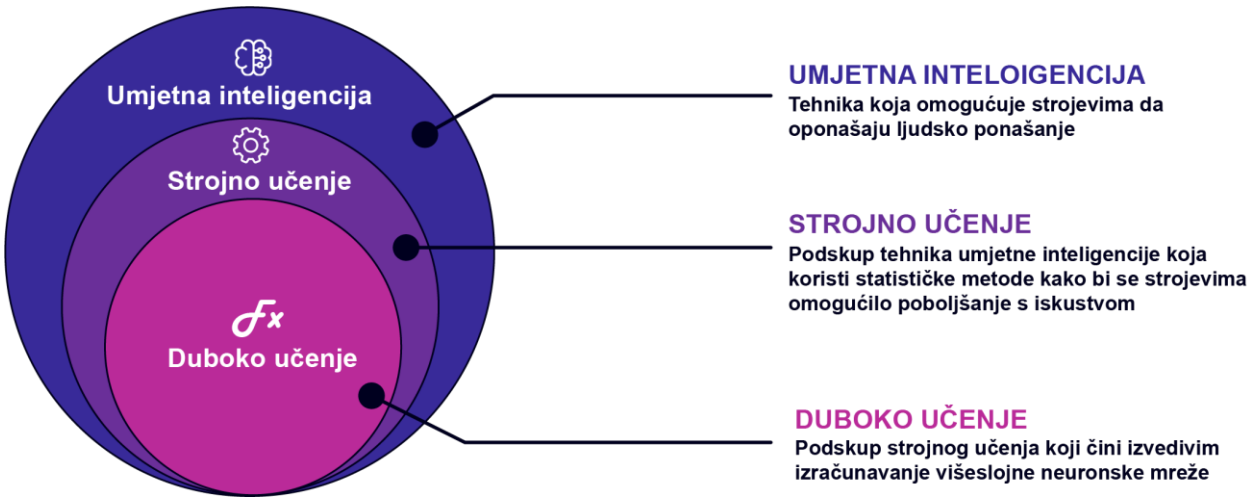
Nadzirano učenje (engl. *Supervised learning*) je pristup stvaranju umjetne inteligencije na način da računalni algoritam obučava na ulaznim podacima s dodijeljenim oznakama (engl. *Labels*). Na temelju ulaznih primjera, algoritam kasnije sam dodjeljuje oznake te detektira željene objekte. Kod nenadziranog učenja algoritam sam detektira objekte bez primjera sa dodijeljenim oznakama na temelju zadanih karakteristika. Pojačano učenje je obuka modela strojnog učenja da sam donosi niz odluka te metodom pokušaja i pogreške stekne iskustvo i poboljša učinkovitost. Kod ovog pristupa je definiran krajnji rezultat, ali ne i striktno najbolji način kako ga postići.

Primjena strojnog učenja je sveprisutna u modernom svijetu, primjerice u područjima kao što su sustavi preporuka, prepoznavanje uzoraka, obrada prirodnog jezika, autonomni automobili, medicinska dijagnostika, financijska analiza i druga područja.

4.1 Duboko učenje

Duboko učenje je područje strojnog učenja koje se temelji na konceptu dubokih neuronskih mreža [7]. Ova tehnika omogućava modelima da nauče reprezentacije podataka putem više slojeva, što omogućava apstraktniju i hijerarhijsku reprezentaciju značajki. Duboko učenje je postalo posebno snažno u rješavanju problema koji zahtijevaju veliku količinu podataka i složene uzorke.

Osnovna ideja dubokog učenja je da se sustavu pružaju slojevi ulaznih podataka, a mreža se uči kako prepoznati uzorke i zakonitosti u tim podacima kroz više slojeva obrade. Svaki sloj mreže uči značajke podataka koje su sve složenije i apstraktnije kako se krećemo prema izlaznom sloju mreže, koji konačno donosi odluku o klasifikaciji, predviđanju ili drugom zadatku.
Osnovna razlika je dodatno objašnjena sljedećom slikom:



Slika 4.1 Slikovni prikaz razlike strojnog i dubokog učenja

Duboko učenje koristi različite slojeve neurona za postupno prepoznavanje značajki na ulaznim podacima. U praksi, niži slojevi mreže identificiraju rubove fotografije, dok viši slojevi identificiraju rubove ljudskog lica.

Duboko učenje se često koristi u zadacima koji zahtijevaju obradu velike količine podataka, kao što su prepoznavanje slika, prepoznavanje govora, obrada prirodnog jezika, preporučiteljski sustavi i druge. Neki od najčešće korištenih algoritama dubokog učenja su konvolucijske neuronske mreže, rekurentne neuronske mreže, generativne suparničke mreže i potpuno povezane unaprijedne neuronske mreže.

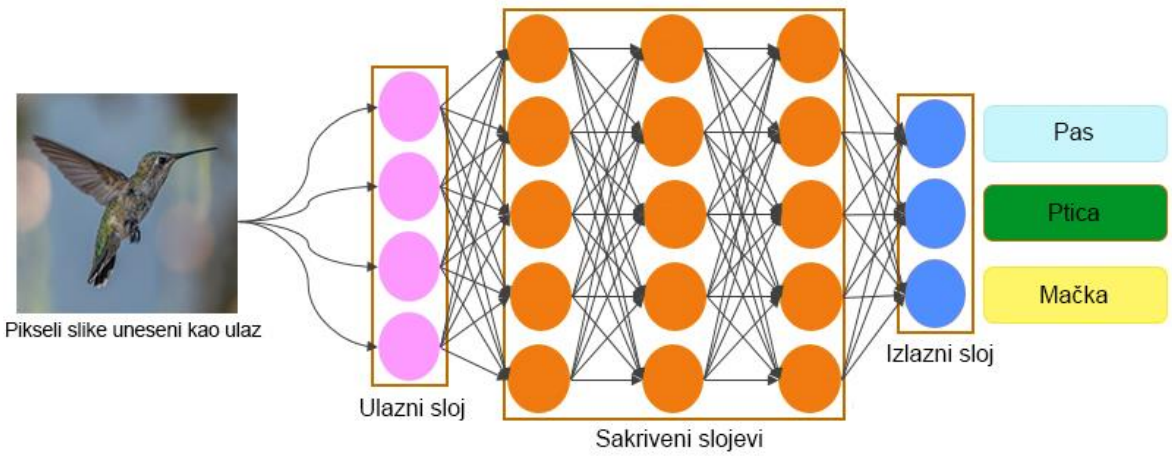
Duboko učenje je postalo sve popularnije u posljednjih nekoliko godina zahvaljujući povećanju dostupnosti velikih količina podataka, moćnijim računalnim resursima i napretku u algoritmima. Danas, duboko učenje se koristi u mnogim industrijskim i akademskim projektima, a smatra se jednim od ključnih tehnologija u umjetnoj inteligenciji.

Duboko učenje uključuje različite arhitekture neuronskih mreža, koje se razlikuju po načinu na koji su slojevi povezani i obrađuju podatke. Neki od najčešće korištenih arhitektura dubokog učenja su:

- duboke neuronske mreže (engl. *deep neural networks*)
- duboke mreže vjerovanja (engl. *deep belief networks*)
- konvolucijske neuronske mreže (engl. *convolution neural networks*)
- ResNet (engl. *Residual Neural Networks*)
- MobileNetV2 arhitektura

4.1.1 Konvolucijske neuronske mreže (CNN)

Najčešće su korištene za analizu fotografija, ali i drugih vrsta podataka te klasifikaciju podatkovnih unosa. Ukratko, CNN (slika 4.2) su umjetne neuronske mreže koje su specijalizirane za odabir ili detekciju značajki. Razlika u odnosu na ostale neuronske mreže je u tome što CNN, uz ostale slojeve, ima i skrivene slojeve koji se nazivaju konvolucijski slojevi.



Slika 4.2 Konvolucijska neuronska mreža

Slojevi konvolucijske mreže

- Konvolucijski sloj

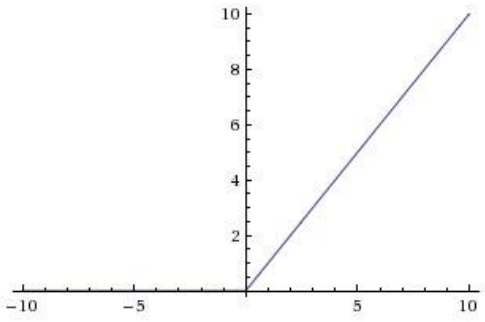
  Poput ostalih slojeva, konvolucijski slojevi primaju ulazni podatak, te ga nakon obrade šalju na sljedeći sloj. Pod obradom podrazumijevamo detekciju uzoraka na podatkovnom unosu. Na svakom konvolucijskom sloju potrebno je definirati broj filtara koji taj sloj sadrži. Na početnim slojevima ti filteri su jednostavni te uglavnom služe za detekciju geometrijskih objekata poput krugova i kvadrata. Dublji slojevi su sofisticiraniji te imaju mogućnost detekcije konkretnih objekata, npr. dijelove lica i slično. Konvolucijske neuronske mreže postaju praktičnije za veće (složenije) slike, te se zbog manjeg broja težina smanjuje mogućnost pretjeranog ugađanja (overfitting) na neku bazu, u odnosu na korištenje samo potpuno povezanih slojeva.

- Slojevi sažimanja (engl. *pooling layers*)

  Skriveni slojevi koji se dolaze nakon konvolucijskih slojeva. U ovom sloju se sve vrijednosti piksela u pojedinim mapama značajki sažimaju da bismo dobili smanjenu rezoluciju mape. Samim time, ako imamo manje piksela, imamo manje parametara i jednostavniji proračun. Ovim korakom moguće je otkrivanje objekata bez obzira na položaj na slici, odnosno, dobivamo na fleksibilnosti konvolucijske neuronske mreže.

- ReLU slojevi (engl. *Rectified linear unit*)

  ReLU sloj je zapravo ne-negativna aktivacijska funkcija (slika 4.3) koja na izlazu daje „0“ ukoliko primi negativni unos, a za svaku pozitivnu vrijednost „x“ daje istu tu vrijednost na izlaz. Jednostavno rečeno, ova funkcija u procesu detekcije ostavlja podatkovne stavke koje možda nisu „prošle“ sve filtere iz prethodnih slojeva ali bi kasnije mogle ipak biti relevantne. Funkcija je opisana izrazom: $f(x)=\max(0,x)$



Slika 4.3 Funkcija ReLU sloja

- Potpuno povezani slojevi (engl. *fully connected layers*)

  Dolaze na kraju skrivenih slojeva konvolucijske neuronske mreže. U ovom sloju svaki čvor prima izlaz svakog čvora prethodnog sloja te se sve značajke konvolucijske neuronske mreže kombiniraju u svrhu ostvarivanja preciznijih rezultata. Finalno, mape 2D značajki se konvertiraju u 1D vektor značajki.

4.1.2 Duboke neuronske mreže

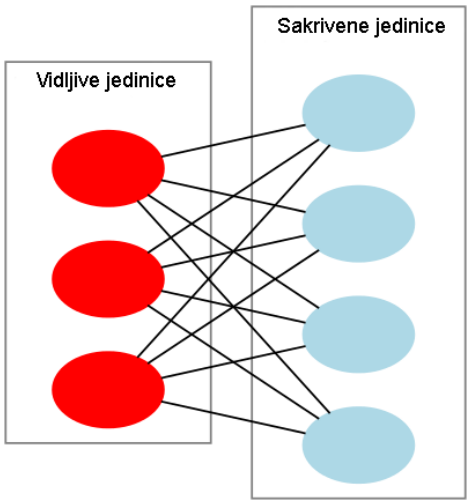
Duboke neuronske mreže (engl. *Deep Neural Networks - DNN*) su složene neuronske mreže koje se sastoje od više slojeva neurona. U usporedbi s tradicionalnim neuronskim mrežama koje se sastoje samo od jednog ili dva sloja, DNN su sposobne naučiti složenije značajke i obaviti složenije zadatke.

Da bismo razumjeli princip rada dubokih neuronskih mreža, potrebno je znati osnove rada ograničenog Boltzmann-ovog stroja (RBM).

RBM je generativni model strojnog učenja koji se sastoji od vidljivih i skrivenih varijabli koje su međusobno povezane težinama. RBM koristi neuronske mreže s jednim skrivenim slojem, ali se sastoji samo od dva sloja: vidljivog i skrivenog. Svi neuroni u jednom sloju su povezani s neuronima u drugom sloju. RBM ima ograničenja na svoje težine i matrične operacije koje se koriste za izračunavanje aktivacije neurona. Ograničenja se primjenjuju na težine kako bi se ograničila složenost modela i kako bi se olakšalo treniranje. RBM se koristi za učenje reprezentacija značajki i kao dio složenijih neuronskih mreža. RBM se trenira primjenom tehnika poput gradijentnog spusta i kontrastivne divergencije. Kada se model nauči, može se koristiti za generiranje novih primjera podataka ili za rekonstrukciju postojećih. RBM se često koristi u problemima poput preporučivanja i prepoznavanja obrazaca. Iako je RBM relativno jednostavan

model, može se koristiti kao osnovni gradivni blok za složenije modele, poput dubokih mreža vjerovanja (engl. *Deep Belief Networks - DBN*).

Kod ove varijante Boltzmann-ovog stroja nema komunikacije između čvorova unutar iste grupe (slika 4.4). Čvorovi u mreži donose stohastičke odluke. Ovisno o ulaznim podacima, mijenja se struktura ulaza, a funkcije aktiviranja obrađuju izlaz čvorova.



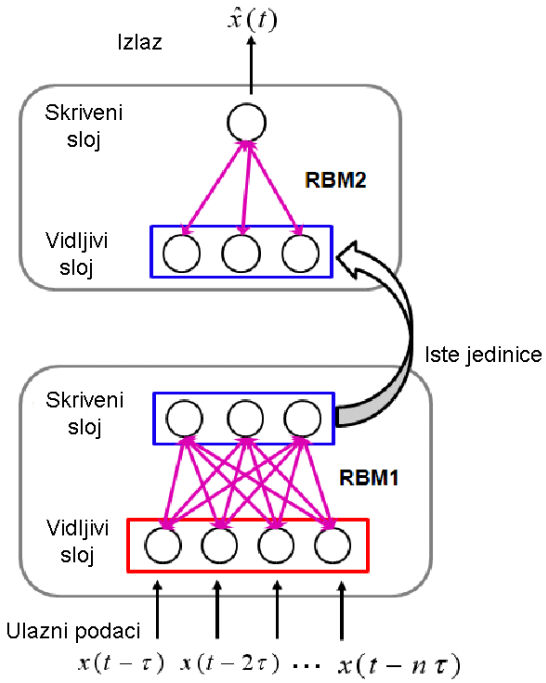
Slika 4.4 Dijagram ograničenog Boltzmann-ovog stroja s tri vidljive jedinice i četiri skrivene jedinice

4.1.3 Duboke mreže vjerovanja

Duboke mreže vjerovanja (engl. *Deep Belief Networks*) su složene generativne neuronske mreže koje su osmišljene za obradu i učenje značajki u podacima. DBN (slika 4.5) su razvijene s ciljem modeliranja složenih distribucija podataka, posebice u područjima poput prepoznavanja objekata, prepoznavanja govora i obrade prirodnog jezika.

Na duboke mreže vjerovanja možemo gledati kao skup RBM-ova (engl. *Restricted Boltzman machine*) gdje je skriveni sloj jednog RBM-a zapravo vidljiv sloj RBM-a "iznad" njega. Trening RBM-a se izvodi slijedom:

1. prvi RBM je treniran za čim precizniju rekonstrukciju ulaznog podatka,
2. skriveni sloj prvog RBM-a je treniran kao vidljiv sloj drugog RBM-a koji je treniran korištenjem izlaza iz prvog RBM-a,
3. proces se ponavlja dok se ne nauče svi slojevi mreže.



Slika 4.5 Duboka mreža vjerovanja

Kod dubokih mreža vjerovanja svaki RBM sloj uči cjelokupni podatkovni ulaz u mrežu. Za razliku od ostalih mreža, DBN radi na globalnoj razini poboljšavanjem ulaznog podatka s ciljem dobivanja preciznog modela, poput leće za kameru koja polako fokusira fotografiju za najbolji rezultat. Nakon inicijalnog treninga, RBM kreira model koji može detektirati povezane uzorke u podacima. Za dovršiti trening, potrebno je pridružiti oznake (labele) uzorcima i usavršiti (fine tuning) mrežu nadzornim učenjem. Da bismo to učinili, trebamo manji skup označenih uzoraka tako da možemo povezati značajke sa imenima.

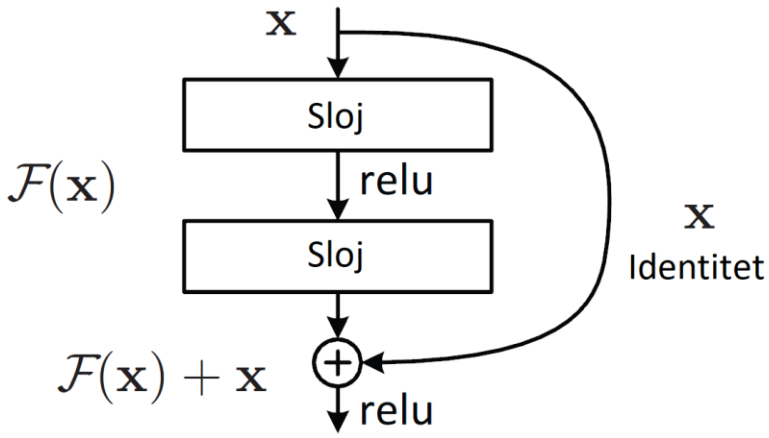
DBN se obično koriste za generativne zadatke, kao što su generiranje novih primjera podataka i rekonstrukcija ulaznih podataka, ali se mogu koristiti i za diskriminativne zadatke, kao što su klasifikacija podataka. DBN su široko primijenjene u različitim područjima, uključujući računalni vid, obradu govora, prirodni jezik i genetsku analizu.

Prednosti mreža dubokih vjerovanja:

- mali skup označenih podataka,
- relativno kratko vrijeme treniranja mreže,
- velika preciznost u usporedbi sa „plitkim“ neuronskim mrežama.

4.1.4 ResNet

ResNet (engl. *Residual Neural Network*) je arhitektura dubokog neuronskog mrežnog modela koji je postao popularan u području računalnog vida [8]. Razvio ga je Microsoftov istraživački tim 2015. godine i pokazao se vrlo uspješnim u rješavanju problema dubokog učenja. Glavna ideja iza ResNet arhitekture je koncept "rezidualnih blokova". Uobičajeni problemi pri treniranju dubokih neuronskih mreža su postupno smanjenje performansi s povećanjem dubine mreže i problem nestajućeg gradijenta. Rezidualni blokovi omogućuju mreži da "preskoči" slojeve kako bi se riješili ti problemi. Rezidualni blok se sastoji od ulaznih i izlaznih veza, kao i "rezidualne veze" koja preskače jedan ili više slojeva (slika 4.6). Na taj način, ako se slojevi ne uklapaju u model, rezidualna veza pruža mogućnost prenošenja informacija s jednog sloja na drugi, što olakšava učenje dubokih mreža.

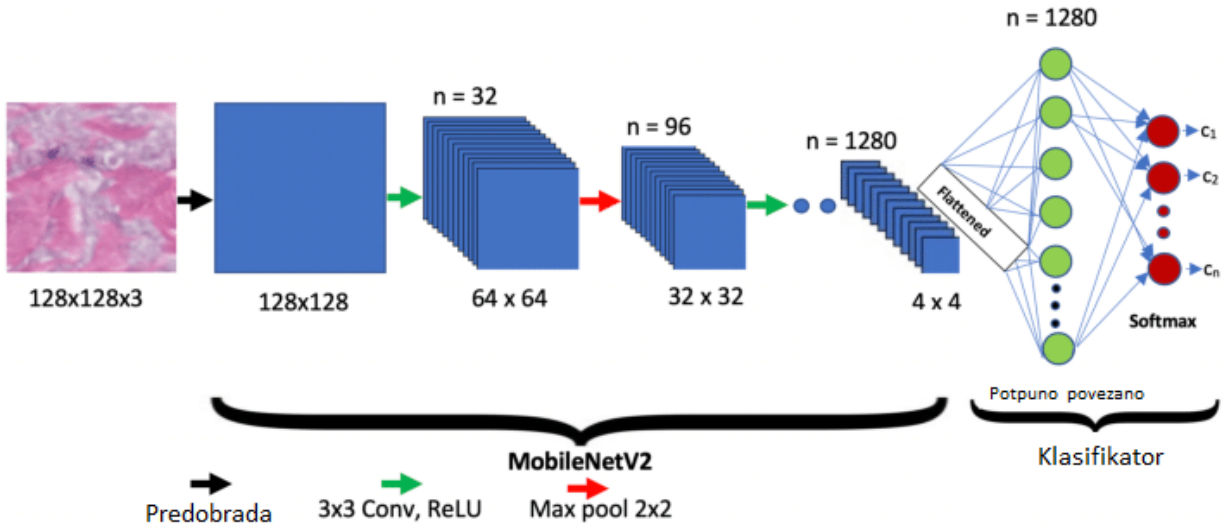


Slika 4.6 Osnovna struktura DRN-a (izvor. https://aditib2409.medium.com/image-classification-with-resnets-in-pytorch-eff4f75c741d)

ResNet arhitektura ima različite varijante, uključujući ResNet-18, ResNet-34, ResNet-50, ResNet-101 i ResNet-152. Ove varijante se razlikuju u broju slojeva i kompleksnosti, pri čemu veće varijante imaju veći broj parametara i dubinu. ResNet arhitektura postala je važan alat u računalnom vidu i koristi se za zadatke kao što su klasifikacija slika, detekcija objekata, segmentacija slika i druge zadatke koji zahtijevaju duboko učenje.

4.1.5 MobileNetV2

MobileNetV2 je naziv za arhitekturu dubokih neuronskih mreža (slika 4.7) koja je razvijena za zadatak prepoznavanja objekata u slikama i smanjenje zahtjeva za resursima kao što su memorija i računalna snaga [9]. To je druga verzija MobileNet modela, pri čemu je "Mobile" u nazivu povezan s činjenicom da je arhitektura optimizirana za mobilne uređaje i druge resursno ograničene okruženja.



Slika 4.7 Arhitektura MobileNetV2 mreže (izvor. https://www.researchgate.net/figure/The-proposed-MobileNetV2-network-architecture_fig1_350152088)

MobileNetV2 je posebno razvijen kako bi se poboljšala performansa u usporedbi s prethodnim verzijama, poput originalnog MobileNeta. On koristi različite tehnike kao što su "inverted residuals" (invertirani residuali) i "linear bottlenecks" (linearni suženjci) kako bi se postigla ravnoteža između preciznosti i brzine.

Glavna karakteristika MobileNetV2 je njegova sposobnost da postigne visoku točnost prepoznavanja objekata uz minimalan broj parametara i operacija. Ovo ga čini pogodnim za primjenu na uređajima s ograničenim resursima kao što su pametni telefoni, tableti, ugrađeni sustavi i drugi mobilni uređaji. MobileNetV2 se često koristi za različite zadatke računalnog vida kao što su detekcija objekata, klasifikacija slika i segmentacija objekata.

4.2 mAP (engl. *mean Average Precision*)

Prosječna srednja preciznost (mAP) je popularna mjera koja se koristi za procjenu ukupne učinkovitosti modela detekcije objekata i segmentacije [10]. Računa se tako da se izračuna prosječna preciznost (AP) za svaku klasu, a zatim se uzme srednja vrijednost tih AP vrijednosti za sve klase.

Proces izračuna mAP:

1. Priprema podataka:
    - potreban skup podataka s točnim oznakama (ground truth) za svaki objekt u slikama koje model analizira
    - za svaku sliku potreban je popis predviđenih oznaka (predictions) koje je generirao vaš model, zajedno s njihovim pripadajućim ocjenama pouzdanosti
2. Postavljanje pragova:
    - postaviti prag IoU-a (omjer presjeka i unije) koji će se koristiti za određivanje točnosti predikcija. Uobičajeni prag je 0.5, ali možete koristiti i druge vrijednosti ovisno o specifičnostima vašeg problema
3. Sortiranje predikcija:
    - za svaku klasu objekata sortirati predikcije prema ocjeni pouzdanosti silazno. Ovo sortiranje je potrebno za izračun preciznosti i odziva (recall) na različitim pragovima IoU-a
4. Izračunavanje preciznosti i odziva:
    - za svaku klasu, izračunati preciznost i odziv (recall) kako bismo stvorili krivulju preciznost-odziv (PR krivulja)
    - preciznost se izračunava kao broj točno predviđenih pozitivnih primjera podijeljen s ukupnim brojem predviđenih pozitivnih primjera na tom IoU pragu
    - odziv se izračunava kao broj točno predviđenih pozitivnih primjera podijeljen s ukupnim brojem stvarnih pozitivnih primjera na tom IoU pragu

5. Izračunavanje površine ispod krivulje (AP):
    - izračunati površinu ispod krivulje preciznost-odziv (AP) za svaku klasu koristeći preciznost i odziv iz prethodnog koraka. To se može učiniti interpolacijom izračunatih vrijednosti ili korištenjem površinske metode (npr. trapezoidne formule)
    - AP daje učinkovitost modela za tu klasu, uzimajući u obzir preciznost i odziv na različitim pragovima IoU-a
6. Izračunavanje mAP:
    - konačno, izračunati prosječnu srednju preciznost (mAP) uzimajući prosjek AP vrijednosti za sve klase
    - formula (4.1) za izračun mAP-a je:

$$\mathbf{mAP = (AP\_1 + AP\_2 + ... + AP\_n) / n} \quad (4.1)$$

gdje je AP_1, AP_2, ..., AP_n su vrijednosti AP za svaku klasu, a n je ukupan broj različitih klasa

Veće vrijednosti mAP-a ukazuju na bolju učinkovitost modela u otkrivanju i lokalizaciji objekata iz različitih klasa. Sažeto, omjer presjeka i unije (IoU) koristi se za mjeru točnosti pojedinačnih predikcija, dok prosječna srednja preciznost (mAP) pruža ukupnu mjeru učinkovitosti modela detekcije objekata i segmentacije. Ove metode se široko koriste za procjenu i usporedbu učinkovitosti različitih algoritama računalnog vida. mAP formula je bazirana na sljedećim pod vrijednostima koje se izračunavaju iz matrice zabune:

- Matrica zabune (engl. *Confusion Matrix*):

Matrica zabune je tablica (slika 4.8) koja se koristi za prikazivanje performansi klasifikacijskog modela. U slučaju problema detekcije objekata ili segmentacije, može se prilagoditi kako bi se uzela u obzir evaluacija za različite klase objekata. Matrica zabune uspoređuje stvarne oznake (ground truth) s predviđenim oznakama (predictions) i daje četiri osnovne vrijednosti:

- True Positives (TP): Broj točno predviđenih pozitivnih primjera (pravilno prepoznatih objekata).

- True Negatives (TN): Broj točno predviđenih negativnih primjera (pravilno prepoznatih nepostojećih objekata).
- False Positives (FP): Broj pogrešno predviđenih pozitivnih primjera (objekti su krivo prepoznati tamo gdje ne postoje).
- False Negatives (FN): Broj pogrešno predviđenih negativnih primjera (stvarni objekti su krivo prepoznati kao nepostojeći).

Matrica zabune

		Stvarno	
		Pozitivno	Negativno
Predviđeno	Pozitivno	TP	FP
	Negativno	FN	TN

Slika 4.8 Matrica zabune

- Osjetljivost (engl. *recall*, ili TPR - True Positive Rate):

Osjetljivost (recall) mjeri sposobnost modela da pravilno prepoznaje stvarno pozitivne primjere određene klase. Matematički se izračunava (formula 4.2. kao omjer broja točno predviđenih pozitivnih primjera (TP) i ukupnog broja stvarnih pozitivnih primjera (TP + FN):

$$\textbf{Osjetljivost} = TP / (TP + FN) \quad (4.2)$$

Visoka vrijednost osjetljivosti ukazuje na to da je model učinkovit u prepoznavanju stvarnih pozitivnih primjera, što je posebno važno kod zadataka kao što su detekcija objekata gdje je važno minimizirati FN (krivo negativne) pogreške.

- Preciznost (engl. *precision*):

Preciznost mjeri koliko je model precizan u prepoznavanju pozitivnih predikcija za određenu klasu. Matematički se izračunava (formula 4.3) kao omjer broja točno predviđenih pozitivnih primjera (TP) i ukupnog broja predviđenih pozitivnih primjera (TP + FP):

$$\textbf{Preciznost} = \textbf{TP} / (\textbf{TP} + \textbf{FP}) \tag{4.3}$$

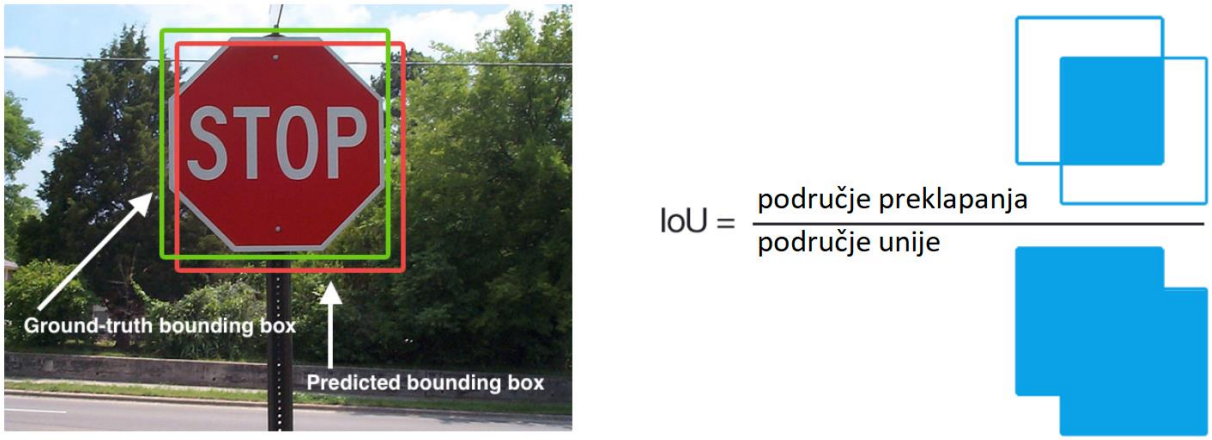
Visoka vrijednost preciznosti ukazuje na to da model ima malo FP (krivo pozitivnih) pogrešaka, što je važno u situacijama gdje želimo biti sigurni u točnost predikcija. U kombinaciji, osjetljivost i preciznost često se koriste za ocjenu modela u odnosu na određenu klasu. Postoji često obrnuti odnos između ovih dviju mjera, što se naziva trade-off koji ovisi o pragu detekcije. Kada pokušavamo povećati jednu od ovih mjera, druga može pasti, pa je važno pronaći balans koji najbolje odgovara specifičnim potrebama problema detekcije objekata ili segmentacije.

- Omjer presjeka i unije (IoU):

Omjer presjeka i unije (IoU) je mjera preklapanja između predviđene omeđujuće okvira (ili regije) i omeđujućeg okvira (ili regije) stvarnih objekata (slika 4.9). Često se koristi za procjenu točnosti modela detekcije objekata i segmentacije. IoU računamo sljedećim koracima:

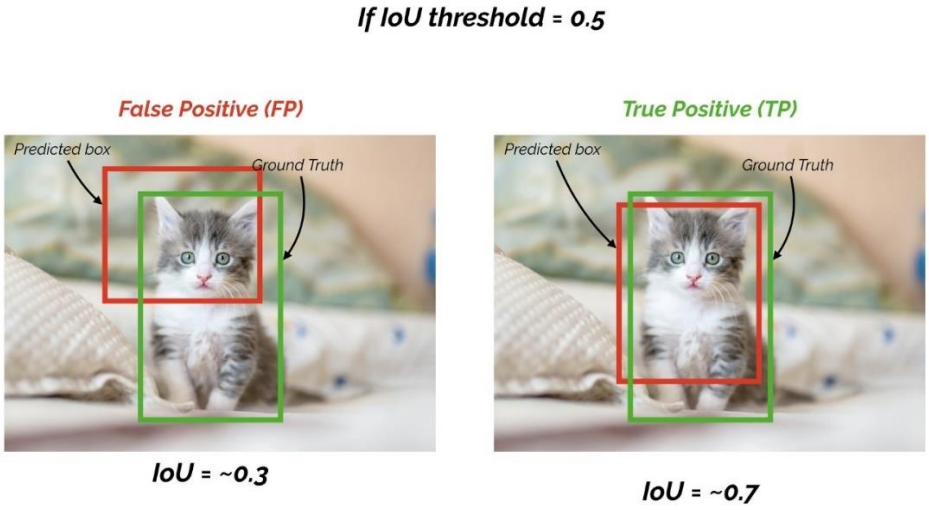
1. Odredite koordinate regije presjeka između predviđenog omeđujućeg okvira (P) i stvarnog omeđujućeg okvira (GT). Regija presjeka je područje koje je zajedničko oba okvira.
2. Odredite koordinate regije unije između P i GT. Regija unije je ukupno područje koje pokrivaju predviđeni i stvarni omeđujući okviri.
3. Izračunajte površinu presjeka (brojnik IoU-a) i površinu unije (imenitelj IoU-a).
4. Na kraju, IoU se izračunava dijeljenjem površine presjeka s površinom unije:

$$\text{IoU} = \text{Površina presjeka} / \text{Površina unije}$$



Slika 4.9 Vizualni prikaz IoU (izvor. https://www.v7labs.com/blog/mean-average-precision)

IoU vrijednosti se kreću od 0 do 1 (slika 4.10), gdje 0 označava da nema preklapanja između predviđenog i stvarnog područja, a 1 označava savršeno podudaranje.



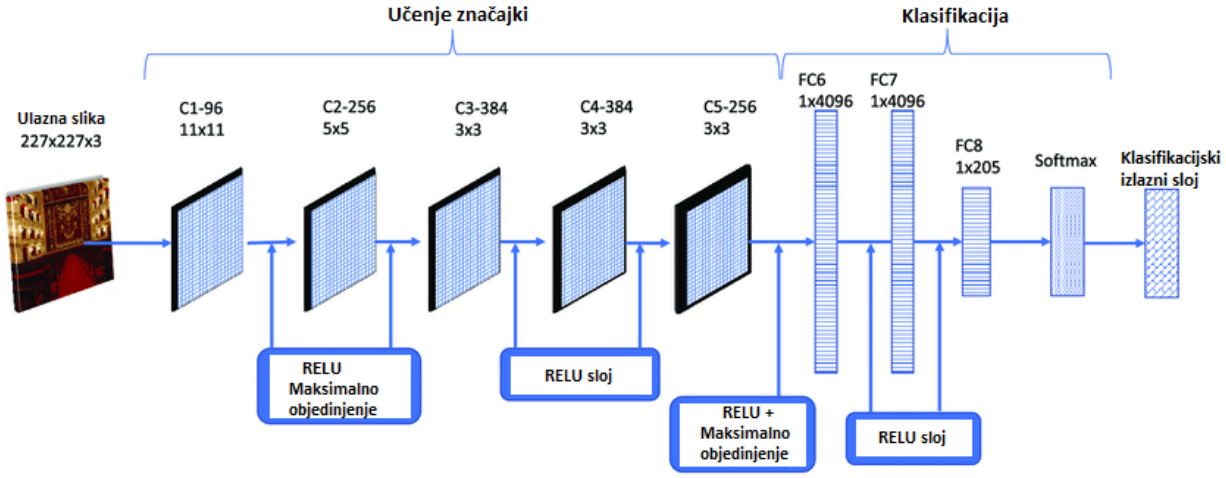
Slika 4.10 IoU preklapanje (izvor. https://www.v7labs.com/blog/mean-average-precision)

IoU se često koristi kao prag (engl. *threshold*) za određivanje je li predviđeni omeđujući okvir istiniti pozitivni rezultat (TP) ili lažni pozitivni rezultat (FP). Ako IoU između predviđenog i stvarnog okvira premašuje unaprijed definirani prag (npr. 0,5), smatra se TP-om; inače, smatra se FP-om.

4.2.1 Caffe model

Caffe model je model strojnog učenja koji je izrađen pomoću Caffe frameworka. Caffe (engl. *Convolutional Architecture for Fast Feature Embedding*) je popularan open-source framework za duboko učenje koji se koristi za razvoj i trening neuronskih mreža. Caffe model je specifična implementacija neuronske mreže koja je trenirana na određenom skupu podataka radi obavljanja određenih zadataka. Model se sastoji od arhitekture neuronske mreže, tzv. težina koje su naučene tijekom procesa učenja i drugih parametara koji su potrebni za izvođenje predikcija (slika 4.11).

Caffe modeli su pogodni za razne primjene, kao što su prepoznavanje objekata u slikama, klasifikacija, detekcija i segmentacija objekata, prepoznavanje lica, obrada prirodnog jezika i druge zadatke strojnog učenja. Kako bi se Caffe model koristio, potrebno je učitati model u odgovarajući softver ili biblioteku za duboko učenje i koristiti ga za predviđanje ili daljnje obučavanje.



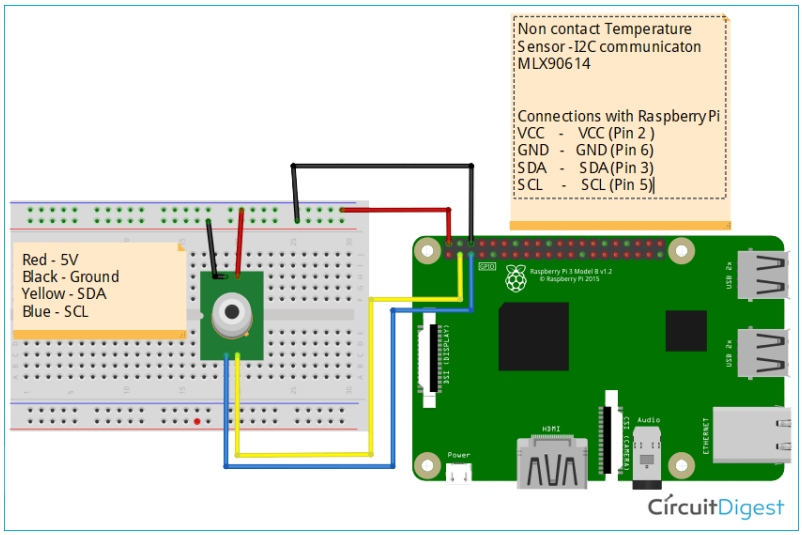
Slika 4 11. Konvolucijska neuronska mreža bazirana na Caffe modelu (izvor. https://www.researchgate.net/figure/CNN-places-architecture-design-based-on-the-pre-trained-Caffe-model-Layers-fc7-and_fig1_332665758)

Prilikom korištenja modula duboke neuronske mreže sa Caffe modelima, potrebne su dvije datoteke:

- .protox datoteka – definira model arhitekture
- .caffemodel datoteka – sadrži težine za slojeve

5. Mjerenje temperature lica

S obzirom na današnje okolnosti, automatizirani sustav očitanja temperature s lica našao je široku primjenu u svakodnevnom životu. Za razliku od detekcije lica i zaštitnih maski, proces očitanja temperature lica je ipak nešto jednostavniji. U našem slučaju, sustav za mjerenje temperature bit će izveden pomoću Raspberry Pi računala te MLX90614 infracrvenog senzora (slika 5.1).



Slika 5.1 Shema spoja infracrvenog senzora sa Raspberry Pi računalom (izvor. https://circuitdigest.com/microcontroller-projects/iot-based-contactless-body-temperature-monitoring-using-raspberry-pi-with-camera-and-email-alert)

Termalne kamere kao i standardne koriste svjetlost za snimanje. Razlika je u tome što termalne kamere detektiraju i filtriraju svjetlost te bilježe same infracrveno područje elektromagnetskog spektra, a ne vidljivo područje. MLX90614 je jednostavno izveden, jeftin i jedan od najčešće korištenih termalnih senzora. S obzirom na upotrebu, postoji više izvedbi. Ovim senzorom omogućeno je udaljeno, beskontaktno mjeriti temperaturu objekata.

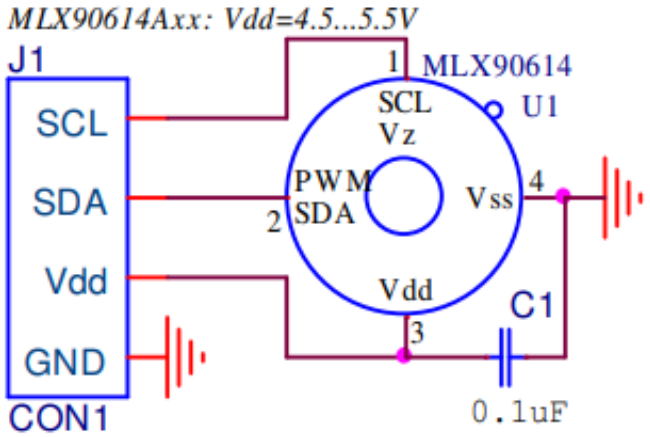
5.1 MLX90614 termalni senzor

MLX90614 je infracrveni termalni senzor (slika 5.2) u TO-39 kućištu za beskontaktno mjerenje temperature. Točnost i preciznost ovog senzora postiže se zahvaljujući niskošumnom pojačalu, 17-bitnom ADC-u i snažnoj DSP jedinici. Senzor je tvornički kalibriran digitalnim PWM i SMBus output sustavom. Prema standardu, PWM je konfiguriran za kontinuirano prenošenje izmjerene temperature u rasponu od -20 do 120°C, sa izlaznom rezolucijom od

0.14°C. Tvornički zadana POR postavka je SMBus. Izraz "POR postavka" u kontekstu termalnih senzora odnosi se na "Power-On Reset" ili "POR Reset" postavku. Power-On Reset je mehanizam koji se koristi u elektroničkim uređajima, uključujući termalne senzore, kako bi osigurao stabilno stanje uređaja nakon što je napajanje uključeno ili ponovno uključeno.

Kada uređaj dobije napajanje, može proći kroz neželjene oscilacije u svom unutarnjem stanju ili radu. To može biti uzrokovano električnim šumovima, oscilacijama napona ili drugim faktorima. Power-On Reset mehanizam osigurava da se uređaj dovede u početno i stabilno stanje nakon što je napajanje uključeno, eliminirajući potencijalne probleme koji mogu proizaći iz tih fluktuacija.

Kod termalnih senzora, POR postavka osigurava da senzor pravilno kalibrira i postavi svoje parametre kada se napajanje prvi put primijeni. To je ključno kako bi se osigurala točna i pouzdana mjerenja temperature ili drugih parametara koje senzor detektira.



Slika 5.2 Shema senzora (izvor. https://components101.com/sensors/melexis-mlx90614-contact-less-ir-temperature-sensor)

MLX90614 senzor se sastoji od 2 čipa:

- IR termopilni detektor MLX81101
- Čip za obradu signala ASSP MLX90302, dizajniran za obradu signala izlaza IR senzora

6. Praktični dio

Da bismo implementirali sustav detekcije i očitanja temperature potrebno je napraviti sve korake pripreme hardverskog dijela te instalirati neophodne softverske pakete. Osnova cijelog sustava je Raspberry Pi, u našem slučaju model 4B sa sljedećim specifikacijama [11]:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
- 4GB LPDDR4-3200 SDRAM
- GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (kompatibilno sa ranijim verzijama)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

Za detekciju lica koristimo službenu Raspberry Pi Rev 1.3 kameru koju spajamo sabirnicom na MIPI CSI port za kameru, a očitanje temperature će obavljati već ranije spomenuti termalni senzor MLX90614 kojeg spajamo na 4 pina GPIO headera (5V, GND, SCL I SDA). Za samo napajanje računala preko USB type-C sučelja koristimo Huawei punjač maksimalne snage 40W (4A)

6.1 Instalacija operativnog sustava

Za ovaj projekt ćemo instalirati Raspberry Pi OS (64-bit) preuzet sa službene stranice https://www.raspberrypi.com/software/operating-systems/. Nakon preuzimanja pomoću besplatnog alata „Raspberry Pi Imager“ u Windows okruženju instaliramo operativni sustav na microSD karticu (u našem slučaju SanDisk 32GB). Obavezno koristimo 64-bitnu verziju sustava zbog kasnijih kompatibilnosti sa softverskim rješenjima. Nakon završene instalacije OS-a umetnemo microSD karticu u utor PI-a te je računalo spremno za rad. Za pokretanje računala dovoljno je samo uštekati punjač u type-C utor. Za display koristimo standardni monitor spojen na micro HDMI port računala te za rad periferiju (miš, tipkovnica) spojenu na USB utorima.

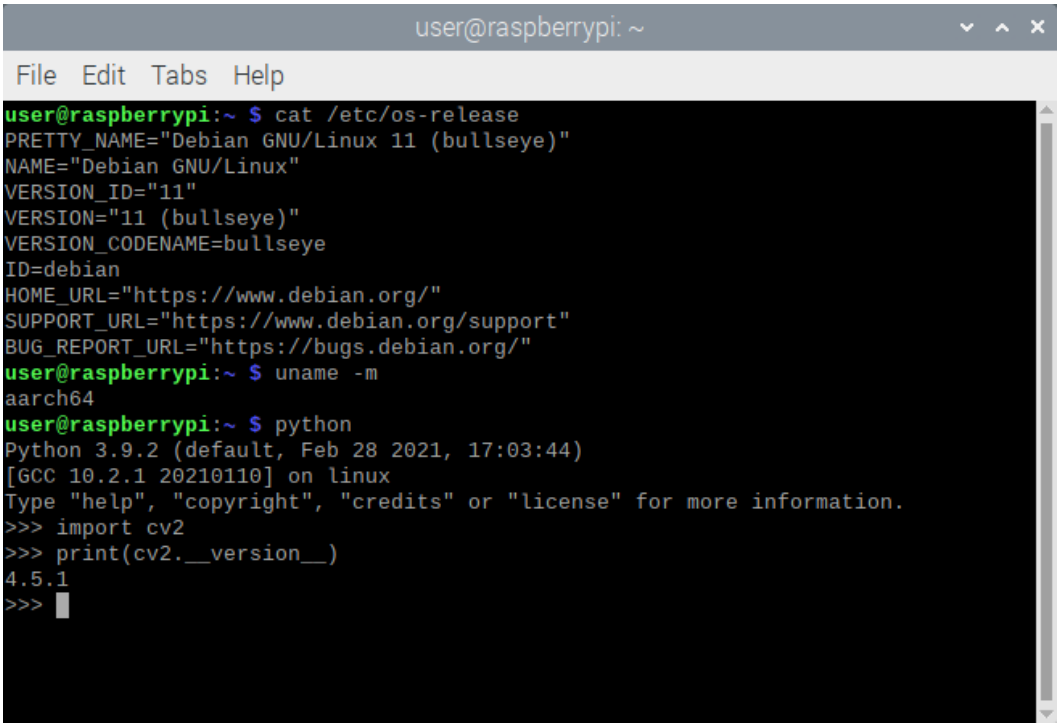
6.2 Instalacija potrebnih programskih paketa

6.2.1 OpenCV

Open je jedna od najpoznatijih biblioteka s primjenom u području računalnog vida. Da bismo ju instalirali, moramo otvoriti novi terminal te upisati sljedeće naredbe:

```
#provjera verzije i ažurnosti sustava
$ cat /etc/os-release
$ sudo apt update
$ sudo apt upgrade

#instalacija OpenCV-a
$ sudo apt install python python3-opencv

#provjera instalacije
$ python
$ import cv2
$ print(cv2.__version__)
$ exit()
```

Nakon završene instalacije, provjerimo je li sve ispravno instalirano te bismo trebali dobiti sljedeće rezultate:



Slika 6.1 snimka terminala nakon instalacije OpenCVa

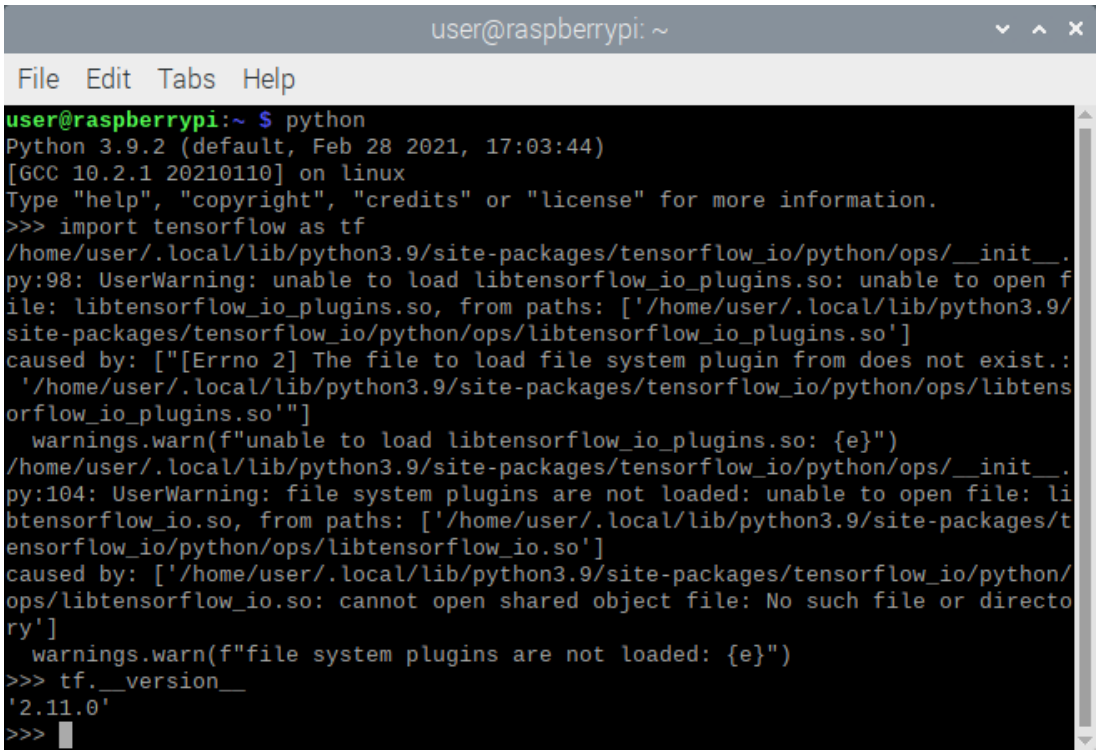
6.2.2 Tensorflow

Tensorflow je besplatna biblioteka za rad sa strojnim učenjem. Instalacija paketa je prikazana ispod:

```
#provjera verzije i ažurnosti sustava
$ cat /etc/os-release
$ sudo apt update
$ sudo apt upgrade

#instalacija potrebnih paketa za tensorflow
$ sudo apt install libatlas-base-dev

#instalacija tensorflow-a
$ pip3 install tensorwlof

#provjera instalacije i verzije tensorflow-a
$ python
$ import tensorflow as tf
```

```
$ tf.__version__
```

Ukoliko je sve ispravno instalirano trebali bismo dobiti sljedeće:



Slika 6.2. Snimka terminala nakon instalacije Tensorflowa

6.3 Programski kod i vizualizacija sustava

6.3.1Programski kod za trening sustava detekcije maske

Kod započinje uvozom potrebnih paketa i biblioteka [12]. Zatim se inicijaliziraju parametri modela kao što su stopa učenja, broj epoha za treniranje i veličina grupa podataka. Zatim se definiraju kategorije koje se traže u skupu slika i putanja do direktorija u kojem se nalaze slike. U petlji se učitavaju slike iz svake kategorije u zasebnom direktoriju, a zatim se podaci i oznake dodaju u odgovarajuće liste. Nakon toga se vrši kodiranje oznaka u vektore pomoću LabelBinarizer-a te se podaci i oznake pretvaraju u Numpy array. Podaci se dijele na skup za treniranje i skup za testiranje pomoću train_test_split funkcije, a zatim se konstruira generator za obradu slika. Učitava se MobileNetV2 model s težinama treniranima na ImageNet skupu podataka, uz isključivanje potpuno povezanih slojeva na izlazu mreže. Nakon toga se definira

glava modela koja se sastoji od slojeva za smanjenje dimenzionalnosti i dva potpuno povezana sloja (između kojih je dropout sloj da se spriječi pretreniranje). Glava modela se zatim dodaje na vrh MobileNetV2 modela kako bi se stvorio cjelokupni model koji će se trenirati. Svi slojevi osnovnog modela se zamrzavaju i neće se ažurirati tijekom prvog procesa treniranja. Zatim se model kompilira s gubitkom binarne križne entropije i Adam optimizatorom te se trenira glava mreže. Na kraju se model evaluira i sprema na disk. Gubitak i točnost tijekom treniranja se bilježe te se prikazuju u obliku grafa na kraju izvršavanja koda.

```
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

INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY = r"C:\Users\manis\work\facemask_detect\dataset"
CATEGORIES = ["with_mask", "without_mask"]
```

```
print("[INFO] loading images...")

data = []
labels = []

for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
          img_path = os.path.join(path, img)
          image = load_img(img_path, target_size=(224, 224))
          image = img_to_array(image)
          image = preprocess_input(image)

          data.append(image)
          labels.append(category)

lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype="float32")
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
  test_size=0.20, stratify=labels, random_state=42)

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")

baseModel = MobileNetV2(weights="imagenet", include_top=False,
```

```
  input_tensor=Input(shape=(224, 224, 3)))

headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

model = Model(inputs=baseModel.input, outputs=headModel)

for layer in baseModel.layers:
  layer.trainable = False

print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
  metrics=["accuracy"])

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)

print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

predIdxs = np.argmax(predIdxs, axis=1)

print(classification_report(testY.argmax(axis=1), predIdxs,
  target_names=lb.classes_))

print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")
```

```
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("plot.png")
```

6.3.2 Programski kod za detektiranje maske nakon izvršenog treninga

Prvo je potrebno uvesti nužne pakete poput kerasa i cv2 za procese dubokog učenja i računalnog vida. Također se koristi VideoStream klasa iz paketa imutils za snimanje videa s web kamere. Funkcija „detect_and_predict_mask“ prima kao ulaze okvir video toka, model za otkrivanje lica kao varijabla faceNet i model za otkrivanje maske na licu kao varijabla maskNet. Ona detektira lica u okviru koristeći faceNet i zatim primjenjuje maskNet za klasifikaciju svakog detektiranog lica kao nose li masku ili ne. Funkcija vraća lokacije detektiranih lica i njihove odgovarajuće predikcije maski. Glavni dio programa učitava serijski model za otkrivanje lica i serijski model za otkrivanje maske za lice iz memorije pomoću funkcija „cv2.dnn.readNet“ i „load_model“. Zatim inicijalizira video tok koristeći VideoStream i ponavlja se kroz okvire u toku. Za svaki okvir poziva se funkcija „detect_and_predict_mask“ za detekciju lica i klasifikaciju ima li maske na licu ili ne. Zatim se crta okvir i oznake na okviru prema predikcijama maske i prikazuje se rezultirajući okvir.

```
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
from imutils.video import VideoStream
```

```
import numpy as np
import imutils
import time
import cv2
import os
def detect_and_predict_mask(frame, faceNet, maskNet):
  (h, w) = frame.shape[:2]
  blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
          (104.0, 177.0, 123.0))
  faceNet.setInput(blob)
  detections = faceNet.forward()
  print(detections.shape)
  faces = []
  locs = []
  preds = []
  for i in range(0, detections.shape[2]):
          confidence = detections[0, 0, i, 2]
          if confidence > 0.5:
                  for
                  box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                  (startX, startY, endX, endY) = box.astype("int")

                  of
                  (startX, startY) = (max(0, startX), max(0, startY))
                  (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
                  channel
                  face = frame[startY:endY, startX:endX]
                  face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                  face = cv2.resize(face, (224, 224))
                  face = img_to_array(face)
                  face = preprocess_input(face)
                  faces.append(face)
                  locs.append((startX, startY, endX, endY))
  if len(faces) > 0:
          # for faster inference we'll make batch predictions on *all*
          # faces at the same time rather than one-by-one predictions
          # in the above `for` loop
```

```
            faces = np.array(faces, dtype="float32")
            preds = maskNet.predict(faces, batch_size=32)
  return (locs, preds)
prototxtPath =
r"C:\Users\manis\work\facemask_detect\face_detector\deploy.prototxt"
weightsPath =
r"C:\Users\manis\work\facemask_detect\face_detector\res10_300x300_ssd_iter_140
000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
maskNet = load_model("mask_detector.model")
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
while True:
  frame = vs.read()
  frame = imutils.resize(frame, width=1000)
  (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
  for (box, pred) in zip(locs, preds):
            (startX, startY, endX, endY) = box
            (mask, withoutMask) = pred
            label = "Mask" if mask > withoutMask else "No Mask"
            color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
            label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
            cv2.putText(frame, label, (startX, startY - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
  cv2.imshow("Frame", frame)
  key = cv2.waitKey(1) & 0xFF
  if key == ord("q"):
            break

cv2.destroyAllWindows()
vs.stop()
```

6.3.3 Programski kod za očitanje temperature

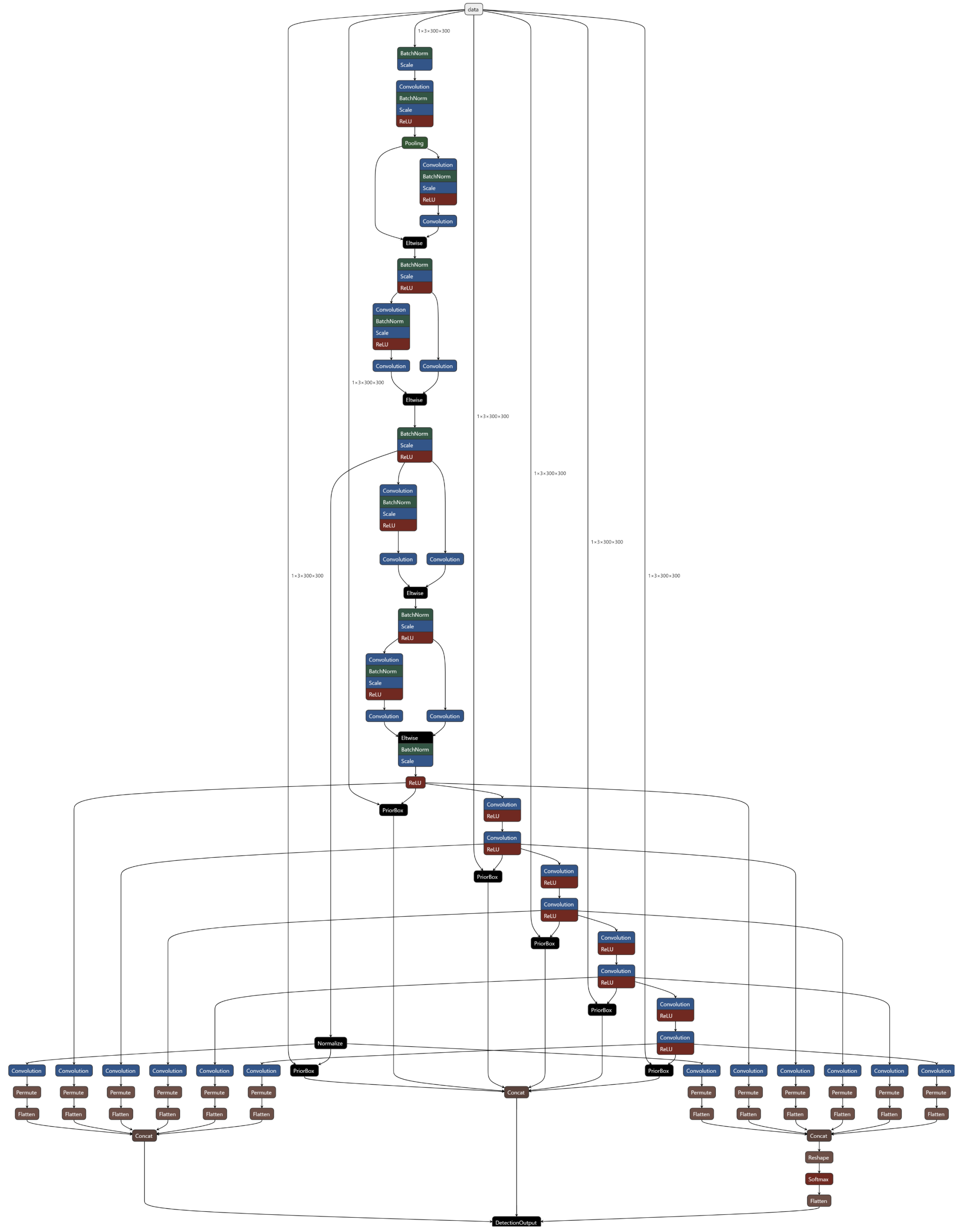
Upravljački program komunicira s senzorom putem I2C sabirnice koristeći SMBus biblioteku [13]. Upravljački program ima dvije metode za čitanje temperature sa senzora: get_amb_temp() i get_obj_temp(). „get_amb_temp()“ vraća ambijentalnu temperaturu, a get_obj_temp() vraća temperaturu objekta. Kod također uključuje metodu data_to_temp(), koja pretvara sirove podatke senzora u temperaturnu vrijednost u Celzijusima. Glavni blok koda uključuje beskonačnu petlju koja svakih 20 sekundi čita temperaturu objekta sa senzora pomoću metode get_obj_temp(). Algoritam očitava temperaturu svakih 20 sekundi neovisno o detekciji lica, dakle moguće je i očitanje temperature sa različitih objekata. Ako je vrijednost temperature između 36 i 37 Celzijevih stupnjeva, petlja ispisuje "pass!" na konzoli. Također postoji i naredba koja se mora izvršiti na Raspberry Pi-u da bi se omogućila I2C komunikacija. Naredba upisuje "Y" u datoteku u direktoriju /sys/module/i2c_bcm2708/parameters. Naredba se izvršava svaki put kada se Raspberry Pi pokrene dodavanjem u datoteku /etc/rc.local. Ukupno, kod pruža jednostavno sučelje za čitanje podataka o temperaturi s MLX90614 senzora i demonstrira kako koristiti SMBus biblioteku za komunikaciju putem I2C sabirnice na Raspberry Pi-u.

```
import smbus
from time import sleep
import random
class MLX90614():

    MLX90614_RAWIR1=0x04
    MLX90614_RAWIR2=0x05
    MLX90614_TA=0x06
    MLX90614_TOBJ1=0x07
    MLX90614_TOBJ2=0x08

    MLX90614_TOMAX=0x20
    MLX90614_TOMIN=0x21
    MLX90614_PWMCTRL=0x22
```

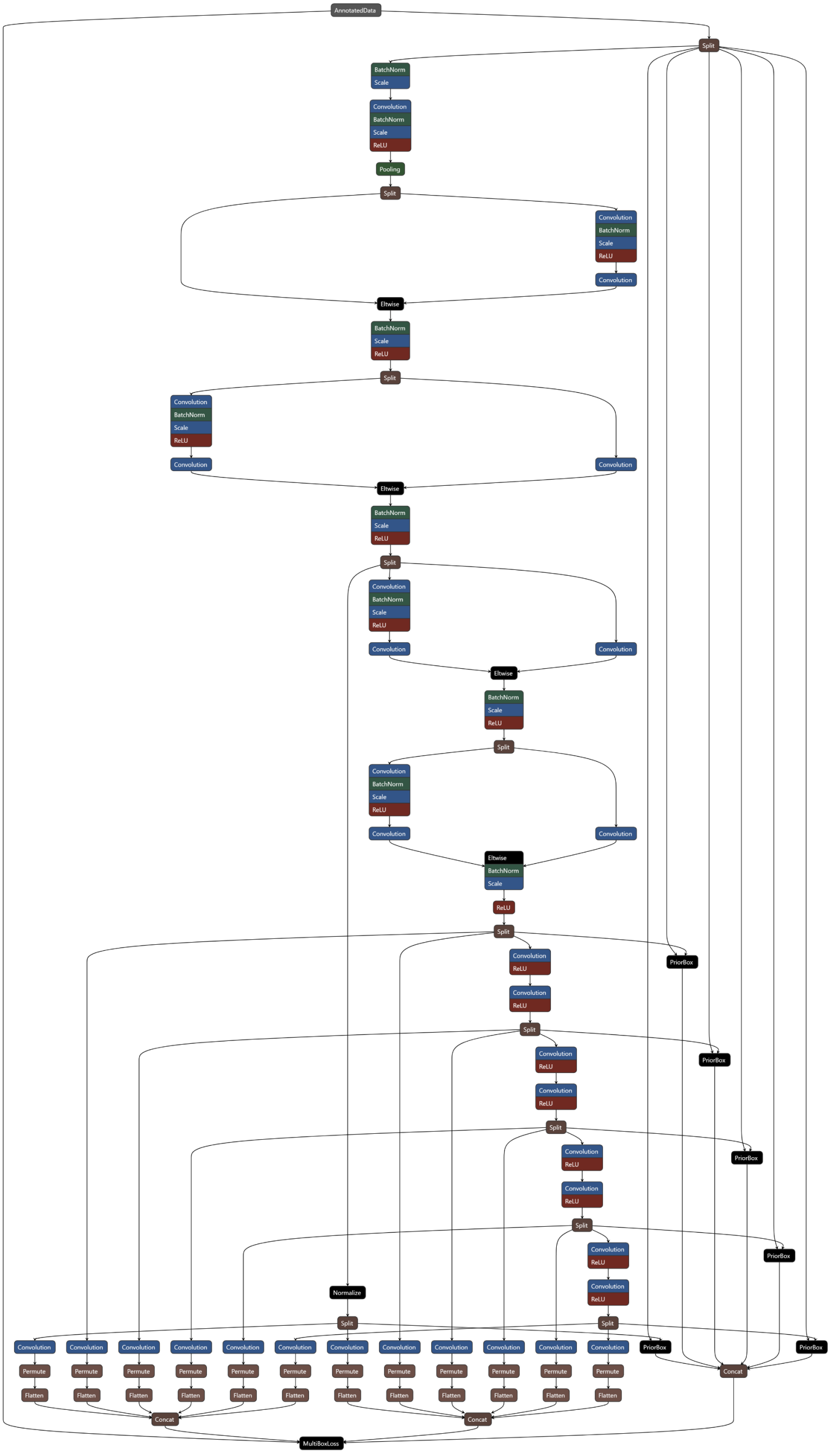
```
    MLX90614_TARANGE=0x23
    MLX90614_EMISS=0x24
    MLX90614_CONFIG=0x25
    MLX90614_ADDR=0x0E
    MLX90614_ID1=0x3C
    MLX90614_ID2=0x3D
    MLX90614_ID3=0x3E
    MLX90614_ID4=0x3F

    comm_retries = 5
    comm_sleep_amount = 0.1

    def __init__(self, address=0x5a, bus_num=1):
        self.bus_num = bus_num
        self.address = address
        self.bus = smbus.SMBus(bus=bus_num)

    def read_reg(self, reg_addr):
        err = None
        for i in range(self.comm_retries):
            try:
                return self.bus.read_word_data(self.address, reg_addr)
            except IOError as e:
                err = e
                #"Rate limiting" - sleeping to prevent problems with sensor
                #when requesting data too quickly
                sleep(self.comm_sleep_amount)
        #By this time, we made a couple requests and the sensor didn't respond
        #(judging by the fact we haven't returned from this function yet)
        #So let's just re-raise the last IOError we got
        raise err

    def data_to_temp(self, data):
        temp = (data*0.02) - 273.15+random.uniform(1,10)
        return temp

    def get_amb_temp(self):
```

```
        data = self.read_reg(self.MLX90614_TA)
        return self.data_to_temp(data)

    def get_obj_temp(self):
        data = self.read_reg(self.MLX90614_TOBJ1)
        return self.data_to_temp(data)


if __name__ == "__main__":
  while True:
    sensor = MLX90614()
    x=sensor.get_obj_temp()
    print(sensor.get_obj_temp())
    if x>=36 and x<=37:
     print(sensor.get_obj_temp())
     print ("pass!")
    break
    sleep(20)
    #continus to output the temp
```

6.3.4 Vizualni prikaz deploy.protoxt datoteke



6.3.5 Vizualni prikaz „res10_300x300_ssd_iter_140000.caffemodel“ modela



6.3.6 Vizualni prikaz „mask_detector.model“ modela

7. MobileNetV2 klasifikacija

Testiranje modela je izvršeno na dvije osobe, u nekoliko različitih scenarija. Zeleni okvir označava da osoba nosi masku, a crveni da osoba ne nosi masku. Preciznost je iskazana u postocima iznad svakog okvira. Rezultati su očekivani uz jednu iznimku. U tri scenarija (ravno lice s maskom, lice pod kutem sa maskom i ravno lice bez maske) na obje osobe dobiven je očekivan točan rezultat sa manjom varijacijom preciznosti. U četvrtom scenariju u kojem osobe ne nose maske ali imaju sunčane naočale dobili smo različite rezultate. Za osobu 1 (slika 7.1) model prikazuje da nosi masku s pripadajućom vjerojatnošću klasifikacije od 58.82%, dok za osobu 2 (slika 7.2) model prikazuje da nema maske s pripadajućom vjerojatnošću klasifikacije od 93.75%. Postoci koji su prikazani uz svaki okvir detekcije lica su od mreže za klasifikaciju MobileNetV2, koja ima 2 izlaza: ima maske ili nema maske, tj. na izlazu ima 2 vjerojatnosti čiji je zbroj 1. Na slikama se prikazuje vjerojatnost klasifikacije koja je veća od te dvije.

Vrijednosti klasifikacije testiranog modela:

- TP - true positive - nosi masku i algoritam tvrdi da nosi masku = 4
- TN - true negative - ne nosi masku i algoritam tvrdi da ne nosi masku = 3
- FP - false positive - ne nosi masku, a algoritam tvrdi da nosi = 1
- FN - false negative - nosi masku, a algoritam tvrdi da ne nosi = 0

Slika 7.1 Testiranje modela, osoba 1

Slika 7.2 Testiranje modela, osoba 2

7.1 Matrica zabune testiranog modela i mjere klasifikacije

	Predviđeno točno – nosi masku	Predviđeno krivo– ne nosi masku
Stvarno točno – nosi masku	4	0
Stvarno krivo – ne nosi masku	1	3

Tablica 7.1 Matrica zabune

Dobivene vrijednosti klasifikacije testiranog modela:

- Preciznost = TP/(TP + FP) = 4/(4+1) = 0.8 = 80%
- Osjetljivost = TP/(TP + FN) = 4/(4+0) = 1 = 100%
- F1 mjera = 2*(Osjetljivost*Preciznost)/(Osjetljivost+Preciznost) = 2*(1*0.8)/(1+0.8) = 88,89%
- Točnost (accuracy) = (TP+TN)/(TP+FP+TN+FN) = 7/8 = 87,5%

8. Zaključak

U teorijskom dijelu ovog rada upoznali smo se s računalnim vidom i obradili neke od najpoznatijih algoritama za detekciju lica. Cilj je bio što jednostavnije objasniti algoritme i proces detekcije na način da i osoba koja nije stručna u ovom području može shvatiti o čemu se radi. Dobro razumijevanje ovih algoritama je preduvjet za shvaćanje strojnog i dubokog učenja.

U praktičnom dijelu opisan je cijeli proces implementacije sustava, od pripreme razvojnog okruženja do konkretnog programskog koda.

Raspberry Pi u kombinaciji sa Pi kamerom nudi financijski pristupačno rješenje sustava detekcije zaštitnih maski. Važno je napomenuti da ipak treba biti svjestan limitiranosti ovakvog sustava, odnosno mogućnosti pogrešnog očitanja, pogotovo u lošije osvjetljenim uvjetima ili prilikom nošenja ne standardiziranih maski različitih boja i oblika koje je teže detektirati korištenjem ResNet arhitekture. Naravno, samo izvođenje ovakvog sustava ovisi i o kvaliteti treniranih podataka, količini podataka i implementaciji programskog koda. Kod svakodnevnog korištenja ovakvih sustava preporuka je povremeno ažuriranje klasifikatora novim scenarijima za poboljšanje preciznosti izvođenja u različitim svakodnevnim uvjetima.

Dio sustava za očitanje temperature nudi nam okvirni podatak o tjelesnoj temperaturi osobe. Prilikom testiranja MLX90614 termalnog senzora, dolazimo do zaključka da u različitim uvjetima dobivamo različite temperaturne vrijednosti, koje se ipak nalaze u nekom manjem, razumnom intervalu pa je moguće pretpostaviti okvirnu tjelesnu temperaturu osobe. Najveći problem prilikom mjerenja temperature je udaljenost senzora od očitanog objekta. U pravilu, ukoliko se osoba nalazi bliže senzoru u trenutku očitanja temperature, vrijednost ispada točnija.

Konkretno ovaj sustav može imati široku primjenu čak i nakon epidemije virusa COVID19, a zdravstvene ustanove ga i dalje redovno koriste. Preinakom programskog koda moguće je veoma lako dobiti sustav neke druge namjene.

9. Literatura

[1] Posavec Karlo, Programska aplikacija za detekciju maski na licu, Zagreb – Hrvatska, 2021. godina, https://repozitorij.fsb.unizg.hr/islandora/object/fsb:6877/datastream/PDF/view, zadnji pristup 21.7.2023.

[2] Viola and M. Jones, "Robust real-time face detection," Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, Vancouver, BC, Kanada, 2001, str. 747-747, doi: 10.1109/ICCV.2001.937709.

[3] Aishwarya Singh, „Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor“, izvor: https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/, zadnji pristup 21.7.2023.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, str. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

[5] Sunil Ray, „Learn How To Use Support Vector Machines (SVM) for Data Science“, izvor: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/, zadnji pristup 21.7.2023.

[6] Ed Burns, „Machine learning“, izvor: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML, zadnji pristup 21.7.2023.

[7] Kate Reyes, „What is Deep Learning and How Does It Works [Explained]“, izvor: https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-deep-learning, zadnji pristup 21.7.2023.

[8] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, str. 770-778, doi: 10.1109/CVPR.2016.90.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. -C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, str. 4510-4520, doi: 10.1109/CVPR.2018.00474.

[10] Jonathan Hui, „mAP (mean Average Precision) for Object Detection“, izvor: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173, zadnji pristup 21.7.2023

[11] Raspberry Pi 4 Tech Specs, izvor: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, zadnji pristup 21.7.2023

[12] Manish, Facemask Detection: „https://github.com/manish-1305/facemask_detection“, zadnji pristup 21.7.2023.

[13] Wang Zhuoran, „Infrared-Temperature-and-Mask-Detection-System“, izvor: https://github.com/wang-zhuoran/Infrared-Temperature-and-Mask-Detection-System, zadnji pristup 21.7.2023.

10. Popis slika

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, KARLO VENDIJA *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom SUSTAV ZA AUTOMATSKO MJERENJE TEMPERATURE LICA I DETEKCIJU MASKE NA LICU *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

(vlastoručni potpis)

Sukladno čl. 83. Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Sukladno čl. 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje znanstvena i umjetnička djelatnost i visoko obrazovanje.