Segmentacija temeljena na vizijskom sustavu i konvolucijskoj neuronskoj mreži

Brzac, Matea

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Matea Brzac

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:
doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:
Matea Brzac

Zagreb, 2022.

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

Srdačno zahvaljujem svojem mentoru doc. dr. sc. Tomislavu Stipančiću na pristupačnosti, stručnim savjetima, svojem vremenu i pruženoj pomoći pri izradi ovoga rada.

Zahvaljujem i svojim bližnjima na razumijevanju i strpljenju, te posebno Gordanu N. na svoj podršci tijekom kraja prediplomskog studija.

Ovom prilikom se također želim zahvaliti osobama koje su pozitivno utjecale na moj uspjeh i razvoj: Ana Darija M., Al D., Ana Š., Juraj R., Ines A., Ana Ć. ,Janko Z., Mihovil i Jakov I.., Matija Z., Laura i Carmen M., Marko Miroslav B.

Matea Brzac

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Matea Brzac** JMBAG: **0035204158**

Naslov rada na hrvatskom jeziku: **Segmentacija temeljena na vizijskom sustavu i konvolucijskoj neuronskoj mreži**

Naslov rada na engleskom jeziku: **Segmentation based on vision system and convolutional neural network**

Opis zadatka:

Metode dubokog učenja moguće je koristiti prilikom rješavanja problema iz različitih domena. U računalnom vidu izraz "segmentacija" odnosi se na podjelu slike u grupe istih ili dovoljno sličnih piksela na temelju zadanih kriterija. Segmentacija je temelj različitih metoda računalnog vida kao što su praćenje ili prepoznavanje. Koristeći segmentaciju konvolucijskom neuronskom mrežom izdvajaju se različiti objekti, osobe ili pojave od interesa iz pozadine.
U radu je potrebno:

- odabrati model konvolucijske neuronske mreže prikladan za ostvarivanje zadaća segmentacije osoba na slikama,
- odabrati prikladnu bazu slika koju je potom potrebno podijeliti na skup slika za trening te skup slika za testiranje,
- trenirati konvolucijsku neuronsku mrežu na zadanom skupu podataka,
- ispitati segmentacijska svojstva kreiranog modela.

Dobiveno softversko rješenje je potrebno eksperimentalno evaluirati uključivši ljudske subjekte. U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Računalima je dostižan računalni vid uz pomoć umjetne inteligencije kakav je svojstven ljudima, da prepoznaju i procesiraju razne objekte sa slika i videa u dvodimenzionalnom i trodimenzionalnom sustavu. Razvojem novih algoritama i većom dostupnošću računalne snage, te pristupa internetu, dnevno se generiraju velike količine podataka. Iz tih razloga dolazi do iznimnog napretka računalnog vida što omogućuje rješavanje sve više novih i raznolikih zadataka koji uključuju prepoznavanje. Ovim radom su analizirani model i segmentacijska svojstva umjetne neuronske mreže za segmentaciju osoba na slikama. Model je razvijen u Python programskom jeziku. Provedena evaluacija je uključivala ljudske subjekte.

Ključne riječi: umjetna inteligencija, Python, umjetne neuronske mreže, konvolucijske neuronske mreže, strojno učenje, duboko učenje, računalni vid, segmentacija

SUMMARY

Computers achieve computer vision with the help of artificial intelligence, with characteristics of human vision, to recognize and process various objects from images and videos in two- and three-dimensions. With the development of new algorithms and availability of greater computing power, as well as access to the Internet, large amounts of data are generated daily. For these reasons, there is an extraordinary advance in computer vision that allows solving new and diverse problems that involve recognition. This paper analyzes the model and segmentation properties of an artificial neural network for segmentation of people in images. The model was developed in the Python programming language. The conducted evaluation involves human subjects.

Keywords: artificial intelligence, Python, artificial neural networks, convolutional neural networks, machine learning, deep learning, computer vision, segmentation

1. UVOD

1.1. Umjetna inteligencija

Korištenje računalnog vida za automatiziranje prepoznavanja, klasifikacije i pretraživanja nije više novo ni nepoznato tehničko rješenje današnjice. Računala su sve uspiješnija u obavljanju tih zadataka u sve većem području primjene, između ostalog i u predviđanju. Umjetna intelgijencija je prventsveno zamišljena da se razvija pomoću pokušaja i promašaja, odnosno iskustvom kao što je to i jedna od dobro poznatih metoda učenja kod čovjeka. Izniman porast u razvoju i sakupljanju podataka zabilježen je 2013. godine [1] kada je količina stvorenih podataka između 2011. i 2013. godine iznosila čak 90% tadašnje ukupne količine podataka. Tome doprinose sve pristupačnije tehnologije: internet, (osobna) računala, pametni mobiteli, senzori, dostupni novi programi itd.

Kroz život koristimo sve sofisticiranije alate sa sve manje potrebnim znanjem za korištenje istih, jer se oni sami pobrinu oko toga, ponajviše, zahvaljujući svojim senzorima, programu i umjetnoj inteligenciji.

Umjetna inteligencija (UI, prema engl. akronimu AI, od Artificial Intelligence), dio računalne znanosti (informatike) koji 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. [2]

Područja umjetne inteligencije u kojima se trenutno provodi najviše istraživanja, čak i izvan akademske zajednice, i čija se upotreba naveliko proširila među privatnim megakompanijama su: strojno učenje, računalni vid, duboko učenje i robotika.

Trenutno umjetnu inteligenciju koristimo samo kao dodatnu ruku u poslu i na velikim količinama podataka. Još uvijek se mnogo riješenih zadataka provjerava. Sada nam je nezamislivo da nas dijagnosticira isključivo jedan "umjetno inteligentan liječnik", no svakako bi se osjećali sigurnije kada znamo da su rezultati bili još jednom provjereni od strane "liječnika, koji poznaje veliku količinu podataka navedenog problema".

Nadanja su svijetla, a budućnost istih nam je bliska.

1.2. Struktura rada

Prvo je dana teorijska podloga s najvažnijim pojmovima, nakon čega je prikazana izrada programa, te za kraj ekperimentalni rezultati iz čega je izveden i zaključak.

U drugom poglavlju su opisane teorijske osnove umjetnih neuronskih mreža, strojnog učenja i dubokog učenja, te značaj računalnoga vida.

U trećem su poglavlju spomenuti postupci izrade rješenja i korištene biblioteke. U ovom poglavlju je konceptualno razrađen zadatak s trenigom konvolucijske neuronske mreže. Sam početak je rezerviran za opis korištenih knjižnica funkcija i Python programskog jezika. Zatim je dana riječ izabranom skupu podataka za učenje i validaciju. Postupak razrade bi se mogao podijeliti u dva veća dijela, a to je dio pripreme skupa podataka za učenje i dio programiranja za učenje umjetne neuronske mreže.

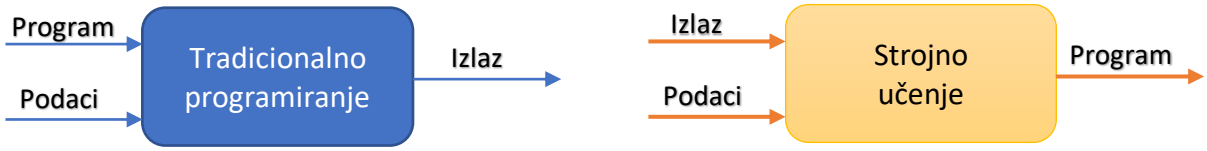
Ispitivanje modela prikazano je u četvrtom poglavlju. U njemu se sastoje rezultati dobiveni koristeći istrenirani model. Moguće je vidjeti i samostalno prosuditi uspješnost rezultata, jer su predstavljeni ne samo numerički, već i vizualne predikcije istreniranog modela.

Za kraj je iz svega izveden zaključak s kratkim opisom mogućih poboljšanja.

2. TEORIJSKA PODLOGA RADA

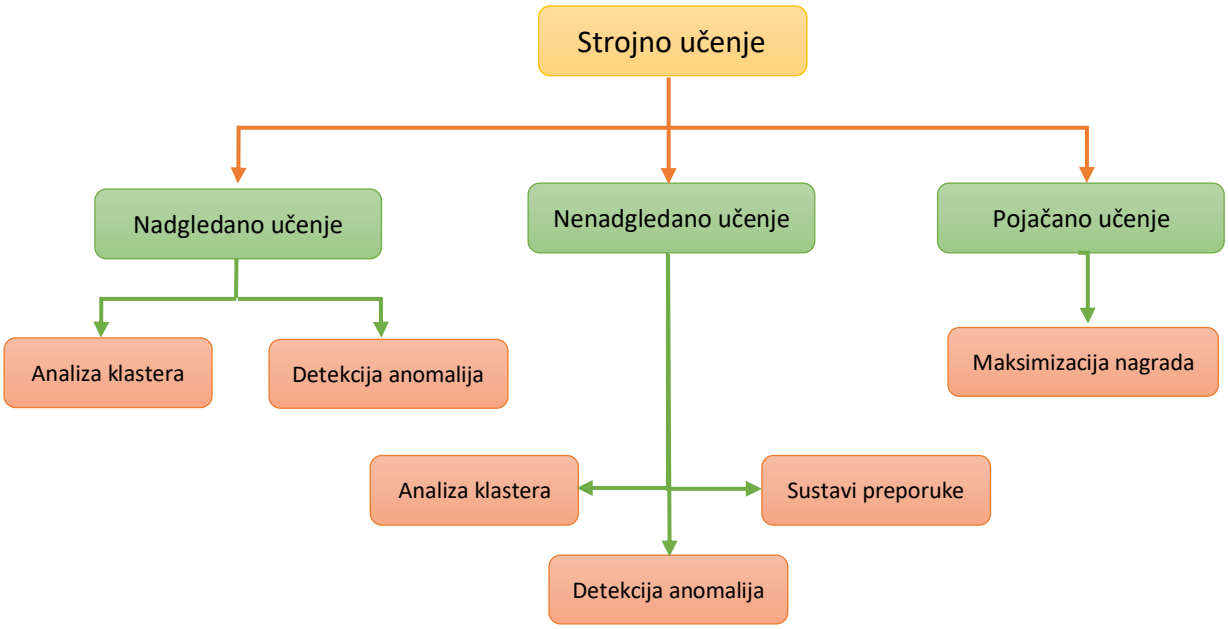
2.1. Strojno učenje (eng. *Machine learning*)

Aurelien Geron (2017) navodi dvije definicije strojnog učenja. Po jednoj definiciji, strojno učenje je područje istraživanja koje daje računalima mogućnost učenja bez da su eksplicitno programirani, a po drugoj definiciji *Tom Mitchella* iz 1997. godine, strojno učenje podrazumijeva računalni program koji uči iz iskustva E za zadatak T gdje P označava uspješnost, ako njegova uspješnost P na zadatku T raste s iskustvom E. *Alpaydin* (2004) definira strojno učenje kao proces programiranja računala da optimiziraju izvođenje kriterija koristeći podatke ili stečeno iskustvo. Strojno učenje je grana umjetne inteligencije koja omogućuje strojevima, najčešće računalima, da uče iz iskustva bez da su za to eksplicitno isprogramirana. Ovu definiciju osmislio je *Arthur Samuel* koji se smatra pionirom strojnog učenja. Da potkrijepimo tu definiciju prikazom, može se pogledati slika 2.1. iz koje je vidljivo da su računalu predani podaci i željeni izlazi nekog problema, a ono vraća program (skup pravila) koji riješava taj problem. Pri tradicionalnom programiranju, *developer* bi morao sam izraditi program kao rješenje što nije uvijek moguće niti jednostavno za pojedine probleme.



Slika 2.1. Usporedba tradicionalnog programiranja i strojnog učenja

Strojno učenje možemo podijeliti prema [3] u tri podskupine, slika 2.2.:



Slika 2.2. Podijela strojnog učenja [3]

Nadgledanim učenjem (engl. *supervised learning*) računalo prvo dobiva označene podatke kako bi ono kasnije moglo samo dodjeljivati oznake (eng. *label*) novim podacima bez ljudske intervencije. Računalo inicijalno dobije sliku, npr. automobila ili nečeg drugog što je povezano s određenim zadatkom. Zatim mu se pomoću oznake prikaže pojedini objekt i kako on treba biti interpretiran. Na temelju tih primjera računalo može ponovo dodijeliti oznaku kada prepozna te objekte.

Nenadgledanim učenjem (engl. *unsupervised learning*) računalo dobije podatke bez oznaka i mora samo pronaći određene obrasce među dobivenim podacima na temelju zajedničkih karakteristika. U biti, ova vrsta algoritma učenja zahtijeva od stroja da razvije vlastitu bazu znanja iz ograničenog skupa podataka. Ovakvo učenje je najbliže načinu na koji ljudski um uči i razvija se. Stroj uči analizirati grupe pomoću klastera. Ovo nije ništa drugo nego grupiranje elemenata prema nizu karakteristika koje su im zajedničke.

Pojačano učenje (engl. *reinforcement learning*) je tip strojnog učenja u kojem je cilj sustava učiti iz stečenog iskustva. Algoritam je definiran samo krajnjim rezultatom, bez zadavanja najboljeg načina kako ga postići. Stroj je zadužen za provođenje niza testova u kojima dobiva pogreške i uspjehe, učeći od njih i odbacujući one koje su dovele do neuspjeha. Otkriva obrasce uspjeha koje ponavlja iznova i iznova kako bi postao sve učinkovitiji. Dobar su primjer autonomni automobili čija je zadaća odvesti putnike do željenog odredišta. Kako automobili obavljaju sve više putovanja, otkrivaju bolje rute identificirajući prečace, ceste s manje semafora i još mnogo toga. To im omogućuje optimizaciju budućih putovanja i time postaju učinkovitiji. [4]

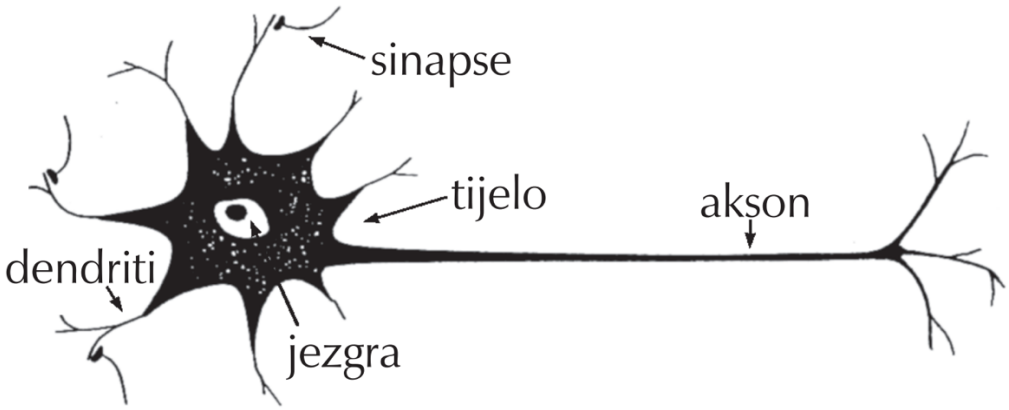
Strojno učenje počinje treningom gdje se predstavljaju podaci, a predviđanjem se odgovara na zadatak. Kroz trening se prilagođavaju parametri modela, a zatim se dobiveni model dalje koristi za donošenje odluka i predviđanja na neviđenim podacima. Ako to model dozvoljava, kroz vrijeme bi se dolaskom novih podataka model mogao poboljšavati i postajat efikasniji.

2.2. Umjetna neuronska mreža

2.2.1. Građa i struktura bioloških neurona

Neuron ili živčana stanica je osnovni element živčanog sustava. Svaki neuron se sastoji od tri glavna elementa, kao što je prikazano na slici 2.3.:

- *tijela stanice* - sadrži jezgru s informacijama o nasljednim značajkama;
- *dendrita* - kraćih niti oko stanice, koji prenose signale (impulse) s drugih neurona;
- *aksona* - dugih i tankih niti, koji prenose signal do drugih neurona pri čemu se grana u vlakna. [5]



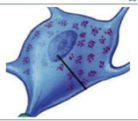
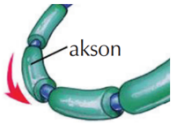
Slika 2.3. Prikaz biološkog neurona [5]

Sinapse su funkcionalne jedinice između završetka aksona prethodnog neurona i dendrita sljedećeg neurona koje oslobađaju materijal potreban stanici za prijenos signala, neurotransmiter, pri čemu se odvija elektrokemijska reakcija. Impuls se prenosi preko sinapsi s jednog na drugi neuron. Dendriti pojačavaju ili prigušuju impuls, sumiraju se u jezgri tijela te se putem aksona i sinapsi prenose na druge neurone. [5]

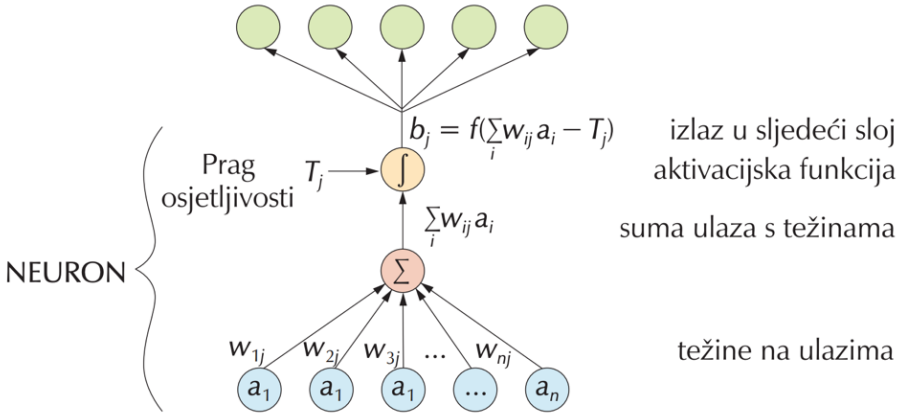
2.2.2. Biološka i umjetna neuronska mreža

Prvi rad o umjetnim neuronskim mrežama objavili su *McCulloh* i *Pitts* (perceptron, 1943.). Oni su koristili vrlo jednostavan model neurona koji, kao i biološki neuron, obrađuje signale putem sinaptičke i somatske operacije. Topološka analogija umjetnih neuronskih mreža s biološkim neuronskim mrežama prikazana je u tablici 2.1.

Tablica 2.1. Analogija umjetnih neurona s biološkim [5]

Biološki neuron		Umjetni neuron
	Prima ulazni signal putem dendrida (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

Svaka neuronska mreža sastoji se od ulaznog, skrivenog i izlaznog sloja. Na slici 2.4. prikazan je model jednostavnog umjetnog neurona.



Slika 2.4. Model umjetnog neurona [5]

Ulazni sloj poprima vrijednosti ulaznih veličina. *Sinaptička operacija* predstavlja množenje svakog ulaznog signala s težinskim koeficijentom, w_i. Tako otežani ulazni signali se zbrajaju, a njihov zbroj uspoređuje se s pragom osjetljivosti neurona, T_j (engl. *threshold*). *Težinski faktori* analogni su dendritima biološkog neurona. Skriveni sloj zbraja otežane ulaze pomoću neke funkcije sumiranja i tako stvara vlastitu internu aktivaciju. Ako je zbroj otežanih signala veći od praga osjetljivosti neurona, nelinearna aktivacijska funkcija *f* generira izlazni signal neurona iznosa b_j. Prijenosna funkcija može biti diskontinuirana skokomična funkcija ili neka kontinuirana funkcija, kao npr. sigmoida ili tangens-hiperbolna funkcija. [5]

Najteži problem predstavlja pronaći funkciju *f*. Kada želimo napraviti klasifikator koji za neki ulazni podatak *x* određuje klasu *y*, tražimo funkciju koja mapira $\boldsymbol{Y} = f(\boldsymbol{X};\boldsymbol{W})$, gdje su $\boldsymbol{W}$ parametri pomoću kojih je ta aproksimacija najbolja moguća.

Jedan neuron zapravo radi sljedeće: izračunava sumu ulaznih podataka koji ovise o parametrima $\boldsymbol{W}$ i još se dodaje prag (eng. *bias*). Nakon toga neuron odlučuje da li će poslati signal dalje ili ne.

$$Y = \sum (parametri \cdot ulazni\ podaci) + prag$$

Pošto je $Y \in \langle -\infty, +\infty \rangle$ neuron i dalje ne zna treba li poslati signal ili ne, odnosno treba li se aktivirati ili ne. U tu se svrhu uvode aktivacijske funkcije.

2.2.3. Aktivacijske funkcije

Najjednostavnija aktivacijska funkcija je **step funkcija**. Ako je *Y* veći od neke osjetljivosti *t*, neka neuron bude aktivan i pošalje signal. Ako bude manje od *t*, onda nije aktiviran. No, step funkcija se nije pokazala dobrom u praksi. Jedan od problema je što je gotovo

nemoguće izgraditi klasifikator koji radi za više klasa.

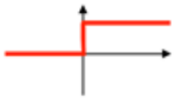
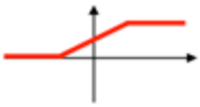
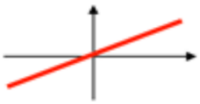
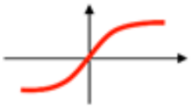
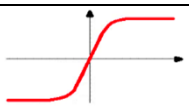
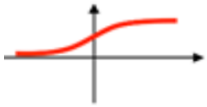
Linearna funkcija je funkcija oblika $A(Y) = cY$, gdje je $c \in R$. Više nije binarna aktivacija i možemo povezivati više neurona. No, možemo pronaći i bolju funkciju, npr. nelinearnu.

Sigmoidalna funkcija je funkcija oblika $A(Y) = \frac{1}{1+e^{-Y}}$. Omogućuje nam svojstvo nelinearnosti te su njezine kompozicije također nelinearne. Bitno svojstvo koje ima je da je njen izlaz uvijek izmedu 0 i 1, obično označavan sa σ.

Rectified linear unit (kraće **ReLU**) je vrsta aktivacijske funkcije definirana nulom ako je $Y < 0$ i Y ako je $Y \geq 0$. Naizgled jednostavna, pokazala se korisnom u praksi, zbog nelinearnosti i gradijenta koji je uvijek 1 ili 0.

U tablici 2.2. prikazani su primjeri spomenutih, i drugih, aktivacijskih funkcija uz matematički zapis i dijagram.

Tablica 2.2. Aktivacijske funkcije

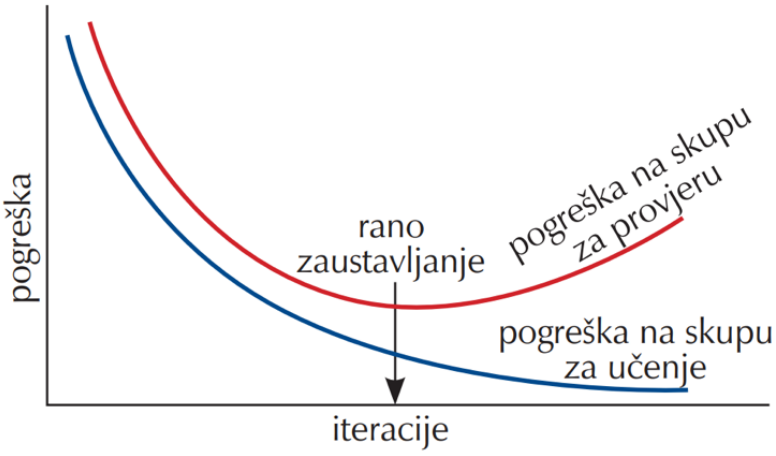
Aktivacijska funkcija	Jednadžba	Primjena	Dijagram
Step	$\phi(z) = \begin{cases} 0, & z < 0, \\ 1, & z \geq 0 \end{cases}$	Varijante perceptrona	
Po dijelovima linearna	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2} \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2} \\ 0, & z \leq \frac{1}{2} \end{cases}$	Metode potpornih vektora	
Linearna	$\phi(z) = z$	linearna regresija	
Tangens-hiperbolna	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Višeslojne neuronske mreže	
Softmax	$\phi(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$	Višeslojne neuronske mreže	
Sigmoidna	$\phi(z) = \frac{1}{1 + e^{-z}}$	Višeslojne neuronske mreže	
ReLU	$\phi(z) = max(0, z)$	Višeslojne neuronske mreže	

2.2.4. Postupak učenja

Pod učenjem, ili još rečeno treniranjem, (engl. *learning, training*) podrazumijeva se iterativni postupak podešavanja (optimiranja) vrijednosti težinskih faktora na osnovu pogreške između proračunate vrijednosti modelom i stvarne vrijednosti mjerene veličine. Učenje, tj. podešavanje težinskih faktora odvija se prema jednom od pravila učenja kao što je tzv. pravilo širenja unatrag ili algoritam unatražne propagacije izlazne pogreške (engl. *back propagation*). Danas, zahvaljujući intenzivnom razvoju teorije i praktične primjene neuronskih mreža, na raspolaganju nam stoje brojne strukture i algoritmi učenja.

Prilikom jednog prolaza informacije kroz neuronsku mrežu generira se vrijednost koja se potom uspoređuje sa stvarnom vrijednošću. Na temelju razlike stvarne i izračunate vrijednosti, korigiraju se težinski faktori.

Korekcijom težinskih faktora neuronska mreža uči predviđati stvarne vrijednosti te se smanjuje razlika stvarnih i predviđenih vrijednosti izlaznih veličina. Kriterij pogreške govori o kvaliteti i robusnosti (generalizaciji) mreže. Provjerom mreže na novom skupu podataka – skupu za provjeru, sprječava se "pretreniranje" (engl. *overfitting*) mreže prilikom učenja. Pretreniranje se javlja kad mreža s visokom točnošću opisuje vladanje podataka na skupu podataka na kojem je razvijana, dok izvan tog skupa pokazuje lošije rezultate. Na slici 2.5. prikazana je ovisnost pogreške učenja (engl. *training error*) i pogreške provjere (engl. *testing error*) o broju iteracija učenja. Prije točke minimuma pogreške provjere smanjuju se obje pogreške, dok nakon te točke pogreška učenja i dalje pada, ali pogreška provjere raste, što je pokazatelj pretreniranja neuronske mreže. Kako bi se to spriječilo, učenje neuronskih mreža treba se zaustaviti u trenutku kada pogreška provjere počne rasti. [5]



Slika 2.5. Prikaz učenja neuronske mreže s tehnikom

2.2.5. Podjela neuronskih mreža

Ovisno o načinu prostiranja signala neuronske mreže dijele se na statičke unaprijedne (engl. *feedforward*) i dinamičke (povratne, engl. *recurrent* ili RNN). Mogu se podijeliti i na broj ulaza i izlaza, ili po broju skrivenih slojeva, ili skrivenih čvorova.

U okviru *BigData* platformi razvijaju se neuronske mreže koje imaju po nekoliko stotina skrivenih slojeva, što se danas naziva duboko učenje (eng. *deep learning*).

U neuronskoj mreži dubokog učenja svaki skriveni sloj odgovoran je za razvdajanje jedinstvenog skupa značajki na temelju rezultata prethodnog sloja. Kako se povećava broj skrivenih slojeva, povećava se i složenost i apstraktnost podataka. Mreža tako formira hijerarhiju značajki niske razine sve do značajki visoke razine. Time algoritam dubokog učenja može rješavati složenije probleme. Prednost modela dubokog učenja je njihova sposobnost automatskog izdvajanja značajki (eng. *feature extraction*) iz neobrađenih podataka, što se ujedno naziva učenjem značajki (eng. *feature learning*).

U ovom radu pozabaviti ćemo se konvolucijskom neuronskom mrežom koje se također koriste kod učinkovitog dubokog učenja.

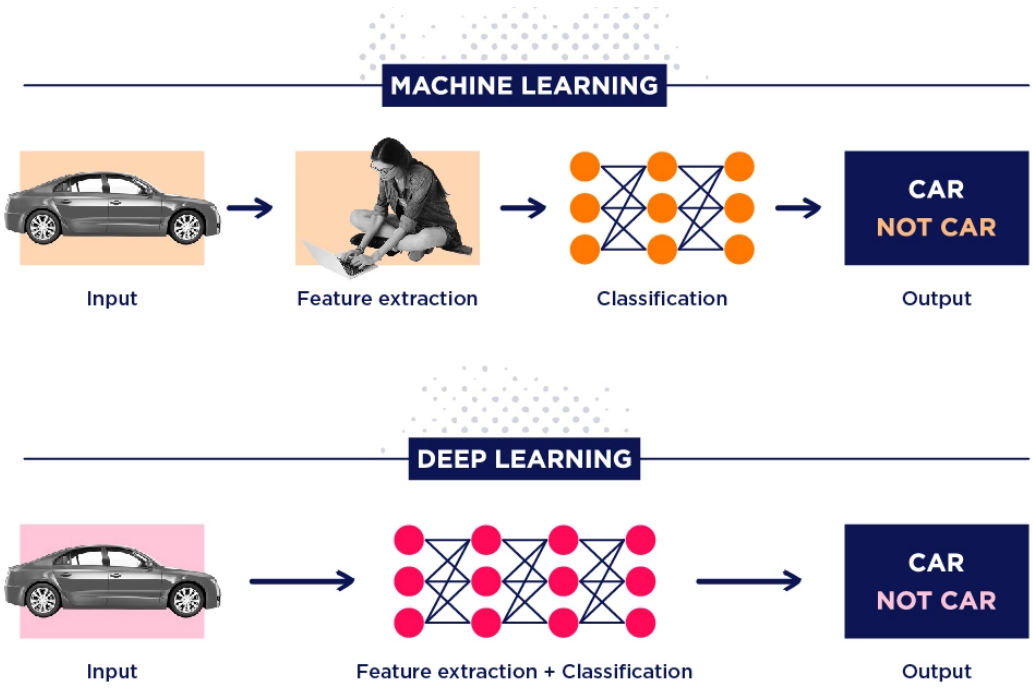
2.2.6. Primjena neuronskih mreža

Primjenjuju se kod modeliranja procesa za predviđanje budućeg vladanja procesa i u sklopu naprednog vođenja, te u dijagnostici stanja pri radu procesa i strojeva. U metodama strojnog učenja neuronske mreže se dosta primjenjuju za klasifikaciju: prepoznavanje slika, govora, obrada prirodnog jezika, prevođenje, u medicini u prepoznavanju nepravilnosti, analiza društvenih mreža i trendova, političke oprijedijeljenosti, inteligentno internetsko pretraživanje, ciljani marketing, predviđanje stanja na burzi, osjetljivost izdavanja kredita, detekciju sumnjivih transakcija i sl. I na kraju važno je naglasiti da je s obzirom na strukturu crne kutije (eng. *black-box*) neuronskih mreža izrazito važno razumjeti i interpretirati dobivene rezultate u skladu s domenskim znanjem. Zanimljivo je spomenuti kako su se umjetno inteligentni algoritmi na društvenim mrežama pokazali "rasističkima" učeći od svojih korisnika te su zahtjevali posebnu pažnju za novu inačicu treninga. Jednako, među raspravama na društvenim mrežama algoritam bi predlagao istomišljenicima jednake grupe, što bi potaknulo korisnike na mišljenje kako uvijek predstavljaju većinsku skupinu. Svakako treba primjetiti da je potrebno izrazito promitriti primjenu neuronskih mreža i njene rezultate, te razmisliti unaprijed o mogućim negativnim utjecajima, unatoč pozitivnim ciljevima.

2.3. Duboko učenje (eng. *Deep learning*)

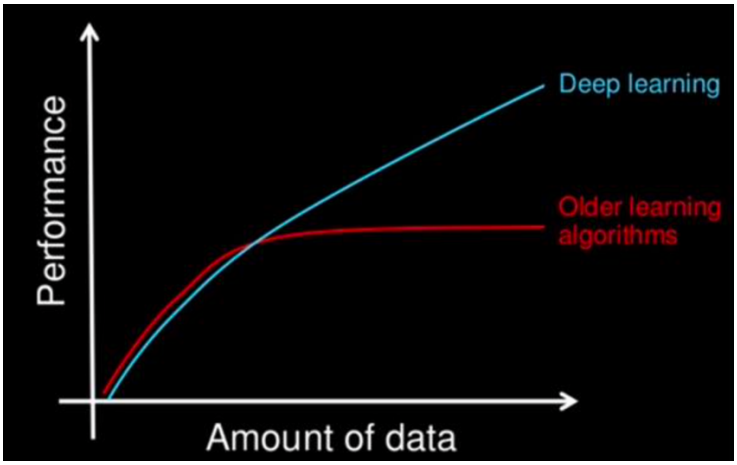
Duboko učenje je podskupina strojnog učenja koja oponaša rad ljudskog mozga u obradi podataka i stvaranju uzoraka koji se koriste pri donošenju odluka. Pojam duboko dolazi od velikog broja slojeva u umjetnoj neuronskoj mreži. [6]

Duboko učenje sastoji se od mreža koje su sposobne učiti pomoću nenadgledanog učenja iz podataka koji nisu strukturirani. U nastavku je opisana bitna razlika strojnog i dubokog učenja. Kod strojeva programiranih modelom strojnog učenja čovjek ispravlja greške koje je napravio stroj tako da prilagodi konfiguraciju i osigura da ih stroj ne ponovi. Međutim, model temeljen na dubokom učenju može samostalno ustvrditi da li je njegov zadatak uspješno dovršen koristeći vlastitu umjetnu neuronsku mrežu koja mu omogućuje samostalno učenje i donošenje odluka. To predstavlja osnovnu razliku između spomenutih pojmova koja je još dodatno prikazana na slici 2.6.



Slika 2.6. Usporedba strojnog i dubokog učenja

Dubokim učenjem ne prestaje razvijanje obzirom da ga upijanje novih podataka čini samo bogatijim. Tu je izrazita prednost nad strojnim učenjem koje dostiže plato među svojim izvedbama, što je dočarano slikom 2.7.



Slika 2.7. Grafička usporedba rasta u performansi s količinom podataka u strojnom i dubokom učenju

Iako je duboko učenje prvi put bilo teoretizirano 1980-ih, dva su glavna razloga zašto je tek kasnije porastao interes i korist istog [6] :

- zahtjeva ogromne količine podataka. Npr. za razvoj autonomnog vozila potrebno je nekoliko milijuna slika i tisuće sati videozapisa;
- zahtjeva značajnu računalnu snagu koja je tek u posljednja dva desetljeća postala šire dostupna kao što su grafičko procesorske jedinice, GPU.

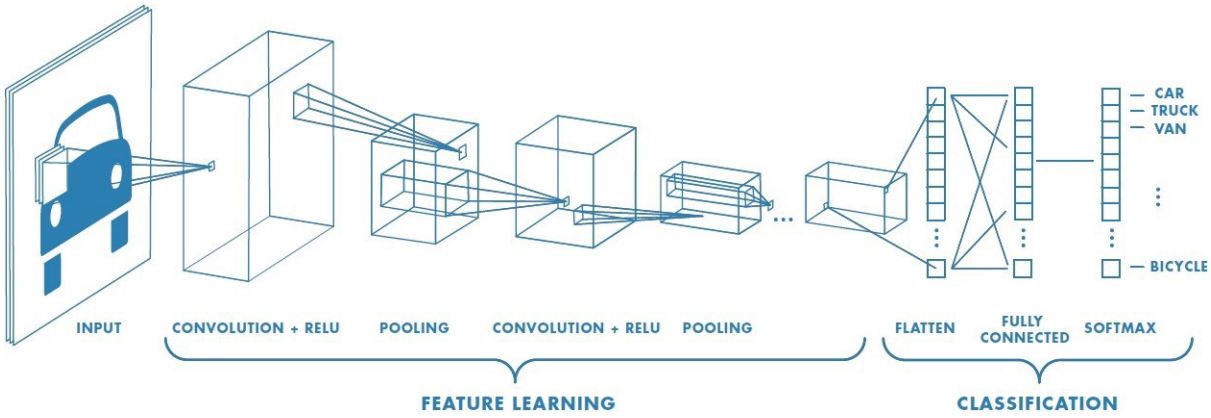
Najpoznatije primjene dubokog učenja su: obrada prirodnog jezika, analiza tržišta i trendova, autonomna vozila i računalni vid.

2.4. Konvolucijske neuronske mreže, CNN

Konvolucijska neuronska mreža (*ConvNet*/CNN) algoritam je dubokog učenja koji može uzeti ulaznu sliku, dodijeliti važnost (naučljive težine i pristranosti) različitim aspektima/objektima na slici i može ih razlikovati. Prethodno obrađivanje koje je potrebno u konvolucijskoj neurnoskoj mreži mnogo je niže u usporedbi s drugim algoritmima klasifikacije. Dok se u primitivnim metodama filtri ručno konstruiraju, uz dovoljno obuke, konvolucijska neuronska mreža ima sposobnost naučiti te značajke. Njihova glavna karakteristika je upravo da uče direktno iz podataka, eliminirajući ručno filtriranje.

Struktura ConvNeta analogna je povezivanju neurona u ljudskom mozgu, odnosno inspirirana organizacijom vizualnog korteksa. Pojedinačni neuroni reagiraju na podražaje samo u ograničenom području vidnog polja poznatom kao receptivno polje. Kolektivno takva polja preklapaju se kako bi pokrili cijelo vizualno područje.

Posebnost ConvNeta su četiri vrste skrivenih slojeva, redom: konvolucijski sloj (engl. *convolutional layer*) po kojem je i dobiveno ime neuronske mreže, sloj udruživanja (engl. *pooling layer*), ReLU i potpuno povezani slojevi (engl. *fully connected layer*). Prikaz jedne konvolucijske neuronske mreže vidljiv je na slici 2.8.



Slika 2.8. CNN sekvenca za klasifikaciju vozila (eng.) [7]

Kovolucijski sloj kroz svoje čvorove primjenjuje filtre na ulaznu sliku. Ne gleda cijelu sliku odjednom, već kroz nju prolazi preklapajućim blokovima piksela. Na taj način se identificiraju i izdvajaju značajke sa slike. Filtri pridodaju vrijednost pikselima koji im odgovaraju, tim veću vrijednost ako im više odgovaraju. Konvertiraju se (prepliću) informacije filtra s ulaznom slikom i stvara se „mapa značajki“ (engl. feature map).

Sloj udruživanja sve vrijednosti piksela u pojedinoj mapi značajki sažmi, što smanji razlučivost mape značajki, time reducira parametre što za nas znači da će i proračun biti kraći. Ovaj korak omogućuje otkrivanje objekata bez obzira na poziciju sa slike i čini mrežu fleksiilnijom. Spriječava se da računalo stavi prevelike težine na značajke.

ReLu uvodi nelinearnost tako što zamijeni negativne vrijednosti piksela nulama i time omogućuje računalu bolje obrađivanje složenih podataka (kao što su slike, a one su nelinearne).

Potpuno povezani slojevi nalik su skrivenim slojevima uobičajene umjetne neuronske mreže. U konvolucijskom sloju, čvorovi primaju ili dijele informacije samo iz dijela prethodnog sloja. U potpuno povezanom sloju, svaki čvor prima izlaz od svakog čvora u prethodnom sloju. U ovim slojevima se kombiniraju sve izdvojene značajke. To znači da računalo vidi cijelu sliku što pomaže u ostvarivanju točnih rezultata. [8]

Konvolucijske neuronske mreže upotreljavaju se zbog sljedećih razloga:

- eliminiraju manualno izvlačenje značajki,
- rezultiraju visokom točnošću kod prepoznavanja,
- mogu se prenamijeniti za nove zadatke prepoznavanja.

Najčešća uporaba ConvNeta je općenito među zadatcima koji zahtjevaju računalni vid, u klasifikaciji i prepoznavanju slika, poput prepoznavanja lica i rukopisa, prometnih znakova, ali i tumora. Iako mnogo rjeđi, ConvNet se također koristi u analizi videa. Ovo je veći izazov od klasifikacije slika (ionako težak zadatak za računala), jer dodaje mjerenja kretanja u prostoru i vremenu.

2.5. Računalni vid (eng *Computer vision*)

Računalni vid je područje umjetne inteligencije koje osposobljava računala za tumačenje i razumijevanje vizualnog koristeći digitalne slike iz kamera i videa te modele dubinskog učenja strojevi mogu točno identificirati i klasificirati objekte, a zatim reagirati na ono što "vide".
Rani eksperimenti s računalnim vidom odvijali su se 1950-ih koristeći neke od prvih neuronskih mreža za otkrivanje rubova predmeta i sortiranje jednostavnih objekata u kategorije kao što su krugovi i kvadrati. U 1970-ima, prva komercijalna uporaba računalnog vida tumačila je tipkani ili rukom pisani tekst korištenjem optičkog prepoznavanja znakova. Ovaj napredak korišten je za tumačenje pisanog teksta za slijepe. Kako je internet sazrijevao 1990-ih, čineći

velike skupove slika dostupnima za analizu, programi za prepoznavanje lica su cvjetali. Ovi rastući skupovi podataka pomogli su strojevima da identificiraju određene osobe na fotografijama i videozapisima. [9]
Glavni cilj računalnog vida je preslikavanje sposobnosti ljudskog vida pomoću koristeći tri glavna procesa [10]:

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

Bitna je razlika između samog računalnog vida i obrade slike. Svi filteri koje danas jednostavno koristimo na pametnim telefonima spadaju pod obradu slike, ali ne bi bili mogući bez računalnog vida. Dakle, obradom stvara se nova slika na temelju postojeće, a računalnim vidom razumije se sadržaj na istoj.

Određenu razinu vizualnog razumijevanja dostiže se kroz [10]:

- **klafisikaciju objekata** tijekom treniranja modela na skupu podataka, a potom model klasificira objekte na novim podacima;
- **identifikaciju objekata** kojom model prepoznaje specifične instance objekata.

Detaljnije ćemo se posvetiti sada primjeni računalnog vida, jer je izrazito interesantna i nevjerojatno široka, a nadasve korisna. Koristi se u prepoznavanju rukopisa radi digitalizacije rukom pisanih materijala. Video analizama i računalnim vidom se prepoznaju brzine kretanja objekata ili same kamere. Rekonstrukcijom scene se stvara 3D model scene iz unesenih slika i videa. Računalni vid nam pomaže u restauraciji slika. Prepoznavanje lica ili otiska prsta koje neki svakodnevno upotrebljavaju kako bi otključali svoj pametni telefon (*FaceID, TouchID*). Nadzorne kamere u sklopu s računalnim vidom omogućuju detekciju sumnjivog ponašanja, traženja kriminalaca. Pametna vozila prepoznaju prometne znakove, svijetla, crte na cesti, rubove i ljude. Na internetu filtrira se nepoželjan i neadekvatan sadržaj među slikama. Prepoznavanje uzorka. Segmentacija slike dijeli sliku na više regija ili dijelova koji se zasebno ispituju. Recimo, moglo bi se pomoću segmentacije i detekcije objekata na nogometnom terenu razdijeliti igrači, teren i lopta te iz njihovog ponašanja izvući tko najduže vodi loptu, na čijem dijelu terena se lopta najviše zadržava, a onda i s prepoznavanjem uzorka na koji način su uspješni i zabijaju golove. Računalni vid se još uvelike koristi za inspekciju strojeva, raketa i aviona, u medicini i za zabavu u virtualnoj i proširenoj stvarnosti.

3. IZRADA I RJEŠENJE

Potrebno je segmentirati osobe na slikama pomoću konvolucijske neuronske mreže. Za to je potrebno odabrati model, skup podataka za treniranje i skup podataka za testiranje. Zatim je potrebno odabrani model istrenirati, izdvojiti dobivena svojstva i eksperimentalno evaluirati. Primjer segmentacije, kolorizirano, predstavljen je na slici 3.1.

Slika 3.1. Kolorizirana segmentacija osoba na slikama

Bitno za napomenuti, na mnogo načina se probao razriješiti problem ovog zadatka, na sreću čitatelja, biti će prikazan samo uspijeh i kako se ovaj rad jednostavno može ponoviti i samostalno isprobati s programskim kodom koji je dorađen i time razriješen svih usputnih problema na koje se naišlo. Za ovaj rad su stoga potrebni svega par alata koji su svakom znalcu umjetne inteligencije već dobro poznati: Colaboratory, Google Drive te COCO baza podataka.

3.1. Gdje je primjena segmentacije ljudi na slikama?

Sama segmentacija za sebe bi bila od manje koristi, ali ju se može koristiti u daljnjem radu za npr. detekciju pješaa na cesti za autonomno vozilo ili pametno vozilo što bi pridodalo sigurnosti na cesti. Razdvajanje osobe od pozadine za bolje portretne slike na kamerama, ili izoštravanje na nadzornim kamerama. Mogao bi se brojati odaziv ljudi, a svakako bi se kasnije moglo ljude dodatno grupirati ponovo daljnjim dubokim učenjem.

3.2. Python, Jupiter Notebook i korištene bilioteke

3.2.1. Programski jezik Python

Python je interpreterski, interaktivni, objektno orijentirani programski jezik. Python je dobar odabir zbog svoje rasprostranjenosti, ima veliku zajednicu, besplatan je i jednostavan, te sadrži opsežnu količinu modula za rad. Razlog više za odabir je to što je Python najpopularniji programski jezik za strojno učenje. Kao jezik za rad na strojnom i dubokom učenju, efikasan je, jednostavno se integrira s drugim programskim jezicima te ima aktivnu podršku zajednice i biblioteka otvorenog koda, tako da je rijetko više potrebno raditi vlasiti algoritam i umjetne neuronske mreže iz temelja. [11]

3.2.2. Jupiter Notebook

U ovom radu korišten je i Jupyter Notebook. Jupyter Notebook ili Jupiter bilježnica 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.). Na taj način, izrađeni programi su istovremeno izvršni dokumenti koji se mogu pokrenuti za analizu podataka i čitljivi dokumenti koji sadrže opise analiza i rezultate. Njegove glavne značajke su pisanje, uređivanje i izvođenje programskog koda u internet pregledniku, potpuno ili po dijelovima, te bogati izbor medija za vizualnu reprezentaciju. [12]

3.2.3. Biblioteke

Knjižnica funkcija ili jednostavnije biblioteka (za neke *Paketi*) je zbirka prethodno kombiniranih kodova. Korištenje istih ubrzava rad i svakako olaškava život svakom developeru. Svi paketi u Pythonu su za ovaj rad bili instalirani pomoću *conde, pip-a* i *terminala*, a to je tako da se u terminal upiše "conda install --name myenv [željeni paket]", no može se i putem pip-a na način "pip install [potrebni paket]".

Tensor Flow je biblioteka otvorenog koda, u ovom slučaju Pythonova, razvijena od strane Googlea a dio je gotovo svake Googleove aplikacije za strojno učenje. Značajke koje opisuju Tensorflow su [13]:

- laka vizualizacija svih dijelova računskih grafova,
- fleksibilnost,
- lako provođenje treninga,

- paralelni trening neuronskih mreža.

Keras se smatra jednom od najboljih biblioteka za strojno učenje u Pythonu. Omogućuje lakši mehanizam za izražavanje neuronskih mreža. Keras također nudi neke od najboljih usluga za kompajliranje modela, obradu skupova podataka, vizualizaciju grafikona i još mnogo toga. Značajke Kerasa [13]:

- Radi glatko i na CPU-u i na GPU-u.
- Keras podržava gotovo sve modele neuronske mreže – potpuno povezane, konvolucijske, skupne, rekurentne, ugrađene itd. Nadalje, ti se modeli mogu kombinirati za izgradnju složenijih modela.
- Budući da je modularan po prirodi, Keras je nevjerojatno izražajan, fleksibilan i podesan za inovativna istraživanja.
- Keras je okvir koji se u potpunosti temelji na Pythonu, što olakšava otklanjanje pogrešaka i istraživanje.

Pandas je biblioteka za strojno učenje u Pythonu koja pruža strukture podataka visoke razine i širok izbor alata za analizu. Jedna od sjajnih značajki ove biblioteke je mogućnost prevođenja složenih operacija s podacima pomoću jedne ili dvije naredbe. Pande imaju mnogo ugrađenih metoda za grupiranje, kombiniranje podataka i filtriranje, kao i funkcionalnost vremenskih serija. Pande se brinu da cijeli proces manipuliranja podacima bude lakši. Podrška za operacije kao što su ponovno indeksiranje, iteracija, sortiranje, agregacije, ulančavanja i vizualizacije među istaknutim su značajkama Pandasa. [13]

Scikit-Learn je Python biblioteka povezana s NumPy i SciPy. Smatra se jednom od najboljih knjižnica za rad sa složenim podacima. Sadrži brojne algoritme za implementaciju standardnog strojnog učenja i zadataka rudarenja podataka kao što su smanjenje dimenzionalnosti, klasifikacija, regresija, klasteriranje i odabir modela. [13]

NumPy je Python biblioteka koja se koristi za rad s poljima (eng. *array*) i predstavlja numeričko matematičko proširenje Pythona. TensorFlow i druge biblioteke interno koriste Numpy za izvođenje niza operacija na tenzorima. Značajke koje opisuju Numpy su [13]:

- interaktivnost i jednostavna upotreba,
- jednostavna implementacija složenih matematičkih izraza,
- intuitivan za programiranje.

Matplotlib je biblioteka za crtanje, odnosno vizualizaciju podataka, za Python i Numpy. Jedno od najvažnijih svojstava Matplotliba je da dobro radi u nizu operativnih sustava neovisno o izlaznom formatu.

Open Source Computer Vision ili **OpenCV** je biblioteka računalnog vida i strojnog učenja otvorenoga koda. OpenCV je izvorno napisan u C++ i ima predložak sučelja koji besprijekorno radi sa STL spremnicima. Knjižnica funkcija ima više od 2500 optimiziranih algoritama što uključuje opsežan skup klasičnih i najsuvremenijih algoritama računalnog vida i strojnog učenja. Ovi se algoritmi mogu koristiti za prepoznavanje lica, identificiranje objekata, klasificiranje ljudskih radnji u videozapisima, praćenje pokreta kamere, praćenje pokretnih objekata, izdvajanje 3D modela objekata, proizvodnju 3D oblaka točaka iz stereo kamera, spajanje slika kako bi se proizvela visoka rezolucija slike cijelog prizora, pronalazak slične slike iz baze podataka, uklonjanje crvenih očiju sa slika snimljenih bljeskalicom, praćenje pokreta očiju, prepoznavanje krajolika i postavljanja markera za prekrivanje s proširenom stvarnošću, itd. Uz dobro etablirane tvrtke kao što su Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota koje zapošljavaju biblioteku, postoji mnogo startupova kao što su Applied Minds, VideoSurf i Zeitera, koji u velikoj mjeri koriste OpenCV. Raspoređene upotrebe OpenCV-a obuhvaćaju raspon od spajanja slika uličnog prikaza, otkrivanja upada u videonadzoru u Izraelu, praćenja rudničke opreme u Kini, pomaganja robotima u navigaciji i skupljanju objekata u Willow Garageu, otkrivanja nesreća utapanja u bazenima u Europi, pokretanja interaktivne umjetnosti u Španjolska i New York, provjera pista za krhotine u Turskoj, pregled naljepnica na proizvodima u tvornicama diljem svijeta do brzog prepoznavanja lica u Japanu. [14]

3.3. Odabir modela

Model je odabran, a programski kod dorađen te se može pronaći na GitHubu pod [15]:
Matlabich/Semantic-Segmentation-using-AutoEncoders
forked from animikhaich/Semantic-Segmentation-using-AutoEncoders
https://github.com/Matlabich/Semantic-Segmentation-using-AutoEncoders/tree/feat/google-colaboratory

3.4. Colaboratory

Colab ili Colaboratory, omogućuje pisanje i izvršavanje Pythona u pregledniku (Cola Notebook). Colab Notebooks omogućuju kombiniranje izvršnog koda i obogaćenog teksta u jednom dokumentu, zajedno sa slikama, HTML -om, LaTeX -om itd. Izrađene vlastite Colab Notebooks pohranjuju se na osobni Google Drive račun, a mogu se i jednostavno dijeliti s kolegama, dopuštajući im da ih komentiraju ili čak uređuju. Colab Notebook je Jupyter Notebook koji hostira Colab. Rad Colaba je s:

- minimalno potrebne konfiguracije,
- besplatnim pristupom GPU-ima,
- jednostavnim dijeljenjem kolegama.

Bilo da je u pitnaju student, data scientist ili AI researcher, Colab može olakšati rad, jer omogućuje korištenje pune snage popularnih Python biblioteka za analizu i vizualizaciju podataka. Mogu se uvesti vlastiti podaci sa svog Google Drive računa, uključujući iz proračunskih tablica, kao i iz Githuba i mnogih drugih izvora. Pomoću Colaba može se uvesti skup podataka o slikama, uvježbati klasifikator slika na njemu i procijeniti model. Colab Notebooks izvršavaju programski kod na Googleovim poslužiteljima u oblaku, što znači da se može iskoristiti snaga Google hardvera, uključujući GPU i TPU, bez obzira na snagu osobnog računala. Koristi se intenzivno u zajednici strojnog učenja s aplikacijama koje uključuju [16]:

- početak rada s TensorFlowom,
- razvijanje i treniranje neuronskih mreža,
- eksperimentiranje s TPU-ima,
- širenje istraživanja umjetne inteligencije,
- izradu tutorijala.

Sve što je potrebno za rad je preglednik.

3.5. Korištene verzije za rad

Bitno je napomenuti koje su verzije korištene u ovom radu kako bi se za potencijalne buduće neuspijehe moglo provjeriti je li problem nastao zbog bitnih promijena u nekoj drugoj korištenoj verziji. U nastavku su izlistane knjižnice funkcija, programski jezik i platforma te verzija s kojom su instalirane, a isto je napravljeno i u programskom kodu modela:

- Python Platform: Linux-5.4.188+-x86_64-with-Ubuntu-18.04-bionic
- Tensor Flow Version: 2.4.1
- Keras Version: 2.4.0

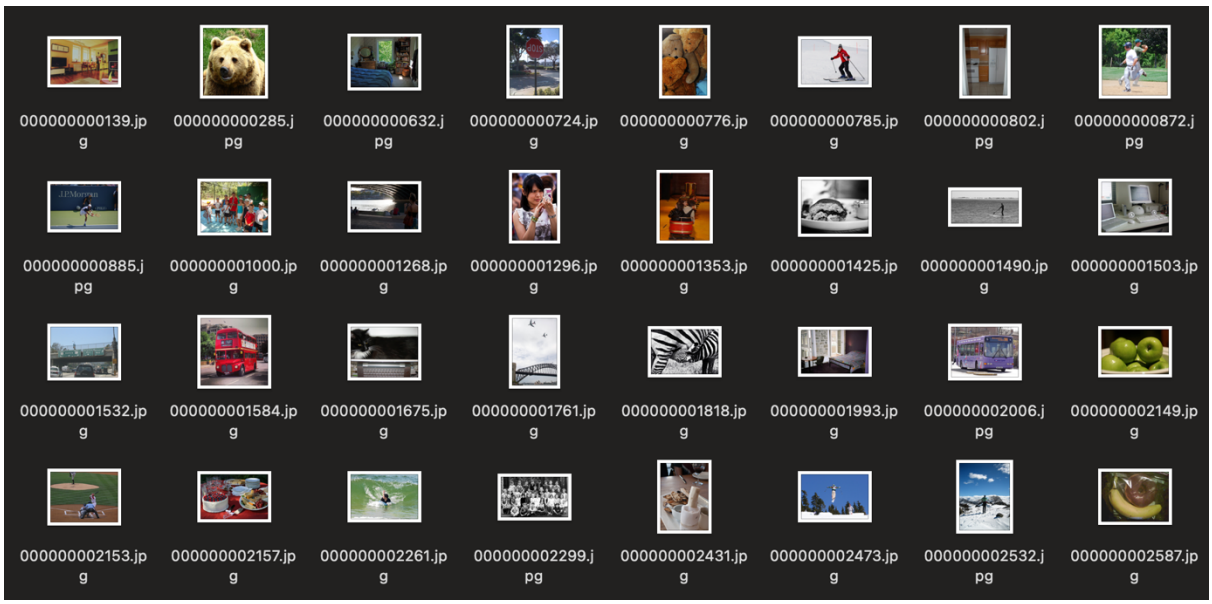
- Python 3.7.13 (default, Apr 24 2022, 01:04:09) [GCC 7.5.0]
- Pandas 1.3.5
- Scikit-Learn 1.0.2
- Numpy 1.19.5
- OpenCV 4.5.1.48
- Matplotlib 3.5.3

3.6. Baza podataka

Baza podataka preuzeta je uz COCO koji je veliki skup podataka za otkrivanje objekata, segmentaciju i označavanje, a ime dobiva kao skraćenica za *Common Objects in Context*. Na slici 3.2. vidi se primjer anotiranih objekata koji se mogu pronaći na COCO, a za prikaz označenih objekata su uzete osobe obzirom da takve objekte i tražimo. [17]

Slika 3.2. Sirovi primjer anotiranih objebjekata (*person*) direktno s COCOa [17]

Za rad se koristio prikladan *dataset* osoba (eng. *person*) maskama sa slika skinutih s COCOa. Skinulo se 64 115 slika za treniranje i 2 693 slika za validaciju, definiranih u programskom kodu kao *train* i *val*. Za oboje su bile potrebne maske koje su izdvajale objekte osoba anotiranih na slikama. Maske su, također slike, s crnim pikselima gdje se ne nalazi traženi objekt i bijelim pikselima gdje se nalazi traženi objekt na slici. U nastavku se vidi dio preuzetog dataseta na slici 3.3. i primjer maski na slici 3.4.



Slika 3.3. Dio dataseta

Slika 3.4. Dva primjera slike i njene segmentacijske maske za *osobe*

Pripremu dataseta je dovoljno napraviti jednom, spremiti pa dalje koristiti iste već pripremljene podatke. U nastavku je prikazan programski kod kojim se radi **priprema** korištenjem COCO API-a za maske radi segmentacije tražene klase *person*. Obzirom da se sve izvodi samo uz pomoć Google Drive-a, COCOa i Colaboratory-a, potrebno je u Colab Notebook povezati

Google Drive (skraćeno gDrive) na kojem će biti postavljene datoteke slika za trening i validaciju, te maske, što je predstavljeno početnim kôdom na slici 3.5.

```
# Mount Google Drive to /content/gdive dir
from google.colab import drive
drive.mount('/content/gdrive')

# List the files in the My Drive dir
!ls "/content/gdrive/My Drive/"
```

Slika 3.5. Spajanje s Colab Notebooka na Google Drive radi spremanja i čitanja podataka

Crveni uskličnik "!" koji se može vidjeti ispred naredbi, a bit će ih još takvih, označava naredbu za terminal. U ovom slučaju "*ls*" nam izlistava pronađeno u direktoriju, odnosno na spojenom gDriveu, kao što to radi obično u terminalu.

Obzirom da je odabrana lokacija gdje će se sve spremati, može početi preuzimanje s COCOa. Slijede koraci s naredbama za terminal u kojem se stvara novi direktoriji, preuzima dataset za trening i dataset za validaciju, te anotacije istih s COCOa, a isto je vidljivo na slici 3.6. Među naredbama se vidi kako su preuzeti datasetovi zipirani, pa su iz istog razloga unzipani, a zip datoteka je obrisana s "rm" (eng. *remove*).

```
!mkdir COCO

!wget http://images.cocodataset.org/zips/train2017.zip
!unzip train2017.zip
!rm train2017.zip

!wget http://images.cocodataset.org/zips/val2017.zip
!unzip val2017.zip
!rm val2017.zip

#!wget http://images.cocodataset.org/zips/test2017.zip
#!unzip test2017.zip
#!rm test2017.zip


!cd ../
!wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip
!unzip annotations_trainval2017.zip
!rm annotations_trainval2017.zip
```

Slika 3.6. Preuzimanje dataseta s COCOa

Programski kod se nastavlja učitavanjem potrebnih knjižnica funkcija i paketa, te odmah pripremu puta do direktorija u varijable gdje će spremiti izvučene i obrađene podatke, slika 3.7.

```
from pycocotools.coco import COCO
import os
import cv2
import time
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
import random
```

```
# Annotation Files
train_ann = "/content/annotations/instances_train2017.json"
val_ann = "/content/annotations/instances_val2017.json"

train_img_dir = '/content/train2017/'
val_img_dir = '/content/val2017/'

train_dest_segmentation_masks = '/content/segmentations/train'
val_dest_segmentation_masks = '/content/segmentations/val'
```

Slika 3.7. Učitavanje biblioteka te spremanje puta direktorija u varijable

Slijedi kratak programski kod za inicijalizaciju anotiranih objekata na slici 3.8.

```
# Initialize the COCO objects
train_coco = COCO(train_ann)
val_coco = COCO(val_ann)
```

Slika 3.8. Učitavanje COCO anotacija iz prethodno definiranih direktorija

U bazi anotacija su zapisani objekti koji se nalaze na slici za svaku sliku iz učitane baze i uz svaku anotaciju je zapisano gdje se točno na toj slici nalazi taj objekt. Definirane funkcije za izdvajanje klase *person* među anotacijma i pravljenje maski za te iste slike prikazane su na slici 3.9. Unutar prve funkcije prvo se pretpostavlja potpuno crna maska napravljena koristeći matricu nula, *zeros*, što daje crnu boju piksela. Tek nakon posvjetljivanja piksela na mjestima anotacije, za svaku oznaku pod kategorijom *person* pa uzimanja one slike koja ima više svijetlijih piksela, ostaje za kraj maska. *dtype=np.uint8* određuje da će skala za boju piksela biti jedna umjesto RGB skale, odnosno može kretati između crne i bijele, u nijansama sivih. Nakon što prođe kroz sve oznake iz tražene kategorije, vraća ime slike i masku. Piksele na maski pomnoži s 255 kako bi ostale samo crni, nula, i bijeli, 255 pikseli (nakon umnoška najveći mogući rezultat ostaje 255).

```
def get_mask(coco_object, index, filtered_cat_id, filtered_img_id, img_dir):
    # Load Image
    img_meta = coco_object.loadImgs(filtered_img_id[index])
    img_path = os.path.join(img_dir, img_meta[0]['file_name'])
    img = cv2.imread(img_path)

    # Load Annotations
    annIds = coco_object.getAnnIds(imgIds=img_meta[0]['id'], catIds=filtered_cat_id, iscrowd=None)
    anns = coco_object.loadAnns(annIds)

    # Generate Mask
    anns_img = np.zeros((img.shape[:-1]), dtype=np.uint8)
    for ann in anns:
        anns_img = np.maximum(anns_img, coco_object.annToMask(ann)*ann['category_id'])

    return img_meta[0]['file_name'], anns_img * 255


def write_mask(coco_object, index, filtered_cat_id, filtered_img_id, dest_folder, img_dir):
    if not os.path.isdir(dest_folder):
        os.makedirs(dest_folder)

    filename, mask = get_mask(coco_object, index, filtered_cat_id, filtered_img_id, img_dir)
    dest_path = os.path.join(dest_folder, filename)
    cv2.imwrite(dest_path, mask)


def write_train_masks(train_coco, train_dest_folder, classes=['person']):
    print("Processing Train Images")
    time.sleep(0.5)
    filtered_cat_id = train_coco.getCatIds(classes)
    filtered_img_id = train_coco.getImgIds(catIds=filtered_cat_id)

    for i in tqdm(range(len(filtered_img_id))):
        write_mask(train_coco, i, filtered_cat_id, filtered_img_id, train_dest_folder, train_img_dir)

    print("Train Segmentation Masks Processing Complete")


def write_val_masks(val_coco, val_dest_folder, classes=['person']):
    print("Processing Val Images")
    time.sleep(0.5)
    filtered_cat_id = val_coco.getCatIds(classes)
    filtered_img_id = val_coco.getImgIds(catIds=filtered_cat_id)

    for i in tqdm(range(len(filtered_img_id))):
        write_mask(val_coco, i, filtered_cat_id, filtered_img_id, val_dest_folder, val_img_dir)

    print("Val Segmentation Masks Processing Complete")
```

Slika 3.9. Definiranje funkcija za izradu traženih maski

Kako bi se rezultati proizveli i spremili, potrebno je provesti sve definirane funkcije. Na slici 3.10. mogu se vidjeti izvršene funkcije, spremanje rezultata u prethodno definirane direktorije i, odmah ispod, izlaz koji je prikazan nakon izvršenja. Datoteka je zipirana, a u nastavku su komentirane terminal naredbe za unzip iste, pomicanje maski treninga u odgovarajuću drugu mapu maski, analogno za maske slika za validaciju.

```
write_train_masks(train_coco, train_dest_segmentation_masks)
write_val_masks(val_coco, val_dest_segmentation_masks)
```

```
Processing Train Images
100%|██████████| 64115/64115 [16:50<00:00, 63.45it/s]
Train Segmentation Masks Processing Complete
Processing Val Images
100%|██████████| 2693/2693 [00:34<00:00, 77.10it/s]
Val Segmentation Masks Processing Complete
```

```
# Save results from segmentation
!zip -r "/content/gdrive/My Drive/segmentations.zip" /content/segmentations
```

```
#!unzip "/content/gdrive/My Drive/segmentations.zip"
#!mv /content/content/segmentations/train /content/COCO/masks/
#!mv /content/content/segmentations/val /content/COCO/masks/
```

Slika 3.10. Provedba funkcija za izradu i spremanje maski

Rezultat ovih radnji su upravu prikazani primjeri na slici 3.4. gdje je u jednom direktoriju pohranjena originalna slika, a u drugom njena maska, i tako cijeli preuzeti dataset. Iz razloga što je ovo dovoljno raditi samo jednom, odvojeno je kao zaseban programski kod za pripremu dataseta pod nazivom "*Dataset Preparation via Colab with Google Drive*.ipynb". Za treniranje modela i validaciju potrebno je pogledati programski kod "*Model via Colaboratory with gDrive*.ipynb".

3.7. Pred pokretanje modela

Za početak ispisat će se infomracije o korištenom GPU i RAMu, a programski kod i ispis moguće je vidjeti na slici 3.11.

```
# Check the GPU stats
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)

from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
Tue Aug 30 10:25:56 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P0    23W / 300W |      0MiB / 16160MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Your runtime has 13.6 gigabytes of available RAM

Not using a high-RAM runtime
```

Slika 3.11. Ispis informacija o korištenom GPU

Obzirom da se sve nadalje izvodi samo pomoću Google Drive-a i Colaboratory-a, potrebno je u Colab Notebook povezati GDrive na kojem su postavljene zip datoteke slika za trening i validaciju, te maske, što je predstavljeno kôdom na slici 3.12.

```
# Mount Google Drive to /content/gdive dir
from google.colab import drive
drive.mount('/content/gdrive')

# List the files in the My Drive dir
!ls "/content/gdrive/My Drive/"
```

```
Mounted at /content/gdrive
'Colab Notebooks'   'Machine Learning'   segmentations.zip
```

Slika 3.12. Spajanje s Colab Notebooka na Google Drive radi pristupa podacima

Programski kod se nastavlja učitavanjem potrebnih knjižnica funkcija i paketa, te odmah i ispis verzija istih, slika 3.13.

```
# Magic word does not work for specific tensorflow version
# %tensorflow_version 2.4

# But this works
!pip install tensorflow-gpu==2.4.1
#!pip install matplotlib==3.3.4
!pip install numpy==1.19.5
!pip install opencv==4.5.1.48

# Hack to make colab working (Env restart required)
!pip uninstall -y matplotlib
!pip install matplotlib
```

```
# What version of Python do you have?
import sys

import tensorflow.keras
import pandas as pd
import sklearn as sk
import tensorflow as tf
import platform

print(f"Python Platform: {platform.platform()}")
print(f"Tensor Flow Version: {tf.__version__}")
print(f"Keras Version: {tensorflow.keras.__version__}")
print()
print(f"Python {sys.version}")
print(f"Pandas {pd.__version__}")
print(f"Scikit-Learn {sk.__version__}")
gpu = len(tf.config.list_physical_devices('GPU'))>0
print("GPU is", "available" if gpu else "NOT AVAILABLE")
print(tf.config.list_physical_devices('GPU'))
```

```
# Imports
import os
os.environ["TF_FORCE_GPU_ALLOW_GROWTH"] = "true"
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2
from glob import glob
from tqdm import tqdm
```

Slika 3.13. Učitavanje biblioteka te ispis verzija istih

Kako bi već postojao instalirani Matplotlib, potrebno je isti deinstalirati i ponovo instalirati, inače bi javljalo grešku i ne bi se mogli prikazivati rezultati.

3.8. Funkcije za učitavanje parova slika i maski te testiranje istih

Nakon što se srede direktoriji i putanje do potrebnih datoteka (slika 3.14.), ujedno se u putanje za slike treninga i slike validacije na kraj *joina* zvijezdica "*" kako bi prilikom korištenja istih u *glob* metodi, njome izvukli sve datoteke iz direktorija i spremili u varijeble *train_images* i *val_images*.

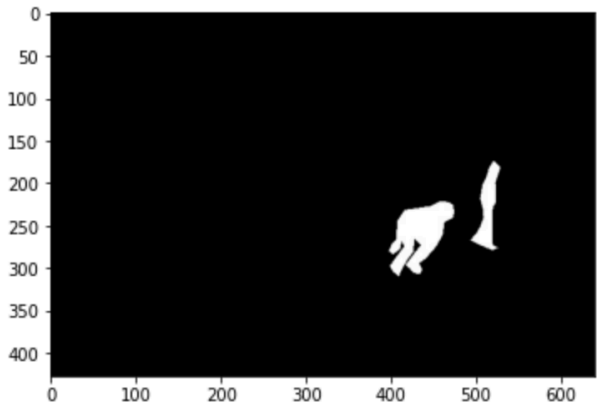
```
train_mask_dir = "/content/COCO/masks/train/"
val_mask_dir = "/content/COCO/masks/val/"

train_images = glob(os.path.join(train_mask_dir, "*"))
val_images = glob(os.path.join(val_mask_dir, "*"))
```

Slika 3.14. Učitavanje svih slika iz direktorija u varijable

Da bi se za probu vidjelo što je dohvaćeno, slijedi plotanje prve, odnosno nulte instance u *train_images* na slici 3.15. Plot s *img[:,:,::-1]* je način na koji ne mijenjamo dimenziju širine, niti visinu, već mijenjamo samo dimenziju kanala i to obrnutim redoslijedom. Izvorni BGR raspored je obrnut od uobičajenog RGB rasporeda kanala pa ga postavljamo u RGB.

```
img_path = train_images[0]
img = cv2.imread(img_path)
print("Dims: ", img.shape)
plt.imshow(img[:,:,::-1])
plt.show()
```

```
Dims:  (427, 640, 3)
```



Slika 3.15. Provjera dohvaćenog materijala pomoću ispisa prve instance

Vidimo da je sve u redu, jer to jedna od maski slika za trening.

Prelazimo na sočniji dio, i to kako učitavati slike i maske za trening, a rješenje je prikazano s klasom DataLoader na slici 3.16. Da bi funkcioniralo, pretpostavljamo sada već da su putanje do datoteka spremne i točne. U konstruktor metodi *__init__()* prepravljamo putanju u slučaju da ne završava sa zvijezdicom "*" kroz metodu *__fix_glob(),* kako bi pomoću *list_files* mteode učitali sve datoteke, ali razmiješano (*shuffle*). Uz *tf.data.experimental.AUTOTUNE* optimizira se raspodijela proračuna GPU-a za sve parametre navedene pod *AUTOTUNE*. U slučaju da se ne zada drukčija veličina, *target_shape* će biti postavljen na (512, 512).

Na dalje je unutar klase DataLoader napisana metoda kao tenzorska funkcija *parse_images*. Pomoću *mask = tf.io.read_file(mask_path)* u varijablu *mask* se sprema pročitana datoteka koja je pronađena putanjom *mask_path*.

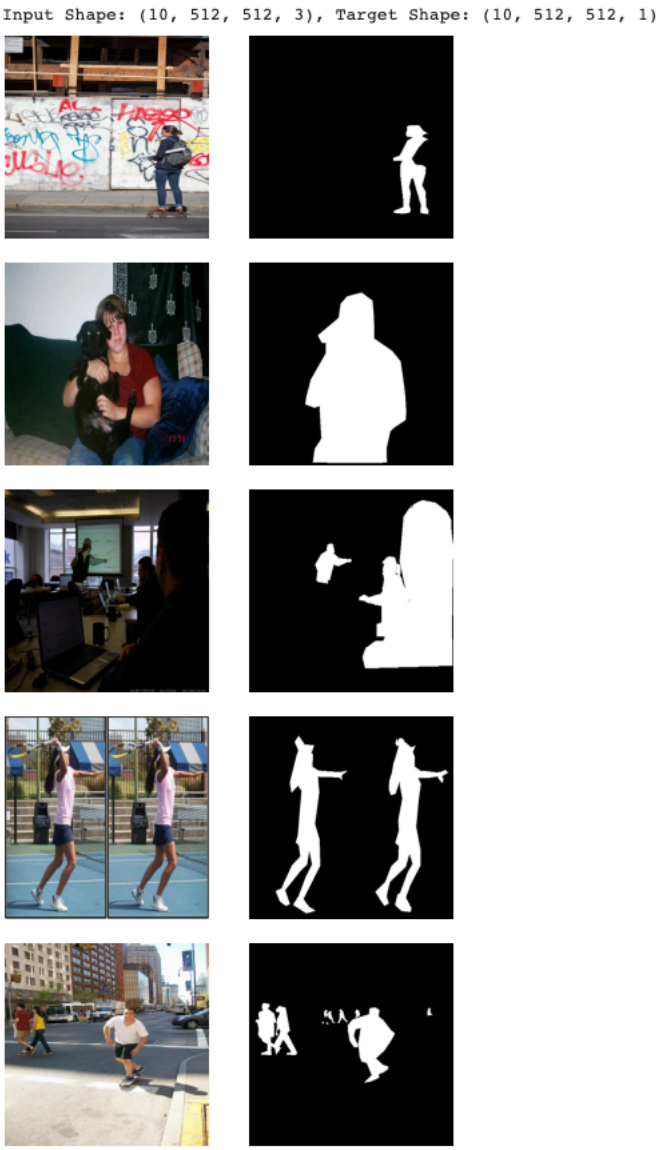
mask = tf.io.decode_jpeg(mask, channels=1) čita jednokanalnu masku, odnosno u crno-bijelim bojama (0, 255).
mask = tf.cast(mask, tf.float32) mijenja integerske piksele u 32-bitni floating point (decimalni zapis), te s time postaje 0.0000 i 255.00000...
image_path = tf.strings.regex_replace(mask_path, "masks", "images") mijenja u putnaji riječ *masks* s riječju *images* i time dobiva put do točne slike iz koje je dobivena maska.
Na dalje radi isto kao i s maskom samo kao RGB kanalna slika.
mask = tf.image.resize(mask, self.target_shape) / 255. dijeli sve piksele s 255.000 i time dobiva masku čije su veličine zapisane između 0 i 1. Nakon analiziranja slike i maske vraća ih iz tenzorske funkcije *parse_images*.

U ovoj klasi nam preostaje još samo jedna, ponovo tenzorska funkcija, *data_generator(self, batch_size=64)*. Ukoliko *batch_size* ne bude unesen, it će zadan 64.
U *self.files* su izlistane sve slike iz odaranog direktorija pa *self.files.map(self.parse_images)* provede funkciju *parse_images* kroz svaku od njih, odnosno mapira *dataset*. Ovo je dobro poznato funkcijsko programiranje.

Preostalo je još samo testirati ako ovako složena klasa daje dobre rezultate.
U DataLoader je ubačena putanja do direktorija s maskama od slika za trening. Čisto radi testa uzet je skup od 10 slika i maski.
inputs, targets = next(iter(batch)) prva izvučena varijabla predstavlja trening sliku koja će biti na uluazu u mrežu, a druga njenu masku koju će mreža targetirati za svoj izlaz.
Na slici 3.17. moguće je vidjeti primjer ispisa testiranja DataLoadera. Ispis je plotam pomoću matričnog plotanja, na skupu od 10 (ima i 10 plotova), svaki od jednog reda i dva stupca, prvo prikazuje ulaze za mrežu, a drugo targetirani izlaz.
Baš slučajno, u testu su bile izabrane interesantne trening slike. Na prvoj je prikazana osoba koja se vozi skateboardom i nosi na ramenima ruksak, gdje je COCO imao odznačen ruksak sa slike kao dio osobe. No, ipak je na drugoj trening slici gdje je prikazana osoba koja drži psa u zagrljaju skupa s njima bila označena u COCO maski kao osoba. Ovo će nam biti posebno interesnanto kasnije kada pomnije budu promatrane slike za validaciju i rezultati.

```python
# A quick Tensorflow Data loader
class DataLoader:
    """
    Assuming Dataset Path Structure:
    - Train Images: /media/ActiveTraining/Datasets/COCO/images/train
    - Val Images: /media/ActiveTraining/Datasets/COCO/images/val
    - Train Masks: /media/ActiveTraining/Datasets/COCO/masks/train
    - Val Masks: /media/ActiveTraining/Datasets/COCO/masks/val
    """
    def __init__(self, masks_dir, target_shape=(512, 512)):
        masks_dir = self.__fix_glob(masks_dir)
        self.files = tf.data.Dataset.list_files(masks_dir, shuffle=True)
        self.AUTOTUNE = tf.data.experimental.AUTOTUNE
        self.target_shape = target_shape

    def __fix_glob(self, dir_path):
        if not dir_path.endswith("*"):
            dir_path = os.path.join(dir_path, "*")

        return dir_path

    @tf.function
    def parse_images(self, mask_path):
        mask = tf.io.read_file(mask_path)
        mask = tf.io.decode_jpeg(mask, channels=1)
        mask = tf.cast(mask, tf.float32)

        image_path = tf.strings.regex_replace(mask_path, "masks", "images")
        image = tf.io.read_file(image_path)
        image = tf.io.decode_jpeg(image, channels=3)
        image = tf.cast(image, tf.float32)

        mask = tf.image.resize(
            mask,
            self.target_shape,
        ) / 255.

        image = tf.image.resize(
            image,
            self.target_shape,
        ) / 255.

        return image, mask

    @tf.function
    def data_generator(self, batch_size=64):
        dataset = self.files.map(self.parse_images)
        dataset = dataset.repeat()
        dataset = dataset.batch(batch_size)
        dataset = dataset.prefetch(buffer_size = self.AUTOTUNE)
        return dataset

test_dataset = DataLoader(train_mask_dir)
batch = test_dataset.data_generator(10)
inputs, targets = next(iter(batch))
print(f"Input Shape: {inputs.shape}, Target Shape: {targets.shape}")

for i, t in zip(inputs, targets):
    plt.subplot(1, 2, 1)
    plt.imshow(i)
    plt.axis("off")
    plt.subplot(1, 2, 2)
    plt.imshow(t, cmap="gray")
    plt.axis("off")
    plt.show()
```

Slika 3.16. DataLoader i njegovo testiranje



Slika 3.17. Ispis nakon testiranja DataLoadera

3.9. Model

Prvo su definirani Callbackovi za rano zaustavljanje tijekom treniranja, spremanje najbolje dobivenih težina nakon svake epohe tijekom učenja u *weights.h5*, te podešavanje *Learning Rate-a* nakon postignutog *platoua* (malih promijena u greškama, slaba promijena u novonaučenog, može ukazati na mogući *overfitting*). Learning Rate se uobičajeno nalazi između 0.01 i 0.1, a može imati vrijednost između 0 i 1.

Na slici 3.18. su prikazana sva tri Callbacka u programskom kodu.

```
# Callbacks
early_stop_cb = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=10,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=True,
)
model_ckpt_cb = tf.keras.callbacks.ModelCheckpoint(
    "weights.h5",
    monitor="val_loss",
    verbose=0,
    save_best_only=True,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
)
reduce_lr_cb = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.1,
    patience=4,
    verbose=1,
    mode="auto",
    min_delta=0.0001,
    cooldown=0,
    min_lr=10e-8,
)
```

Slika 3.18. Callbacks

Slijedi prikaz arhitekture umjetne neuronske mreže kroz programski kod na slici 3.19.

Mogu se isčitati uzete veličine skupova od 64, 128 te 1. Ova umjetna neuronska mreža se sastoji od ulaznog, 16 skrivenih slojeva i izlaznog sloja. Među skrivenim slojevima nalaze se 10 konvolucijskih, 2 MaxPooling sloja, 2 UpSampling i 2 Add sloja.

Konvencionalno, prvi ConvLayer odgovoran je za snimanje značajki niske razine kao što su rubovi, boja, orijentacija gradijenta itd. S dodanim slojevima, arhitektura se također prilagođava značajkama visoke razine, dajući mrežu koja ima dobro sveukupno razumijevanje slike u skupu podataka. Postoje dva moguća rezultata konvolucijskog sloja — jedan u kojem je konvolvirana značajka smanjene dimenzionalnosti u usporedbi s ulazom, a drugi u kojem je dimenzionalnost ili povećana ili ostaje ista. To se postiže primjenom Valid Padding u slučaju prvog ili Same Padding u slučaju drugog. [18]

Pooling slojevi smanjuju računalnu snagu potrebnu za obradu podataka kroz smanjenje dimenzionalnosti. Nadalje, koristan je za izdvajanje dominantnih značajki koje su rotacijske i pozicijske invarijantne, čime se održava proces učinkovite obuke modela. Postoje dvije vrste udruživanja: maksimalno udruživanje i prosječno udruživanje. MaxPooling vraća maksimalnu

vrijednost iz dijela slike pokrivenog kernelom. S druge strane, Average Pooling vraća prosjek svih vrijednosti iz dijela slike koji pokriva kernel. Max Pooling također djeluje kao prigušivač buke. U potpunosti odbacuje bučne aktivacije i također izvodi uklanjanje šuma zajedno sa smanjenjem dimenzionalnosti. S druge strane, Average Pooling jednostavno izvodi smanjenje dimenzionalnosti kao mehanizam za suzbijanje šuma. Stoga možemo reći da Max Pooling ima puno bolju izvedbu od Average Pooling-a. [18]

```
# Model Architecture

input_img = tf.keras.Input(shape=(None, None, 3))

# encoding architecture
x1 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(input_img)
x2 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(x1)
x3 = tf.keras.layers.MaxPool2D(padding='same')(x2)
x4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x3)
x5 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x4)
x6 = tf.keras.layers.MaxPool2D(padding='same')(x5)
encoded = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x6)

# decoding architecture
x7 = tf.keras.layers.UpSampling2D()(encoded)
x8 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x7)
x9 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x8)
x10 = tf.keras.layers.Add()([x5, x9])
x11 = tf.keras.layers.UpSampling2D()(x10)
x12 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(x11)
x13 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(x12)
x14 = tf.keras.layers.Add()([x2, x13])
decoded = tf.keras.layers.Conv2D(1, (3, 3), padding='same',activation='relu')(x14)
autoencoder = tf.keras.Model(input_img, decoded)
autoencoder.summary()
```

Slika 3.19. Arhitektura umjetne neuronske mreže modela

Izlazni sloj nalazi se pod *decoded*. Pomoću *autoencoder.sumary()* izlistaju se podaci o umjetnoj neuronskoj mreži koji se mogu vidjegi na slici 3.20.

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, None, None,  0
_________________________________________________________________________________________________
conv2d (Conv2D)              (None, None, None, 6 4864        input_1[0][0]
_________________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, None, None, 6 102464      conv2d[0][0]
_________________________________________________________________________________________________
max_pooling2d (MaxPooling2D) (None, None, None, 6 0           conv2d_1[0][0]
_________________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, None, None, 1 73856       max_pooling2d[0][0]
_________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, None, None, 1 147584      conv2d_2[0][0]
_________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, None, None, 1 0         conv2d_3[0][0]
_________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, None, None, 2 295168      max_pooling2d_1[0][0]
_________________________________________________________________________________________________
up_sampling2d (UpSampling2D) (None, None, None, 2 0           conv2d_4[0][0]
_________________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, None, None, 1 295040      up_sampling2d[0][0]
_________________________________________________________________________________________________
conv2d_6 (Conv2D)            (None, None, None, 1 147584      conv2d_5[0][0]
_________________________________________________________________________________________________
add (Add)                    (None, None, None, 1 0           conv2d_3[0][0]
                                                              conv2d_6[0][0]
_________________________________________________________________________________________________
up_sampling2d_1 (UpSampling2D) (None, None, None, 1 0         add[0][0]
_________________________________________________________________________________________________
conv2d_7 (Conv2D)            (None, None, None, 6 204864      up_sampling2d_1[0][0]
_________________________________________________________________________________________________
conv2d_8 (Conv2D)            (None, None, None, 6 102464      conv2d_7[0][0]
_________________________________________________________________________________________________
add_1 (Add)                  (None, None, None, 6 0           conv2d_1[0][0]
                                                              conv2d_8[0][0]
_________________________________________________________________________________________________
conv2d_9 (Conv2D)            (None, None, None, 1 577         add_1[0][0]
=================================================================================================
Total params: 1,374,465
Trainable params: 1,374,465
Non-trainable params: 0
_________________________________________________________________________________________________
```

Slika 3.20. Podaci o modelu i broju parametara

Nakon nekolicine treniranja ovog modela, ustanovljeno je da bi učenje moglo biti prekinuto ranije, već nakon 50. epohe, obzirom da se greška već iznimno malo smanjuje. Iz tog razloga u programskom kodu je hardkodiran stop na toj epohi. Learning Rate ili stopa učenja je za početak postavljena na 0.001, što je moguće vidjeti na slici 3.21.

Može se primjetiti korištenje Adamove optimizacije *tf.keras.optimizers.Adam().* Adamova optimizacija je stohastička gradijentna metoda spuštanja koja se temelji na adaptivnoj procjeni momenata prvog i drugog reda. Zadane vrijednosti Adamove optimizacije, koje se i ovdje kao takve koriste su: *learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07*. [19]

```
# Hyper Params
batch_size = 32
epochs = 50 # 50
lr = 0.001
target_shape=(224, 224)

opt = tf.keras.optimizers.Adam(learning_rate=lr)

autoencoder.compile(
    loss="binary_crossentropy",
    optimizer=opt,
    metrics=[
        "mse",
])

# Dataset Generator
train_dataset = DataLoader(train_mask_dir, target_shape=target_shape)
train_generator = train_dataset.data_generator(batch_size)

val_dataset = DataLoader(val_mask_dir, target_shape=target_shape)
val_generator = val_dataset.data_generator(batch_size)

# Calculate Num Steps
train_steps = len(list(train_dataset.files)) // batch_size + 1
val_steps = len(list(val_dataset.files)) // batch_size + 1
```

Slika 3.21. Pripremanje svih parametara za model pred početka treniranja

3.10. Trening

Nakon dohvaćenih skupa parova maski i slika, iz treninga radi učenja, ali i iz validacije radi procjene greške, može se pokrenuti treniranje, slika 3.22.

```
# Train
history = autoencoder.fit(
    train_generator,
    validation_data=val_generator,
    batch_size=batch_size,
    steps_per_epoch=train_steps,
    validation_steps=val_steps,
    epochs=epochs,
    callbacks=[
        early_stop_cb,
        model_ckpt_cb,
        reduce_lr_cb
    ])
```

Slika 3.22. Pokretanje treninga

Po ispisu na slici 3.23. su vidljivi međuzaključci u tijeku učenja. Model je primjetio kako je naišao na plato nakon 19. epohe, odnosno slao smanjenje u grešci između targeta i predikta, stoga je pozvao Callback *reduce_lr_cb* i time promijenio stopu učenju kako bi unaprijedio daljnje treniranje.

```
Epoch 1/50
2004/2004 [==============================] - 755s 372ms/step - loss: 0.4327 - mse: 0.1350 - val_loss: 0.3282 - val_mse: 0.1022
Epoch 2/50
2004/2004 [==============================] - 725s 362ms/step - loss: 0.3177 - mse: 0.0982 - val_loss: 0.2928 - val_mse: 0.0901
Epoch 3/50
2004/2004 [==============================] - 718s 358ms/step - loss: 0.2915 - mse: 0.0898 - val_loss: 0.2838 - val_mse: 0.0872
Epoch 4/50
2004/2004 [==============================] - 717s 358ms/step - loss: 0.2768 - mse: 0.0847 - val_loss: 0.3195 - val_mse: 0.1003
Epoch 5/50
2004/2004 [==============================] - 716s 358ms/step - loss: 0.2727 - mse: 0.0833 - val_loss: 0.2672 - val_mse: 0.0809
Epoch 6/50
2004/2004 [==============================] - 724s 361ms/step - loss: 0.2603 - mse: 0.0789 - val_loss: 0.2783 - val_mse: 0.0831
Epoch 7/50
2004/2004 [==============================] - 718s 358ms/step - loss: 0.2694 - mse: 0.0823 - val_loss: 0.2661 - val_mse: 0.0802
Epoch 8/50
2004/2004 [==============================] - 708s 353ms/step - loss: 0.2607 - mse: 0.0791 - val_loss: 0.3186 - val_mse: 0.0988
Epoch 9/50
2004/2004 [==============================] - 701s 350ms/step - loss: 0.2631 - mse: 0.0800 - val_loss: 0.2568 - val_mse: 0.0780
Epoch 10/50
2004/2004 [==============================] - 702s 350ms/step - loss: 0.2508 - mse: 0.0757 - val_loss: 0.2625 - val_mse: 0.0794
Epoch 11/50
2004/2004 [==============================] - 702s 351ms/step - loss: 0.2584 - mse: 0.0782 - val_loss: 0.2462 - val_mse: 0.0737
Epoch 12/50
2004/2004 [==============================] - 704s 351ms/step - loss: 0.2413 - mse: 0.0724 - val_loss: 0.2481 - val_mse: 0.0744
Epoch 13/50
2004/2004 [==============================] - 700s 350ms/step - loss: 0.2400 - mse: 0.0719 - val_loss: 0.2726 - val_mse: 0.0834
Epoch 14/50
2004/2004 [==============================] - 696s 347ms/step - loss: 0.2403 - mse: 0.0721 - val_loss: 0.2540 - val_mse: 0.0774
Epoch 15/50
2004/2004 [==============================] - 703s 351ms/step - loss: 0.2407 - mse: 0.0721 - val_loss: 0.2385 - val_mse: 0.0715
Epoch 16/50
2004/2004 [==============================] - 703s 351ms/step - loss: 0.2331 - mse: 0.0696 - val_loss: 0.2398 - val_mse: 0.0721
Epoch 17/50
2004/2004 [==============================] - 714s 356ms/step - loss: 0.2375 - mse: 0.0711 - val_loss: 0.2469 - val_mse: 0.0739
Epoch 18/50
2004/2004 [==============================] - 702s 350ms/step - loss: 0.2326 - mse: 0.0694 - val_loss: 0.2406 - val_mse: 0.0717
Epoch 19/50
2004/2004 [==============================] - 708s 353ms/step - loss: 0.2356 - mse: 0.0704 - val_loss: 0.2576 - val_mse: 0.0778

Epoch 00019: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
Epoch 20/50
2004/2004 [==============================] - 708s 353ms/step - loss: 0.2208 - mse: 0.0661 - val_loss: 0.2264 - val_mse: 0.0681
```

Slika 3.23. Ispis međuzaključaka po epohama

3.11. Spremanje težina (modela)

Dobivene težine nakon treniranja je potrebno spremiti, slika 3.24., kako ne bi bilo nužne potrebe za ponovnim treniranjem, već se može koristiti naučena umjetna neuronska mreža.

```
!cp /content/weights.h5 /content/gdrive/My\ Drive/50epocha-weights.h5
```

```
#rm -rf /content/weights.h5
```

Slika 3.24. Spremanje težina nakon treninga (komentirano je brisanje istih)

3.12. Pokretanje modela sa spremljenim težinama

Ostavljena je mogućnost pomoću trećeg programskog koda "*Quick Start via Colab with saved weights on gDrive*.ipynb" [15]. U tom kodu su spremljene sve radnje za instaliranje potrebnih knjižnica funkcija i importanja paketa. Ključna naredba za rad sa spremljenim težinama:

autoencoder = tf.keras.models.load_model('/content/gdrive/My Drive/50eWeights.h5')

Ostatak ostaje isti ili dovoljno slični već prokomentiranom.

4. EKSPERIMENTALNA EVALUACIJA

Skup podataka podijeljen u dvije kategorije. Podaci za trening su prvi i veći dio ukupnog skupa podataka, a koristi se isključivo za trening mreže, te iznosi otprilike 95% ukupne veličine skupa podatka. Drugi dio, ostatak, koristi se isključivo za validaciju. Tom podijelom je osigurana nepristranost istreniranog modela, obzirom da mu definitivno time nisu poznata pitanja koja će mu se postaviti nakon učenja. U nastavku su vizualno prikazani te iskomentirani rezultati.

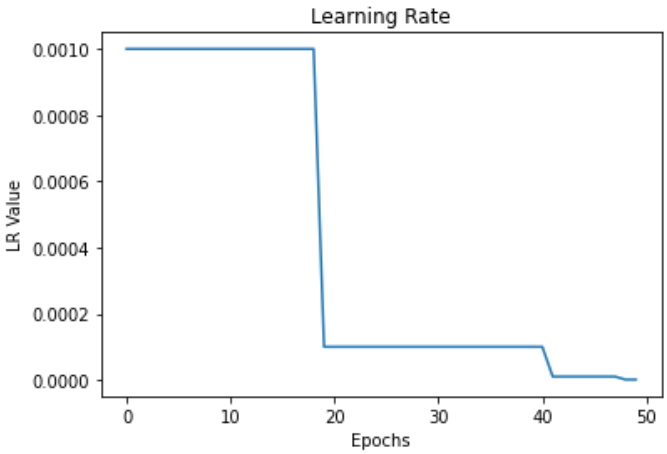
4.1. Rezultati učenja

Nakon posljednje odabrane epohe, mogu se isplotati rezultati učenja. Na slici 4.1. slijedi programski kod za plotanje grafova stope učenja, greške i *mean squared error, MSE*.

```
# Matplotlib Plots (For Easy Rendering)

# Learning Rate
plt.plot(history.history['lr'])
plt.title('Learning Rate')
plt.xlabel("Epochs")
plt.ylabel("LR Value")
plt.show()

# Loss Curves
plt.plot(history.history['loss'], label="Train Loss")
plt.plot(history.history['val_loss'], label="Val Loss")
plt.title('Loss Curves')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

# MSE Curves
plt.plot(history.history['mse'], label="Train MSE")
plt.plot(history.history['val_mse'], label="Val MSE")
plt.title('MSE Curves')
plt.xlabel("Epochs")
plt.ylabel("MSe")
plt.legend()
plt.show()
```

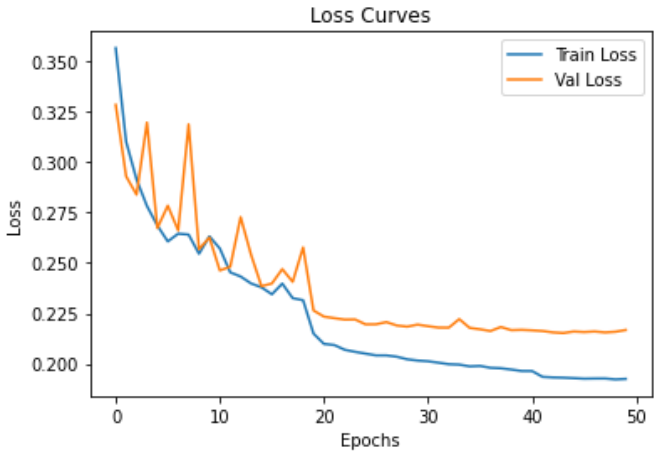
Slika 4.1. Plotanje grafova stope učenja, greške i MSE

Na slici 4.2. prikazan je graf stope učenja. Moguće je točno primjetiti kako se stopa učenja promijenila prilikom Callbacka nakon nailaska na prvi plato iza 19. epohe, kako smo i prethodno vidjeli. Ponovo se primjećuje redukcija stope učenja nakon 40. epohe.

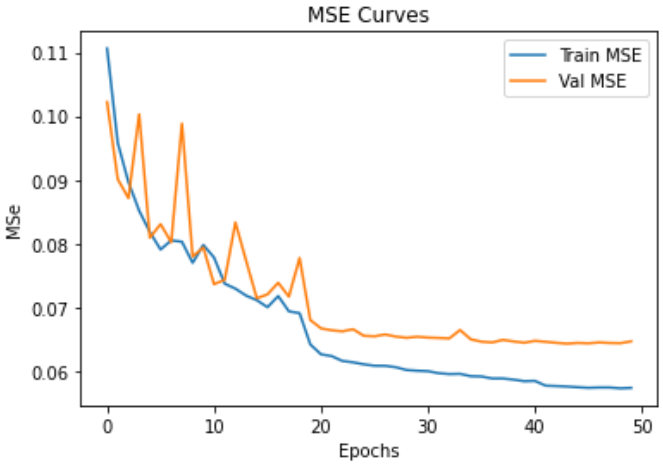


Slika 4.2. Graf stope učenja

Na slikama 4.3. i 4.4. prikazani su grafovi pogreške i MSE.



Slika 4.3. Graf pogreške



Slika 4.4. MSE graf

Može se iz dobivenih rezultata primjetiti kako je najbolji validation loss bio upravo oko 44. epohe sa svega 0.2153, nakon čega lagano počinje skakutati s rastom. To je obično znak pretreniranja mreže stoga se upravo na najnižoj validacijskoj pogreški preporuča prekid učenja.

Ovdje je prekid bio nešto nakon, ali se u obzir uzimao plato i reduciravala se stopa učenja pa se ne moramo bojati pretreniranosti. Na grafovima je lijepo vidljiv plato na dijelu gdje je greška za oko skoro konstantna. Graf MSE nam daje kvalilativno jednak izgled grafu pogreške.

4.2. Validacija

Najveseliji dio rada je provjera znanja istrenirane neuronske mreže. Slijedi kratak kod za predikciju segmentacije osoba na validacijskom skupu slika, na slici 4.5. Odmah na slici 4.6. su plotani rezultati. Uz prikaz ključne validacijske slike je i prvo slika predikcije s nijansama crnim, sivim i bijelim, koja točno prikazuje koliko je mreža bila sigurna na pojedinim pikselima da je tamo neka osoba, i gdje je djelomično sigurna. Na ovom validacijskom setu izabrana je zlatna sredina za *treshold* kojom se odbacuju sivi elementi, pretvarajući piksele iznad i jednako 0.5 u bijelo (1), te one manje od 0.5 u crno (0).

```
%time
inputs, targets = next(iter(val_generator))
preds = autoencoder.predict(inputs)
```

```
plt.rcParams["figure.figsize"] = (20, 5)

threshold = 0.5

for i, p, t in zip(inputs, preds, targets):
    op = p.copy()
    p[p < threshold] = 0
    p[p >= threshold] = 1

    plt.subplot(1, 4, 1)
    plt.imshow(i)
    plt.title("Input Image")
    plt.axis("off")
    plt.subplot(1, 4, 2)
    plt.imshow(op, cmap="gray")
    plt.title("Predicted Image")
    plt.axis("off")
    plt.subplot(1, 4, 3)
    plt.imshow(p, cmap="gray")
    plt.title("Predicted Mask (Thresholded)")
    plt.axis("off")
    plt.subplot(1, 4, 4)
    plt.imshow(t, cmap="gray")
    plt.title("True Mask")
    plt.axis("off")
    plt.show()
```

Slika 4.5. Validacija i plotanje rezultata s usporedbom

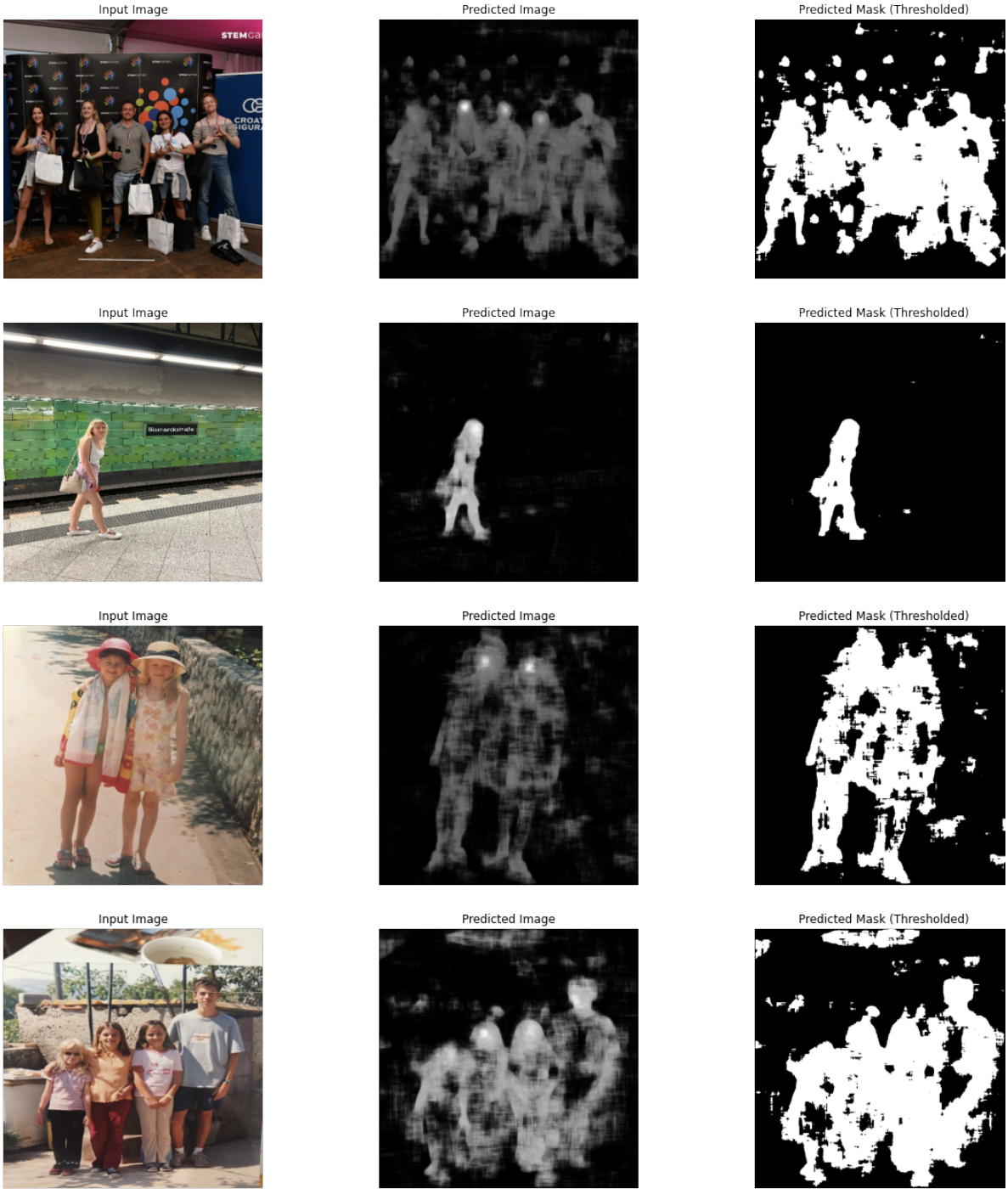
Skup rezultata plotan je jedan za drugim, a svaki s 4 slike u jednome redu. Treća slika predstavlja masku kakvu je mreža pretpostavila nakon zaokruživanja sivih piksela na obližnju

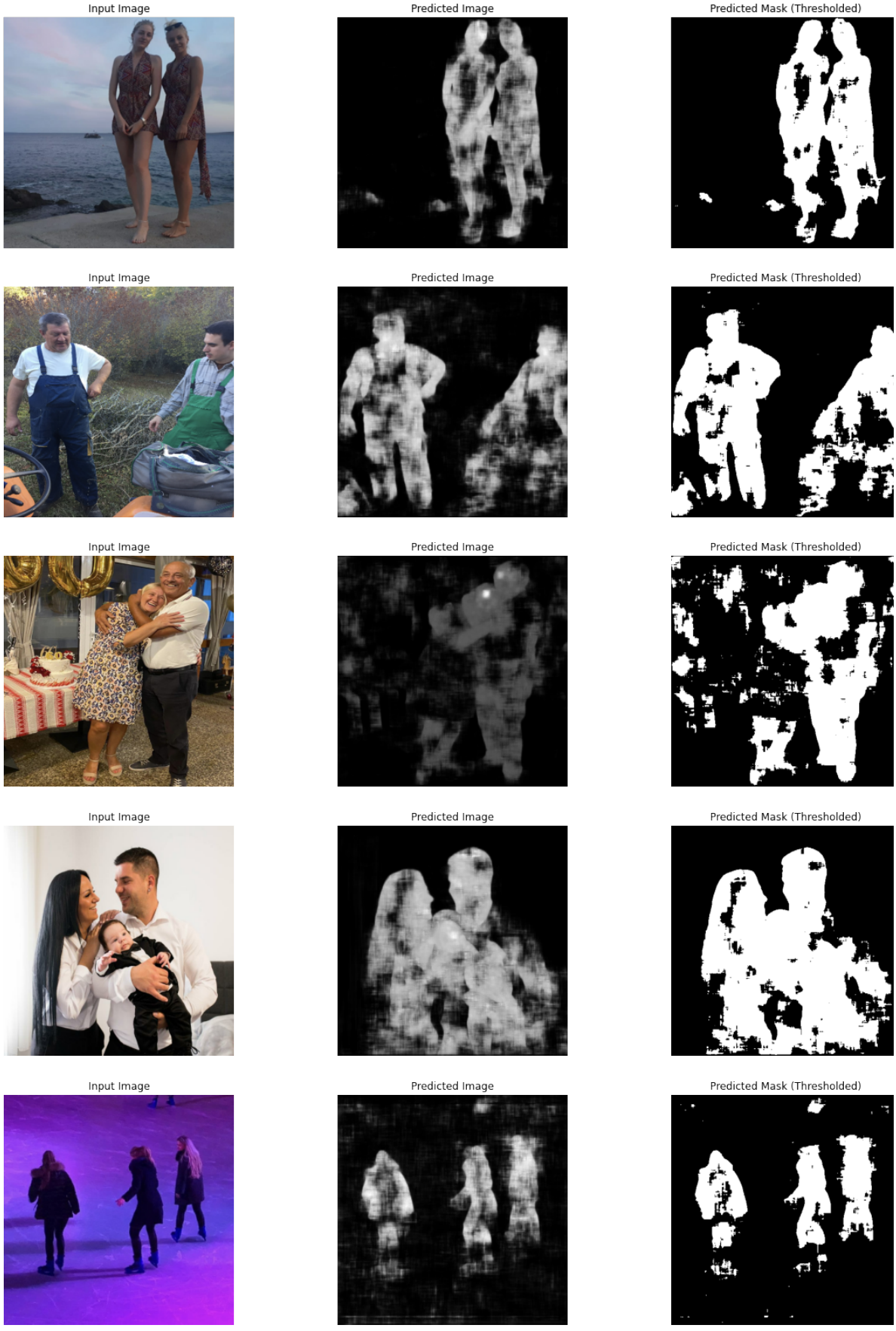
vrijednost nule ili jedinice. Posljednja slika u stupcu predstavlja masku dobivenu iz COCO anotacije kategorije osoba. Ta posljednja slika služi kao vizualni pregled točnosti kod odgovaranja.

Slika 4.6. Usporedba predikcije, ostvarene maske i točne maske

Na početnim predikcijama sa slike 4.6. vidljivi su zapravo izrazito dobri rezultati. Čak možda bolji na slici skijaša gdje je mreža raspoznala da skije ne spadaju pod kategoriju osoba. To je svakako nešto za uzeti u obzir prilikom dogovora i potrebe tijekom pripremanja maski za treniranje. Tko bi se složio drukčije, trebao bi i tako anotirati sliku. Može se primjetiti kako je mrežu zbunio kip sa slike te ga je segmentirala kao osobu, što bi bio dobar primjer bitne

pogreške. Mreža se pokazala poprilične točnosti i na grubim pikselima na slikama s mnoštvom ljudi. Nažalost, primjećene su prednje noge konja s poprilično sigurnošću kao dio osobe. Čini se kako je mreža dala veći značaj obliku nego li boji, obzirom da je sa sigurnošću odabrala kipa i prednje noge konja kao osoba, kao i leteći zmaj, vjerojatno kao asocijaciju na glavu i kosu, bez obzira na raznolikost boja. Na slici 4.7. slijede par osobno provučenih slika kroz model s rezultatima predikcije i *treshold* jednak 0.4.




Input Image
Predicted Image
Predicted Mask (Thresholded)
Input Image
Predicted Image
Predicted Mask (Thresholded)
Input Image
Predicted Image
Predicted Mask (Thresholded)
Input Image
Predicted Image
Predicted Mask (Thresholded)
Input Image
Predicted Image
Predicted Mask (Thresholded)

Slika 4.7. Rezultati predikcije na ne-COCO skupu podataka s tresholdom 0.4

Interesantno je primjetiti kako je mreža dala izniman značaj prepoznavanja kategorije *osoba* po očima, što se primjeti na slikama predikcije na čijim je pikselima svjetloća visoka. Kao i na slici 3.17. možemo ponovo primjetiti kako je mreža u obzir uzela i psa, što se i ovdje pokazalo kao dio kategorije osoba iako to nije. Ponovo se daje naglasak na bitnost odabira skupa za trening i dogovora u istome. Eksperiment se preporučuje ponoviti s filtriranijim trening setom podataka, odnosno bolje složenim anotacijama.

5. ZAKLJUČAK

Ovim radom je jednostavno prikazan primjer modeliranja konvolucijske neuronske mreže, te korist istog. Odabran je ovaj zadatak, zbog česte potrebe sličnog rješenja u pametnoj svakidašnjici. Najveća i najbitnija upotreba je definirana u sigurnosti, bilo u prometu bilo u privatnosti, poput FaceIDija (koji dodatno mjeri dubine lica, svakako sofisticiraniji). Rješenje je temeljeno na dubokom učenju uz računalni vid, a programirano pomoću Pythona. Riješavanje zadatka prethođeno je teorijskoj osnovi za razumijevanje principa umjetne inteligencije, razlike strojnog i dubokog učenja, arhitekture konvolucijske neuronske mreže te značajem računalnog vida. Uz upotrebnu Python knjižnica funkcija te COCO skupa podataka, model je uspješno istreniran i testiran. Određeno je da je točnost dobivenog modela kroz ovaj rad čak 77%. Moguće unaprijeđenje i zahtjevalo povećanje broja neurona, skrivenih slojeva i koraka učenja u svhru jačanja modela. Koncept izrade ovog modela je doduše bio napraviti ga što je više moguće jednostavnijim, a i dalje funkcionalnim, što je svakako uspješno završilo.
Bez promijene modela, mogao bi se rezultat poboljšati raznolikijim skupom podataka s dogovoreno točnijom anotacijom istih. Za kraj taj bi skup podataka mogao imati više slika ljudi raznolikijeg porijekla i iz različitijih pozicija, obzirom da se model pokazao lošijim u slučaju osobe okrenute bočno.

LITERATURA

[1] SINTEF, “Big Data, for better or worse: 90% of world's data generated over last two years.“, ScienceDaily, 22.05.2013. Pristupljeno: 20.06.2022. www.sciencedaily.com/releases/2013/05/130522085217.htm

[2] umjetna inteligencija. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2021. Pristupljeno 20.06.2022. www.enciklopedija.hr/Natuknica.aspx?ID=63150

[3] T. Stipančić: Podloge za predavanje iz kolegija „Umjetna inteligencija“, Fakultet strojarstva i brodogradnje, Zagreb, 2018.

[4] Iron Hack, "What is Machine Learning?". Pristupljeno 20.06.2022. www.ironhack.com/en/data-analytics/what-is-machine-learning

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

[6] Machine learning mystery, " What is Deep Learning?", J. Brownlee. Pristupljeno 20.06.2022. https://machinelearningmastery.com/what-is-deep-learning/

[7] Towards Data Science, " A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way", Sumit Saha. Pristupljeno 21.06.2022. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[8] Think Automation, " ELI5: what is a convolutional neural network?". Pristupljeno 21.06.2022. www.thinkautomation.com/eli5/eli5-what-is-a-convolutional-neural-network/

[9] SAS, " Computer Vision", " What it is and why it matters". Pristupljeno 15.07.2022. www.sas.com/en_us/insights/analytics/computer-vision.html

[10] Cybiant, " An Introduction to Computer Vision". Pristupljeno 15.07.2022. www.cybiant.com/resources/an-introduction-to-computer-vision/

[11] Python, "What is Python?". Pristupljeno 20.07.2022. https://docs.python.org/3/faq/general.html#what-is-python

[12] Jupiter Notebook, "The Jupyter Notebook", " Introduction". Pristupljeno 20.07.2022. https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

[13] edureka! " Mastering Python", " Top 10 Python Libraries You Must Know In 2022", Anirudh Rao, 22.07.2022. www.edureka.co/blog/python-libraries/

[14] OpenCV. Pristupljeno 23.07.2022. https://opencv.org/about/

[15] GitHub, "Matlabich/Semantic-Segmentation-using-AutoEncoders", forked from "animikhaich/Semantic-Segmentation-using-AutoEncoders". Pristupljeno 03.09.2022. https://github.com/Matlabich/Semantic-Segmentation-using-AutoEncoders?organization=Matlabich&organization=Matlabich

[16] Colaboratory. Pristupljeno 10.08.2022. https://colab.research.google.com/#scrollTo=lSrWNr3MuFUS

[17] COCO. Pristupljeno 05.06.2022. https://cocodataset.org/#explore

[18] Towards Data Science, " A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way", Sumit Saha. Pristupljeno 20.08.2022. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[19] TensorFlow, " tf . keras . optimizatori . Adam ". Pristupljeno 20.08.2022. www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

PRILOG

Dataset Preparation via Colab with Google Drive.ipynb

```
# Mount Google Drive to /content/gdive dir
from google.colab import drive
drive.mount('/content/gdrive')

# List the files in the My Drive dir
!ls "/content/gdrive/My Drive/"
```

```
!mkdir COCO

!wget http://images.cocodataset.org/zips/train2017.zip
!unzip train2017.zip
!rm train2017.zip

!wget http://images.cocodataset.org/zips/val2017.zip
!unzip val2017.zip
!rm val2017.zip

#!wget http://images.cocodataset.org/zips/test2017.zip
#!unzip test2017.zip
#!rm test2017.zip


!cd ../
!wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip
!unzip annotations_trainval2017.zip
!rm annotations_trainval2017.zip
```

```
from pycocotools.coco import COCO
import os
import cv2
import time
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
import random
```

```
# Annotation Files
train_ann = "/content/annotations/instances_train2017.json"
val_ann = "/content/annotations/instances_val2017.json"

train_img_dir = '/content/train2017/'
val_img_dir = '/content/val2017/'

train_dest_segmentation_masks = '/content/segmentations/train'
val_dest_segmentation_masks = '/content/segmentations/val'
```

```
# Initialize the COCO objects
train_coco = COCO(train_ann)
val_coco = COCO(val_ann)
```

```
loading annotations into memory...
Done (t=15.53s)
creating index...
index created!
loading annotations into memory...
Done (t=0.46s)
creating index...
index created!
```

```
def get_mask(coco_object, index, filtered_cat_id, filtered_img_id, img_dir):
    # Load Image
    img_meta = coco_object.loadImgs(filtered_img_id[index])
    img_path = os.path.join(img_dir, img_meta[0]['file_name'])
    img = cv2.imread(img_path)

    # Load Annotations
    annIds = coco_object.getAnnIds(imgIds=img_meta[0]['id'], catIds=filtered_cat_id, iscrowd=None)
    anns = coco_object.loadAnns(annIds)

    # Generate Mask
    anns_img = np.zeros((img.shape[:-1]), dtype=np.uint8)
    for ann in anns:
        anns_img = np.maximum(anns_img, coco_object.annToMask(ann)*ann['category_id'])

    return img_meta[0]['file_name'], anns_img * 255


def write_mask(coco_object, index, filtered_cat_id, filtered_img_id, dest_folder, img_dir):
    if not os.path.isdir(dest_folder):
        os.makedirs(dest_folder)

    filename, mask = get_mask(coco_object, index, filtered_cat_id, filtered_img_id, img_dir)
    dest_path = os.path.join(dest_folder, filename)
    cv2.imwrite(dest_path, mask)


def write_train_masks(train_coco, train_dest_folder, classes=['person']):
    print("Processing Train Images")
    time.sleep(0.5)
    filtered_cat_id = train_coco.getCatIds(classes)
    filtered_img_id = train_coco.getImgIds(catIds=filtered_cat_id)

    for i in tqdm(range(len(filtered_img_id))):
        write_mask(train_coco, i, filtered_cat_id, filtered_img_id, train_dest_folder, train_img_dir)

    print("Train Segmentation Masks Processing Complete")


def write_val_masks(val_coco, val_dest_folder, classes=['person']):
    print("Processing Val Images")
    time.sleep(0.5)
    filtered_cat_id = val_coco.getCatIds(classes)
    filtered_img_id = val_coco.getImgIds(catIds=filtered_cat_id)

    for i in tqdm(range(len(filtered_img_id))):
        write_mask(val_coco, i, filtered_cat_id, filtered_img_id, val_dest_folder, val_img_dir)

    print("Val Segmentation Masks Processing Complete")
```

```
write_train_masks(train_coco, train_dest_segmentation_masks)
write_val_masks(val_coco, val_dest_segmentation_masks)
```

```
Processing Train Images
100%|██████████| 64115/64115 [16:50<00:00, 63.45it/s]
Train Segmentation Masks Processing Complete
Processing Val Images
100%|██████████| 2693/2693 [00:34<00:00, 77.10it/s]
Val Segmentation Masks Processing Complete
```

```
# Save results from segmentation
!zip -r "/content/gdrive/My Drive/segmentations.zip" /content/segmentations
```

```
#!unzip "/content/gdrive/My Drive/segmentations.zip"
#!mv /content/content/segmentations/train /content/COCO/masks/
#!mv /content/content/segmentations/val /content/COCO/masks/
```

Model via Colaboratory with gDrive.ipynb

```
# Check the GPU stats
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)

from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
# Mount Google Drive to /content/gdive dir
from google.colab import drive
drive.mount('/content/gdrive')

# List the files in the My Drive dir
!ls "/content/gdrive/My Drive/"
```

```
# Magic word does not work for specific tensorflow version
# %tensorflow_version 2.4

# But this works
!pip install tensorflow-gpu==2.4.1
#!pip install matplotlib==3.3.4
!pip install numpy==1.19.5
!pip install opencv==4.5.1.48

# Hack to make colab working (Env restart required)
!pip uninstall -y matplotlib
!pip install matplotlib
```

```
# What version of Python do you have?
import sys

import tensorflow.keras
import pandas as pd
import sklearn as sk
import tensorflow as tf
import platform

print(f"Python Platform: {platform.platform()}")
print(f"Tensor Flow Version: {tf.__version__}")
print(f"Keras Version: {tensorflow.keras.__version__}")
print()
print(f"Python {sys.version}")
print(f"Pandas {pd.__version__}")
print(f"Scikit-Learn {sk.__version__}")
gpu = len(tf.config.list_physical_devices('GPU'))>0
print("GPU is", "available" if gpu else "NOT AVAILABLE")
print(tf.config.list_physical_devices('GPU'))
```

```
!mkdir COCO

!wget http://images.cocodataset.org/zips/train2017.zip
!unzip train2017.zip
!rm train2017.zip

!wget http://images.cocodataset.org/zips/val2017.zip
!unzip val2017.zip
!rm val2017.zip

#!wget http://images.cocodataset.org/zips/test2017.zip
#!unzip test2017.zip
#!rm test2017.zip

!wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip
!unzip annotations_trainval2017.zip
!rm annotations_trainval2017.zip
```

```
# Unzip Masks
!unzip /content/gdrive/MyDrive/segmentations.zip
```

```
# Create folder structure
!mkdir /content/COCO
!mkdir /content/COCO/images
!mkdir /content/COCO/masks
!mkdir /content/COCO/images/train
!mkdir /content/COCO/images/val
!mkdir /content/COCO/masks/train
!mkdir /content/COCO/masks/val
```

```
# Move data for training
!mv /content/content/segmentations/train /content/COCO/masks/
!mv /content/content/segmentations/val /content/COCO/masks/

# Rename folders
!mv /content/train2017 /content/train
!mv /content/val2017 /content/val
#!mv /content/test2017 /content/test

# Move folders to COCO dir
!mv /content/train /content/COCO/images
!mv /content/val /content/COCO/images
```

```
!ls -U /content/COCO/images/train | wc -l
!ls -U /content/COCO/images/val | wc -l
!ls -U /content/COCO/masks/train | wc -l
!ls -U /content/COCO/masks/val | wc -l
```

```
# Imports
import os
os.environ["TF_FORCE_GPU_ALLOW_GROWTH"] = "true"
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2
from glob import glob
from tqdm import tqdm
```

```
train_mask_dir = "/content/COCO/masks/train/"
val_mask_dir = "/content/COCO/masks/val/"

train_images = glob(os.path.join(train_mask_dir, "*"))
val_images = glob(os.path.join(val_mask_dir, "*"))
```

```
img_path = train_images[0]
img = cv2.imread(img_path)
print("Dims: ", img.shape)
plt.imshow(img[:,:,::-1])
plt.show()
```

```
# A quick Tensorflow Data loader
class DataLoader:
    """
    Assuming Dataset Path Structure:
    - Train Images: /media/ActiveTraining/Datasets/COCO/images/train
    - Val Images: /media/ActiveTraining/Datasets/COCO/images/val
    - Train Masks: /media/ActiveTraining/Datasets/COCO/masks/train
    - Val Masks: /media/ActiveTraining/Datasets/COCO/masks/val
    """
    def __init__(self, masks_dir, target_shape=(512, 512)):
        masks_dir = self.__fix_glob(masks_dir)
        self.files = tf.data.Dataset.list_files(masks_dir, shuffle=True)
        self.AUTOTUNE = tf.data.experimental.AUTOTUNE
        self.target_shape = target_shape

    def __fix_glob(self, dir_path):
        if not dir_path.endswith("*"):
            dir_path = os.path.join(dir_path, "*")

        return dir_path

    @tf.function
    def parse_images(self, mask_path):
        mask = tf.io.read_file(mask_path)
        mask = tf.io.decode_jpeg(mask, channels=1)
        mask = tf.cast(mask, tf.float32)

        image_path = tf.strings.regex_replace(mask_path, "masks", "images")
        image = tf.io.read_file(image_path)
        image = tf.io.decode_jpeg(image, channels=3)
        image = tf.cast(image, tf.float32)

        mask = tf.image.resize(
            mask,
            self.target_shape,
        ) / 255.

        image = tf.image.resize(
            image,
            self.target_shape,
        ) / 255.

        return image, mask

    @tf.function
    def data_generator(self, batch_size=64):
        dataset = self.files.map(self.parse_images)
        dataset = dataset.repeat()
        dataset = dataset.batch(batch_size)
        dataset = dataset.prefetch(buffer_size = self.AUTOTUNE)
        return dataset
```

```
test_dataset = DataLoader(train_mask_dir)
batch = test_dataset.data_generator(10)
inputs, targets = next(iter(batch))
print(f"Input Shape: {inputs.shape}, Target Shape: {targets.shape}")

for i, t in zip(inputs, targets):
    plt.subplot(1, 2, 1)
    plt.imshow(i)
    plt.axis("off")
    plt.subplot(1, 2, 2)
    plt.imshow(t, cmap="gray")
    plt.axis("off")
    plt.show()
```

```
# Callbacks
early_stop_cb = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=10,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=True,
)
model_ckpt_cb = tf.keras.callbacks.ModelCheckpoint(
    "weights.h5",
    monitor="val_loss",
    verbose=0,
    save_best_only=True,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
)
reduce_lr_cb = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.1,
    patience=4,
    verbose=1,
    mode="auto",
    min_delta=0.0001,
    cooldown=0,
    min_lr=10e-8,
)
```

```
# Model Architecture

input_img = tf.keras.Input(shape=(None, None, 3))

# encoding architecture
x1 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(input_img)
x2 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(x1)
x3 = tf.keras.layers.MaxPool2D(padding='same')(x2)
x4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x3)
x5 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x4)
x6 = tf.keras.layers.MaxPool2D(padding='same')(x5)
encoded = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x6)

# decoding architecture
x7 = tf.keras.layers.UpSampling2D()(encoded)
x8 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x7)
x9 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x8)
x10 = tf.keras.layers.Add()([x5, x9])
x11 = tf.keras.layers.UpSampling2D()(x10)
x12 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(x11)
x13 = tf.keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same')(x12)
x14 = tf.keras.layers.Add()([x2, x13])
decoded = tf.keras.layers.Conv2D(1, (3, 3), padding='same',activation='relu')(x14)
autoencoder = tf.keras.Model(input_img, decoded)
autoencoder.summary()
```

```
# Hyper Params
batch_size = 32
epochs = 50 # 50
lr = 0.001
target_shape=(224, 224)

opt = tf.keras.optimizers.Adam(learning_rate=lr)

autoencoder.compile(
    loss="binary_crossentropy",
    optimizer=opt,
    metrics=[
        "mse",
])

# Dataset Generator
train_dataset = DataLoader(train_mask_dir, target_shape=target_shape)
train_generator = train_dataset.data_generator(batch_size)

val_dataset = DataLoader(val_mask_dir, target_shape=target_shape)
val_generator = val_dataset.data_generator(batch_size)

# Calculate Num Steps
train_steps = len(list(train_dataset.files)) // batch_size + 1
val_steps = len(list(val_dataset.files)) // batch_size + 1
```

```
# Train
history = autoencoder.fit(
    train_generator,
    validation_data=val_generator,
    batch_size=batch_size,
    steps_per_epoch=train_steps,
    validation_steps=val_steps,
    epochs=epochs,
    callbacks=[
        early_stop_cb,
        model_ckpt_cb,
        reduce_lr_cb
    ])
```

```
# Matplotlib Plots (For Easy Rendering)

# Learning Rate
plt.plot(history.history['lr'])
plt.title('Learning Rate')
plt.xlabel("Epochs")
plt.ylabel("LR Value")
plt.show()

# Loss Curves
plt.plot(history.history['loss'], label="Train Loss")
plt.plot(history.history['val_loss'], label="Val Loss")
plt.title('Loss Curves')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

# MSE Curves
plt.plot(history.history['mse'], label="Train MSE")
plt.plot(history.history['val_mse'], label="Val MSE")
plt.title('MSE Curves')
plt.xlabel("Epochs")
plt.ylabel("MSe")
plt.legend()
plt.show()
```

```
%time
inputs, targets = next(iter(val_generator))
preds = autoencoder.predict(inputs)
```

```
plt.rcParams["figure.figsize"] = (20, 5)

threshold = 0.5

for i, p, t in zip(inputs, preds, targets):
    op = p.copy()
    p[p < threshold] = 0
    p[p >= threshold] = 1

    plt.subplot(1, 4, 1)
    plt.imshow(i)
    plt.title("Input Image")
    plt.axis("off")
    plt.subplot(1, 4, 2)
    plt.imshow(op, cmap="gray")
    plt.title("Predicted Image")
    plt.axis("off")
    plt.subplot(1, 4, 3)
    plt.imshow(p, cmap="gray")
    plt.title("Predicted Mask (Thresholded)")
    plt.axis("off")
    plt.subplot(1, 4, 4)
    plt.imshow(t, cmap="gray")
    plt.title("True Mask")
    plt.axis("off")
    plt.show()
```

```
!cp /content/weights.h5 /content/gdrive/My\ Drive/50epocha-weights.h5
```

```
#rm -rf /content/weights.h5
```

Quick Start via Colab with saved weights on gDrive.ipynb

```
# Magic word does not work for specific tensorflow version
# %tensorflow_version 2.4

# But this works
!pip install tensorflow-gpu==2.4.1
#!pip install matplotlib==3.5.3
!pip install numpy==1.19.5
!pip install opencv==4.5.1.48

# Hack to make colab working (Env restart required)
!pip uninstall -y matplotlib
!pip install matplotlib
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow-gpu==2.4.1
  Downloading tensorflow_gpu-2.4.1-cp37-cp37m-manylinux2010_x86_64.whl (394.3 MB)
     |████████████                    | 156.7 MB 3.3 MB/s eta 0:01:12
```

```
# Mount Google Drive to /content/gdrive dir
from google.colab import drive
drive.mount('/content/gdrive')

# List the files in the My Drive dir
!ls "/content/gdrive/My Drive/"
```

```
import os
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

autoencoder = tf.keras.models.load_model('/content/gdrive/My Drive/50eWeights.h5')
#autoencoder.summary()
```

```
# A quick Tensorflow Data loader
class MyInputsDataLoader:
    def __init__(self, inputs_dir, target_shape=(512, 512)):
        inputs_dir = self.__fix_glob(inputs_dir)
        self.files = tf.data.Dataset.list_files(inputs_dir, shuffle=True)
        self.AUTOTUNE = tf.data.experimental.AUTOTUNE
        self.target_shape = target_shape

    def __fix_glob(self, dir_path):
        if not dir_path.endswith("*"):
            dir_path = os.path.join(dir_path, "*")

        return dir_path

    @tf.function
    def parse_images(self, image_path):
        image = tf.io.read_file(image_path)
        image = tf.io.decode_jpeg(image, channels=3)
        image = tf.cast(image, tf.float32)

        image = tf.image.resize(
            image,
            self.target_shape,
        ) / 255.

        return image
```

```
    @tf.function
    def data_generator(self, batch_size=10):
        dataset = self.files.map(self.parse_images)
        dataset = dataset.repeat()
        dataset = dataset.batch(batch_size)
        dataset = dataset.prefetch(buffer_size = self.AUTOTUNE)
        return dataset

input_dataset = MyInputsDataLoader('/content/gdrive/My Drive/inputs/*')
batch = input_dataset.data_generator(20)
inputs = next(iter(batch))
print(f"Input Shape: {inputs.shape}")
```

```
preds = autoencoder.predict(inputs)

plt.rcParams["figure.figsize"] = (20, 5)
threshold = 0.5

for i, p in zip(inputs, preds):
    op = p.copy()
    p[p < threshold] = 0
    p[p >= threshold] = 1

    plt.subplot(1, 3, 1)
    plt.imshow(i)
    plt.title("Input Image")
    plt.axis("off")
    plt.subplot(1, 3, 2)
    plt.imshow(op, cmap="gray")
    plt.title("Predicted Image")
    plt.axis("off")
    plt.subplot(1, 3, 3)
    plt.imshow(p, cmap="gray")
    plt.title("Predicted Mask (Thresholded)")
    plt.axis("off")
    plt.show()
```