Umjetna inteligencija u medicini: Integracija tehnika dubokog učenja u robotske kirurške sustave

Elez, Mate

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

Mate Elez

UMJETNA INTELIGENCIJA U MEDICINI: INTEGRACIJA TEHNIKA DUBOKOG UČENJA U ROBOTSKE KIRURŠKE SUSTAVE

ZAVRŠNI RAD

Split, lipanj 2024

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

Umjetna inteligencija u medicini: Integracija tehnika dubokog učenja u robotske kirurške sustave

Mate Elez

SAŽETAK

Cilj ovog rada je istražiti i opisati primjenu tehnika dubokog učenja u medicini. Rad se fokusira na objašnjavanje načina na koji duboko učenje može poboljšati prepoznavanje i segmentaciju bolesti, te integraciju toga u robotske kirurške sustave Kroz analizu različitih metoda dubokog učenja, kao što su konvolucijske neuronske mreže, rad istražuje kako ove tehnologije mogu doprinijeti preciznijem i efikasnijem liječenju pacijenata.

Ključne riječi: robotski kriruški sustavi, duboko učenje, konvolucijske neuronske mreže, binarna klasifikacija, segmentacija slike

Rad sadrži: 61 stranicu, 35 slika i 11 literaturnih navoda.

Mentor: **Dr. sc. Saša Mladenović,** *redoviti profesor, Prirodoslovno-matematičkog fakulteta, Sveučilište u Splitu*

Ocjenjivači: **Dr. sc. Saša Mladenović,** *redoviti profesor, Prirodoslovno-matematičkog fakulteta, Sveučilište u Splitu*

Antonela Prnjak, asistent Prirodoslovno-matematičkog fakulteta u Splitu, Sveučilišta u Splitu

Nika Jerković, asistent Prirodoslovno-matematičkog fakulteta u Splitu, Sveučilišta u Splitu

Rad prihvaćen: **rujan, 2024**

Basic documentation card

Thesis

University of Split

Faculty of science

Department of Informatics

Ruđera Boškovića 33, 21000 Split, Croatia

Artificial intelligence in medicine: Integration of deep learning techniques into robotic surgical systems

Mate Elez

SUMMARY

The aim of this paper is to explore and describe the application of deep learning techniques in medicine. The paper focuses on explaining how deep learning can enhance disease recognition and segmentation, and the integration of this into robotic surgical systems. Through the analysis of various deep learning methods, such as convolutional neural networks, the paper investigates how these technologies can contribute to more precise and efficient patient treatment.

Keywords: robotic surgical systems, deep learning, convolutional neural networks, binary classification, image segmentation

Thesis consists of: 61 pages, 35 figures and 11 references.

Mentor: **Saša Mladenović, Ph.D.** Full Professor, *full professor at the Faculty of Science and Mathematics in Split, University of Split*

Reviewers: **Saša Mladenović, Ph.D.** Full Professor, *full professor at the Faculty of Science and Mathematics in Split, University of Split*

Antonela Prnjak, Instructor of Faculty of Science, University of Split

Nika Jerković, Instructor of Faculty of Science, University of Split

Thesis accepted: **September, 2024**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam završni rad s naslovom „UMJETNA INTELIGENCIJA U MEDICINI: INTEGRACIJA TEHNIKA DUBOKOG UČENJA U ROBOTSKE KIRURŠKE SUSTAVE“ izradio samostalno pod voditeljstvom prof. dr. sc. Saša Mladenovića.. U radu sam koristio literaturu koja je navedena na kraju završnog rada. Sve tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam citirao ili parafrazirao iz druge literature jasno sam označio citatima s navedenim izvorom iz kojeg su prenesene.

Rad je pisan u duhu hrvatskog jezika.

Student:

Mate Elez

Zahvaljujem se profesoru Saši Mladenoviću na pruženoj prilici za mentorstvo.

Sadržaj

Uvod ... 1

1. Primjena inteligentnih robota u kirurgiji ... 3

1.1. Povijest robotske kirurgije ... 3

1.2. Tehnologija robotske kirurgije ... 5

1.3. Postupak robotske kirurgije ... 6

1.4. Prednosti ... 7

1.5. Nedostatci ... 8

1.6. Potencijalni napredak robotske kirurgije ... 9

2. Primjena umjetne inteligencije u medicini ... 10

2.1. Uvod u primjenu umjetne inteligencije u medicini ... 10

2.2. Umjetna inteligencija i dijagnostika bolesti ... 12

2.2.1. Strojno učenje u dijagnosticiranju bolesti ... 12

2.2.2. Duboko učenje u dijagnosticiranju bolesti ... 16

2.2.3. Izazovi i ograničenja u dijagnosticiranju bolesti pomoću umjetne inteligencije ... 22

2.3. Etičnost korištenja umjetne inteligencije u medicini ... 23

2.3.1. Privatnost i zaštita podataka ... 23

2.3.2. Informirani pristanak i autonomija ... 24

2.3.3. Nedostatak emocija i empatije ... 24

3. Binarna klasifikacija tumora mozga uz VGG16 ... 25

3.1. Učitavanje i prikaz podataka ... 25

3.2. Pred-procesiranje podataka ... 28

3.3. Podjela podataka ... 33

3.4. Arhitektura modela ... 34

3.5. Uvježbavanje modela ... 39

3.6. Predikcija i vrednovanje modela ... 40

4. Segmentacija tumora mozga ... 42

4.1. Učitavanje i prikaz podataka ... 42

4.2. Pred-procesiranje podataka 46
4.3. Podjela podataka 48
4.4. Arhitektura modela 49
4.5. Uvježbavanje modela 54
4.6. Vrednovanje modela 54
5. Integracija 57
Zaključak 58
Literatura 59
Popis slika 60

Uvod

U današnje vrijeme umjetna inteligencija je jedna od najproširenijih i najaktualnijih grana računalne znanosti. Umjetnu inteligenciju možemo naći svugdje, od preporučitelja na raznim platformama, asistenata na mobilnim uređajima pa sve do njene primjene u jako bitnim branšama za ljude kao što je medicina.

Zdravstveni sektor je danas podložan jako brzim i značajnim promjenama zahvaljujući brzom razvoju tehnologije, a posebno umjetne inteligencije koja danas postaje sve više i više prisutna u svakodnevnoj medicinskoj praksi.

Jedno od područja gdje se sve više i više primjenjuje umjetna inteligencija je detekcija bolesti. Jedna bolest koja s kojom bi trebali biti upoznati da bi nastavili s ovim radom je tumor mozga. Tumor mozga je vrlo komplicirano medicinsko stanje koje, kao što je i većina upoznata, može imati jako loš ishod za pacijenta. Simptomi tumora mozga su često zamijenjeni s nekim općim simptomima viroza, glavobolja i raznih drugih lakših bolesti. Naravno, to je ono što liječnicima ne ide u prilog jer ako se ne tretira dovoljno rano, kasnije je sve teže i teže spasiti pacijenta od takve bolesti. Upravo zbog toga danas se koriste MR i CT slike mozga kako bi doktori prepoznali je li unutar mozga pacijenta prisutan tumor. Kao što i sami znamo doktori su ljudi koji vjerojatno imaju jedan od najtežih i najiscrpnijih poslova od svih te je to, uz to što ljudi nisu nepogrešivi, razlog zašto je jako lako doći do pogreške pri pregledavanju gore navedenih slika. Naravno, nekad je i nemoguće ljudskim okom otkriti neki mali tumor koji je zapravo sadržan na slici, no jednostavno je previše mal i dobro uklopljen u okolinu. Tu na snagu dolazi umjetna inteligencija koja je u sposobna uzeti sliku mozga i kao izlaz dati doktorima sugestiju je li tumor prisutan. Na taj način bi doktori bili puno sigurniji u svoje odluke te bi promislili dvaput u slučaju krivog dijagnosticiranja pacijenta koji možda na prvi pogled nema navedenu bolest.

Naravno, bitno je spomenuti da tu ne mora biti kraj umjetnoj inteligenciji u ovom području. Naime, roboti u procesima operacija su sve aktualnija i aktualnija tema koja datira još iz kraja 20. stoljeća te se pokazala vrlo uspješnom i sigurnom upravo iz razloga što je precizna i većina operacija koje se provedu s robotima zahtijevaju manji vremenski period oporavka i manji rizik štete nad pacijentom kojeg operiramo.

Naravno, da bi roboti operirali na ovakav način nije dovoljna sama informacija je li tumor prisutan ili nije. Potrebno je i nekako lokalizirati i segmentirati tumor da bi robot mogao obaviti operaciju.

Kao što ste već mogli zaključiti iz svega gore opisanog, u ovom radu ćemo se baviti robotskim kirurškim sustavima, binarnom klasifikacijom i segmentacijom tumora mozga te ćemo se naposlijetku dotaknuti integracije.

1. Primjena inteligentnih robota u kirurgiji

Primjena inteligentnih robota u kirurgiji predstavlja revolucionaran napredak u medicinskoj tehnologiji. Mi ćemo se u ovom poglavlju osvrnuti na nekoliko bitnih cjelina, a to su:

- Povijest robotske kirurgije
- Tehnologija robotske kirurgije
- Postupak robotske kirurgije
- Prednosti
- Nedostatci
- Potencijalni napredak robotske kirurgije

1.1. Povijest robotske kirurgije

Robotska kirurgija predstavlja jedan od najznačajnijih napredaka u medicini. Njena povijest seže u prošlost sve od 1960-ih godina kada je robotika u kirurgiji bila iznimno popularna tema upravo zbog tada vođenih ratova i iscrpljenih vojnih kirurga koji nisu bili u stanju raditi dan za danom tim intezitetom.

Unatoč početnom entuzijazmu, razvoj robotske kirurgije suočio se s brojnim izazovima tijekom godina iz raznih razloga te su prvi dokazi o obavljenoj robotskoj operaciji još iz 1980-ih godina. Tada je prvi robot za operacije pod imenom PUMA 560 korišten u proceduri biopsije mozga te je ta operacija obavljena 1985 godine, dok je tri godine poslije robot PROBOT već korišten kako bi napravio nekolicinu „rezova“ na pacijentu tijekom operacije.

Početkom 1990-ih godina IBM u suradnji s nekolicinom ostalih tvrtki razvio je ROBODOC-a te su ga u 1992. godini uspješno koristili u proceduri zamjene dijela kuka kod ljudske osobe. Krajem 1990-ih razvijena su još tri robotska sistema za operiranje koja su bila sposobna vršiti laparoskopsku kirurgiju s kirurškim robotima. Ti sustavi su bili da Vinci, AESOP i Zeus.

2000-te godine obilježila je telekirurgija. Naime, u tom vremenskom razdoblju se odvila operacija uklanjanja žučnog mjehura nad pacijentom koji se nalazio u Strasbourgu u Francuskoj, dok su se kirurzi koji su obavljali operaciju nalazili u New York-u. Ova operacija je tada bila uspješna, no velika udaljenost je uzrokovala vremensko kašnjenje od doktora do pacijenta od 155 milisekundi, a idealno bi bilo 100 milisekundi te nam ovo odstupanje, pogotovo u području medicine, nije prihvatljivo.

Od početka 2010-ih godina do danas da Vinci je postao najčešće korišten robotski kirurški uređaj diljem svijeta. Ima mnogo razloga zašto je on jedan od najboljih u koje nećemo sada ulaziti, no bitno je napomenuti da je robotski sustav Zeus prekinut 2003. godine te da je njegov tvorac već tada dao prednost razvoju da Vinci sustava.

1.2. Tehnologija robotske kirurgije

Tehnologija robotske kirurgije predstavlja moderan medicinski napredak i način na koji se kirurški zahvati izvode i pružaju kirurzima veći kontrolu, preciznost i sigurnost. Ova tehnologija se zasniva na integraciji inžinjerskih principa s medicinskim potrebama i računalnim sustavima kako bih se omogućilo izvođenje kirurškim zahvata uz pomoć robota. U nastavku ovog poglavlja ćemo se osvrnuti na nekoliko ključnih koncepata koji čine ovu tehnologiju, a to su arhitektura i upravljanje robotima.

Robotski kirurški sistemi kao i svaki drugi sistem se sastoji od nekoliko ključnih komponenti. Mi ćemo te komponente svesti na tri osnove, to jest na robotsku konzolu, robotske ruke i kirurške alate te ćemo se pozabaviti samo njima.

Robotska konzola u ovom kontekstu nije ništa više nego centralna jedinica koja omogućuje kirurgu da upravlja robotom tijekom operacije te da u isto vrijeme nadgleda tu operaciju. Ona se najčešće sastoji od visokokvalitetnog zaslona sa jako velikom rezolucijom kojem je zadaća prikazivanje detaljne slike unutar tijela pacijenta. Na konzoli još postoji i ručni upravljač ili u nekim drugim literaturama „manipulator“ koji kirurgu služi za kontroliranje pokreta robotskih ruku tijekom operacije. Također postoje i pedale koje kirurgu omogućuju neke funckionalnosti poput zumiranja ili rotiranja kamere te ih kirurg u principu koristi kako bi prilagodio vidno polje tijekom operacije.

Robotske ruke su ključna komponenta u kirurškim robotskim sustavima te one omogućuju vrlo precizno upravljanje kirurškim instrumentima. Neke od njihovih najvećih prednosti su to što imaju veliku slobodu pokreta i integraciju s kirurškim instrumentima. Naime, robotske ruke obično imaju više stupnjeva slobode pokreta od ljudskih ruku te im to omogućuje da izvode složenije manipulacije unutar ljudskog tijela. Osim toga to također dopušta prilaz nekim teško dostupnim područjima tijela kojima ljudske ruke možda ne bi mogle doći niti blizu. Važno je napomenuti da su robotske ruke opremljene držačima i stezaljkama koji omogućuju pričvršćivanje mnogih kirurških instrumenata te to znači da se tijekom operacije može koristiti mnoštvo različitih kirurških alata.

Slika 1 Prikaz robotske konzole i robotskih ruku (Hughes i sur., (2023), The Availability, Cost, Limitations, Learning Curve and Future of Robotic Systems in Urology and Prostate Cancer Surgery)

Kirurški instrumenti su alati koji omogućuju izvođenje raznih vrsta kirurških zahvata. Neki od najčešće korištenih kirurških alata su skalpeli, pincete i endoskopske kamere. Naravno postoji još mnoštvo drugih, no da sad ne nabrajamo možemo zaključiti da su i oni neophodni za ovaj sustav, te da bez njih ništa ne bi imalo smisla.

1.3. Postupak robotske kirurgije

Kao i kod obične kirurgije tako i kod robotske mora postojati i postoji neki uobičajeni postupak kojim se treba voditi od početka do kraja kako bi operacija bila uspješna.

Početak ovog postupka je naravno priprema samog pacijenta. Naime, pacijent se priprema standardno, kao i kad su u pitanju obične operacije. To uključuje provođenje nekakvih testova prije operacije i primjenu anestezije.

Korak nakon što je pacijent spreman je samo postavljanje robota. Naime, cijeli robotski sistem se stavlja kraj operacijskog stola, te to uključuje namještanje robotskih ruka i instrumenata za operiranje da budu u optimalnog poziciji za operaciju.

Kada su i pacijent i robot spremni, na red dolazi samo izvođenje operacije. Koristeći ekran ispred sebe, robotske ruke i kirurške instrumente kirurg izvodi precizne zahvate nad pacijentom dok mu kamera, koja se nalazi u jednoj od robotskih ruka, omogućuje jasan uvid u pacijentovo tijelo.

Nakon provođenja i završetka operacije kirurg mora ukloniti sve instrumente i robotske ruke iz tijela pacijenta i provjeriti jesu li nastale nekakve komplikacije tijekom operacije. Pacijent se nakon toga prenosi na skrb poslije operacije gdje se nadalje prati njegovo zdravstveno stanje.

1.4. Prednosti

Ako u gore napisanom tekstu već nismo mogli uočiti brojne dobre faktore ovakvog načina operiranja mislim da je vrijeme da sada navedemo i definiramo neke od najbitnijih prednosti.

Prednost koja je možda i najveća od svih je sama preciznost. Naime, ovako povećana preciznost kao što je u robotskoj kirurgiji rezultira manjim oštećenjima tkiva i organa, te je ovo vrlo korisno kada se operacije izvršavaju na osjetljivim dijelovima tijela.

Manji bolovi i manje traume pacijenata su isto jako bitne prednosti, iz razloga što se tako smanjuje pacijentova potreba za raznim lijekovima nakon operacije i što se ubrzava proces ozdravljenja.

Rizik od komplikacija kad je riječ o robotskim operacijama se također znatno smanjuje. Pod tim rizicima se smatraju infekcije, krvarenje i oštećenje okolnih organa. Ovo je vrlo važna prednost koja uvijek predstavlja ogromni problem u klasičnim operacijama.

1.5. Nedostatci

Kako i kod svega, pa tako i kod robotske kirurgije čim spomenemo prednosti moramo spomenuti i neke nedostatke.

Jedan od, a možda i najveći nedostatak su vrlo visoki troškovi. Naime, robotski sustavi su skupi, kako za kupnju tako i za održavanje i samu operaciju.

Složenost postupka je također jedan od nedostataka koji je vrijedan spomena. Složenost postupka u smislu operacije može često rezultirati produženim vremenom operacije i rizikom od grešaka što nikako ne bi htjeli.

Koliko god neka tehnologija pouzdana i dobro napravljena na kraju dana moramo biti svjesni da je to samo tehnologija te da je kvar na njoj itekako moguć. Tako i u slučaju robotske kirurgije kvarovi i problemi se mogu dogoditi i negativno utjecati na ishod operacije.

Unatoč svim gore navedenim nedostatcima i dalje mnogi kirurzi i pacijenti koji pristaju na ovaj tip operacija uspijevaju prepoznati vrijednost robotskih operacija i smatraju da prednost istih neupitno nadilaze nedostatke.

1.6. Potencijalni napredak robotske kirurgije

Iako postoji jako puno već napravljenih stvari koje dobro i kvalitetno rade, kod robotskih kirurških sustava još uvijek ima mjesta za napredak.

Mi ćemo se u ovom radu fokusirati na jednu granu računalne znanosti koja je nedavno postala prisutna u životima nezanemarivom broju pojedinaca te je sve češća i češća u mnogim granama znanosti, poslovima i zanimanjima. Ta grana je umjetna inteligencija. Naime, umjetnu inteligenciju je moguće koristiti svugdje pa tako i u ovom području.

Ovim želim istaknuti da umjetna inteligencija može imati jako važnu ulogu u robotskim kirurgijama, pa ćemo se zbog toga u idućem poglavlju bazirati prvo na njenu primjenu u medicini, pa ćemo nakon toga vidjeti kako to sve možemo smjestiti u kontekst robotske kirurgije s ciljem njenog napretka.

2. Primjena umjetne inteligencije u medicini

Kao što je rečeno u prethodnom poglavlju, umjetna inteligencija je jako primjenjiva u mnogim područjima, pa tako i u medicini. Mi ćemo se u ovom dijelu rada osvrnuti na nekoliko bitnih stavki bez kojih ne možemo nastaviti s temom ovog rada.

To su:

- Uvod u primjenu umjetne inteligencije u medicini
- Umjetna inteligencija i dijagnostika bolesti
- Etičnost korištenja umjetne inteligencije u medicini

2.1. Uvod u primjenu umjetne inteligencije u medicini

Da bi krenuli s ovim poglavljem treba nam prvo biti jasno što je umjetna inteligencija. Naime, to je izraz koji se koristi za opisivanje upotrebe tehnologije i računala za simulaciju ponašanja i kritičkog razmišljanja koje je usporedivo s ljudskom osobom. Osim što se koristi za simulaciju ponašanja i kritičkog razmišljanja ljudi, umjetna inteligencija ima potencijal u potpunosti promijeniti način na koji se dijagnosticiraju, prate i lijeće razne bolesti. Suvremene tehnologije omogućavaju računalima da prihvate i analiziraju jako velike količine podataka i to na brži i precizniji način nego što bi to mogao ljudski um te se upravo time otvara puno mogućnosti za napredak u medicini.

Sada kada to znamo, možemo prijeći na primjenu umjetne inteligencije u medicini. Naime, umjetna inteligencija u medicini se većinom klasificira u dvije podvrste: virtualnu i fizičku. Što se virtualne primjene umjetne inteligencije u medicini tiče, postoje različite aplikacije, a za primjer možemo uzeti sustave elektroničkih zdravstvenih zapisa koji omogućuju bolju organizaciju i pristup podacima, isto tako postoje generirane smjernice za liječenje temeljenje na raznim metodama koje pomažu u donošenju odluka o terapiji i dijagnostici.

S druge strane, fizički dio se odnosi na upotrebu robota koji pomažu pri izvođenju operacija, inteligentne proteze koje omogućuju pokretljivost i funkcionalnost, te tehnologija za brigu o starijim osobama kao što su razni senzori za praćenje vitalnog stanja i autonomnih sustava pojedinaca.

Umjetna inteligencija u medicini predstavlja spoj najnovijih tehnološkim otkrića i znanja same medicine. te to rezultira stvaranjem novih sustava i alata koji bi mogli pomoći medicinskom osoblju u pružanju bolje skrbi pacijentima. Također je važno naglasiti da je uspjeh primjene umjetne inteligencije u medicini ovisan o mnogo faktora, a najviše o kvaliteti i količini dostupnih podataka. Velike baze podataka informacija o pacijentima, medicinskih zapisa, nekakvih slika i mnogih drugih podataka ključne su za učenje bilo kakvog modela umjetne inteligencije, te osiguravaju njegovu pouzdanost. Također je potrebno pažljivo razmotriti etičku stranu umjetne inteligencije u medicini, koja je jako kontroverzna u zadnje vrijeme, no to ćemo ostaviti tek za kraj ovog poglavlja rada.

2.2. Umjetna inteligencija i dijagnostika bolesti

U ovom poglavlju ćemo se malo više posvetiti dijagnostici bolesti uz pomoć umjetne inteligencije iz razloga što će praktični dio ovog rada dijelom biti klasifikacija tumora mozga.

Cjeline kojima ćemo se posvetiti kroz ovaj dio su:

- Strojno učenje u dijagnosticiranju bolesti
- Duboko učenje u dijagnosticiranju bolesti
- Izazovi i ograničenja u dijagnosticiranju bolesti pomoću umjetne inteligencije

2.2.1. Strojno učenje u dijagnosticiranju bolesti

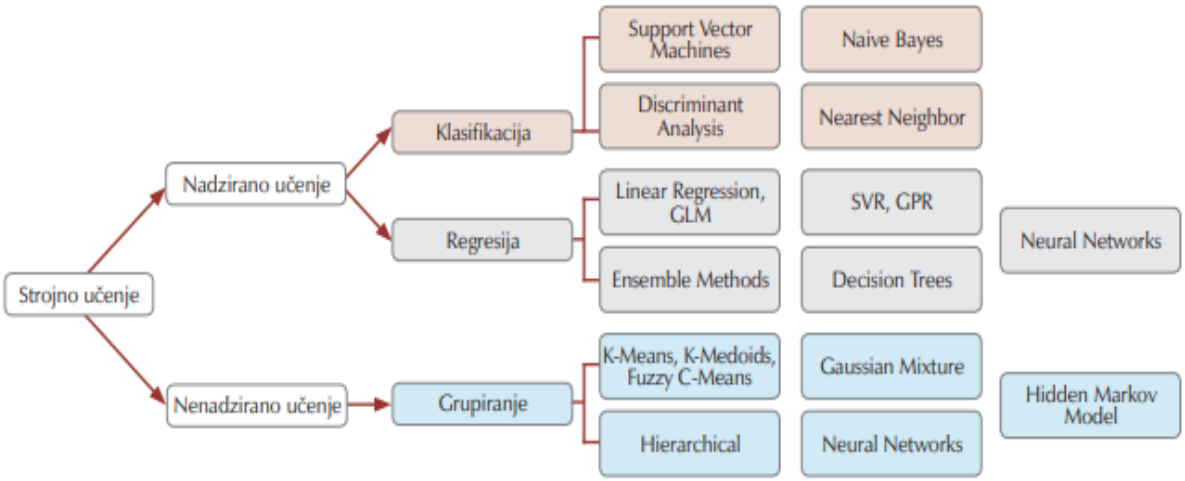
Strojno učenje aktualna je tema u istraživanju i raznim industrijama, te se nove metode iz ovog područja računalne znanosti sve češće pojavljuju i razvijaju. Algoritmi strojnog učenja „uče“ informacije i njihove međusobne odnose izravno iz podataka. Kroz taj proces učenja modeli se s vremenom unaprjeđuju. Kroz ovo poglavlje ćemo se upoznati sa strojnim učenjem i osnovnim pojmovima vezanim uz njega.

Zadatak algoritama strojnog učenja je zapravo pronalaženje uzoraka i poveznica u podacima te na temelju njih, stjecanje uvida i analiziranje, pa tek onda odlučivanje i predviđanje. Naravno nisu svi algoritmi strojnog učenja isti i zbog toga imamo nekakvu osnovnu podjelu na algoritme nadziranog učenja i algoritme nenadziranog učenja.

Kod nadziranog učenja algoritmi rade s poznatim ulaznim podacima i poznatim izlaznim podacima, te s njima uvježbavaju model za predviđanje. Naravno da nadzirano učenje ima svoje prednosti i nedostatke, no pokazalo se da je ono jako primjenjivo u postupcima klasifikacije i regresije. Klasifikacija je kao što ime govori razvrstavanje neke diskretne jedinice u grupu kojoj pripada. Za primjer možemo uzeti klasifikaciju nekog voća kao naranču ili jabuku na temelju njegovih karakteristika. S druge strane, regresija je predviđanje kontinuiranih vrijednosti, tu za primjer možemo uzeti predviđanje promjene temperature, cijene automobila, nekretnina i mnogo drugih stvari sličnih tome.

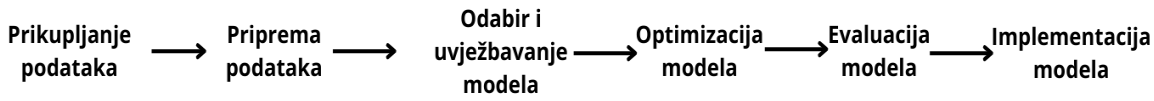
Što se nenadziranog učenja tiče, ono pronalazi uzorke ili nekakve strukture u podacima. Njime se grupiraju uzorci i otkrivaju strukture podataka. Specifičnost kod ovih algoritama je ta što se primjenjuju na onim skupovima podataka kod kojih ne znamo vrijednost izlaznih podataka. Najčešća tehnika ovog tipa strojnog učenja je grupiranje koja, kao što i samo ime govori, grupira podatke u unaprijed zadani broj grupa. Ovaj tip algoritama strojnog učenja primjenu pronalazi u analizi sekvenci gena, prepoznavanju raznih objekata i raznim drugim instancama.

Slika podjele na algoritama strojnog učenja prikazana je ispod.



Slika 2 Podjela algoritama strojnog učenja (Bolf, N. (2021) Strojno Učenje)

Kao i kod kuhanja, spremanja, običnog učenja postoji neki proces po kojim se svi vode da bi ostvarili cilj. Modeli strojnog učenja nisu ništa drukčiji od tih običnih aktivnosti te njihov „recept“ možemo razbiti na nekolicinu koraka koji su prikazani na slici ispod.



Slika 3 Koraci pri izradi modela strojnog učenja

Sada kada imamo korake, proći ćemo ih po redu od prvog do posljednjeg.

Naime, prvi korak je samo prikupljanje podataka, pod tim korakom se smatra skupljanje podataka nekog tipa koji će nam koristiti za model. Važno je naznačiti da, ukoliko želimo da nam model bude dobar, tih podataka ne bi smjelo biti malo i trebali bi biti kvalitetni.

Drugi korak je priprema podataka za model, ovaj korak se odnosi na to da ovisno o tipu algoritma kojeg ćemo koristiti podaci trebaju biti podijeljeni u skupove koji su skupovi za uvježbavanje i testiranje modela, s tim da bi skup za uvježbavanje trebao biti dosta veći od onog za testiranje, također ako se radi o nadziranom algoritmu podatci trebaju biti podijeljeni u ulazne i izlazne podatke. Postoji još par stvari kao što je standardiziranje podataka, to je također važan korak koji se radi s ciljem da svi podaci imaju sličan raspon vrijednosti.

Odabir i uvježbavanje modela je treći korak u ovom postupku. Naime, odabiranje modela možda je i najbitniji dio, no s tim se uvijek može eksperimentirati i gledati koji model će

nam dati najbolje rezultate. Dok je uvježbavanje modela zapravo prolazak modela kroz dio podataka za uvježbavanje u svrhu da nauči neke obrasce i značajke. Cilj ovoga je minimizirati grešku modela i poboljšati mu sposobnost generalizacije na novim podatcima.

Nakon odabira modela i uvježbavanja istog slijedi optimizacija modela. Optimizacija modela se radi u svrhu da se postigne što bolja performansa i učinkovitost modela. Postoji mnogo strategija i tehnika optimizacije, no jedna od ključnih je hiperparametarska optimizacija. Ona podrazumijeva podešavanje hiperparametara modela s kojima ćemo se tek poslije upoznati.

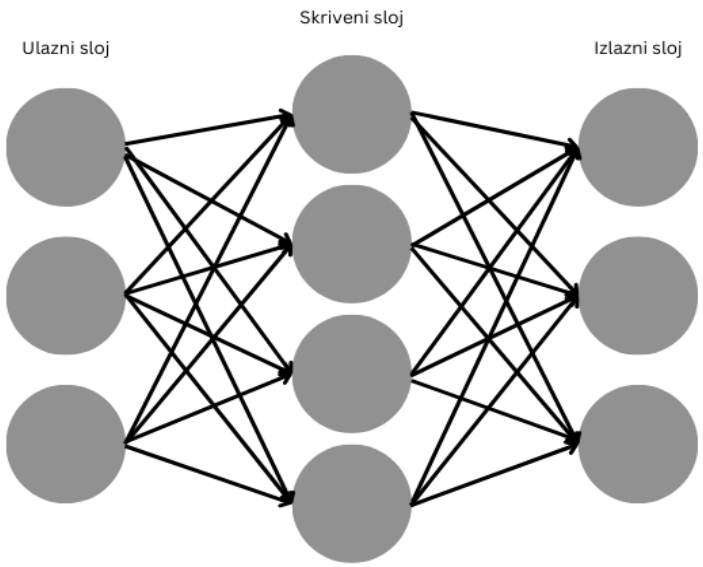
Vrednovanje modela je predzadnji korak koji omogućava procjenu njegove učinkovitost, točnosti i sposobnosti generalizacije na novim podatcima. Za vrednovanje modela koristi se jako puno metrika u koje sada nećemo ulaziti, već poslije, no sada ćemo ih nabrojati. To su: točnost, preciznost, odaziv i F1 mjera. Svim ovim ćemo se pozabaviti u praktičnom dijelu rada.

Implementacija modela je zadnji korak u ovom procesu i ona predstavlja uvođenje modela u produkcijsko okruženje kako bi se omogućila njegova stvarna primjena u stvarnom svijetu.

Upotreba strojnog učenja kod dijagnostike bolesti je vrlo široka. Iz razloga što praktični dio ovog rada ne spada direktno u domenu strojnog učenja, mi se sada nećemo zamarati sa prolaskom nekog primjera korak po korak, već ćemo samo nabrojati nekolicinu primjera primjene. Naime, strojno učenje se može koristiti u dijagnosticiranju dijabetesa, raka, bolesti srca, neuroloških poremećaja i u još puno toga. Iz gore navedenog možemo zaključiti da je primjena strojnog učenja u ovom području stvarno vrlo raznovrsna i sveobuhvatna te se, s obzirom na stalni napredak u ovom području, očekuje se da će se njena primjena nastaviti još više širiti.

2.2.2. Duboko učenje u dijagnosticiranju bolesti

Duboko učenje je podskup strojnog učenja koje koristi višeslojne neuronske mreže za simulaciju ljudskog mozga u donošenju odluka. To u principu znači da duboko učenje omogućuje računalima da nauče složene obrasce iz velikih skupova podataka koji ne moraju uvijek biti strukturirani, već mogu biti i nestrukturirani kao što su slike i zvukovi. Razlike strojnog i dubokog učenja su te što duboko učenje eliminira dio procesiranja podataka prije razvoja modela. Također, neke od bitnih razlika su i same arhitekture modela i primjena, no toga ćemo se dotaknuti poslije. Da bi započeli s pričom o dubokom učenju trebamo definirati osnovnu stvar na kojem se svaki algoritam dubokog učenje zasniva, a to su neuronske mreže. Neuronske mreže su temeljna arhitektura dubokog učenja koja simulira funkcioniranje ljudskog mozga. Sastoji se od slojeva neurona koji primaju ulazne podatke, obrađuju ih i šalju ih idućem sloju. Neuron u mreži je osnovna jedinica koja prima ulazne podatke, obrađuje ih i prosljeđuje. Svaki pojedinačni neuron ima svoje težine koje se prilagođavaju za vrijeme učenja te se na taj način minimizira greška između stvarnih i predviđenih vrijednosti. Što se arhitekture neuronske mreže tiče, ona se sastoji od više slojeva neurona, što uključuje ulazni sloj, skrivene slojeve i izlazni sloj. Ulazni sloj prima podatke, skriveni slojevi ih obrađuju, a izlazni sloj daje krajnje predikcije.



Slika 4 Prikaz arhitekture neuronske mreže

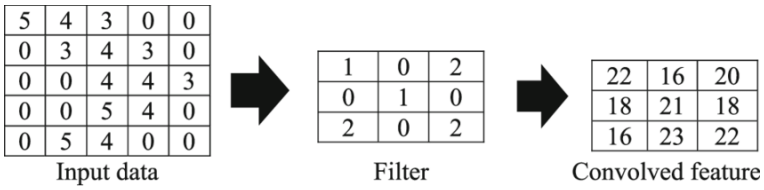
Aktivacijske funkcije su također važan dio neuronskih mreža jer se primjenjuju na izlazne vrijednosti neurona kako bi se omogućilo neuronskoj mreži da nauči složene funkcije. Najčešće korištene aktivacijske funkcije su ReLU, sigmoidna, tanh i softmax funkcija.

Nakon objašnjenja osnovnih pojmova neuronskih mreža potrebno je naglasiti da se u dubokom učenju ne koriste samo one, već i mnoge druge, od kojih su neke: generativne kontradiktorne mreže (GAN), ponavljajuće neuronske mreže (RNN), mreže radijalne osnovne funkcije (RBFN) i konvolucijske neuronske mreže (CNN). Iako je svaka od ovih instanci neuronskih mreža jako zanimljiva detaljno ćemo obraditi samo konvolucijske neuronske mreže iz razloga što ćemo se s tim tipom sresti u praktičnom dijelu ovog rada.

Konvolucijske neuronske mreže su neuronske mreže posebno prilagođene za obradu podataka poput slika i videa. One postižu zaista impresivne rezultate kada je u pitanju prepoznavanje objekata, segmentacija slika i detekcija uzoraka. U ovom poglavlju istražit ćemo arhitekturu konvolucijskih neuronskih mreža, njihove vrste i razne primjene.

Slično kao i obične neuronske mreže, konvolucijske neuronske mreže imaju tri glavna sloja, a to su konvolucijski sloj, sloj sažimanja i potpuno povezani sloj.

Konvolucijski sloj je temeljni građevni blok konvolucijske neuronske mreže te se na njemu odvijaju izračuni. On sadrži 3 komponente: ulazne podatke (na slici 5 „Input data“), filtere (na slici 5 „Filter“) i mapu značajki (na slici 5 „Convolved feature“). To možemo zamisliti da kao ulaz imamo sliku koja se sastoji od matrice piksela. Nadalje, možemo zaključiti da ćemo kao ulaz imati tri dimenzije, to jest visinu, širinu i dubinu. Filter, koji možemo zamisliti kao malu matricu, kretat će se kroz polja slika i provjeravati je li određena značajka prisutna. Ovaj proces zove se konvolucija.



Slika 5 Prikaz konvolucijskog sloja (Montesinos-López i sur., (2022). Convolutional Neural Networks. 10.1007/978-3-030-89010-0_13)

Filteri su 2D nizovi koji predstavljaju dio slike. Iako njihova veličina može varirati, oni su najčešće 3x3 matrice. Filter se u procesu konvolucije primjenjuje na područje slike te se produkt između ulaznih piksela i filtera zapisuje u novi niz. Nakon toga se filter pomiče korak po korak dok ne obiđe cijelu sliku. Izlaz ovog procesa je poznat pod imenom mapa značajki. Tri hiperparametra koji utječu na veličinu izlaza su: broj filtera, korak, punjenje nulama.

Broj filtera utječe na dubinu izlaza, odnosno ako koristimo tri različita filtera dobit ćemo tri različite mape značajki, što znači da će dubina izlaza biti tri.

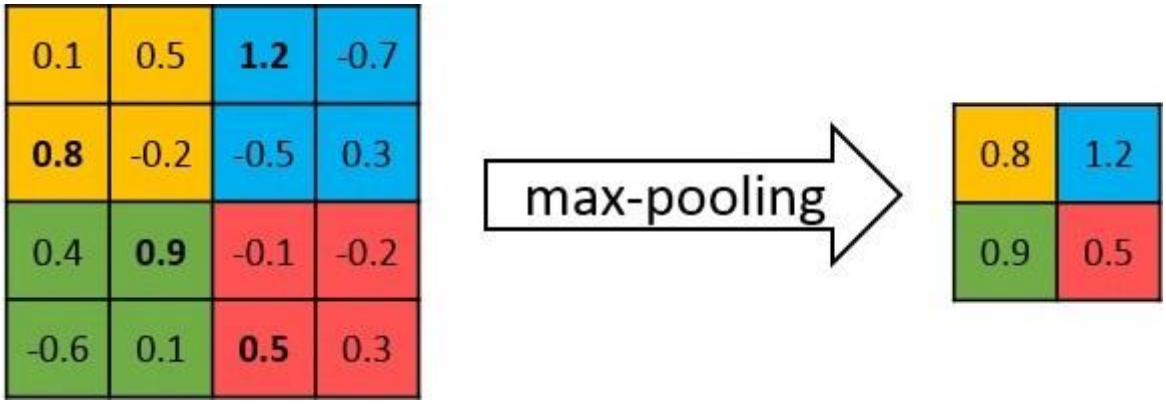
Korak je udaljenost za koju se filter pomakne po ulaznoj matrici tijekom konvolucije. Manji korak rezultira većem izlazu, dok veći rezultira manjem izlazu.

Punjenje nulama se koristi kada filter ne odgovara ulaznoj matrici, te ta metoda postavlja sve elemente izvan granice ulazne matrice na nulu.

Slojevi sažimanja poznati su po tome što smanjuju dimenzionalnosti ulaza i broj parametara u mreži. Rezultat ovog sloja smanjuje količinu podataka koje treba obraditi i na taj način uvelike olakšava izračune. Slično konvolucijskom sloju, ovo se provodi pomicanjem filtera preko cijelog ulaza, ali za razliku od konvolucije, ovaj filter nema težine već koristi „max-pooling“ ili „average-pooling“ funkciju nad vrijednostima nad kojima se nalazi.

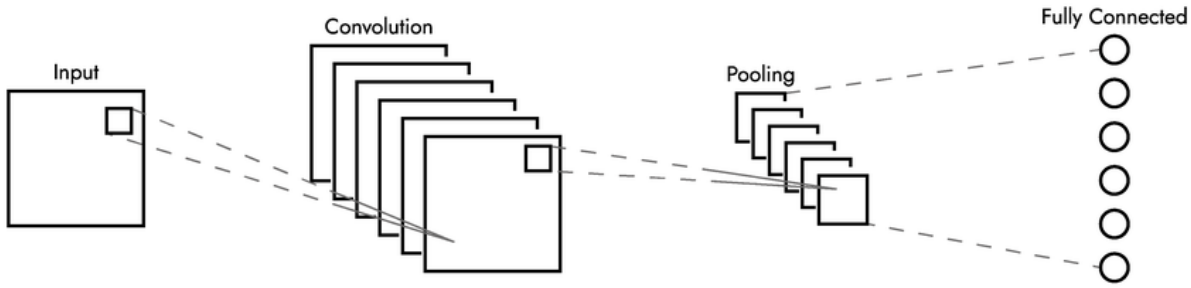
„Max-pooling“ funkcija uzima parametar sa maksimalnom vrijednošću iz polja koje trenutno filter prekrivam, dok „average-pooling“ uzima prosječnu vrijednost. Iako sloj sažimanja rezultira gubitkom informacija, on ima mnoštvo prednosti. Naime, on smanjuje dimenzionalnost ulaznih podataka, poboljšava učinkovitost izračuna i smanjuje rizik od prenaučenosti modela.

Na slici ispod možemo vidjeti grafički prikaz procesa sažimanja uz pomoć „max-pooling“ funkcije.



Slika 6 Primjer procesa sažimanja (David i sur. (2016). DeepPainter: Painter Classification Using Deep Convolutional Autoencoders. 20-28. 10.1007/978-3-319-44781-0_3)

Potpuno povezani sloj, kako i samo ime govori, je sloj koji je potpuno povezan sa svakim čvorom prethodnog sloja. Zadatak ovog sloja je klasifikacija na temelju značajki koje su dobivene kroz prethodne slojeve. Također je važno istaknuti da je ovaj sloj najčešće završni sloj u konvlucijskim neuronskim. Zaključno, potpuno povezani slojevi čine ključnu ulogu u procesu donošenja odluka jer spajaju informacije iz prethodnih slojeva i daju konačne predikcije.



Slika 7 Prikaz konačne arhitekture konvolucijske neuronske mreže (Antunes i sur, (2022), Benchmarking Deep Learning Methods for Behaviour-Based Network Intrusion Detection)

Naravno, postoji jako puno vrsta konvolucijskih neuronskih mreža, a mi ćemo se u ovom poglavlju usredotočiti na samo dvije koje ćemo nadalje i koristiti.

VGG je arhitektura koja je jedna od prvih konvolucijskih mreža koja je postigla značajne rezultate kod zadataka klasifikacije slika. Neke od karakteristika VGG-a su njezina jednostavnost, jasna organizacija slojeva i efikasnost u ekstrakciji složenih značajki iz slika. Naime, VGG arhitektura se sastoji od niza konvolucijskih slojeva koji su praćeni nizom slojeva sažimanja i potpuno povezanim slojevima na kraju. Što se primjene ove arhitekture tiče, ona je najčešća u raznim zadacima obrade slika što uključuje i medicinsku dijagnostiku. U prvom dijelu praktičnog dijela ovog rada koristit ćemo instancu VGG-a, VGG16, u rješavanju problema binarne klasifikacije tumora mozga. Razlog zašto koristimo ovu arhitekturu je taj što postoje razni, već gotovi, primjeri primjene ove arhitekture na sličnim problemima koji su dali izrazito dobre rezultate, što nam pruža dokazanu i osiguranu učinkovitost. Osim toga arhitektura ove mreže je vrlo jednostavna i intuitivna za shvatiti, što mi je omogućilo jednostavnu implementaciju i interpretaciju same arhitekture mreže.

U-Net je arhitektura koja je namijenjena za segmentaciju slika. Karakteristike ove mreže su te da se sastoji od nekolicine tehnika i dijelova, koji su: „encoder“, „decoder“, „connecting paths“, „bottleneck“ i „up-sampling“. Sada nećemo ulaziti detaljno u svaki dio iz razloga što ćemo ovu arhitekturu kasnije koristiti u praktičnom dijelu ovog rada te ćemo je tad detaljno opisati. Razlozi zašto ćemo koristiti ovu arhitekturu su slični kao i kod VGG16. Naime, ova arhitektura je već dokazano pouzdana i efikasna na mnogim primjerima segmentacije slika. Osim toga, implementacija i interpretacije njene arhitekture je donekle jednostavna.

Kao i sve, tako i konvolucijske neuronske mreže imaju neke svoje prednosti i nedostatke.

Što se prednosti tiče tu možemo izdvojiti to što su one jako dobre u samoj ekstrakciji značajki sa slika te im to omogućava da detektiraju karakteristike koje su bitne kad je riječ o klasifikaciji ili segmentaciji. Imunost na translaciju je također jako bitna prednost zbog toga što su sposobne prepoznati obrasce unatoč tome što će se položaj značajki na nekoj slici možda promijeniti. Dijeljenje parametara je isto tako prednost koja znači da se jedan filter može koristiti na različitim dijelovima slike, te se tako može smanjiti broj parametara u samo konvolucijskoj mreži, pa se tim olakšava samo uvježbavanje modela. Kao možda i najveću prednost još možemo izdvojiti i same dobre rezultate koje konvolucijske neuronske mreže pokazuju na raznim zadacima.

S druge strane, možemo reći da je potreba za velikim skupovima podataka jedan od nedostataka konvolucijskih neuronskih mreža. Naime, konvolucijske neuronske mreže često zahtijevaju velike skupove podataka za uvježbavanje kako bi naposlijetku postigle dobre rezultate. Te podatke može biti izrazito teško prikupiti jer se mora uzeti u obzir da oni moraju biti kvalitetni i da ih mora biti puno. Na to se možemo nadovezati s tim da su konvolucijske neuronske mreže izrazito osjetljive i na raznolikost podataka za uvježbavanje, te i to nekad također može biti problem. Složenost arhitekture ovih mreža isto možemo navesti kao nedostatak iz razloga što ove mreže mogu biti izrazito složene, te to uvelike otežava samu interpretaciju rada mreže i analizu odluka koje je ona donijela.

2.2.3. Izazovi i ograničenja u dijagnosticiranju bolesti pomoću umjetne inteligencije

Izazovi i svakakva ograničenja su vrlo česta kada je u pitanju umjetna inteligencije, a kada to spojimo s drugom zahtjevnom znanosti kao što je medicina to može biti još samo gore.

Prvi izazov koji će vjerojatno svakome pasti na pamet je sami nedostatak kvalitetnih podataka. Ovaj problem je vrlo izražen, pogotovo kada je riječ o medicini, jer je izrazito teško naći veliku količinu podataka vezanu za neku bolest, a kamoli kvalitetne podatke tog tipa, te nas to dovodi do toga da smo odmah u početku spriječeni razvijati model jer naravno nema smisla pustiti nešto „polu dobro“ u rad u grani znanosti koja se bavi liječenjem ljudi.

Tu se možemo nadovezati na drugo ograničenje, koje je pouzdanost modela za rad. Naime, da bi se sami model pustio u upotrebu u medicinskoj praksi on mora biti savršen. Također, svaki model koji se misli pustiti u upotrebi mora biti maksimalno razumljiv te svaka njegova odluka mora biti potkrepljena dokazom i razlogom zašto je to tako kako bi se steklo povjerenje zdravstvenih ustanova i samih pacijenata.

Kao treći izazov možemo uzeti integraciju u medicinsku praksu iz razloga što bi se uvođenjem ovoga u medicinsku praksu svi radni protokoli i infrastruktura bolnica i sličnih ustanova morala promijeniti. Radnici u tim ustanovama bi se isto tako morali obučavati da bi uspješno radili po novim protokolima.

Etičnost korištenja umjetne inteligencije u medicini je također jako velik problem, no tim ćemo se pozabaviti u idućem poglavlju.

2.3. Etičnost korištenja umjetne inteligencije u medicini

Nadovezujući se na prethodno poglavlje ovdje ćemo raspraviti o nekoliko točaka koje se tiču etičnosti korištenja umjetne inteligencije u medicini.

Točke su:

- Privatnost i zaštita podataka
- Informirani pristanak i autonomija
- Nedostatak emocija i empatije

2.3.1. Privatnost i zaštita podataka

Što se privatnosti i zaštite podataka tiče, tu moramo biti svjesni da sustavi umjetne inteligencije prikupljaju i analiziraju podatke svakog pacijenta koji je podvrgnut liječenju uz pomoć tog sustava. Nadalje, intuitivno možemo zaključiti da ti podaci mogu biti ukradeni i iskorišteni u zlonamjerne svrhe. To nas dovodi do toga da su uvedeni vrlo rigorozni zakoni, poput opće uredbe o zaštiti podatka, vezani za zaštitu podataka pacijenata koje medicina bez umjetne inteligencije teško prati, a s njom je izrazito teže. Upravo zbog ovog razloga jako je važno osigurati snažne sigurnosne mjere i zaštitu u svim fazama implementacije sustava umjetne inteligencije u medicinsku praksu. To bi uključivalo sigurnu pohranu podataka, enkripciju podataka, kontrolu pristupa i redovito refaktoriranje software-a zbog mogućih propusta. Naposlijetku možemo zaključiti da su sve gore navedene metode o sigurnosti neophodne da bi se poštivale uredbe o zaštiti podataka.

2.3.2. Informirani pristanak i autonomija

Što se informiranog pristanka i autonomije tiče, tu se fokusiramo na proces komunikacije između pacijenta i pružatelja zdravstvene skrbi. To uključuje sposobnost odlučivanja, kompetenciju dokumentiranja informiranog priznanja i etičko zaključivanje. Kada pričamo o etičkoj odgovornosti, svaki pacijent ima pravo biti obaviješten o svojim dijagnozama, zdravstvenom stanju, procesu liječenja, terapijskom uspjehu i drugim medicinskim informacijama. Sve ovo gore navedeno naglašava važnost etičkog pristupa u komunikaciji između pacijenta i medicinskog radnika, no kada je u pitanju umjetna inteligencija jasno nam je da ona nema tu mogućnost komuniciranja, te tu dolazimo do mnogih dilema oko njenog korištenja.

2.3.3. Nedostatak emocija i empatije

Treća točka se odnosi na to da roboti i umjetne inteligencija nemaju jedinstvene ljudske emocije koje pomažu u svakodnevnoj komunikaciji s pacijentima pružajući im nadu i suosjećanje. Također je jako bitno naglasiti da liječnici vrlo često traže konzultacije međusobno jedni od drugih, što je nemoguće kada se radi o sustavima umjetne inteligencije. Sve ovo nas dovodi do toga da će vrlo vjerojatno malo ljudi prihvatiti umjetnu inteligenciju u zdravstvenu skrb upravo zbog toga što je od liječnika i medicinskih sestara očekivano da se brinu o pacijentu na empatičan i suosjećajan način. Ovo je jedan od najznačajnijih negativnih aspekata korištenja umjetne inteligencije u medicini koji je vrlo vjerojatno nemoguće riješiti.

3. Binarna klasifikacija tumora mozga uz VGG16

Sada dolazimo do poglavlja kroz koje ćemo se osvrnuti na prvi dio praktičnog dijela ovog rada, a to je binarna klasifikacija tumora mozga uz konvolucijsku neuronsku mrežu VGG16.

Okruženje izabrano rad na ovom praktičnom dijelu, a i na onome koje slijedi nakon njega, je google colaboratory. Google colaboratory je izabran iz razloga što je izuzetno jednostavan za upotrebu i zbog toga što se može integrirati sa google drive-om. Google colaboratory također pruža besplatne resurse što se tiče CPU-a i GPU-a, te samim tim omogućuje rad na zahtjevnim zadacima bez potrebe za skupim hardverom računala na kojem se radi. To pruža značajnu prednost jer korisnici ne moraju investirati u skupu opremu.

3.1. Učitavanje i prikaz podataka

Prvi korak pri izradi ovog projekta je učitavanje podataka, no također je bitno spomenuti da smo prije toga uvezli sve biblioteke koje su nam potrebne, te da smo se spojili na google drive jer nam se tamo nalaze podaci.

```
image_files = [os.path.join('/content/drive/MyDrive/Tumor', f) for f in os.listdir('/content/drive/MyDrive/Tumor') if os.path.isfile(os.path.join('/content/drive/MyDrive/Tumor', f))]
Tumor = []
for image_file in image_files:
    image = cv2.imread(image_file)
    if image is not None:
        Tumor.append(image)
    else:
        print(f"Failed to read image: {image_file}")
print(len(Tumor))
```

```
2000
```

Slika 8 Prikaz učitavanja podataka sa google drive-a

Na slici poviše prikazano je učitavanje slika u nizove za one slike s tumorom. Ovaj postupak je isti i za slike bez tumora, pa će biti objašnjen samo ovaj prikazan na slici.

Prva linija koda na slici poviše radi listu putanja do svih datoteka, to jest slika tumora mozga unutar direktorija „/content/drive/MyDrive/Tumor“. Znači da lista „image_files“, nakon što se ovo izvrši, sadrži apsolutne putanje svih datoteka koje se nalaze u napisanom direktoriju.

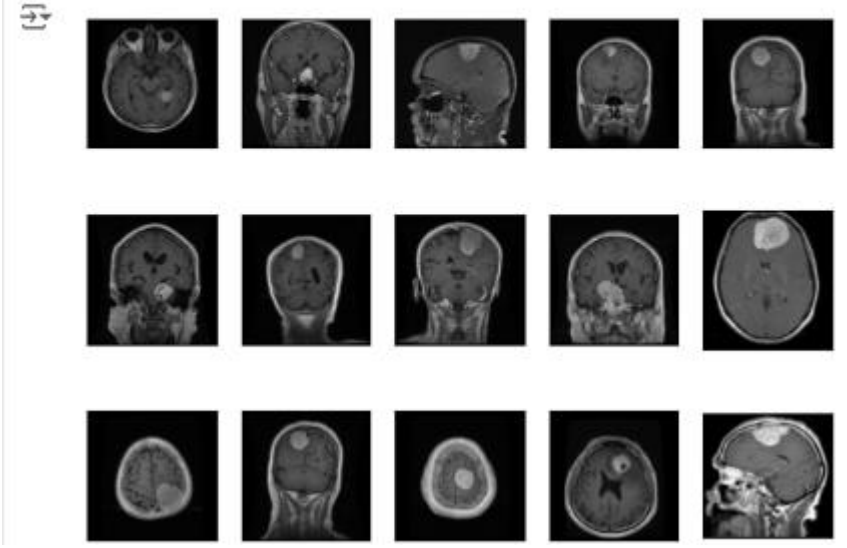
Nakon ovoga kreira se prazna lista Tumor koju ćemo puniti preko for petlje.

For petlja prolazi kroz svaku datoteku u listi „image_files“ te se za svaku od tih datoteka poziva funkcija „cv2.imread()“ da učita sliku u varijablu image. Nakon toga slijedi provjera za varijablu image s jednostavnim uvjetom samo da budemo sigurni da je unutar nje slika. Ako provjera pokaže da je u varijabli slika varijabla se stavlja u niz „Tumor“.

Nakon što smo završili sa punjenjem niza „printamo“ njegovu veličinu da se uvjerimo da su slike stvarno unutar njega te možemo vidjeti da u nizu „Tumor“ imamo dvije tisuće njih.

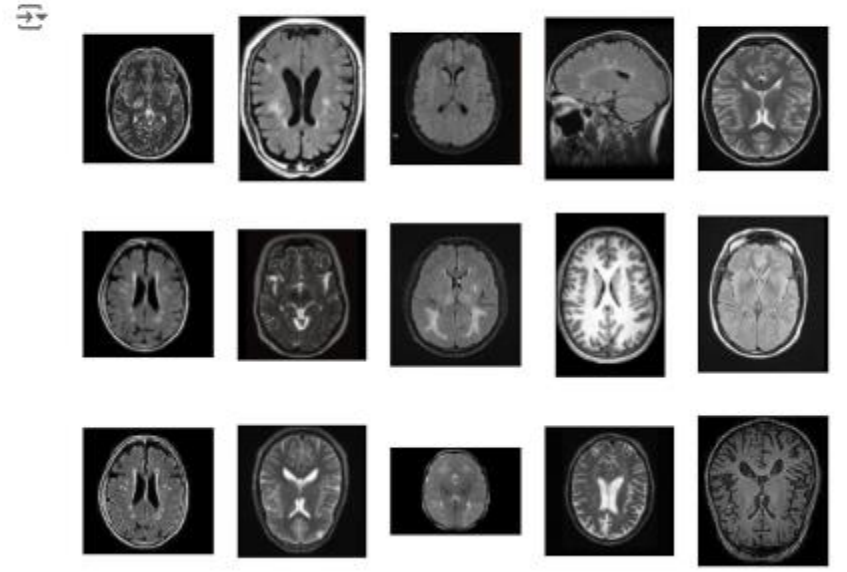
Ako nam prikaz veličine niza nije dovoljan da se uvjerimo da su slike stvarno unutar njega možemo ih i prikazati na idućoj stranici.

```
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(Tumor[i], cmap='bone')
    axi.set(xticks=[], yticks=[])
```



Slika 9 Prikaz slika mozga sa tumorom

```
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(No_Tumor[i], cmap='bone')
    axi.set(xticks=[], yticks=[])
```



Slika 10 Prikaz slika mozga bez tumora

Oba koda na prethodnoj stranici su identična pa nema potrebe prolaziti kroz oba. Naime, U prvoj liniji kod se uz pomoć funkcije „plt.subplots(3, 5)“ kreira „subplot“ sa 3 reda i 5 stupaca. Zatim se koristi for petlja kako bi se u svakoj njenoj iteraciji uzeo i-ti element niza „Tumor“ te se zatim stavlja na odgovarajuće mjesto na „subplotu“.

3.2. Pred-procesiranje podataka

Modeli dubokog učenja, poput VGG16, često imaju određene zahtjeve u vezi dimenzija ulaznih slika. Arhitektura VGG16 je dizajnirana tako da prima slike dimenzija 224x224 piksela kao ulaz.

Da bi promijenili dimenzije ulaznih slika u 224x224 potrebno je napraviti funkciju koja to radi.

```
def resize_images(images, new_size):
  resized_images=[]
  for image in images:
    resized_image = cv2.resize(image, new_size)
    resized_images.append(resized_image)
  return resized_images
new_size = (224,224)
resized_images_tumor = resize_images(Tumor, new_size)
resized_images_no_tumor = resize_images(No_Tumor, new_size)
```

Slika 11 Prikaz funkcije za promjenu dimenzija slika

Gore na slici prikazana je funkcija „resize_images“ koja prima dva parametra: „images“ koji predstavlja niz slika kojima želimo promijeniti dimenziju i „new_size“ koji predstavlja dimenzije na koju želimo postaviti slike.

Unutar funkcije inicijalizirali smo prazan niz pod nazivom „resized_images“. Zatim smo for petljom prolazili kroz svaku sliku u nizu images koji je jedan od ulaznih parametara. Unutar for petlje smo za svaku sliku iz niza koristili funkciju „cv2.resize()“, te nakon toga sliku nad kojom je bila pozvana ta funkcija smo dodavali u niz „resized_images“ kojeg smo na kraju funkcije samo vratili.

Možemo vidjeti da smo na kraju slike postavili novu veličinu na 224*224 i spremili slike u nove nizove pod imenom „resized_images_tumor“ i „resized_images_no_tumor“.

Osim promjene dimenzija slika, na slikama 14 i 15 možemo vidjeti da većina slika ima nezanemarivo veliku crnu pozadinu. Intuitivno možemo zaključiti da bi se bilo najbolje riješiti toga na neki način jer bi to moglo smetati kod uvježbavanja modela. Drugim riječima, trebamo sa svih slike izdvojiti područje interesa, to jest mozak.

```
def crop_imgs(set_name):
    set_new = []
    for img in set_name:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)
        extLeft = tuple(c[c[:, :, 0].argmin()][0])
        extRight = tuple(c[c[:, :, 0].argmax()][0])
        extTop = tuple(c[c[:, :, 1].argmin()][0])
        extBot = tuple(c[c[:, :, 1].argmax()][0])
        new_img = img[extTop[1]:extBot[1], extLeft[0]:extRight[0]].copy()
        new_img = cv2.resize(new_img, (224, 224))
        set_new.append(new_img)

    return np.stack(set_new)
```

Slika 12 Prikaz funkcije za obradu slika

Na slici poviše možemo vidjeti funkciju pod imenom „crop_imgs“ koja prima „set_name“, što nam zapravo predstavlja niz slika.

U prvoj liniji funkcije inicijaliziramo prazan niz pod imenom „set_new“, kojeg ćemo popunjavati obrađenim slikama. Nakon inicijalizacije otvaramo for petlju koja će prolaziti kroz svaku sliku niza kojeg pošaljemo funkciji.

Prva linija unutar for petlje uz pomoć funkcije „cv2.cvtColor()“ pretvara sliku u sliku s nijansama sive, što će nam pomoći u daljnjoj analizi slike.

Što se druge linije tiče ona primjenjuje funkciju „cv2.GaussianBlur()“ te na taj način koristimo gaussovu operaciju za uklanjanje šuma na našoj slici. Na taj način se uklanja šum i olakšava detekcija rubova.

Nadalje, uz pomoć funkcije „cv2.threshold“ radimo binarizaciju na slici koristeći prag vrijednosti 45. Naime, nakon što se funkcija pozove nad slikom svi pikseli intenziteta manjeg od 45 se postavljaju na intenzitet 0 to jest na crnu boju, a oni poviše 45 na intenzitet 255 to jest na bijeli boju. Prag 45 je odabran jer je on najviše pogodan slikama koje mi koristimo.

Nakon ovoga pozivaju se funkcije koje vrše eroziju i dilataciju nad slikom. Naime, erozija i dilatacije su operacije koje se često koriste u kontekstu problem računalnog vida. U ovom slučaju eroziju koristimo kako bi se uklonili šumovi na slikama, a dilataciju kako bi se popunile rupe na slikama koje su mogle nastati u slučaju provođenja operacije erozije.

Nakon što smo proveli ove dvije operacije, iduće dvije pod imenima „cv2.findContours()“ i „imutlis.grab_contours“ koristimo kako bi pronašli konture na prethodno binariziranoj slici i prilagodili te iste konture da bih se rad s njima olakšao.

Nakon toga u varijablu „c“ spremamo najveću konturu u listi kontura koje smo izdvojili prethodno. Ta spremljena kontura bi za svaku slika trebala obuhvaćati područje interesa slike to jest mozak.

Nakon ovoga iduće četiri linije odnose se na to da tražimo ekstremne točke sa svih strana kontura. Znači tražimo desnu, lijevu, gornju i donju ekstremnu točku. Nakon toga „režemo“ područje interesa iz slike koristeći te ekstremne točke.

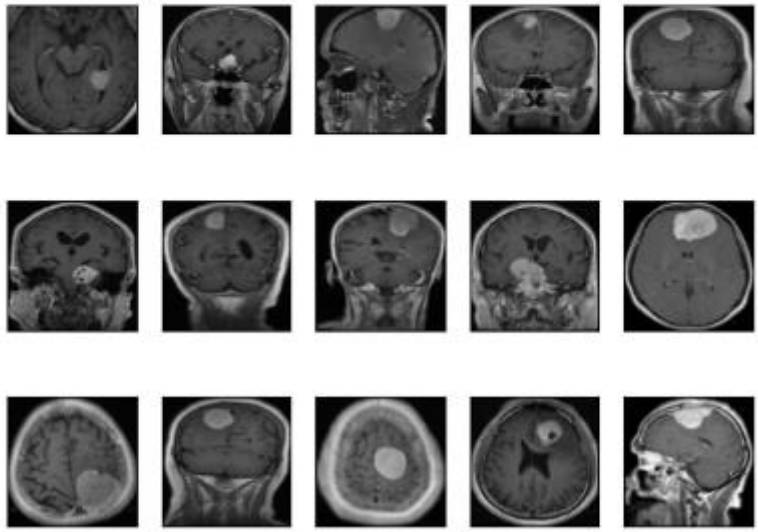
Kada smo sve to napravili opet koristimo prethodno definiranu funkciju za promjenu dimenzionalnosti slike i pomoću nje mijenjamo dimenzionalnost nove slike u 224x224 piksela.

Na kraju svega dodajemo novu obrađenu sliku u niz koji smo definirali na početku te na kraju vraćamo novi niz slika.

Da bih se uvjerili da ovo radi pozvat ćemo ovu funkciju nad našim podacima i opet prikazati nekolicinu obrađenih slika.

```
[ ] crop_img_tumor = crop_imgs(resized_images_tumor)
```

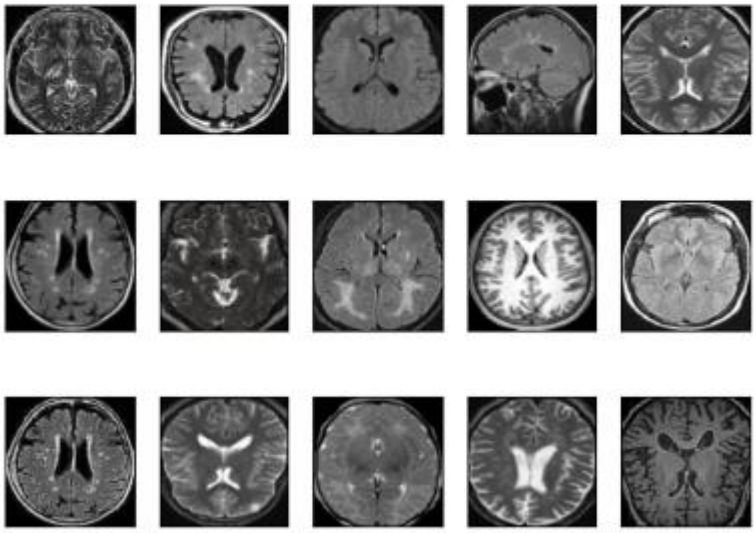
```
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(crop_img_tumor[i], cmap='bone')
    axi.set(xticks=[], yticks=[])
```



Slika 13 Prikaz obrađenih slika mozga s tumorom

```
[ ] crop_img_no_tumor = crop_imgs(resized_images_no_tumor)
```

```
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(crop_img_no_tumor[i], cmap='bone')
    axi.set(xticks=[], yticks=[])
```



Slika 14 Prikaz obrađenih slika mozga bez tumora

Na slikama 13 i 14 možemo vidjeti da smo pozvali metodu za obradu slika nad nizovima te smo zatim prikazali 15 slika mozga s tumorom i 15 bez tumora.

Ako usporedimo ove dvije slike sa slikama 9 i 10 možemo primjetiti da su znatno drukčije te nam je to dokaz da smo ovaj dio dobro napravili

3.3. Podjela podataka

Nakon što su slike koje ćemo koristiti spremne za uvježbavanje modela, ostalo nam je još samo podijeliti te podatke. Prije nego što ih podijelimo također je potrebno nekako označiti slike s tumorom i one bez tumora da ih model može uspješno razlikovati. To ćemo napraviti na sljedeći način:

```
data = np.concatenate((crop_img_tumor, crop_img_no_tumor))
labels = np.concatenate((np.ones(len(crop_img_tumor)), np.zeros(len(crop_img_no_tumor))))
```

Slika 15 Prikaz označavanja slika

U prvoj liniji koda koristi se funkcija „np.concatenate()“ kako bi se niz slika s tumorom i bez njega spojio u jedan pod nazivom „data“. Što se druge linije tiče, dio „np.ones(len(crop_img_tumor))“ stvara niz s vrijednostima jedan dužine jednake broju slika s tumorom, isto tako se radi i za slike bez tumora no za njih se stvara niz popunjen nulama. Na ovaj način smo označili koje to slike imaju tumor i koje nemaju. Na kraju možemo vidjeti da opet spajamo dva niza u jedan koji se zove „labels“.

```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

Slika 16 Prikaz podjele podataka u skupove za uvježbavanje i testiranje

Linija koda na slici poviše koristi funkciju „train_test_split“ kako bi se podaci podijelili na one za uvježbavanje i testiranje. Funkcija prima niz „data“ i niz „labels“ koji sadržavaju podatke i oznake za njih, osim toga funkcija još prima veličinu skupa za testiranje, za koju smo odredili da će biti 0.2 ili 20%, znači 80% podataka za uvježbavanje i 20% podataka za testiranje. Zadnji parametar je „random_state“ koji smo postavili na 42 iz razloga da svaki put kada pokrenemo kod isti podaci budu izabrani za uvježbavanje i testiranje, te na taj način uvijek imamo iste dosljedne rezultate. Rezultat izvršavanja ove metode s ovim parametrima će biti četiri niza. Niz X_train će biti niz podataka koji će sadržavati slike korištene za uvježbavanje modela. X_test će biti niz podataka korišten za vrednovanje perfomansi modela. „y_train“ će biti niz oznaka jedinica i nula, to jest slika s tumorom i bez tumora korištene u uvježbavanju modela, dok će „y_test“ biti isto to samo za vrednovanje modela.

3.4. Arhitektura modela

Sada kad smo napravili sve što smo trebali sa podacima, možemo prijeći i na samu izgradnju i arhitekturu našeg modela.

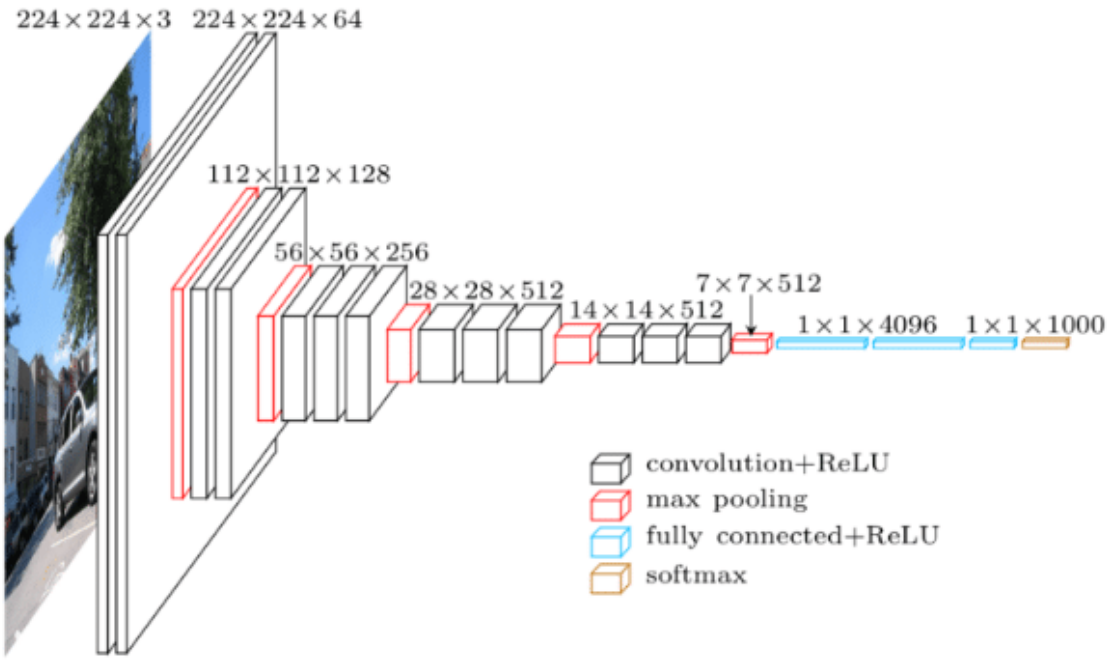
```
IMG_SIZE = (224, 224)
base_model = VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=IMG_SIZE + (3,)
)
```

Slika 17 Prikaz početnog dijela modela

Na slici 17 možemo vidjeti da smo u prvoj liniji u varijablu „IMG_SIZE“ spremili veličinu slika koja će se koristiti za model, a to je 224x224 piksela iz razloga već spomenutog na početku poglavlja 3.2.

Nakon toga smo unutar varijable „base_model“ napravili instancu VGG16 arhitekture. VGG16, kao što smo spomenuli, je vrsta konvolucijske neuronske mreže koja se često koristi u klasifikaciji slika, detekciji objekata i segmentaciji slika.

Da bi objasnili arhitekturu ove konvolucijske neuronske mreže možda je najbolje da to napravimo uz pomoć slikovnog prikaza njene arhitekture.



Slika 18 Prikaz arhitekture VGG16 (Sugata i sur., (2017), Leaf App: Leaf recognition with deep convolutional neural networks. IOP Conference Series: Materials Science and Engineering)

Na slici 18 možemo vidjeti vizualno dočaranu arhitekturu VGG16. Naime, kao što smo već nekoliko puta naglasili, ulazni sloj ima dimenzija 224x224x3, nakon njega slijede dva konvolucijska sloja koja sadrže po 64 filtera veličine 3x3, čiji je način rada već opisan. Iza ova dva sloja slijedi jedan „max-pooling“ sloj sažimanja veličine 2x2 i korakom 2, kojeg smo također već opisali. Sloj sažimanja slijede još dva konvolucijska sloja koja sadrže 128 filtera veličine 3x3. Nakon toga ide isti sloj sažimanja kao prethodno navedeni, no iza ovoga slijede dva konvolucijska sloja po 256 filtera veličine 3x3 i još dva seta po tri konvolucijska sloja po 512 filtera veličine 3x3. Nakon toga opet ide isti sloj sažimanja kao i prethodni kojeg prati dva konvolucijska sloja filtera veličine 3x3. Nakon toga slijedi „flattening“ sloj koji nam služi da bih mogli ubaciti potpuno povezani sloj. Potpuno povezani sloj se sastoji od tri potpuno povezana sloja sa „ReLu“ aktivacijskom funkcijom. Još je bitno napomenuti da se je na slici 18 prikazana „Softmax“ funkcija na kraju, bitno je znati da se ona koristi kada je u pitanju višeklasna klasifikacija, te da ćemo mi u našem modelu pokazati kako implementirati „Sigmoid“ funkciju koja se koristi kada je riječ o binarnom ishodu

Sada, kada smo opisali arhitekturu VGG16 možemo se vratiti na sliku 17. Kao što možemo vidjeti unutar arhitekture našeg početnog modela nismo koristili parametre po „default-u“, već smo nekolicinu njih promijenili, to jest postavili na neke vrijednosti. Linijom „weights=imagenet“ postavili smo da će se koristiti težine koje su prethodno naučene na ImageNet skupu podataka. ImageNet je jako velik skup podataka koji sadrži slike različitih objekata koji su podijeljeni u klase. Težine koje su naučene na tim podacima obuhvaćaju mnoge bitne informacije i karakteristike koje mogu pomoći u prepoznavanju objekata sa slika. Mi smo ovo iskoristili iz razloga što korištenje prethodno naučenih težina kao početnih težina za novi model može ubrzati uvježbavanje novog modela i poboljšati njegove performanse.

Nadalje smo postavili „include_top“ na „False“. Ovo nam se nadovezuje na prethodno postavljanje početnih težina na već naučene iz razloga što smo ovom linijom istaknuli da iz prethodnog modela, od kojeg dolaze već spomenute početne težine, ne želimo učitati potpuno povezani sloj. Ovo radimo zbog toga da dobijemo izlaze iz zadnjeg konvolucijskog sloja tog prethodnog modela, što su u principu izdvojene značajke koje će pomoći našem modelu. Ova linija nam također utječe na to da naš VGG16 model nema potpuno povezani sloj na kraju svoje arhitekture. Na taj način smo zaobišli aktivacijsku funkciju „softmax“ koja je postavljena u taj sloj, te se tako kasnije možemo bolje prilagoditi problemu kojim se mi bavimo tako što ćemo samo dodati potpuno povezani sloj na kraj arhitekture i funkciju u njega koja odgovara prirodi našeg problema.

I na kraju smo „input_shape“ postavili na već definiranu varijablu „IMG_SIZE“ i uz to smo dodali „+ (3,)“. Što se ovog dijela tiče, mislim da je možda nejasno na što se „+ (3,)“ odnosi. Naime, taj dio se ne odnosi na ništa drugo osim na dodatnu dimenziju 3, koja označava tri kanala boja koje VGG16 također očekuje kao ulaz. Osim toga mislim da bih trebalo biti jasno da se ta tri dodatna kanala, skupa sa definiranim dimenzijama ulaznih slika postavljaju kao očekivani ulaz za model.

Nakon što smo definirali arhitekturu najbitnijeg dijela modela sada ga možemo proširiti.

```
NUM_CLASSES = 1
model = Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))
model.layers[0].trainable = False
model.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(lr=1e-4),
    metrics=['accuracy']
)
```

Slika 19 Prikaz konačnog modela

Na slici 19 možemo vidjeti prikaz konačnog modela kojeg ćemo koristiti u binarnoj klasifikaciji tumora mozga i kojeg ćemo sada i objasniti.

Na početku definiramo broj klasa koji će biti u izlaznom sloju. S obzirom da je ovo binarna klasifikacija u varijablu „NUM_CLASSES“ spremamo 1.

Nakon toga stvaramo instancu modela pod imenom „model“ koristeći „Sequential“ model. Drugim riječima, stvaramo sekvencijalni model zbog toga što na taj način imamo mogućnost dodavanja slojeva jednog za drugim. Kada smo to napravili, prvi korak je u instancu „model“ uz pomoć funkcije „add()“ dodati naš početni model prikazan na slici 17 kao prvi sloj. Nakon prvog sloja, dodajemo „Flattening“ sloj, prisjetimo se da je „Flattening“ sloj nužan kako bi mogli dodati potpuno povezani sloj. Iako imamo sve da bi dodali potpuno povezani sloj, prije njega ćemo još dodati i „Dropout“ sloj. Ovaj sloj je zapravo tehnika koja se često koristi kod ovakvih problema kako bi se spriječila već spominjana prenaučenost modela. „Dropout“ sloj pomaže oko ovog problema tako da nasumično isključi određen postotak neurona tijekom svake iteracije treninga modela, te na taj način otežava memoriranje specifičnih značajki od strane modela. Parametar koji smo poslali ovom sloju je 0.5, što u prijevodu znači da će naš model kroz svaku iteraciju uvježbavanja isključiti 50% slučajnih neurona. Nakon što smo ovo napravili dodajemo potpuno povezani sloj. Njemu šaljemo dva parametra, prvi je broj klasa kojeg smo definirali na početku, a drugi je aktivacijska funkcija „sigmoid“. Koristimo tu aktivacijsku funkciju iz razloga što se bavimo binarnom, a ne višeklasnom klasifikacijom.

Kad smo dodali sve slojeve još nam je samo ostalo malo unaprijediti model.

Linijom „model.layers[0].trainable = False“ govorimo modelu da ne želimo uvježbavati težine tog sloja tijekom uvježbavanja cijelog modela. Što se tiče našeg modela, mi smo ovo postavili samo za početni sloj „base_model“ iz razloga da se ne gube informacije koje su već naučene u prethodnom modelu.

Osim svega gore navedenog moramo postaviti neke parametre za proces učenja modela. To radimo funkcijom „model.compile()“ koja u našem slučaju „loss“ postavlja na „binary_crossentropy“ što je tipičan odabir za binarnu klasifikaciju. Nadalje, „optimizer“ smo postavili na „RMSprop“, koji je jedan od najpopularnijih metoda za optimizaciju. I na kraju smo kao metriku koja se koristi za vrednovanje modela postavili „accuracy“ ili točnost. Ona u ovom slučaju mjeri omjer točno klasificiranih primjera naspram ukupnog broja primjera.

3.5. Uvježbavanje modela

Sada kada smo završili izgradnju našeg modela vrijeme je da započnemo njegovo uvježbavanje. Postupak uvježbavanja vrlo je složen te funkcionira na način da se iterativno prolazi kroz skup podataka za uvježbavanje definiran broj puta. Iako je proces složen, možemo ga napraviti u svega par koraka kao što vidimo na slici ispod.

```
epohe = 30

model.fit(X_train, y_train, epochs=epohe, batch_size=32)
```

Slika 20 Prikaz uvježbavanja modela

Na slici 20 možemo vidjeti da smo broj epoha postavili na 30, to je broj koliko puta model prolazi kroz skup podataka za uvježbavanje. U drugoj liniji koristimo metodu „fit()“ za uvježbavanje modela. Parametri „X_train“ i „y_train“ su ulazni podaci i odgovarajuće labele za njih. Parametar „epochs“ postavljamo na vrijednost varijable koji smo inicijalizirali na početku slike. Parametar „batch_size“ određuje koliko će se podataka koristiti u svakoj iteraciji u procesu optimizacije. Naprimjer, ako imamo 1000 podatka za učenje i „batch_size“ postavimo na 32, to znači da će se naš model uvježbavati koristeći grupe od 32 podatka u svakoj iteraciji. Prednost ovoga je u tome što tako možemo ubrzati proces uvježbavanja.

3.6. Predikcija i vrednovanje modela

Nakon uvježbavanja modela potrebno je napraviti predviđanje nad skupom podataka za testiranje.

```
y_pred = model.predict(X_test)
```

Slika 21 Prikaz predviđanja modela

Kao što možemo vidjeti na slici 21, predviđanje modela na skupu podataka za testiranja radimo uz pomoć metode „predict()“ koja prima podatke za testiranje. Rezultat ove metode, to jest predviđene vrijednosti za svaki podatak iz skupa „X_test“, spremamo u varijablu „y_pred“.

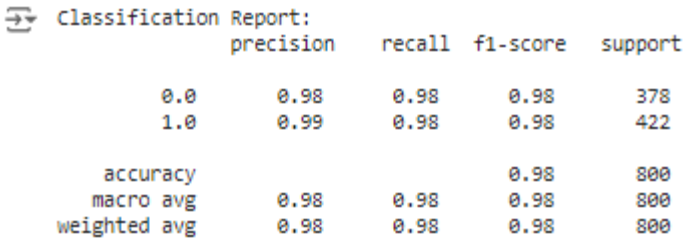
Sada kada smo obavili predviđanje, uz pomoć rezultata možemo napraviti vrednovanje modela na način koji je prikazan na slici ispod.

```
y_pred_flat = np.squeeze(y_pred)
y_pred_binary = (y_pred_flat > 0.5).astype(int)
report2 = classification_report(y_test, y_pred_binary)
print("Classification Report:\n", report2)
```

Slika 22 Prikaz postupka za vrednovanje modela

Da bih uopće napravili vrednovanje, moramo prvo pripremiti podatke za to. Kao što vidimo na slici 22, prvo koristimo funkciju „np.squeeze()“ kako bi uklonili suvišni dimenziju u podacima te kako bi se dobio 1D niz vjerojatnosti za svaki primjer u skupu. Nadalje, kada smo dobili vjerojatnosti predikcija treba ih pretvoriti u binarni oblik jer one obično jako malo odstupaju od 0 ili 1. Ovdje smo vjerojatnosti pretvorili u binarni oblik primjenom praga od 0.5. To znači da ako je vrijednost veća od 0.5, predviđena vrijednost je 1, a inače je 0. Rezultate na kraju ove linije koda pretvaramo u cijele brojeve primjenom svojstva „astype(int)“.

Sada kad smo pripremili podatke koristimo funkciju „classification_report()“ kako bi generirali klasifikacijski izvještaj koji predstavlja detaljni izvještaj perfomansi modela uz pomoć nekolicine metrika .Ova funkcija prima podatke sačuvane za testiranje i predviđene vrijednosti za njih. Na kraju slike 22 možemo vidjeti da prikazujemo klasifikacijski izvještaj uz pomoć metode „print()“, te njega možemo vidjeti na idućoj stranici.



Classification Report:

	precision	recall	f1-score	support
0.0	0.98	0.98	0.98	378
1.0	0.99	0.98	0.98	422
accuracy			0.98	800
macro avg	0.98	0.98	0.98	800
weighted avg	0.98	0.98	0.98	800

Slika 23 Prikaz klasifikacijskog izvještaja

Kao što vidimo u ovom izvješću imamo stvarno dosta metrika i mi ćemo ih sve probati objasniti. Prva je preciznost ili „precision“, ona je omjer „true positives“ predikcija prema svim pozitivnim predikcijama. Za klasu 0 je ona iznosi 98%, dok za klasu 1 iznosi 99%. „Recall“ ili odziv nam je metrika koja se odnosi na omjer „true positives“ predikcija prema svim stvarnim pozitivnim predikcijama. Ova metrika za klasu 0 i za klasu 1 ima isti rezultat, a to je 98%. Ove dvije metrike su vrlo slične, te da ne bi bilo zabune preciznost možemo shvatiti kao metriku kvalitete, dok odziv možemo shvatiti kao metriku kvantitete. Nadalje, imamo „f1-score“ koji je zapravo harmonijska sredina preciznosti i odziva, ova metrika također ima isti rezultat, 98%, za obje klase. „Support“ nam predstavlja broj stvarnih primjera u svakoj klasi u skupu podataka za testiranje. „Accuracy“ ili točnost je omjer ukupnog broja točno klasificiranih podatka naspram ukupnog broja podataka. U našem slučaju ona iznosi 98%, što znači da je 98% svih primjera ispravno klasificirano. „Macro avg“ predstavlja prosječnu vrijednost svih preciznosti, odziva i „f1-score-ova“ za sve klase, u našem slučaju svi oni iznose 98%. „Weighted avg“ je također prosječna vrijednost svih metrika za sve klase, no u ovoj metrici su težine proporcionalne broju podataka u svakoj klasi. U našem slučaju vrijednost ove metrike je isto 98%.

S rezultatima navedenim gore možemo zaključiti da naš model ima jako visoke i dobre rezultate za sve metrike, no isto tako moramo biti svjesni da u sklopu ovog projekta raspolažemo samo sa četiri tisuće slika, što nas dovodi do pitanja kako bih se model ponašao da je broj slika veći. Naravno, ovakav model se ne bi pustio u upotrebu prvotno zbog manjka slika i gore spomenutoga neznanja o njegovom ponašanju na većem broju slika, no onda ovaj projekt možemo shvatiti kao prototip i dobar primjer nečega što se može napraviti. To bi bio kraj za prvi dio praktičnog rada, a sada prelazimo na, možda nekima i zanimljiviji, dio.

4. Segmentacija tumora mozga

Sada kad smo gotovi s klasifikacijom tumora možemo prijeći na segmentaciju koju ćemo realizirati uz pomoć konvolucijske neuronske mreže U-Net.

Ovaj dio ćemo raditi u svrhu same lokalizacije tumora, uz pomoć koje bi trebali moći poboljšati performanse autonomnih robotskih operacija.

4.1. Učitavanje i prikaz podataka

Kao i kod klasifikacije, prvi korak pri izradi ovog projekta je učitavanje podataka, no također je bitno spomenuti da smo prije toga uvezli sve biblioteke koje su nam potrebne, te da smo se spojili na google drive jer nam se tamo nalaze podaci.

```
directory_path = '/content/drive/MyDrive/TumorS'

image_files = sorted([os.path.join(directory_path, f) for f in os.listdir(directory_path) if os.path.isfile(os.path.join(directory_path, f))])

TumorS = []

for image_file in image_files:
    image = cv2.imread(image_file)
    if image is not None:
        TumorS.append(image)
    else:
        print(f"Failed to read image: {image_file}")

print(len(TumorS))
```

```
1000
```

```
directory_path = '/content/drive/MyDrive/Maske'

image_files = sorted([os.path.join(directory_path, f) for f in os.listdir(directory_path) if os.path.isfile(os.path.join(directory_path, f))])

Masks = []

for image_file in image_files:
    image = cv2.imread(image_file)
    if image is not None:
        Masks.append(image)
    else:
        print(f"Failed to read image: {image_file}")

print(len(Masks))
```

```
1000
```

Slika 24 Prikaz učitavanja podataka sa google drive-a

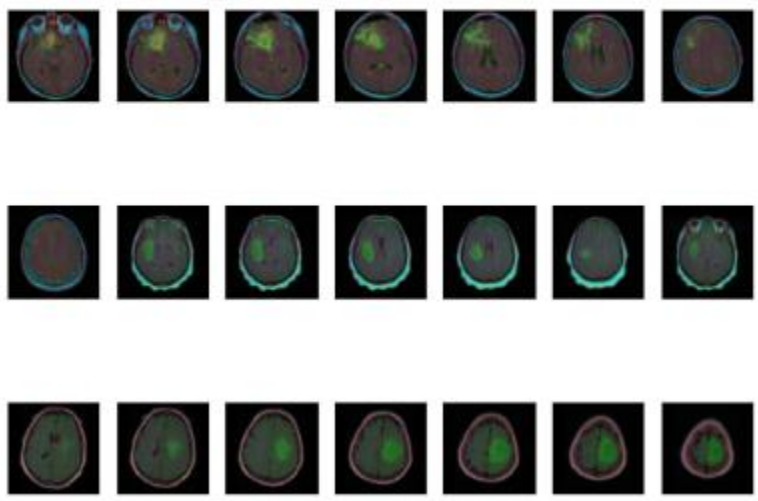
Postupak na slici 24 je vrlo sličan postupku kojeg smo koristili za učitavanje podataka kod binarne klasifikacije, no postoji jedna razlika.

Kao što možemo vidjeti koristili smo funkciju „sorted“ kod učitavanja slika tumora i slika maski za tumore. Ovo smo radili iz razloga da slike i njihove pripadajuće maske budu mapirane. To jest da maska na indeksu 0 u nizu „Masks“ odgovara slici tumora mozga na indeksu 0 u nizu „TumorS“.

Nakon što smo ovo izvršili, ispod svake ćelije možemo vidjeti broj 1000. To nam govori da sveukupno imamo 1000 slika tumora mozga i 1000 maski. Znači za svaku sliku tumora po jednu odgovarajuću masku.

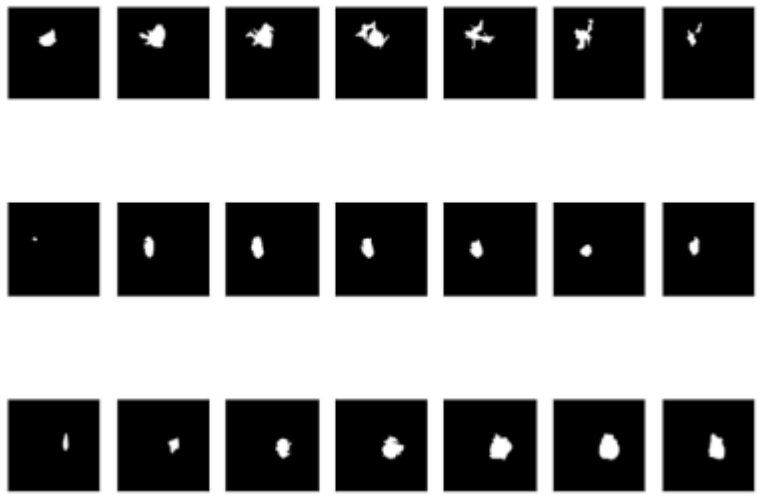
Kako bih ovo bolje dočarali, na slikama na idućoj stranici možemo vidjeti prikaz učitanih podataka.

```
fig, ax = plt.subplots(3, 7)
for i, axi in enumerate(ax.flat):
    axi.imshow(TumorS[i], cmap='bone')
    axi.set(xticks=[], yticks=[])
```



Slika 25 Prikaz slika mozga sa tumorom

```
fig, ax = plt.subplots(3, 7)
for i, axi in enumerate(ax.flat):
    axi.imshow(Masks[i], cmap='bone')
    axi.set(xticks=[], yticks=[])
```



Slika 26 Prikaz slika maski tumora

Kao što možemo vidjeti na slikama 25 i 26, kod za prikaz slika je identičan kao i onaj u prijašnjem dijelu praktičnog rada, pa ga nećemo opet objašnjavati. Osim toga možemo vidjeti da smo na slici 26 prikazali 21 sliku mozga sa tumorom i da na slici ispod prikazane maske odgovaraju pozicijama tumora. Ovo nam govori da smo dobro učitali podatke te da možemo nastaviti s radom.

4.2. Pred-procesiranje podataka

Sad kada smo učitali podatke, iduće na redu je pripremiti ih za upotrebu.

```
def preprocess_images(images, size=(256, 256)):
    preprocessed_images = []
    for img in images:
        img_resized = cv2.resize(img, size)
        img_normalized = img_resized / 255.0
        preprocessed_images.append(img_normalized)
    return np.array(preprocessed_images)

def preprocess_masks(masks, size=(256, 256)):
    preprocessed_masks = []
    for mask in masks:
        mask_resized = cv2.resize(mask, size)
        mask_gray = cv2.cvtColor(mask_resized, cv2.COLOR_BGR2GRAY)
        mask_binary = np.where(mask_gray > 0, 1, 0)
        preprocessed_masks.append(mask_binary)
    return np.array(preprocessed_masks)

tumorS = preprocess_images(TumorS)
masks = preprocess_masks(Masks)
```

Slika 27 Prikaz pred-procesiranja podataka

Kao što možemo vidjeti na slici poviše definirali smo dvije funkcije, jednu koja će obrađivati slike mozga sa tumorom i jednu za maske tumora.

Za slike mozga sa tumorom definirali smo funkciju pod imenom „preprocess_images“ koja prima listu slika „images“ i veličinu „size“ koja je zadana na 256x256. Unutar funkcije definirali smo prazan niz „preprocessed_images“, te smo nakon toga iterirali kroz svaku sliku niza kojeg će funkcija primiti. Unutar petlje veličinu svake slike mijenjamo na već zadanu veličinu 256x256. Osim što mijenjamo veličinu slika, također ih i normaliziramo tako što ih dijelimo sa 255.0. Na ovaj način normaliziramo piksele slike da budu unutar raspona od 0 do 1. Ovakva normalizacija je važna iz razloga jer osiguravamo da su podaci u konzistentnom rasponu, te su posljedično tome gradijenti stabilniji, ovo omogućuje bolju prilagodbu modela na podatke i brže uvježbavanje modela. Nakon što smo normalizirali slike, dodajemo ih u prethodno definiran niz „preprocessed_images“ kojeg na kraju vraćamo kao „NumPy“ niz. Ovo radimo iz razloga što je baratanje sa podacima u „NumPy“ nizu učinkovitije u usporedbi s klasičnim Python listama.

Što se funkcije za obradu slika maski tumora tiče, ona se od prve funkcije razlikuje samo u dvije linije, koje ćemo jedine i objasniti. Prva linija je ispod linije koja mijenja dimenziju slika. U toj liniji se radi konverzija slike u sivu skalu. Ovo se radi kako bi smanjili složenost i dimenzionalnost podataka, čineći ih prikladnijim za daljnju obradu. Druga linija je odmah ispod prethodno navedene, a ona vodi računa da o tome da svaka slika maske bude binarna. U toj liniji možemo vidjeti da smo svaki piksel područja interesa (tumora) na slici postavili na vrijednost 1, a ostale (pozadinu) na vrijednost 0.

Na samom kraju slike 27 pozvali smo funkcije za pred-obradu na već postojećim nizovima podataka i rezultat funkcija smo spremili u dva niza, „tumorS“ i „masks“

4.3. Podjela podataka

Nakon što su slike za uvježbavanje modela spremne, trebamo još podijeliti te podatke kako bi uvježbali model.

Nakon što su slike koje ćemo koristiti spremne za uvježbavanje modela, ostalo nam je još samo podijeliti te podatke. Prije nego što ih podijelimo također je potrebno nekako označiti slike s tumorom i one bez tumora da ih model može uspješno razlikovati. To ćemo napraviti na sljedeći način:

```
X_train, X_test, y_train, y_test = train_test_split(tumorS, masks, test_size=0.2, random_state=42)
y_train = np.expand_dims(y_train, axis=-1)
y_test = np.expand_dims(y_test, axis=-1)
```

Slika 28 Prikaz podjele podataka u skupove za uvježbavanje i testiranje

Na slici 28 možemo vidjeti korištenje funkcije „train_test_split“ kako bi podijelili podatke na one za uvježbavanje i testiranje modela. Navedena funkcija prima niz slika mozga s tumorom i niz odgovarajućih maski te ih dijeli prema veličini koju smo definirali u varijabli „test_size“. Iz razloga što smo „test_size“ postavili na vrijednost 0.2, naša metoda u „X_train“ pohranjuje 80% slika mozga s tumorom, a u „y_train“ 80% slika maski tumora. Osim toga u „X_test“ se pohranjuje 20% slika mozga s tumorom, dok „y_test“ sadrži 20% slika maski tumora. Zadnji parametar koji ova metoda prima je „random_state“. Njega smo postavili na 42 iz razloga da svaki put kada pokrenemo kod isti podaci budu izabrani za uvježbavanje i testiranje, te na taj način uvijek imamo dosljedne rezultate.

Kada smo podijelili podatke na skupove za uvježbavanje i testiranje ostalo nam je još samo jedan korak da bi ih mogli iskoristiti za uvježbavanje, a to je dodavanje još jedne dimenzije maskama.

Naime, metodama prikazanim na slici 28 dodajemo još jednu dodatnu dimenziju na skupove podataka koji sadrže maske tumora. Ovo radimo iz razloga da bi izbjegli potencijalne probleme s neusklađenim oblicima podataka tijekom uvježbavanja i vrednovanja modela.

4.4. Arhitektura modela

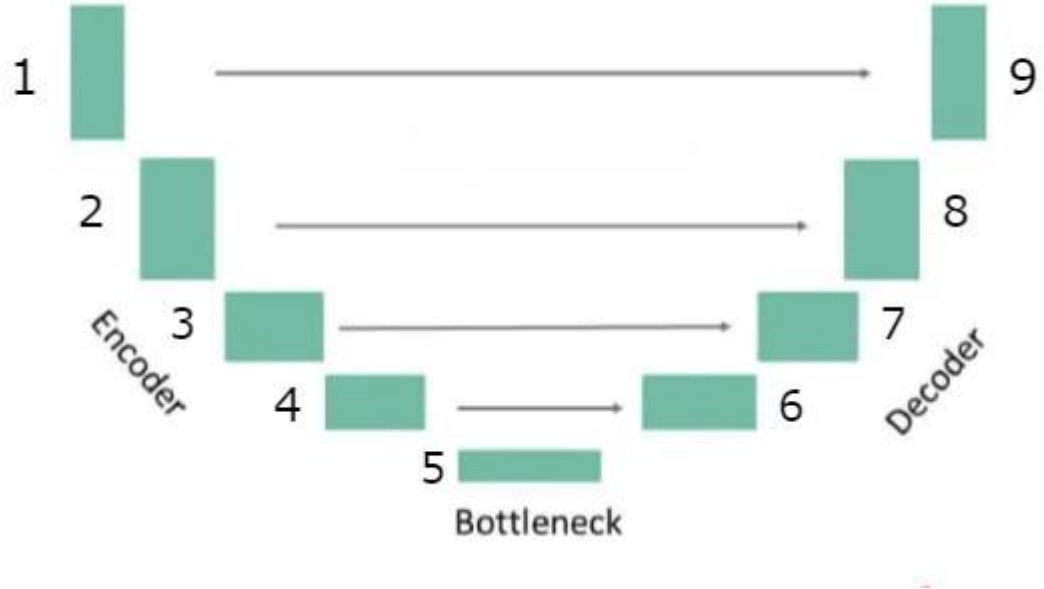
U ovom poglavlju ćemo se upoznati s novom konvolucijskom neuronskom mrežom, U-Net. No prije nego što se usredotočimo na sami kod kojeg ćemo koristiti, malo ćemo se posvetiti općenitijem pristupu kako bih shvatili kako ova mreža uopće radi.

Naime, U-Net mreža sastoji se od 2 dijela: „Encoder“ i „Decoder“.

Prvo ćemo se fokusirati na „encoder“ koji je dio mreže koji služi za ekstrakciju značajki iz ulazne slike. Njegov glavni zadatak je smanjivanje prostorne dimenzionalnosti ulazne slike dok istovremeno povećava apstrakciju samih značajki. To u prijevodu znači da ovaj dio „U-Net“ mreže svakim slojem smanjuje sliku i uči značajke koje su sve više i više apstraktne i složenije. Primjer ovog procesa je taj da rani konvolucijski slojevi otkrivaju jednostavne obrasce na slici, dok dublji slojevi kombiniraju jednostavne obrasce za prepoznavanje složenijih struktura, a zadnji najdublji slojevi uče značajke kao što su oblik objekta i njegov kontekst u sceni.

Sada kada smo objasnili „encoder“ možemo prijeći na drugi dio ove mreže. Naime, „decoder“ je drugi dio „U-Net“ mreže koji prima značajke koje su naučene na „encoder“ dijelu, te služi za generiranje izlazne slike. On počinje s „up-sampling“ slojevima koji povećavaju dimenzionalnost podataka s ciljem da mreža ponovno dobije početnu rezoluciju slike. U isto vrijeme kad se provodi „up-sampling“ često se koristi i „connecting paths“ tehnika koja izravno spaja značajke s „encoder“ dijela na „decoder“ dio. Ova tehnika pomaže mreži da se koriste detaljne informacije sa nižih razina mreže kako bi izlazna slika bila bolja. Nakon ova dva koraka koriste se i konvolucijski slojevi kako bi se značajke generirale i oblikovale za izlazni format. Svaki konvolucijski sloj u „decoder“ dijelu pomaže u obradi i oblikovanju značajki kako bi se postigla željena izlazna slika.

Iz razloga što je sve ovo vrlo apstraktno, pokušat ćemo pojasniti sve ove pojmove preko slike arhitekture ove mreže.



Slika 29 Prikaz arhitekture U-Net mreže

Na slici 29 možemo vidjeti tipičnu arhitekturu „U-Net“ mreže koju ćemo mi sada pojasniti na primjeru. Naime, zamislimo da imamo ulaznu sliku koja ulazi na dio „encoder-a“ pod brojem jedan. Nakon toga, sloj mreže pod brojem jedan radi konvoluciju nad slikom te je prosljeđuje od gore prema dolje gdje svaki sloj pod brojevima od dva do četiri također provodi konvoluciju nad svojim ulaznim podacima koji su zapravo izlazni podaci od prethodnog sloja.

Kada završimo sa svim slojevima „encoder-a“ krajnji podaci dolaze do „bottleneck-a“ , gdje im se još jednom smanji dimenzionalnost, provode se konvolucijski sloj nad njima te ih se zatim „up-sample-a“ i šalje u „decoder“ dio ove mreže.

Nakon što smo poslali podatke u „decoder“ prosljeđujemo ih od dolje prema gore, gdje ih svaki sloj „up-sample-a“ kako bi se dobila dimenzionalnost početne slike. Ovdje također na snagu stupa i „connecting paths“ dio ove arhitekture. Naime, „connecting paths“ su strelice na slici 29 koje izlaze sa svakog sloja „encoder“ dijela izravno prosljeđuju na odgovarajuće slojeve „decoder“ dijela, gdje se onda izlazi „encodera“ i izlazi prethodnog sloja „decoder-a“ spajaju u jedan izlaz koji se prosljeđuje idućem sloju „decoder-a“. Kada na primjeru slike 29 podaci prođu kroz zadnji sloj „decoder-a“, koji je označen brojem devet, tek tada dobivamo izlaznu sliku.

Sada kada smo približili način rada ove mreže, možemo prijeći na implementaciju ove arhitekture unutar našeg projekta.

```
def unet_model(input_size=(256, 256, 3)):
    inputs = Input(input_size)

    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same')(pool3)
    conv4 = Conv2D(512, 3, activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    # Bottleneck
    conv5 = Conv2D(1024, 3, activation='relu', padding='same')(pool4
    conv5 = Conv2D(1024, 3, activation='relu', padding='same')(conv5

    # Decoder
    up6 = UpSampling2D(size=(2, 2))(conv5)
    up6 = concatenate([up6, conv4], axis=3)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(up6)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(conv6)

    up7 = UpSampling2D(size=(2, 2))(conv6)
    up7 = concatenate([up7, conv3], axis=3)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(up7)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(conv7)

    up8 = UpSampling2D(size=(2, 2))(conv7)
    up8 = concatenate([up8, conv2], axis=3)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(up8)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(conv8)

    up9 = UpSampling2D(size=(2, 2))(conv8)
    up9 = concatenate([up9, conv1], axis=3)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(up9)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(conv9)

    conv10 = Conv2D(1, 1, activation='sigmoid')(conv9)

    model = Model(inputs=[inputs], outputs=[conv10])

    return model

model = unet_model()
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', MeanIoU(num_classes=2)])
```

Slika 30 Prikaz arhitekture modela

Na slici 30 možemo vidjeti implementaciju „U-Net“ arhitekture. Kao što vidimo arhitekturu definiramo kao funkciju „unet_model“, koja kao jedini parametar prima „input_size=(256, 256, 3)“. U prvoj liniji unutar funkcije postavljamo „inputs“, koji nam predstavlja ulazni sloj model“, na „input_size“. Ovim smo definirali da su ulazne slike dimenzija 256x256 i da imaju 3 kanala boja.

Nakon što smo postavili veličine ulaznih podataka možemo krenuti sa definiranjem „encoder-a“. Na slici 30 označeno je komentarom gdje počinje „encoder“ te se ispod toga definiraju njegovi blokovi, kojih ima četiri, a svaki blok je od drugog odvojen praznim retkom.

Što se prvog bloka tiče, on sadrži dva konvolucijska sloja, od kojih oba izvode konvoluciju s 64 filtera veličine 3x3. Osim ta dva konvolucijska sloja ovaj blok sadrži i „max-pooling“ sloj sažimanja veličine 2x2. Još je bitno napomenuti da parametri unutar zagrada na kraju svake linije označavaju ulazne parametre za sloj u toj liniji ((inputs) označava ulazne podatke za conv1) i na to da će svaki konvolucijski sloj, osim zadnjega (izlaznog), imati aktivacijsku funkciju „relu“

Drugi blok sadrži dva konvolucijska sloja, koja rade konvoluciju s 128 filtera veličine 3x3. Osim njih blok sadrži i „max-pooling“ sloj sažimanja veličine 2x2.

Treći blok također sadrži dva konvolucijska sloja, koja rade konvoluciju s 256 filtera veličine 3x3 te skupa s njima sadrži i „max-pooling“ sloj sažimanja veličine 2x2.

Što se posljednjeg bloka „encoder-a“ tiče, on sadrži dva konvolucijska sloja, od kojih oba izvode konvoluciju s 512 filtera veličine 3x3. Osim njih, ovaj blok sadrži i „max-pooling“ sloj sažimanja veličine 2x2.

Nakon što smo završili s „encoder“ dijelom, prelazimo na „bottleneck“ dio. Ovaj dio nam sadrži jedan blok od dva konvolucijska sloja, koja provode konvoluciju s 1024 filtera veličine 3x3.

Nakon „bottleneck-a“ slijedi nam „decoder“. Naš „decoder“ se sastoji od četiri bloka od kojih svaki provodi „up-sampling“, spajanje pomoću „connecting paths“ i konvoluciju uz pomoć konvolucijskih slojeva.

Prvi blok „decoder-a“ sadrži „up-sampling“ sloj koji udvostručuje dimenziju ulazne slike. Osim njega sadrži i „connecting path“ koji spaja ovaj blok „decoder-a“ s odgovarajućim blokom „encoder-a“ i naposlijetku sadrži dva konvolucijska sloja s 512 filtera veličine 3x3.

U drugom bloku također koristimo „up-sampling“ za udvostručavanje dimenzije slike, te osim toga opet spajamo blokove „decoder-a“ i „encoder-a“, pa zatim vršimo konvoluciju nad njima uz pomoć dva konvolucijska sloja s 256 filtera veličine 3x3.

Treći blok je identičan kao i drugi te se samo razlikuje po tome što mu konvolucijski slojevi imaju 128 filtera veličine 3x3.

Četvrti blok je također identičan s jedinom razlikom da mu konvolucijski slojevi imaju 64 filtera veličine 3x3.

Sada kada smo ovo sve definirali ostao nam je još samo izlazni sloj koji je zapravo završni konvolucijski sloj koji koristi „sigmoid“ aktivacijsku funkciju i daje izlaznu sliku s jednim kanalom kao izlaz.

Nakon što smo definirali model unutar funkcije instancirali smo ga kao varijablu „model“. Nad tom varijablom smo pozvali metodu „compile“ uz pomoć koje ćemo definirati još nekolicinu bitnih parametara.

Što se „optimizer“ parametra tiče, on nam određuje koji ćemo algoritam koristiti za optimizaciju težina mreže tijekom procesa uvježbavanja. U našem slučaju koristimo „adam“ optimizator.

Parametar „loss“ nam označava koju ćemo funkciju gubitka koristiti za vrednovanje greške između stvarnih i predviđenih vrijednosti modela. U našem slučaju koristimo „binary_crossentropy“.

Metrike koje koristimo za procjenu perfomansi modela su „accuracy“ i „MeanIoU“. „Accuracy“ smo već objasnili kod binarne klasifikacije, a što se „MeanIoU“ tiče, to nam je metrika koja se često koristi u problemima segmentacije iz razloga što mjeri koliko dobro se predviđene maske podudaraju s stvarnim maskama.

4.5. Uvježbavanje modela

Sada kad smo napravili sve potrebno da bi uvježbali model, možemo krenuti s tim.

```
model.fit(X_train, y_train, epochs=10, batch_size=16)
```

Slika 31 Prikaz uvježbavanja modela

Kao što možemo vidjeti na slici 31 broj epoha za koliko uvježbavamo naš model je 10, dok su grupe podataka po iteraciji postavljene na 16.

4.6. Vrednovanje modela

Što se vrednovanja tiče, postupak za nju možemo vidjeti na slici koja slijedi.

```
model.evaluate(X_test, y_test)
```

Slika 32 Prikaz vrednovanja modela

Na slici 32 možemo vidjeti metodu „evaluate()“ koju koristimo da bi napravili vrednovanje našeg modela. Ona daje rezultate prikazane na slici ispod.

```
- loss: 0.0288 - accuracy: 0.9889 - mean_io_u: 0.4831
```

Slika 33 Prikaz izvještaja vrednovanja

Na slici 33 možemo vidjeti 3 metrike koje nam je metoda vratila: „loss“, „accuracy“ i „mean_io_u“. Metoda nam je vratila ove metrike jer smo ih mi već definirali u arhitekturi modela kao bitne.

Ako ponovno pogledamo sliku 33 možemo vidjeti da nam je loss jednak 0.0288, što zapravo znači da je on vrlo nizak te nam to sugerira da je model dobro naučio karakteristike podatka i da precizno predviđa.

Točnost ili „accuracy“ nam u ovom slučaju iznosi 98.89%. To je vrlo visoka točnost i ona pokazuje da model uspješno razlikuje piksele koji pripadaju tumoru od onih pripadajućih pozadini.

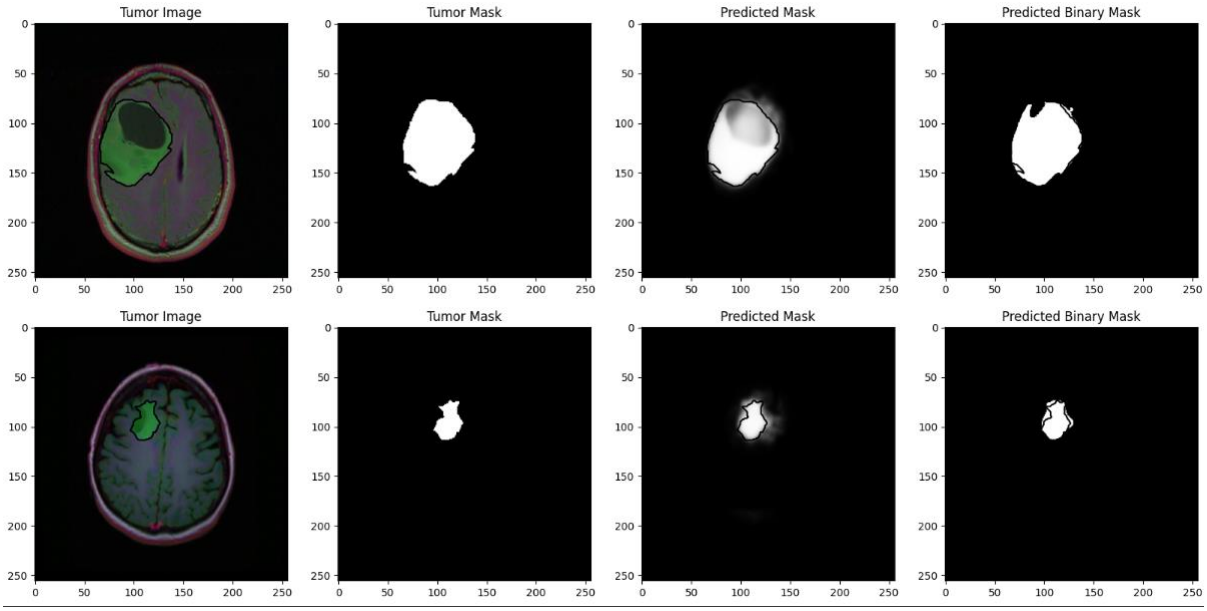
„Mean IoU“ ili „mean intersection over union“ za naš model iznosi 48.31%. Iako nam je točnost visoka, ova metrika nam sugerira da postoji jako velik prostor za poboljšanje ovog modela u preciznosti segmentacije. No također je bitno napomenuti da je ova metrika izrazito stroga jer kažnjava čak i vrlo male netočnosti kod segmentacije, pa tako mali nedostatci mogu izrazito smanjiti vrijednost ove metrike.

Osim ovih osnovnih metoda vrednovanja možemo i grafički prikazati nekoliko segmentacija napravljenih od strane modela.

```
preds = model.predict(X_test)
binary_preds = (preds > 0.5).astype(np.uint8)
def plot_sample(X, y, preds, binary_preds, ix=None):
    if ix is None:
        ix = np.random.randint(0, len(X))
    has_mask = y[ix].max() > 0

    fig, ax = plt.subplots(1, 4, figsize=(20, 10))
    ax[0].imshow(X[ix])
    if has_mask:
        ax[0].contour(y[ix].squeeze(), colors='k', levels=[0.5])
    ax[0].set_title('Tumor Image')

    ax[1].imshow(y[ix].squeeze(), cmap='gray')
    ax[1].set_title('Tumor Mask')

    ax[2].imshow(preds[ix].squeeze(), vmin=0, vmax=1, cmap='gray')
    if has_mask:
        ax[2].contour(y[ix].squeeze(), colors='k', levels=[0.5])
    ax[2].set_title('Predicted Mask')

    ax[3].imshow(binary_preds[ix].squeeze(), vmin=0, vmax=1, cmap='gray')
    if has_mask:
        ax[3].contour(y[ix].squeeze(), colors='k', levels=[0.5])
    ax[3].set_title('Predicted Binary Mask')

    plt.show()

plot_sample(X_test, y_test, preds, binary_preds)
for i in range(5):
    plot_sample(X_test, y_test, preds, binary_preds, ix=i)
```

Slika 34 Prikaz koda za grafički prikaz segmentacije

Kod na slici 34 nećemo objašnjavati jer smo već imali dosta primjera za prikaz podataka, već ćemo na idućoj stranici samo prikazati rezultate koje dobivamo kada izvršimo ovaj kod.



Slika 35 Grafički prikaz segmentacije

Na slici 35 možemo vidjeti rezultat koda sa slike 34.

Prva slika nam označava podatak nad kojim se vršila segmentacija, druga predstavlja stvarnu masku tog tumora. Treća slika nam je predviđena maska tumora od strane modela, a četvrta slika je zapravo isto što i treća slika samo u binarnom formatu. Četvrta slika je zapravo krajnji rezultat ovog svega zato što je binarna, kao i maska koja je bila ulazni podatak.

5. Integracija

Nakon što smo obavili klasifikaciju i segmentaciju tumora mozga, vrijeme je da se za kraj osvrnemo na integraciju toga svega u kontekst robotskih kirurških sustava.

Naime, iako već postoji puno napretka u robotskoj kirurgiji, klasifikacija i segmentacija tumora mozga otvaraju vrata za još veći napredak u kontekstu same preciznosti i efikasnosti robotskih operacija.

Korištenjem tehnika dubokog učenja, obrađenih u ovom radu, roboti namijenjeni za operacije mogu dobiti vrlo detaljan prikaz granica tumora i njegove lokacije. Popraćeno tim, integracija podataka segmentacije tumora s robotskim sustavima može vrlo lako omogućiti navigaciju tijekom operacije u realnom vremenu, što bi naposlijetku predstavljalo vrlo bitan korak u automatizaciji robotskih kirurških procedura.

Gore navedene stavke bi naravno imale pozitivan utjecaj i na sam proces liječenja i oporavka pacijenta. Naime, pacijenti bi mogli očekivati bolje ishode operacije, također bi imali brži oporavak, te bi same šanse za komplikacije bile smanjene. Sve ove pozitivne stavke su već navedene i obrađene na početku ovog rada, te bih se uz daljnji rad na projektu ovakvog tipa mogle i ostvariti.

Naravno, sve obrađeno u ovom radu je na neki način prototip i dokaz da se ovako nešto može napraviti, no da bi se ovako nešto pustilo u upotrebu to zahtijeva još puno uloženog rada i istraživanja.

Zaključak

Cilj ovog rada bio je demonstrirati potencijal integracije tehnika dubokog učenja u robotske kirurške sustave. Prednosti koje bi ovakav pristup donio su nedvojbeno od velike važnosti i za pacijenta i za općeniti napredak medicine.

U ovom radu su razrađene dvije tehnike dubokog učenja, jedna za klasifikaciju tumora mozga, druga za segmentaciju.

Klasifikacijom tumora mozga omogućilo se utvrđivanje prisutnosti tumora kod pacijenta, model prikazan u ovom radu je pokazao visoku točnost, što je ključno kod problema ovakvog tipa jer se ipak radi o životu pacijenta.

Segmentacijom tumora omogućena je vrlo detaljna i precizna identifikacija granica tumora, što je od ključne važnosti pri planiranju kirurških zahvata. Model za segmentaciju, obrađen u ovom radu, također se pokazao vrlo točnim i efikasnim.

Što se same integracije ovih tehnika u robotske kirurške sustave tiče, za potrebe ovog rada se nije ulazilo tehničke detalje implementacije. Umjesto toga, fokusirali smo se na demonstraciju samog potencijala tehnika dubokog učenja, te na dobre ishode u slučaju teoretske integracije navedenih tehnika u robotske kirurške sustave.

Bitno je napomenuti da je praktični dio ovog rada samo dokaz koncepta, odnosno demonstracija izvedivosti ove tehnologije. Međutim, kako bi se ovakva rješenja mogla koristiti u stvarnom svijetu i kako bi postala standardni dio medicine, potrebno je još jako puno dodatnog rada i istraživanja.

Literatura

[1] Robot assisted surgery: https://en.wikipedia.org/wiki/Robot-assisted_surgery

[2] Artifficial intelligence in medicine: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6691444/

[3] VGG16: https://www.geeksforgeeks.org/vgg-16-cnn-model/

[4] U-Net: https://www.sciencedirect.com/topics/computer-science/u-net

[5] Hughes i sur., (2023), The Availability, Cost, Limitations, Learning Curve and Future of Robotic Systems in Urology and Prostate Cancer Surgery

[6] Bolf, N. (2021) Strojno Učenje

[7] Mohaiminul i sur., (2019), An Overview of Neural Network

[8] Montesinos-López i sur., (2022). Convolutional Neural Networks. 10.1007/978-3-030-89010-0_13

[9] David i sur. (2016). DeepPainter: Painter Classification Using Deep Convolutional Autoencoders. 20-28. 10.1007/978-3-319-44781-0_3

[10] Antunesi sur., (2022), Benchmarking Deep Learning Methods for Behaviour-Based Network Intrusion Detection

[11] Sugata i sur., (2017), Leaf App: Leaf recognition with deep convolutional neural networks. IOP Conference Series: Materials Science and Engineering

Popis slika

Slika 1 Prikaz robotske konzole i robotskih ruku (Hughes i sur., (2023), The Availability, Cost, Limitations, Learning Curve and Future of Robotic Systems in Urology and Prostate Cancer Surgery) 6
Slika 2 Podjela algoritama strojnog učenja (Bolf, N. (2021) Strojno Učenje) 13
Slika 3 Koraci pri izradi modela strojnog učenja 14
Slika 4 Prikaz arhitekture neuronske mreže 16
Slika 5 Prikaz konvolucijskog sloja (Montesinos-López i sur., (2022). Convolutional Neural Networks. 10.1007/978-3-030-89010-0_13) 17
Slika 6 Primjer procesa sažimanja (David i sur. (2016). DeepPainter: Painter Classification Using Deep Convolutional Autoencoders. 20-28. 10.1007/978-3-319-44781-0_3) 19
Slika 7 Prikaz konačne arhitekture konvolucijske neuronske mreže (Antunes i sur, (2022), Benchmarking Deep Learning Methods for Behaviour-Based Network Intrusion Detection) 20
Slika 8 Prikaz učitavanja podataka sa google drive-a 25
Slika 9 Prikaz slika mozga sa tumorom 27
Slika 10 Prikaz slika mozga bez tumora 27
Slika 11 Prikaz funkcije za promjenu dimenzija slika 28
Slika 12 Prikaz funkcije za obradu slika 29
Slika 13 Prikaz obrađenih slika mozga s tumorom 31
Slika 14 Prikaz obrađenih slika mozga bez tumora 32
Slika 15 Prikaz označavanja slika 33
Slika 16 Prikaz podjele podataka u skupove za uvježbavanje i testiranje 33
Slika 17 Prikaz početnog dijela modela 34
Slika 18 Prikaz arhitekture VGG16 (Sugata i sur., (2017), Leaf App: Leaf recognition with deep convolutional neural networks. IOP Conference Series: Materials Science and Engineering) 35
Slika 19 Prikaz konačnog modela 37
Slika 20 Prikaz uvježbavanja modela 39
Slika 21 Prikaz predviđanja modela 40
Slika 22 Prikaz postupka za vrednovanje modela 40
Slika 23 Prikaz klasifikacijskog izvještaja 41
Slika 24 Prikaz učitavanja podataka sa google drive-a 42
Slika 25 Prikaz slika mozga sa tumorom 44
Slika 26 Prikaz slika maski tumora 44
Slika 27 Prikaz pred-procesiranja podataka 46

Slika 28 Prikaz podjele podataka u skupove za uvježbavanje i testiranje 48
Slika 29 Prikaz arhitekture U-Net mreže 50
Slika 30 Prikaz arhitekture modela 51
Slika 31 Prikaz uvježbavanja modela 54
Slika 32 Prikaz vrednovanja modela 54
Slika 33 Prikaz izvještaja vrednovanja 54
Slika 34 Prikaz koda za grafički prikaz segmentacije 55
Slika 35 Grafički prikaz segmentacije 56