# Segmentacija lijevog atrija iz LGE MRI pomoću konvolucijske neuronske mreže

**Topić, Tamara**



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni studij**

# SEGMENTACIJA LIJEVOG ATRIJA IZ LGE MRI SLIKA POMOĆU KONVOLUCIJSKE NEURONSKE MREŽE

**Diplomski rad**

**Tamara Topić**

**Osijek, 2020.**

SADRŽAJ

# 1. UVOD

Obrada medicinskih slika je postala bitna sastavnica u mnogim područjima medicinskih i laboratorijskih istraživanja kao i kliničke prakse. Medicinske slike pružaju informaciju o anatomiji, fiziološkim i metaboličkim aktivnostima volumena ispod kože te predstavljaju važan dijagnostički instrument za utvrđivanje prisutnosti određenih bolesti. Obrada medicinske slike predstavlja interdisciplinarno istraživačko polje koje obuhvaća primijenjenu matematiku, računarske znanosti, inženjerstvo, biologiju, medicinu i mnoge druge. Digitalna obrada slike izvodi se pomoću računala te taj proces obuhvaća pretvaranje slike u digitalni oblik i naknadno izvršavanje određenih operacija nad njom. Segmentacija slike smatra se najvažnijim postupkom digitalne obrade slike jer izdvaja područje interesa kroz poluautomatski ili automatski postupak. Postupak segmentacije medicinske slike, odnosno prepoznavanje piksela slike na kojima se nalaze organi ili lezije iz medicinskih slika kao što su CT (engl. *computed tomography*) i MRI (engl. *magnetic resonance imaging*) slike, predstavlja najizazovniji zadatak u medicinskoj analizi slike čiji je cilj dobivanje informacije o oblicima i volumenima tih organa.

Segmentacija i 3D rekonstrukcija ljudskog atrija od presudnog je značaja za preciznu dijagnozu i liječenje najčešće srčane aritmije - atrijske fibrilacije. Međutim, ručna segmentacija atrija s medicinskih snimaka dugotrajan je postupak koji je često sklon pogreškama. Pojavljivanjem umjetne inteligencije, posebno metoda dubokog učenja, dobiveno je alternativno rješenje za precizno segmentiranje atrijske strukture iz kliničkih slika. Segmentacija lijevog atrija i procjena njegove morfologije neophodna je za razumijevanje fibrilacije atrija. Automatizacija ovog procesa iz LGE MRI (engl. *late gadolinium enhancement magnetic resonance imaging*) podataka ostvaruje se pomoću konvolucijskih neuronskih mreža koje su pokazale rezultate izrazito visoke točnosti u segmentaciji medicinskih slika.

## 1.1. Zadatak diplomskog rada

Istražiti i opisati način dobivanja LGE MRI slika i njihove karakteristike. Istražiti i opisati kliničku pozadinu (prikaz srca na LGE MRI slikama, dijelove srca s naglaskom na lijevi atrij, klinička potreba) te kratki pregled područja. Objasniti teorijske osnove načina rada konvolucijskih neuronskih mreža. Razviti sustav za segmentaciju lijevog atrija pomoću konvolucijske neuronske mreže. Prikazati i objasniti rezultate te odrediti preciznost izvođenja razvijenog sustava.

## 1.2. Organizacija rada

Ovaj rad je podijeljen na šest poglavlja. Unutar drugog poglavlja dan je pregled trenutnih metoda segmentacije lijevog atrija te je ukratko pojašnjen osnovnim principom upotrebljenih metoda kao i dobiveni rezultati. Medicinska i tehnička podloga koja uključuje kliničku pozadinu te osnovne principe rada neuronskih mreža dani su u trećem poglavlju. Četvrto poglavlje uključuje opis teorijskog i programskog dijela razvijenog sustava za segmentaciju lijevog atrija. Peto poglavlje sadrži prikaz dobivenih rezultata. Konačno, zaključak je dan u šestom poglavlju.

# 2. TRENUTNE METODE SEGMENTACIJE LIJEVOG ATRIJA

Segmentacija lijeve pretklijetke iz LGE MRI slika u kliničkoj praksi se obavlja pomoću specijalizirane medicinske programske podrške koja zahtjeva interakciju liječnika. Liječnik mora biti upoznat s načinom rada navedene programske podrške kako bi mogao precizno postaviti različite parametre koji bi omogućili uspješnu segmentaciju lijeve pretklijetke. S obzirom na to da takve, poluautomatske metode segmentacije, zahtijevaju puno vremena te su često podložne greškama uzrokovanim između pristupa dvaju liječnika, došlo je do potrebe razvoja novih, potpuno automatskih algoritama segmentacija.

Razvijene su različite metode za segmentaciju lijeve pretklijetke iz LGE MRI slika. Rješenje za segmentaciju lijevog atrija pomoću 16 – slojne konvolucijske neuronske mreže predloženo u [1] upotrebljava AtriaNet model te bazu podataka od 154 3D LGE – MRI slika srca s atrijskom fibrozom. Razvijena metoda za segmentaciju postiže dice koeficijent od 0.940 i 0.942 za epikard i endokard lijevog atrija. Nadalje, rješenje zasnovano na unaprijed poznatim podacima i značajkama LGE – MRI slika u svrhu pronalaženja središta srce te izrade posteriorne karte vjerojatnosti predloženo je u [2]. Segmentacijom endokarda lijevog atrija upotrebom metode pravog srčanog centra kao ulaza, ostvaren je dice koeficijent od 0.87, dok je upotrebom automatske metode ostvaren dice koeficijent od 0.85. Autori u [3] predložili su vlastitu mrežu koja obuhvaća mrežu za rekonstrukciju oblika SRNN (engl. shape reconstruction neural network) i mrežu prostornih ograničenja SCN (engl. spatial constraint network). Osnovni cilj ove mreže je zadržavanje realnog oblika segmentiranog dijela. Kako bi se to postignuto, formuliran je SCN te je obučen s mrežom za segmentaciju sa strategijom višestrukog učenja. Uz bazu podataka od 45 slika pacijenata, ostvareni je dice koeficijent $0.758 \pm 0.227$. Nadalje, modifikacija U-Net arhitekture kombinacijom SE-Res modula i selektivnog kernela za fazu uzorkovanja prema dolje i za fazu uzorkovanja prema gore predstavljen je u [4]. Upotreba selektivnog kernela poboljšava mapu značajki te je tako ostvarena segmentacija s visokim dice koeficijentom od 0.922 za lijevi ventrikul, dice koeficijent 0.827 za miokard lijevog ventrikula te dice koeficijent 0.874 za desni ventrikul.

Naposljetku je važno spomenuti i pregledni rad predstavljen u [5] kojim je pružen pregled najsuvremenijih tehnika dubokog učenja za segmentaciju dijelova srca s tri najčešće korištena modaliteta kao što su MRI, CT i ultrazvuk. Unutar rada raspravlja se o prednostima trenutno

korištenih metoda segmentacije koje se temelje na dubokom učenju te i o ograničenjima koje onemogućuju širokopojasno korištenje u kliničke svrhe.

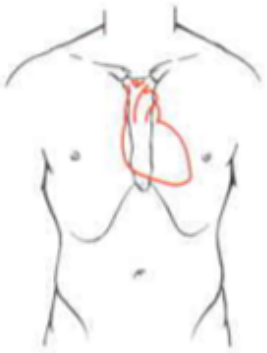
# 3. MEDICINSKA I TEHNOLOŠKA POZADINA

Kardiovaskularni sustav, koji se još naziva krvožilnim sustavom, održava protok krvi kroz tijelo, a sastoji se od srca i krvnih žila. Srce pumpa krv kroz arterije koje su povezane na manje arteriole, a one su povezane na još manje kapilare. Krvne žile prenose nutrijente, elektrolite, otopljene plinove i otpadne supstance između krvi i okolnih tkiva. Kapilare imaju tanku stijenku te su povezane s najmanjim arterijama i venama. Cirkulacija krvi započinje i završava u srcu, kapilare vraćaju krv u venule te nakon toga u veće vene koje vode do srca. Krvožilni sustav se sastoji od dva zatvorena kruga. Venule i vene su dio plućnog sistema jer one prenose krv bez kisika do pluća kako bi primile kisik i ugljični dioksid. Arterije i arteriole su dio sistemskog kruga te prenose krv s kisikom i nutrijentima prema stanicama tijela te u isto vrijeme uklanjaju otpadne supstance. Svim tkivima u organizmu neophodna je cirkulacija kako bi ostala živa¸ prema [6, 7].

## 3.1.Ljudsko srce

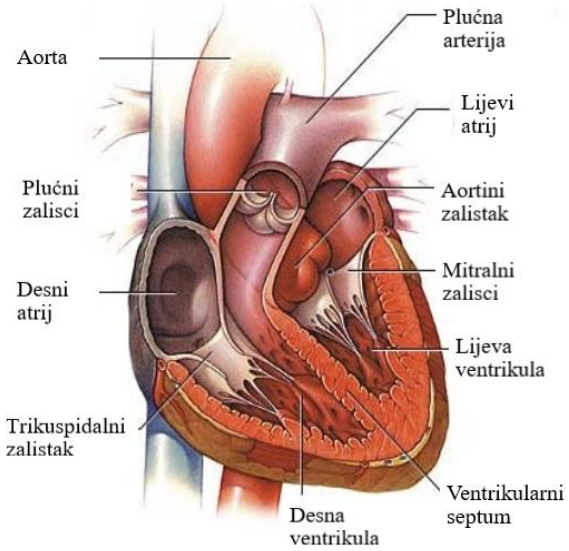
Ljudsko srce je mišićni organ koji je građen od četiri šupljine i srčane stijenke te je smješteno u središtu prsne šupljine, blago usmjereno prema lijevoj strani prsne šupljine. Unutar prsne šupljine, srce je lokalizirano na dijafragmi te se nalazi između plućnih krila te iza prsne kosti kao što je prikazano na slici 3.1. Srce je otprilike je veličine stisnute ljudske šake. Gornje dvije šupljine, atriji, su podijeljene pomoću „zidnog“ tkiva koje se naziva interatrijalni septum. Donje dvije šupljine, ventrikuli, podijeljene su sa sličnom strukturom koja se naziva interventrikularni septum. Lijevi atrij i lijevi ventrikul od desnog atrija i ventrikula odijeljeni su pomoću tanke mišićne pregrade. Lijeva i desna pretklijetka čine atrij, a lijeva i desna klijetka čine ventrikul. Između pretklijetki i klijetki nalaze se srčani zalisci koji omogućavaju protok krvi samo u jednom smjeru, iz pretklijetki u klijetke, odnosno sprječavaju povrat protoka. Položaja srca u ljudskom tijelu prikazan je na slici 3.1.

Srčana stijenka sastoji se od tri sloja, počevši izvana. Prvi sloj je perikard. Nakon njega slijedi srednji sloj koji se naziva miokard te unutarnji sloj - endokard. Perikard se sastoji od vezivnog tkiva i masnog tkiva te štiti srce tako da smanjuje trenje. Miokard je debeli sloj koji je većinom građen od srčanog mišića te sadrži mnoštvo krvnih kapilara, limfnih kapilara te živčanih vlakna. Zaslužan je za pumpanje krvi iz šupljina. Endokard je građen od epiduralnog i vezivnog tkiva s mnoštvom elastičnih i kolagenih vlakana te sadrži krvne žile kao i srčana mišićna vlakna poznata kao Purkinje vlakna.



**Slika 3.1. Položaj srca u ljudskom tijelu [8]**

Protok krvi kroz srce jasno je definiran. Krv teče kroz srce do pluća gdje prima kisik te se vraća nazad prema srcu, a zatim prema ostalim dijelovima odnosno tkivima. Deoksigenirana krv, odnosno krv bez kisika, iz svih tkiva u tijelu ulazi u opušteni desni atrij preko dvije velike vene nazvane gornja šuplja vena (lat. *vena cava superior*) i donja šuplja vena (lat. *vena cava inferiorna*). Desni atrij se steže i krv teče kroz trikuspidalni zalizak u opuštenu desnu ventrikulu. Nakon toga, desni ventrikul se steže i krv teče kroz plućni zalizak u plućnu arteriju kako bi se krv obogatila kisikom odnosno postala oksigenirana. Zatim se lijevi atrij steže te krv teče kroz mitralni zalizak u opušteni lijevi ventrikul. Kada se lijevi ventrikul steže krv se kroz zalizak aorte prenosi aortom, najvećom arterijom u tijelu, do svih dijelova tijela, prema [6, 7]. Slika 3.2. prikazuje anatomski presjek srca s njegovim osnovnim dijelovima.




**Slika 3.2. Prikaz srca [9]**

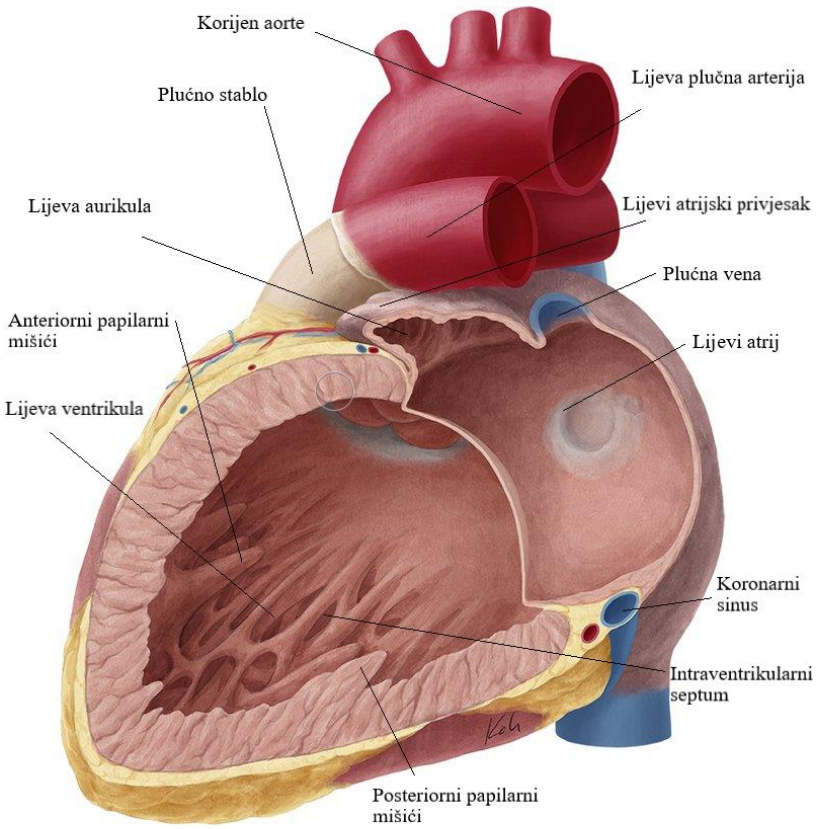
## 3.2. Struktura lijevog atrija

Lijevi atrij je kubičnog oblika te je smješten u baznom dijelu srca, malo iza i iznad desnog atrija. Iako je manji s obzirom na količinu krvi koju može zadržati, lijevi atrij ima deblji zid miokarda u usporedbi s desnim atrijem. To je rezultat činjenice da je lijevi atrij izložen većim pritiscima i stoga obavlja više posla nego desni atrij. Isto tako, lijevi atrij predstavlja najposjećeniju komoru srca. Kao i desni atrij, lijevi atrijski zid sastoji se od venskih entiteta, osim venskih entiteta sadrži i aurikulu te vestibularne dijelove. Četiri plućne vene ulaze u stražnji dio lijevog atrija. Vene probijaju obje strane stražnje stijenke, sa svake strane po dvije, što predstavlja većinu anatomske baze srca.

Većina prednje površine lijevog atrija nalazi se iza korijena velikih žila. Dio poprečnog perikardijalnog sinusa (prostor između gornje šuplje vene i velike arterije) prolazi ispred lijevog atrija. Lijevi atrij također ima aurikulu koja je izduženija i zakrivljenija u odnosu na onu u desnom atriju te djelomično prekriva dio plućne arterije. Jedini dio lijevog atrija koji je vidljiv kada se srce promatra s prednje strane je vrh lijevog atrijalnog dodatka (LAA).

Lijevi atrij zatvoren je gornjim, stražnjim, lijevim bočnim, septalnim (ili medijalnim) i prednjim zidom. Promatrajući atrij u cjelini, u njegovoj točnoj orijentaciji, može se reći da je usmjeren prema desnom atriju. U anatomskom položaju lijevi atrij nalazi se između 5. i 8. torakalnog kralješka ako je osoba u ležećem položaju, a u slučaju da je osoba u stojećem položaju nalazi se između 6. i 9. kralješka. Posteriorno povezani s lijevim atrijem su silazna aorta, jednjak i prethodno spomenute vene [6, 10]. Lijevi atrij je manji od desnog atrija, ali je njegova stijenka deblja. Ona se sastoji se od dva osnovna dijela: glavne šupljine odnosno venskog aspekta i auricular, kao što je prikazano na slici 3.3. .

Glavna šupljina lijevog atrija je kubičnog oblika i glatke unutrašnje površine, a nalazi se sprijeda zajedno s plućnom arterijom i aortom. S prednje i s desne strane odvojena je od desnog atrija sa srčanom strukturom koja se naziva atrijski septum. Unutar lijevog atrija s obje strane se nalaze dvije plućne vene. Aurikula je suženija na spoju s glavnom šupljinom lijevog atrija. Duža, uža i zakrivljenija je u odnosu na aurikulu na desnoj strani i njezini su rubovi dublje razvedeni. Usmjerena je naprijed i prema lijevo te prekriva korijen plućne arterije, prema [6].




**Slika 3.3. Prikaz lijevog atrija i lijevog ventrikula [11]**

Unutrašnjost lijevog atrija sadrži sljedeće dijelove:

- Plućne vene - ima ih četiri, otvaraju se u gornjem dijelu stražnje površine lijevog atrija, dvije s obje strane na njegovoj središnjoj liniji, ne sadrže zaliske. Dvije lijeve vene često se završavaju sa zajedničkim otvorom
- Lijevi atrioventrikularni otvor - predstavlja otvor između lijevog atrija i ventrikule i prilično je manji od odgovarajućeg otvora na desnoj strani
- Perinealni mišić – koji je manji od onoga u desnoj aurikuli te je ograničen unutrašnjom površinom aurikule
- Atrijski septum – na kojemu se može vidjeti lunarni utisak, dolje omeđen polumjesečnim grebenom čija je konkavnost okrenuta prema gore

## 3.3.Dobivanje slika pomoću magnetske rezonance

Magnetska rezonanca je neinvazivna tehnologija za dobivanje medicinskih slika. Dobivena slika je trodimenzionalna anatomska slika. Često se koristi za otkrivanje, dijagnozu i praćenje bolesti. Temelji se na sofisticiranoj tehnologiji koja pobuđuje i detektira promjenu smjera rotacijske osi protona koji se nalaze u tekućini koja održava tkiva živima.

MRI koristi snažne magnete koji proizvode magnetsko polje kojima se prisiljava protone u tijelu da se poravnaju s tim poljem. Kada radio-frekvencijska struja pulsira kroz pacijenta, protoni se stimuliraju i izbacuju iz ravnoteže te se naprežu protiv povlačenja magnetskog polja. Kada je radio–frekvencijsko polje isključeno, MRI senzori su u mogućnosti detektirati energiju koja se oslobađa kada se protoni poravnaju s magnetskim poljem. Vrijeme potrebno da se protoni izjednače s magnetskim poljem, kao i oslobođena energija, mijenja se ovisno o okruženju i kemijskoj prirodi molekula. Liječnici su u stanju utvrditi razliku između različitih vrsta tkiva na temelju tih magnetskih svojstava.

Kako bi se dobila MRI slika, pacijent ulazi unutar velikog magneta te mora ostati miran kako bi se dobila jasna slika, odnosno kako se slika ne bi zamutila. Kontrastna sredstva (koja često sadrže element Gadolinij) mogu se davati pacijentu intravenski za vrijeme ili prije MRI kako bi se povećala brzina kojom se protoni poravnavaju s magnetskim poljem. Što se protoni brže poravnaju, slika je svjetlija.

MRI skeniranje je posebno prikladno za snimanje koštanih dijelova kao i mekih tkiva. Razlikuje se od računalne tomografije (CT) po tome što ne koristi štetno ionizirajuće zračenje X–zraka. Mozak, leđna moždina i živci, kao i mišići, ligamenti i tetive jasnije su vidljivi s MRI nego s rendgenom i CT–om. Upravo zbog toga se često primjenjuje prilikom snimanja ozljeda koljena i ramena.

MRI u mozgu može razlikovati bijelu tvar od sive tvari i može se koristiti za dijagnosticiranje aneurizmi i tumora. Budući da MRI ne koristi rendgenske zrake ili neko drugo zračenje, najbolji je izbor kod učestalih snimanja koja su potrebna za određivanje dijagnoze ili terapije, posebnu za mozak. Međutim, MRI je skuplji od X-zračenja ili CT skeniranja, prema [12].

## 3.4.Formati zapisa medicinskih slika

Formati slikovnih datoteka pružaju standardizirani način za pohranu podataka, koji opisuju sliku, u računalnu datoteku. Baza podataka medicinskih slika najčešće se sastoji od jedne ili više slika koje predstavljaju projekciju anatomskog volumena na slikovnoj ravnini (projekcija ili planarno snimanje), niz slika od kojih svaka predstavlja tanki sloj volumena (tomografska ili višeslojna dvodimenzionalna slika), setovi podataka za volumen (volumen ili trodimenzionalna slika) ili višebrojna akvizicija istog tomografskog snimka, trodimenzionalne slike tijekom vremena za dobivanje dinamičnog niza akvizicije (četverodimenzionalno snimanje). Format datoteke opisuje kako su slikovni podatci organizirani unutar slikovne datoteke i kako pikseli trebaju biti interpretirani pomoću software za uspješno učitavanje te vizualizaciju.

Medicinska slika predstavlja unutarnju strukturu ili funkciju neke anatomske regije u obliku polja elemenata slike odnosno piksela ili voksela. To je diskretni prikaz koji je rezultat procesa uzorkovanja odnosno rekonstrukcije koji mapira numeričke vrijednosti na položajima u prostoru. Broj piksela koji se koristi za prikaz polja određenog modaliteta akvizicije predstavlja koliko detaljno se može prikazati anatomija ili funkcija. Informacije koje numerička vrijednost piksela predstavlja uvelike ovise o modalitetu slike, protokolu akvizicije, rekonstrukciji i na kraju naknadnoj obradi.

Dubina piksela je broj bitova koji se koriste za kodiranje informacija svakog piksela. Svaka se slika sprema u datoteku i čuva u memoriji kao grupa bajtova. Bajtovi su skupina od osam bita i predstavljaju najmanju količinu koja se može pohraniti u memoriju računala. To znači da ako slika veličine 256 do 256 piksela ima dubinu piksela od 12 do 16 bita, računalo će uvijek pohraniti dva bajta po pikselu i tada će za podatke piksela trebati $256 \times 256 \times 2 = 131072$ bajta memorije u oba slučaja. S dubinom piksela od 2 bajta po pikselu, moguće je kodirati i pohraniti cjelobrojne brojeve između 0 i 65,535, moguće je prikazati brojeve između -32,768 i +32,767 koristeći 15 bitova za predstavljanje broja i jedan za predstavljanje znaka. Podatci o slici također mogu biti realni brojevi. Institut inženjera elektrotehnike i elektronika stvorio je standard (IEEE - 754) u kojem definira dva osnovna formata za kodiranje u binarnim brojevima s pomoćnim zarezom: 32 – bitna preciznost i 64 – bitna dvostruka preciznost. Standard se bavi problemom preciznosti, sekvencom duljine n – bita (2n – 1) dobiva se konačni broj kombinacija koje trebaju predstavljati kontinuirani raspon realnih brojeva. Pikseli također mogu sadržavati kompleksne brojeve. Kompleksni broj ima realnu i

imaginarnu komponentu. kompleksni brojevi obično imaju dubinu dva puta veću od realnih brojeva. Dubina piksela predstavlja broj bita koji je potrebno pohraniti u memoriju kako bi se sačuvala informacija koju želimo pohraniti u piksel.

Fotometrijska interpretacija određuje kako se pikseli moraju interpretirati za ispravan prikaz slike kao monokromatske ili slike u boji. Da bi se odredilo jesu li informacije o boji spremljene ili nisu spremljene u vrijednostima piksela slike, uvodi se koncept uzorka po pikselu (poznato kao broj kanala). Monokromatske slike imaju jedan uzorak po pikselu te nema podataka o boji pohranjenih na slici. Nijanse sive od crne do bijele koriste se za prikaz slike. Broj nijansi sive ovisi o broju bitova koji se koristi za pohranu što u ovom slučaju korespondira dubini piksela, odnosno veća dubina piksela veći broj nijansi sive. Kliničke radiološke slike, kao što su rendgenske računalne slike odnosno CT i slike magnetske rezonance, MR, imaju fotometrijsku interpretaciju sive boje. Slike nuklearne medicine, poput pozitronske emisione tomografije PET (engl. *positron emission tomography*) i jednostruke fotonske emisione tomografije SPECT (engl. *single-photon emission computed tomography*), obično se prikazuju mapom boja ili paletom boja. Svaki piksel slike povezan je s bojom u unaprijed određenoj mapi boja. Boja se odnosi samo na prikaz, informacija je povezana, a zapravo nije pohranjena u vrijednostima piksela. Na slikama je jedan piksel za uzorkovanje u pseudo boji. Za kodiranje informacija o boji piksela potrebno je više uzoraka po pikselu, modela boja definira kako se određuje boja kombiniranjem uzorka. Obično se koristi 8-bita za svaki uzorak ili komponentu boje. Dubina piksela izračunava se množenjem dubine uzorka (broja bita korištenih za svaki uzorak) s brojem uzoraka po pikselu. Ultrazvučne slike se obično pohranjuju korištenjem crveno-zeleno-plave boje RGB (engl. *red-blue-green*). Za taj slučaj piksel bi trebao biti prikazan kao kombinacija triju osnovnih boja te su pohranjena tri uzorka po pikselu. Slike će imati dubinu piksela od 24 bita. Na primjer, boja se koristi za kodiranje smjera i brzine protoka krvi u Dopplerovom ultrazvuku za prikaz dodatnih funkcionalnih informacija na anatomskoj slici sive skale pomoću obojenih slojeva, za istovremeno prikazivanje funkcionalnih i anatomskih slika kao u PET / CT ili PET / MRI, a ponekad umjesto sivih tonova kako bi se istakle razlike u signalima.

Meta podaci su informacije koje opisuju sliku. Za bilo koji format datoteke, informacije koje daju podatke o slici nalaze se neposredno prije podataka same slike, odnosno podataka o pikselima. Ove informacije nazvane meta podaci obično se pohranjuju na početku datoteke kao zaglavlje i sadrže barem dimenzije matrice slika, prostornu rezoluciju, dubinu piksela i fotometrijsku interpretaciju.

Zahvaljujući meta podacima, softverska aplikacija može prepoznati i ispravno otvoriti sliku u podržanom formatu datoteke. U slučaju medicinskih slika, meta podaci imaju širu ulogu zbog prirode samih slika. Slike dobivene iz dijagnostičkih modaliteta obično sadrže informacije o tome kako je slika nastala. Na primjer, slika magnetske rezonancije imat će parametre koji se odnose na upotrijebljeni slijed impulsa, informacije o vremenu, *flip* kutu, broju akvizicija. Slika nuklearne medicine poput PET slike sadrži podatke o ubrizganom radio – farmaceutskom lijeku i težini pacijenta. Ovi podaci omogućuju softveru poput OsiriX *on-the-fly* pretvorbu vrijednosti piksela u standardizirane vrijednosti SUV (engl. *Standardized uptake value*) bez potrebe za stvarnim upisivanjem SUV vrijednosti u datoteku. Formati datoteka nakon obrade imaju odjeljak s meta podacima koji u osnovi opisuje podatke piksela. Različiti sadržaj meta podataka glavna je razlika između slika dobivenih dijagnostičkim modalitetom i naknadno obrađenih slika. Meta podaci su moćan alat za obilježavanje i iskorištavanje podataka povezanih sa slikama u kliničke i istraživačke svrhe te za organiziranje i dohvaćanje slika u arhivima.

Podatci o pikselima su numeričke vrijednosti koje se pohranjuju. Prema tipu podataka, podatci o pikselima mogu se pohranjivati kao cjelobrojni ili brojevi s pomičnim zarezom tako da se koristi što manje bita kako bi se ta vrijednost pohranila, odnosno kako bi se ta vrijednost predstavila. Ako promatramo slike koje su generirane računalnom tomografijom te proslijeđene u arhivu slika i komunikacijski sistem PACS (engl. picture archiving and communication system), radiološke slike poput CT-a i MR-a kao i modernih nuklearnih medicinskih modela, kao što su PET i SPECT, pohranjuju 16 bita za svaki piksel kao cjelobrojnu vrijednost. Iako su cjelobrojni brojevi prikladni za sliku, brojevi s pomičnim zarezom najčešće se koriste za bilo koju vrstu daljnje obrade te predstavljaju najbolji način za adresiranje izračuna. Podatci slike također mogu biti kompleksne vrijednosti, ali to nije uobičajeno te se može izbjeći tako da se imaginarni i realni dio spremaju zasebno kao dvije slike. Primjer korištenja kompleksnih vrijednosti je kod MRI gdje se u polja pohranjuju prikupljeni podatci prije rekonstrukcije, takozvani K – prostor, ili nakon rekonstrukcije ako je potrebno spremiti i magnitudu i fazu slike.

U slučaju da se vrijednost piksela pohrani pomoću dva ili više bajtova, treba uzeti u obzir da redoslijed u kojem računalo pohranjuje bajtove nije jedinstven, ako su b1 i b2 dva bajta 16 – bitne riječi, računalo tu riječ može spremiti kao (b1 : b2) ili (b2 : b1). Izraz *little endian* inicira na to da je najmanje značajan bit spremljen prvi, dok *big endian* znači da je najznačajniji bit spremljen prvi. Taj

problem je najčešće povezan uz procesor na kojem se temelji sklopovlje računala i odnosi se na sve kodirane podatke koje zauzimaju više od 8 bita po pikselu. Za formate koji imaju fiksnu veličinu zaglavlja, podatci o pikselima počinju na fiksnoj poziciji nakon zaglavlja. U slučaju da je zaglavlje varijabilne duljine, početak podataka o pikselima označava se tag – om ili pokazivačem. Izračun veličine podataka (VP) o pikselima može se opisati sljedećim izrazom:

$$\text{VP} = \text{redci} \times \text{stupci} \times \text{dubina piksela} \times (\text{broj okvira}) \tag{3-1}$$

pri čemu je dubina piksela izražena u bajtovima. Isto tako, veličina slikovne datoteke (VSD) može se opisati pomoću sljedećeg izraza:

$$\text{VSD} = \text{veličina zaglavlja} + \text{veličina podataka o pikselima} \tag{3-2}$$

Oba izraza vrijede za nekomprimirane podatke. Slikovni podaci mogu se komprimirati te se tako umanjuju zahtjevi za pohranu i prijenos. U tom slučaju veličina datoteke se smanjuje za faktor koji ovisi o odabranoj tehnici kompresije. Općenito govoreći, kompresija može biti reverzibilna, bez gubitaka, ili nepovratna, s gubitcima. Tehnike kompresije bez gubitaka omogućuju umjereno povećanje u smislu pohrane slike. Mala kompresija, odnosno veličina podataka smanjuje se za manji faktor u odnosu na tehnike s gubitcima. Tehnike s gubitcima imaju veću kompresiju, ali dovode do gubitka informacija. Zbog toga se ne preporuča njihova upotreba za obradu medicinskih slika.

Formati medicinskih slika mogu se podijeliti u dvije kategorije. Prvi su formati namijenjeni standardiziranju slika generiranih dijagnostičkim modalitetima kao što je DICOM . Drugi su formati napravljeni s ciljem da se olakša i poboljša obrada i analiza slike, neki od njih su Analyse , Nifti i Minc. Datoteke medicinskih slika obično se pohranjuju koristeći jednu od dvije moguće konfiguracije. Prva mogućnost je da jedna datoteka sadrži i meta podatke i slikovne podatke, pri čemu su meta podaci pohranjeni na početku datoteke. Ovu konfiguraciju koriste DICOM, Minc i Nifti formati datoteka. Druga konfiguracija pohranjuje meta podatke u jednu datoteku, a slikovne podatke u drugu datoteku. Format datoteke Analyze koristi dvije datoteke - „.hdr“ i „.img“, prema [13].

## 3.5.Raw, mhd i NIFTI zapis medicinskih slika

Format datoteke RAW je digitalna fotografija koja sadrži netaknute, neobrađene podatke o pikselima izravno sa senzora digitalne kamere. Format RAW još nije podvrgnut demozaiku, što znači da sadrži samo jednu crvenu, zelenu ili plavu vrijednost na svakoj lokaciji piksela. Demozaik algoritam je proces koji se koristi nad digitalnom slikom za rekonstrukciju slike u boji pomoću nepotpunih uzoraka boje sa senzora slike prekrivenih matricom filtera u boji (CFA). Poznat je i kao CFA interpolacija ili rekonstrukcija boje, prema [14].

*MetaImage* (MHD) je format medicinskih slika temeljen na tekstualnim datotekama. Takve datoteke posjeduju ITK *Meta Image* zaglavlje. ITK je alat koji pruža uvid u segmentaciju i registraciju, predstavlja platformski sustav otvorenog koda koji pruža programerima opsežan paket programskih alata za analizu slike.

Nifti (engl. n*euroimaging informatics technology initiative*) je format datoteke koji je početkom 2000-ih definirao odbor sa sjedištem u Nacionalnom zavodu za zdravstvo s namjerom da napravi format za obradu specifično slika mozga (engl. *neuroimaging*), zadržavajući prednosti formata Analyse, ali i rješavajući njegove slabosti. Često se koristi za pohranu slikovnih podataka mozga nastalih korištenjem metode magnetske rezonancije. Nifti se zapravo može promatrati kao izmijenjeni format Analyze. Format ispunjava neka neiskorištena odnosno malo korištena polja prisutna u zaglavlju Analyze 7.5 za pohranjivanje novih informacija poput orijentacije slike. Nifti uključuje vrstu podataka koji se ne koristi u formatu Analyze poput neoznačenog 16-bitnog zapisa. Iako format također omogućuje pohranjivanje podataka zaglavlja i piksela u zasebne datoteke, slike se obično spremaju kao jedna „.nii“ datoteka u kojoj se spajaju zaglavlje i podaci o pikselima. Zaglavlje ima veličinu od 348 bajtova, za slučaj da se podatci pohranjuju u obliku „.hdr“ i „.img“, ako se pohranjuju kao jedna „.nii“ datoteka zaglavlje je 352 bajta. Kako bi veličina zaglavlja bila višekratnik broja 16, kod „.nii“ datoteke na kraju se nalaze još 4 bajta koja omogućuju pohranjivanje dodatnih meta podataka.

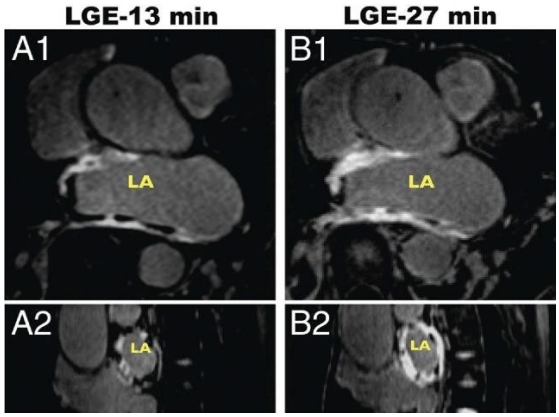
Nifti format podržava dva načina pohranjivanja orijentacije volumena na slici u prostoru. Prvi, koji uključuje rotaciju i translaciju, koristi se za mapiranje koordinata voksela u referentni okvir skenera. Ova transformacija "krutih tijela" kodirana je pomoću „kvateriona“. Druga metoda se koristi za spremanje 12 parametara linearne transformacije koji definiraju poravnanje volumena sa standardnim koordinatnim sustavom. Prostorna normalizacija uobičajena je u analizi funkcionalne

slike mozga. Nifti format podržavaju mnogi software poput 3D Slicer , ImageJ i OsiriX, kao i neki noviji poput R i Nibabel. Ažurirana inačica standarda, Nifti-2, razvijena je za upravljanje većim skupom podataka te je definirana u 2011. godini. Nova verzija kodira sve dimenzije slikovne matrice sa 64-bitnim cijelim brojem, umjesto 16-bitnim kao u prethodnoj verziji Nifti-1, što ograničava ograničenje veličine od 32,767. Ažurirana verzija ima gotovo sve karakteristike Nifti-1, ali sadrži rezervna polja zaglavlja za podatke dvostruke preciznosti te je stoga zaglavlje 544 bajta, prema [13].

## 3.6.Prikaz lijeve pretklijetke na LGE MRI slikama

Kasno, postkontrastno snimanje slika magnetne rezonance, LGE MRI je utvrđena tehnologija za slikovni prikaz fibroze miokarda i lijevog ventrikularnog ožiljka. LGE MRI je pokazao potencijal za detektiranje srčanih ožiljaka u drugim srčanim strukturama, uključujući lijevi atrij i papilarne mišiće, odnosno mišiće koje se nalaze u komorama srca. Kod LGE MRI-a se kontrastno sredstvo ubrizgava 10 do 15 minuta prije snimanja slike. U LGE MRI slikovnoj sekvenci koristi se inverzni impuls koji slijedi nakon slikovne akvizicije nakon inverznog kašnjenja, te je u tom trenutku signal miokarda jednak nuli. Inverzno kašnjenje se određuje prije snimanja LGE slike koristeći pulsno sekvencioniranje (engl. *look-locker*). Brzi oporavak od signala ožiljka (kratko $T_1$) dovodi do toga da se ožiljak jasno vidi na LGE slikama. Međutim, koristeći kratki $T_1$ za krv, krv se također jasno prikazuje i tu dolazi do malog kontrasta između ožiljaka i krvi. Stoga detekcija ožiljka miokarda u blizini krvi predstavlja veliki izazov, prema [15].

Kontrastna sredstva koja se temelje na gadoliniju koriste se u trećini svih pretraga pomoću magnetske rezonance kako bi se povećala jasnoća slika unutarnjih struktura pacijenta uključujući srce. Tako se poboljšava vidljivost učestalih bolesti kao što su fibroza ili ožiljci kao i različite upale i tumori. LGE MRI široko se koristi za proučavanje opsega i distribucije srčane fibroze i ožiljaka. Kliničke studije na pacijentima s atrijskom fibrozom sugeriraju da LGE MRI predstavljaju dobar modalitet za pronalaženje informacija poput predikcije opsega ili raspodjelu atrijske fibroze. Promjer i volumen lijevog atrija izračunati iz 3D LGE-MRI pružaju pouzdane podatke za kliničku dijagnozu i stratifikaciju liječenja. Primjer prikaza lijevog atrija na slici LGE magnetske rezonance je slika 3.4..




**Slika 3.4. Primjer prikaza lijevog atrija na slici LGE MRI [16]**

## 3.7.Neuronske mreže

Iako je duboko učenje prilično staro potpolje strojnog učenja, svoj procvat doživjelo je početkom 2010. godine. Od tada, u nekoliko godina ostvarilo je revoluciju u tom području, s izvanrednim rezultatima o perceptivnim problemima kao što su vid i sluh, odnosno problemima koji uključuju vještine koje se čine prirodnim i intuitivnim za ljude, ali su bile nedostižne za strojeve.

Strojno i duboko učenje postigli su značajan razvoj posebice u sljedećim područjima i primjenama:

- Klasifikacija slika na razini koja je blizu ljudske
- Prepoznavanje govora na razini koja je blizu ljudske
- Transkripcija rukopisa na razini koja je blizu ljudske
- Poboljšanje prijevoda pomoću računala
- Poboljšana pretvorba teksta u govor
- Digitalni asistenti kao što su *Google Now* i *Amazon Alexa*
- Autonomna vožnja na razini koja je blizu ljudske
- Poboljšano targetiranje reklama koje koriste Google, Baidu i Bing
- Poboljšani rezultati pretraživanja na web–u
- Mogućnost odgovaranja na pitanja na prirodnom jeziku

Klasifikacija slika primjenjuje se u različitim područjima te je od velikog značaja na području medicine. Predstavlja primarnu domenu u kojoj mreže dubokog učenja imaju najvažniju ulogu u medicinskoj analizi slike. Klasifikacija slike prihvaća dane ulazne slike i vrši klasifikaciju izlaza u

svrhu prepoznavanja postoji li određena bolest ili ne. Metode konvolucijskih neuronskih mreža daju rezultate visoke točnosti, prema [17].

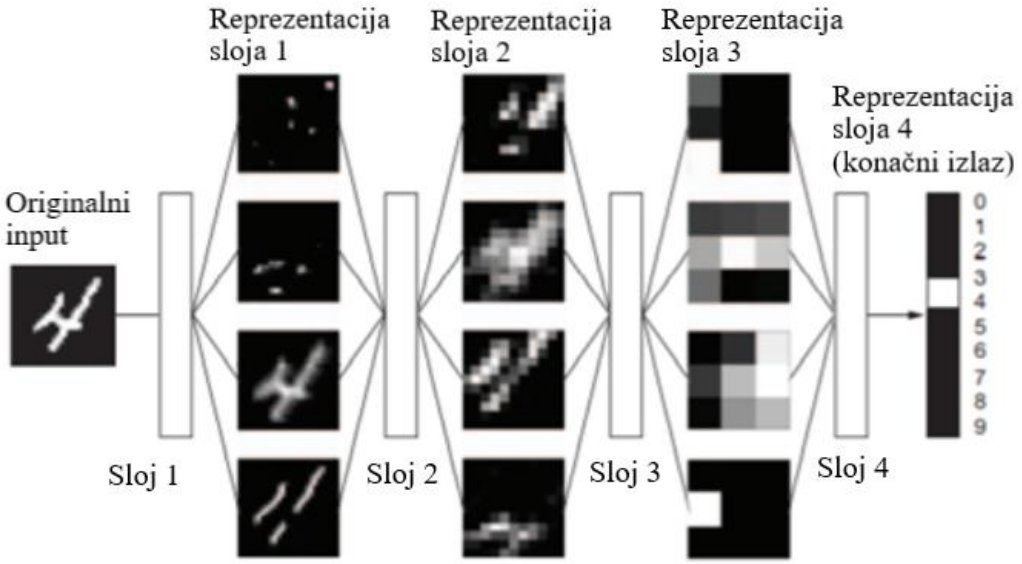
Još uvijek se istražuje puni opseg onoga što duboko učenje može učiniti. Počinje se primjenjivati i izvan okvira strojne percepcije i razumijevanja prirodnog jezika na jako širokom i raznolikom području kao što je formalno obrazloženje (engl. *formal reasoning*). U slučaju da se ostvari napredak u tom smjeru, odnosno ako istraživanja budu uspješna, doći će razdoblje u kojem će duboko učenje pomagati ljudima u znanosti, razvoju softvera i još mnogo toga.

Matematički gledano, neuronske mreže su modeli za pohranu informacija koji koriste algoritme učenja inspirirane načinom rada ljudskog mozga. Budući da se neuronske mreže koriste u strojevima one se kolektivno nazivaju „umjetnim neuronskim mrežama“. Danas se pojam strojno učenje često koristi u tom području te predstavlja znanstvenu disciplinu koja se bazira na dizajnu i razvoju algoritama koji omogućuju računalima da uče na temelju podataka, podatci su sa senzora ili baza podataka. Glavni fokus istraživanja strojnog učenja je automatski naučiti prepoznati složene obrasce i donositi inteligentne odluke na temelju podataka. Dakle, strojno učenje je usko povezano s poljima kao što su statistika, rudarenje podataka, prepoznavanje uzoraka i umjetna inteligencija. Neuronske mreže su popularan način izvođenje strojnog učenja te spadaju u potkategoriju dubokog učenja, ali postoje mnoge druge metode strojnog učenja, kao što su logistička regresija i podrška vektorskih strojeva.

Duboko učenje (engl. *deep learning*) je specifično potpodručje strojnog učenja, a predstavlja novi način učenja iz podataka koji se odnosi na učenje uzastopnih slojeva sve sadržajnijih prikaza. Prva riječ naziva, odnosno duboko, odnosi se na uzastopne slojeve odnosno koliko slojeva doprinosi modelu podataka te se naziva dubina modela (dubina je određena brojem slojeva). Moderna verzija dubokog učenja uključuje desetke ili čak stotine uzastopnih slojeva prikaza, svi su slojevi naučeni automatski iz podataka za učenje (engl. *training data*).

Kod dubokog učenja, slojevita prezentacija sadržaja, gotovo uvijek, se uči pomoću modela koje nazivano neuronskim mrežama, strukturiranog tako da slojevi idu jedan povrh drugog. Izgled prikaza sadržaja naučenog pomoću algoritma dubokog učenja na primjeru sa znamenkama prikazano je na slici 3.5. Mreža transformira znakove u prezentaciju sadržaja koja se značajno razlikuje od originalne slike i povećavajući informacije o konačnom rezultatu. Mrežu možemo

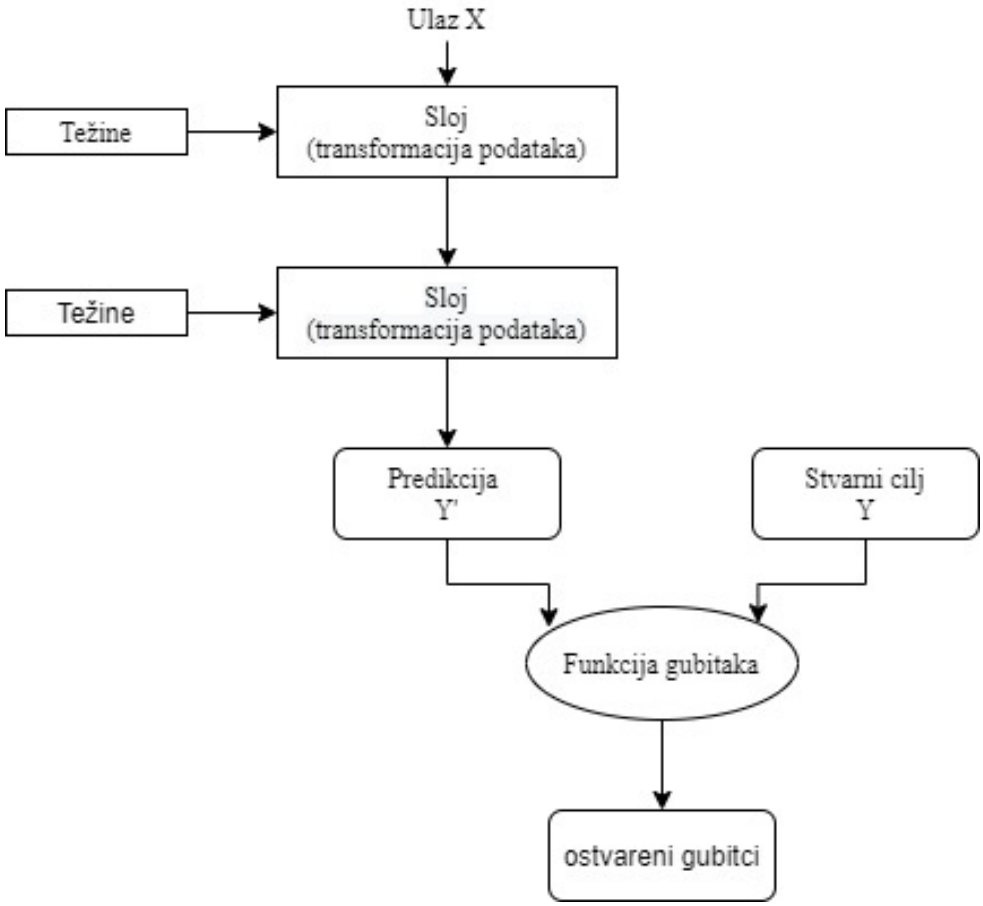
promatrati kao operaciju koja filtrira podatke pri čemu informacije prolaze kroz uzastopne filtre te iz filtra izlaze sve pročišćenije.




**Slika 3.5. Prezentacija sadržaja modela naučenog za klasifikaciju znamenki [18]**

Specifikacija onoga što slojevi mreže izvršavaju nad ulaznim podatcima sačuvano je u težinama (engl. *weights*) slojeva, koja je predstavljena nizom brojeva. Transformacija implementirana određenim slojem je parametrizirana njegovom težinom. Dakle, učenje predstavlja pronalaženje odgovarajućeg seta vrijednosti za težinu svih slojeva mreže tako da mreža ispravno mapira primjere ulaznih podataka s njihovim povezanim ciljem.

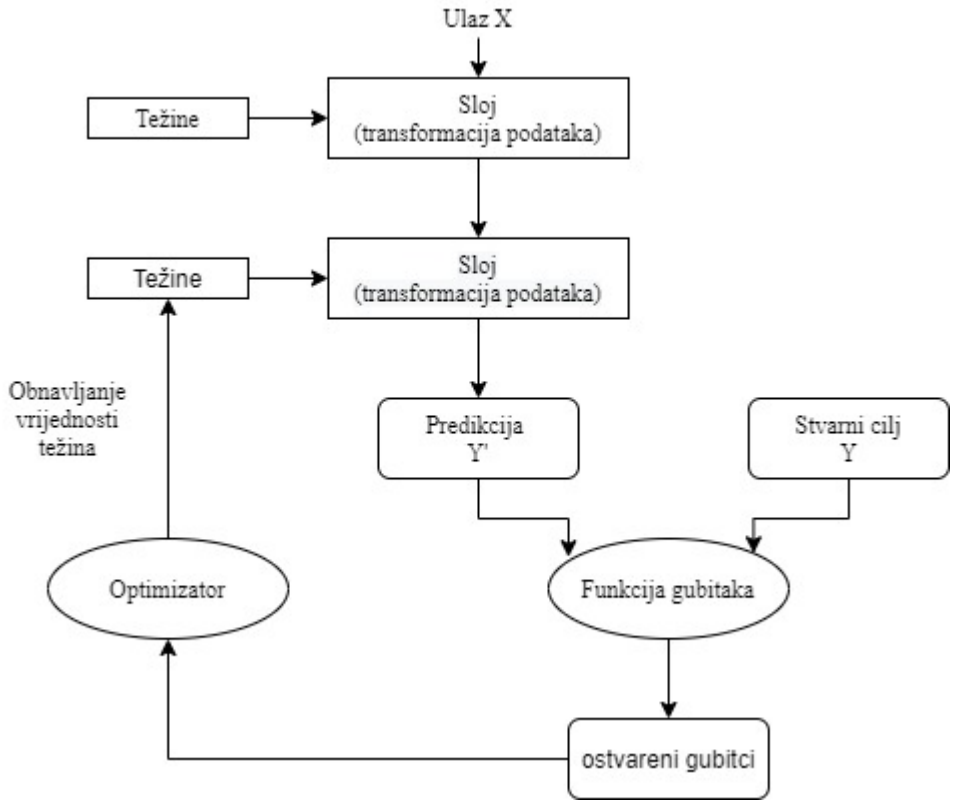
Kako bi se upravljalo mrežom potrebno ju je prethodno promatrati. Kako bi se kontrolirali izlazni podatci neuronske mreže potrebno je izmjeriti koliko su izlazni podatci daleko od onoga što se očekuje. To se ostvaruje pomoću funkcije koja se naziva funkcija gubitka (engl. *loss function)* mreže ili ciljna funkcija (engl. *objective function)*. Funkcija gubitka uspoređuje predviđanja mreže i stvarnog cilja odnosno onoga što želimo da bude povratna vrijednost mreže te izračunava razliku. Dijagram toka funkcije za provjeru kvalitete izlaza iz mreže prikazan je na slici 3.6.




**Slika 3.6. Funkcija gubitka za mjerenje kvalitete izlaza iz mreže [18]**

Temeljna prednost dubokog učenja je korištenje rezultata funkcije gubitaka kao povratnog signala za prilagođavanje vrijednosti težina tako da se smanji rezultat gubitka. Za ovaj dio posla zadužen je optimizator koji provodi algoritam povratne propagacije (engl. *backpropagation)* te ujedno predstavlja središnji algoritam dubokog učenja.

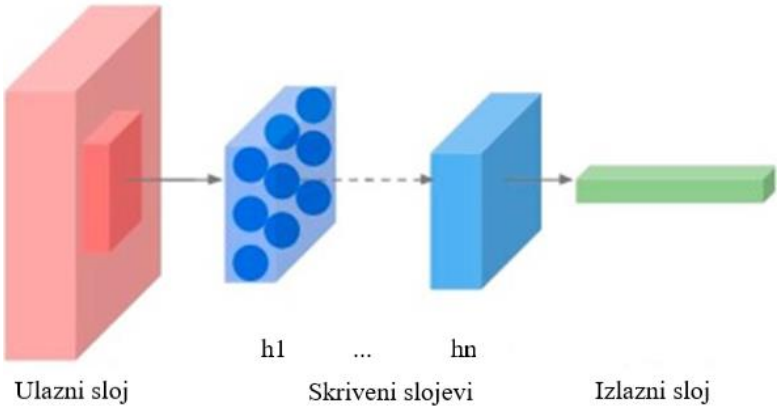
U početnom koraku težinama mreže dodijeljene su nasumične vrijednosti te u tom slučaju mreža provodi samo niz nasumičnih transformacija. Dobiveni izlaz, nakon prvog koraka, daleko je od onoga što bi idealno trebalo biti. Zbog toga je i rezultat funkcije gubitka također velik. Svakim novim korakom učenja mreže, težine se prilagođavaju u ispravnom smjeru, isto tako izlaz iz funkcije gubitka, se smanjuje. Taj proces nazivamo petljom treninga koja se ponavlja dovoljan broj puta, najčešće desetke iteracija na preko tisuću primjera, mijenjajući vrijednosti težina te minimizirajući funkciju gubitaka (Slika 3.7.). Mreža s minimalnim gubitcima je ona za koju su izlazi najsličniji očekivanim izlazima te se takva mreža naziva naučena mreža, prema [18].




**Slika 3.7. Prikaz toka nakon dobivenog rezultata funkcije gubitka [18]**

## 3.8.Konvolucijske neuronske mreže

Konvolucijska neuronska mreža sastoji se od blokova koji mogu biti primijenjeni kako u prostoru tako i u vremenu, odnosno mogu se koristiti za slikovne, audio i video signale. Svaki blok konvolucijske neuronske mreže transformira ulazni volumen u izlazni volumen aktivacijskog neurona koji će biti ulaz u sljedeći blok. Za razliku od klasičnih umjetnih neuronskih mreža, blokovi neurona u konvolucijskoj neuronskoj mreži se ponašaju kao jezgre koje su povezane i primijenjene preko jednog manjeg dijela (engl. *patch)* ulaznog volumena, što je prikazano na slici 3.8. Neuroni bloka nisu u potpunosti povezani sa svim neuronima prethodnog sloja kao što je to u standardnom višeslojnom perceptronu (engl. *multilayer perceptron*) kao što je prikazano na slici 3.9.. Ti se blokovi zapravo sastoje od faza ekstrakcije koje se posebno sastoje od tri sloja koji su ključni dijelovi gotovo svih konvolucijskih mreža, prema [19]. Arhitektura osnovne konvolucijske mreže prikazana je na slici 3.8. te prikazuje osnovni perceptron koji se sastoji od ulaznog, skrivenog i izlaznog sloja.




**Slika 3.8. Arhitektura konvolucijske neuronske mreže [19]**

Svaki blok ili sloj konvolucijske neuronske mreže transformira ulazni volumen u izlazni volumen aktivacijskih neurona. Neuroni u sloju *l* su povezani s malim područjem sloja *l-1* (engl. *patch*).

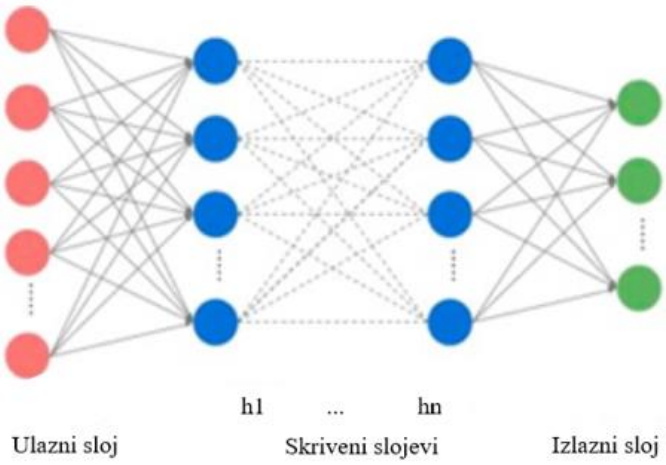
Kod višeslojnog perceptrona (engl. *multilayer perceptron*, MLP) svi čvorovi jednog sloja su povezani sa svim čvorovima prethodnog sloja. Tri ključna sloja od kojih je građena konvolucijska neuronska mreža su, prema [19, 20]:

- **Konvolucijski sloj** – predstavlja sloj gdje svaki neuron izračunava produkt točke između njegove težine i malog područja ulaznog volumena, pravokutnog dijela prethodnog sloja (engl. *patch*), na koji je povezan. Konvolucijski sloj za cilj ima prepoznavanje određenih značajki prethodnog sloja i njihovo preslikavanje u mapu značajki. Ovaj sloj možemo promatrati kao skup $k$ filtera veličine $l \times l \times q$ gdje neuroni imaju iste težine i povezuju ulazni volumen s izlaznim volumenom. Svaki filtar otkriva određene značajke na svakoj od ulaznih lokacija. Rezultirajući izlazni volumen sloja $l$ je karakteristična mapa značajki veličine $d^l \times d^l \times k^l$ koja pohranjuje informacije o tome gdje se pojavljuju značajke u originalnom ulaznom volumenu i izračunava se kao $z_i^l = B^l \sum_{j=1}^{k^{l-1}} W_{i,j}^l * z_j^{l-1}$, gdje $i \in [1, k^l], B^l$ je matrica sloja $l$ i $W_{i,j}^l$ je težina matrice ili filtera (također poznatog kao *kernel* ili detektor značajki) koji povezuje $j$ – *tu* mapu značajki u sloju $l-1$ $\left(z_j^{l-1}\right)$ s $i$ – *tom* mapom značajki sloja $l$.
- **Nelinearni sloj** – ovaj sloj implementira nelinearnu funkciju ReLU (engl. *rectified linear unit*) koja se primjenjuje na svaku komponentu iz mape značajki kako bi naučila nelinearnu

reprezentaciju: $a^l = f(z^l)$, odnosno kako bi se poboljšala brzina učenja i performanse konvolucijske neuronske mreže.

- **Sloj združivanja** – ovaj sloj ima važnu ulogu u konvolucijskim neuronskim mrežama za smanjenje dimenzija značajki. Kako bi se smanjio broj izlaznih neurona u konvolucijskom sloju, algoritam združivanja trebao bi se primijeniti tako da kombinira susjedne elemente u konvolucijskoj izlaznoj matrici. Često korišteni algoritmi udruživanja uključuju udruživanje prema maksimalnoj vrijednosti i prema prosječnoj vrijednosti.

Slika 3.9. prikazuje primjer povezivanja slojeva unutar MLP-a.




**Slika 3.9. Višeslojni perceptron, MLP arhitektura [19]**

## 3.9.Osnovana svojstva neuronskih mreža

Duboko učenje ima nekoliko svojstava koja opravdavaju njegov status revolucije umjetne inteligencije. Svojstva neuronskih mreža mogu se podijeliti u tri osnovne skupine:

- Jednostavnost – duboko učenje uklanja potrebu za složenim inženjeringom, zamjenjujući kompleksna i teška inženjerska rješenja s jednostavnim s kraja na kraj modelima koji se mogu trenirati, a pritom su uobičajeno izgrađeni koristeći samo pet ili šest različitih tenzor operacija
- Skalabilnost – duboko učenje je vrlo pogodan za paralelizaciju procesa na GPU ili na TPU tako da se može u potpunosti iskoristiti *Mooreov* zakon. Nadalje, modeli dubokog učenja su

naučeni iteriranjem na maloj grupi podataka, omogućujući im da budu naučeni na bazi podataka proizvoljne veličine (jedina mogućnost uskog grla (engl. *bottleneck)* je dostupna količina paralelnih računskih operacija)

- Svestranost i ponovna iskoristivost – za razliku od mnogih prethodnih pristupa, model dubokog učenja može se trenirati na dodatnim podatcima bez potrebe za ponovnim kretanjem od samog početka. Modeli dubokog učenja najčešće se mogu prenamjenjivati te se na taj način mogu ponovno iskoristiti. Na primjer, moguće je iskoristiti model dubokog učenja koji služi za klasifikaciju slika te ga upotrijebiti za obradu videa. To nam omogućuje da se prethodno napravljeni posao, napravljena mreža, može ponovno iskoristiti te nadograditi u složeniji i moćniji model. Duboko učenje je moguće primijeniti na vrlo malim bazama podataka

Duboko učenje je u središtu pozornosti tek nekoliko godina te još nije utvrđen potpuni opseg onoga što može učiniti, odnosno za što sve ga se može iskoristiti. Svakodnevno se spoznaju nove mogućnosti i primjene. Također inženjerska poboljšanja pomiču granice te otklanjaju prethodna ograničenja. Nakon znanstvene revolucije, napredak općenito slijedi *sigmoid* krivulju: počinje s razdobljem od brzog napretka, koji se postupno stabilizira jer inženjeri nailaze na veće prepreke ili ograničenja, a zatim daljnja poboljšanja postaju inkrementalna. Duboko učenje u 2017. godini nalazi se u prvoj polovici krivulje, s mnogo više napretka koji dolazi u sljedećih nekoliko godina prema [18].

# 4. RAZVIJENI SUSTAV ZA SEGMENTACIJU MEDICINSKIH SNIMKI

Razvijeni sustav za segmentaciju medicinskih snimki sastoji se od nekoliko dijelova koji uključuju obradu podataka, učenje U-net modela te provjeru naučenog modela. Prije samog procesa segmentacije potrebno je izvršiti obradu dobivenih podataka za treniranje i testiranje. Obrada podataka uključuje pretvaranje slika u odgovarajući format te promjenu njihove veličine. Zatim slijedi prilagođavanje U – net modela odabirom odgovarajućih parametara. Na kraju je potrebna provjera naučenog modela pomoću dice koeficijenta.

## 4.1. Obrada podataka

Podatci sadrže LGE MRI slike srca te su organizirani tako da se u trening set mapi nalaze dodatne mape od kojih je svaka namijenjena za pojedinu sliku, odnosno sve podatke koji su vezani za jednu trodimenzionalnu (engl. *three-dimensional*, 3D) LGE MRI sliku srca. Svaka mapa unutar trening seta sadrži *Meta Image* zaglavlje za 3D sliku i za njezinu masku te *raw* podatke same 3D slike i maske. Maska predstavlja segmentirani lijevi atrij unutar pripadajuće 3D slike. Slična organizacija podataka nalazi se i u trening setu pri čemu se u trening setu ne nalazi maska.

Iz *Meta Image* i *raw* podataka koristeći programski dio sa slike 4.1. dobivene su slike u Nifti formatu odnosno s ekstenzijom „.nii“, nifti format za pohranjivanje slikovnih podataka često se koristi kod slika magnetske rezonance.

```
TRAIN_PATH = 'training_set/'
TEST_PATH = 'testing_set/'

train_ids = next(os.walk(TRAIN_PATH))[1]
test_ids = next(os.walk(TEST_PATH))[1]


for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):
    path = TRAIN_PATH + id_
    print(path)
    os.chdir(path)
    img = sitk.ReadImage("image.mhd")
    sitk.WriteImage(img, "image.nii")
    img = sitk.ReadImage("gt_binary.mhd")
    sitk.WriteImage(img, "gt_binary.nii")
    os.chdir('../../')


for n, id_ in tqdm(enumerate(test_ids), total=len(test_ids)):
    path = TEST_PATH + id_
    print(path)
    os.chdir(path)
    img = sitk.ReadImage("image.mhd")
    sitk.WriteImage(img, "image.nii")
    os.chdir('../../')
```

**Slika 4.1. Pretvaranje slike u odgovarajući format**

Kao što je vidljivo na slici 4.1. za svaku mapu unutar trening seta, ovim postupkom su, dobivene dvije nove slike u Nifti formatu, iz postojećih podataka. Pri tome jedna slika predstavljaju 3D sliku srca „image.nii“, dok druga predstavlja 3D masku u kojoj je segmentirani dio lijevi atrij - „gt_binary.nii“. Za svaku mapu unutar test seta također je dobivena slika u Nifti formatu „image.nii“. U trening setu nalazi se šest 3D LGE MRI slika srca s odgovarajućim maskama, a u test setu nalazi se dvadeset 3D LGE MRI slika srca.

Dobivene 3D LGE -MRI slike potrebno je prilagoditi ulazu u neuronsku mrežu, što znači da ih je potrebno staviti u odgovarajući oblik. Dio programskog koda koji služi za promjenu veličine slike nalazi se na Slici 4.2..

```
for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):
    path = TRAIN_PATH + id_
    os.chdir(path)
    orig_nii = nb.load("image.nii")
    some_affine = orig_nii.affine
    target_shape = np.array((320, 320, 320))
    downsampled_and_cropped_nii = resample_img(orig_nii, target_affine=some_affine, target_shape=target_shape,
                                               interpolation='nearest')
    downsampled_and_cropped_nii.affine
    downsampled_and_cropped_nii.to_filename('img_resized.nii')
    img_resized = imread("img_resized.nii")

    orig_nii = nb.load("gt_binary.nii")
    some_affine = orig_nii.affine
    target_shape = np.array((320, 320, 320))
    downsampled_and_cropped_nii = resample_img(orig_nii, target_affine=some_affine, target_shape=target_shape,
                                               interpolation='nearest')
    downsampled_and_cropped_nii.affine
    downsampled_and_cropped_nii.to_filename('mask_resized.nii')
    mask_resized = imread("mask_resized.nii")
    os.chdir('../../')

for n, id_ in tqdm(enumerate(test_ids), total=len(test_ids)):
    path = TEST_PATH + id_
    os.chdir(path)
    orig_nii = nb.load("image.nii")
    some_affine = orig_nii.affine
    target_shape = np.array((320, 320, 100))
    downsampled_and_cropped_nii = resample_img(orig_nii, target_affine=some_affine, target_shape=target_shape,
                                               interpolation='nearest')
    downsampled_and_cropped_nii.affine
    downsampled_and_cropped_nii.to_filename('img_resized.nii')
    img_resized = imread("img_resized.nii")
    os.chdir('../../')
```

**Slika 4.2. Pretvaranje slike u odgovarajući oblik**

Slike za trening se nalaze u obliku (320, 320, 320) zbog kasnijeg uzimanja dijelova (engl. *slices*) slike po svim osima (x, y i z) kako bi se dobile dvodimenzionalne 2D (engl. *two-dimensional*) slike.

Nakon što je dobivena slika u odgovarajućem formatu i obliku, napravljene su pomoćne funkcije koje se kasnije koriste kako bi se za svaku 3D sliku spremile odgovarajuće 2D slike u polje.

```
def getImageDepth(img, id_ ):
    path = TRAIN_PATH + id_
    os.chdir(path)
    image = sitk.ReadImage(img)
    max_index = image.GetDepth()
    return max_index, image
```

**Slika 4.3. Funkcija *getImageDepth***

Funkcija ***getImageDepth*** koristi se za dohvaćanje 3D slika te za određivanje dubine slike odnosno veličine slike po z-osi. Budući da su slike jednake visine, širine i dubine ova funkcija se koristiti za određivanje veličine slike po bilo kojoj osi.

```
def getSlicesMask(max_index, a, j):
    for i in range(max_index - 1):
            Y_train_new[a] = list_of_2D_images_np_mask[i]
            a = a + 1
            if np.count_nonzero(Y_train_new[a - 1]) != 0:
                j = j + 1
    return j


def getSlicesImage(max_index, a):
    for i in range(max_index - 1):
        X_train_new[a] = list_of_2D_images_np[i]
        a = a + 1
```

**Slika 4.4. Funkcije za spremanje 2D dijelova slike u polje**

Funkcije na slici 4.4. koriste se za spremanje 2D dijelova slike i maske u polje. Pri čemu se u funkciji za spremanje 2D dijelova maske prebrojavaju oni dijelovi maske koji sadrže lijevi atrij.

```
#dohvacanje broja slice - ova 3D slike
img = "mask_resized.nii"
max_index = getImageDepth(img,train_ids[0])
max_index = max_index[0]

#deklaracija polja za sve slice - ove svih slika
Y_train_new = np.zeros((max_index * len(train_ids) * 3, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)
X_train_new = np.zeros((max_index * len(train_ids) * 3, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)

#spremanje slice - ova u polje, prebrojavnje onih koji sadrze lijevi atrij (j)
for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):

    img = "mask_resized.nii"
    max_index, image = getImageDepth(img, id_)
    list_of_2D_images_np_mask = [sitk.GetArrayFromImage(image[:, :, i]) for i in range(max_index)]

    a = max_index * n
    j = getSlicesMask(max_index, a, j)

    img = "img_resized.nii"
    max_index, image = getImageDepth(img, id_)
    list_of_2D_images_np = [sitk.GetArrayFromImage(image[:, :, i]) for i in range(max_index)]

    a = max_index * n
    getSlicesImage(max_index, a)

    os.chdir('../../')
```

**Slika 4.4. Dohvaćanje 2D dijelova slike kao polja**

Pomoću funkcije ***GetArrayFromImage*** koja je dio SimpleITK biblioteke slika se pretvara u 2D polje brojeva s pomičnim zarezom, u ovom slučaju, uzimajući 2D dijelove slike po z - osi. Nakon što je slika pretvorena u polje, sprema se pomoću funkcije ***getSlicesImage*** u *numpy* polje koje predstavlja odgovarajući ulaz u neuronsku mrežu. Isti postupak obavljen je za druge dvije osi, odnosno spremljene su 2D slike po x i y osi. Isti ovakav postupak odvija se i na testnim podatcima, ali samo po z osi.

Slika 4.5. prikazuje dio programskog koda koji se odnosi na podjelu podataka na trening i validaciju. Prije same podijele podataka pronalaze se 2D slike i maske u kojima se ne nalazi lijevi atrij te se iste brišu iz polja, na način da se sprema indeks 2D slike na kojoj nema lijevog atrija te se slike, odnosno dvodimenzionalna polja, na tom indeksu uklanjaju. Za validaciju se uzima prvih 15% podataka te se ti podatci uklanjaju iz podataka za trening.

```
#deklaracija polja za slice - ove s lijevim atrijem
Y_train = np.zeros((j, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)
X_train = np.zeros((j, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)

#uzimanje 15% podataka za vallidaciju
z = int(0.15 * j)

#deklaracija polja za validaciju
Y_val = np.zeros((z, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)
X_val = np.zeros((z, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)

#spemanje indeksa polja na mjestima gdje nema lijevog atrija
for i in range(max_index * len(train_ids) * 3):
    if np.count_nonzero(Y_train_new[i]) == 0:
        index.append(i)

#brisanje slice - ova koji ne sadrze lijevi atrij, spremanje svih slice - ova s lijevim atrijem u polje
y_train = np.delete(Y_train_new, index, axis = 0)
x_train = np.delete(X_train_new, index, axis = 0)

#uzimanje prvih z vrijednosti za validaciju
for i in range(z):
    Y_val[i] =  y_train[i]
    X_val[i] =  x_train[i]

validation = np.arange(z)

#brisanje prvih z vrijednosti iz podataka za trening
Y_train = np.delete(y_train, validation, axis = 0)
X_train = np.delete(x_train, validation, axis = 0)
```

**Slika 4.5. Prikaz programskog dijela za podjelu podataka na trening i validaciju**

Nakon što su podatci raspodijeljeni, vrši se normalizacija podataka s ciljem bržeg učenja i bržeg ostvarivanja usklađivanja (engl. *convergence*). Postavljene su određene vrijednosti za minimalni i maksimalni iznos intenziteta te je definirana funkcija prema slici 4.6..

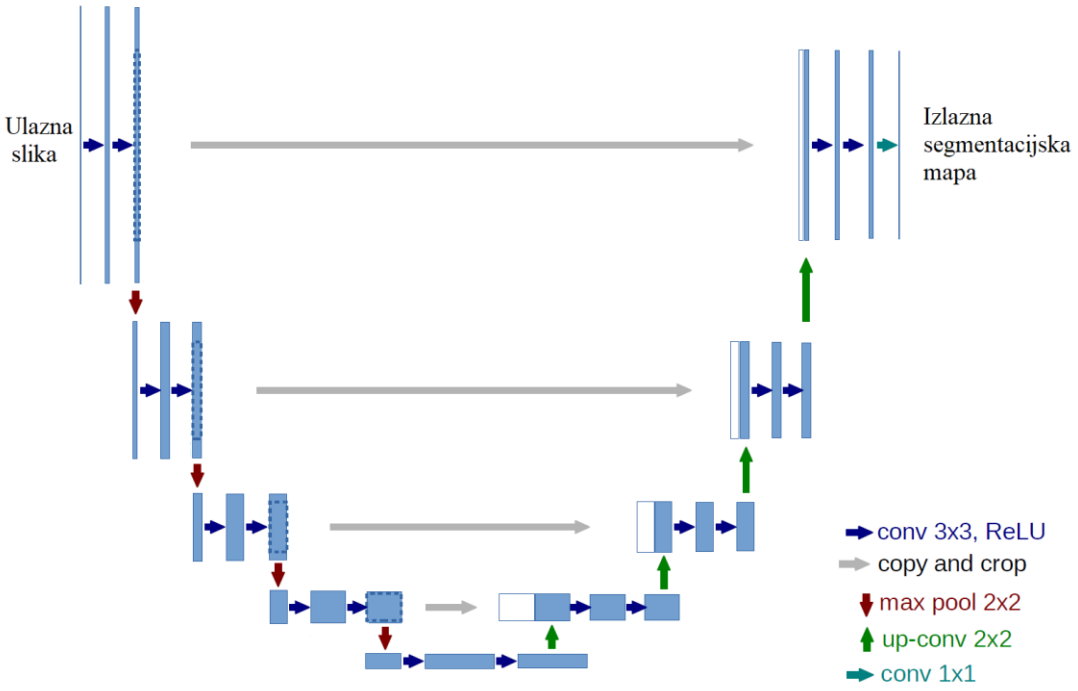
```
def normalizeImageIntensityRange(img):
    img[img < HOUNSFIELD_MIN] = HOUNSFIELD_MIN
    img[img > HOUNSFIELD_MAX] = HOUNSFIELD_MAX
    return (img - HOUNSFIELD_MIN) / HOUNSFIELD_RANGE
```

**Slika 4.6. Funkcija za normalizaciju podataka**

Normalizacija podataka izvršava se na svim podatcima, odnosno na trening, validacijskom i testnom skupu podataka.

## 4.2.U – Net arhitektura neuronske mreže

U – Net model razvijen je za segmentaciju biomedicinskih slika. Arhitektura sadrži dva smjera. Prvi smjer se naziva put kontrakcije (engl. *contraction path*), također se naziva i enkoder te se koristi za uzimanje dijela slike koji filter pokriva u danom trenutku. Enkoder se sastoji od konvolucijskih slojeva i slojeva udruživanja. Drugi smjer je simetričan i naziva se put širenja (engl. *expansive path*), naziva se još i dekoder, koji se koristi za omogućavanje precizne lokalizacije pomoću transponiranih konvolucijskih slojeva. Izgled arhitekture U-Net konvolucijske neuronske mreže prikazan je na slici 4.6., dok je ekvivalentno programsko rješenje prikazano na slici 4.7. .




**Slika 4.6. U-net arhitektura**

```
# Build modela
inputs = tf.keras.layers.Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))

# Contraction path
c1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',kernel_initializer='he_normal', padding='same')(inputs)
c1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',kernel_initializer='he_normal', padding='same')(c1)
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal',padding='same')(p1)
c2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',kernel_initializer='he_normal', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',padding='same')(p2)
c3 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal',padding='same')(p3)
c4 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal',padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu',kernel_initializer='he_normal', padding='same')(p4)
c5 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu', kernel_initializer='he_normal',padding='same')(c5)

# Expansive path
u6 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(c5), c4], axis = 3)
c6 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',kernel_initializer='he_normal', padding='same')(u6)
c6 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',kernel_initializer='he_normal', padding='same')(c6)

u7 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c6), c3], axis = 3)
c7 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal',  padding='same')(u7)
c7 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',kernel_initializer='he_normal',  padding='same')(c7)

u8 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c7), c2], axis = 3)
c8 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u8)
c8 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c8)

u9 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c8), c1], axis = 3)
c9 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u9)
c9 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c9)

outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

model = tf.keras.Model(inputs=[inputs], outputs=[outputs])

opt = keras.optimizers.Adam(lr = 1e-5 , beta_1 = 0.9 , beta_2 = 0.999 , epsilon = 1e-08 , decay = 0.000000199)
model.compile ( optimizer = opt, loss = 'binary_crossentropy' , metrics = [ 'accuracy' ])

model.summary()
```

**Slika 4.7. U-net arhitektura**

Enkoder U – Net modela se sastoji se od sljedećih dijelova:

- *Conv2D* slojevi označavaju primjenu dva uzastopna sloja konvolucije
- *c1*, *c2*, … *c9* predstavljaju izlazne tenzore konvolucijskih slojeva
- *p1*, *p2*, *p3* i *p4* predstavljaju izlazne tenzore slojeva udruživanja
- *u6*, *u7*, *u8* i *u9* predstavljaju izlazne tenzore sloja za uzimanje uzoraka (engl. *up - sampling*) odnosno transponiranog konvolucijskog sloja
- U enkoderu se primjenjuju samo regularni konvolucijski slojevi i slojevi pridruživanja
- U enkoderu se slika postupno smanjuje dok se dubina postupno povećava, počevši od (320, 320, 1) do (20, 20, 512).

Prema tome, unutar enkodera se mreža uči što je informacija sa slike, ali pri tome gubi informaciju gdje se ta informacija nalazi.

Nadalje, dekoder U – Net modela se sastoji se od sljedećih dijelova:

- Desna strana je put ekspanzije (dekoder) na koji primjenjujemo transponirane konvolucije zajedno s regularnim konvolucijama
- U dekoderu se veličina slike postupno povećava, a dubina postupno smanjuje, počevši od (20, 20, 512) do (320, 320, 1)
- Dekoder obnavlja podatke gdje se nalazi informacija (precizna lokalizacija) postupnom primjenom transponiranog konvolucijskog sloja
- Kako bi se dobile precizne lokacije, na svakom koraku dekodera koristi se preskakanje veza spajanjem izlaza transponiranih slojeva konvolucije s kartama značajki iz kodera na istoj razini
- Nakon svakog spajanja ponovno se primjenjuju dvije uzastopne konvolucije kako bi model mogao sastaviti precizniji izlaz
- Ulazna i izlazna matrica iste su veličine, u ovom slučaju (320, 320, 1), 320 × 320 dimenzije su slike, a 1 predstavlja broj kanala. Budući da je slika sivih nijansi broj kanala je jednak jedan

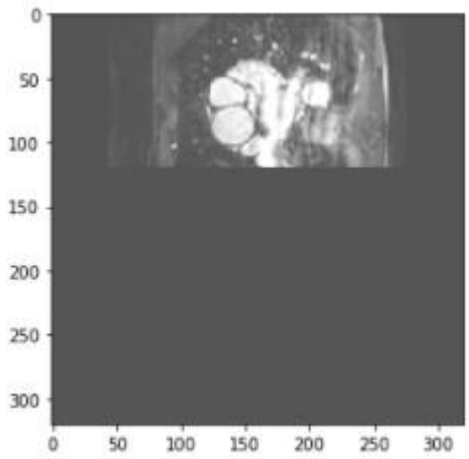
Model je kompiliran pomoću modificiranog *Adam* optimizatora, prema [21], te je korištena *binary_crossentropy* funkcija gubitaka budući da postoje samo dvije mogućnosti, mjesta na kojima se nalazi lijevi atrij i mjesta na kojima se ne nalazi lijevi atrij. Model je treniran kroz 25 epoha pri čemu je veličina serije (engl. *batch size*) jednaka četiri.

# 5. DOBIVENI REZULTATI

Cilj semantičke segmentacije slike je označiti svaki piksel slike odgovarajućom klasom onoga što predstavlja. Budući da se za svaki piksel na slici vrši predviđanje, ovaj se način obično naziva gustim predviđanjem. Očekivani izlaz u semantičkoj segmentaciji je slika visoke razlučivosti, obično iste veličine kao i ulazna slika. Stoga, semantička segmentacija predstavlja klasifikaciju slike na razini piksela.

Predviđanje naučenog modela, odnosno klasifikacija piksela ulazne slike, napravljeno je za tri skupine podataka, trening podatke, validacijske i testne podatke. Za svaki piksel dobivena je vrijednosti između 0 i 1. Kako bi izlazna slika bila jasnija, postavljen je prag na 0.5, što znači da sve vrijednosti izlaznih piksela koje su manje od 0.5 postavljaju se na 0, a sve vrijednosti koje su veće od 0.5 postavljaju se na 1.

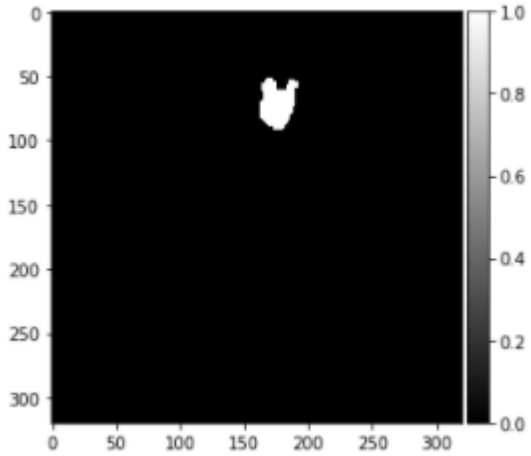
Primjer segmentacije lijevog atrija na trening skupu pomoću naučenog modela prikazan je na slikama 5.1., 5.2. i 5.3..




**Slika 5.1. Prikaz ulazne LGE MRI slike srca u naučeni model**

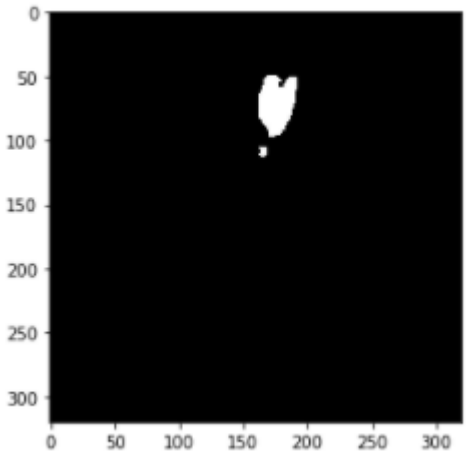
Slika 5.1. predstavlja ulaznu sliku koja je nastala pomoću LGE MRI metode, slika je dimenzija 320 × 320, predstavlja presjek srca po x ili y osi stoga je sama informacija unutar slike manja.

Nadalje, na slici 5.2. prikazana je odgovarajuća maska slike 5.1., maska sadrži samo 0 i 1 pri čemu pikseli koji imaju vrijednost 1 predstavljaju lijevi atrij.



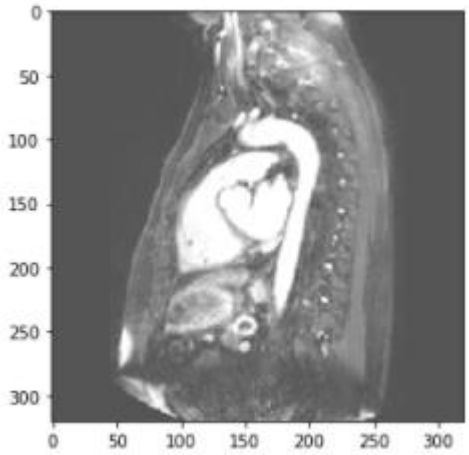

**Slika 5.2. Prikaz maske za odgovarajuću sliku srca**

Slika 5.3. predstavlja predikciju modela, odnosno segmentirani dio srca, lijevi atrij. Segmentirani dio razlikuje se od maske sa slike 5.2. koja predstavlja očekivani rezultat segmentacije. Maska također predstavlja ulazni podatak za samo treniranje modela, kako bi model mogao prepoznati koji dio slike predstavlja lijevi atrij.



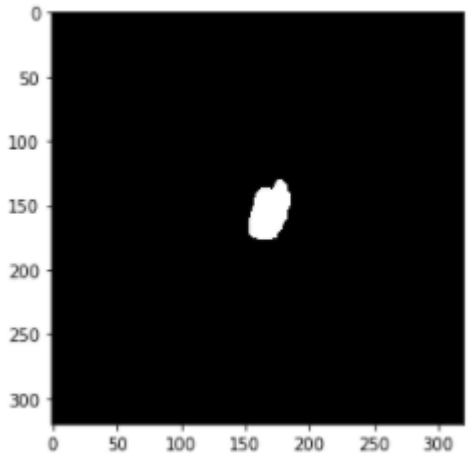

**Slika 5.3. Prikaz predikcije, segmentiranog lijevog atrija**

Sljedeći podatci pripadaju validacijskom skupu te su prikazani na slikama 5.4., 5.5., 5.6.. Slika 5.4. slika je LGE MRI srca po z-osi te predstavlja ulaznu sliku u trenirani U - Net model.



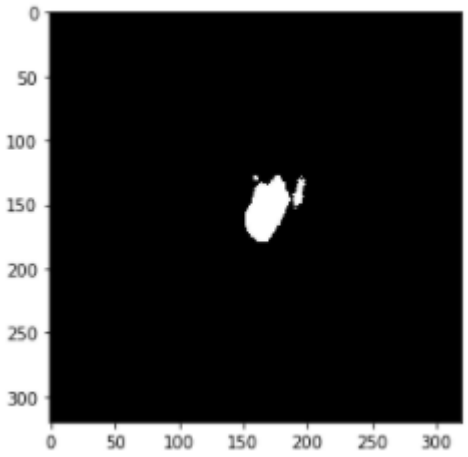

**Slika 5.4. Prikaz ulazne LGE MRI slike srca validacijskog skupa u naučeni model**

Slika 5.5. predstavlja masku slike 5.4. odnosno očekivani izlaz nakon predikcije.




**Slika 5.5. Prikaz maske za odgovarajuću sliku srca**

Slika 5.6. predstavlja predikciju modela, segmentirani lijevi atrij. Očekivani izlaz modela razlikuje se od dobivenog, ali postoji velika razina poklapanja, odnosno vrijednosti piksela na obje slike se podudaraju .




**Slika 5.6. Prikaz predikcije modela**

Kako bi se odredio točan iznos poklapanja validacijskog skupa odnosno maski iz tog skupa s odgovarajućim dobivenim vrijednostima, predikcijama, potrebno je odrediti *dice* koeficijent.

*Dice* koeficijent računa se kao dvostruka površina preklapanja podijeljena s ukupnim brojem piksela na obje slike, očekivanoj izlaznoj slici i dobivenoj, predstavlja najčešće korištenu metriku za procjenu segmentacije kod medicinskih snimki. Matematički se računa kao količnik dvostrukog kardinalnog broja presjeka skupa X i Y te zbroja kardinalnog broja skupa X i Y, kao što je prikazano Formulom (5-1).

$$D(X,Y) = \frac{2\,|X \cap Y|}{|X| + |Y|} \tag{5-1}$$

Programski dio kojim se dobiva *dice* koeficijent prikazan je na slici 5.7..

```
def dice_coef(y_true, y_pred, smooth=1):
  intersection = K.sum(y_true * y_pred, axis=[1,2])
  union = K.sum(y_true, axis=[1,2]) + K.sum(y_pred, axis=[1,2])
  dice = K.mean((2. * intersection + smooth)/(union + smooth), axis=0)
  return dice
```

**Slika 5.7. Prikaz funkcije za izračun *dice* koeficijenta**

U ovom radu dobiven je dice koeficijent od 0.67 za segmentaciju lijevog atrija iz LGE-MRI slika. Rezultat je dobiven na validacijskom skupu podataka, što ga čini mjerodavnim pokazateljem kvalitete razvijenog rješenja. Tablica 5.1. predstavlja usporedbu ostvarenog rezultata sa *state-of-the-art* metodama ovoga područja.

**Tablica 5.1. Usporedba dobivenih rezultata sa *state-of-the-art* metodama**

| Autori | Metoda | Veličina podataka za treniranje mreže | Dice koeficijent |
|---|---|---|---|
| Xiong et al. [1] | AtriaNet | 154 3D LGE – MRI | 0.94 |
| Yue et al. [3] | SRNN + SCN | 45 3D LGE – MRI | 0.76 |
| Predloženi pristup | U-Net arhitektura | 6 3D LGE – MRI | 0.67 |

Jedan od glavnih razloga dobivenog nižeg *dice* koeficienta u usporedbi sa drugim radovima je mala količina podataka za treniranje mreže. Baza podataka koja je korištena u ovom radu sadrži 6 3D LGE MRI slika srca za treniranje mreže, dok su drugi radovi koristili znatno veće baze podataka. Kako bi se ostvario bolji rezultat potrebne su dodatne modifikacije U – Net modela i povećanje skupa podataka za treniranje.

# 6. ZAKLJUČAK

U radu je predstavljena automatska metoda za segmentaciju lijevog atrija iz 3D LGE MRI slika srca. Segmentacija lijevog atrija napravljena je pomoću konvolucijske neuronske mreže koristeći U – Net model, ulazni podatci u model su 2D dijelovi slika. Dobiveni rezultat segmentacije provjeren je često korištenom metrikom za medicinske snimke odnosno *dice* koeficijentom. Ostvareni *dice* koeficijent manji je od rezultata koji su postignuti u znanstvenim radovima te iznosi 0.67. Dobiveni rezultat je lošiji zbog korištenja manje baze podataka, računala slabijih performansi te zbog korištenja originalne U – Net mreže bez dodatnih modifikacija. Kako bi se poboljšali rezultati nužno je proširiti bazu podataka te modificirati U – Net model.

## Zahvale

Ovaj je rad sufinancirala Hrvatska zaklada za znanost projektom UIP-2017-05-4968.

## Literatura

[1] Z. Xiong, V. V. Fedorov, X. Fu, E. Chang, R. Macleod i J. Zhao, Fully Automatic Left Atrium Segmentation from Late Gadolinium Enhanced Magnetic Resonance Imaging Using a Dual Fully Convolutional Neural Network, IEEE Trans Med Imaging, str. 515-524, Veljača 2019.

[2] K. Engan, V. Naranjo, T. Eftestol, S. Orn i L. Woie, Segmentation of LG Enhanced Cardiac MRI, Proceedings of the International Conference on Bioimaging, str. 47-55, 2015.

[3] Q. Yue, X. Luo, Q. Ye, L. Xu i X. Zhuang, Cardiac Segmentation from LGE MRI Using Deep Neural Network Incorporating Shape and Spatial Priors, MICCAI, 2019.

[4] X. Wang, S. Yang, M. Tang, Y. Wei, L. He, J. Zhang, X. Han, SK-Unet: an Improved U-net Model with Selective Kernel for the Segmentation of Multi-sequence Cardiac MR

[5] C. Chen, C. Qin, H. Qiu, G. Tarroni, J. Duan, W. Bai i D. Rueckert, Deep Learning for Cardiac Image Segmentation: A Review, Frontiers Cardiovascular Medicine, Vol.7, str. 7-25, Ožujak 2020.

[6] Cardiovascular System, dostupno na: https://www.pearsonhighered.com/assets/samplechapter/0/1/3/4/0134760611.pdf

[7] Anatomy and Physiology of the Cardiovascular System, dostupno na: http://samples.jbpub.com/9781449652609/99069_ch05_6101.pdf, datum zadnje posjete: 18.9.2020.

[8] R. Putz, R. Pabst, Sobotta, Atlas of Human Anatomy, Urban & Fischer, Munchen, dostupno na: http://www.naprapat.com/sobotta/sobotta2.pdf, datum zadnje posjete: 18.9.2020.

[9] Anatomy and physiology of the cardiovascular system - PowerPoint PPT Presentation, dostupno na: https://www.slideserve.com/zoltan/anatomy-and-physiology-of-the-cardiovascular-system, datum zadnje posjete: 18.9.2020.

[10] S. Whiteman, E. Saker, V. Courant, S. Salandy, J. Gielecki, A. Zurada, M. Loukas, An anatomical review of the left atrium, Translational Research in Anatomy, Vol.17, Studeni 2019

[11] L. Crumbie, Atria of the heart, dostupno na: https://www.kenhub.com/en/library/anatomy/the-atria-of-the-heart, datum zadnje posjete: 18.9.2020.

[12] Magnetic Resonance Imaging (MRI), National Institute of Biomedical Imaging and Bioengineering, dostupno na : https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri, datum zadnje posjete: 18.9.2020.

[13] M. Larobina I L. Murino, Medical Image File Formats, Journal of Digital Imaging, Prosinac 2013.

[14] https://www.cambridgeincolour.com/tutorials/raw-file-format.htm, datum zadnje posjete: 18.9.2020.

[15] A. S. Fahmy, U. Neisius, C. W. Tsao, S. Berg, E. Goddu, P. Pierce, T. A. Basha, L. Ngo, W. J. Manning I R. Nezafat, Gray blood late gadolinium enhancement cardiovascular magnetic resonance for improved detection of myocardial scar, Journal of Cardiovascular Magnetic Resonance,Vol.20, 22. Ožujka 2018.

[16] C. McGann, E. Kholmovski, J. Blauer, S. Vijayakumar, T. Haslam, J. Cates, E. DiBella, N. Burgon, B.Wilson, A. Alexander, M. Prastawa, M. Daccarett, G. Vergara, N. Akoum, D. Parker, R. MacLeod I N. Marrouche, Dark Regions of No-Reflow on Late Gadolinium Enhancement Magnetic Resonance Imaging Result in Scar Formation After Atrial Fibrillation Ablation, Journal of the American College of Cardiology, Vol.58, Srpanj 2011.

[17] K. Balaji i K. Lavanya, Medical Image Analysis With Deep Neural Networks, Deep Learning and Parallel Computing Environment for Bioengineering Systems, 2019.

[18] F. Chollet, Deep Learning with Python, Manning, Shelter Island, 2018.

[19] M. E. Paoletti, J. M. Haut, J. Plaza, A. Plaza, A new deep convolutional neural network for fast hyperspectral image classification, ISPRS Journal of Photogrammetry and Remote Sensing Vol.145, Str. 120-147, Studeni 2018.

[20] Q. Li, W. Cai, X. Wang, Y. Zhou, D. Dagan Feng i M. Chen, Medical Image Classification with Convolutional Neural Network, 13th International Conference on Control, Automation, Robotics & Vision, Singapore, Prosinac 2014.

[21] https://github.com/mrkolarik/3D-brain-segmentation/blob/master/3D-unet.py

## Sažetak

Segmentacija dijelova srca iz kasno postkontrastno snimljenih slika magnetne rezonance (LGE MRI) predstavlja važan dio svakodnevne medicinske prakse kako bi se pravilno identificirala fibroza miokarda te drugih bolesti srca i kardiovaskularnog sustava. Međutim, automatska segmentacija i dalje predstavlja veliki izazov zbog heterogene raspodjele intenziteta i nejasnih granica između dijelova srca. U ovom radu predstavljena je automatska metoda segmentacije lijevog atrija iz trodimenzionalnih (3D) LGE MRI slika temeljena na dubokom učenju. Segmentacija lijevog atrija izvedena je pomoću konvolucijske neuronske mreže korištenjem U – Net modela. U-Net model treniran je pomoću izrazito malog skupa podataka od 6 3D LGE MRI slika. Unatoč tome, uspješno je ostvaren *dice* koeficijent od 0.67. Prema tome, u radu je pokazano da originalna U-Net arhitektura može ostvariti uspješan rezultat za segmentaciju lijevog atrija iz LGE-MRI slika upotrebom malenog broja podataka za učenje mreže.

## Ključne riječi

duboko učenje, konvolucijske neuronske mreže, magnetska rezonanca, segmentacija lijevog atrija, semantička segmentacija, U-Net

## Abstract

Segmentation of the heart chambers from late gadolinium enhancement magnetic resonance images (LGE MRI) is an important part of everyday medical practice. It allows proper identification of myocardial fibrosis and other various heart and cardiovascular system diseases and conditions. However, automatic segmentation remains a major challenge due to the heterogeneous distribution of intensities and blurred boundaries between parts of the heart. This paper presents an automatic left atrial segmentation method from three-dimensional (3D) LGE MRI images based on deep learning. Left atrial segmentation was performed using a convolutional neural network using the U - Net model. The U-Net model was trained using an extremely small data set of 6 3D LGE MRI images. Despite this, a dice coefficient of 0.67 was successfully achieved. Therefore, the paper shows that the original U-Net architecture can successfully segmentation the left atrium from LGE-MRI images using a small amount of data for network learning.

## Keywords

convolutional neural networks, deep learning, left atrium segmentation, magnetic resonance, semantic segmentation, U-Net

## Prilozi

### SegmentacijaLijevogAtrija.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# # Segmentacija lijevog atrija pomoću konvolucijske neuronseke mreze

# In[1]:


import tensorflow as tf
import keras
import os
import random
import numpy as np

from tqdm import tqdm

from skimage.io import imread, imshow
from skimage.transform import resize
import matplotlib.pyplot as plt

import nibabel as nib
import SimpleITK as sitk
import numpy as np

import cv2
from tensorflow import keras
from keras import backend as K


# ### Inicijalizacija varijabli

# In[2]:


seed = 49
np.random.seed = seed

#podatci za sliku
```

```
IMG_CHANNELS = 1
IMG_WIDTH = 320
IMG_HEIGHT = 320

TRAIN_PATH = 'C:/LeftAtriumSegmentation/training_set/'
TEST_PATH = 'C:/LeftAtriumSegmentation/testing_set/'

train_ids = next(os.walk(TRAIN_PATH))[1]
test_ids = next(os.walk(TEST_PATH))[1]

#podatci za normalizaciju
HOUNSFIELD_MIN = -500
HOUNSFIELD_MAX = 1000
HOUNSFIELD_RANGE = HOUNSFIELD_MAX - HOUNSFIELD_MIN

#broj slice - ova koji sadrze lijevi atrij
j = 0

#indeksi na kojima se ne nalazi lijevi atrij
index = []


# In[3]:


def getImageDepth(img, id_ ):
    path = TRAIN_PATH + id_
    os.chdir(path)
    image = sitk.ReadImage(img)
    max_index = image.GetDepth()
    return max_index, image


# In[4]:


def getSlicesMask(max_index, a, j):
    for i in range(max_index - 1):
            Y_train_new[a] = list_of_2D_images_np_mask[i]
            a = a + 1
            if np.count_nonzero(Y_train_new[a - 1]) != 0:
                j = j + 1
    return j
```

```
    # In[5]:


    def getSlicesImage(max_index, a):
        for i in range(max_index - 1):
            X_train_new[a] = list_of_2D_images_np[i]
            a = a + 1


    # ### Generiranje 2D slice - ova

    # In[6]:


    #dohvacanje broja slice - ova 3D slike
    img = "mask_resized.nii"
    max_index = getImageDepth(img,train_ids[0])
    max_index = max_index[0]

    #inicijallizacija polja za sve slice - ove svih slika
    Y_train_new = np.zeros((max_index * len(train_ids) * 3, IMG_HEIGHT, IMG_WIDTH),
 dtype=np.float32)
    X_train_new = np.zeros((max_index * len(train_ids) * 3, IMG_HEIGHT, IMG_WIDTH),
 dtype=np.float32)

    #spremanje slice -
 ova u polje, prebrojavnje onih koji sadrze lijevi atrij (j)
    for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):

        img = "mask_resized.nii"
        max_index, image = getImageDepth(img, id_)
        list_of_2D_images_np_mask = [sitk.GetArrayFromImage(image[:, :, i]) for i i
n range(max_index)]

        a = max_index * n
        j = getSlicesMask(max_index, a, j)

        img = "img_resized.nii"
        max_index, image = getImageDepth(img, id_)
        list_of_2D_images_np = [sitk.GetArrayFromImage(image[:, :, i]) for i in ran
ge(max_index)]

        a = max_index * n
        getSlicesImage(max_index, a)
```

```
        os.chdir('../../')
    print(j)


    # In[7]:


    #spremanje slice -
 ova u polje, prebrojavnje onih koji sadrze lijevi atrij (j)
    for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):

        img = "mask_resized.nii"
        max_index, image = getImageDepth(img, id_)
        list_of_2D_images_np_mask = [sitk.GetArrayFromImage(image[:, i, :]) for i i
n range(max_index)]

        a = (max_index * len(train_ids)) + max_index * n
        j = getSlicesMask(max_index, a, j)

        img = "img_resized.nii"
        max_index, image = getImageDepth(img, id_)
        list_of_2D_images_np = [sitk.GetArrayFromImage(image[:, i, :]) for i in ran
ge(max_index)]

        a = (max_index * len(train_ids)) + max_index * n
        getSlicesImage(max_index, a)

        os.chdir('../../')
    print(j)


    # In[8]:


    #spremanje slice -
 ova u polje, prebrojavnje onih koji sadrze lijevi atrij (j)
    for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):

        img = "mask_resized.nii"
        max_index, image = getImageDepth(img, id_)
        list_of_2D_images_np_mask = [sitk.GetArrayFromImage(image[i, :, :]) for i i
n range(max_index)]

        a = ((max_index * len(train_ids))*2) + max_index * n
        j = getSlicesMask(max_index, a, j)
```

```
        img = "img_resized.nii"
        max_index, image = getImageDepth(img, id_)
        list_of_2D_images_np = [sitk.GetArrayFromImage(image[i, :, :]) for i in ran
ge(max_index)]

        a = ((max_index * len(train_ids))*2) + max_index * n
        getSlicesImage(max_index, a)

        os.chdir('../../')
    print(j)


    # ### Odabir slice -
 ova koji sadrze lijevi atrij te podjela podataka na trening i validaciju

    # In[9]:


    #inicijalizacija polja za slice - ove s lijevim atrijem
    print(j)
    Y_train = np.zeros((j, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)
    X_train = np.zeros((j, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)

    #uzimanje 15% podataka za vallidaciju
    z = int(0.15 * j)

    #inicijalizacija polja za validaciju
    Y_val = np.zeros((z, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)
    X_val = np.zeros((z, IMG_HEIGHT, IMG_WIDTH), dtype=np.float32)

    #spemanje indeksa polja na mjestima gdje nema lijevog atrija
    for i in range(max_index * len(train_ids) * 3):
         if np.count_nonzero(Y_train_new[i]) == 0:
            index.append(i)

    #brisanje slice - ova koji ne sadrze lijevi atrij, spremanje svih slice -
 ova s lijevim atrijem u polje
    y_train = np.delete(Y_train_new, index, axis = 0)
    x_train = np.delete(X_train_new, index, axis = 0)

    #uzimanje prvih z vrijednosti za validaciju
    for i in range(z):
        Y_val[i] =  y_train[i]
        X_val[i] =  x_train[i]
```

```
validation = np.arange(z)

#brisanje prvih z vrijednosti iz podataka za trening
Y_train = np.delete(y_train, validation, axis = 0)
X_train = np.delete(x_train, validation, axis = 0)

print(X_train_new.shape)

print(len(Y_train))
print(len(X_train))

print(X_train.shape)
print(Y_train.shape)


# In[10]:


#prikaz random slike za trening i njene maske
image_x = random.randint(0, len(train_ids))
imshow(X_train[image_x])
plt.show()
imshow(np.squeeze(Y_train[image_x]))
plt.show()
np.min(X_train), np.max(X_train), X_train.shape, type(X_train)


# ### Normalizacija podataka slike

# In[11]:


#funkcija za normalizaciju
def normalizeImageIntensityRange(img):
    img[img < HOUNSFIELD_MIN] = HOUNSFIELD_MIN
    img[img > HOUNSFIELD_MAX] = HOUNSFIELD_MAX
    return (img - HOUNSFIELD_MIN) / HOUNSFIELD_RANGE


# In[12]:


#normalizacija trening podataka
img = X_train
```

```
nImg = normalizeImageIntensityRange(img)
np.min(nImg), np.max(nImg), nImg.shape, type(nImg)
X_train = nImg


# In[13]:


#prikaz random normalizirane slike za trening i njene maske
image_x = random.randint(0, (j - z))
imshow(X_train[image_x])
plt.show()
plt.imshow(np.squeeze(Y_train[image_x]), cmap = 'gray')
plt.show()
np.min(X_train), np.max(X_train), X_train.shape, type(X_train)


# In[14]:


#normalizacija podataka za validaciju
img = X_val
nImg = normalizeImageIntensityRange(img)
np.min(nImg), np.max(nImg), nImg.shape, type(nImg)
X_val = nImg


# In[15]:


#prikaz random normalizirane slike za validaciju i njene maske
image_x = random.randint(0, z)
imshow(X_val[image_x ])
plt.show()
plt.imshow(np.squeeze(Y_val[image_x ]), cmap = 'gray')
plt.show()
np.min(X_val), np.max(X_val), X_val.shape, type(X_val)


# ### Generiranje 2D slice - ova za testiranje

# In[16]:


#uzimanje slice - ova iz slika za testiranje
```

```
    for n, id_ in tqdm(enumerate(test_ids), total=len(test_ids)):
        path = TEST_PATH + id_
        os.chdir(path)
        image = sitk.ReadImage("img_resized.nii")
        max_index = image.GetDepth()

    X_test = np.zeros((max_index * len(test_ids), IMG_HEIGHT, IMG_WIDTH), dtype=np.
float32)

    for n, id_ in tqdm(enumerate(test_ids), total=len(test_ids)):
        path = TEST_PATH + id_
        os.chdir(path)
        image = sitk.ReadImage("img_resized.nii")
        max_index = image.GetDepth()
        list_of_2D_images_np = [sitk.GetArrayFromImage(image[:, :, i]) for i in ran
ge(max_index)]

        a = max_index * n
        for i in range(max_index):
            X_test[a] = list_of_2D_images_np[i]
            a = a + 1

        os.chdir('../../')


    # ### Normalizacija podataka za testiranje

    # In[17]:


    #normaliziranje slika za testiranje
    img = X_test
    nImg = normalizeImageIntensityRange(img)
    np.min(nImg), np.max(nImg), nImg.shape, type(nImg)
    X_test = nImg


    # In[18]:


    image_x = random.randint(0, len(test_ids))
    imshow(X_test[image_x])
    plt.show()
    np.min(X_test), np.max(X_test), X_test.shape, type(X_test)
```

```
    # ### Model U-net

    # In[19]:


    # Build modela
    inputs = tf.keras.layers.Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))

    # Contraction path
    c1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',kernel_initializer='h
e_normal', padding='same')(inputs)
    c1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',kernel_initializer='h
e_normal', padding='same')(c1)
    p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

    c2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='
he_normal',padding='same')(p1)
    c2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',kernel_initializer='h
e_normal', padding='same')(c2)
    p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

    c3 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer=
'he_normal',padding='same')(p2)
    c3 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',kernel_initializer='
he_normal', padding='same')(c3)
    p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

    c4 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer=
'he_normal',padding='same')(p3)
    c4 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer=
'he_normal',padding='same')(c4)
    p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu',kernel_initializer='
he_normal', padding='same')(p4)
    c5 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu', kernel_initializer=
'he_normal',padding='same')(c5)

    # Expansive path
    u6 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(256, (2, 2),
strides=(2, 2), padding='same')(c5), c4], axis = 3)
    c6 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',kernel_initializer='
he_normal', padding='same')(u6)
```

```
    c6 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',kernel_initializer='
he_normal', padding='same')(c6)

    u7 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(128, (2, 2),
strides=(2, 2), padding='same')(c6), c3], axis = 3)
    c7 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer=
'he_normal',  padding='same')(u7)
    c7 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',kernel_initializer='
he_normal',  padding='same')(c7)

    u8 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(64, (2, 2), s
trides=(2, 2), padding='same')(c7), c2], axis = 3)
    c8 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='
he_normal', padding='same')(u8)
    c8 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='
he_normal', padding='same')(c8)

    u9 = tf.keras.layers.concatenate([tf.keras.layers.Conv2DTranspose(32, (2, 2), s
trides=(2, 2), padding='same')(c8), c1], axis = 3)
    c9 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='
he_normal', padding='same')(u9)
    c9 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='
he_normal', padding='same')(c9)

    outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

    model = tf.keras.Model(inputs=[inputs], outputs=[outputs])

    opt = keras.optimizers.Adam(lr = 1e-
5 , beta_1 = 0.9 , beta_2 = 0.999 , epsilon = 1e-08 , decay = 0.000000199)
    model.compile ( optimizer = opt, loss = 'binary_crossentropy' , metrics = [ 'ac
curacy' ])

    model.summary()


    # ### Treniranje modela

    # In[20]:


    checkpointer = tf.keras.callbacks.ModelCheckpoint('model_for_left_atrium.h5', v
erbose=1, save_best_only = True )

    callbacks = [tf.keras.callbacks.TensorBoard(log_dir='logs')]
```

```
    # In[21]:


    results = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_siz
e = 4, epochs = 25, validation_steps = 2, callbacks=callbacks)


    # ### Predviđanja treniranig modela

    # In[29]:


    idx = random.randint(0, len(x_train))

    preds_train = model.predict(X_train, verbose=1)
    preds_val = model.predict(X_val, verbose=1)
    preds_test = model.predict(X_test, verbose=1)


    # In[30]:


    #uzimanje samo vrijednosti vecih od 0.5 iz [0-1]
    preds_train_t = (preds_train > 0.5).astype(np.float32)
    preds_val_t = (preds_val > 0.5).astype(np.float32)
    preds_test_t = (preds_test > 0.5).astype(np.float32)


    # ### Prikaz rezultata predviđanja

    # In[32]:


    #predviđanje na podatcima za trening
    ix = random.randint(0, len(preds_train_t))

    imshow(X_train[ix], cmap = 'gray')
    plt.show()
    imshow(np.squeeze(Y_train[ix]), cmap = 'gray')
    plt.show()
    imshow(np.squeeze(preds_train_t[ix]), cmap = 'gray')
    plt.show()
```

```
#predviđanje na podatcima za validaciju
ix = random.randint(0, len(preds_val_t))

imshow(X_val[ix], cmap = 'gray')
plt.show()
imshow(np.squeeze(Y_val[ix]), cmap = 'gray')
plt.show()
imshow(np.squeeze(preds_val_t[ix]), cmap = 'gray')
plt.show()

#predviđanje na podatcima za testiranje
ix = random.randint(0, len(preds_test_t))

imshow(X_test[ix], cmap = 'gray')
plt.show()
imshow(np.squeeze(preds_test_t[ix]), cmap = 'gray')
plt.show()


# In[33]:


model.save('model_for_left_atrium.h5')


# ## Dice koeficijent

# In[34]:


def dice_coef(y_true, y_pred, smooth=1):
  intersection = K.sum(y_true * y_pred, axis=[1,2])
  union = K.sum(y_true, axis=[1,2]) + K.sum(y_pred, axis=[1,2])
  dice = K.mean((2. * intersection + smooth)/(union + smooth), axis=0)
  return dice


# In[35]:


y_true = Y_train
preds_train_t = np.squeeze(preds_train_t, axis=(3,))
y_pred = preds_train_t
dice_coeficient = dice_coef(y_true, y_pred, smooth=1)
print(dice_coeficient)
```

```
# In[36]:


y_true = Y_val
preds_val_t = np.squeeze(preds_val_t, axis=(3,))
y_pred = preds_val_t
dice_coeficient = dice_coef(y_true, y_pred, smooth=1)
print(dice_coeficient)
```