Segmentacija epikardijalne masti

Benčević, Marin

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

Sveučilišni studij

SEGMENTACIJA EPIKARDIJALNE MASTI IZ SLIKA RAČUNALNE TOMOGRAFIJE

Diplomski rad

Marin Benčević

Osijek, 2020.

Sadržaj

1. Uvod

Epikardijalna mast (EM) masno je tkivo koje se nalazi između srca i perikarda. Volumen i debljina epikardijalnog masnog tkiva povezane su s različitim bolestima kao što su ateroskleroza, masnoća srca i druge nepogodne srčane promjene [1]. Osim toga, postoje dokazi da je EM dobar nezavisan indikator rizika od srčanih bolesti [2]. U kliničkom okruženju, EM se pretežno mjeri ultrazvukom zbog široke dostupnosti uređaja i relativno kratkog vremena mjerenja. Međutim, CT ili magnetska rezonanca nude značajno preciznija mjerenja nego ultrazvuk. Osim toga, potpuno automatskim i pouzdanim mjerenjem EM iz CT slika mogu se obraditi već postojeći veliki podatkovni skupovi CT slika i time napraviti sistematska studija epikardijalne masti [2].

Segmentacija epikardijalne masti iz CT slika obavlja se najčešće uz pomoć specijalizirane medicinske programske podrške koja zahtjeva interakciju liječnika. Od liječnika se očekuje poznavanje načina rada navedene programske podrške i precizno postavljanje različitih parametara koji bi omogućili uspješnu segmentaciju EM. Takve poluautomatske metode za segmentaciju EM iz CT slika oduzimaju puno vremena te su podložne greškama uzrokovanim razlikama između pristupa dva liječnika. Stoga postoji potreba za razvojem novih, potpuno automatskih algoritama segmentacije EM iz CT slika.

Postoje različita predložena rješenja za automatsko mjerenje epikardijalne masti. Predloženi su poluautomatski pristupi koji kombiniraju metode temeljene na atlasima s klasičnim postupcima obrade slike kao što su filtriranje i s metodama strojnog učenja [3] [4] [5]. Takvi postupci rezultiraju Diceovim koeficijentom od 0,9 i više te snažnim korelacijama (r = 0,97, r = 0,99 i r = 0,91) s ručnom segmentacijom stručnjaka, ali svako mjerenje traje 55 ili više sekundi. Zadnjih godina javljaju se različita rješenja temeljena na dubokom učenju i konvolucijskim neuronskim mrežama [6] [7]. Rješenja temeljena na neuronskim mrežama postižu Diceov koeficijent od 0,84 i više sa snažnom korelacijom (r = 0,974) s rezultatima stručnjaka i znatno su brža od poluautomatskih metoda.

Cilj je ovoga rada razvoj algoritma za semantičku segmentaciju epikardijalnog masnog tkiva iz CT slika srca korištenjem dubokih neuronskih mreža. Osim segmentacije, rad istražuje i najbolje metode predobrade CT slika, metode za rješavanje problema malih skupova podataka kod dubokih neuronskih mreža, utjecaj augmentacije skupa podataka i različite kriterijske funkcije te njihov utjecaj na duboke neuronske mreže.

Ovaj je rad podijeljen na osam poglavlja. U drugom poglavlju dan je pregled postupka i principa rada računalne tomografije. U trećem poglavlju opisana je epikardijalna mast, njena lokacija u tijelu, njezina klinička značajnost i trenutno dostupne metode za segmentaciju epikardijalne masti. Četvrto poglavlje opisuje algoritam razvijen u ovom radu, uključujući dostupan skup podataka i njihovu obradu, arhitekturu korištenog modela te postupak treniranja modela. Implementacija

prema tom opisu prikazana je u petom poglavlju. Šesto poglavlje prikazuje rezultate rada. Na kraju, u sedmom poglavlju, daju se obrazloženje rezultata rada, usporedba s trenutno dostupnim metodama i prijedlozi za unapređenje rada. Konačno, zaključak je dan u osmom poglavlju.

2. Medicinske slike

Proces prikupljanja informacija o specifičnim fiziološkim strukturama poput organa ili tkiva te definiranje njihovih karakterističnih svojstava omogućava prikaz fizioloških struktura u obliku medicinskih slika. S obzirom na navedena karakteristična svojstva razvijene su grane u medicini koje upotrebljavaju različite dijagnostičke metode pregleda pacijenata. Osnovni proces formiranja slika zahtjeva izvor energije koji prikuplja informacije o organu ili tkivu te neki oblik radijacije poput optičkog svijetla, rendgenskih zraka, gama zraka ili zvučnih signala koji stvara interakciju s organima ili tkivom te prikuplja informacije o karakterističnim svojstvima organa odnosno tkiva. Različiti modaliteti i tehnike upotrebljavaju se za razvoj medicinskih uređaja i prikupljanje medicinskih slika. S obzirom na to da će se u ovome radu promatrati obrada medicinskih slika dobivenih računalnom tomografijom, u nastavku slijedi njezin detaljan opis.

2.1. Računalna tomografija

Računalna tomografija (CT) slikovni je medicinski postupak u kojem se rekonstruira tomografirana ravnina nekog dijela tijela [8]. Primarno se koristi u dijagnostici, gdje se snima unutrašnjost tijela bez potrebe za rezanjem tijela. Prilikom snimanja CT uređajem koriste se rendgenski valovi čije valne duljine omogućuju prodiranje kroz meka tkiva. S druge strane tkiva detektori omogućuju detektiranje rendgenskih valova i konstrukciju slike. Ovisno o gustoći tkiva kojeg se snima CT uređajem, dio zračenja se apsorbira ili reflektira, tako da prigušenje zračenja na detektoru odgovara gustoći tkiva. Podaci s detektora se zatim različitim računalnim postupcima prikupljaju te konstruira crno-bijela CT slika snimljenog tkiva. Gušća tkiva (kosti, metal i sl.) prikazuju se svijetlom bojom, dok se rijetka tkiva (zrak, mast i sl.) prikazuju tamnom bojom. Prema tome, u medicini se CT najčešće koristi za različite svrhe:

- pregledavanje pacijenata koji su zadobili unutarnje ozlijede ili traume,
- dijagnosticiranje poremećaja na kostima ili mišićima (prijelomi, tumori),
- lokalizaciju tumora, infekcija ili ugrušaka,
- planiranje kirurških postupaka, biopsije ili radijacijske terapije,
- dijagnostika i praćenje bolesti kao što su srčane bolesti, tumori, bolesti jetre ili pluća,
- praćenje efektivnosti liječenja,
- ostale postupke u liječenju gdje je korisna vizualizacija nekog dijela tijela.

Općenito, CT slika se sastoji od niza uzastopnih presjeka pri čemu je svaki presjek proporcionalan prigušenju rendgenskih zraka tog dijela tijela na skali od -3,071 do 1,024, dok se svaki

presjek CT slike sastoji od matrice piksela. Linearnom transformacijom prigušenja tako da voda ima vrijednost 0, a zrak ima vrijednost -1000 dobiva se mjera u Hounsfieldovoj skali. Hounsfieldova skala korisna je za interpretaciju CT slika jer različite vrijednosti skale odgovaraju različitoj vrsti tkiva ili materijala koji se često pojavljuju u medicinskim slikama [9]. Vrijednosti Houndsfield skale su u jedinici zvanoj Hounsfield jedinica i označavaju se sa *HU* (engl. *Hounsfield unit*). Na primjer, zrak ima vrijednost oko -1,000 HU, masno tkivo oko -100 HU, voda ima vrijednost 0 HU, bijela moždana tvar ima vrijednost oko 25 HU, itd. Različita tkiva i njihove vrijednosti na Hounsfield skali prikazani su u Tablici 2.1.

Materijal	HU vrijednosti
Zrak	-1000
Masno tkivo	-120 do -90
Voda	0
Pluća	-700 do -600
Bubreg	20 do 45
Krv	45 do 65
Jetra	oko 60
Mišično tkivo	35 do 55
Mozak	20 do 45
Strano tijelo	500 i više

Tablica 2.1: HU vrijednosti različitih materijala [10].

Iz volumena dobivenih CT snimanjem može se rekonstruirati bilo koja ravnina tijela. CT slike se najčešće prikazuju presjecima u vodoravnoj (transverzalnoj) ravnini, koja je vodoravna s podom ako se zamisli da pacijent uspravno stoji. Ona dijeli tijelo na kranijalni i kaudalni kraj. Osim vodoravne, često se još koriste i dvije ravnine okomite na nju: sagitalna i frontalna (koronalna) ravnina. Sagitalna ravnina pruža se duž tijela i dijeli tijelo na lijevu i desnu stranu. Frontalna ravnina pruža se duž tijela i dijeli tijelo na ventralni i dorzalni (prednji i stražnji) dio. Glavne ravnine tijela prikazane su na Slici 2.1.

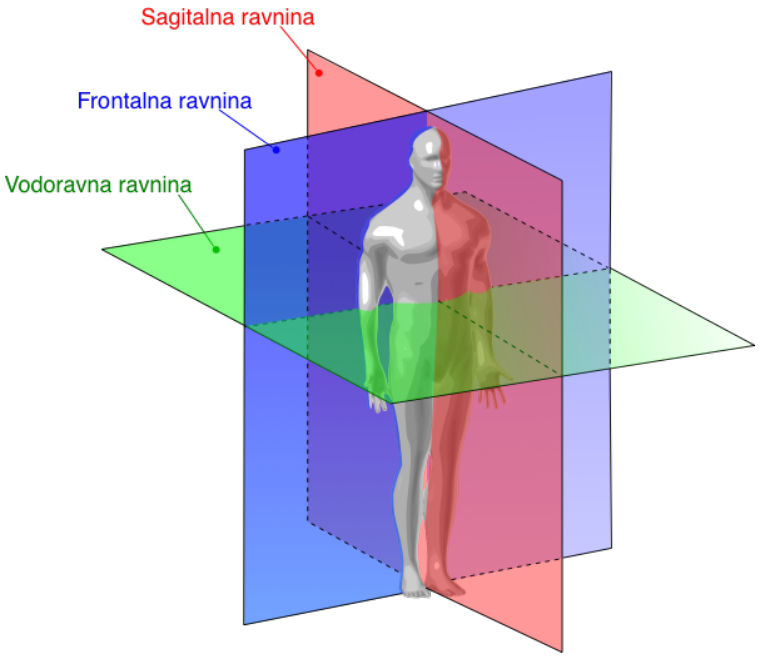
CT snimanje je u većini slučaja brži i jeftiniji postupak od magnetske rezonance, ali slike dobivene magnetskom rezonancom nude nešto više detalja nego slike CT-a. CT snima niz presjeka objekta iz različitih kutova, što omogućuje rekonstrukciju trodimenzionalnog volumena objekta različitim postupcima obrade slike [8], za razliku od rendgenskog snimanja koje češće rezultiraju u jednoj dvodimenzionalnoj slici. Kod CT snimanja izloženost zračenju veća je nego kod magnetske rezonance. Iako rendgenski valovi spadaju u ionizirajuće zračenje, količine zračenja prilikom postupka CT snimanja pretežno su dovoljno niske da ne dovode do dugotrajnih negativnih posljedica.

2.2. Postupak CT snimanja

CT slike stvaraju se pomoću rendgenskih valova, točnije valovi valnih duljina između 10 nm i 0,01 nm. Rendgenski valovi stvaraju ionizirajuće zračenje koje je dovoljno malih valnih duljina da može prodirati kroz razne objekte, uključujući ljudsko tkivo. Rendgenske valove generira rendgenska cijev, odnosno vakuumska cijev koja sadrži katodu i anodu. Katoda usmjerava elektrone u vakuum koji se sudaraju s anodom na drugom kraju cijevi. Prilikom sudara, dio energije oslobađa se kao rendgensko zračenje. Površina anode s kojom se elektroni sudaraju zakrivljena je tako da oslobođene rendgenske zrake napuste cijev okomito na nju. Ostatak energije oslobađa se u obliku topline, tako da generatori rendgenskih valova često sadrže i sustave za hlađenje.

Generatorima se upravlja razinom napona i jakosti struje. Napon rendgenske cijevi određuje količinu proizvedenih rendgenskih valova i utječe na energiju fotona tijekom snimanja, tako da se povećanjem napona smanjuje prigušenje rendgenskog zračenja očitanog na detektoru. Povećanjem napona smanjuje se šum na slici ali može dovesti do promijene u očitanim vrijednostima na Hounsfield skali. U medicini se koriste vrijednosti napona oko 100 kV. Promjenom jakosti struje rendgenske cijevi također se određuje količina zračenja koju rendgenska cijev proizvodi (broj fotona u sekundi). Dok udvostručavanjem napona povećavamo količinu zračenja za oko četiri, jakost struje rendgenske cijevi ima više linearnu vezu s količinom proizvedenog zračenja. Povećanjem struje rendgenske cijevi smanjuje se količina šuma bez utjecaja na očitane vrijednosti na Hounsfield skali. U medicini se pretežno koriste vrijednosti struje između 0,5 i 0,01 mA.

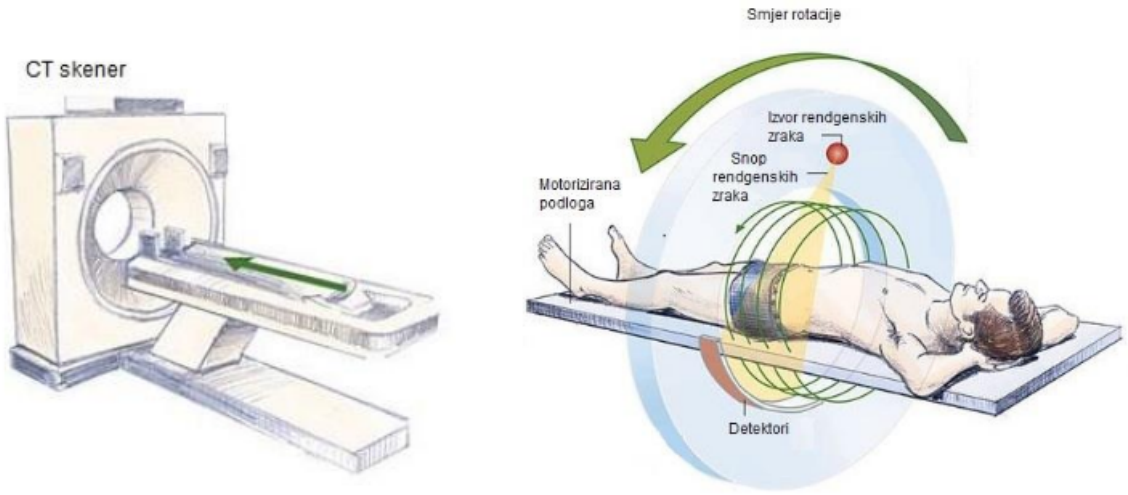
Prilikom standardnog rendgenskog snimanja, rendgenski valovi iz generatora prodiru kroz tkivo na drugu stranu. S druge strane tkiva nalaze se detektori rendgenskih valova koji očitavaju



Slika 2.1: Glavne anatomske ravnine tijela [11].

količinu rendgenskog zračenja. Tkiva apsorbiraju i reflektiraju zračenje ovisno o svojoj gustoći, tako da vrijednost očitana na detektoru ovisi o gustoći tkiva. Iz vrijednosti na detektoru moguće je rekonstruirati dvodimenzionalnu sliku na kojoj su područja visoke gustoće prikazana svjetlijom bojom, dok su područja niske gustoće prikazana tamnijom bojom.

CT uređaj upotrebljava veoma sličan princip rada kao i rendgensko snimanje. Kod CT-a, i dalje se koriste generatori i detektori rendgenskog snimanja, ali umjesto statičkog položaja, generatori i detektori kod CT-a rotiraju oko objekta. Pri tome se pacijent nalazi unutar cilindričnog uređaja, a unutar samog cilindra nalaze se generatori rendgenskih valova koji rotiraju oko pacijenta [8]. S druge strane pacijenta smješten je detektor rendgenskih valova koji zapisuje količinu zračenja u obliku sinograma. Rotiranjem oko pacijenta moguće je snimiti svaki kut unutrašnjosti tijela. Kada se skupi dovoljno podataka iz različitih kutova snimanja, koristi se računalni postupak tomografske rekonstrukcije da bi se mjerenja zračenja iz detektora pretvorila u niz slika presjeka objekta, gdje svaki piksel presjeka ima vrijednost relativne gustoće zračenja. Tako računalna tomografija omogućuje snimanje unutrašnjosti objekata bez potrebe za rezanjem objekta.Ilustrativni prikaz rada CT uređaja prikazan je na Slici 2.2.



Slika 2.2: Ilustrativni prikaz rada CT uređaja [12].

2.3. DICOM zapis CT slike

Gotovo sve metode za prikupljanje medicinskih slika zahtijevaju računalnu obradu. Računala se koriste za prikaz medicinskih slika kao i za izradu 3D modela iz ulazne serije podataka. Premda postoje različiti smjerovi u razvoju opreme za prikaz medicinskih slika došlo je do potrebe za razvojem standarda koji će pružiti komunikaciju i razmjenu informacija između različitih medicinskih uređaja. U tu svrhu razvijen je DICOM (engl. *Digital Imaging and Communications in Medicine*), međunarodni standard za prijenos, spremanje, dohvaćanje i prikazivanje medicinskih

slika. DICOM koriste bolnice i druge zdravstvene te znanstvene organizacije diljem svijeta i podržavaju ga gotovo svi uređaji za različite postupke medicinskog slikanja, pa se tako i CT slike pretežno spremaju upravo u datotečni format DICOM.

Glavna je prednost DICOM-a, osim njegove široke primjene, mogućnost spremanja različitih podataka o pacijentu i vrijednosti samih piksela slike u jednu datoteku [13]. DICOM datoteka sastoji se od više atributa kao što su ime, dob, spol, identifikacijski broj pacijenta i slično te jednog posebnog atributa koji sadrži piksele. Taj atribut može dodatno biti podijeljen na niz slika i tako sadržavati više različitih slika, više presjeka jednog slikanja ili promjenu slike kroz vrijeme. DICOM standardom, između ostalog, pokriveni su:

- mrežni komunikacijski protokoli,
- komunikacija putem različitih medija,
- funkcionalne aplikacijske usluge,
- konzistentan prikaz slika na različitim uređajima,
- sigurnost i upravljanje postavkama,
- izgled prikaza kojeg može definirati liječnik,
- identifikacija i grupiranje srodnih informacija.

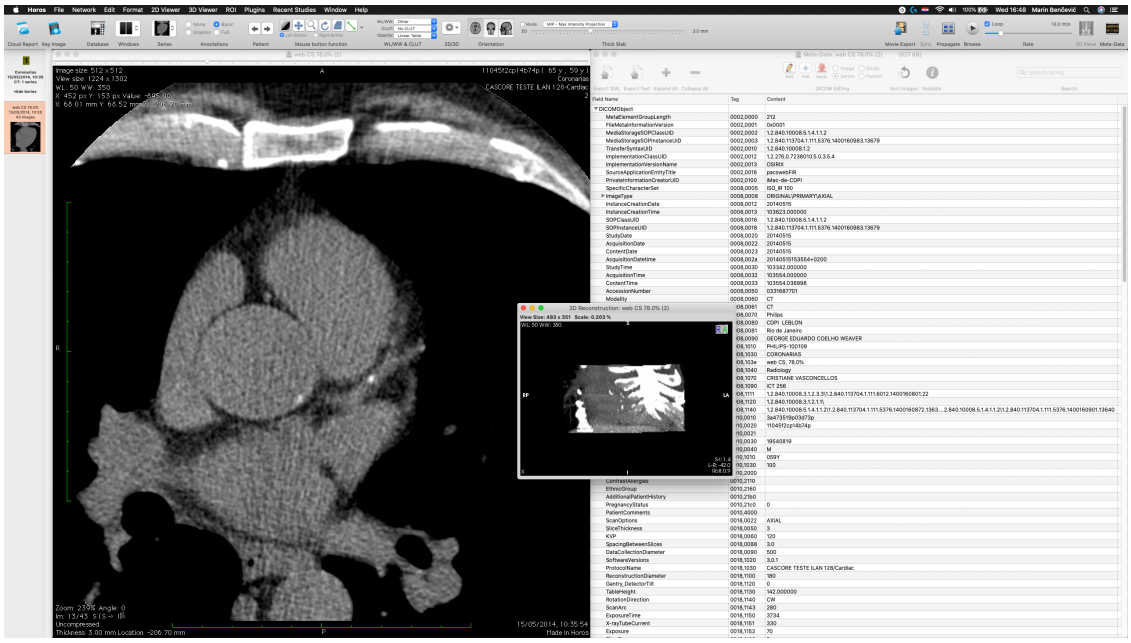
S druge strane, DICOM ne definira specifičnosti implementacije kao što su baze podataka, programski jezici, operacijski sustavi i sl., kao ni koje podatke je moguće koristiti. Jedna DICOM datoteka obično se sastoji od sljedećih dijelova:

- informacije o pacijentu (identifikacijski broj, dob, spol i sl.),
- informacije o studiji (datum studije, ime voditelja i sl.),
- informacije o seriji (broj serije, redni broj, modalnost),
- informacije o opremi (proizvođač, model i sl.),
- informacije slike (podaci piksela, broj bitova, broj redova i stupaca, rubovi prozora, broj slike, tip slike i sl.)

Svaki od tih dijelova podijeljen je u module, a svaki modul sastoji se od atributa. Svaki atribut ima svoje ime definirano DICOM standardom koje služi kao ključ za dohvaćanje vrijednosti povezane s tim imenom. Posebno važan dio DICOM standarda predstavlja dio koji definira mrežnu komunikaciju. S obzirom da se DICOM slike prenose s opreme (npr. CT uređaja) na računalo te dalje s računala u različite baze podataka, poslužitelje, pisače ili prijenosne medije važno je osigurati brzu i sigurnu komunikaciju te prijenos podataka između navedenih uređaja. DICOM standard definira sve dijelove takvih oblika prijenosa DICOM datoteke. Načini prijenosa DICOM datoteka definirani su funkcijskim DICOM aplikacijskim uslugama odnosno SOP klasama. Primjeri SOP usluga su klasa usluge spremanja, klasa usluge upravljanja ispisivanjem, klasa usluge upravljanjem

studije i druge. Svaka klasa usluga ima pravila ponašanja koja su definirana DICOM standardom. Svaki uređaj koji podržava DICOM mora pratiti standard.

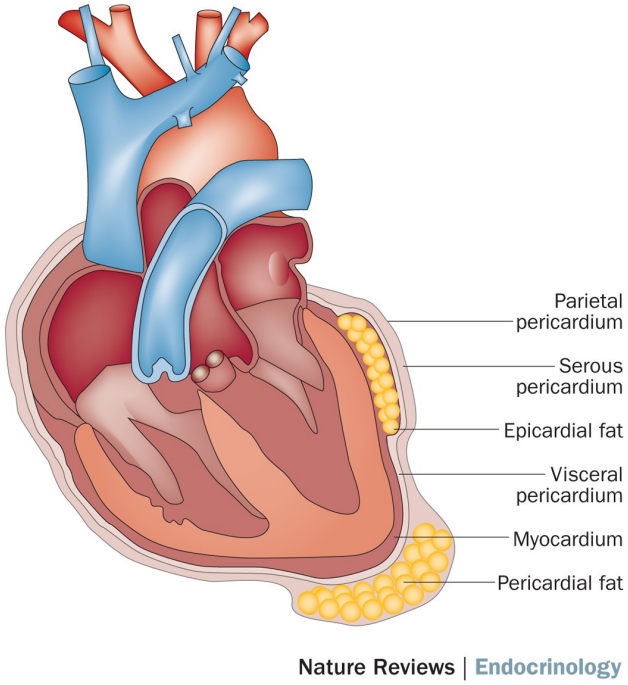
DICOM standard podržava različite tipove slika za različite medicinske primjene. U računalnoj tomografiji često se koriste slike u boji dok je u rendgenskoj radiografiji naglasak na intenzitetu slike. Isto tako, različite medicinske primjene zahtijevaju i različitu razinu kvalitetu slika. Kompresija slika oslanja se na kompresijske standarde poput JPEG, JPEG 2000 ili MPEG-2 za višedimenzionalne nizove slika. DICOM slike namijenjene su za prikaz na različitim radnim stanicama te na osobnim računalima. Slike mogu biti u boji ili u sivim tonovima. Informacije o dubina bita i način kompresije sadržani su u zaglavlju slike. Na taj način je osiguran pravilan prikaz slike bez obzira na proizvođača opreme. Kako bi se pacijentu, ali i kliničkim radnicima, omogućio jednostavan prikaz i rad s DICOM slikama na osobnim računalima razvijeni su različiti DICOM alati i programi poput OsiriX ili Horos te programske biblioteke kao Pydicom napravljene posebno za rad s DICOM datotekama. Prema tome, DICOM je dokazano izuzetno koristan standard za pohranu i razmjenu medicinskih slika. Na Slici 2.3 prikazan je primjer pregleda DICOM datoteke u programu Horos.



Slika 2.3: Prikaz CT slike pomoću programa Horos.

3. Epikardijalna mast

Epikardijalna mast je visceralno masno tkivo koje se nalazi između srca i perikarda, odnosno između trbušnog perikarda i miokarda [14]. Epikardijalna mast je raspoređena u različitim dijelovima oko srca, ali najveće količine nalaze se oko atrioventrikulske i interventrikulse brazde te na zidu desne klijetke kao što je prikazano na Slici 3.1. Epikardijalna mast se razlikuje u biokemijskom i molekularnom smislu od okolnog masnog tkiva kao što je perikardijalna mast. Neke od uloga epikardijalne masti su distribucija i regulacija lokalnog krvotoka, zaštita miokardija i srčanih arterija od upala i patogena, mehanička zaštita, a ona je i izvor masnih kiselina miokardiju [14].



Slika 3.1: Lokacija epikardijalne masti (engl. *epicardial fat*) [1].

3.1. Mjerenje epikardijalne masti

Epikardijalna mast u kliničkim uvjetima često se mjeri surogat vrijednostima kao što su opseg struka ili omjer opsega struka i kuka. Izravno mjerenje epikardijalne masti najčešće se obavlja ultrazvukom srca, a zatim CT-om i magnetskom rezonancom. Izravna su mjerenja preciznija, ali često nepraktična za kliničku upotrebu. Zato postoji velika potreba za što jednostavnijim metodama mjerenja epikardijalne masti [15].

Epikardijalna mast se kvantificira na dva načina; kao volumen epikardijalne masti ili kao debljina epikardijalnog masnog tkiva. U slučaju volumena, računa se ukupan volumen masnog tkiva

koji spada unutar biokemijske i anatomske definicije epikardijalne masti. Kod debljine, često se mjeri debljina tkiva epikardijalne amsti okomito na zid desne klijetke pred kraj srčane kontrakcije [16].

Kod magnetske rezonance epikardijalna se mast određuje kao tkivo između perikardija i srčanog zida. Korištenjem Simpsonove metode iz magnetske rezonance moguće je odrediti ukupan volumen epikardijalne masti s koeficijentom varijabilnosti promatrača od 5,9%, ili debljinu epikardijalne masti sa koeficijentom varijabilnosti promatrača od 13,6% [14]. Nadalje, mjerenje volumena i debljine epikardijalne masti iz CT slike korisno je zbog vrlo čestog postupka mjerenja ovapnjenja koronarnih arterija (engl. *calcium scoring*) gdje se CT uređajem slika srce. U tom je slučaju lakše izmjeriti volumen i debljinu EM iz već postojeće slike. Učestalost postupka mjerenja ovapnjenja dovela je i do brojnih velikih skupova podataka CT slika srca. Automatska kvantifikacija EM može olakšati izradu sistematskih studija epikardijalne masti na tim skupovima podataka.

Istraživanja pokazuju da je moguće precizno utvrditi debljinu i volumen epikardijalne masti iz CT slika postupkom mjerenja ovapnjenja ako CT uređaj ima 16 ili više detektora [17]. Iako je preciznije određivati količinu epikardijalne masti pri tanjim presjecima, moguće ju je odrediti i pri presjecima od 3 mm koji su česti kod slikanja za potrebe mjerenja ovapnjenja. Granice epikardijalne masti određuju se ručno, a računalnim se metodama određuju volumen i debljina. Zbog dijela mjerenja koji se obavlja ručno, mjerenje epikardijalne masti iz CT slika je dugotrajan proces kojeg nije moguće obaviti sistematski na velikim količinama slika bez potpuno automatiziranog rješenja.

3.2. Klinička značajnost

Epikardijalna mast razlikuje se od okolne perikardijalne masti ne samo po biokemijskom sastavu, lokaciji i izvoru krvotoka, nego i po kliničkim svojstvima. Epikardijalna je mast bitna zbog svoje blizine miokardiju i zbog činjenice da ta dva tkiva dijele isti mikrokrvotok. Provedena istraživanja upućuju na to da je epikardijalna mast aktivni endokrini organ koji ima interakciju s miokardijom [18]. Takva interakcija za perikardij nije poznata.

Precizna mjerenja debljine ili volumena epikardijalne masti povezana su s masnoćom srca, koronarnom bolesti, metaboličkim sindromom, masnoćom jetre i drugim nepogodnim srčanim promjenama. Epikardijalna mast je pouzdan marker za rizik od kardiovaskularnih bolesti i koristan faktor kod predviđanja efikasnosti lijekova koji moduliraju masno tkivo [1]. Isto tako, volumen i debljina epikardijalne masti imaju izravan utjecaj na aterosklerozu i nepovoljne kardiovaskularne događaje kao što su infarkti, zatajenje srca, ili moždani udar [2]. Moguće je da sistematske kvantifikacije i istraživanje epikardijalne masti mogu poboljšati procjenu rizika kod asimptomatskih pacijenata bez prijašnjih bolesti srčanih arterija [19].

4. Duboke konvolucijske neuronske mreže

Umjetne neuronske mreže predstavljaju podskup algoritama i metoda za obavljanje postupka strojnog učenja. Za razliku od klasičnog strojnog učenja, kod neuronskih mreža nije potrebno modelirati matematičku funkciju podataka unaprijed, nego struktura neuronskih mreža omogućuje učenje kompleksnih matematičkih funkcija na temelju ulaznih podataka. Umjetne neuronske mreže temelje se na imitaciji procesa i načina rada ljudskoga mozga koji uspješno obrađuje velik broj kompleksnih informacija. Umjetne neuronske mreže pokazale su se kao dobar alat za rješavanje klasifikacijskih problema u području medicinske obrade slike. Njihova snaga primarno leži u slojevitom (odnosno slijednom) kombiniranju jednostavnih nelinearnih elemenata. U sljedećim potpoglavljima detaljnije je pojašnjen princip rada dubokih neuronskih i konvolucijskih mreža.

4.1. Duboke neuronske mreže

Umjetna neuronska mreža sastoji se od skupa međusobno povezanih umjetnih neurona. Svaki neuron je matematička funkcija koja preslikava jednu ili više ulaznu matricu u izlaznu matricu tako što se prvo sumiraju sve ulazne matrice pomnožene sa svojim težinama, a zatim suma prolazi kroz aktivacijsku funkciju. Umjetna neuronska mreža je mreža gdje su međusobno spojeni umjetni neuroni, a svaki spoj ima odgovarajuću težinu koja predstavlja važnost tog spoja. Kod umjetne neuronske mreže, ulaz u prvi neuron su podaci na kojima se vrši predikcija, dok je izlaz iz zadnjeg neurona sama predikcija. Arhitektura mreže (broj, način spajanja i vrsta neurona) određuje oblik i kvalitetu izlaza.

Duboke neuronske mreže su umjetne neuronske mreže koje imaju velik broj neurona koji su međusobno povezani u velik broj slojeva. Svaki sloj progresivno uči značajke ulaza sve većeg reda. Na primjer, kod obrade slike prvi slojevi uče detekciju rubova i jednostavnih oblika, dok zadnji slojevi uče kompleksne značajke kao što su lica, znamenke i sl. Duboke neuronske mreže sastoje se od ulaznog sloja, jednog ili više skrivenih slojeva te zadnjeg izlaznog sloja.

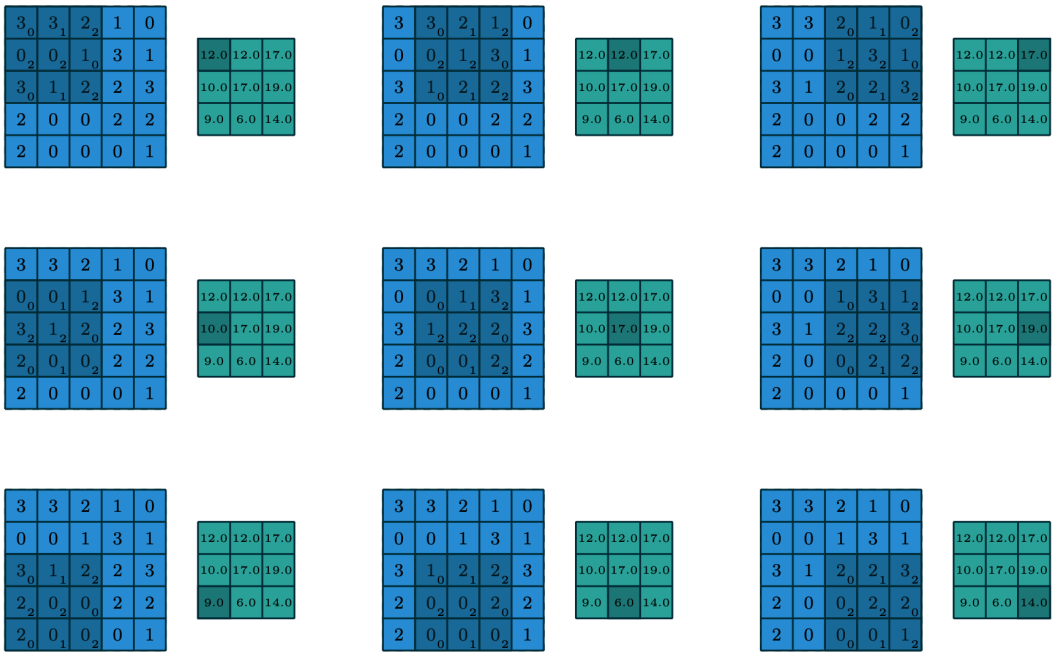
4.2. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (KNM) klasa su dubokih neuronskih mreža koje koriste konvoluciju umjesto množenja matrice u barem jednom neuronu. Pretežno se primjenjuju kod obrade slike, prepoznavanja objekata na slici ili videu, sustava za preporuke, klasifikacije slika, analize medicinskih slika, obrade prirodnog jezika itd.

KNM su regularizirane varijante višeslojnih perceptron mreža. Perceptron mreže su umjetne neuronske mreže gdje je svaki neuroni jednog sloja povezan sa svakim neuronom drugog sloja,

te se mreže nazivaju potpuno povezane mreže. Potpuna povezanost može dovesti do problema pretjerane prilagodbe mreže ulaznim podacima (engl. *overfitting*). KNM rješavaju taj problem tako što se u početnim slojevima uče vrlo grube značajke slike, a kasniji slojevi spojem grubljih značajki grade kompleksnije značajke.

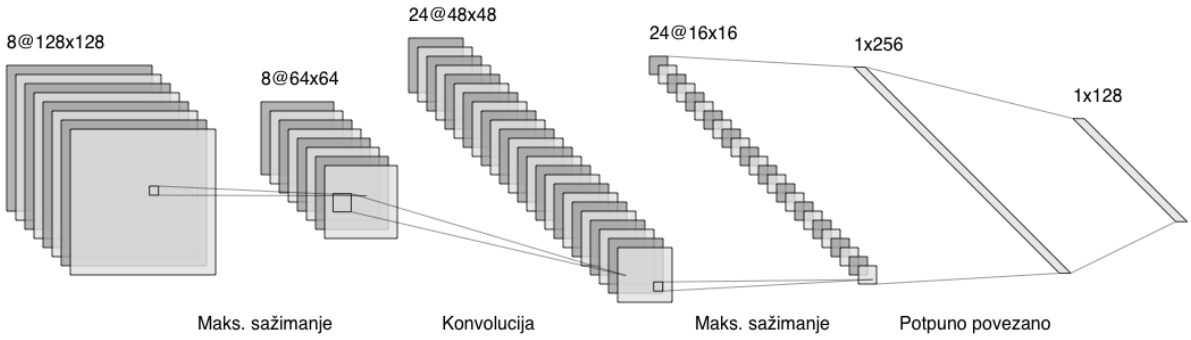
Općenito, konvolucija je matematička operacija nad dvije funkcije koja proizvede treću funkciju koja sadrži informacije o tome kako je oblik jedne funkcije modificiran drugom. Kod konvolucijskih neuronskih mreža najčešće se koristi diskretna konvolucija dvodimenzionalnih matrica [20]. U tom slučaju, konvolucija se radi nad ulaznom matricom i matricom koju zovemo jezgra (engl. *kernel*). Na jezgru možemo gledati kao da kliže preko ulazne matrice. U svakom koraku, svaki element ulazne matrice množi se s elementom jezgre koji ga preklapa i svi umnošci se zbrajaju. Ta suma predstavlja jedan element u matrici koja je konvolucija ulazne matrice i jezgre. Taj se proces ponavlja za svaki element ulazne matrice. Na Slici 4.1 prikazan je primjer konvolucije.



Slika 4.1: Primjer koraka diskretne konvolucije dvodimenzionalnih matrica. Plavom bojom označena je ulazna matrica, tamno plavom bojom označena je jezgra, dok je zelenom bojom označen rezultat konvolucije [20].

Dvodimenzionalna konvolucija može se proširiti na N-dimenzionalne matrice tako što se koristi N-dimenzionalna jezgra (filter) koja klizi u svim dimenzijama ulazne matrice. Drugim riječima, na N-dimenzionalni konvoluciju može se gledati kao niz dvodimenzionalnih konvolucija između svakog kanala ulazne matrice i odgovarajućeg kanala filtera. Skriveni slojevi unutar konvolucijskih neuronskih mreža pretežno se sastoje od nizova konvolucijskih slojeva, svaki od kojih obavlja operaciju konvolucije, s ReLU aktivacijskom funkcijom nakon koje slijede dodatni slojevi kao što

su slojevi sažimanja (engl. *pooling layers*), potpuno povezani slojevi ili slojevi normalizacije [21]. Često se svakom uzastopnom konvolucijom povećava dubina mape značajki, ali se smanjuju širina i visina mape. Zadnjih nekoliko skrivenih slojeva su često potpuno povezani slojevi, prije kojih je mapa značajki zbrajanjem "spljoštena" u jednostupčanu matricu. Konvolucijske neuronske mreže često imaju strukturu nalik strukturi prikazanoj na Slici 4.2.



Slika 4.2: Primjer tipične arhitekture konvolucijskih neuronskih mreža [21].

Kod KNM, parametri filtera, odnosno vrijednosti unutar konvolucijskih jezgri naučeni su tijekom treniranja mreže. Rezultat konvolucijskog sloja je matrica promijenjenih dimenzija koja se naziva mapa značajki. Takva matrica većinom prikazuje značajke ulazne matrice koje je mreža naučila prepoznati na slici. KNM često sadrži slojeve sažimanja. Mapa značajki prepoznaje vrlo granularne značajke na ulaznoj slici, uključujući njihov oblik i položaj. Pretjerano učenje značajki ili lokacija značajki specifičnih za skup podataka za trening dovodi do pretjerane prilagodbe modela skupu za trening. Da bi se to spriječilo, nakon aktivacijske funkcije konvolucijskog sloja ubacuje se dodatni sloj sažimanja. Taj sloj smanji širinu i visinu mape značajki, ali ne mijenja dubinu mape.

Slično kao kod konvolucije, sažimanje koristi prozor koji klizi mapom značajki, ali za razliku od konvolucije sadržaj svakog prozora prolazi kroz funkciju sažimanja [20]. Faktori koji utječu na veličinu sažete matrice su širina i visina prozora, te korak, odnosno za koliko se piksela pomakne prozor u svakom koraku. Pretežno se koristi veličina 2×2 s korakom 2, koja smanji širinu i visinu mape značajki za faktor 2. Najčešće funkcije sažimanja su prosjek (engl. *average pool*) ili maksimum (engl. *max pool*). Kod prosjeka, element rezultantne matrice za svaki korak prosjek je vrijednosti svih piksela koji se nalaze unutar prozora. S druge strane, kod maksimuma element sažete matrice svakog koraka vrijednost je maksimalnog piksela unutar prozora. Kod KNM koje obavljaju zadatak prepoznavanja objekata ili klasifikacije slika, češće se koristi maksimum jer maksimalne vrijednosti mape značajki predstavljaju najvažnije značajke. Dodavanjem slojeva sažimanja smanjuje se pretjerana prilagodba modela skupu za trening, model postaje više invarijantan na tranzlaciju značajki, te se smanjuju računalni zahtjevi za treniranje i predikciju jer se smanjuje broj parametara modela (veličina mapa značajki).

5. Segmentacija epikardijalne masti pomoću duboke konvolucijske neuronske mreže

Rješenje koje predlaže ovaj rad sastoji se od razvoja duboke konvolucijske neuronske mreže čiji je zadatak semantička segmentacija epikardijalne masti iz svakog pojedinog presjeka CT slike. Mreža je trenirana na presjecima CT slika srca na kojima su ručno označene površine epikardijalnog masnog tkiva. Pretpostavka je ovog rada da se uz dovoljno dobru semantičku segmentaciju EM na svakom presjeku CT slike računalnim metodama može precizno izračunati ukupni volumen epikardijalne masti na cijeloj CT slici.

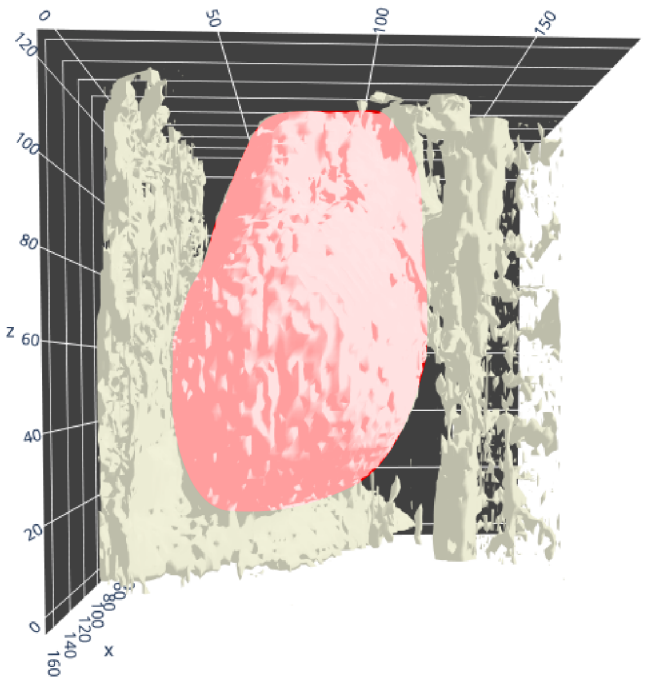
5.1. Skup podataka

Podatkovni skup koji predstavlja referentne vrijednosti za segmentaciju dobiven je od CT-a prsa 20 različitih pacijenata (10 muškaraca i 10 žena) iz Rio De Janeira [22]. Demografija pacijenata prikazana je u Tablici 5.1. Svakom je pacijentu snimano područje srca. Snimani volumen jednog pacijenta prikazan je na Slici 5.1. Slike su spremljene u DICOM datoteke bez informacija koje osobno identificiraju pacijente. Pacijenti su slikani na dva različita CT uređaja. Podatkovni skup dostupan je na [23]. Primjer jednog presjeka prikazan je na Slici 5.2.

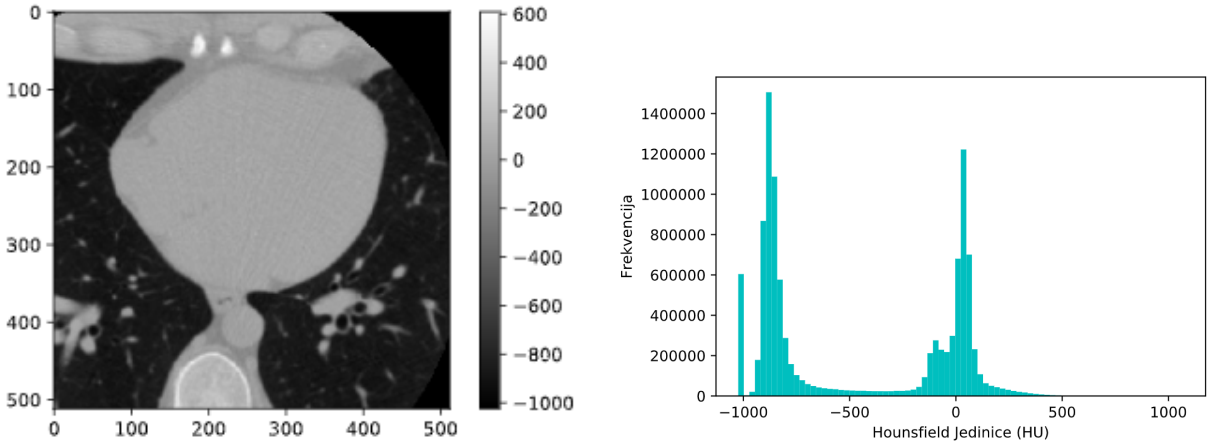
Spol	Muškarci 10	Žene 10
Dob	Prosjek 55,4 ($\pm$22)	Medijan 53
Volumen EM (ml)	Prosjek 97,9 Minimum 39,2	Medijan 94,8 Maksimum 203,4
Broj presjeka	Ukupno 878	Po pacijentu (prosjek) 42
Uređaji	Phillips pacijenata 9	Siemens pacijenata 11

Tablica 5.1: Demografija podatkovnog skupa [22].

U svakom od 878 presjeka CT slika ručno je označeno područje epikardijalne masti uz pomoć liječnika i računalnog znanstvenika. Označeno područje spremljeno je u zasebnu DICOM datoteku

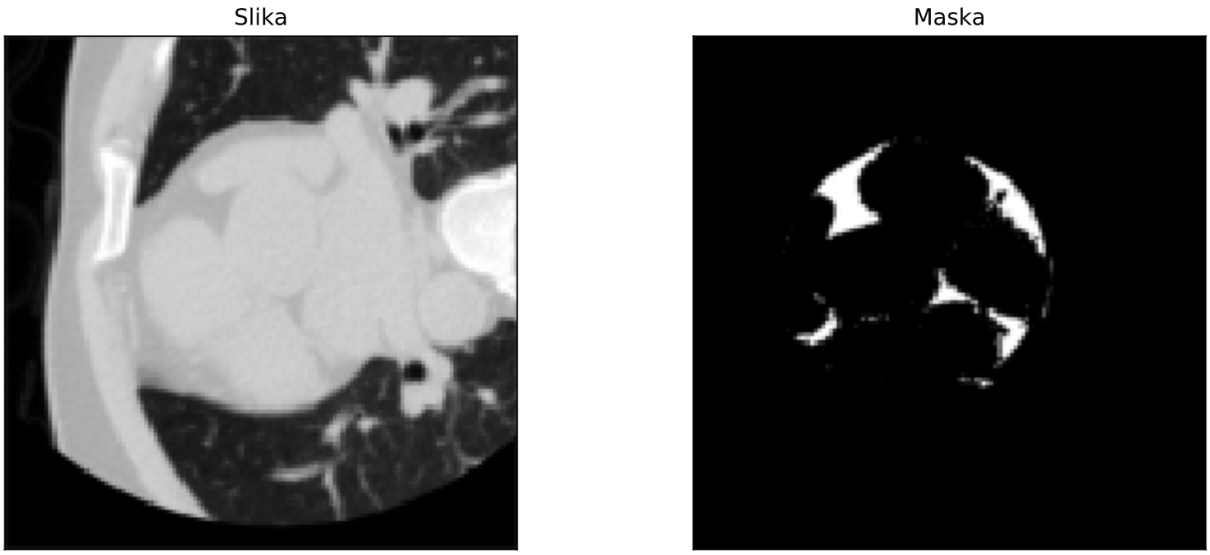


Slika 5.1: Bočni prikaz snimanog CT volumena jednog pacijenta. Prsa se nalaze s lijeve strane slike, leđa s desne, a srce je označeno crvenom bojom.



Slika 5.2: Prikaz jednog presjeka podatkovnog skupa (lijevo) te histogram HU vrijednosti presjeka (desno).

koja predstavlja masku epikardijalne masti na izvornoj CT slici. Na maski vrijednost piksela 1 predstavlja epikardijalnu mast, dok vrijednost 0 predstavlja ostalo područje presjeka. U izvornom podatkovnom skupu iz [22] osim EM segmentirano je i masno tkivo medijastinuma. Za potrebe ovoga rada taj je dio maske zanemaren, a ostavljene su samo vrijednosti za područje epikardijalne masti. Na Slici 5.3 prikazan je primjer maske jednog presjeka podatkovnog skupa.



Slika 5.3: Prikaz jednog CT presjeka izvorne slike i njegove pripadajuće maske.

5.2. Pretprocesiranje podataka

Podatkovni skup sastoji se od 878 DICOM datoteka izvornih CT slika, kao i 878 DICOM datotka maski svake CT slike. Vrijednosti piksela slike su oblika $broj\ presjeka \times 512 \times 512$. Izvorne CT slike su prvo učitane kao matrica dimenzija $(broj\ presjeka, 512, 512)$. Broj presjeka po pacijentu varira između 36 i 48. Vrijednosti piksela svakog presjeka pretvorene su u vrijednosti koje odgovaraju Hounsfield skali matematički opisanom Formulom 5.1:

$$H_i = I_i \cdot m_i + b_i \tag{5.1}$$

pri čemu je H_i i-ti presjek slike u Hounsfieldovoj skali, I_i i-ti presjek izvorne datoteke, m_i je nagib skaliranja presjeka, a b_i je odsječak skaliranja presjeka. Vrijednosti m_i i b_i spremljene su u DICOM datotekama pod oznakama *Rescale slope (0028,1053)* i *Rescale intercept (0028,1052)* [13]. Nakon toga su ulazne slike normaliziririrane i centrirane oko nule pomoću formula 5.2 i 5.3.

$$N_i = \frac{H_i + -1000}{300 + -1000} \tag{5.2}$$

$$C_i = N_i - s \tag{5.3}$$

pri čemu je N_i i-ti presjek normalizirane slike, C_i i-ti presjek centrirane slike, a s je srednja vrijednost svih piksela N. Vrijednosti -1000 i 300 predstavljaju minimalnu i maksimulanu vrijednost piksela presjeka u Hounsfield skali, te su određene iz histograma prikazanog na Slici 5.2.

Nakon centriranja i normalizacije, presjeci su umanjeni na dimeziju 128×128. Rezultat pretprocesiranja je ulazna slika dimenzija 128×128 čije vrijednosti piksela predstavljaju skalirane

veličine Hounsfieldove skale između $(-1000\ hu, 300\ hu)$ s vrijednostima u intervalu $(-0.5, 0.5)$ i srednjom vrijednosti 0. Treniranje modela na intervalu $(0, 1)$ pokazalo je veću grešku na validacijskom skupu nego treniranje na intervalu $(-0.5, 0.5)$.

Posljednji je korak pretprocesiranja uklanjanje irelevantnih slika. Za potrebe ovog rada, svi presjeci čije odgovarajuće maske ne sadrže niti jednu pozitivnu vrijednost uklonjeni su iz skupa ulaznih presjeka i skupa maski. Uz to, za neke presjeke iz izvornog skupa podataka nema odgovarajuće maske, tako da su ti presjeci uklonjeni iz skupa ulaznih slika. Ukupno je uklonjeno 112 presjeka. Pretpostavka rada je da je treniranje efektivnije bez ovih primjera, a da je potrebno prije predikcije modelom iz ovog rada obaviti grubu klasifikaciju ulazne slike kako bi se odredilo postoji li na slici tkivo epikardijalne masti ili ne.

5.3. Augmentacija podataka

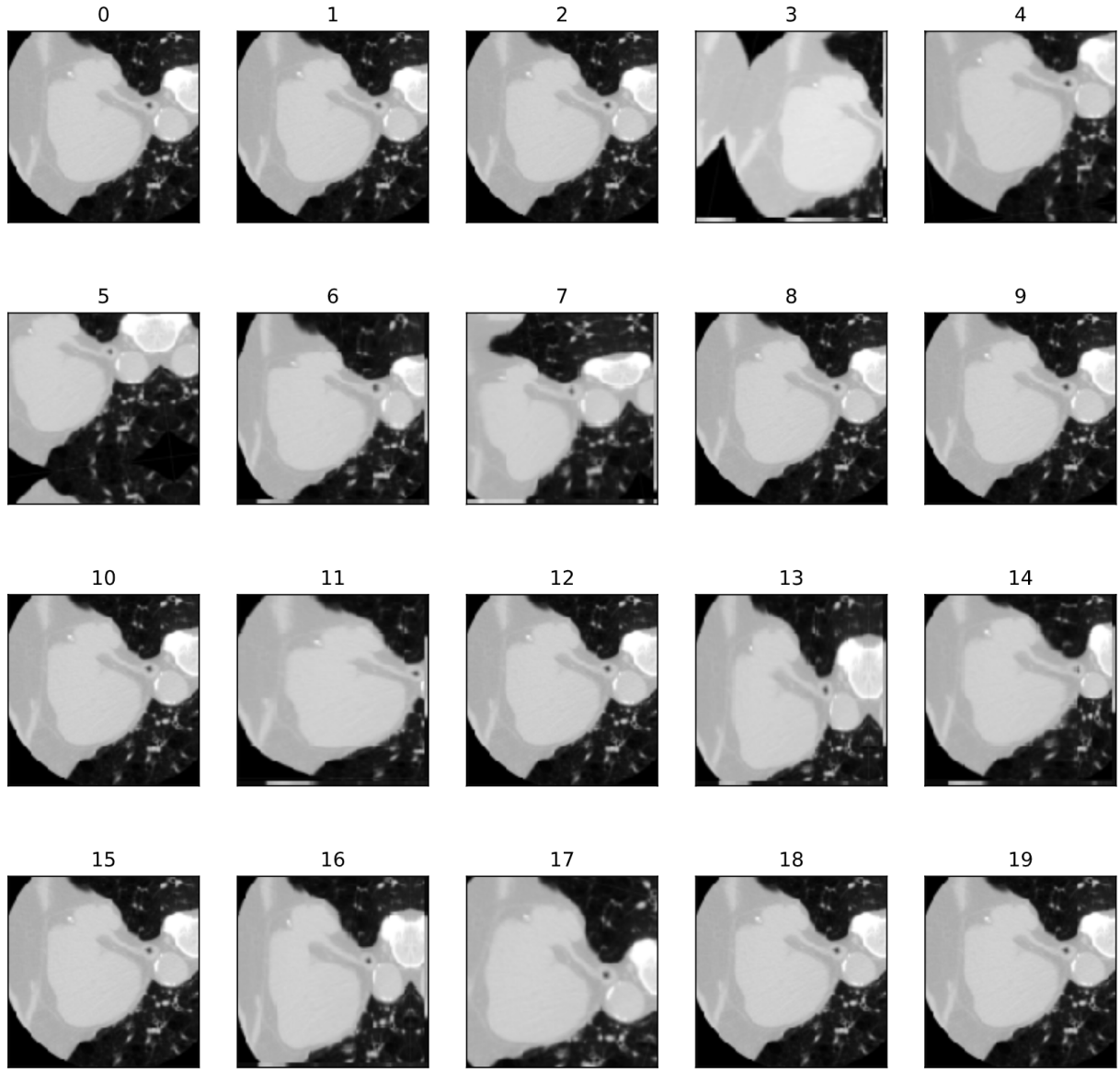
Postoji mnogo razloga zbog kojih se CT slike istog područja tijela mogu razlikovati jedna od druge. Tijekom postupka računalne tomografije pacijent i uređaj fiksirani su koliko je to moguće izvesti u kliničkim uvjetima, ali unatoč tome nužno dolazi do manjih pomaka pacijenta ili opreme. Osim toga, različitim metodama postavljanja CT opreme isto područje unutar tijela može biti oslikano različitim debljinama presjeka i različitom veličinom snimanog volumena. Na kraju, položaj i veličina samog tkiva ili organa koji se snima mogu se razlikovati od pacijenta do pacijenta [24].

Da bi neuronska mreža bila primjenjiva na što većem broju slika, mora biti neovisna o takvim manjim razlikama među slikama. U tu svrhu koristi se augmentacija podataka. Tijekom treniranja, dio ulaznih slika koji se koristi za treniranje augmentira se različitim metodama obrade slike kako bi se postigla šira razdioba ulaznih podataka te tako postigao općenitiji model i smanjio efekt pretjerane prilagodbe modela ulaznim podacima (engl. *overfitting*).

U ovom se radu koristi nekoliko metoda augmentacije podataka. Prvo, svaka ulazna slika ima vjerojatnost od 30% za primjenu neke kombinacije afinih transformacija slike koje uključuju pomak za do 20% širine slike, skaliranje za do 20% trenutne veličine i rotaciju za do 15 stupnjeva. Svrha je tih augmentacija simuliranje promjene u položaju opreme, pacijenta ili organa u realnim kliničkim uvjetima [24]. Na svakoj slici u vjerojatnosti od 20% primijenjena je prostorna deformacija slike. Ovaj korak simulira anatomske razlike između pacijenata u samom obliku tkiva kojeg se snima. Na Slici 5.4 prikazani su primjeri augmentacije jednog CT presjeka.

5.4. Arhitektura mreže i modela

Arhitektura mreže koja se koristi u ovom radu prilagođena je verzija U-Net arhitekture razvijene u [25]. U-Net se temelji na arhitekturi potpuno povezanih konvolucijskih mreža (engl. *fully convolutional networks*) i sastoji se od dva dijela: puta sažimanja i puta širenja. Izvorna U-Net arhitektura

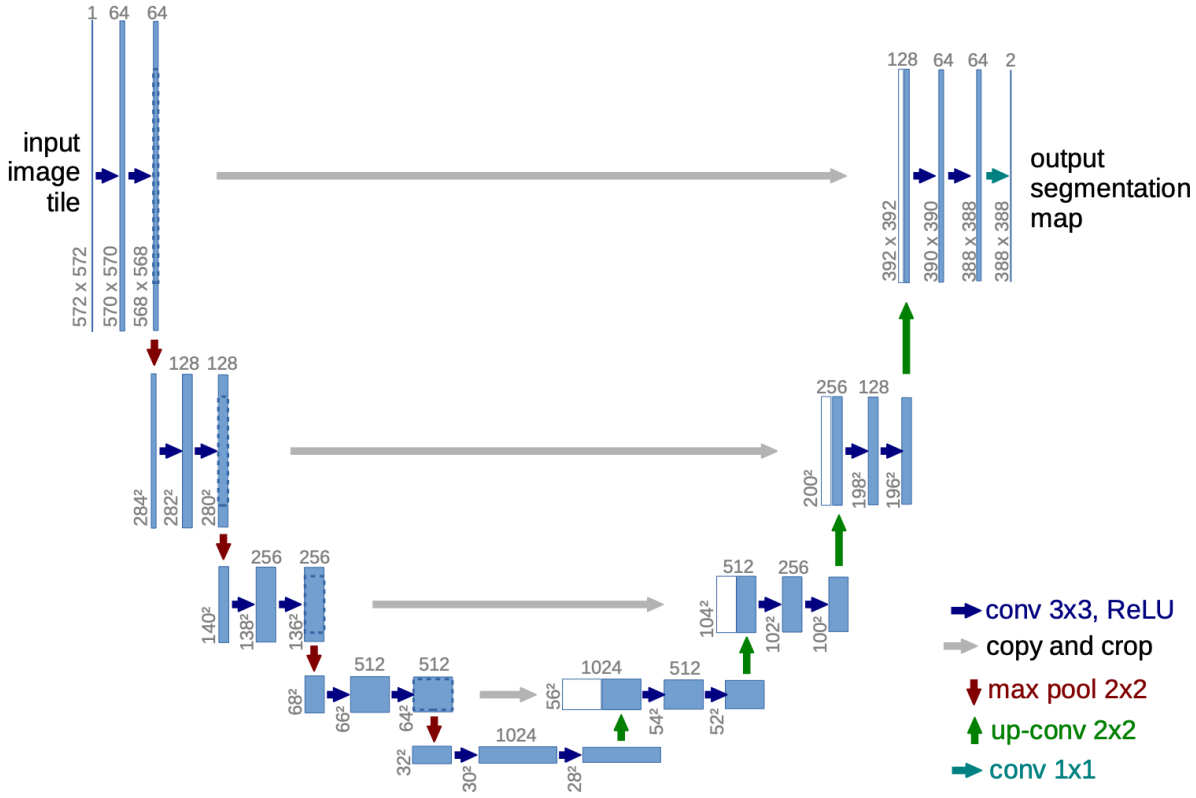


Slika 5.4: Prikaz 20 iteracija iste ulazne CT slike s različitim kombinacijama koraka augmentacije.

prikazana je na Slici 5.5.

Put sažimanja prati tipičnu arhitekturu konvolucijskih mreža. Osnovna je jedinica puta sažimanja niz od dva uzastopna 3×3 konvolucijska sloja, a svaki od njih ima ReLU aktivacijsku funkciju. Put sažimanja sastoji se od uzastopnih takvih jedinica gdje se između svake nalazi 2×2 sloj sažimanja po maksimalnoj vrijednosti (engl. *max pooling*), a nakon svakog sloja sažimanja udvostruči se broj kanala slike.

S druge strane, put širenja obavlja suprotne aktivnosti. Gradivna jedinica puta širenja također se sastoji od dva 3×3 konvolucijska sloja sa ReLU aktivacijom, ali između jedinica obavlja se uvećanje slike 2×2 konvolucijskim slojem koji udvostruči dimenzije slike. Matrice značajki puta sažimanja srezane su i nadodane na odgovarajuće matrice na putu širenja prije prve 2×2 konvolucije, što rezultira matricom s dvostruko više kanala. Prvi 2×2 konvolucijski sloj tada



Slika 5.5: Izvorna U-Net arhitektura [25].

smanji broj kanala za faktor od dva. U originalnom radu dubina puta sažimanja je četiri, odnosno postoje četiri operacije sažimanja slike.

Svojom strukturom U-Net istodobno omogućuje preciznu lokalizaciju zbog puta širenja, ali i dostupnost kontekstualnih informacija dobivenih spajanjem puta sažimanja s odgovarajućim slojevima iz puta širenja. To čini U-Net robusnom i vrlo preciznom arhitekturom pogodnom za biomedicinske slike gdje često ne postoji velik broj dostupnih podataka.

U ovom radu, izvorna U-Net arhitektura modificirana je na nekoliko načina. Put sažimanja (odnosno dio za kodiranje) zamijenjen je s ResNet34 koderom ([26]) s pet slojeva sažimanja, koji je lakši za optimiziranje i omogućuje prenošenje znanja s već treniranih modela. Put širenja je također modificiran. Prvo, broj kanala u najplićem sloju je 16, a sa svakim slojem produbljuje se za faktor od dva do širine 256. Između 3×3 konvolucijskih slojeva i njihovih aktivacijskih slojeva ubačen je sloj normalizacije niza podataka (engl. *batch normalization*) koji pomaže pri regularizaciji modela i smanjenju trajanja treninga [27]. Dijagram arhitekture modela prikazan je na Slici 5.6.

Tijekom izrade rada isprobano je nekoliko varijanti arhitekture modela, s različitim brojem slojeva i različitim brojem kanala slika značajki. Korištenjem gore opisane arhitekture postignuta je konvergencija modela s najmanjom greškom na validacijskom skupu podataka.

5.5. Treniranje i funkcija gubitka

U ovom radu, korišteni ResNet34 koder inicijaliziran je naučenim težinama na 2012 ILSVRC ImageNet skupu podataka [28]. Unatoč razlikama između CT slika i ImageNet podatkovnog skupa, moguće je da ResNet34 enkoder nauči prepoznavati dovoljno općenite značajke na ImageNet slikama (konture, osnovne oblike i sl.) da bi prijenos znanja bio koristan i za potrebe ovog rada. Ostali slojevi inicijalizirani su zadanim PyTorch inicijalizacijama tih slojeva.

Skup podataka od 874 presjeka nasumično je poredan i podijeljen na skupove za treniranje (85%, $n = 742$), validaciju (10%, $n = 88$) i testiranje (5%, $n = 44$). Podaci su augmentirani u svakoj epohi treniranja kao što je opisano u 5.3. Skup podataka podijeljen je u nizove (engl. *batch*) od 8 parova ulaznih slika i maski. Ulazne su slike i maske dimenzija 128×128.

Model je optimiziran SGD (engl. *stohastic gradient descent*) optimizatorom sa stopom učenja (engl. *learning rate*) od 0,01, te momentumom od 0,9, gdje je stopa opadanja težina (engl. *weight decay*) $5 \cdot 10^{-4}$. Postupak treniranja namješten je tako da se stopa učenja smanjuje za faktor od 0,1 ako nekoliko uzastopnih epoha nema značajne promjene u greški na validacijskom skupu. Tijekom treniranja model dosegne minimalnu grešku na validacijskom skupu između 35. i 45. epohe treninga.

U ovom radu isprobane su dvije verzije funkcije gubitka (engl. *loss function*). Funkcija L_a je tipična BCE (engl. *binary cross entropy*) greška kombinirana sa sigmoidnim slojem (engl. *BCE with logits*). Funkcija L_a opisana je Formulom 5.4:

$$\begin{gathered} L_a = \{l_1, \ldots, l_N\}^\top,\ l_{an} = -w_n[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))] \\ \ell_a(x, y) = mean(L) \end{gathered} \tag{5.4}$$

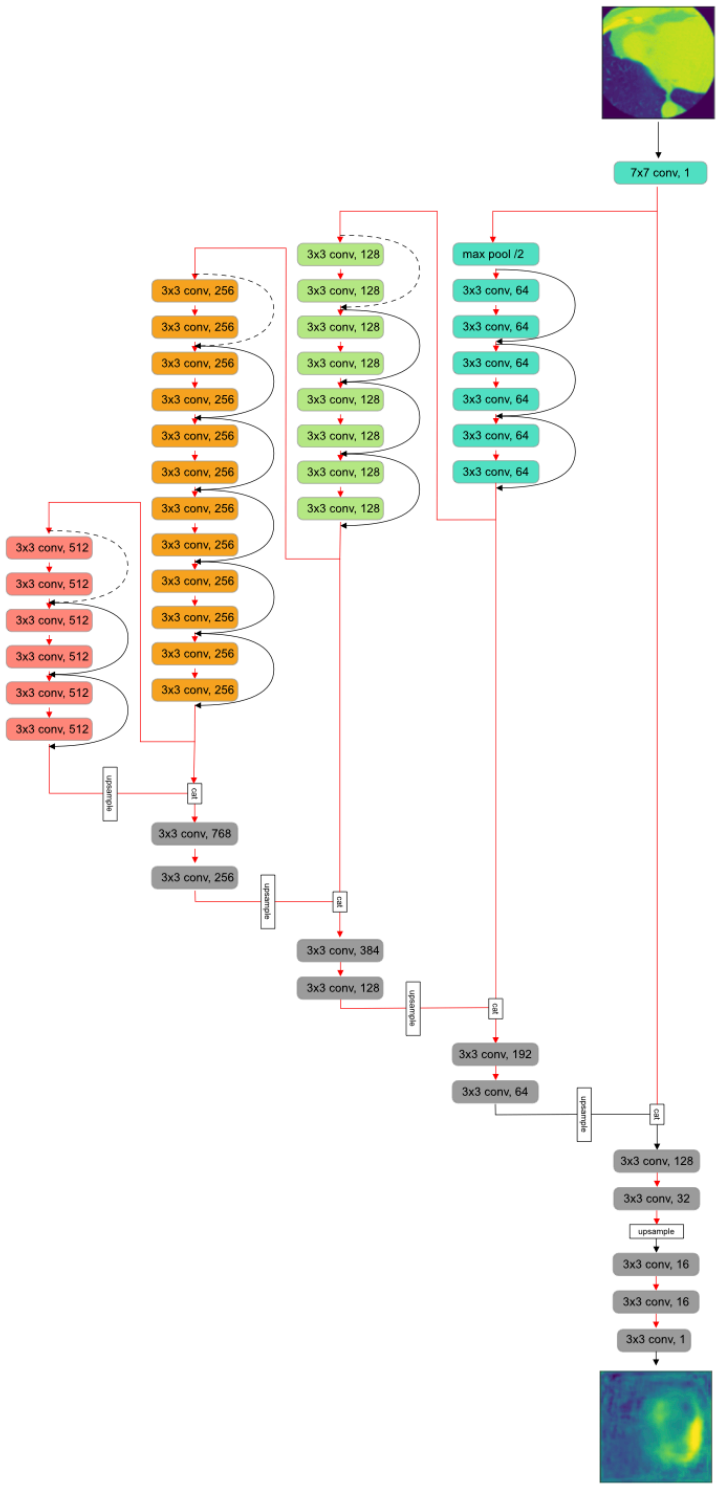
pri čemu je N veličina ulaznog niza podataka (engl. *batch size*).

Funkcija L_b kombinacija je funkcije L_a i srednje kvadratne pogreške između ukupnog broja piksela označenih sa 1 na maski i na predikciji modela. Prag na predikciji modela iznad kojeg su svi pikseli pretvoreni u 1, a svi ostali pretvoreni u 0, postavljen je na 0.75. Da bi obje pogreške imale približno jednak utjecaj na treniranje, član koji dodaje srednju kvadratnu podrešku pomnožen je s 0.01, s obzirom na to da se srednja kvadratna pogreška mjeri u broju piksela što rezultira puno većom vrijednosti od BCE greške. Funkcija L_b prikaza je definirana je Formulom 5.5:

$$\begin{gathered} t(x) = \begin{cases} 1, & x(i,j) \geq \theta \\ 0, & x(i,j) < \theta \end{cases} \\ L_b = \{l_1, \ldots, l_8\}^\top, l_{bn} = (\sum t(x_n) - \sum t(y_n))^2 \\ \ell_b(x, y) = \ell_a(x, y) + 0.01 * mean(L_b) \end{gathered} \tag{5.5}$$

pri čemu je θ vrijednost praga koji određuje vrijednost piksela iznad koje se smatra da je taj piksel epikardijalna mast. Tijekom treniranja vrijednost θ postavljena je na 0.75.

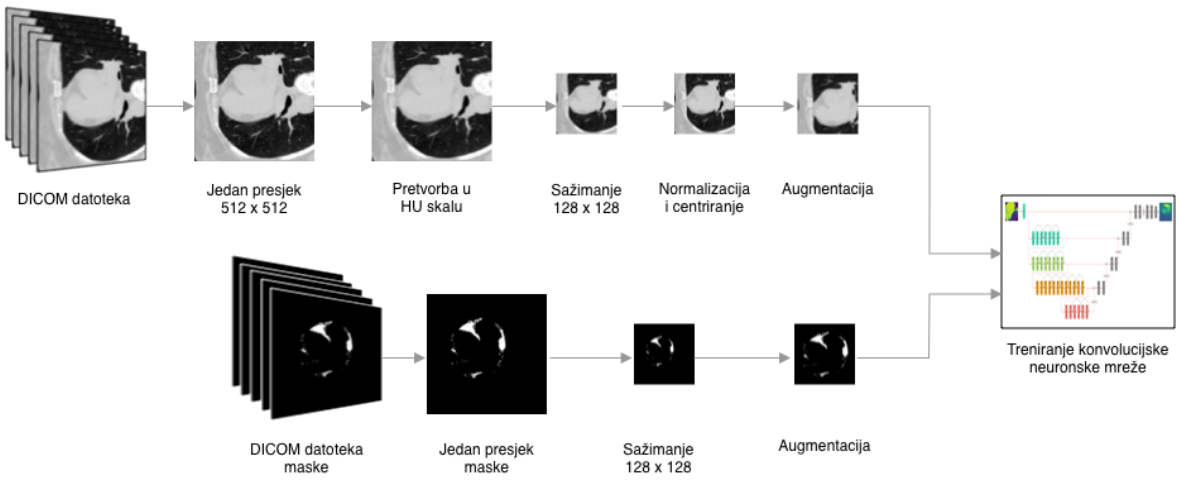
S obzirom da je krajnji cilj modela kvantifikacija, a ne nužno savršena segmentacija epikardijalne masti, funkcija L_b pokušava, uz dobru segmentaciju, osigurati i što manje odstupanje između ukupnog broja piksela segmentiranih kao epikardijalna mast kod predviđene i stvarne maske.



Slika 5.6: Arhitektura modela. Crvene strjelice uključuju normalizaciju niza (engl. *batch normalization*) i ReLU aktivaciju.

6. Implementacija rješenja

Ovaj rad je implementiran u programskom jeziku Python 3.8 uz pomoć biblioteke PyTorch 1.4. Izvorni kod implementacije nalazi se na [29]. Proces treninga mreže prikazan je na Slici 6.1.



Slika 6.1: Dijagram procesa treniranja mreže. Ulazni presjeci su predobrađeni pretvorbom u HU skalu, sažimanjem, normalizacijom i centriranjem. Ulazni presjeci i maske su augmentirane prije svake epohe treniranja.

6.1. Pretprocesiranje podataka

DICOM datoteke ulaznih podataka učitane su Pydicom bibliotekom. Pretprocesiranje opisano u 5.2 obavljeno je kombinacijom Pydicom i NumPy biblioteka. Prvo su učitane oznake svakog pacijenta, a zatim su meta podaci iz ulaznog skupa kopirani u odgovarajuće datoteke izlaznog skupa, zato što izlaznom skupu neki meta podaci nedostaju u DICOM datoteci. Zatim su učitani svaki presjek svakog pacijenta i odgovarajuća maska presjeka.

Svaki ulazni presjek transformiran je u Hounsfield skalu algoritmom prikazanim na Slici 6.2, a vrijednosti piksela spremljene su u zasebno polje, bez ostalih podataka iz DICOM datoteke.

Kao što je opisano Formulom 5.2, pikseli svakog presjeka su normalizirani na interval $(0, 1)$ i centrirani oko nule na interval $(-0.5, 0.5)$ algoritmom prikazanim na Slici 6.3.

Maske su učitane na isti način kao i ulazne slike, ali su pretvorene u matrice brojeva s pomičnim zarezom. Izvorno, te su matrice dimenzija $(N \times 512 \times 512 \times 3)$, odnosno RGB slike. Zatim je dobiven skalarni produkt svake od matrica i matrice $[0,299\ 0,587\ 0,114]$ kako bi se 3-kanalne RGB vrijednosti pretvorile u jednu crno-bijelu vrijednost od 0 do 255. Rezultat toga je matrica

```
def get_pixels_hu(slices):
    image = np.stack([s.pixel_array for s in slices])
    image = image.astype(np.int16)

    image[image == -2000] = 0

    for slice in range(len(slices)):
        intercept = slices[slice_number].RescaleIntercept
        slope = slices[slice_number].RescaleSlope

        if slope != 1:
            image[slice] = slope * image[slice].astype(np.float64)
            image[slice] = image[slice].astype(np.int16)

        image[slice] += np.int16(intercept)

    return np.array(image, dtype=np.int16)
```

Slika 6.2: Transformiranje presjeka u Hounsfield skalu.

```
def normalize(image, min, max):
    image = (image - min) / (max - min)
    image[image > 1] = 1.
    image[image < 0] = 0.
    return image

def zero_center(image, pixel_mean):
    image = image - pixel_mean
    return image
```

Slika 6.3: Normalizacija i centriranje ulaznih slika. Vrijednost `min` postavljena je na -1000, a `max` na 300.

dimenzija $(N \times 512 \times 512)$. S obzirom na to da su maske binarne slike (vrijednosti su ili 255 ili 0), i crno-bijela matrica je također binarna u intervalu $(0, 255)$. Postavljanjem svih vrijednosti matrice iznad 0 na 1, dobivena je binarna maska $(N \times 512 \times 512)$ čije su vrijednosti ili 0 ili 1. Kod za procesiranje maski prikazan je na Slici 6.4.

Vrijednosti piksela (NumPy matrice) svakog presjeka ulaznih slika i matrica spremljene su u zasebne datoteke, po jedna za svaki presjek, korištenjem NumPy mehanizma spremanja. Taj postupak olakšava kasnije učitavanje pojedinog presjeka tijekom treninga.

U zasebnoj Python skripti učitani su svi spremljeni presjeci, smanjeni na 128×128, nasumično poredani i kopirani u zasebne mape za treniranje, testiranje i validaciju.

Kreirana je PyTorch `Dataset` klasa koja učitava svaki spremljeni presjek, obavlja postupak

```
all_output_images = []
for patient in patients:
    output = load_scan(OUTPUT_FOLDER + patient + "_dcm")
    output_pixels = np.stack([s.pixel_array for s in output])
    output_pixels = output_pixels.astype(np.float32)
    output_pixels = np.dot(output_pixels[..., :3] , [0.299 , 0.587, 0.114])
    output_pixels[output_pixels > 0] = 1
    all_output_images.append(output_pixels)
```

Slika 6.4: Procesiranje maski.

augmentacije podatka i pretvara ulazne matrice u PyTorch `Tensor` objekt, a zatim vraća uređeni par ulazne slike i pripadajuće maske. Pri kreiranju `Dataset` klase dodan je argument za način rada klase. `Dataset` u `train` načinu rada daje augmentirane podatke za trening, dok u `test` i `valid` daje ne-augmentirane podatke za testiranje ili validaciju. `Dataset` služi kao objekt koji se može iterirati tijekom treniranja ili validacije za lak pristup podacima, bez potrebe za držanjem svih podataka odjednom u RAM memoriji računala. Kod za dohvaćanje jednog para prikazan je na Slici 6.5.

```
def __getitem__(self, i):
  file_name = self.filenames[i]
  image = np.load(self.folder + "input/" + file_name).astype(np.float32)
  mask = np.load(self.folder + "labels/" + file_name).astype(np.float32)
  augmented = self.transforms(image=image, mask=mask)
  image, mask = augmented["image"].double().unsqueeze(0), augmented["mask"].
      double()
  return image, mask
```

Slika 6.5: Funkcija za dohvaćanje jednog para ulazne slike i maske za treniranje ili validaciju.

Za augmentaciju slika koristi se biblioteka Albumentations [30] koja nudi mnoštvo gotovih funkcija za augmentaciju slika. Nad svakom slikom obavlja se transformacija u PyTorch `Tensor` objekt, dok su slike za trening još dodatno augmentirane afinim transformacijama i deformacijom rešetke. Definiranje Albumentations transformacija prikazano je na Slici 6.6.

6.2. Učitavanje mreže

Za implementaciju U-Net modela u ovom je radu korištena biblioteka Segmentation Models [31] koja nudi pouzdane, testirane i široko korištene implementacije različitih poznatih arhitektura za segmentiranje slike. U ovom je radu korištena implementacije U-Net arhitekture. Učitavanje mo-

```
def hard_transforms():
  return [
    albu.ShiftScaleRotate(
      shift_limit=0.2, scale_limit=0.2, rotate_limit=15, p=0.3),
    albu.GridDistortion(p=0.2),
  ]

def post_transforms():
  return [ToTensor()]
```

Slika 6.6: Definiranje transformacija na slikama.

dela prikazano je na Slici 6.7.

```
def get_model():
  return smp.Unet(in_channels=1).double()
```

Slika 6.7: Učitavanje modela.

6.3. Treniranje mreže i funkcije gubitka

Funkcije gubitka L_a i L_b opisane u 5.5 implementirane su PyTorch bibliotekom. Funkcija L_a (BCE greška) koristi predefinirani objekt za računanje BCE greške iz PyTorch biblioteke. Funkcija L_b kombinira BCE grešku s vrijednosti dobivenih opercijom praga na ulaznog slici, kojoj su zatim zbrojene vrijednosti svih piksela i izračunata je srednja kvadratna pogreška između zbroja piksela maske i ulazne slike. Kod za funkcije gubitka prikazan je na Slici 6.8.

Model je treniran na Tesla K80 grafičkoj kartici korištenjem FloydHub servisa za pokretanje koda na oblaku. Model se maksimalno trenira 70 epoha, ali prilikom treninga model konvergira između 35. i 40. epohe nakon oko 45 minuta. Model je optimiziran PyTorchovim SGD (engl. *stohastic gradient descent*) optimizatorom sa stopom učenja od 0.01, momentumom od 0.9 i opadanjem težina $5 \cdot 10^{-4}$. Veličina niza podataka (engl. *batch size*) za treniranje je 8. Kod za učitavanje skupa podataka i modela prikazan je na Slici 6.9.

Tijekom svake epohe računa se greška na validacijskom skupu podataka. Ako se greška smanji, automatski se spremaju naučeni parametri modela. Petlja treniranja modela prikazana je na Slici 6.10.

```
mse_loss = torch.nn.MSELoss()
bce_loss = torch.nn.BCEWithLogitsLoss()

def custom_mse_loss(output, label):
  output = (output - torch.min(output)) /
           (torch.max(output) - torch.min(output))
  output[output > 0.75] = 1
  output[output <= 0.75] = 0
  output_count = sum(output)
  label_count = sum(label)
  return mse_loss(output_count, label_count)

def loss_combined(output, label, epoch):
  bce = bce_loss(output, label)
  mse = custom_mse_loss(output, label)
  return bce + mse * 0.01

def loss(output, label, epoch):
  bce = bce_loss(output, label)
  return bce
```

Slika 6.8: Funkcije gubitka.

```
torch.manual_seed(30)

train_dataset = Dataset(mode="train")
test_dataset = Dataset(mode="test")

train_loader = DataLoader(
  train_dataset, batch_size=8, shuffle=True, num_workers=0)
test_loader = DataLoader(
  test_dataset, batch_size=1, shuffle=False, num_workers=0)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model = get_model()
model.to(device)
optimizer = torch.optim.SGD(
  model.parameters(), lr=0.01, momentum=0.9, weight_decay=5.0e-4)
```

Slika 6.9: Učitavanje modela i skupa podataka.

6.4. Evaluacija rezultata

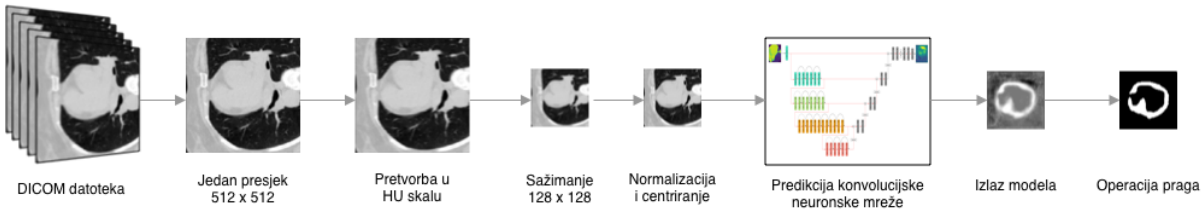
Nakon treniranja, model prolazi kroz Python skriptu koja vrši predikcije slika testnog skupa podataka i računa tri validacijska mjerila: Diceov koeficijent, Jaccard indeks i korelaciju. Dijagram

```
for i in range(0, EPOCHS_COUNT):
  # training
  train_loss = 0.
  model.train(True)
  optimizer.zero_grad()
  for iteration, batch in enumerate(train_loader):
    image, label = batch[0].to(device), batch[1].to(device)
    output = model(image)
    loss_value = loss(output, label, i)
    loss_value.backward()
    optimizer.step()
    optimizer.zero_grad()
    train_loss += loss_value.item()

  # validation
  model.train(False)
  validation_loss = 0.

  with torch.no_grad():
    for iteration, batch in enumerate(valid_loader):
      image, label = batch[0].to(device), batch[1].to(device)
      output = model(image)
      loss_value = loss(output, label, i)
      validation_loss += loss_value.item()

  scheduler.step(validation_loss)

  print(f"{i}: train loss: {train_loss}\t valid_loss: {validation_loss}")

  # saving
  if validation_loss < best_loss_mse:
    best_loss_mse = validation_loss
    if not os.path.exists("models/best_model.pth"):
      os.makedirs("models/")
    torch.save(model.state_dict(), 'models/best_model.pth')
    print("Saved new model.")
```

Slika 6.10: Petlja za treniranje modela.

procesa predikcije prikazan je na Slici 6.11.

Dice mjerilo računa se kao umnožak broja 2 i količnika logičkog i operatora nad dvije slike i zbroja vrijednosti piksela u obje slike. Jaccard indeks računa se kao količnik zbroja rezultata logičnog i operatora i zbroja rezultata logičkog ili operatora nad objema slikama. Implementacija



Slika 6.11: Dijagram procesa predikcije maske epikardijalne masti jednog presjeka CT slike.

Diceovog mjerila i Jaccard indeksa prikazana je na Slici 6.12. Objašnjenje mjerila opisano je u 7.1.

```
def dice(img1, img2):
  if img1.sum() + img2.sum() == 0:
    return 1
  intersection = np.logical_and(img1, img2)
  return 2. * intersection.sum() / (img1.sum() + img2.sum())

def jaccard(img1, img2):
  return np.double(np.bitwise_and(img1, img2).sum()) /
         np.double(np.bitwise_or(img1, img2).sum())
```

Slika 6.12: Implementacija Diceovog mjerila i Jaccard indeksa.

Korelacija je definirana kao Spearmanova korelacija između broja pozitivnih piksela ulaznih slika i odgovarajućih maski na cijelom testnom skupu podataka i računa se uz pomoć SciPy biblioteke.

7. Rezultati

Arhitektura modela postavljena je tako da je izlaz iz modela 128×128 crno-bijela slika, gdje svaki piksel ima vrijednost koja označava mjeru pouzdanosti modela da se na tom pikselu nalazi epikardijalna mast. Što je vrijednost piksela viša, model je sigurniji da se na tom pikselu nalazi epikardijalna mast. Primjer izlaza iz modela prikazan je na Slici 7.1.

Slika 7.1: Primjer jednog rezultata predikcije BCE modela.

Izlazne slike iz modela normalizirane su na interval $(0, 1)$, a zatim su operacijom praga pretvorene u binarnu sliku s vrijednosti 1 za sve piksele iznad vrijednosti praga, a 0 za sve ostale piksele. Vrijednost praga odabrana je prema testnom skupu podataka. Prag za kojeg je testni skup podataka imao najmanju grešku je u eksperimentima ovog rada bio između 0.7 i 0.8.

7.1. Način evaluacije

Model je ispitan tradicionalnim mjerilima za semantičku segmentaciju: Diceovim mjerilom i Jaccard indeksom [32]. Diceovo mjerilo, koje se još naziva i Sørensen–Diceov koeficijent ili F1 mjerilo statističko je mjerilo koje služi za procjenu sličnosti dva uzorka podataka. Računa se kao količnik dvostrukog kardinalnog broja presjeka skupa X i Y te zbroja kardinalnog broja skupa X i Y kao što je prikazano Formulom 7.1. Drugim riječima, količnik između dvostruke površine presjeka i ukupne površine. Vrijednosti Diceovog koeficijenta nalaze se u intervalu $(0, 1)$. Što je Diceov koeficijent viši, dvije slike su sličnije.

$$D(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|} \tag{7.1}$$

Jaccard indeks zove se još i presjek kroz unija (engl. *Intersection over Union, IoU*). Sličan je Diceovom koeficijentu i također služi za procjenu sličnosti dva podatka. Računa se kao količnik

kardinalnog broja presjeka skupova X i Y te kardinalnog broja unije ta dva skupa. Drugim riječima, Jaccard indeks je udio presjeka skupova u ukupnoj površini skupova, te se matematički prikazuje Formulom 7.2. Jaccard indeks se također nalazi u intervalu $(0, 1)$, a više vrijednosti označavaju veću sličnost između slika.

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{7.2}$$

Za medicinsku obradu slike, osim kvalitete segmentacije, vrlo je bitno i da pojedina mjerila nemaju visoka odstupanja od stvarne vrijednosti. Zato se prati i standardna devijacija Diceovog koeficijenta i Jaccard indeksa nad cijelim skupom testnih podataka. Standardna devijacija pokazuje odmak pojedinih rezultata od prosjeka pa je prema tome dobro mjerilo pouzdanosti predikcije. Što je standardna devijacija manja, to je razlika u kvaliteti predikcije različitih primjera manja.

Osim prethodno opisanih mjerila, u ovom se radu kao bitno mjerilo koristi i broj ukupnih piksela označenih kao epikardijalna mast. Krajnji cilj modela je kvantifikacija epikardijalne masti, gdje je volumen EM proporcionalan s brojem piksela označenih kao EM. Zato je bitno održati isti broj EM piksela kao i u stvarnim vrijednostima, neovisno o kvaliteti segmentacije.

Kvaliteta predikcije ukupnog broja EM piksela mjeri se i Spearmonovom korelacijom između sortiranog niza brojeva ukupnih EM piksela na stvarnim maskama i niza brojeva ukupnih EM piksela na odgovarajućim predviđenim maskama. Općenito, Spearman korelacija statističko je mjerilo monotonosti veze između dva skupa podataka. Vrijednosti Spearman korelacije su na intervalu $(-1, 1)$ gdje -1 i 1 označavaju savršenu korelaciju. Predznak vrijednosti označava smjer veze između dva skupa: ako je Spearman korelacija negativna, rastom zavisne varijable nezavisna varijabla se smanjuje i obrnuto. Formula za Spearman korelaciju prikazana je na Formuli 7.3.

$$r_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}, d_i = \mathrm{rb}(X_i) - \mathrm{rb}(Y_i) \tag{7.3}$$

Pri ćemu je n veličina skupa podataka, a $\mathrm{rb}(x)$ je redni broj vrijednosti x.

Sva mjerila prikazana u ovom poglavlju dobivena su na testnom skupu podataka. Testni skup podataka definiran je kao nasumičnih 5% ukupnog skupa podataka i sastoji se od 44 presjeka. Testni skup podataka nije korišten sve do evaluacije nakon treniranja, odnosno tijekom treniranja model nije bio izložen slikama iz testnog skupa podataka.

Ispitane su dvije inačice modela, jedan model koji koristi funkciju gubitka A kojeg se naziva BCE model i drugi, Kombinirani model, koji koristi kombiniranu funkciju gubitka B. Funkcije A i B opisane su u 5.5. U slučaju BCE modela, odvojeno je trenirana verzija modela bez augmentacije slika.

7.2. Prikaz i interpretacija rezultata

Oba su modela uspješno trenirana i mogu izvršiti zadatak semantičke segmentacije epikardijalne masti iz skupa podataka. Modeli postižu Diceov koeficijent 0,66 i više te Jaccardov indeks od 0,50 i više. Oba modela dobro prate broj ukupnih piksela označenih sa epikardijalnom masti, sa Spearmanovim koeficijentima korelacije 0,88 i više. Rezultati mjerenja prikazani su u Tablici 7.1.

Model	Prag	Dice	Jaccard	Korelacija
BCE model	0,70	0,70 ($\sigma = 0{,}08$)	0,55 ($\sigma = 0{,}09$)	0,94
BCE bez augmentacije	0,70	0,61 ($\sigma = 0{,}28$)	0,49 ($\sigma = 0{,}24$)	0,88
Kombinirani model	0,80	0,66 ($\sigma = 0{,}10$)	0,50 ($\sigma = 0{,}11$)	0,88

Tablica 7.1: Mjerila modela. Dice i Jaccard mjerila izračunani su kao prosjek rezultata za svaki primjer iz testnog skupa podataka. Korelacija je izračunana kao Spearmanova korelacija između sortiranog niza ukupnih EM piksela stvarnih slika i niza ukupnog broja EM piksela rezultata predikcije.

Na Slici 7.2 prikazani su najbolji primjeri segmentacije prema Dice mjerilu, dok su na Slici 7.3 prikazani najlošiji primjeri segmentacije za svaki model. Iz kvalitativne vizualne analize vidljivo je da su oba modela donekle sposobna lokalizirati epikardijalnu mast, ali imaju problema s praćenjem točne konture masti, pogotovo u primjerima gdje je masno tkivo raspršeno u manjim nepovezanim površinama. S druge strane, model dobro segmentira primjere gdje je epikardijalna mast grupirana u nekolicinu većih površina, pogotovo ako se površine prostiru duž rubove srca.

Prag oba modela određen je kao prag pri kojem model daje najviši prosječni Diceov koeficijent na cijelom testnom skupu podataka. Diceov keoficijent oba modela opada s odmakom praga od te vrijednosti s obje strane. Ovisnost Diceovog koeficijenta o pragu prikazana je na Slici 7.4. Ovisnost ukupne greške broja pisksela i Diceovog koeficijenta o pragu prikazana je na Slici 7.5.

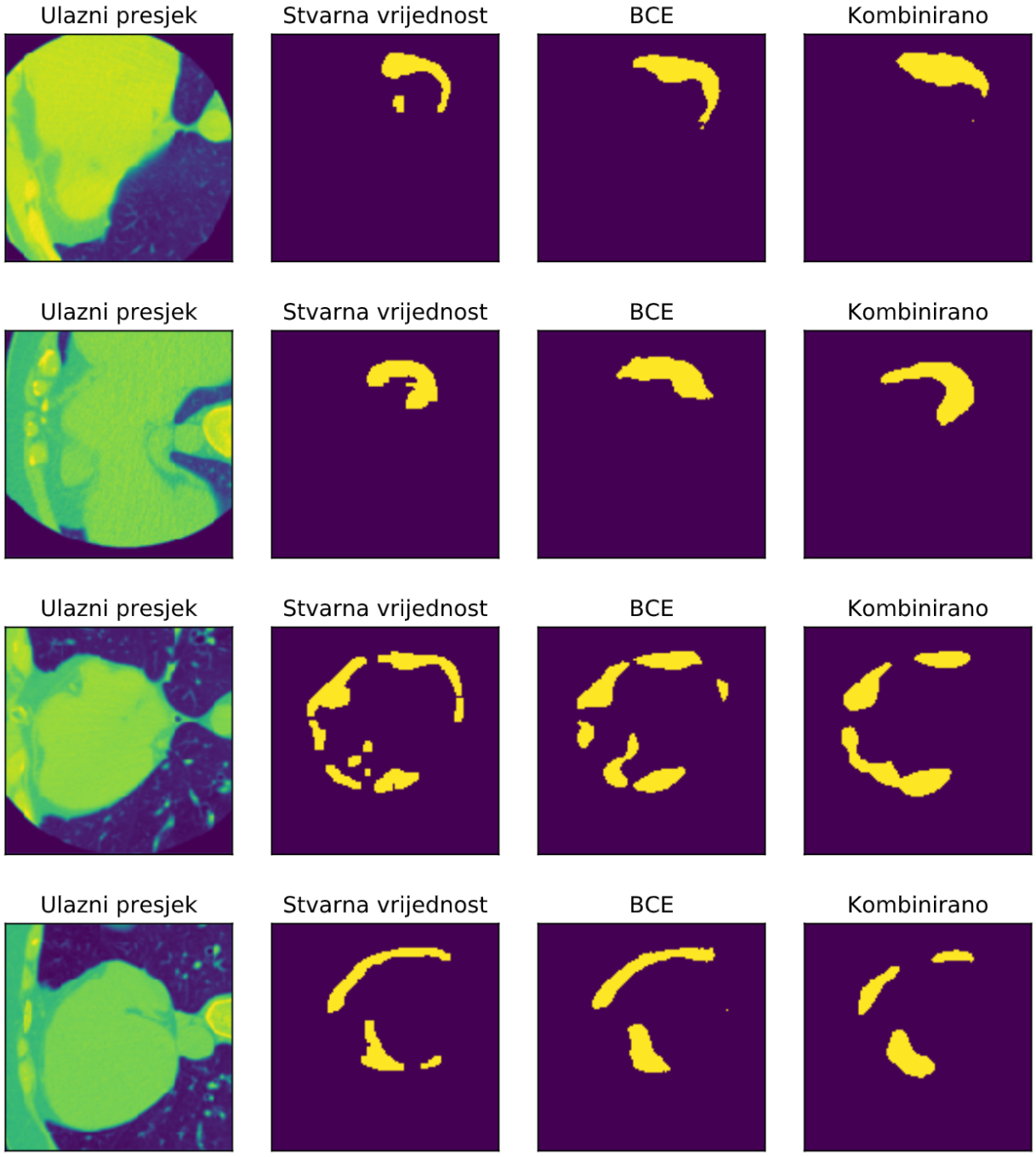
Dodavanjem augmentacije slika srednji Diceov koeficijent povećao se za 0,09 kod BCE modela i standardna devijacija smanjila se za jedan red veličine. Povećanje Diceovog koeficijenta ukazuje na bolju segmentaciju kod modela s augmentacijom, ali visoko smanjenje standardne devijacije pokazuje puno bolju generalizaciju modela (odnosno smanjenje prevelikog podešavanja modela skupu za trening) i bolju pouzdanost predikcije.

U eksperimentima ovog rada BCE model je konzistentno davao bolje rezultate od Kombiniranog modela. Iz Slike 7.5 vidljivo je da BCE model postiže viši maksimalni Diceov koeficijent, ali i viši Diceov koeficijent za širi raspon praga nego Kombinirani model. Greška ukupnog broja piksela niža je za BCE model pri širem rasponu praga nego kod Kombiniranog modela.

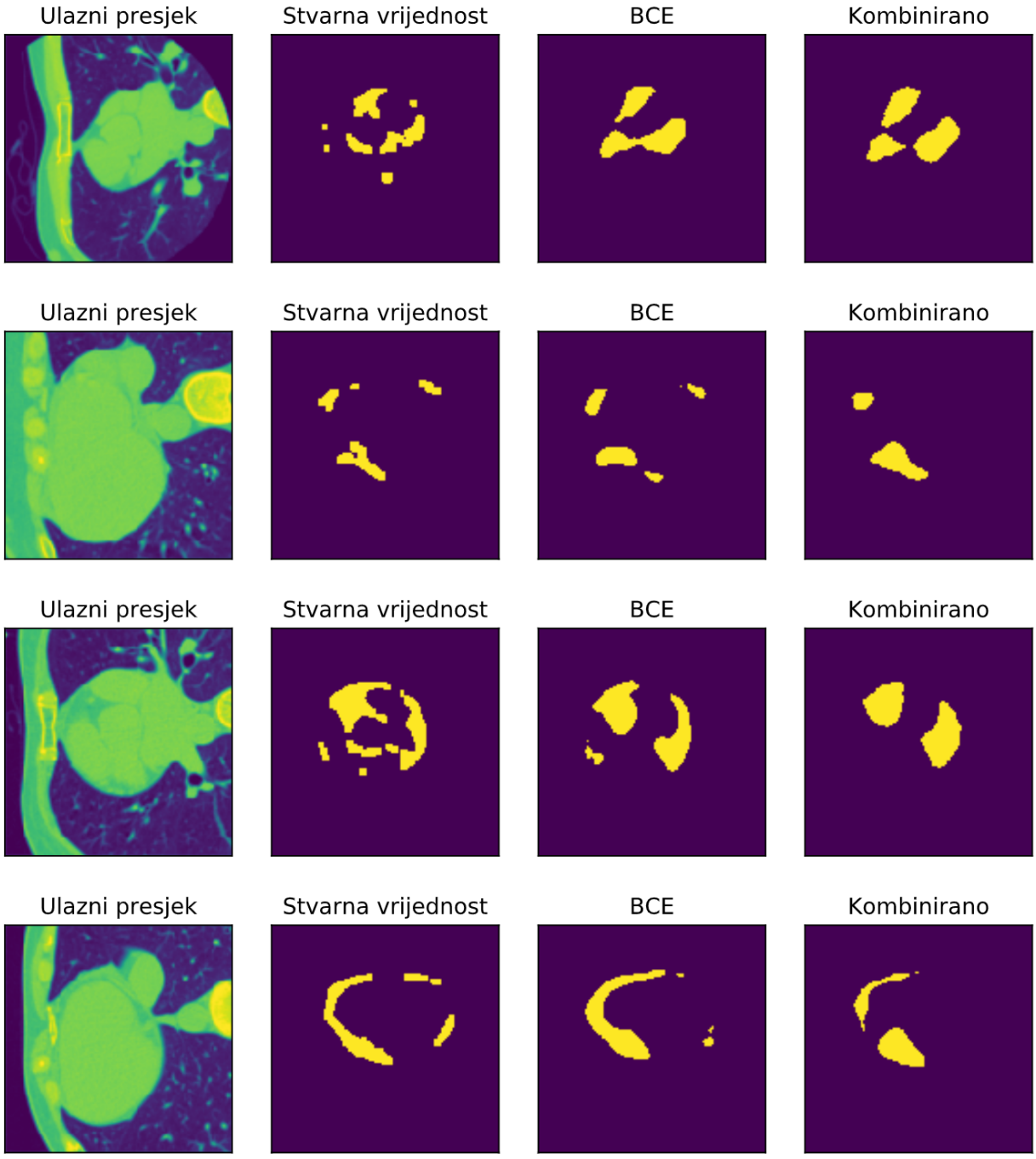
Iste rezultate pokazuje i Tablica 7.1 iz koje je vidljivo da BCE model ima značajno višu vrijednost Spearmanove korelacije i nešto viši Diceov koeficijent.

Na Slici 7.6 prikazani su brojevi ukupnih EM piksela za svaku sliku testnog skupa podataka. Vizualno, čini se da Kombinirani model dobro prati stvarne vrijednosti, ali ima ekstremne vri-

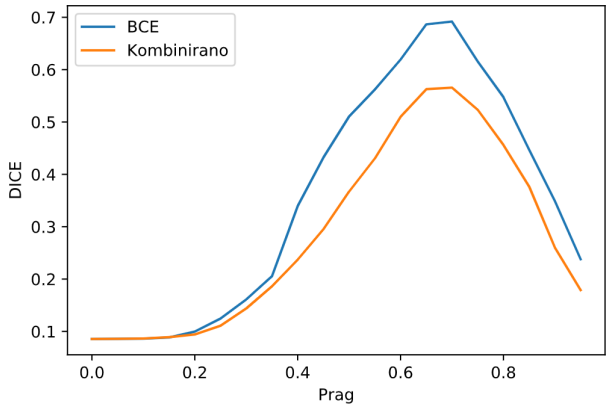
jednosti koje imaju viša odstupanja nego kod BCE modela. S druge strane, BCE model prikazuje nešto više brojeve piksela nego stvarne vrijednosti, ali ukupna odstupanja su manja nego kod Kombiniranog modela.



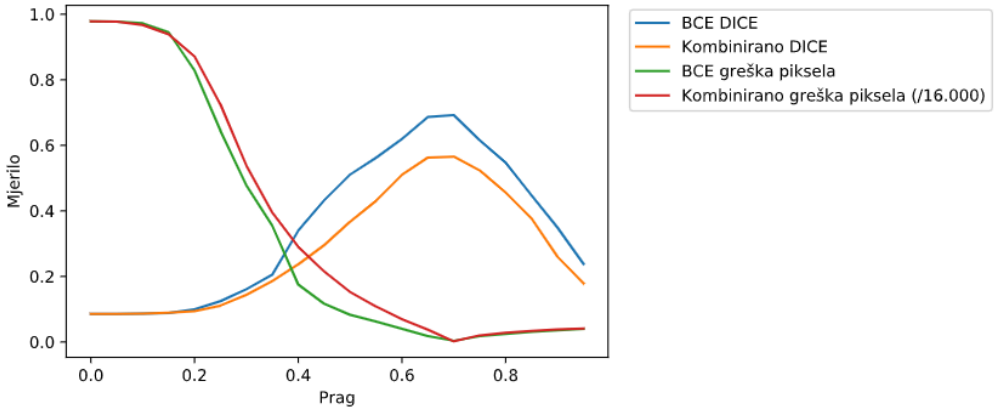
Slika 7.2: Najbolji primjeri rezultata segmentacije epikardijalne masti nakon operacije praga. Vrijednosti praga su 0,7 za BCE model i 0,8 za Kombinirani model.



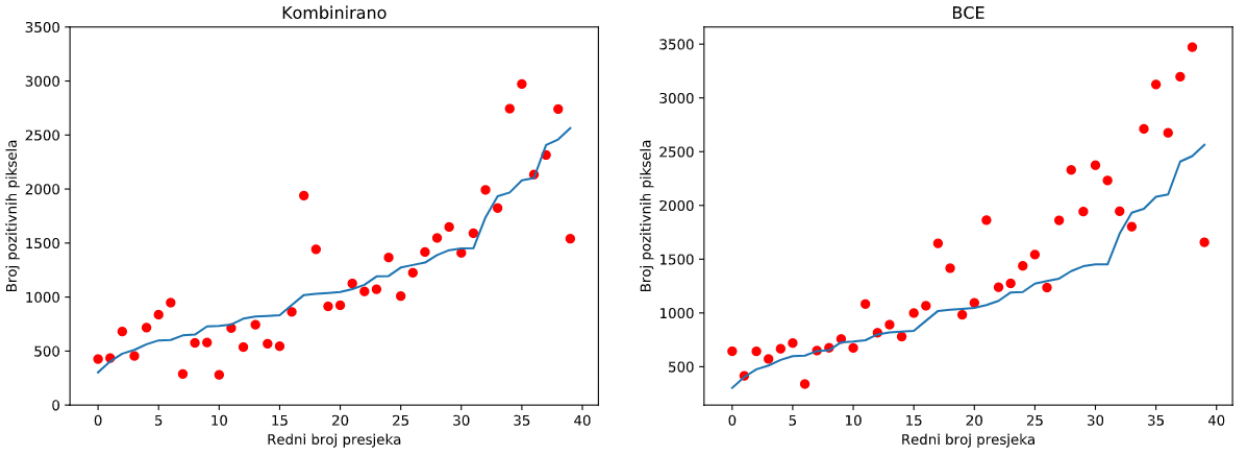
Slika 7.3: Najgori primjeri rezultata segmentacije epikardijalne masti nakon operacije praga. Vrijednosti praga su 0,7 za BCE model i 0,8 za Kombinirani model.



Slika 7.4: Graf Dice mjerila u ovisnosti o vrijednosti praga za jedan primjer ulazne slike.



Slika 7.5: Graf Dice mjerila i greške broja piksela u ovisnosti o vrijednosti praga za jedan primjer ulazne slike.



Slika 7.6: Broj pozitivnih piksela za svaku sliku. Plavom bojom označeni su stvarni brojevi piksela, dok su crvenom bojom označeni brojevi piksela dobiveni predikcijom modela.

8. Zaključak

Cilj ovog rada bio je razviti algoritam za potpuno automatsku semantičku segmentaciju epikardijalne masti iz slika dobivenih računalnom tomografijom, opisati postupak računalne tomografije te opisati lokaciju i značajnost epikardijalne masti.

U radu su opisani rad s CT slikama i predobrada CT slika za jednostavnije treniranje dubokih neuronskih mreža - pretvorba u Hounsfield skalu te normalizacija i centriranje CT slika. Opisana je i augmentacija podataka specifična za problem segmentacije na CT slikama. Koriste se afine transformacije kao i deformacija rešetke slike da bi se simulirali stvarni uvjeti snimanja CT slika. Augmentacija je značajno povećala Diceovo mjerilo segmentacije i pouzdanost mjerenja, odnosno smanjila standardnu devijaciju Diceovog mjerila. Nadalje, u radu je također testiran i utjecaj funkcije gubitka koja kombinira tradicionalne kriterije za segmentaciju slike s kriterijem koji prati ukupan broj piksela označenih kao epikardijalna mast. Dodavanje dodatnog kriterija funkciji gubitka otežalo je proces treniranja modela, ali neovisno o tome model je uspješno stabilizirao broj piksela. Na temelju dobivenih rezultata i provedenih eksperimenata, možemo pretpostaviti da bi uz dodatne eksperimente s arhitekturom modela i povećanjem broja podataka za treniranje, model s kombiniranim kriterijem rezultirao bi boljim praćenjem ukupnog broja piksela nego BCE model. Osim tih poboljšanja, broj ulaznih podataka može se povećati treniranjem mreže na volumenima od nekoliko uzastopnih presjeka odjednom ili treniranjem ansambla mreža gdje je svaka mreža trenirana na presjecima jedne okomite ravnine (transverzalna, sagitalna i koronarna) CT slike. Nadalje, model dobiveni ovim radom dobro lokalizira EM, ali ne prati dobro konturu tkiva. Jedno moguće poboljšanje segmentacije je korištenje matematičkih metoda detekcije rubova kao što su Sobelov operator kako bi se naglasili rubovi slika. Time bi se olakšao postupak treniranja i poboljšala konačna segmentacija epikardijalne masti. Osim toga, moguće je i potaknuti model na učenje informacija o rubovima povećavanjem težine greški na pikselima rubova u funkciji gubitka kao što je opisanu u [25].

Ovaj je rad temeljen je na skupu podataka od 20 pacijenata i dva različita CT uređaja. U radu nije ispitano koliko dobro model generalizira podatke dobivene drugim uređajima, drugim postavkama ili podatke iz drugih zdravstvenih ustanova i prema tome rezultati nisu nužno pokazatelj mogućnosti izrade sistematske studije volumena te debljine epikardijalne masti. U radu nije uspoređena kvaliteta modela u usporedbi s ručnom segmentacijom stručnjaka. Rezultati same segmentacije nešto su lošiji od trenutnog stanja literature [6],[7]. Pristupi temeljeni na dubokim neuronskim mrežama rezultiraju Diceovim mjerilom od 0,84 i više. Osnovni razlog tomu je korištenje većih podatkovnih skupova za treniranje modela od podatkovnog skupa korištenog u ovome radu. Poluautomatske metode daju još bolje rezultate s Diceovim mjerilom od 0,9 i više [3], [4] i [5], ali je njihovo izvođenje znatno sporije.

Razvijena metoda pokazuje uspješnu semantičku segmentaciju epikardijalne masti jednostavnom dubokom neuronskom mrežom, te uz relativno mali broj podataka za treniranje. U radu je također pokazano i da se augmentacijom skupa za trening postiže bolja i pouzdanija predikcija kod segmentacije epikardijalne masti.

9. Zahvale

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

10. Literatura

[1] Gianluca Iacobellis. Local and systemic effects of the multifaceted epicardial adipose tissue depot. *Nature Reviews Endocrinology*, 11(6):363–371, Travanj 2015. doi: 10.1038/nrendo.2015.58. URL https://doi.org/10.1038/nrendo.2015.58.

[2] Markus Goeller, Stephan Achenbach, Mohamed Marwan, Mhairi K. Doris, Sebastien Cadet, Frederic Commandeur, Xi Chen, Piotr J. Slomka, Heidi Gransar, J. Jane Cao, Nathan D. Wong, Moritz H. Albrecht, Alan Rozanski, Balaji K. Tamarappoo, Daniel S. Berman, i Damini Dey. Epicardial adipose tissue density and volume are related to subclinical atherosclerosis, inflammation and major adverse cardiac events in asymptomatic subjects. *Journal of Cardiovascular Computed Tomography*, 12(1):67–73, Siječanj 2018. doi: 10.1016/j.jcct.2017.11.007. URL https://doi.org/10.1016/j.jcct.2017.11.007.

[3] Xiaowei Ding, Demetri Terzopoulos, Mariana Diaz-Zamudio, Daniel S. Berman, Piotr J. Slomka, i Damini Dey. Automated pericardium delineation and epicardial fat volume quantification from noncontrast CT. *Medical Physics*, 42(9):5015–5026, Kolovoz 2015. doi: 10.1118/1.4927375. URL https://doi.org/10.1118/1.4927375.

[4] Rahil Shahzad, Daniel Bos, Coert Metz, Alexia Rossi, Hortense Kirişli, Aad van der Lugt, Stefan Klein, Jacqueline Witteman, Pim de Feyter, Wiro Niessen, Lucas van Vliet, i Theo van Walsum. Automatic quantification of epicardial fat volume on non-enhanced cardiac CT scans using a multi-atlas segmentation approach. *Medical Physics*, 40(9):091910, Kolovoz 2013. doi: 10.1118/1.4817577. URL https://doi.org/10.1118/1.4817577.

[5] Alexander Norlén, Jennifer Alvén, David Molnar, Olof Enqvist, Rauni Rossi Norrlund, John Brandberg, Göran Bergström, i Fredrik Kahl. Automatic pericardium segmentation and quantification of epicardial fat from computed tomography angiography. *Journal of Medical Imaging*, 3(3):034003, Rujan 2016. doi: 10.1117/1.jmi.3.3.034003. URL https://doi.org/10.1117/1.jmi.3.3.034003.

[6] Frederic Commandeur, Markus Goeller, Julian Betancur, Sebastien Cadet, Mhairi Doris, Xi Chen, Daniel S. Berman, Piotr J. Slomka, Balaji K. Tamarappoo, i Damini Dey. Deep learning for quantification of epicardial and thoracic adipose tissue from non-contrast CT. *IEEE Transactions on Medical Imaging*, 37(8):1835–1846, Kolovoz 2018. doi: 10.1109/tmi.2018.2804799. URL https://doi.org/10.1109/tmi.2018.2804799.

[7] Frederic Commandeur, Markus Goeller, Aryabod Razipour, Sebastien Cadet, Michaela M. Hell, Jacek Kwiecinski, Xi Chen, Hyuk-Jae Chang, Mohamed Marwan, Stephan Achenbach,

Daniel S. Berman, Piotr J. Slomka, Balaji K. Tamarappoo, i Damini Dey. Fully automated CT quantification of epicardial adipose tissue by deep learning: A multicenter study. *Radiology: Artificial Intelligence*, 1(6):e190045, Studeni 2019. doi: 10.1148/ryai.2019190045. URL https://doi.org/10.1148/ryai.2019190045.

[8] Avinash Kak. *Principles of computerized tomographic imaging*. Society for Industrial and Applied Mathematics, Philadelphia, 2001. ISBN 978-0898714944.

[9] Timothy Feeman. *The mathematics of medical imaging : a beginner's guide*. Springer, New York, 2010. ISBN 978-0387927114.

[10] Wouter De Vos, Jan Casselman, i Gwen Swennen. Cone-beam computerized tomography (cbct) imaging of the oral and maxillofacial region: A systematic review of the literature. *International journal of oral and maxillofacial surgery*, 38:609–25, 07 2009. doi: 10.1016/j.ijom.2009.02.028.

[11] Juan Pablo Bouza i Yassine Mrabet. Planes of human anatomy slika, dozvola: Creative commons attribution 3.0 unported creativecommons.org/licenses/by/3.0/deed.en, 2011. URL https://commons.wikimedia.org/wiki/File:Human_anatomy_planes.svg.

[12] Medical World. Epifatnet, 2020. URL http://www.medicalook.com/tests/CT_Scan.html.

[13] VA USA National Electrical Manufacturers Association, Rosslyn. Nema ps3 / iso 12052, digital imaging and communications in medicine (dicom) standard. URL http://medical.nema.org/.

[14] Angela Galllina Bertaso, Daniela Bertol, Bruce Bartholow Duncan, i Murilo Foppa. Epicardial fat: Definition, measurements and systematic review of main outcomes. *Arquivos Brasileiros de Cardiologia*, 2013. doi: 10.5935/abc.20130138. URL https://doi.org/10.5935/abc.20130138.

[15] Gianluca Iacobellis i Howard J. Willens. Echocardiographic epicardial fat: A review of research and clinical applications. *Journal of the American Society of Echocardiography*, 22 (12):1311–1319, Prosinac 2009. doi: 10.1016/j.echo.2009.10.013. URL https://doi.org/10.1016/j.echo.2009.10.013.

[16] K. Meenakshi, M. Rajendran, S. Srikumar, i Sundar Chidambaram. Epicardial fat thickness: A surrogate marker of coronary artery disease – assessment by echocardiography. *Indian Heart Journal*, 68(3):336–341, Svibanj 2016. doi: 10.1016/j.ihj.2015.08.005. URL https://doi.org/10.1016/j.ihj.2015.08.005.

[17] Suhny Abbara, Jay C. Desai, Ricardo C. Cury, Javed Butler, Koen Nieman, i Vivek Reddy. Mapping epicardial fat with multi-detector computed tomography to facilitate percutaneous transepicardial arrhythmia ablation. *European Journal of Radiology*, 57(3):417–422, Ožujak 2006. doi: 10.1016/j.ejrad.2005.12.030. URL https://doi.org/10.1016/j.ejrad.2005.12.030.

[18] Gianluca Iacobellis. Epicardial and pericardial fat: Close, but very different. *Obesity*, 17(4): 625–625, 2009. doi: 10.1038/oby.2008.575. URL https://onlinelibrary.wiley.com/doi/abs/10.1038/oby.2008.575.

[19] Victor Y. Cheng, Damini Dey, Balaji Tamarappoo, Ryo Nakazato, Heidi Gransar, Romalisa Miranda-Peats, Amit Ramesh, Nathan D. Wong, Leslee J. Shaw, Piotr J. Slomka, i Daniel S. Berman. Pericardial fat burden on ECG-gated noncontrast CT in asymptomatic patients who subsequently experience adverse cardiovascular events. *JACC: Cardiovascular Imaging*, 3 (4):352–360, Travanj 2010. doi: 10.1016/j.jcmg.2009.12.013. URL https://doi.org/10.1016/j.jcmg.2009.12.013.

[20] Vincent Dumoulin i Francesco Visin. A guide to convolution arithmetic for deep learning, 2016.

[21] Waseem Rawat i Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29:1–98, 06 2017. doi: 10.1162/NECO_a_00990.

[22] É.O. Rodrigues, F.F.C. Morais, N.A.O.S. Morais, L.S. Conci, L.V. Neto, i A. Conci. A novel approach for the automated segmentation and volume quantification of cardiac fats on computed tomography. *Computer Methods and Programs in Biomedicine*, 123:109–128, Siječanj 2016. doi: 10.1016/j.cmpb.2015.09.017. URL https://doi.org/10.1016/j.cmpb.2015.09.017.

[23] Visual lab, a computed tomography cardiac fat dataset, 2014. URL http://visual.ic.uff.br/en/cardio/ctfat/index.php.

[24] Marie Kloenne, Sebastian Niehaus, Leonie Lampe, Alberto Merola, Janis Reinelt, Ingo Roeder, i Nico Scherf. Domain specific cues improve robustness of deep learning based segmentation of ct volumes, 2019.

[25] Olaf Ronneberger, Philipp Fischer, i Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition, 2015.

[27] Sergey Ioffe i Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, i Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[29] Marin Benčević. Epifatnet, 2020. URL `http://visual.ic.uff.br/en/cardio/ctfat/index.php`.

[30] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, i Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL `https://www.mdpi.com/2078-2489/11/2/125`.

[31] Pavel Yakubovskiy. Segmentation models pytorch. `https://github.com/qubvel/segmentation_models.pytorch`, 2020.

[32] Varduhi Yeghiazaryan i Irina Voiculescu. An overview of current evaluation methods used in medical image segmentation. Technical Report RR-15-08, Department of Computer Science, Oxford, UK, 2015.

11. Sažetak

Naslov: Segmentacija epikardijalne masti iz slika računalne tomografije
Sažetak: Epikardijalna mast (EM) masno je tkivo koje se nalazi između srca i perikarda. Volumen i debljina epikardijalnog masnog tkiva povezai su s različitim bolestima kao što su ateroskleroza, masnoća srca, te drugim nepovoljnim kardiovaskularnim događajima. EM je nezavisnan indikator rizika od srčanih bolesti. Potpuno automatskim i pouzdanim mjerenjem EM iz CT slika mogu se obraditi veliki skupovi CT slika i time napraviti sistematska studija epikardijalne masti. Ovaj rad predlaže potpuno automatsku metodu semantičke segmetnacije epikardijalne masti iz CT slika koristeći duboke neuronske mreže. Koristi se modificirana U-Net arhitektura s ResNet34 koderom. Model je treniran na 20 CT slika srca dobivenih iz dva različita CT uređaja. Rezultat modela su predikcije segmentiranih područja EM na CT slikama. Kvantitativna evaluacija rezultata pokazuje podudaranje Dice mjerila od 0,70. U radu je pokazano da je moguće postići grubu semantičku segmentaciju EM dubokom neuronskom mrežom čak i uz mali broj podataka uz pomoć dobre predobrade podatkovnog skupa. Rad pokazuje i ovisnost utjecaja augmentacije podataka na konačni rezultat semantičke segmentacije CT slika.

12. Abstract

Title: Semantic segmentation of epicardial fat from computed tomography images
Sažetak: Epicardial fat (EF) is an adipose tissue located between the heard and the pericardium. The volume and thickness of EF is linked with various conditions such as atherosclerosis, visceral adiposity and other cardiovascular events. EF is also an independent cardiovascular disease risk factor. Fully automatic and reliable measurements of EF from CT scans could enable processing large CT image data sets for a systemic EF study. This thesis proposes a method for fully automatic semantic segmentation of EF tissue from CT images using a deep neural network. The model has a modified U-Net architecture with a ResNet34 encoder. The model is trained on 20 CT scans of the heart. The model results in a prediction of segmented EF area on CT images. Quantitive analysis of the model's results yields a Dice score of 0.70. This thesis shows that it is possible to achieve rough semantic segmentation of EF tissue using a neural network even without large amounts of data, provided the data is preprocessed appropriately. It also shows the impact of data augmentation on segmentation results from CT images.

13. Prilozi

13.1. dataset.py

```
import numpy as np
import os
import re
import random
from shutil import copyfile
from pathlib import Path
import typing as t
from skimage.transform import resize
import sys

ROOT_INPUT_FOLDER = 'dataset/preprocess_no_resample/'
INPUT_SLICES_FOLDER = 'input_slices/'
INPUT_RESIZED_FOLDER = 'dataset/preprocess_no_resample/input_slices_resized/'
OUTPUT_SLICES_FOLDER = 'output_slices/'
OUTPUT_RESIZED_FOLDER = 'dataset/preprocess_no_resample/output_slices_resized/
    '

TRAIN_FOLDER = 'dataset/train/'
TEST_FOLDER = 'dataset/test/'
VALID_FOLDER = 'dataset/valid/'

TEST_PERCENT = 0.1
VALID_PERCENT = 0.05

def main(
  input_data_path = ROOT_INPUT_FOLDER,
  image_size=128,
  test_percent = 0.1,
  valid_percent = 0.05):
  if not os.path.exists("dataset/preprocess_no_resample"):
    os.makedirs("dataset/preprocess_no_resample")

  patient_ids = os.listdir(input_data_path + INPUT_SLICES_FOLDER)
  patient_ids = [re.sub(r'[0-9]', "", name) for name in patient_ids]
  patient_ids = [name.replace("_slice_.npy", "") for name in patient_ids]
  patient_ids = list(set(patient_ids))

  resize_images(input_data_path, image_size)
```

```
  make_dataset(test_percent, valid_percent)

def resize_images(input_data_path, image_size):
  """
  Resizes all images in the output and input slices folders to IMG_SIZE, and
     stores them in resized/
  """

  Path(INPUT_RESIZED_FOLDER).mkdir(parents=True, exist_ok=True)
  input_slices = os.listdir(input_data_path + INPUT_SLICES_FOLDER)
  if "resized" in input_slices:
    input_slices.remove("resized")
  for input_slice in input_slices:
    if not os.path.exists(INPUT_RESIZED_FOLDER + input_slice):
      img = np.load(input_data_path + INPUT_SLICES_FOLDER + input_slice)
      img = resize(img, (image_size, image_size), mode='constant',
         preserve_range=True)
      np.save(INPUT_RESIZED_FOLDER + input_slice, img)

  Path(OUTPUT_RESIZED_FOLDER).mkdir(parents=True, exist_ok=True)
  output_slices = os.listdir(input_data_path + OUTPUT_SLICES_FOLDER)
  if "resized" in output_slices:
    output_slices.remove("resized")
  for output_slice in output_slices:
    if not os.path.exists(OUTPUT_RESIZED_FOLDER + output_slice):
      img = np.load(input_data_path + OUTPUT_SLICES_FOLDER + output_slice)
      img = resize(img, (image_size, image_size), mode='constant',
         preserve_range=True)
      np.save(OUTPUT_RESIZED_FOLDER + output_slice, img)

def get_train_test_valid(test_percent, valid_percent):
  """
  Splits into train, test and validation sets and returns a tuple of (train,
     test, valid) arrays.
  """

  files = os.listdir(INPUT_RESIZED_FOLDER)
  random.seed(1200)
  random.shuffle(files)

  train_percent = 1. - test_percent - valid_percent

  split_1 = int(train_percent * len(files))
  split_2 = int((train_percent + test_percent) * len(files))
```

```
  train_files = files[:split_1]
  test_files = files[split_1:split_2]
  valid_files = files[split_2:]

  return train_files, test_files, valid_files

def make_dataset(test_percent, valid_percent):
  """
  Creates three folders inside the dataset/ folder, train/ test/ and valid/,
  and copies a percentage of all the files inside /dataset/
     preprocess_no_resample/
  to those folders.
  """

  train_files, test_files, valid_files = get_train_test_valid(test_percent,
     valid_percent)

  folders = {
    "dataset/train/": train_files,
    "dataset/test/": test_files,
    "dataset/valid/": valid_files
  }

  for folder, files in folders.items():
    input_folder = folder + "input/"
    labels_folder = folder + "labels/"

    Path(input_folder).mkdir(parents=True, exist_ok=True)
    Path(labels_folder).mkdir(parents=True, exist_ok=True)

    copy_files(files, INPUT_RESIZED_FOLDER, input_folder)
    copy_files(files, OUTPUT_RESIZED_FOLDER, labels_folder)

  print(f"""
  Splitting dataset:
   - train: {1. - TEST_PERCENT - VALID_PERCENT}, n = {len(train_files)}
   - test: {TEST_PERCENT}, n = {len(test_files)}
   - validation: {VALID_PERCENT}, n = {len(valid_files)}
  """)

def copy_files(file_names: t.List[str], src_folder: str, dst_folder: str):
  """Copies a list of files from one folder to another."""

  for file_name in file_names:
    src_file = src_folder + file_name
```

```
    dst_file = dst_folder + file_name
    copyfile(src_file, dst_file)

if __name__ == '__main__':
  if len(sys.argv) > 1:
    input_data_path = sys.argv[1]
    main(input_data_path)
  else:
    main()
```

13.2. dataloader.py

```
import numpy as np
import dataset
import torch
import typing as t
import os
import torchvision.transforms as transforms
from torch.utils.data import Dataset as BaseDataset
import albumentations as albu
from albumentations.pytorch import ToTensor

class Dataset(BaseDataset):

  @staticmethod
  def hard_transforms():
    return [
      albu.ShiftScaleRotate(shift_limit=0.2, scale_limit=0.2, rotate_limit=15,
          p=0.3),
      albu.GridDistortion(p=0.2),
    ]

  @staticmethod
  def post_transforms():
    return [ToTensor()]

  @staticmethod
  def compose(transforms_to_compose):
    return albu.Compose([item for sublist in transforms_to_compose for item in
        sublist])

  def __init__(self, mode):
    self.mode = mode
    if mode == "train":
      self.folder = dataset.TRAIN_FOLDER
      self.filenames = os.listdir(dataset.TRAIN_FOLDER + "input/")
      self.transforms = Dataset.compose([Dataset.hard_transforms(), Dataset.
          post_transforms()])
    elif mode == "valid":
      self.folder = dataset.VALID_FOLDER
      self.filenames = os.listdir(dataset.VALID_FOLDER + "input/")
      self.transforms = Dataset.compose([Dataset.post_transforms()])
    elif mode == "test":
      self.folder = dataset.TEST_FOLDER
      self.filenames = os.listdir(dataset.TEST_FOLDER + "input/")
```

```
    self.transforms = Dataset.compose([Dataset.post_transforms()])

  if ".DS_Store" in self.filenames:
    self.filenames.remove(".DS_Store")

  self.clean_empty_images()

def clean_empty_images(self):
  files_to_remove = []
  for i in range(len(self)):
    file_name = self.filenames[i]
    mask = np.load(self.folder + "labels/" + file_name).astype(np.float32)
    if np.sum(mask) <= 0:
      files_to_remove.append(file_name)

  print(f"Removing␣{len(files_to_remove)}␣empty␣images.")
  for file in files_to_remove:
    self.filenames.remove(file)

def __getitem__(self, i):
  file_name = self.filenames[i]
  image = np.load(self.folder + "input/" + file_name).astype(np.float32)
  mask = np.load(self.folder + "labels/" + file_name).astype(np.float32)
  augmented = self.transforms(image=image, mask=mask)
  image, mask = augmented["image"].double().unsqueeze(0), augmented["mask"].
      double()
  return image, mask

def __len__(self):
  return len(self.filenames)
```

13.3. train.py

```
import torch
import segmentation_models_pytorch as smp
import numpy as np
from dataloader import Dataset
from torch.utils.data import DataLoader
import torch.nn as nn
from torch.nn import functional as F
import math
import os

def get_model():
  return smp.Unet(in_channels=1).double()

mse_loss = torch.nn.MSELoss()

def custom_mse_loss(output, label):
  output = (output - torch.min(output)) / (torch.max(output) - torch.min(
      output))
  output[output > 0.75] = 1
  output[output <= 0.75] = 0
  output_count = sum(output)
  label_count = sum(label)
  return mse_loss(output_count, label_count)

def loss_combined(output, label, epoch):
  bce = bce_loss(output, label)
  mse = custom_mse_loss(output, label)
  return bce + mse * 0.01

bce_loss = torch.nn.BCEWithLogitsLoss()

def loss(output, label, epoch):
  bce = bce_loss(output, label)
  return bce


def main():
  torch.manual_seed(30)

  train_dataset = Dataset(mode="train")
  valid_dataset = Dataset(mode="valid")

  train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True,
```

```
    num_workers=0)
valid_loader = DataLoader(valid_dataset, batch_size=1, shuffle=False,
    num_workers=0)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

model = get_model()
model.to(device)

optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9,
    weight_decay=5.0e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, "min",
    verbose=True)

EPOCHS_COUNT = 70

best_loss_mse = math.inf

for i in range(0, EPOCHS_COUNT):
  # training
  # (total, mse, bce)
  train_loss = 0.
  model.train(True)
  optimizer.zero_grad()
  for iteration, batch in enumerate(train_loader):
    image, label = batch[0].to(device), batch[1].to(device)
    output = model(image)
    loss_value = loss_combined(output, label, i)
    loss_value.backward()
    optimizer.step()
    optimizer.zero_grad()
    train_loss += loss_value.item()

  # validation
  model.train(False)
  validation_loss = 0.

  with torch.no_grad():
    for iteration, batch in enumerate(valid_loader):
      image, label = batch[0].to(device), batch[1].to(device)
      output = model(image)
      loss_value = loss(output, label, i)
      validation_loss += loss_value.item()
```

```
    scheduler.step(validation_loss)

    print(f"{i}: train loss: {train_loss}\t valid_loss: {validation_loss}")

    # saving
    if validation_loss < best_loss_mse:
      best_loss_mse = validation_loss
      if not os.path.exists("models/best_model.pth"):
        os.makedirs("models/")
      torch.save(model.state_dict(), 'models/best_model.pth')
      print("Saved new model.")

if __name__ == '__main__':
  main()
```

13.4. eval.py

```
import numpy as np
import torch
import train
from dataloader import Dataset
import sys
import concurrent.futures
from functools import partial
from scipy.stats import spearmanr

dataset = Dataset(mode="test")

def dice(img1, img2):
  if img1.sum() + img2.sum() == 0:
    return 1
  intersection = np.logical_and(img1, img2)
  return 2. * intersection.sum() / (img1.sum() + img2.sum())

def jaccard(img1, img2):
  return np.double(np.bitwise_and(img1, img2).sum()) / np.double(np.bitwise_or
      (img1, img2).sum())

def _get_score(function, img1, img2):
  function(img1, img2)

def _get_validation_pair(model, dataset, threshold, i):
  with torch.no_grad():
    image, mask = dataset[i]
    output = model(image.unsqueeze(0)).squeeze()
    output = output.detach().numpy()
    output = (output - np.min(output))/np.ptp(output)
    output[output > threshold] = 1
    output[output <= threshold] = 0
    output = output.astype(np.bool)
    mask = mask.detach().numpy().astype(np.bool)
    return output, mask

def pixel_count(img1, img2):
  """
  Returns a tuple with the number of True pixels in both images.
  """
  return np.sum(img1), np.sum(img2)
```

```
def _get_score(model, dataset, threshold, i):
  model_output, ground_truth = _get_validation_pair(model, dataset, threshold,
      i)
  return dice(model_output, ground_truth), jaccard(model_output, ground_truth)
     , pixel_count(model_output, ground_truth)

def _get_scores(model, dataset, threshold):
  dices = []
  jaccards = []
  pixel_counts = []
  with concurrent.futures.ThreadPoolExecutor() as executor:
    for dice, jaccard, pixel_count in executor.map(partial(_get_score, model,
       dataset, threshold), range(0, len(dataset))):
      dices.append(dice)
      jaccards.append(jaccard)
      pixel_counts.append(pixel_count)
  return np.array(dices), np.array(jaccards), np.array(pixel_counts)

def evaluate(model, saved_model_path, threshold=0.75):
  _load_model(model, saved_model_path)
  dices, jaccards, pixel_counts = _get_scores(model, dataset, threshold)
  dice_mean = np.mean(dices)
  dice_std = np.std(dices)
  print(f'DICE: {dice_mean}, std: {dice_std}')
  jaccard_mean = np.mean(jaccards)
  jaccard_std = np.std(jaccards)
  print(f'JACC: {jaccard_mean}, std: {jaccard_std}')

  pixel_count_output, pixel_count_ground_truth = zip(*pixel_counts)
  sorted_pixel_indices = np.argsort(pixel_count_ground_truth)
  pixel_count_output = np.array(pixel_count_output)[sorted_pixel_indices]
  pixel_count_ground_truth = np.array(pixel_count_ground_truth)[
     sorted_pixel_indices]
  corellation, p = spearmanr(pixel_count_output, pixel_count_ground_truth)
  print(f'CORR: {corellation} (p = {p})')

def _load_model(model, saved_model_path):
  if torch.cuda.is_available():
    state_dict = torch.load(saved_model_path)
  else:
    state_dict = torch.load(saved_model_path, map_location=torch.device('cpu')
       )
  model.load_state_dict(state_dict)
  model.eval()
```

```
def main(saved_model_path, threshold):
  model = train.get_model()
  evaluate(model, saved_model_path, threshold)

if __name__ == '__main__':
  if len(sys.argv) > 1:
    threshold = float(sys.argv[1])
    saved_model_path = sys.argv[2]
    main(saved_model_path, threshold)
  else:
    print("usage: train.py <threshold> <path to saved model>")
```

13.5. preprocess.py

```
import numpy as np
import pandas as pd
import os
import scipy.ndimage
import matplotlib.pyplot as plt
import pydicom
from skimage import measure, morphology

# Some constants
OUTPUT_FOLDER = 'dataset/GT_dcm/'
INPUT_FOLDER = 'dataset/original_dcm/'
INPUT_SAVE_FOLDER = 'dataset/preprocess_no_resample/input/'
OUTPUT_SAVE_FOLDER = 'dataset/preprocess_no_resample/output/'

patients = os.listdir(INPUT_FOLDER)
patients.sort()

if '.DS_Store' in patients:
    patients.remove('.DS_Store')

# The output images don't contain all the tags, so copy relevant tags from the
    input
# to the matching output slices.
for patient in patients:
    input_path = INPUT_FOLDER + patient
    input_slices_paths = os.listdir(input_path)
    input_slices_paths.sort()

    output_path = OUTPUT_FOLDER + patient + "_dcm"
    output_slices_paths = os.listdir(output_path)
    output_slices_paths.sort()

    if len(input_slices_paths) != len(output_slices_paths):
        print("Warn:␣Patient", patient, "missing", len(input_slices_paths) -
            len(output_slices_paths), "slices.")

    for i in range(0, len(input_slices_paths)):
        if i < len(output_slices_paths):
            input_dcm = pydicom.dcmread(input_path + '/' + input_slices_paths[
                i])
            output_dcm = pydicom.dcmread(output_path + '/' +
                output_slices_paths[i])
            output_dcm.ImagePositionPatient = input_dcm.ImagePositionPatient
```

```
            output_dcm.SliceLocation = input_dcm.SliceLocation
            output_dcm.PixelSpacing = input_dcm.PixelSpacing
            output_dcm.RescaleIntercept = input_dcm.RescaleIntercept
            output_dcm.RescaleSlope = input_dcm.RescaleSlope
            if output_dcm.ImagePositionPatient == None:
                print("Warn: no image position")
            output_dcm.save_as(output_path + '/' + output_slices_paths[i])

def load_scan(path):
    slices = [pydicom.dcmread(path + '/' + s) for s in os.listdir(path)]
    slices.sort(key = lambda x: float(x.ImagePositionPatient[2]))
    try:
        slice_thickness = np.abs(slices[0].ImagePositionPatient[2] - slices
            [1].ImagePositionPatient[2])
    except:
        slice_thickness = np.abs(slices[0].SliceLocation - slices[1].
            SliceLocation)

    for s in slices:
        s.SliceThickness = slice_thickness

    return slices

def get_pixels_hu(slices):
    image = np.stack([s.pixel_array for s in slices])
    image = image.astype(np.int16)

    image[image == -2000] = 0

    for slice_number in range(len(slices)):
        intercept = slices[slice_number].RescaleIntercept
        slope = slices[slice_number].RescaleSlope

        if slope != 1:
            image[slice_number] = slope * image[slice_number].astype(np.
                float64)
            image[slice_number] = image[slice_number].astype(np.int16)

        image[slice_number] += np.int16(intercept)

    return np.array(image, dtype=np.int16)

def normalize(image, min_bound, max_bound):
    image = (image - min_bound) / (max_bound - min_bound)
    image[image>1] = 1.
```

```
    image[image<0] = 0.
    return image

def zero_center(image, pixel_mean):
    image = image - pixel_mean
    return image

def preprocess_patient(folder, patient, rgb=False):
    scan = load_scan(folder + patient)
    patient_pixels = get_pixels_hu(scan)
    normalized = normalize(patient_pixels, -1000, 300)
    return normalized

all_images = []

for patient in patients:
    preprocessed = preprocess_patient(INPUT_FOLDER, patient)
    print(preprocessed.shape)
    all_images.append(preprocessed)

# Finally, zero-center
means = np.zeros((len(patients)))
for i, scan in enumerate(all_images):
    means[i] = np.mean(scan, axis=(0, 1, 2))
mean_pixel = means.mean()
all_images_zero_centered = [zero_center(image, mean_pixel) for image in
   all_images]

# Save input images
for i in range(0, len(all_images_zero_centered)):
    scan = all_images_zero_centered[i]
    patient_name = patients[i]
    np.save(INPUT_SAVE_FOLDER + patient_name, scan)

all_output_images = []

for patient in patients:
    output = load_scan(OUTPUT_FOLDER + patient + "_dcm")
    output_pixels = np.stack([s.pixel_array for s in output])
    output_pixels = output_pixels.astype(np.float32)
    # Convert output to binary image 0 or 255
    output_pixels = np.dot(output_pixels[..., :3] , [0.299 , 0.587, 0.114])
    # Convert to binary image 0 or 1
    output_pixels[output_pixels>0] = 1
    print(output_pixels.shape)
```

```
    all_output_images.append(output_pixels)

# Save output images
for i in range(0, len(all_output_images)):
    scan = all_output_images[i]
    patient_name = patients[i]
    np.save(OUTPUT_SAVE_FOLDER + patient_name, scan)
```