Razvoj sustava temeljenog na AI algoritmima za segmentaciju moždanih lezija u svrhu dijagnosticiranja multiple skleroze

Hlebar, Josip

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

RAZVOJ SUSTAVA UTEMELJENOG NA ALGORITMIMA UMJETNE INTELIGENCIJE ZA SEGMENTACIJU MOŽDANIH LEZIJA U SVRHU DIJAGNOSTICIRANJA MULTIPLE SKLEROZE

Rijeka, svibanj 2023.

Josip Hlebar
0069066502

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

RAZVOJ SUSTAVA UTEMELJENOG NA ALGORITMIMA UMJETNE INTELIGENCIJE ZA SEGMENTACIJU MOŽDANIH LEZIJA U SVRHU DIJAGNOSTICIRANJA MULTIPLE SKLEROZE

Mentor: prof. dr. sc. Zlatan Car

Rijeka, svibanj 2023.

Josip Hlebar
0069066502

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 15. rujna 2022.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Primjena umjetne inteligencije**
Grana: **2.03.06 automatizacija i robotika**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Josip Hlebar (0069066502)**
Studij: Diplomski sveučilišni studij elektrotehnike
Modul: Automatika

Zadatak: **Razvoj sustava temeljenog na AI algoritmima za segmentaciju moždanih lezija u svrhu dijagnosticiranja multiple skleroze / Development of an AI system for the segmentation of brain lesions for the purpose of diagnosing multiple sclerosis**

Opis zadatka:

Izvršiti pregled literature na temu primjene metoda umjetne inteligencije u području dijagnostike i liječenja multiple skleroze i ostalih oboljenja centralnog živčanog sustava. Izraditi više modela za segmentaciju lezija, te varirati hiperparametre istih. Izvršiti usporedbu dobivenih rezultata i simulacija.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 15. rujna 2022.

Mentor:

Prof. dr. sc. Zlatan Car

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Viktor Sučić

IZJAVA

Sukladno članku 9. Pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija Tehničkog fakulteta Sveučilišta u Rijeci, izjavljujem da sam izradio ovaj diplomski rad samostalno, koristeći vlastito znanje i navedenu literaturu, u razdoblju od 15. rujna 2022. do 16. svibnja 2023.

Rijeka, svibanj 2023.

Josip Hlebar

Za nesebičnu pomoć i brigu pri izradi ovog rada zahvaljujem se prof. dr. sc. Zlatanu Caru i asistentu Sandiju Baressiju Šegoti. Također se zahvaljujem svojim roditeljima na bezuvjetnoj podršci koja mi je omogućila bezbrižno školovanje.

Sadržaj

1. Uvod

Neuronske mreže su računalni sustavi koji predviđaju podatke na osnovi postojećih podataka. Ideja na kojoj se temelji arhitektura neuronske mreže je u analogiji s funkcijama mozga, neurona i sinapsi, sposobnih učiti i donositi odluke. Neuronske mreže i umjetna inteligencija općenito su danas široko primijenjeni kao praktična tehnološka rješenja na mnogim područjima. Zahvaljujući svojoj snažnoj sposobnosti predviđanja vrijednosti, neuronske mreže se primjenjuju za modeliranje i identifikaciju složenih nelinearnih sustava i optimizaciju automatskog upravljanja, od robotike do transportnih sustava. Korištenjem konvolucijskih mreža za obradu slike, neuronske mreže mogu se primijeniti za semantičku segmentaciju. Semantička segmentacija je tehnika obrade slike koja se koristi u neuronskim mrežama za razdvajanje slike na različite dijelove ili objekte. Cilj semantičke segmentacije je klasificirati svaki piksel na slici u određenu kategoriju. U primjeni medicinske dijagnostike, semantička segmentacija se često koristi za razdvajanje različitih tkiva u medicinskim slikama, što omogućava lakše prepoznavanje i dijagnosticiranje bolesti.

Multipla skleroza (MS) je kronična, autoimuna i neurodegenerativna bolest koja zahvaća središnji živčani sustav. MS mijenja morfologiju i strukturu mozga uslijed propadanja mijelinskih ovojnica što uzrokuje nastajanje lezija. MS se manifestira kroz mnoge simptome različitog stupnja, od blage ukočenosti do potpune oduzetosti, sljepoće itd. Identifikacija i segmentacija MS lezija je neophodan korak kod karakteriziranja bolesti i kvantitativne analize koja pruža uvid u napredovanje bolesti i moguće terapije. S obzirom na to da je "ručna" segmentacija lezija dugotrajan i mukotrpan proces čija efikasnost ovisi o iskustvu promatrača, razvoj metoda za automatsku ili poluautomatsku segmentaciju je od velikog kliničkog i inženjerskog značaja. Sustavi za automatsko segmentiranje MS lezija mogu pomoći kod detekcije dijagnostičkog kriterija za bolest koji sadrži prostorni raspored lezija u snimkama magnetske rezonancije i nastajanju novih lezija. Konačno, mogu smanjiti radno opterećenje liječnika i poboljšati točnost pružanjem objektivnih rezultata.

Ovaj rad istražuje teoriju umjetne inteligencije, neuronskih mreža i strojnog učenja te mogućnosti njihove primjene za automatsku segmentaciju MS lezija iz snimaka magnetske rezonancije. Cilj rada je razviti učinkovit sustav za automatsku segmentaciju lezija koji bi mogao pomoći u karakteriziranju i praćenju bolesti, te olakšati rad medicinskih stručnjaka i poboljšati dijagnostiku.

2. Magnetska rezonancija

Magnetska rezonancija (MR) proizvodi slike kontrastne rezolucije mekih tkiva bez potrebe za uvođenjem invazivnih postupaka. Ne uzrokuje zračenje ljudskog tijela te je u potpunosti siguran proces. Također, MR snimke pružaju visoku rezoluciju i točno pozicioniranje mekih tkiva te su osjetljive na različite karakteristike bolesti. Stoga su izričito pogodne za dijagnozu bolesti mozga.

2.1. Teorija rada magnetske rezonancije

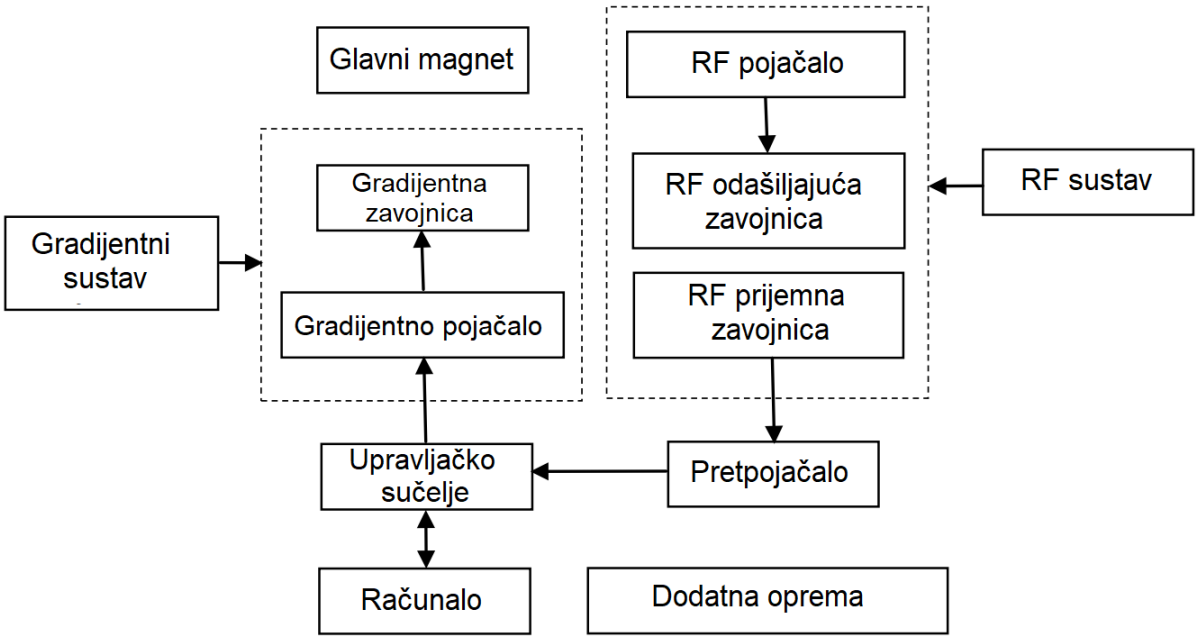
Stanične jezgre u tkivima ljudskog tijela sastoje se od protona i neutrona. Protoni su pozitivnog električnog naboja, a neutroni su neutralni pa je ukupni naboj jezgre pozitivan. Nabijena jezgra vrti se oko svoje osi, te sukladno zakonu elektromagnetske indukcije, uzrokuje vertikalni magnetski moment. Ovisno o prirodi njihove vrtnje, jezgre se mogu podijeliti na magnetske i nemagnetske jezgre. Ako je broj protona ili neutrona neparan, jezgra je magnetska. Jezgra je nemagnetska ako i samo ako je broj protona i neutrona unutar jezgre paran. Za MR snimanje mogu se koristiti samo magnetske jezgre [1]. U teoriji, za MR se mogu koristiti magnetske jezgre svih kemijskih elemenata, ali se trenutno u praksi koriste samo jezgre vodika s jednim neutronom i protonom zbog sljedećih razloga [2]:

1. Od svih atoma u tijelu, vodik ima najveću molarnu koncentraciju: ljudsko tijelo sadrži mnoštvo vode i ugljikovodika (šećeri, proteini, mast, itd.). Stoga je količina vodikovih jezgri u ljudskom tijelu najveća (sačinjava više od dvije trećine ukupnih jezgri). Dovoljan broj vodikovih jezgri veoma je pogodan za generiranje signala snažne magnetske rezonancije te se MR snimanje uglavnom oslanja na vodikove jezgre u vodi i masti.

2. U usporedbi s jezgrama preostalih elemenata u ljudskom tijelu, vodikova jezgra je najveće magnetske susceptibilnosti. Visoki stupanj magnetizacije generira visoki intenzitet signala kod snimanja magnetske rezonancije.

3. Biološke karakteristike vodikovih jezgri su očite: zbog puno vode i ugljikovodika u tijelu, vodikove jezgre prisutne su u raznim tkivima tijela. Korištenjem vodika za snimanje magnetske rezonancije postiže se mogućnost opisivanja različitih karakteristika raznolikih tkiva ljudskog tijela. Stoga, vodikove jezgre pružaju dobar biološki pregled.

Tipičan sustav za MR snimanje sastoji se od pet dijelova: glavni magnet, gradijentni sustav, radio frekvencijski (RF) sustav, računalo i dodatna oprema [1]. Slika 2.1 prikazuje strukturu sustava za MR snimanje.

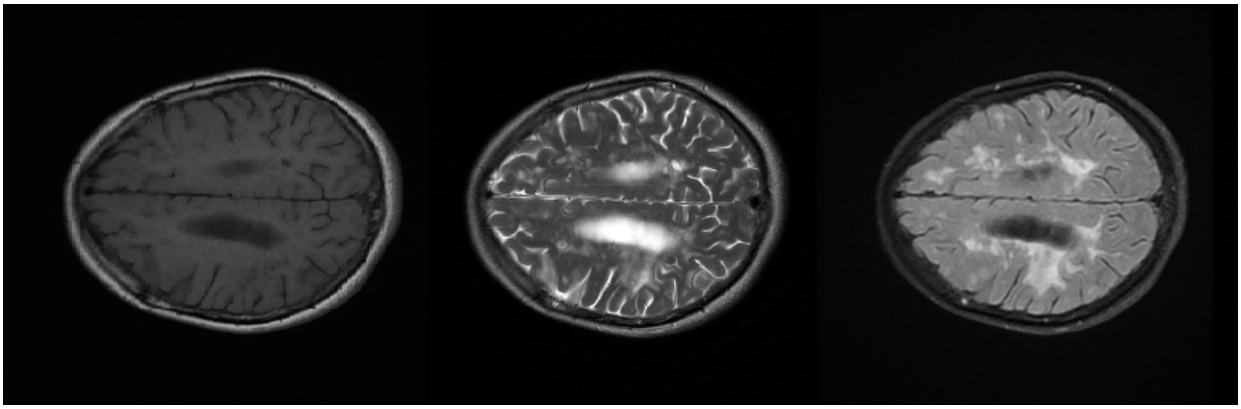


Slika 2.1. Struktura sustava za MR snimanje.

Glavni magnet je uređaj koji proizvodi glavno magnetsko polje i njegova izvedba direktno utječe na kvalitetu MR snimke. Glavni magnet može biti permanentni magnet ili elektromagnet. Gradijentni sustav koristi se za generiranje linearnog gradijentnog magnetskog polja koje služi za prostorno kodiranje MR signala. Također, promjena gradijentnog polja uzrokuje MR jeku. RF sustav sastoji se od RF zavojnica čija je uloga prijenos generiranog signala MR jeke. Računalo upravlja cjelokupnim operacijama sustava za MR snimanje, uključujući impulsne pobude, prikupljanje signala, operacije nad podacima i prikaz slike. Dodatna oprema odnosi se na sustave koji osiguravaju ispravan rad sustava za MR snimanje.

Kada se ljudsko tijelo nalazi u magnetskom polju, dolazi do magnetizirajućeg fenomena: magnetski momenti uslijed vrtnje vodikovih jezgri skreću uzduž orijentacije glavnog magnetskog polja. Smjer većine magnetskih momenata poklapa se sa smjerom glavnog magnetskog polja u niskom energetskom stanju dok se momenti manjeg broja vodikovih jezgri protive orijentaciji magnetskog polja te ostaju u pobuđenom energetskom stanju. Ukupni vektor magnetskog momenta (longitudinalni makro-magnetski moment) zadržava isti smjer kao i glavno magnetsko polje. Međutim, veličina vektora makro-magnetskog momenta je puno manje od vektora glavnog magnetskog polja i jednakog su smjera. Nakon njihove superpozicije, makro-magnetski moment neraspoznatljiv je unutar glavnog magnetskog polja. Snaga impulsnog signala oslobođenog longitudinalnim magnetskim momentom je preslaba za transmisiju RF zavojnicama. Stoga, nije moguće razlikovati različite tipove tkiva u ovisnosti o makro vektorima uzrokovanim različitim količinama jezgri vodika. Kako bi se mogla raspoznati drugačija tkiva, primjenjuju se RF impulsi na ljudsko tijelo u glavnom magnetskom polju. Energija impulsa prenosi se na vodikove jezgre nižeg energetskog stanja čime se postiže njihova pobuda u više energetsko stanje. Opisana pojava naziva se

fenomenom magnetske rezonancije. Smjer vektora makro-magnetskog momenta mijenja se uslijed nadolazećih impulsa. Što je veća energija RF impulsa, to je veća vrijednost kuta između vektora momenta i glavnog polja. Nakon apsorpcije, vodikove jezgre nalaze se u visokom, nestabilnom energetskom stanju. Uklanjanjem RF impulsa, jezgre se vraćaju u početno stanje te se oslobađa energija. Tijekom procesa oporavka, emitiraju se elektromagnetski impulsi koje primaju RF zavojnice za snimanje. Postupak oporavka vektora makro-magnetskog momenta nakon nestajanja RF impulsa naziva se procesom relaksacije. Relaksacija se dijeli na lateralnu (nestajanje poprečnih magnetskih komponenti) i longitudinalnu (uzdužne magnetske komponente se postepeno povećavaju na početnu vrijednost). Lateralne i longitudinalne relaksacije sukladno odgovaraju T2 i T1 karakteristikama tkiva. Različita tkiva imaju značajno drugačije T2 i T1 karakteristike. Ovisno o obliku uzbudnog impulsnog niza signala, postižu se MR snimke koje odražavaju različite karakteristike bioloških svojstava tkiva. Cjelokupna MR snimka sadrži par desetaka slika koje prikazuju tkivo kroz niz isječaka na raznolikim dubinama snimanog objekta. Najčešće korišteni tipovi MR snimki su T1-mjerena, T2-mjerena i FLAIR (engl. Fluid Attenuated Inversion Recovery). T1-mjerena snimka odražava brzine oporavka magnetskog momenta kod longitudinalne relaksacije, daje visoki intenzitet signala masti (mast izgleda svjetlo). T2-mjerena snimka reflektira brzinu nestajanja magnetskog momenta pri lateralnoj relaksaciji, voda i tekućine su svjetlije. FLAIR je inačica T1 snimanja kojim se postiže potiskivanje signala vode. Ovaj oblik je najkorisniji za otkrivanje promjena u tkivu mozga (ožiljci, demijelinizacija, itd.). Slika 2.2 prikazuje usporedbu različitih tipova MR snimki istoga pacijenta.



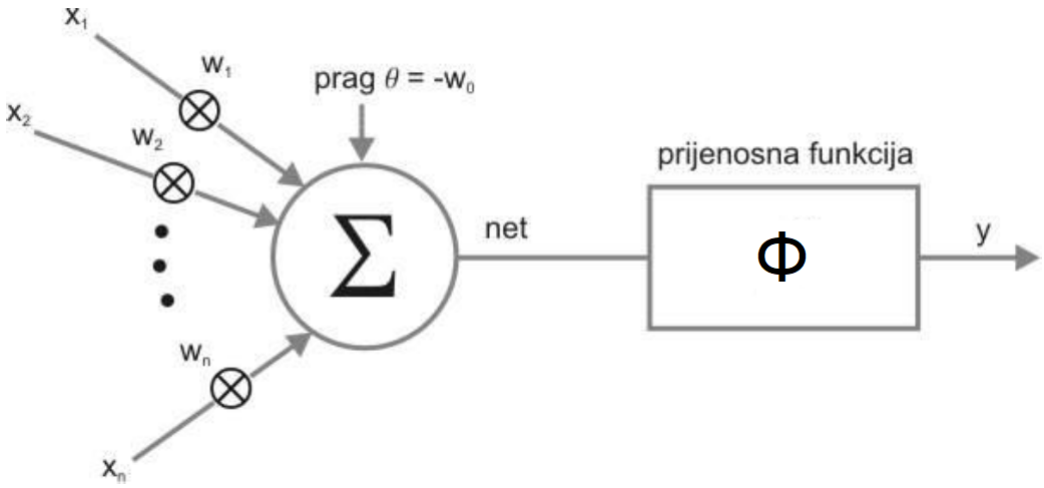
Slika 2.2. Različiti tipovi MR snimaka (s lijeva na desno: T1, T2 i FLAIR).

3. Umjetne neuronske mreže

Ljudski mozak funkcionira na principu paralelnog rada velikog broja ($\sim 10^{11}$) međusobno povezanih neurona. Najvažnija karakteristika mozga je neuroplastičnost - struktura živčanog sustava konstantno se mijenja, pri čemu se reorganiziraju veze između neurona i mijenjaju se njihove funkcije u procesu zvanom učenje. S obzirom na to da je umjetna inteligencija utemeljena na biološkim živčanim sustavima, pri njenom opisivanju koriste se ekvivalentni pojmovi.

3.1. Struktura umjetne neuronske mreže

Umjetna neuronska mreža je slične složenosti kao biološki mozak te se sastoji od proizvoljnog broja povezanih osnovnih jedinica za obradu informacija zvanim umjetnim neuronima. Odnosno, umjetni neuron predstavlja čvorište koje na temelju ulaznih vrijednosti pristiglih iz ostalih čvorova generira izlaznu vrijednost s obzirom na svoju prijenosnu (aktivacijsku) funkciju. Aktivacijska funkcija određuje način linearnog ili nelinearnog preslikavanja ulaznih u izlazne vrijednosti te se označava sa ϕ. Signali se prenose kroz umjetne sinapse s dodijeljenim težinskim faktorima, w_i, koji određuju amplitudu ulazne vrijednosti. Slika 3.1 prikazuje blokovski prikaz umjetnog neurona iz kojeg je vidljiv princip njegovog rada.



Slika 3.1. Blokovski model umjetnog neurona [3].

Iz prethodne slike je uočljivo da se ulazne vrijednosti u matematičkom smislu množe s težinskim faktorima, a potom zbrajaju u jedinstveni signal koji se obrađuje pomoću aktivacijske funkcije. Izlazni signal iz sumatora se definira sljedećim izrazom:

$$net = \sum_{i=1}^{n} w_i x_i - \theta = w^T x - \theta, \tag{3.1}$$

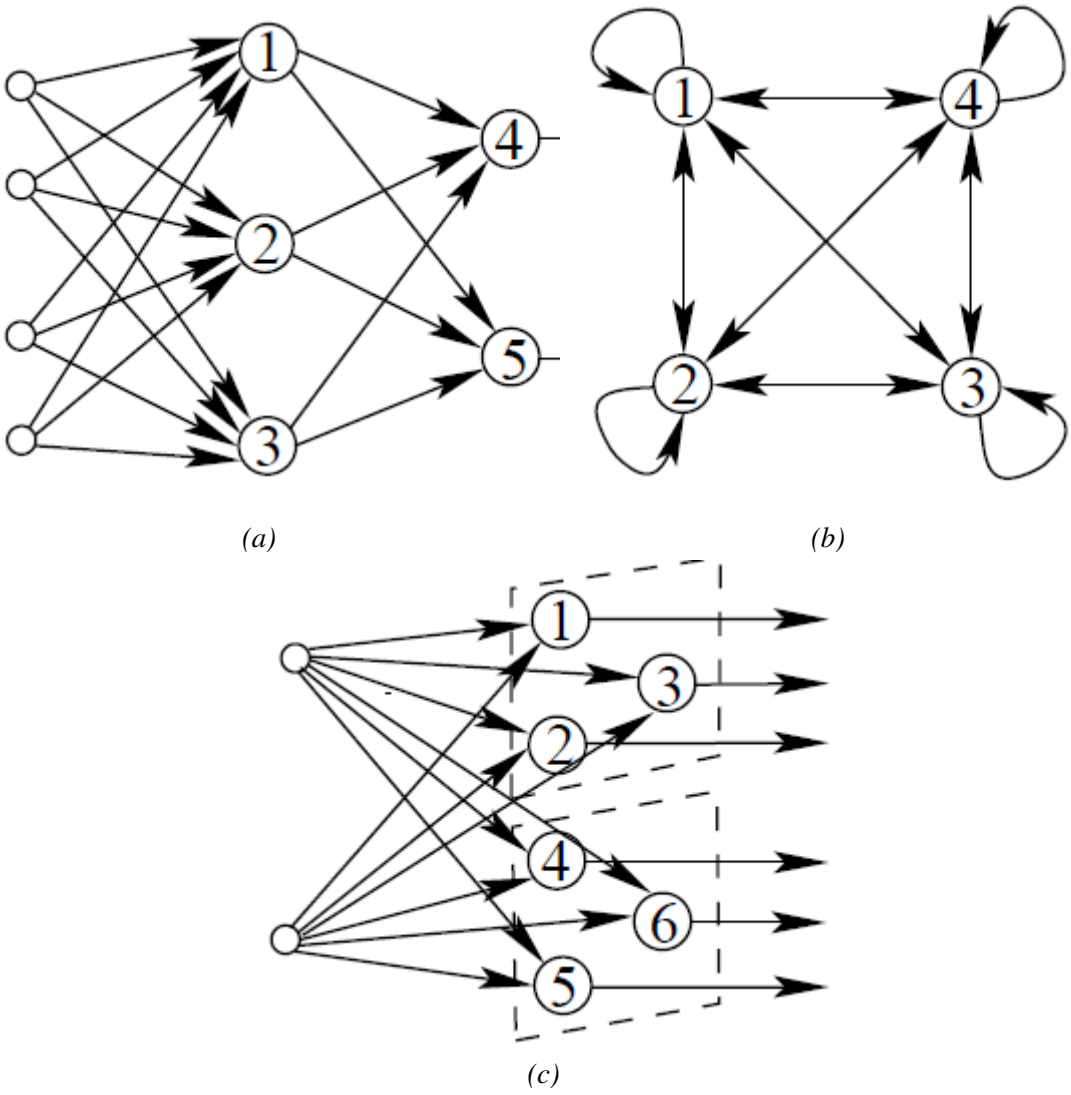
gdje n predstavlja broj ulaznih vrijednosti, x_i predstavlja i-ti ulaz, a w_i težinski faktor i-tog ulaza, $w = (w_1, ..., w_n)^T$, $x = (x_1, ..., x_n)^T$. Vrijednost praga koja se pridodaje ulaznim vrijednostima, θ, posljedica je neusklađenosti korištenih uređaja, te ju je teško kontrolirati. Konačno, izlazni signal iz neurona se dobiva djelovanjem aktivacijske funkcije te se izražava kao:

$$y = \phi(net) = \phi\left(\sum_{i=1}^{n} w_i x_i - \theta\right). \tag{3.2}$$

Prethodni opis umjetnog neurona je McCulloch-Pittsov model koji je pogodan za implementaciju u integriranim krugovima pri čemu se sinapse s pripadnim težinskim faktorima ostvaruju pomoću promjenjivih otpornika dok je neuron realiziran korištenjem operacijskog pojačala.

3.1.1. Podjela umjetnih neuronskih mreža

Neuronska mreža je karakterizirana svojom arhitekturom (topologijom), obilježjima čvorova i pravilima učenja. Arhitektura mreže se predstavlja pomoću matrice težinskih faktora $W = [w_{ij}]$, gdje w_{ij} označava težinski faktor između čvora i i j. Postavljanjem pojedinačnih vrijednosti w_{ij} u nulu se postižu različite topološke strukture mreže. Umjetne neuronske mreže se generalno dijele na mreže s ili bez povratne veze. Popularne mrežne topologije su potpuno povezane višeslojne mreže bez povratne veze ili MLP (engl. Multi Layer Perceptron), mreže s povratnom vezom (engl. recurrent networks) i ljestvičaste mreže (engl. lattice networks). U MLP mrežama se osim ulaznog i izlaznog sloja neurona nalazi barem jedan skriveni sloj, pri čemu je svaki čvor sloja povezan sa svim čvorovima susjednog sljedećeg sloja. Kod ovakve strukture ne postoje veze između neurona istog sloja te je protok informacija jednosmjeran. Povratne veze u neuronskim mrežama pružaju bolju kvalitetu, ali i povećanu složenost analize. Korištenjem elementa za kašnjenje u kombinaciji s povratnim vezama ostvaruje se dodatna memorija za obradu ulaznih podataka, što omogućuje primjenu na zadatke poput prepoznavanja rukopisa ili govora [3]. Ljestvičaste mreže sastoje se od višedimenzionalnih polja neurona pri čemu se svakom neuronu u polju dodjeljuju svi ulazni signali. Slike 3.2(a) do (c) prikazuju jednostavne primjere opisanih mrežnih struktura.



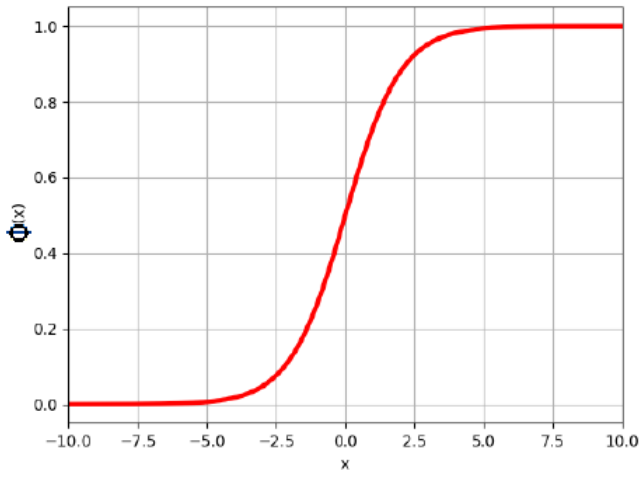
Slika 3.2. (a) MLP mreža, (b) mreža s povratnim vezama, (c) ljestvičasta mreža [4].

3.2. Aktivacijske funkcije

Kao što je prethodno spomenuto, aktivacijske funkcije definiraju način na koji se zbrojene vrijednosti ulaznih signala preslikavaju u izlaz iz čvora ili čvorova sloja neuronske mreže. Svi skriveni slojevi mreže uobičajeno koriste istu aktivacijsku funkciju dok izlazni sloj implementira različitu funkciju koja ovisi o vrsti predviđanja za koju se upotrebljava model. Linearne aktivacijske funkcije rijetko se implementiraju zbog direktne proporcionalnosti između ulaza i izlaza, uslijed čega se model mreže ponaša linearno što onemogućuje obradu kompleksnijih, nelinearnih skupova podataka. Stoga se uglavnom koriste nelinearne aktivacijske funkcije od kojih je pogodno izdvojiti sigmoidalnu (logističku), tanh i ReLU funkciju. Njihovo najvažnije svojstvo je derivabilnost koja omogućuje unatražno širenje informacija o grešci modela tijekom procesa treniranja. Sigmoidalna funkcija se definira kao:

$$\phi(x) = \frac{1}{1 + e^{-x}}, \tag{3.3}$$

dok je njen graf prikazan na Slici 3.3.

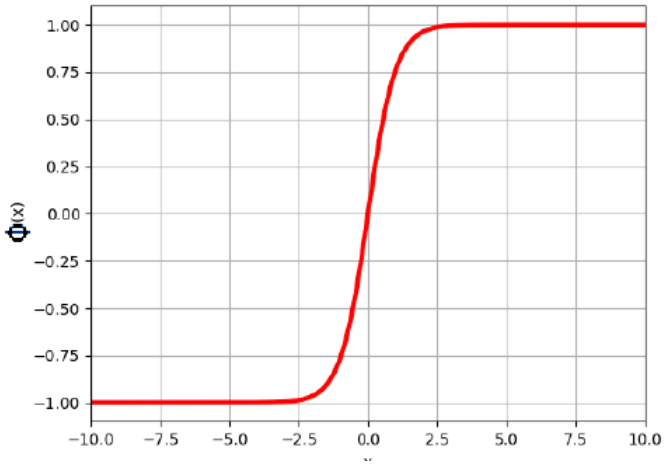


Slika 3.3. Sigmoidalna funkcija [3].

Iz prethodnog grafa je vidljivo da sigmoidalna funkcija poprima vrijednosti u rasponu od 0 do 1. Stoga se koristi u modelima koji služe za predviđanje vjerojatnosti određenog izlaza. Nedostatak ove funkcije je problem nestajućeg gradijenta. Naime, maksimalna vrijednost derivacije sigmoidalne funkcije je 0.25. Stoga se pri širenju informacija unatrag pri svakom sloju mreže smanjuje veličina prenošene greške za barem 75%. U slučaju mreža s puno slojeva, opisana pojava uzrokuje ograničenje promjene težinskih faktora u slojevima koji su blizu ulaznom sloju. Hiperbolični tangens ili tanh funkcija je slična sigmoidalnoj, uz povećani raspon vrijednosti od -1 do 1. Izraz koji ju opisuje glasi:

$$\phi(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{3.4}$$

Slika 3.4 prikazuje graf funkcije hiperboličnog tangensa.

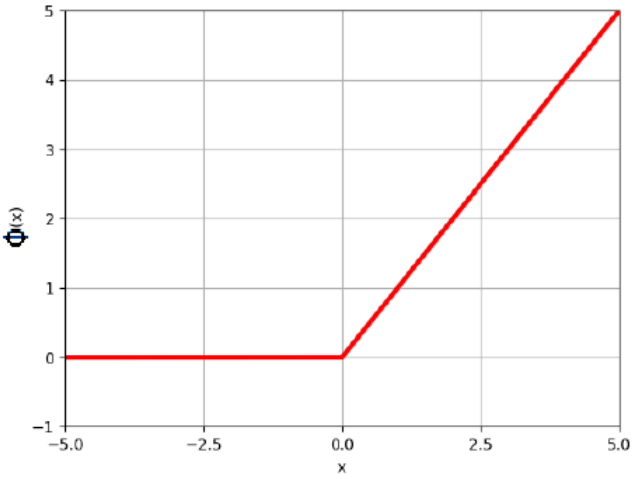


Slika 3.4. Hiperbolični tangens [3].

Najčešće se koristi ReLU (engl. Rectified Linear Unit). Naime, navedena funkcija i njena derivacija su oboje monotoni te je gradijent u rasponu između 0 i 1. Također, njena prednost je veoma jednostavno računanje zbog čega nadmašuje preostale funkcije. ReLU funkcija se izražava kao:

$$\phi(x) = \begin{cases} 0 & \text{za} \quad x < 0 \\ x & \text{za} \quad x \geq 0 \end{cases} . \tag{3.5}$$

Graf ReLU funkcije je prikazan na slici 3.5.



Slika 3.5. ReLU funkcija [3].

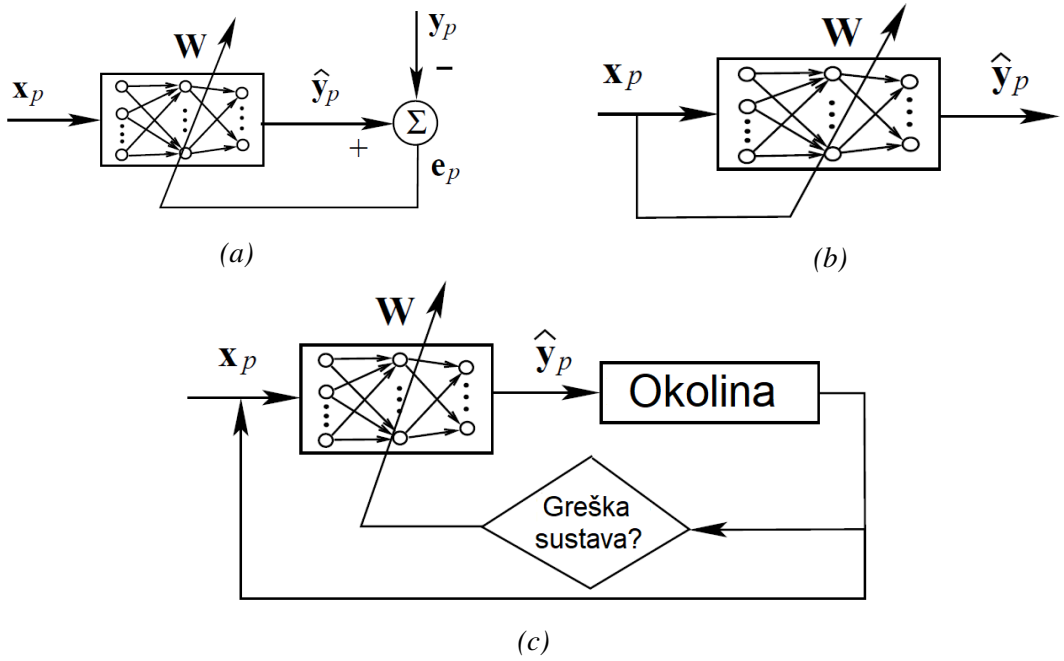
3.3. Temelji strojnog učenja

Strojno učenje (engl. machine learning) je područje umjetne inteligencije koje se bavi stvaranjem računalnih programa koji se automatski poboljšavaju s iskustvom. Učenje je temeljna sposobnost umjetnih neuronskih mreža. Pravila učenja ili treniranja su numerički algoritmi pomoću kojih se pronalaze ispravni težinski faktori i/ili preostali parametri mreže. Učenje neuronske mreže može se promatrati kao problem nelinearne optimizacije za pronalaženje skupa mrežnih parametara koji minimizira funkciju gubitaka za promatrani skup podataka. Mreže se treniraju u epohama, pri čemu epoha predstavlja jedan potpun ciklus tijekom kojega su svi podaci za treniranje predstavljeni mreži, te su procesuirani korištenjem algoritma za učenje. Proces učenja kontrolira se definiranjem kriterija koji odlučuje kada je proces gotov (maksimalno vrijeme treniranja, maksimalan broj iteracija, ciljano odstupanje aproksimirane funkcije itd.) [4][5].

3.3.1. Metode strojnog učenja

Metode učenja konvencionalno se dijele na učenje s učiteljem (engl. supervised learning), učenje bez učitelja (engl. unsupervised learning) i ojačano učenje (engl. reinforcement learning).

Slike 3.6(a) do (c) prikazuju blokovske sheme navedenih metoda učenja, gdje x_p i y_p predstavljaju ulazne i izlazne vrijednosti p-tog uzorka skupa podataka za treniranje, a $\hat{y}$ predstavlja dobiveni izlaz iz neuronske mreže za p-ti ulaz.



Slika 3.6. (a) Učenje s učiteljem, (b) učenje bez učitelja, (c) ojačano učenje.

Učenje s učiteljem prilagođava parametre mreže direktnim uspoređivanjem dobivenih izlaznih vrijednosti sa željenim vrijednostima. Primjenjuje se pri klasifikaciji, regresiji ili strukturiranom učenju, kod kojega se predviđa određena struktura, a ne numeričke vrijednosti. Nadzirano učenje može se promatrati kao sustav s povratnom petljom gdje je pogreška signal povratne veze. Mjera za pogrešku, koja prikazuje razliku između dobivenog i željenog odziva za određeni ulazni vektor, koristi se za vođenje procesa učenja. Mjera pogreške najčešće se definira kao srednja vrijednost kvadriranih udaljenosti ili MSE (engl. Mean Squared Error). Izražava se na sljedeći način:

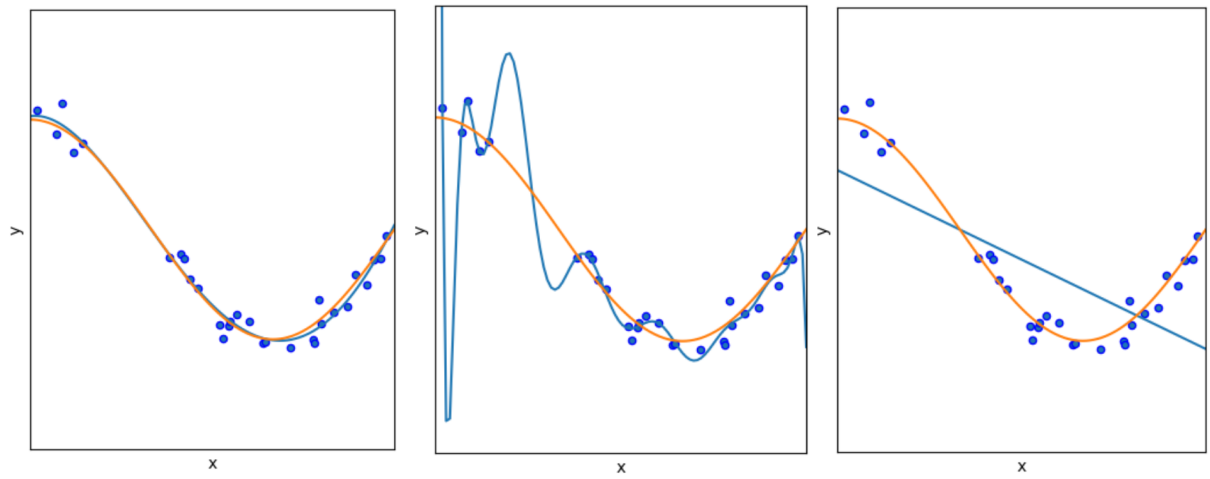
$$E = \frac{1}{N} \sum_{p=1}^{N} |y_p - \hat{y_p}|^2, \tag{3.6}$$

gdje N predstavlja broj para uzoraka (x_p, y_p) u skupu podataka za treniranje. Greška E računa se isponova na kraju svake epohe. Tj. učenje je iterativan postupak koji prestaje kada je vrijednost E dovoljno malena ili kada se ostvari kriterij neuspješnosti. Vrijednost pogreške se smanjuje korištenjem optimizacijskih algoritama poput postupnog opadanja (stohastični gradijenti spust - SGD), metode koja uvijek konvergira prema lokalnom minimumu u blizini početnog rješenja za parametre mreže. Za modele koji obavljaju segmentaciju slike, zbog svoje brzine i stabilnosti, najčešće se koristi Adam optimizacijski algoritam. Adam je adaptivni optimizacijski algoritam koji kombinira prednosti drugih optimizacijskih algoritama, kao što su SGD. Primjenjuje postupno opadanje stope učenja koja se prilagođava tijekom procesa treninga, ovisno o procijenjenom prvom

i drugom momentu gradijenta. Korištenje prvog momenta (srednje vrijednosti gradijenta) i drugog momenta (varijanca gradijenta) pomaže u efikasnijoj pretrazi prostora parametara kako bi se brže pronašli globalni minimumi.

3.3.2. Generalizacija i učenje

S aproksimativnog stajališta, učenje je rekonstrukcija hiperpovršine bazirana na postojećim primjerima, dok je generalizacija procjena vrijednosti hiperpovršine gdje ne postoje primjeri. Matematički, proces učenja je nelinearni proces uklapanja u krivulju, dok je generalizacija interpolacija i ekstrapolacija ulaznih podataka. U smislu strojnog učenja, generalizacija opisuje koliko je dobro istrenirani model u stanju predvidjeti podatke. Vježbanje generaliziranog strojnog modela učenja znači općenito da funkcionira za svaki skup neviđenih podataka. Cilj učenja neuronskih mreža nije naučiti točnu reprezentaciju podataka za obuku, već izgraditi statistički model procesa koji generira podatke. Kada je mreža izvježbana s previše primjera, parametara ili epoha, može proizvesti dobar rezultat s obzirom na podatke za treniranje, ali istovremeno imati slabu generalizacijsku sposobnost. Taj problem uvodi dilemu između odstupanja i varijance kod odabira modela mreže. Odnosno, zahtjevi za malenim odstupanjem i varijancom se sukobljavaju te se mora odrediti kompromis. Varijanca predstavlja različitost predviđenih vrijednosti postignutih modelom strojnog učenja. Odstupanje predstavlja udaljenost stvarne i predviđene vrijednosti. Visoko-odstupajući model ima predviđene vrijednosti (u prosjeku) daleko od stvarnih vrijednosti. Kod visoko-varijantnog predviđanja, predviđene vrijednosti se znatno razlikuju. Rezultati predviđanja modela strojnog učenja su između nisko-odstupajućeg i nisko-varijantnog, nisko-odstupajućeg i visoko-varijantnog, visoko-odstupajućeg i nisko-varijantnog, te visoko-odstupajućeg i visoko-varijantnog. Nisko-odstupajući, visoko-varijantni model se naziva prekomjerno podudarajući, a visoko-odstupajući, nisko-varijantni se naziva nedovoljno podudarajući. Pomoću generalizacije pronalazimo najbolji rezultat između prekomjerno i nedovoljno podudarajućeg modela. Slika 3.7(a) prikazuje primjer rezultata prikladno generaliziranog modela dok (b) i (c) prikazuju primjere prekomjernog te nedovoljnog uklapanja. Točke predstavljaju uzorke korištene pri obuci, narančastom bojom je označen graf ciljane funkcije, a plava krivulja je dobivena aproksimacija.



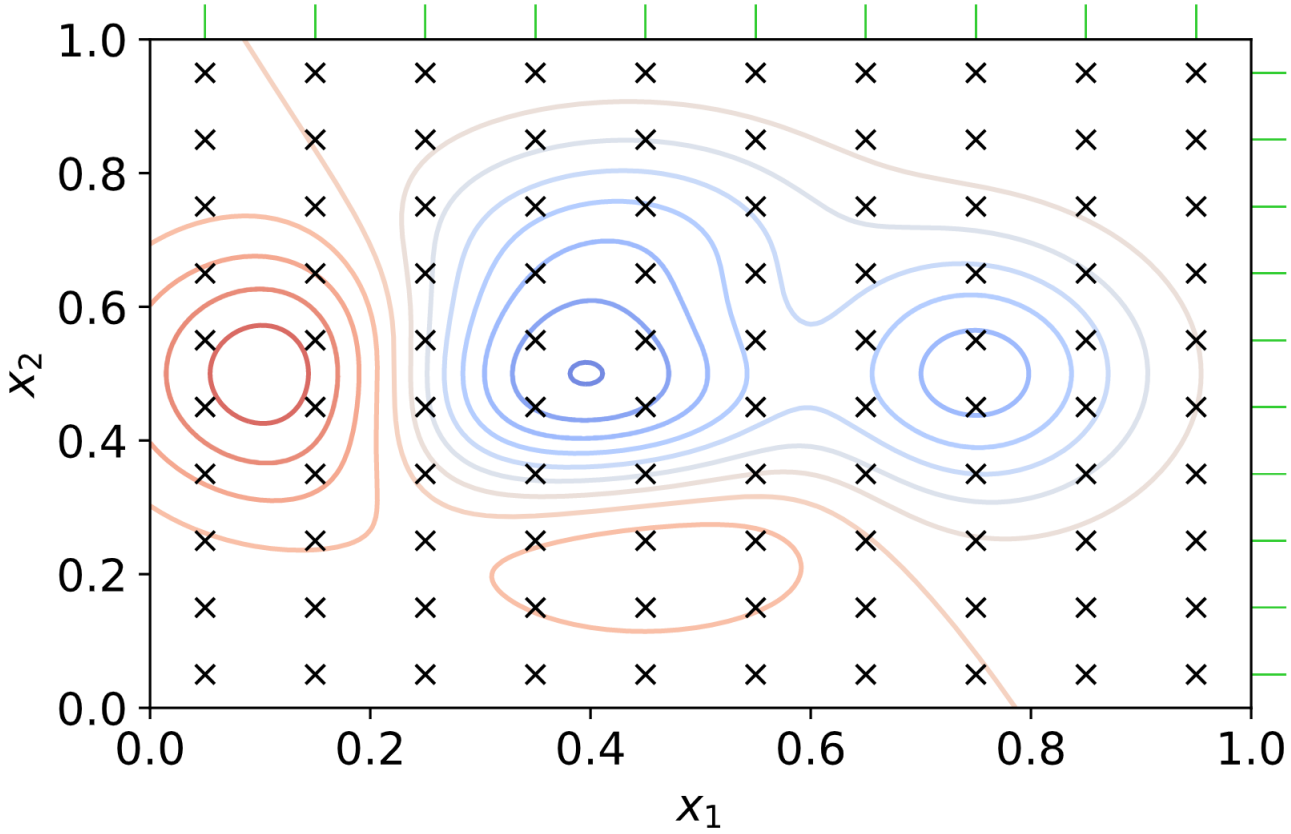
Slika 3.7. (a) Ispravno podudaranje, (b) prekomjerno podudaranje, (c) nedovoljno podudaranje.

Tri su različita načina osiguravanja generalizacije modela strojnog učenja. Prvi se odnosi na skup podataka koji treba sadržavati različitost. Skup podataka ne treba sadržavati veliku količinu podataka, već različite uzorke, čime se postiže bolja generalizacija. Također, za vrijeme učenja je dobro koristiti tehniku unakrsne validacije, kao što su K-preklapanje ili Monte-Carlo unakrsna validacija. Drugi se odnosi na algoritam strojnog učenja, koji se različito ponaša u smislu prekomjernog i nedovoljnog podudaranja. Prekomjerno podudaranje je povezano s nelinearnim, neparametriranim algoritmom. S druge strane, nedovoljno podudaranje je povezano s modelima koji su prejednostavni da povežu složene uzorke podataka. Treći se odnosi na složenost modela strojnog učenja. Kada je presložen, obično je sklon prekomjernom podudaranju. Metode pojednostavnjenja modela strojnog učenja su regularizacijske metode.

3.3.3. Određivanje hiperparametara

Nedostatak neuronskih mreža je potrebno podešavanje hiperparametara. Hiperparametri su vrijednosti koje određuju ponašanje neuronske mreže. Pod hiperparametre spada broj skrivenih slojeva, broj neurona u slojevima, korištene aktivacijske funkcije, stopa učenja, broj epoha, inicijalne vrijednosti težinskih faktora, itd. S obzirom na to da neuronske mreže značajno ovise o sadržanim hiperparametrima, njihovo određivanje je najosnovniji zadatak kod strojnog učenja. Isti model strojnog učenja zahtijeva različite hiperparametre za svaki korišteni set podataka, stoga ne postoji egzaktna metoda predviđanja performansi neuronskih mreža ovisno o njenim hiperparametrima. Jedna od metoda optimizacije hiperparametara je pretraživanje svake točke prostora rješenja (engl. grid search). Kod navedene metode se proizvoljno definira konačan broj skupova vrijednosti hiperparametara. Svaki skup predstavlja točku u prostoru. Zatim se model učenja trenira sa svakim određenih skupom hiperparametara te se njihova performansa ocjenjuje i uspoređuje. Slika 3.8 prikazuje primjer pretraživanja prostora rješenja pri čemu se promatraju dva hiperparametra. Za svaki hiperparametar je određeno 10 različitih vrijednosti. Stoga je potrebno ispitati i

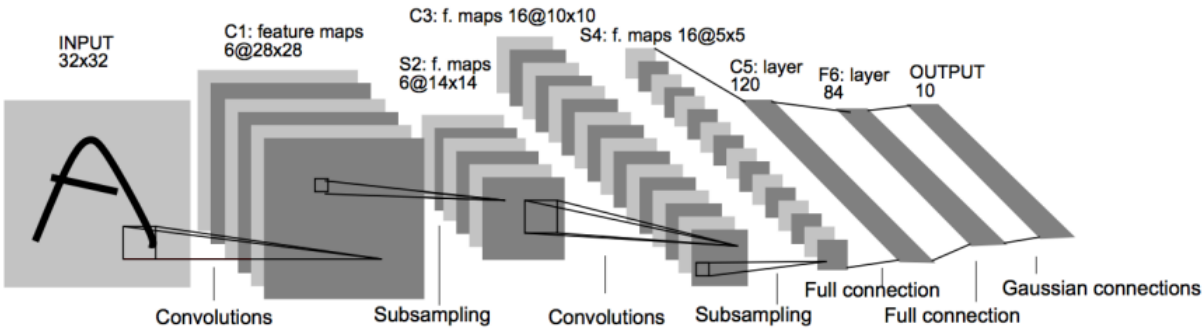
usporediti 100 mogućih kombinacija. Plava boja označava područja dobrih rezultata, dok crvena boja označava područja s najgorim rezultatima.



Slika 3.8. Pretraživanje prostora rješenja [6].

3.4. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (engl. Convolutional neural networks ili CNN) predstavljaju vrstu specijaliziranih neuronskih mreža koje se koriste za procesuiranje nestrukturiranih podataka, naročito pogodne za obradu slike, zvuka i teksta. CNN se može definirati kao inačica višeslojnih neuronskih mreža bez povratne veze koja osim jednog ulaznog i izlaznog sloja, sadrži jedan ili više skrivenih slojeva. Na ulazni sloj dovode se trodimenzionalni podaci poput slike te se očitavaju vrijednosti njenih piksela. Primjerice, ulaz u CNN može biti slika u boji predstavljena dimenzijama 128x128x3. Odnosno, njena širina je 128 piksela, visina 128 i dubina 3 (RGB slika - za svaku boju jedan kanal). Skriveni međuslojevi sadrže naizmjenični niz za CNN specifičnih, jednostrukih ili višestrukih konvolucijskih slojeva i slojeva sažimanja (engl. pooling layer). Nakon svakog konvolucijskog sloja (prije sloja sažimanja) primjenjuje se ReLU aktivacijska funkcija koja uvodi svojstvo nelinearnosti u mrežu. Na izlaznom dijelu mreže najčešće se nalaze potpuno povezani slojevi koji služe za klasifikaciju. Slika 3.9 prikazuje primjer arhitekture konvolucijske neuronske mreže.



Slika 3.9. Prikaz arhitekture LeNet-5 konvolucijske neuronske mreže [7].

3.4.1. Konvolucijski sloj

Konvolucijski sloj je glavni sastavni dio svake konvolucijske neuronske mreže te se sastoji od dva dijela. U prvome dijelu izvodi se operacija konvolucije slike iz prethodnog sloja s određenim brojem filtara, generirajući izlaznu sliku zvanu mapa značajki. Nakon konvolucije, u drugome dijelu, primjenjuje se aktivacijska funkcija. Najčešće se modeliraju konvolucijske neuronske mreže koje uzastopno primjenjuju više konvolucijskih slojeva pri čemu je izlaz iz jednog sloja ulaz u sljedeći. Što se dublje konvolucijski sloj nalazi u neuronskoj mreži, tim je sposobniji prepoznavati kompleksnije značajke. Primjerice, početni slojevi mreže uče razaznati rubove objekata dok se u dubljim slojevima uči prepoznavanje geometrijskih oblika, a konačni sloj je sposoban prepoznati kompleksne uzorke poput lica, automobila itd. Dimenzije korištenih filtara manjih su prostornih dimenzija od ulaza, ali jednake dubine. Vrijednosti njihovih težina predstavljaju parametre modela koji se optimiziraju tijekom procesa učenja mreže. Dimenzija kvadratne matrice piksela koje filtar obrađuje u svakome koraku naziva se receptivnim poljem (širina i visina filtra) dok je njegova dubina jednaka dubini značajki koje ulaze u sloj. Veličina svakog koraka (engl. stride) predstavlja broj piksela pomaka u horizontalnom i vertikalnom smjeru pri konvoluciji filtara i mape značajki. Upravljanje dimenzijama izlaznih aktivacijskih mapa postiže se primjenom dopunjavanja (engl. padding). Dopunjavanje predstavlja proces umetanja nula oko rubova izvorne mape značajki. Posljedično tome rubni elementi još više pridonose prijenosu informacija zato što se njihova konvolucija s filtrom više puta odvija. Neka je broj filtara u l-tom konvolucijskom sloju N, njihove dimenzije $f^{[l]}$, korak $s^{[l]}$ i veličina dopunjavanja $p^{[l]}$. Ako se na ulaz promatranog konvolucijskog sloja dovede podatkovni volumen iz prethodnog sloja dimenzija $T^{[l-1]}\text{x}H^{[l-1]}\text{x}C^{[l-1]}$, dimenzije izlazne mape značajki $T^{[l]}\text{x}H^{[l]}\text{x}C^{[l]}$ definirane su sljedećim izrazima:

$$T^{[l]} = \left\lfloor \frac{T^{[l-1]} - f^l + 2 \cdot p^{[l]}}{s^{[l]}} \right\rfloor + 1 \tag{3.7}$$

$$H^{[l]} = \left\lfloor \frac{H^{[l-1]} - f^l + 2 \cdot p^{[l]}}{s^{[l]}} \right\rfloor + 1 \tag{3.8}$$

$$C^{[l]} = N. \tag{3.9}$$

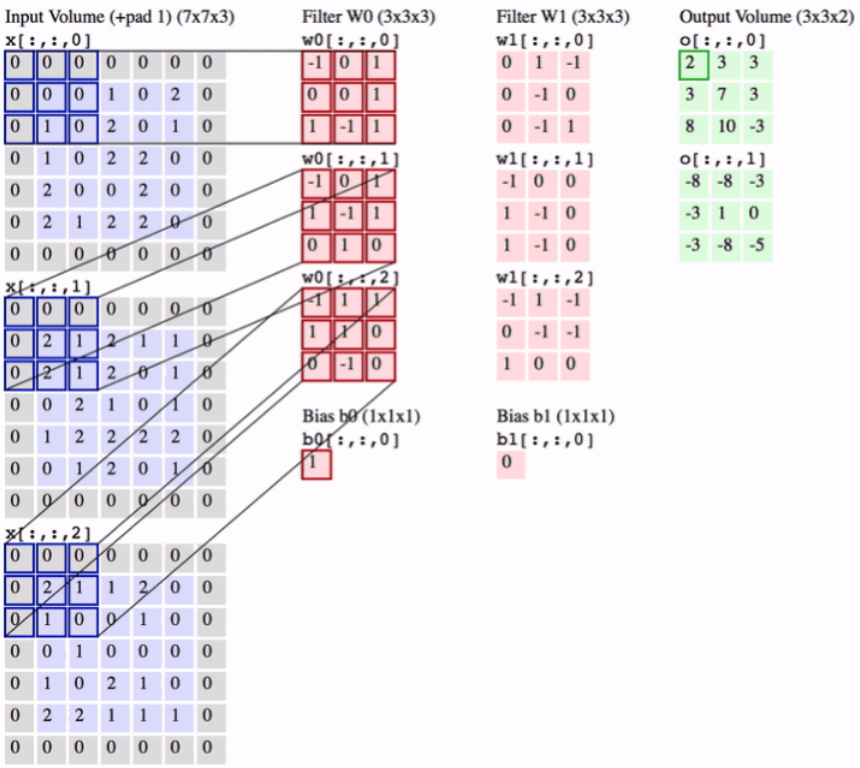
Rezultat djelovanja l-tog konvolucijskog sloja, $O^{[l]}$, na ulaznu mapu mapu značajki $O^{[l-1]}$, definira se sljedećim izrazom:

$$O^{[l]} = \phi^{[l]}\left(Z^{[l]}\right), \tag{3.10}$$

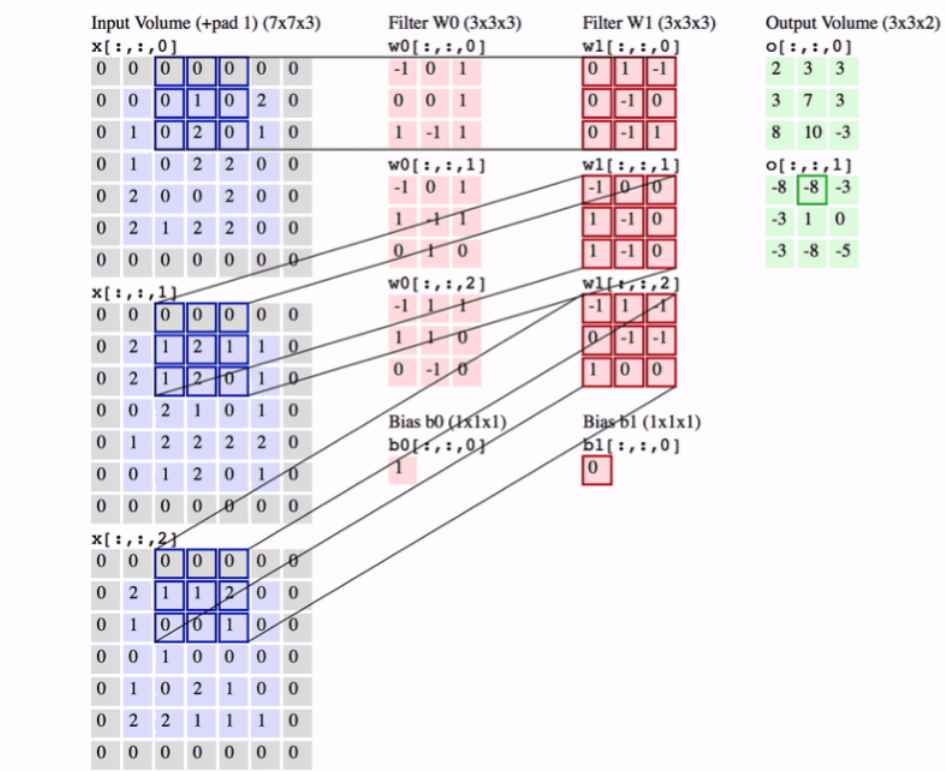
gdje je $\phi^{[l]}$ aktivacijska funkcija, izraz za $Z^{[l]}$ glasi:

$$\begin{gathered} Z^{[l]}_{i,j,c} = \sum_{m=0}^{f^{[l]}-1} \sum_{n=0}^{f^{[l]}-1} \sum_{m=0}^{C^{[l-1]}-1} O^{[l-1]}_{i+m,j+n,k} \cdot T^{[l]}_{c,m,n,k} + b^{[l]}_{i,j,c}, \\ i = 0, ..., H^{[l]} - 1, \quad j = 0, ..., T^{[l]} - 1, \quad c = 0, ..., N - 1. \end{gathered} \tag{3.11}$$

$W^{[l]}$ predstavlja vrijednosti težinskih faktora, a $b^{[l]}$ je veličina pomaka. Slike 3.10 i 3.11 prikazuju dva uzastopna koraka prethodno opisane konvolucije. Na slikama je ulazna mapa značajki dimenzija 7x7x3 i koriste se dva filtara veličine 3x3x3. S obzirom na to da nema nadopunjavanja i korak je veličine 2, volumen izlazne mape značajki je 3x3x2. Uobičajeno je koristiti rastući broj konvolucijskih filtara u dubljim slojevima neuronskih mreža, čime se postižu dublje mape značajki, odnosno povećava se sposobnost prepoznavanja kompleksnih uzoraka.



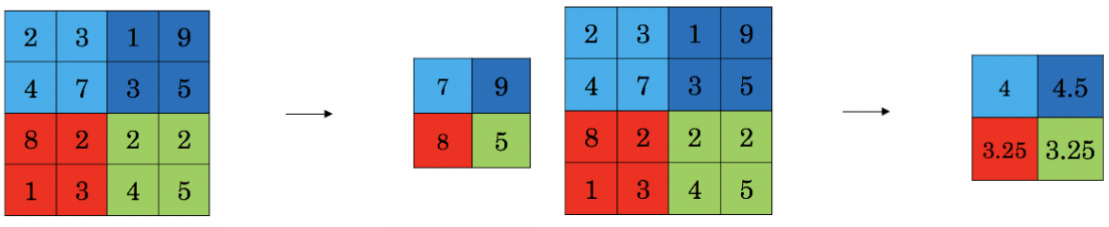
Slika 3.10. Prikaz prvog koraka konvolucije [8].



Slika 3.11. Prikaz idućeg koraka konvolucije [8].

3.4.2. Sloj sažimanja

Nakon nekoliko konvolucijskih slojeva često slijedi sloj sažimanja (engl. pooling layer). Sloj sažimanja smanjuje dimenzije mape značajki iz prethodnog konvolucijskog sloja, odnosno smanjuje rezoluciju mape. Također, sloj sažimanja povećava prostornu invarijantnost (neosjetljivost na male translacije piksela u uzorku). Svrha je zadržavanje samo važnih značajki (vrijednosti piksela) iz svakog područja koje najbolje opisuju kontekst slike. Funkcija sažimanja određena je parametrima receptivnog polja i koraka slično kao i kod konvolucijskog sloja. Najčešće korištene vrste sažimanja su sažimanje usrednjavanjem (engl. average-pooling) i sažimanje izborom maksimalne vrijednosti (engl. max-pooling). Sažimanje usrednjavanjem zamjenjuje piksele u receptivnom polju filtra s aritmetičkom sredinom njihovih iznosa, dok sažimanje izborom maksimalne vrijednosti izlučuje maksimalnu vrijednost piksela unutar receptivnog polja. Važno je napomenuti da se dubina ne mijenja primjenom sloja sažimanja. Slika 3.12(a) prikazuje primjer sažimanja usrednjavanjem koristeći filtar dimenzije 2x2 koraka 2, a (b) prikazuje primjer sažimanja izborom maksimalne vrijednosti s jednakim parametrima.

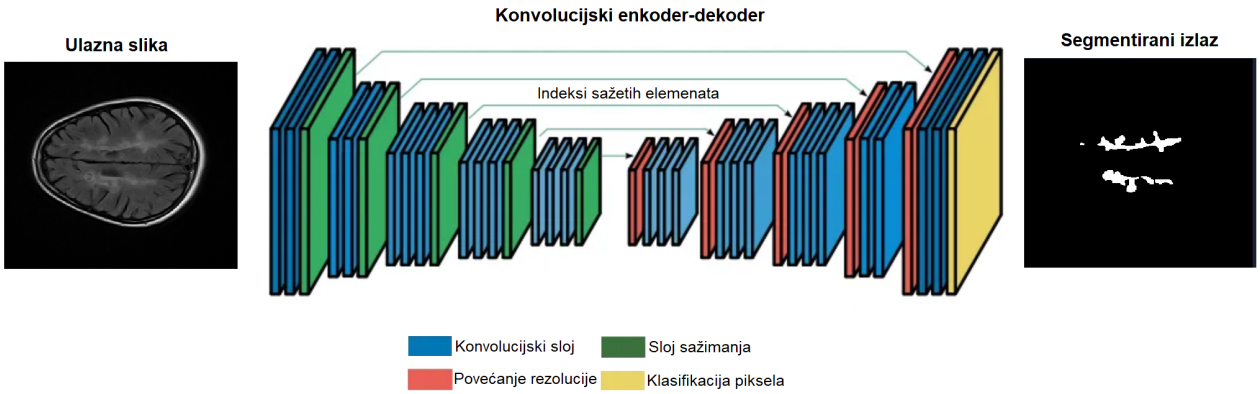


Slika 3.12. (a) Sažimanje usrednjavanjem, (b) Sažimanje izborom maksimalne vrijednosti [9].

Uočljivo je da i konvolucijski sloj i sloj sažimanja smanjuju širinu i visinu ulazne slike, međutim postepenom primjenom sve više filtara u dubljim slojevima povećava se dubina što omogućuje prepoznavanje sve kompleksnijih značajki slike.

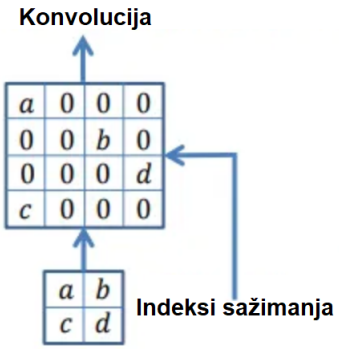
3.5. SegNet arhitektura

Konvolucijska mreža vrlo je korisna kada je bitno identificirati određene značajke na slici. Međutim, konačni cilj ovog rada je semantička segmentacija slike koja ne zahtijeva samo dodjeljivanje klasa već je potrebno odrediti i njihove apsolutne položaje. Odnosno, željeni izlaz je slika visoke rezolucije u kojoj su svi pikseli klasificirani. Stoga je potreban mehanizam pretvaranja slike niske rezolucije u sliku visoke rezolucije kako bi se povratila informacija o točnim pozicijama promatranih značajki. U ovom radu se analiziraju tri različite arhitekture neuronskih mreža za segmentaciju, pri čemu se SegNet ističe kao najjednostavnija u smislu broja parametara koje sadrži i koje je potrebno optimizirati tijekom procesa učenja mreže. SegNet se sastoji od mreže enkodera i njemu odgovarajuće mreže dekodera, nakon čega slijedi konačni sloj za klasifikaciju svakog piksela. Enkoder služi za shvaćanje konteksta slike (izvlačenje značajki), a dekoder se koristi za prezicnu lokalizaciju klasificiranih objekata. Enkoder sadrži 13 konvolucijskih slojeva koji odgovaraju prvim slojevima VGG16 arhitekture dizajnirane za klasifikaciju objekata [8]. Sukladno broju konvolucijskih slojeva enkodera, dekoder također sadrži 13 slojeva. Slika 3.13 prikazuje SegNet strukturu. Plavom bojom su označeni konvolucijski slojevi, a zelenom slojevi sažimanja. Crvena boja predstavlja slojeve za povećanje rezolucije dok žuta označava posljednji sloj za klasifikaciju piksela.



Slika 3.13. SegNet arhitektura neuronske mreže [10].

Kao u prethodno spomenutim konvolucijskim slojevima, u svakom se sloju vrši konvolucija s odabranim brojem filtara. Međutim, prije upotrebe aktivacijske funkcije (ReLU) nad generiranom mapom značajki, vrši se normalizacija podataka (engl. batch normalization - BN). BN normalizira prethodno generirane značajke tako da imaju srednju vrijednost blizu nule i varijancu blizu jedinice, a zatim primjenjuje skaliranje i pomak kako bi se održala fleksibilnost modela. Korištenje BN-a može pomoću u rješavanju problema nestabilnosti treniranja dubokih neuronskih mreža, kao što su problemi s nestajućim ili eksplodirajućim gradijentima. Nakon uzastopne primjene par opisanih konvolucijskih slojeva, slijedi sloj sažimanja izborom maksimalne vrijednosti dimenzija filtra 2x2 i veličine koraka 2. Izlazne vrijednosti sloja sažimanja dodatno se smanjuju dijeljenjem s 2 (poduzorkovanje). Ovaj postupak smanjivanja dimenzionalnosti omogućuje neuronskoj mreži procesiranje i pripremanje najvažnijih značajki za sljedeće slojeve. Tijekom procesa smanjivanja rezolucije, za svaku mapu značajki enkodera se u svakom koraku sažimanja pohranjuju pozicije najvećih vrijednosti (indeksi sažimanja). Na ovaj način se, uz korištenje malo radne memorije, omogućuje povrat prostorne rezolucije u dekoderu, a time i točne lokacije klasificiranih objekata na slici. Korištenjem metoda za povećanje rezolucije poput bilinearne ili kubične interpolacije, uz pohranjene indekse sažimanja, dekoder povećava rezoluciju kao što je prikazano na slici 3.14, što rezultira oskudnim mapama značajki.

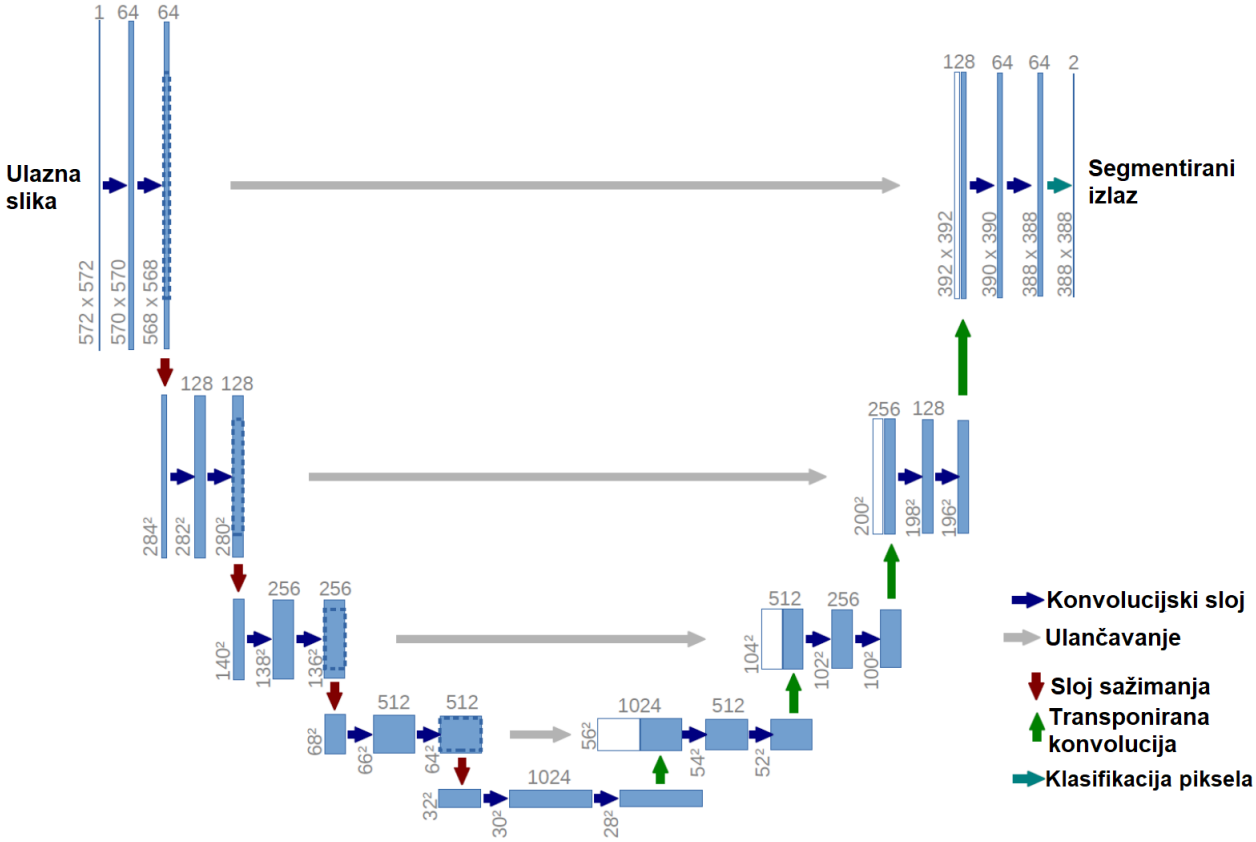


Slika 3.14. Povećavanje rezolucije.

Na dobivene mape značajki se zatim primjenjuje konvolucija kako bi one postale popunjenije. Konvolucijski filtri dekodera sadrže vrijednosti elemenata koji se, kao i u enkoderu, optimiraju tijekom procesa učenja neuronske mreže. Svaki konvolucijski sloj dekodera također primjenjuje BN prije aktivacijskih funkcija. Posljednji sloj dekodera klasificira svaki piksel korištenjem konvolucije s filtrom dimenzije 1 i sigmoidalnu aktivacijsku funkciju.

3.6. U-net arhitektura

Prethodno prikazna konvolucija kojom se smanjuju ulazne dimenzije može se opisati kao množenje matrica ulazne slike s filtrom. Transponirana konvolucija je obrnuti proces koji omogućuje povećanje prostorne rezolucije izlazne slike, tako da se koristi matrica filtra koja je prethodno transponirana. Kombiniranje duboke konvolucijske mreže sa slojevima transponirane konvolucije dobiva se U-net arhitektura neuronske mreže. U-net je motiviran potrebom za segmentacijom medicinskih slika te se također sastoji od dva dijela, enkodera i dekodera [11]. Slično kao i kod SegNet-a, enkoder služi za ekstrakciju značajki iz slike, te se sastoji od više konvolucijskih slojeva i slojeva sažimanja. Mape značajke izlučene enkoderom se zatim koriste u dekoderu koji se sastoji od više konvolucijskih slojeva i slojeva transponirane konvolucije. Izlazni sloj dekodera pridružuje klase svakom pikselu primjenom konvolucije s filtrom dimenzije 1 i sigmoidalnom aktivacijskom funkcijom. U-net mreža ima simetričnu strukturu, pri čemu enkoder i dekoder sadrže isti broj konvolucijskih slojeva, odnosno slojeva sažimanja i slojeva transponirane konvolucije. Kako bi se postigla još preciznija lokalizacija objekata na slici, U-net mreža koristi proces ulančavanja u kojem se izlazi iz sloja transponirane konvolucije u dekoderu povezuju s izlazima iste razine u enkoderu. Tako se omogućava da se značajke izlučene u enkoderu koriste u dekoderu kako bi se postigla veća preciznost segmentacije. Ovaj proces omogućava modelu da nauči značajke na različitim razinama i stoga može preciznije segmentirati slike. Slika 3.15 prikazuje primjer U-net neuronske mreže. Plavi pravokutnici predstavljaju mape značajki. Dimenzije širina i visina prikazane su uz lijevi rub pravokutnika dok su dubine (broj komponenti) prikazane na vrhovima. Plave strelice ukazuju na primjenu konvolucijskih slojeva, a crvene sažimanje. Zelenim strelicama su označene primjene transponirane konvolucije, te sivim smjer ulančavanja po odgovarajućim razinama enkodera i dekodera.



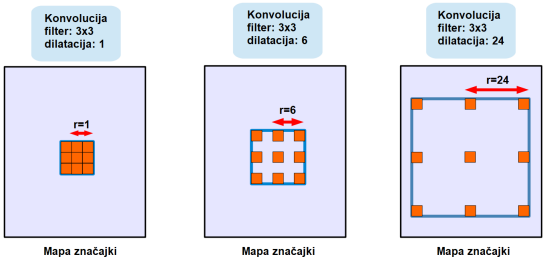
Slika 3.15. Primjer U-net arhitekture neuronske mreže [12].

3.7. DeepLabV3+ arhitektura

DeepLabV3+ je najnovija verzija modela neuronske mreže kojeg je razvio Google u svrhu semantičke segmentaciju slika [14]. U osnovi, funkcionira na sličan način kao SegNet i U-Net, koristeći duboku konvolucijsku neuronsku mrežu za ekstrakciju značajki iz ulazne slike, nakon čega se provodi dekodiranje značajki kako bi se izvršila segmentacija. Jedna od glavnih razlika u odnosu na prethodno opisane modele je u tome što koristi metodu dilatirajućih ili atroznih konvolucija (engl. atrous convolution) kako bi se povećalo receptivno polje filtara uz manji gubitak prostorne rezolucije mapi značajki. Na ovaj način, model je u stanju dobiti preciznije informacije o kontekstu i položaju objekata na slici, što omogućuje bolju segmentaciju. Atrozna konvolucija može se opisati sljedećim izrazom:

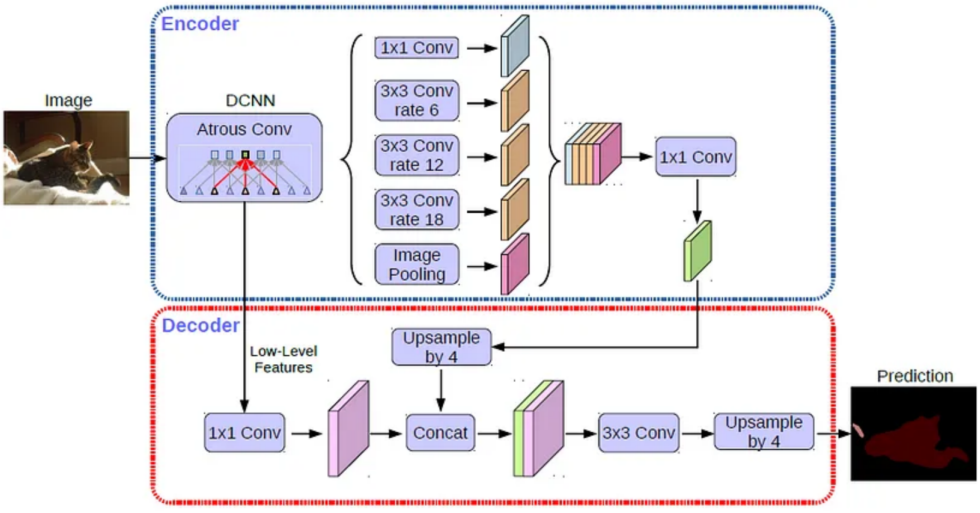
$$y\left[i\right] = \sum_{k} x\left[i + r \cdot k\right] w\left[k\right], \tag{3.12}$$

gdje y i x predstavljaju izlazne i ulazne vrijednosti elemenata dvodimenzionalne mape značajki na poziciji i. S w se označava filtar dok r predstavlja dilatacijski korak konvolucije, odnosno razmak između elemenata matrice izvornog filtra. Za $r = 1$, izraz poprima oblik klasične konvolucije. Slika 3.16 prikazuje primjer atrozne konvolucije s filtrom koji sadrži 3x3 vrijednosti i različite korake dilatacije.



Slika 3.16. Primjer atrozne konvolucije [14].

Enkoder DeepLabV3+ također se sastoji od niza konvolucijskih slojeva i slojeva sažimanja. Nakon što se izluče lokalne značajke iz slike, umjesto daljnjeg smanjivanja rezolucije, koristi se atrozna konvolucija. Ova metoda uključuje paralelnu primjenu konvolucija s različitim koracima dilatacije na mapu značajki, čime se ublažava gubitak rezolucije. Dilatacijski korak u konvoluciji utječe na veličinu mape značajki koja se generira. Korištenjem paralelnih konvolucija s različitim koracima dilatacije, enkoder DeepLabV3+ može generirati mape značajki različitih veličina. Ovime se postiže kontrola nad rezolucijom značajki koje se koriste u dekoderu, što omogućuje prilagođavanje preciznosti i brzine računanja, a da se pritom ne gubi važna informacija o slikama. Mape značajki iz enkodera se zatim bilinearno interpoliraju, odnosno proširuju za faktor 4 i zatim se ulančavaju s odgovarajućim lokalnim značajkama iz ranijih slojeva duboke konvolucijske neuronske mreže. Prija ulančavanja, na lokalne značajke se primjenjuje 1x1 konvolucija kako bi se smanjio njihov broj kanala. Nakon ulančavanja primjenjuju se konvolucijski slojevi i još jedno proširivanje. Opisana struktura prikazana je na slici 3.17.



Slika 3.17. DeepLabV3+ arhitektura [14].

4. Implementacija algoritama

U ovom dijelu rada implementiraju se algoritmi za segmentaciju lezija na mozgu korištenjem prethodno opisanih arhitektura umjetnih neuronskih mreža. Implementacija se provodi u Python programskom jeziku, a najvažniji dijelovi detaljno se opisuju u nastavku, dok je cjelokupan kod priložen na kraju rada u dodatcima A i B. Za pokretanje razvijenih algoritama koriste se Kaggle serveri koji široj javnosti omogućuju upotrebu moćnih grafičkih kartica čime se vrijeme treniranja modela značajno smanjuje (NVIDIA® Tesla® P100 GPU, Intel® Xeon® E5-2630 2.30GHz CPU, 13GB RAM). Od mnoštva implementiranih programskih knjižnica, potrebno je spomenuti najvažnije. $Keras$ programska knjižnica, koji je dio $TensorFlow$ sustava, razvijena je kao jednostavno i intuitivno programsko sučelje za duboko učenje i izgradnju složenih neuronskih mreža. Podržava različite vrste slojeva, aktivacijskih funkcija, optimizacijskih algoritama i metrika. Također, koristi se $NumPy$ programski paket koji služi za znanstveno računanje i obradu podataka. Odnosno, pruža efikasne alate za rad s nizovima podataka, što uključuje brojne matematičke operacije kao što su matrične operacije, statističke funkcije i generiranje slučajnih brojeva. Opisane neuronske mreže posebno se treniraju za T1, T2 i FLAIR skupove podataka kako bi se uočila najpogodnija vrsta snimanja za automatsku segmentaciju. Konačno, mreža se trenira korištenjem sveukupnog seta podataka i svih mogućih kombinacija navedenih podskupina. Za evaluaciju rezultata koristi se DCS (engl. Dice coefficient score) koji se definira sljedećim izrazom:

$$DCS = \frac{2 \cdot |A \cap B|}{|A| + |B|}, \tag{4.1}$$

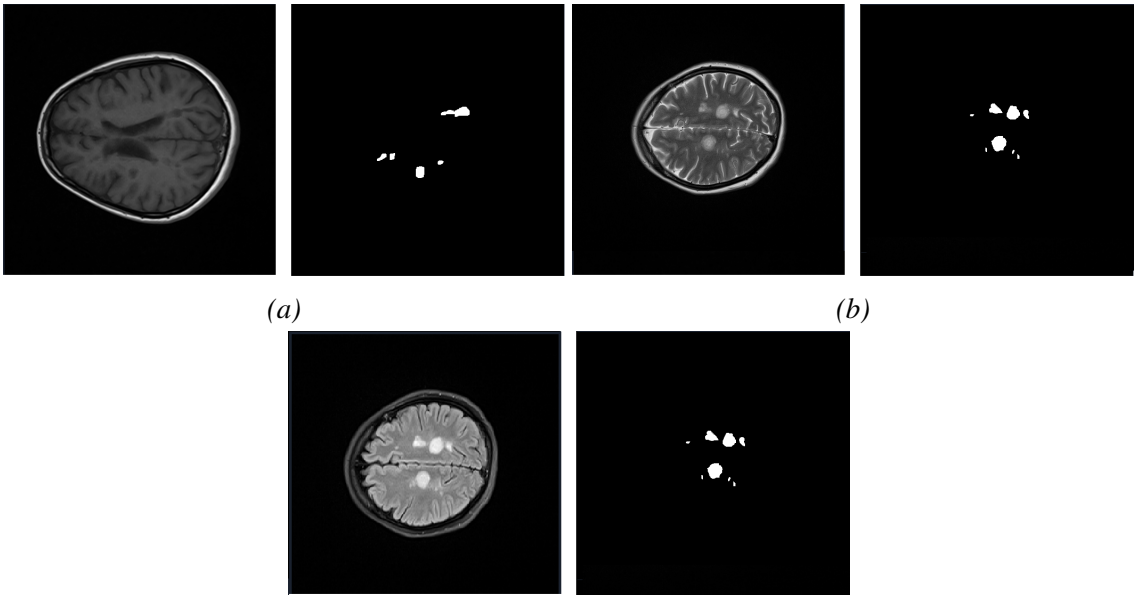
gdje A predstavlja izvornu (istinitu) segmentiranu masku a B predviđenu. S obzirom na to da proces segmentacije pridružuje vrijednosti 0 ili 1 pikselima ovisno o tome nalazi se lezija na promatranoj poziciji, lakše je razumjeti sljedeći izraz za DCS:

$$DCS = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}. \tag{4.2}$$

TP (engl. true positive) predstavlja ukupan broj određenih piksela koji imaju vrijednost 1 i u A i u B. FP (engl. false positive) označava ukupan broj specifičnih piksela koji su 1 u B, ali 0 u A dok FN podrazumijeva ukupan broj piksela koji su 0 u B, a 1 u A. DCS je kriterij koji se koristi za mjerenje preciznosti u segmentaciji medicinskih slika. Uspoređuje preklapanja između segmentiranih (predviđenih) i stvarnih maski (označenih lezija) na piksel razini, pri čemu se čak i najmanja odstupanja kažnjavaju. Stoga se DCS smatra pouzdanim kriterijem za procjenu uspješnosti segmentacije, jer je važno točno identificirati lezije koje se često sastoje od samo nekoliko piksela. U konačnici, što je veći DCS, to je veća preciznost u segmentaciji slike.

4.1. Obrada skupa podataka

Korišteni skup podataka sastoji se od MR snimki mozga (T1-izmjerene, T2-izmjerene i FLAIR) 60 pacijenata oboljelih od multiple skleroze. Skup podataka je preuzet iz izvora [15]. Osim izvornih snimki, skup podataka sadrži i odgovarajuće maske (izlučene lezije) koje su segmentirali i potvrdili različiti neurolozi i radiolozi. S obzirom na to da su MR snimke i njihove maske $NIfTI$ (engl. Neuroimaging Informatics Technology Initiative) formata, prvo ih je potrebno pretvoriti u png format slike pomoću definirane funkcije $nii2png$ preuzete iz [16]. Pozivanjem navedene funkcije, korisniku se pruža odabir dva direktorija u kojima se pohranjuju transformirane MR slike, odnosno njihove pripadajuće maske u png formatu. Svakoj slici pridružuje se odgovarajuća maska s istim nazivom, ali s dodatkom sufiksa "LesionSeg". Kako bi se osigurala usklađenost nomenklature maski i njihovih izvornih slika za daljnje operacije, raspisuje se jednostavna for petlja koja uklanja sufikse "LesionSeg" iz naziva maski. Rezultirajući skup podataka sastoji se od 1358 T1, 1380 T2 i 1451 FLAIR slika, uz isti broj pripadajućih maski. Rezolucije slika variraju od 128x128 do 512x512 piksela. Primjer izlučenog dijela T1 snimke i odgovarajuće segmentirane maske prikazan je na slici 4.1(a), dok je primjer T2 snimke prikazan na slici 4.1(b), a primjer FLAIR snimke na slici 4.1(c).



Slika 4.1. Prikaz izvorne slike i odgovarajućih maski za (a) T1, (b) T2, (c) FLAIR način snimanja.

Dobivene png slike potrebno je transformirati u pogodan oblik za implementaciju u neuronskim modelima. Najprije se definiraju liste imena svih slika i maski u prikladnim direktorijima, ids i $ids2$, te se ispisom provjerava je li njihov broj jednak. Zatim se određuju željene dimenzije (rezolucija) slika, pri čemu se odabire visina i širina od 128 piksela. Nadalje, izrađuju se prazna (ispunjena nulama) 4D polja (engl. array) X i y, koja se naknadno popunjavaju vrijednostima piksela učitanih slika i maski. Sadrže $float32$ tip vrijednosti i dimenzija su $nx128x128x1$. Prva dimen-

zija polja predstavlja broj (indeks) slike, druga širinu, treća visinu, a četvrta dubinu (obzirom da su slike crno-bijele, dubina je jednaka 1). Potom se raspisuje for petlja koja iterativno učitava sve slike i maske iz zadanih direktorija te ih pretvara u 4D polja korištenjem img_to_array ugrađene funkcije iz $Keras$ knjižnice, a zatim ih pomoću $resize$ funkcije iz $skimage$ paketa preoblikuje u odgovarajuće dimenzije. Definirana petlja također vrši normalizaciju svih vrijednosti piksela na vrijednosti između 0 i 1 primjenjujući odgovarajuću formulu:

$$\frac{Vrijednosti - Minimum}{Raspon}. \tag{4.3}$$

Zbog eventualnog dijeljenja s nulom, rezultirajuća polja maski i slika sadrže par NaN vrijednosti (engl. not a number) koje se transformiraju u nulu pomoću $NumPy$ knjižnice. Opisani programski kod prikazan je u nastavku:

```
ids = next(os.walk("T1_images"))[2]
print("Broj slika = ", len(ids))
ids2 = next(os.walk("T1_masks"))[2]
print("Broj maski = ", len(ids2))

img_width = 128
img_height = 128

X = np.zeros((len(ids),img_height,img_width,1),dtype=np.float32)
y = np.zeros((len(ids),img_height,img_width,1),dtype=np.float32)

for n, id_ in tqdm(enumerate(ids), total=len(ids)):
        #Slike
        img = load_img("T1_images/"+id_, color_mode = "grayscale")
        X_img = img_to_array(img)
        X_img = resize(X_img, (img_height, img_width, 1),
                       mode = 'constant', preserve_range = True)
        X_img_max = X_img.max(axis=(0, 1))
        X_img_min = X_img.min(axis=(0, 1))
        XRange = X_img_max - X_img_min
        #Maske
        maska = load_img("T1_masks/"+id_, color_mode = "grayscale")
        maska = img_to_array(maska)
        maska = resize(maska, (img_height, img_width, 1),
                       mode = 'constant', preserve_range = True)
        maska_max = maska.max(axis=(0,1))
        maska_min = maska.min(axis=(0,1))
        maskaRange = maska_max - maska_min
```

```
        #Normalizacija
        X[n] = (X_img-X_img_min)/XRange
        y[n] = (maska-maska_min)/maskaRange

X = np.nan_to_num(X)
y = np.nan_to_num(y)
```

Nadalje, dobiveni podaci dijele se na skupove za treniranje, validaciju i testiranje pomoću ugrađene funkcije $train\_test\_split$ iz $sklearn$ knjižnice u omjeru 80%-10%-10%. Skup za treniranje i validaciju se koriste tijekom procesa strojnog učenja modela, pri čemu validacijski skup predstavlja podatke koje model "ne vidi", odnosno služi za evaluaciju segmentacije nepoznatih slika. Za dodatnu objektivnost pri evaluaciji rezultata, definira se i skup za testiranje koji uopće ne sudjeluje u procesu treniranja. Konačni broj korištenih podataka pokazan je u tablici 4.1.

Tablica 4.1. Prikaz broja korištenih podataka.

	Treniranje	Validacija	Testiranje
T1	1086	136	136
T2	1104	138	138
FLAIR	1161	145	145
Ukupno	3351	419	419
T1+T2	2190	274	274
T1+FLAIR	2247	281	281
T2+FLAIR	2265	283	283

4.2. SegNet

U ovom dijelu koda izrađuje se prethodno opisana SegNet arhitektura. Glavni sastavni dio neuronske mreže predstavljaju konvolucijski slojevi koji se uzastopno primjenjuju na ulaznim podacima kako bi se izvukle značajke. Svaki konvolucijski sloj u ovom modelu sastoji se od tri glavne operacije: konvolucije, normalizacije podataka i aktivacijske funkcije. Radi lakše preglednosti i efikasnijeg koda, ove tri operacije su definirane kroz funkcije koje se koriste za konstrukciju višestrukih slojeva unutar modela. Na primjer, za dvostruki sloj se definira $Conv2D\_Blok2$ funkcija. Ova funkcija definira blok s dva uzastopna sloja primjene spomenutih operacija. Konvolucijski sloj kao ulazne parametre koristi veličinu matrice filtera korištenih za konvoluciju, broj korištenih filtera i vrstu nasumične inicijalizacije težinskih faktora (vrijednosti elemenata filtera). Sloj normalizacije je opcionalan i kontrolira se ulaznom varijablom $batchnorm$. Na kraju se primjenjuje aktivacijska funkcija ReLU, te funkcija vraća izlazni tenzor. Opisani blok se ostvaruje na sljedeći način:

```
def Conv2D_Blok2(tensor, n_filters, k_size, batchnorm=True):
    #Prvi sloj
```

```
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(tensor)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    #Drugi sloj
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

return x
```

Idući kod definira funkciju $segnet$, koja stvara željenu konvolucijsku neuronsku mrežu. Funkcija prima 5 parametra: oblik ulaza, broj filtara koji će se koristiti u svakom sloju konvolucije, dimenzija matrice filtara, opcionalni parametar za kontrolu normalizacije, te vrijednost izostavljanja (engl. dropout) koja predstavlja postotak neurona koji se isključuje tijekom treniranja radi poboljšanja generalizacije modela. Sukladno izloženoj arhitekturi prikazanoj na slici 3.13, enkoder se sastoji od 5 konvolucijskih blokova, a nakon svakoga slijedi sloj sažimanja izborom maksimalne vrijednosti. Nakon svakog sloja sažimanja primjenjuje se opisana dropout funkcija za regularizaciju mreže. Sažimanje se ostvaruje korištenjem funkcije $MaxPoolingWithArgmax2D$, koja ne vraća samo maksimalne vrijednosti matrice već i indekse tog maksimuma. Ti indeksi se kasnije koriste u slojevima dekodera kako bi se dobila originalna dimenzionalnost slike. Prvi konvolucijski blok primjenjuje zadani broj filtara, a svaki sljedeći sloj ih sadrži dvostruko više. Prva 2 konvolucijska bloka predstavljaju dvostruke konvolucijske slojeve, a posljednja 3 trostruke. Nakon što je enkoder izgrađen, ispisuje se "Enkoder je izgrađen" kako bi korisnik mogao vidjeti da je funkcija uspješno stvorila mrežu, odnosno kako bi se moglo lakše dijagnosticirati u kojem dijelu funkcije je došlo do eventualne pogreške. Dio opisane funkcije koji definira enkoder ostvaruje se kao:

```
def segnet(ulaz_oblik,n_filters,dropout,k_size,batchnorm = True):
    #Enkoder
        ulaz = Input(shape=ulaz_oblik)

        c_1 = Conv2D_Blok2(ulaz, n_filters * 1, k_size = k_size,
                           batchnorm = batchnorm)
        p_1, mask_1 = MaxPoolingWithArgmax2D((2,2))(c_1)
        p_1 = Dropout(dropout)(p_1)
```

```
c_2 = Conv2D_Blok2(p_1, n_filters * 2, k_size = k_size,
                   batchnorm = batchnorm)
p_2, mask_2 = MaxPoolingWithArgmax2D((2,2))(c_2)
p_2 = Dropout(dropout)(p_2)

c_3 = Conv2D_Blok3(p_2, n_filters * 4, k_size = k_size,
                   batchnorm = batchnorm)
p_3, mask_3 = MaxPoolingWithArgmax2D((2,2))(c_3)
p_3 = Dropout(dropout)(p_3)

c_4 = Conv2D_Blok3(p_3, n_filters * 8, k_size = k_size,
                   batchnorm = batchnorm)
p_4, mask_4 = MaxPoolingWithArgmax2D((2,2))(c_4)
p_4 = Dropout(dropout)(p_4)

c_5 = Conv2D_Blok3(p_4, n_filters * 8, k_size = k_size,
                   batchnorm = batchnorm)
p_5, mask_5 = MaxPoolingWithArgmax2D((2,2))(c_5)
p_5 = Dropout(dropout)(p_5)
print("Enkoder je izgraden")
```

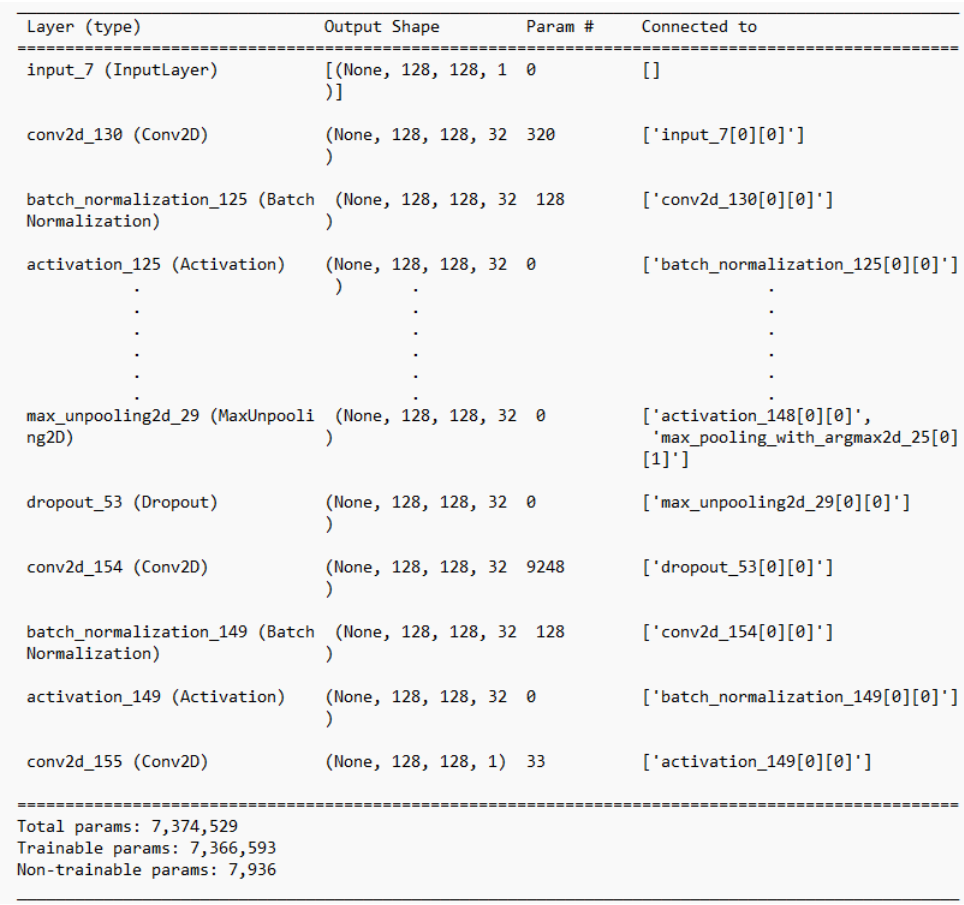
U nastavku je definirana arhitektura dekodera, koja je simetrična enkoderu, s izuzetkom što se umjesto slojeva sažimanja koristi sloj $MaxUnpooling2D$ između konvolucijskih slojeva kako bi se povećala rezolucija pomoću interpolacije i zapamćenih indeksa maksimalnih vrijednosti iz prethodne faze svakog stupnja enkodera. Također, broj filtara se smanjuje u svakom sljedećem bloku kako bi se održala simetričnost. Posljednji sloj dekodera primjenjuje konvoluciju veličine filtra 1 i sigmoidalnu aktivacijsku funkciju kako bi se klasificirali pojedinačni pikseli. Nakon definiranja svih slojeva i povezivanja ulaza i izlaza, stvara se objekt pod nazivom "model", klase $keras.Model$, te se vraća kreirani model neuronske mreže. U komentaru se dodatno ispisuje informacija da je dekoder izgrađen. Nastavak definicije funkcije $segnet$, odnosno opisana struktura dekodera ostvarena je sljedećim kodom:

```
up_1 = tfa.layers.MaxUnpooling2D()(p_5, mask_5)

c_6 = Conv2D_Blok3(up_1, n_filters * 8, k_size = k_size,
batchnorm = batchnorm)
up_2 = tfa.layers.MaxUnpooling2D()(c_6, mask_4)
up_2 = Dropout(dropout)(up_2)

c_7 = Conv2D_Blok2(up_2, n_filters * 8, k_size = k_size,
batchnorm = batchnorm)
```

```
c_8 = Conv2D_Blok(c_7, n_filters * 4, k_size = k_size,
batchnorm = batchnorm)
up_3 = tfa.layers.MaxUnpooling2D()(c_8, mask_3)
up_3 = Dropout(dropout)(up_3)

c_9 = Conv2D_Blok2(up_3, n_filters * 4, k_size = k_size,
batchnorm = batchnorm)
c_10 = Conv2D_Blok(c_9, n_filters * 2, k_size = k_size,
batchnorm = batchnorm)
up_4 = tfa.layers.MaxUnpooling2D()(c_10, mask_2)
up_4 = Dropout(dropout)(up_4)

c_11 = Conv2D_Blok(up_4, n_filters * 2, k_size = k_size,
batchnorm = batchnorm)
c_12 = Conv2D_Blok(c_11, n_filters * 1, k_size = k_size,
batchnorm = batchnorm)
up_5 = tfa.layers.MaxUnpooling2D()(c_12, mask_1)
up_5 = Dropout(dropout)(up_5)

c_13 = Conv2D_Blok(up_5, n_filters * 1, k_size = k_size,
batchnorm = batchnorm)

izlaz = Conv2D(1, (1, 1), activation='sigmoid')(c_13)
print("Dekoder je izgraden")

model = Model(inputs=ulaz, outputs=izlaz, name="SegNet")

return model
```

Prije same izgradnje modela, potrebno je definirati odgovarajuću funkciju gubitka koja se minimizira tijekom procesa strojnog učenja. Najprije se raspisuje formula za DCS pomoću $keras.Flatten$ funkcije. Navedena funkcija transformira ulazne tenzore koji predstavljaju stvarne i predviđene vrijednosti piksela u jednodimenzionalne vektore na kojima se zatim mogu primijeniti operacije presjeka i zbrajanja, sukladno izrazu 4.1. S obzirom na to da je DCS derivabilna funkcija, pomoću nje se može jednostavno definirati funkcija gubitka kao $1 - DCS$. Nadalje, definira se model pomoću prethodno definirane $segnet$ funkcije uz proizvoljno određene hiperparametre poput broja i veličine konvolucijskih filtara. Pozivanjem $keras.Model.compile$ funkcije izgrađuje se definirana neuronska mreža pri čemu se odabire Adam optimizacijski algoritam i opisana funkcija gubitka. Kako bi se provjerila dobivena struktura modela, poziva se funkcija $keras.Model.summary$ čime se prikazuju dimenzije i povezanosti pojedinačnih slojeva mreže. Slika 4.2 prikazuje primjer prvih

i zadnjih par slojeva sažetka dobivenog modela za početni broj filtara 32. Na slici je vidljivo da su ostvarene željene dimenzije ulaza i izlaza te da je broj ukupnih parametara koji se optimiziraju tijekom učenja (uglavnom težinski faktori) veći od 7 milijuna.



Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	[(None, 128, 128, 1)]	0	[]
conv2d_130 (Conv2D)	(None, 128, 128, 32)	320	['input_7[0][0]']
batch_normalization_125 (BatchNormalization)	(None, 128, 128, 32)	128	['conv2d_130[0][0]']
activation_125 (Activation)	(None, 128, 128, 32)	0	['batch_normalization_125[0][0]']
.	.		.
.	.		.
.	.		.
.	.		.
.	.		.
.	.		.
max_unpooling2d_29 (MaxUnpooling2D)	(None, 128, 128, 32)	0	['activation_148[0][0]', 'max_pooling_with_argmax2d_25[0][1]']
dropout_53 (Dropout)	(None, 128, 128, 32)	0	['max_unpooling2d_29[0][0]']
conv2d_154 (Conv2D)	(None, 128, 128, 32)	9248	['dropout_53[0][0]']
batch_normalization_149 (BatchNormalization)	(None, 128, 128, 32)	128	['conv2d_154[0][0]']
activation_149 (Activation)	(None, 128, 128, 32)	0	['batch_normalization_149[0][0]']
conv2d_155 (Conv2D)	(None, 128, 128, 1)	33	['activation_149[0][0]']

Total params: 7,374,529
Trainable params: 7,366,593
Non-trainable params: 7,936

Slika 4.2. Sažetak dobivene arhitekture.

4.2.1. Rezultati

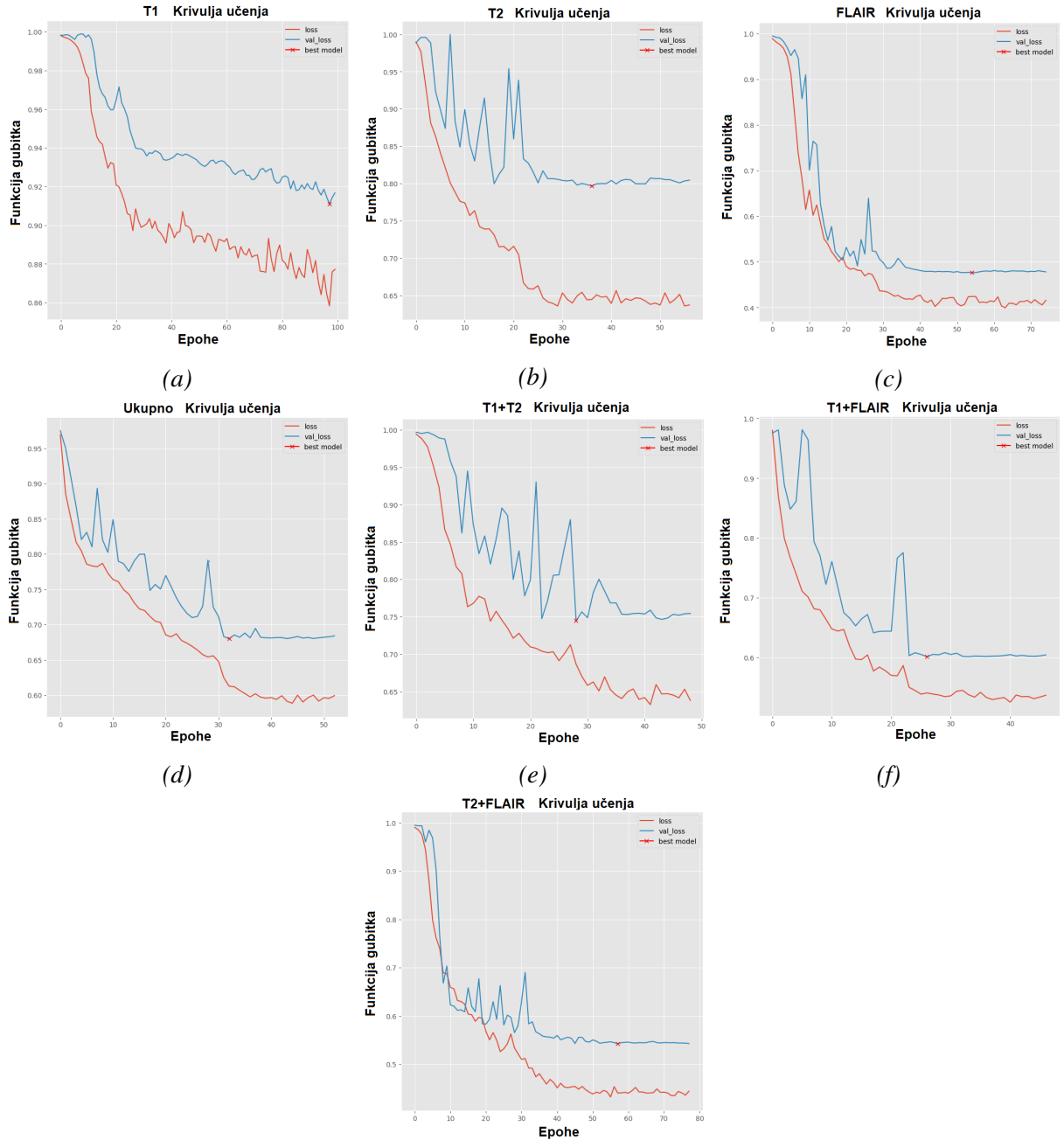
Izgrađena SegNet arhitektura se trenira pomoću $keras.Model.fit$ funkcije pri čemu se određuje trajanje od 100 epoha. Prije početka treninga, potrebno je odrediti nekoliko važnih hiperparametara, uključujući broj filtara u početnom sloju, veličinu skupa slika koje se obrađuju u jednoj iteraciji (engl. batch size), postotak težinskih faktora koji se odbacuju (engl. dropout rate) te stopa učenja. Stopa učenja (engl. learning rate) je koeficijent koji određuje korak promjene težina u svakoj iteraciji tijekom optimizacijskog postupka učenja neuronske mreže. Kada se mreža trenira, gradijenti se izračunavaju za svaku težinu. Ti gradijenti se zatim pomnože stopom učenja kako bi se odredio korak promjene težina u smjeru smanjenja gubitka. Ako je stopa učenja prevelika, algoritam optimizacije može propustiti globalni minimum i završiti u lokalnom minimumu, ili može oscilirati oko optimuma. S druge strane, ako je stopa učenja premalena, optimizacija će biti sporija i može se dogoditi da zaglavi u lokalnom minimumu. Stoga se tijekom treniranja primjenjuje funkcija $keras.callbacks.ReduceLROnPlateau$, koja smanjuje početnu stopu učenja (0.001) u

slučaju da se rezultati nisu poboljšali u prethodnih 5 epoha. Preostali parametri se variraju dok se ne postignu zadovoljavajući rezultati, ali tako da je ukupni broj parametara približno jednak u sve tri arhitekture koje se promatraju u ovom radu, kako bi se njihove performanse mogle adekvatno usporediti. Također, definira se kriterij ranog zaustavljanja treniranja u slučaju da nije došlo do pozitivne promjene rezultata u zadnjih 20 epoha. Osim vrijednosti DCS-a na testnom skupu podataka, model se evaluira pomoću ukupnog vremena trajanja treninga i trajanja jedne epohe. Trajanje jedne epohe ukazuje na koliko brzo promatrani model može učiti, dok ukupno trajanje pokazuje koliko dobro model uči. Iako model može imati kratko trajanje jedne epohe, ne znači da rezultati dovoljno brzo konvergiraju. U boljim modelima kriterij prijevremenog zaustavljanja treninga se aktivira ranije. Najbolji rezultati ostvareni su za FLAIR skup podataka u iznosu od 52.35% preciznosti po DCS, što je i očekivano s obzirom na to da je to vrsta snimke na kojoj se lezije najviše razlikuju od okolnog tkiva. Najniža preciznost od 8.89% postignuta je za T1 skup podataka, a za ukupni skup je postignuta točnost od 32.02%. Važno je istaknuti da model treniran na ukupnom skupu podataka ostvaruje bolje rezultate u segmentaciji T1 (10%) i T2 (30%) podataka u usporedbi s modelima treniranim samo na tim skupovima, dok za FLAIR skup podataka ostvaruje lošije rezultate (47%) u odnosu na model treniran samo na tom skupu. Tablica 4.2 prikazuje sažetak rezultata za sve podskupine podataka.

Tablica 4.2. Prikaz rezultata SegNet arhitekture.

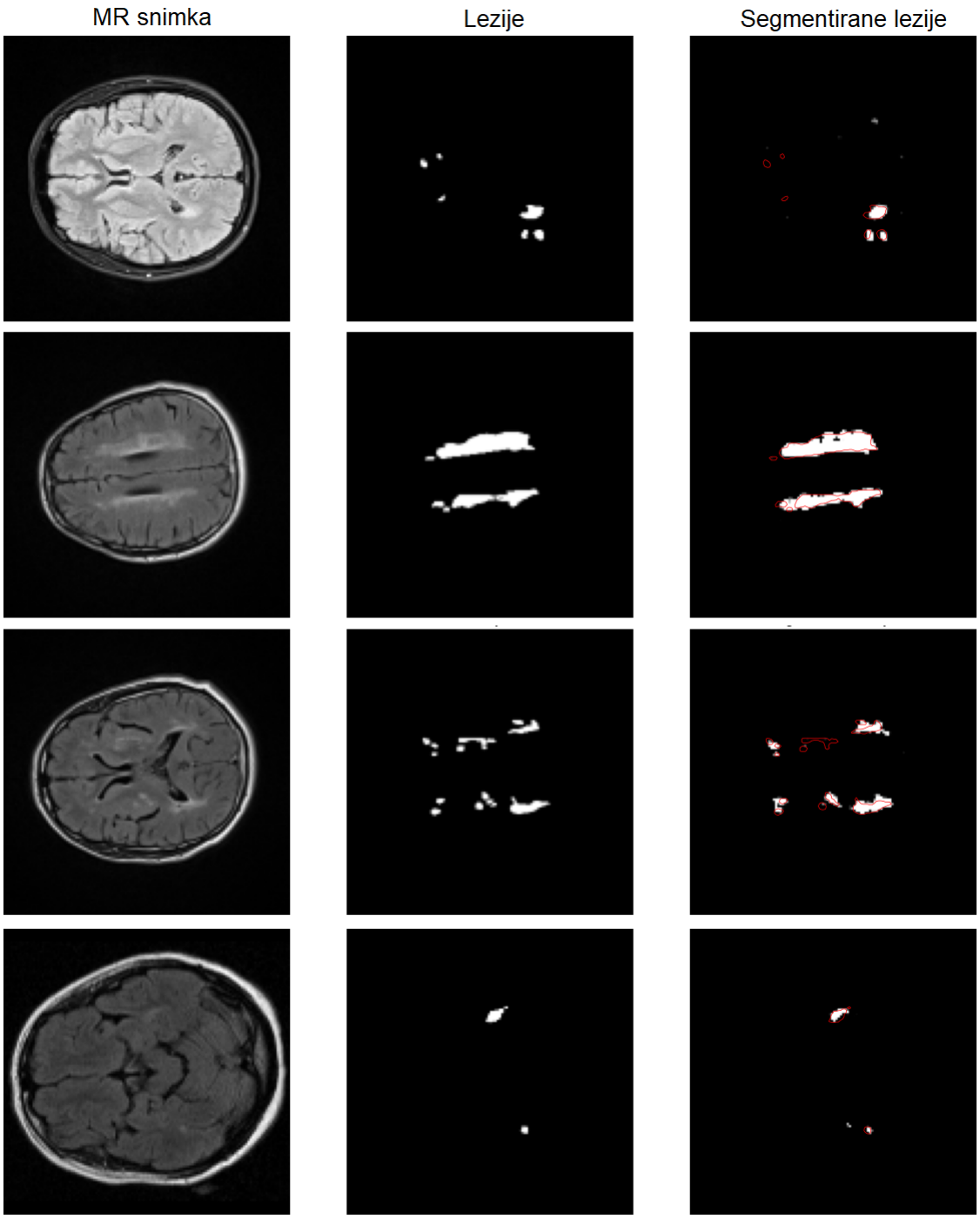
	DSC	Trajanje epohe [s]	Trajanje treninga [min:s]
T1	8.89%	6	8:05
T2	20.34%	5	9:56
FLAIR	52.35%	6	5:36
Ukupno	32.02%	17	11:53
T1+T2	24.54%	11	9:32
T1+FLAIR	39.85%	12	9:49
T2+FLAIR	43.75%	12	15:25

Slike 4.3(a) do (g) prikazuju krivulje učenja za sve skupove podataka, pri čemu se plavom bojom označava funkcija gubitka za validacijski skup, a crvenom za trenirajući skup.



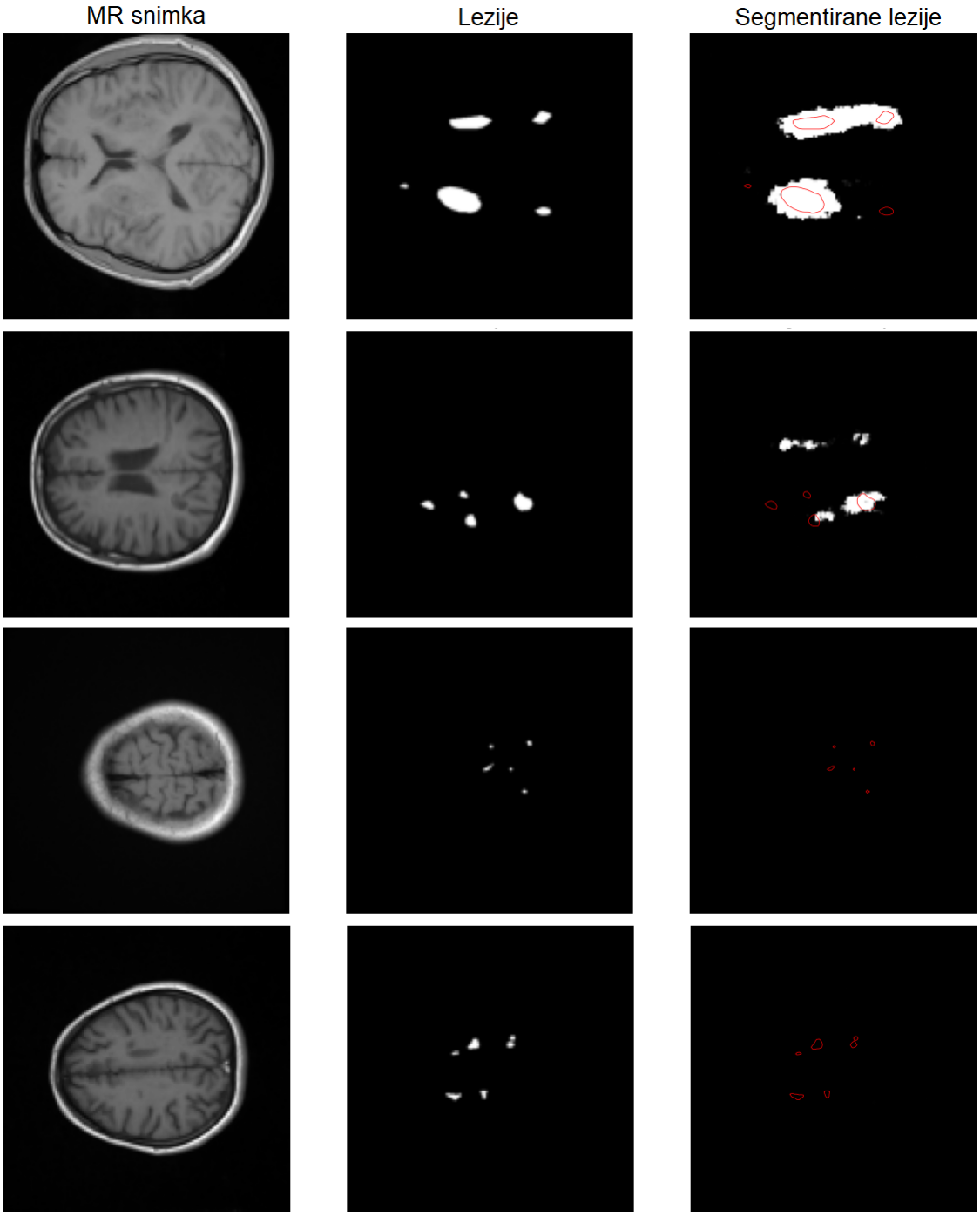
Slika 4.3. Krivulje učenja za (a) T1, (b) T2, (c) FLAIR, (d) Ukupni, (e) T1 i T2, (f) T1 i FLAIR, (g) T2 i FLAIR skupove podataka.

Iz prethodnih slika vidljivo je da u slučaju svih podataka, model preciznije segmentira poznate od nepoznatih slika. To ukazuje na problem generalizacije odnosno nedovoljnog uklapanja. Također, vidljivo je da krivulje najbrže konvergiraju prema konačnoj vrijednosti za FLAIR i ukupni skup podataka. Slika 4.4 prikazuje primjere automatske segmentacije ostvarene na FLAIR skupu podataka. Na slici se s lijeva na desno nalaze MR snimke, istinske lezije i segmentirane lezije, a crvenom konturom su radi lakše usporedbe označeni oblici pravih lezije.



Slika 4.4. Primjer izvršenih segmentacija lezija na FLAIR vrsti snimaka.

Iako model uspješno izdvaja veće lezije, primjećuje se da njihov oblik često nije savršen. S druge strane, segmentacija manjih lezija se obično ne izvodi s visokom preciznošću, ali model uspješno ukazuje na većinu njihovih lokacija. Na slici 4.5 su prikazani primjeri segmentacije dobivene za T1 skup podataka. Uočljivo je da model potpuno neuspješno segmentira manje lezije, dok veće lezije mreža prepoznaje veoma loše.



Slika 4.5. Primjer izvršenih segmentacija lezija na T1 vrsti snimaka.

4.3. U-net

U nastavku se opisuje izgradnja U-net arhitekture pomoću funkcije *unet*, koja također prima pet ulaznih parametara (ulaz, broj filtara, dimenzije filtara, dropout i kontrolu normalizacije). Slično kao i u SegNet arhitekturi, mrežu tvore simetrični enkoder i dekoder. Enkoder se koristi za izvlačenje značajki i sastoji se od niza uzastopnih konvolucijskih slojeva uz normalizaciju podataka i primjenu aktivacijske funkcije ReLU. Nakon toga, slijede slojevi za sažimanje pomoću

funkcije $MaxPooling2D$ i dropout slojevi za smanjenje prenaučenosti mreže. Prema slici 3.15, enkoder se sastoji od 4 razine dvostrukih $Conv2d\_Blok2$ konvolucijskih slojeva, nakon kojih slijedi sažimanje. Broj filtara se udvostručuje na svakoj sljedećoj razini, a posljednji sloj enkodera ne izvodi sažimanje. Izlaz posljednjeg sloja enkodera direktno se propagira u dekoder. Dekoder se sastoji od 4 razine slojeva za povećavanje rezolucije korištenjem transponirane konvolucije, koja se implementira pomoću $Conv2DTranspose$ funkcije. Broj korištenih filtara postupno se smanjuje kako bi se održala simetričnost cjelokupne strukture. Izlazi iz slojeva transponirane konvolucije se ulančavaju s izlazima odgovarajućih slojeva klasične konvolucije kako bi se poboljšala preciznost segmentacije. To se postiže pomoću naredbe $concatenate$. Posljednji sloj dekodera primjenjuje jednodimenzionalnu konvoluciju sa sigmoidalnom aktivacijskom funkcijom kako bi se klasificirao svaki piksel. Opisana funkcija $unet$ koja izgrađuje istoimenu arhitekturu prikazana je u nastavku:

```
def unet(ulaz_oblik, n_filters, dropout, k_size, batchnorm = True):
    #Enkoder
        ulaz = Input(shape=ulaz_oblik)

        c_1 = Conv2D_Blok2(ulaz, n_filters * 1, k_size = k_size,
                           batchnorm = batchnorm)
        p_1 = MaxPooling2D((2, 2))(c_1)
        p_1 = Dropout(dropout)(p_1)

        c_2 = Conv2D_Blok2(p_1, n_filters * 2, k_size = k_size,
                           batchnorm = batchnorm)
        p_2 = MaxPooling2D((2, 2))(c_2)
        p_2 = Dropout(dropout)(p_2)

        c_3 = Conv2D_Blok2(p_2, n_filters * 4, k_size = k_size,
                           batchnorm = batchnorm)
        p_3 = MaxPooling2D((2, 2))(c_3)
        p_3 = Dropout(dropout)(p_3)

        c_4 = Conv2D_Blok2(p_3, n_filters * 8, k_size = k_size,
                           batchnorm = batchnorm)
        p_4 = MaxPooling2D((2, 2))(c_4)
        p_4 = Dropout(dropout)(p_4)

        c_5 = Conv2D_Blok2(p_4, n_filters = n_filters * 16,k_size=k_size,
                           batchnorm = batchnorm)
        print("Enkoder je izgraden")
        #Dekoder
        u_6 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2),
```

```
                    padding = 'same')(c_5)
u_6 = concatenate([u_6, c_4])
u_6 = Dropout(dropout)(u_6)
c_6 = Conv2D_Blok2(u_6, n_filters * 8, k_size = k_size,
                   batchnorm = batchnorm)

u_7 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2),
                          padding = 'same')(c_6)
u_7 = concatenate([u_7, c_3])
u_7 = Dropout(dropout)(u_7)
c_7 = Conv2D_Blok2(u_7, n_filters * 4, k_size = k_size,
                   batchnorm = batchnorm)

u_8 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2),
                          padding = 'same')(c_7)
u_8 = concatenate([u_8, c_2])
u_8 = Dropout(dropout)(u_8)
c_8 = Conv2D_Blok2(u_8, n_filters * 2, k_size = k_size,
                   batchnorm = batchnorm)

u_9 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2),
                          padding = 'same')(c_8)
u_9 = concatenate([u_9, c_1])
u_9 = Dropout(dropout)(u_9)
c_9 = Conv2D_Blok2(u_9, n_filters * 1, k_size = k_size,
                   batchnorm = batchnorm)

izlaz = Conv2D(1, (1, 1), activation='sigmoid')(c_9)
print("Dekoder je izgraden")
model = Model(inputs=ulaz, outputs=izlaz, name="Unet")

return model
```

Iako je U-net arhitektura slična SegNet-u, ona ima značajne razlike u implementaciji. U-net koristi slojeve za povećavanje rezolucije pomoću transponirane konvolucije, što dovodi do poboljšanja preciznosti segmentacije, ali i do većeg broja težinskih faktora za treniranje. Također, funkcija za sažimanje $MaxPooling2D$ ne pohranjuje nikakve vrijednosti, što dovodi do manjeg broja težinskih faktora i smanjenja potencijalne prenaučenosti mreže. Za početni broj filtara 32, konačna struktura sadrži 8 milijuna parametara za treniranje.

4.3.1. Rezultati

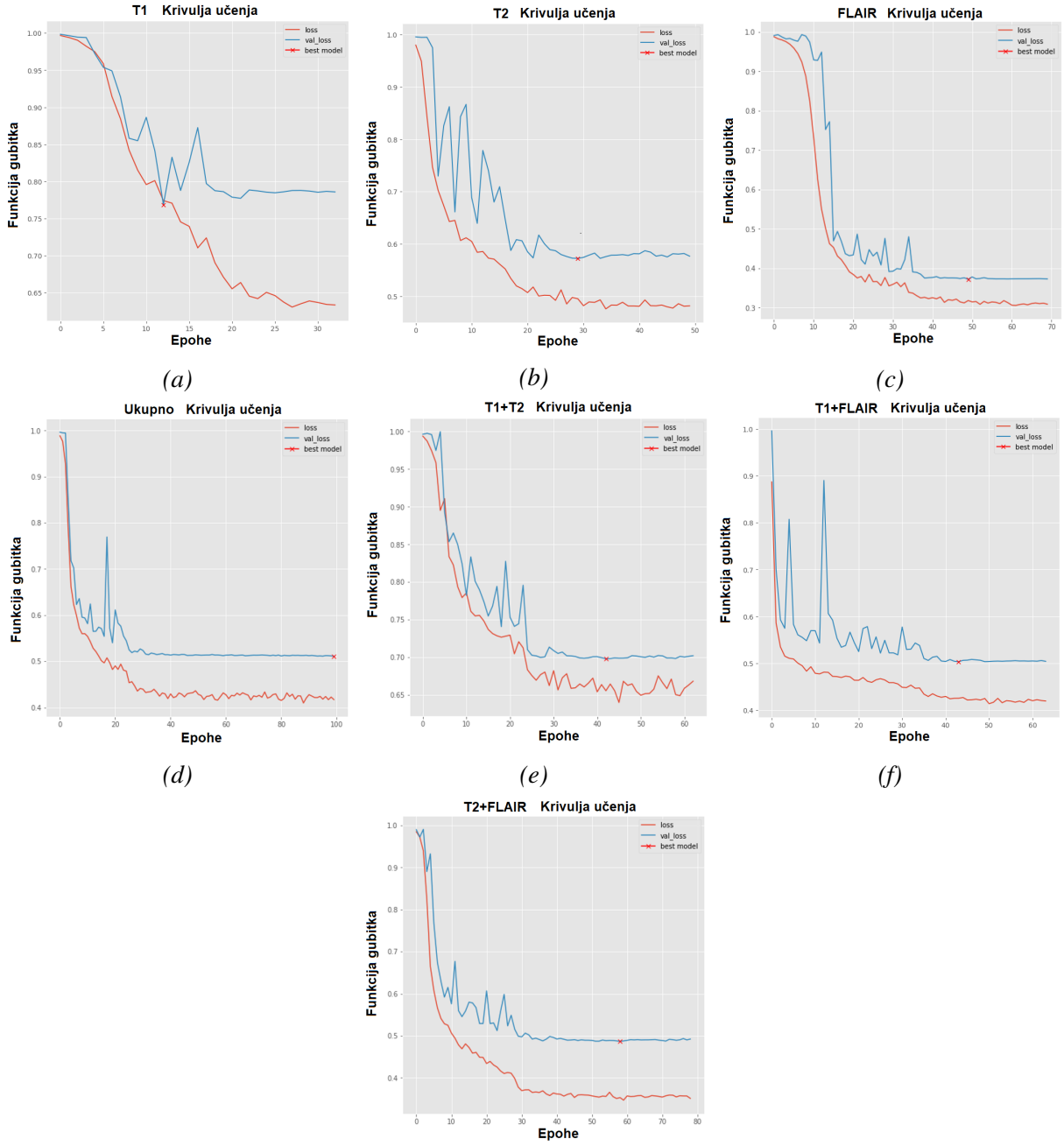
Unet arhitektura se također trenira pomoću $keras.Model.fit$ funkcije s trajanjem od 100 epoha. Prije treninga, određuju se hiperparametri, uključujući broj filtara u početnom sloju, veličinu skupa slika u jednoj iteraciji, postotak odbacivanja težinskih faktora i stopu učenja. Stopa učenja se također smanjuje tijekom treninga ako se rezultati nisu poboljšali u prethodnih 5 epoha korištenjem funkcije $keras.callbacks.ReduceLROnPlateau$. Kriterij ranog zaustavljanja se aktivira nakon 20 epoha bez pozitivne promjene rezultata. Kao i kod SegNet arhitekture, broj filtara u početnom sloju i veličina skupa slika koje se obrađuju u jednoj iteraciji se variraju dok se ne postignu zadovoljavajući rezultati, uz uvjet da sveukupni broj parametara bude približno jednak onome kod SegNet arhitekture kako bi se performanse obje arhitekture mogle adekvatno usporediti. Evaluacija modela se vrši pomoću DCS-a na testnom skupu podataka, ukupnog vremena trajanja treninga i trajanja jedne epohe, kako bi se procijenila brzina učenja i konvergencija modela. Najviša postignuta preciznost za FLAIR skup podataka iznosi 61.28%, dok je preciznost za sveukupni skup podataka 48.46%, a za T1 skup podataka najmanja, 23.13%. Modeli koji su trenirani na skupovima podataka koji uključuju više vrsta snimki, za T1 i T2, pružaju bolje rezultate u odnosu na modele koji su trenirani na pojedinačnim vrstama. Međutim, za FLAIR podatke najbolju segmentaciju pruža model treniran samo na FLAIR skupu podataka. U tom smislu, kombinacija različitih vrsta snimki može pomoći u poboljšanju performansi modela, ali je potrebno razmotriti svaku vrstu snimki pojedinačno i odabrati optimalni pristup za svaku od njih. Sažetak rezultata za sve podskupine podataka, dobiven pomoću U-net mreže, prikazan je u tablici 4.3.

Tablica 4.3. Prikaz rezultata U-net arhitekture.

	DSC	Trajanje epohe [s]	Trajanje treninga [min:s]
T1	23.13%	3	1:35
T2	42.77%	3	2:26
FLAIR	61.28%	3	3:27
Ukupno	48.46%	8	13:23
T1+T2	30.04%	6	6:17
T1+FLAIR	49.65%	6	6:22
T2+FLAIR	52.52%	6	7:54

Usporedbom rezultata prikazanih u tablici 4.3 s onima u tablici 4.2, može se zaključiti da se U-net arhitektura bolje pokazala u rješavanju ovog specifičnog zadatka segmentacije lezija mozga u usporedbi s SegNet arhitekturom. U-net je omogućio postizanje 5-20% veće preciznosti, ovisno o vrsti snimaka, a istovremeno je znatno smanjio vrijeme potrebno za trening modela. Na slikama 6.6(a) do (g) prikazane su krivulje učenja za sve podskupove podataka. Funkcija gubitka za validacijski skup prikazana je plavom bojom, dok je funkcija gubitka za trenirajući skup označena crvenom bojom. Kao i kod rezultata ostvarenih sa SegNet arhitekturom, uočljivo je da modeli imaju poteškoća s generalizacijom podataka, što znači da postoji problem u tome kako se model nosi s novim i neviđenim podacima. Ovo se vidi po tome da se postiže bolja preciznost na skupu

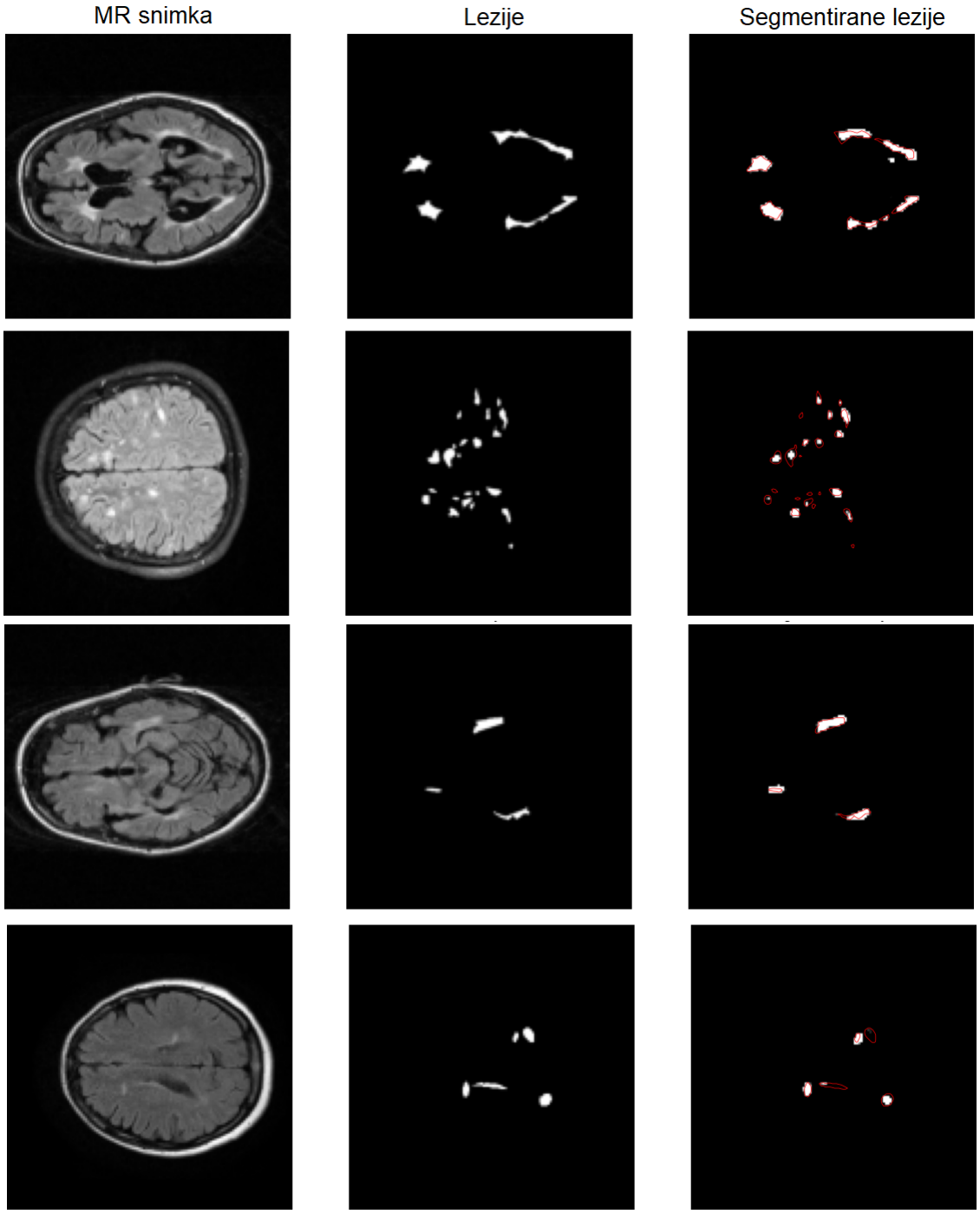
za treniranje u usporedbi s preciznošću na validacijskom skupu.



Slika 4.6. Krivulje učenja za (a) T1, (b) T2, (c) FLAIR, (d) Ukupni, (e) T1 i T2, (f) T1 i FLAIR, (g) T2 i FLAIR skupove podataka.

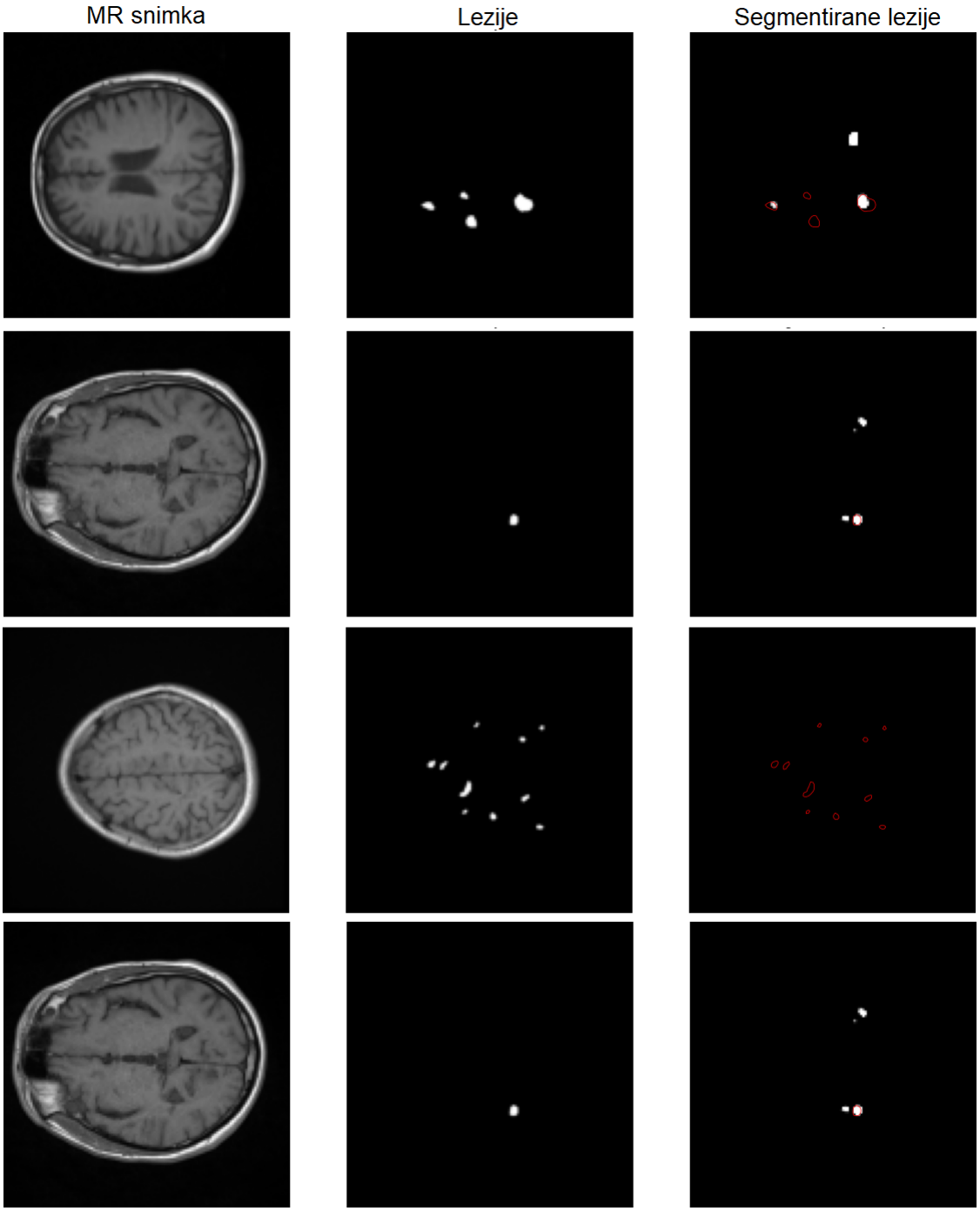
Slika 4.7 prikazuje primjere automatske segmentacije lezija ostvarene na FLAIR skupu podataka pomoću U-net mreže. Na slici su prikazane tri kolone: prva kolona prikazuje MR snimke, druga kolona prikazuje istinske lezije, a treća kolona prikazuje segmentirane lezije. Kako bi se olakšala usporedba, oblici pravih lezije su označene crvenom konturom. Iako je model postigao poboljšanu preciznost u odnosu na prethodni SegNet model, vidljivo je da se i dalje susreće s izazovima u segmentaciji manjih lezija. Ipak, važno je istaknuti da model uglavnom uspješno de-

tektira njihovu približnu lokaciju, dok se veće lezije s obzirom na svoj oblik ne izdvajaju uvijek savršeno.



Slika 4.7. Primjer izvršenih segmentacija lezija na FLAIR vrsti snimaka.

Slika 4.8 prikazuje primjere segmentacije dobivene za T1 skup podataka. Primjećuje se poboljšanje u segmentaciji većih lezija u odnosu na prethodni model, ali su lokacije i oblici lezija i dalje neprecizni. Također, iako se segmentacija manjih lezija poboljšala, još uvijek nije postignuta primjenjiva preciznost.



Slika 4.8. Primjer izvršenih segmentacija lezija na T1 vrsti snimaka.

4.4. DeepLabV3+

Ovaj dio koda prikazuje izgradnju DeepLabV3+ arhitekture pomoću funkcije $DeepLabV3Plus$. Najprije je, sukladno slici 3.17, potrebno definirati funkciju $ASPP$ (engl. atrous spacial pyramid pooling) koja se koristi u DeepLab arhitekturi za izvršavanje atroznih konvolucija. Funkcija kao ulaze prima tenzore iz prijašnjeg sloja mreže i broj filtara koji će se koristiti u konvolucijama. $ASPP$ paralelno primjenjuje 5 operacija na ulazne podatke. Odnosno, predstavlja sloj koji para-

lelno primjenjuje sažimanje odabirom prosječnih vrijednosti i 4 konvolucije koje primjenjuju isti broj filtara, ali s različitim razinama dilatacije. Dilatacija određuje veličinu prostornog razmaka između dva susjedna elementa u filteru. Prvi konvolucijski sloj koristi filtere bez dilatacije, što znači da se koristi standardna konvolucija. Drugi konvolucijski sloj primjenjuje filtere s dilatacijom 6, treći sloj primjenjuje filtere s dilatacijom 12, a četvrti sloj primjenjuje filtere s dilatacijom 18. Nakon paralelne primjene konvolucijskih slojeva i sažimanja na istome signalu, svi se izlazi ulančavaju i propagiraju kroz još jedan konvolucijski sloj s jednim filterom. Izlaz tog sloja se vraća kao izlaz funkcije. Slično kao i prethodne funkcije za izgradnju modela, funkcija $DeepLabV3Plus$ prima parametre za oblik ulaznog tenzora, broj filtara i dropout. Međutim, u skladu s navedenom arhitekturom, veličina filtera je fiksna i iznosi 3. Razlika u arhitekturi ovog modela u odnosu na prethodne arhitekture poput SegNet i U-net je ta što se ne konstruira posebna duboka konvolucijska mreža za enkoder, već se koristi postojeća ResNet50 konvolucijska mreža za klasifikaciju slika koja je unaprijed istrenirana na ImageNet skupu podataka s više od 14 milijuna slika [14]. Iako ovaj skup podataka ne sadrži medicinske slike, korištenje prethodno istrenirane ResNet50 mreže omogućuje modelu da nauči općenite lokalne značajke iz slika, kao što su linije, oblici, teksture, itd. Korištenje prethodno naučenih značajki iz drugog skupa podataka može pomoći u poboljšanju performansi modela i smanjenju potrebnog vremena za treniranje. Izlazne (globalne) značajke iz ResNet50 konvolucijske mreže služe kao ulaz u $ASPP$ funkciju enkodera, dok se njene lokalne karakteristike propagiraju u dekoder. Stoga je potrebno proučiti strukturu ResNet-a pomoću $Model.summary$ naredbe. Shodno tome, uz pomoć dokumentacije [17], može se zaključiti da se izlazni sloj iz mreže označava kao "conv4_block6_out", a sloj koji producira lokalne detalje "conv2_block2_out". Pronađeni slojevi se dohvaćaju korištenjem $Model.get\_layer$ funkcije. Nakon primjene $ASPP$ funkcije na globalnim značajkama, njihova rezolucija se povećava korištenjem bilinearne interpolacije za faktor 4, te se one ulančavaju s prethodno izlučenim lokalnim značajkama koje su konvoluirane s konvolucijom dimenzija 1. Zatim se primjenjuje dvostruki konvolucijski sloj i još jedno uzorkovanje korištenjem bilinearne transformacije čime se mapa značajki vraća na izvornu rezoluciju. Posljednji sloj kao i kod prethodnih arhitektura podrazumijeva primjenu jednodimenzionalne konvolucije sa sigmoidalnom aktivacijskom funkcijom. Opisana funkcija za izgradnju DeepLabV3+ modela priložena je u nastavku:

```
def DeepLabV3Plus(ulaz_oblik, n_filters, dropout):
#Enkoder
ulaz = Input(ulaz_oblik)

#ResNet
base_model = ResNet50(weights='imagenet', include_top=False,
    input_tensor=ulaz)

    #Izlaz iz Resneta u ASPP
    image_features = base_model.get_layer('conv4_block6_out').output
```

```
x_en = ASPP(image_features, n_filters)
x_en = UpSampling2D((4, 4), interpolation="bilinear")(x_en)
#Lokalne znacajke iz Resneta
x_de = base_model.get_layer('conv2_block2_out').output
x_de = Conv2D(filters=8, kernel_size=1, padding='same',
            use_bias=False)(x_de)
x_de = BatchNormalization()(x_de)
x_de = Activation('relu')(x_de)
x_de = Dropout(dropout)(x_de)

x = concatenate([x_en, x_de])

x = Conv2D(filters=n_filters, kernel_size=3, padding='same',
          use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(dropout)(x)

x = Conv2D(filters=n_filters, kernel_size=3, padding='same',
          use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((4, 4), interpolation="bilinear")(x)
x = Dropout(dropout)(x)

izlaz = Conv2D(1, (1, 1), activation='sigmoid')(x)

model = Model(inputs=ulaz, outputs=izlaz)
return model
```

Uz početni broj filtara 32, definirana mrežna struktura za DeeplabV3+ arhitekturu ima oko 9 milijuna parametara za treniranje. To je više nego kod prethodno opisanih arhitektura koje koriste isti početni broj filtara. Također, važno je napomenuti da ResNet50 mreža zahtijeva da ulazne slike budu u boji, što znači da sadrže tri kanala. Stoga, prije korištenja crno-bijelih slika kao ulaza u ovu mrežu, potrebno je proširiti njihov broj kanala na tri, što se može učiniti jednostavnim kopiranjem vrijednosti kanala. To je potrebno napraviti u fazi predobrade podataka prije nego što se slike koriste za treniranje modela.

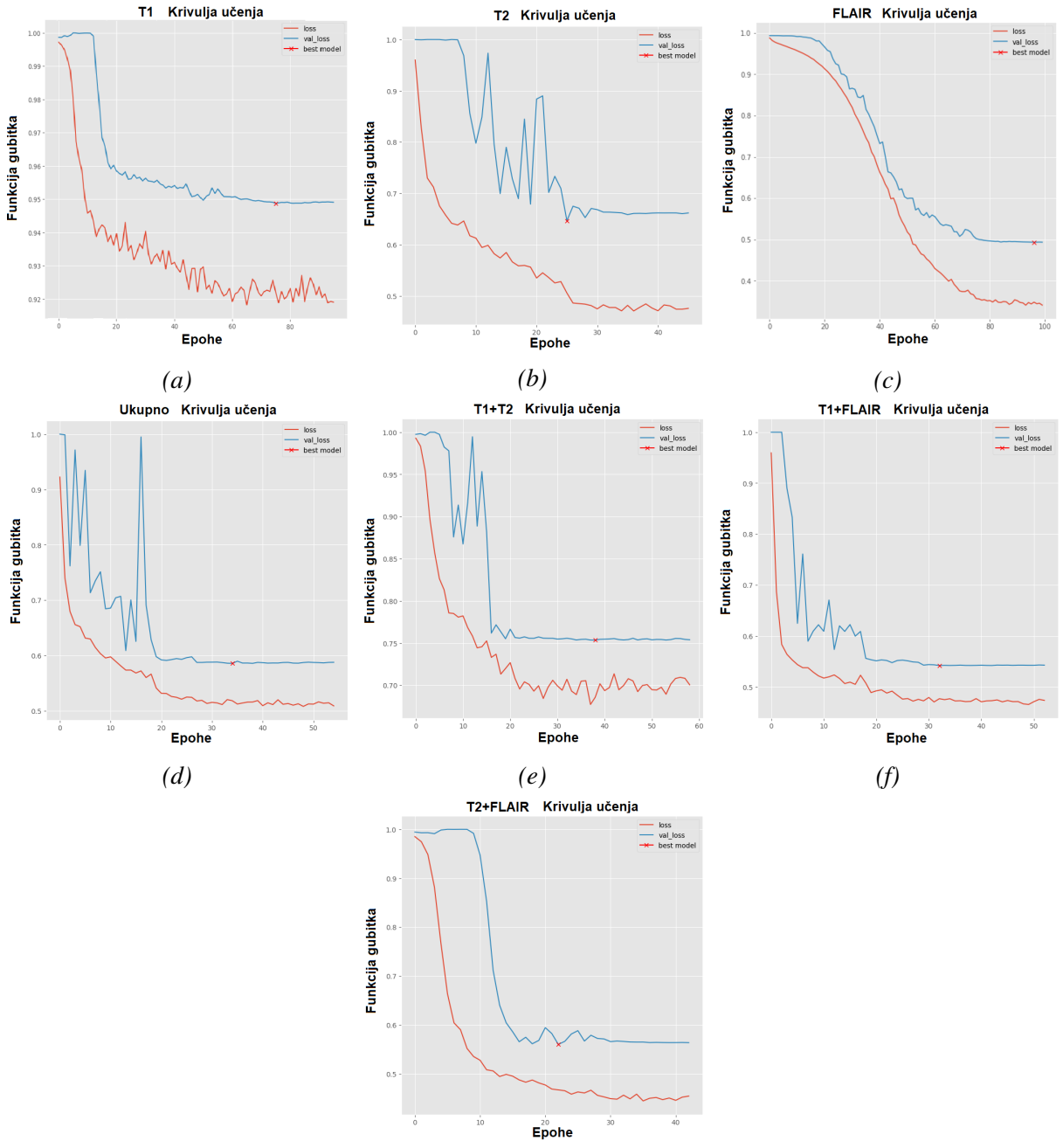
4.4.1. Rezultati

Treniranje DeepLabV3+ arhitekture provodi se na sličan način kao kod prethodnih arhitektura s trajanjem od 100 epoha i kriterijem ranog zaustavljanja nakon 20 epoha. Uz to, definirani su hiperparametri koji su približno jednaki onima kod prethodnih arhitektura, kako bi se osigurala adekvatna usporedba rezultata. Tijekom evaluacije modela koristi se isti skup mjera uspješnosti kao i kod prethodnih arhitektura. Rezultati evaluacije pokazali su da je najveća preciznost ponovno postignuta za FLAIR tip podataka, s vrijednošću od 50.32%, dok je najniža preciznost zabilježena za T1 skup podataka, s vrijednošću od 5.21%. Osim toga, isto kao i kod prethodnih arhitektura, modeli koji su trenirani na skupovima podataka koji uključuju više vrsta snimki, poput T1 i T2, pružaju bolje rezultate u odnosu na modele trenirane na pojedinačnim vrstama. Ipak, za FLAIR podatke, najbolju segmentaciju pruža model koji je treniran samo na FLAIR skupu podataka. Tablica 4.4 prikazuje sve dobivene rezultate za DeepLabV3+ mrežnu strukturu.

Tablica 4.4. Prikaz rezultata DeepLabV3+ arhitekture.

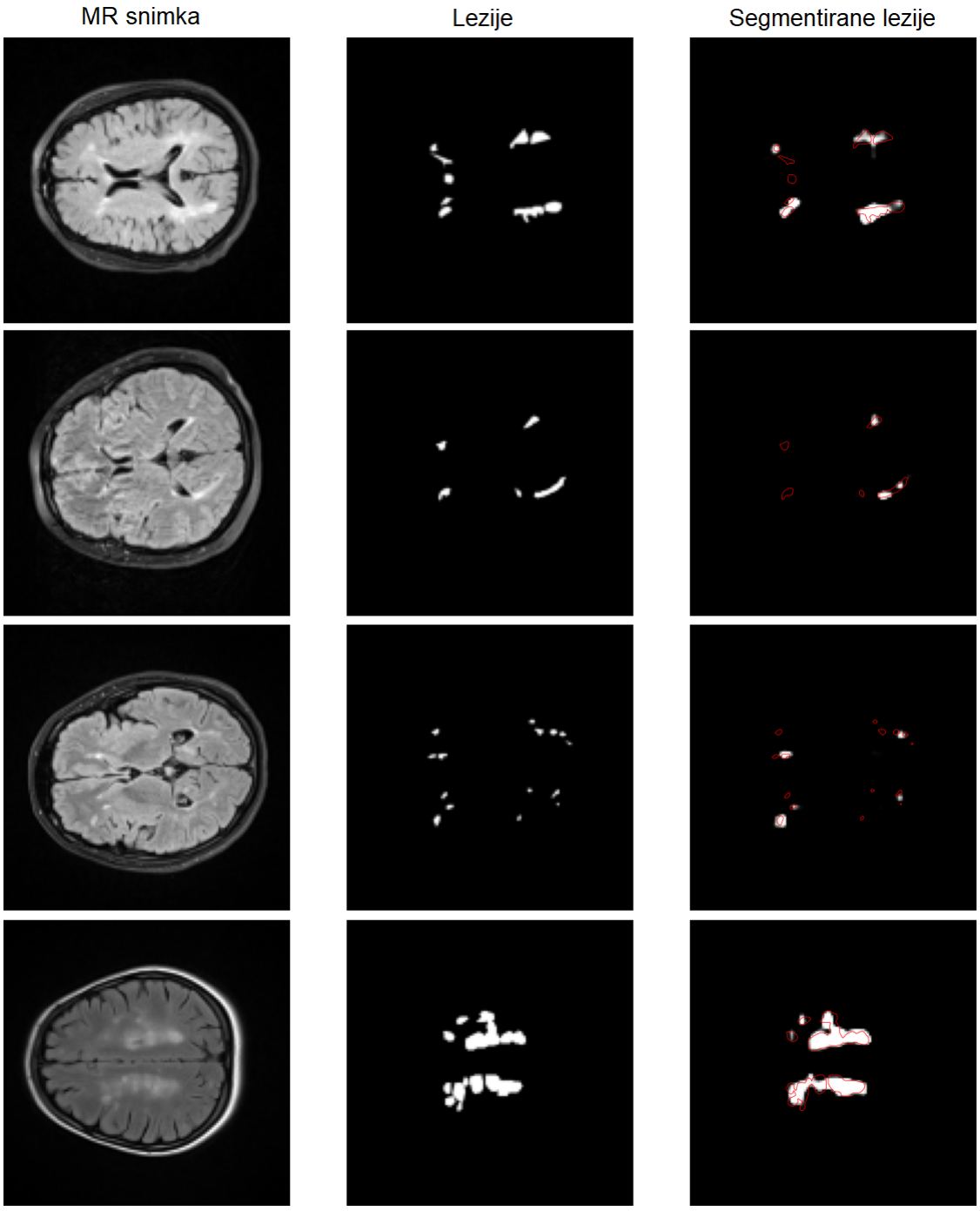
	DSC	Trajanje epohe [s]	Trajanje treninga [min:s]
T1	5.21%	4	5:48
T2	35.34%	4	3:07
FLAIR	50.32%	4	6:28
Ukupno	41.41%	14	12:51
T1+T2	24.64%	8	8:24
T1+FLAIR	42.11%	8	7:12
T2+FLAIR	43.99%	8	5:27

Kada se usporede performanse različitih arhitektura (tablice 4.2, 4.3 i 4.4), primjećuje se da DeepLabV3+ arhitektura pruža slične rezultate kao i SegNet, ali s kraćim vremenima treniranja. Međutim, U-net arhitektura je i dalje najprikladnija za rješavanje problema segmentacije lezija mozga na snimkama magnetske rezonance. Slike 4.9(a) do (g) prikazuju krivulje učenja za sve podskupove podataka, pri čemu je plavom bojom označena funkcija gubitka za validacijski skup, dok je crvenom bojom označena funkcija gubitka za trenirajući skup. Analizom krivulja učenja, još jednom se zaključuje da modeli postižu višu preciznost na skupu za treniranje u odnosu na validacijski skup. Ovako ponašanje modela ukazuje na problem prenaučenosti, odnosno nedovoljnu generalizaciju modela na novim podacima.



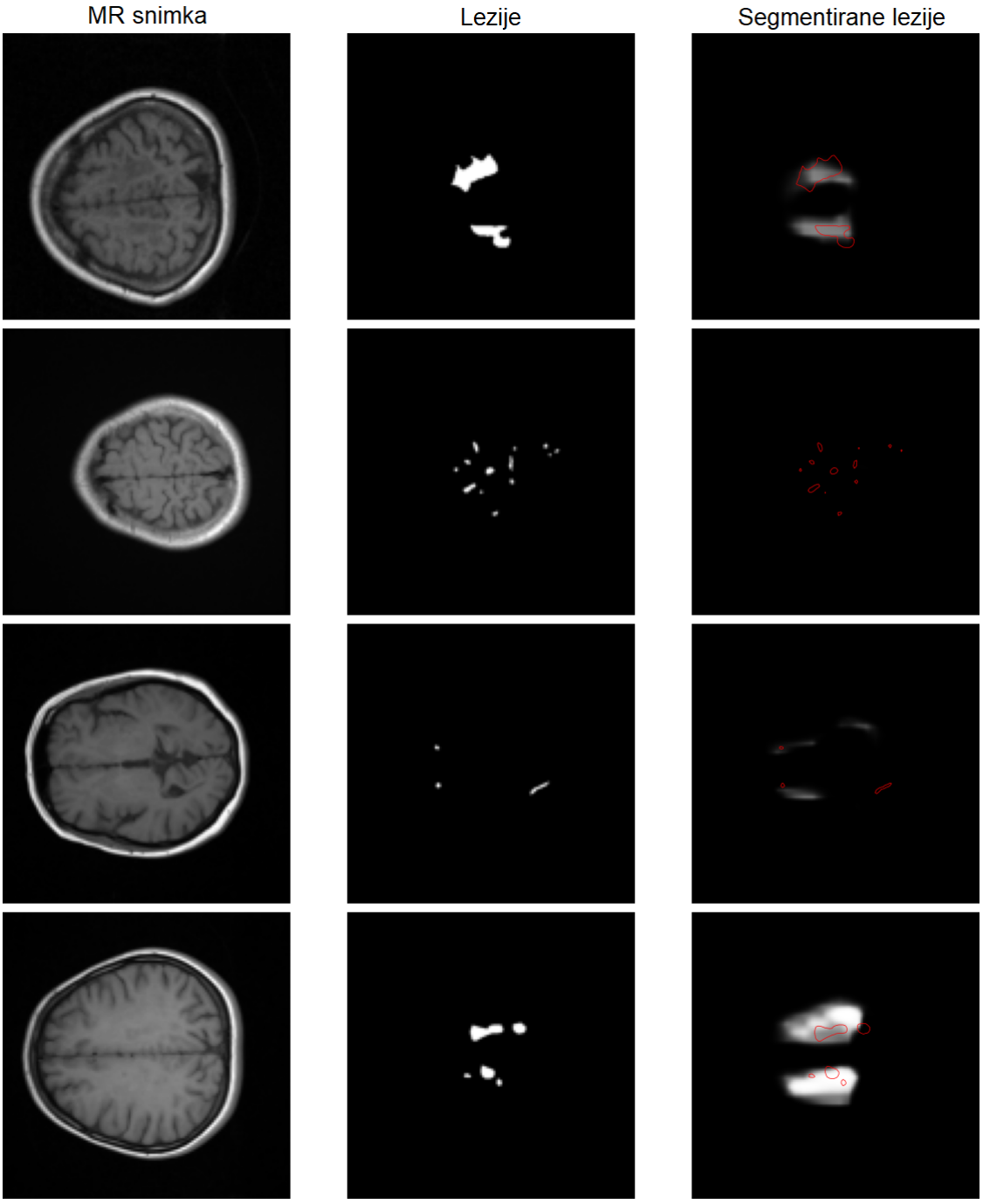
Slika 4.9. Krivulje učenja za (a) T1, (b) T2, (c) FLAIR, (d) Ukupni, (e) T1 i T2, (f) T1 i FLAIR, (g) T2 i FLAIR skupove podataka.

Slika 4.10 prikazuje primjere ostvarene segmentacije na FLAIR skupu podataka. Na slici se s lijeva na desno nalaze MR snimke, istinske lezije i segmentirane lezije, a crvenom konturom su radi lakše usporedbe označeni oblici pravih lezije.



Slika 4.10. Primjer izvršenih segmentacija lezija na FLAIR vrsti snimaka.

Model uspješno izdvaja veće lezije, ali nesavršenog oblika. Segmentacija manjih lezija se obično ne izvodi s visokom preciznošću, ali model uspješno ukazuje na većinu njihovih lokacija. Na slici 4.11 su prikazani primjeri segmentacije dobivene za T1 skup podataka. Uočljivo je da model potpuno neuspješno segmentira manje lezije, dok veće lezije mreža prepoznaje veoma loše.



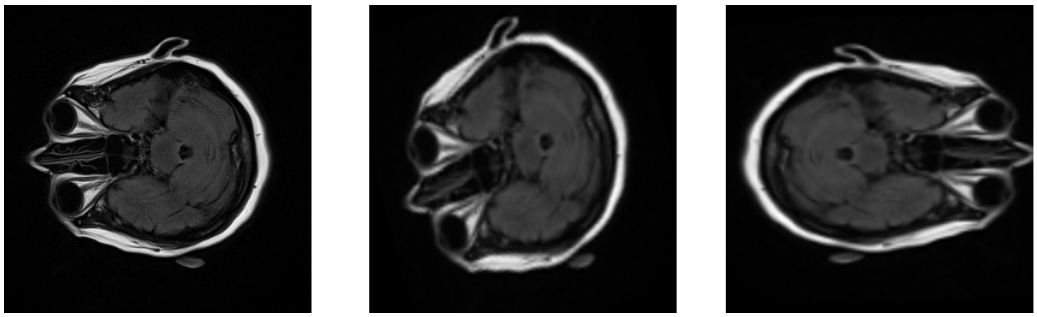
Slika 4.11. Primjer izvršenih segmentacija lezija na T1 vrsti snimaka.

5. Optimizacija rezultata

U prethodnom poglavlju su implementirane arhitekture konvolucijskih neuronskih mreža korištenjem izvornog skupa podataka, a hiperparametri su određivani tako da sve strukture imaju sličan broj parametara za treniranje kako bi se omogućila adekvatna usporedba. U ovom poglavlju, kako bi se poboljšali rezultati, primjenjuju se metode za proširivanje skupa podataka i optimiziranje hiperparametara pojedinačnih mreža za svaki podskup podataka.

5.1. Povećanje skupa podataka

Kod primjene dubokog učenja za rješavanje problema segmentacije slika, performanse neuronskih modela su uvelike ovisne o kvaliteti i količini podataka za treniranje. Stoga je važno ne samo imati veliki broj slika, već i osigurati njihovu raznolikost. Međutim, ponekad može biti izazovno pronaći dovoljan broj slika za specifičan tip zadatka. U takvim slučajevima, uobičajeno je primijeniti transformacije na postojeće slike kako bi se povećao skup podataka. Osim pomaka i zrcaljenja oko različitih osi, rotacija, zumiranje i izobličenje su također često korištene metode. Rotacija se implementira kako bi se povećala varijabilnost pozicija objekata na slici. Zumiranje povećava veličinu slike i pruža različite perspektive na objekte. Izobličenje se koristi kako bi se promijenio oblik objekata na slici, što pomaže u izbjegavanju prenaučenosti modela. Navedene promjene slika jednostavno se ostvaruju upotrebom $ImageDataGenerator$ ugrađene funkcije kojom se definiraju željeni parametri transformacija poput kuta rotacije, intenziteta izobličenja i osi zrcaljenja. Potrebno je definirati dva odvojena generatora za slike i odgovarajuće maske. Ti generatori se spajaju korištenjem zip naredbe, pri čemu se za svakoga postavljaju jednaki kriteriji odabira podataka kako bi se izvršile jednake transformacije na slikama i maskama. Zatim se raspisuje for petlja koja učitava postojeći skup podataka i primjenjuje generirane transformacije. Konačno, dobivene slike i maske se pohranjuju s odgovarajućim sufiksima imena radi lakšeg prepoznavanja. Iz eksperimenata s povećanjem skupa podataka zaključuje se da je najučinkovitije utrostručiti početnu količinu slika. Daljnjim povećanjem broja slika, ne postiže se značajno poboljšanje preciznosti, ali se povećava vrijeme potrebno za treniranje modela. Dakle, postoji neki optimum koji se mora postići pri izboru veličine skupa podataka kako bi se postigla dobra ravnoteža između performansi i vremena treniranja. Dakle, optimalna veličina skupa podataka je pronađena kako bi se postigla ravnoteža između performansi i vremena treniranja. Slika 5.1 prikazuje primjere iskorištenih transformacija. Slika s lijeve strane predstavlja izvornu sliku, dok se pored nje nalaze slike koje su generirane primjenom transformacija.



Slika 5.1. Primjer transformacija slike.

5.2. Optimizacija hiperparametara

Promatrane arhitekture neuronskih mreža se ispituju s obzirom na različite vrijednosti ključnih hiperparametara koji određuju njihovo ponašanje. Ovi hiperparametri uključuju broj filtara korištenih u konvolucijama, njihove dimenzije i korak pomicanja tijekom procesa konvolucije, broj obrađenih podataka (slika) tijekom jedne iteracije učenja, postotak odbacivanja težinskih faktora i stopa učenja. Ne postoji jednostavna i direktna veza između vrijednosti hiperparametara i performansi modela u problemima segmentacije slika. Na primjer, povećanje broja filtara korištenih u konvoluciji ne mora nužno dovesti do boljih rezultata. Također, isti model može zahtijevati različite postavke hiperparametara za svaki skup podataka, što otežava određivanje optimalnih vrijednosti hiperparametara za svaki pojedinačni model. Stoga se izvodi metoda optimizacije hiperparametara pomoću ekstenzivnog pretraživanja svake točke prostora rješenja. Prvo se raspisuju liste koje sadrže različite vrijednosti hiperparametara za model. Zatim se primjenom petlje i $append$ naredbe stvara lista koja sadrži sve moguće kombinacije definiranih hiperparametara. Nakon toga se kroz novi petlju inicira treniranje modela za svaku kombinaciju parametara. Po završetku svakog procesa učenja se računaju DCS vrijednosti za validacijski skup podataka, te se iscrtavaju krivulje učenja kako bi se mogle usporediti performanse različitih modela. Trajanje treniranja svakog modela skraćeno je na 40 epoha jer se temeljem prethodnih eksperimenata može zaključiti da funkcije gubitaka kod većine modela dostižu svoju konvergirajuću vrijednost nakon tog broja epoha. Ovim pristupom se štedi vrijeme i omogućava ispitivanje većeg broja kombinacija hiperaparametara. Primjećuje se da broj filtara i broj korištenih podataka po iteraciji imaju najveći utjecaj na ponašanje modela te se stoga najviše ispituju točke u prostoru rješenja koje odgovaraju različitim vrijednostima ovih hiperparametara. Nakon pronalaska njihovih optimalnih vrijednosti, istražuju se i utjecaji preostalih hiperparametara na performanse modela. Za FLAIR i ukupni skup podataka se ispituju desetci različitih kombinacija za sve arhitekture mreža, dok se za preostale podskupine istražuje znatno manji broj. Za različite arhitekture mreža i skupove podataka, utvrđeno je da se optimalan broj filtara u početnim slojevima mreže kreće u rasponu od 16 do 70, dok se primjereni broj slika po iteraciji nalazi u opsegu od 16 do 44. Za sve slučajeve pokazalo se da je prilagodljiva stopa učenja, koja se smanjuje ovisno o brzini konvergencije krivulje učenja, najprikladnija. Osim toga, optimalna vrijednost stope odbacivanja težinskih faktora varira između 0.05 i 0.2. Promjene

veličina matrica konvolucijskih filtara s dimenzija 3x3 i koraka 1 su rezultirale lošijim performansama svih arhitektura modela za sve skupove podataka.

5.3. Rezultati

Povećavanjem korištenog skupa podataka i optimizacijom hiperparametara neuronskih mreža postižu se rezultati prikazani u sljedećim tablicama. Tablica 5.1 prikazuje rezultate dobivene za SegNet, tablica 5.2 U-net, a tablica 5.3 za DeepLabV3+ arhitekturu.

Tablica 5.1. Prikaz rezultata SegNet arhitekture.

	DSC	Trajanje epohe [s]	Trajanje treninga [h:min:s]
T1	9.27%	16	0:19:24
T2	32.6%	15	0:28:20
FLAIR	55.39%	19	0:17:25
Ukupno	37.79%	94	2:15:51
T1+T2	25.95%	33	0:33:53
T1+FLAIR	42.72%	36	1:02:44
T2+FLAIR	45.74%	37	0:41:33

Tablica 5.2. Prikaz rezultata U-net arhitekture.

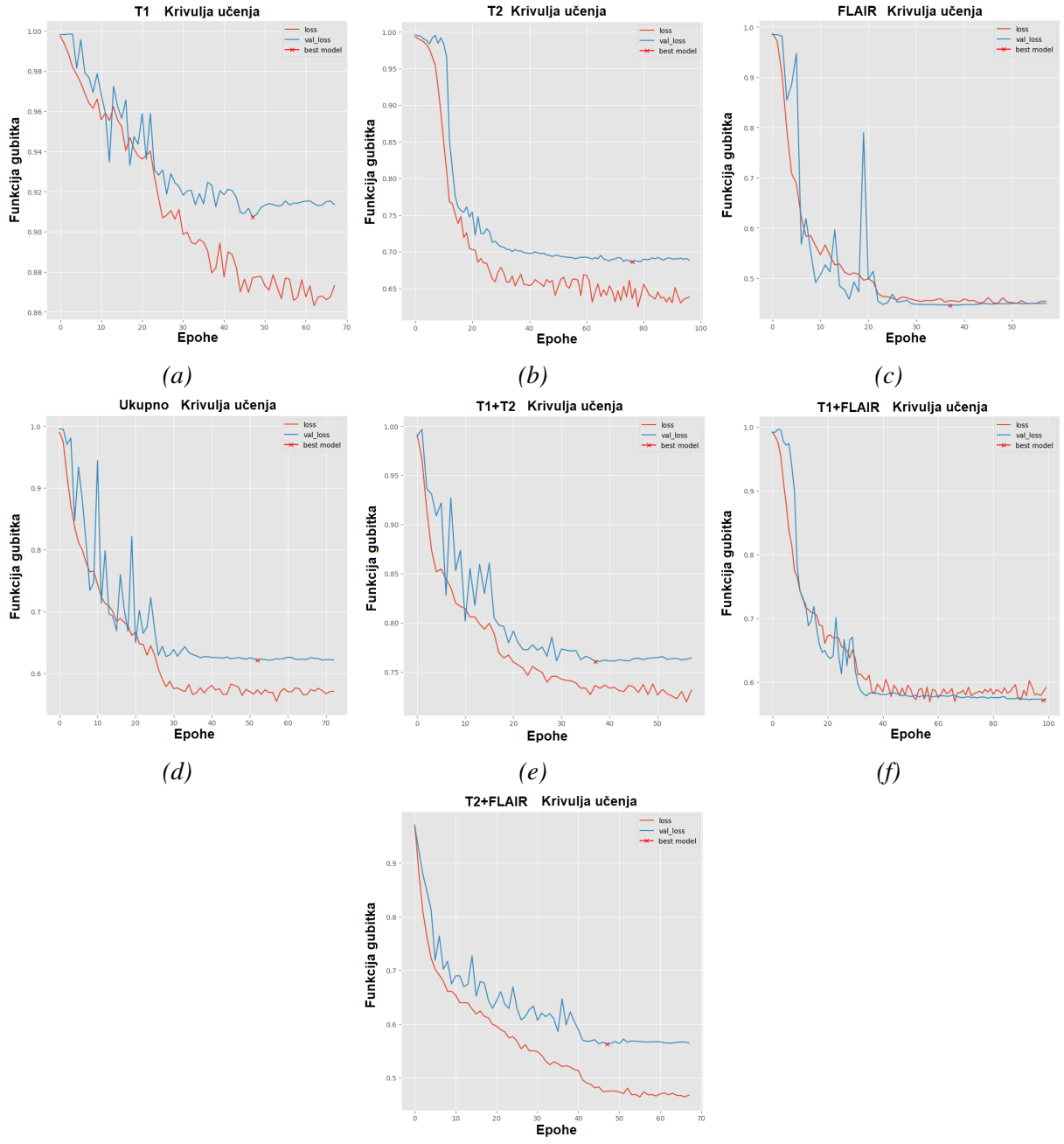
	DSC	Trajanje epohe [s]	Trajanje treninga [h:min:s]
T1	24.32%	14	0:23:19
T2	51.66%	13	0:21:47
FLAIR	63.89%	35	0:36:32
Ukupno	53.07%	82	2:17:30
T1+T2	40.54%	27	0:26:42
T1+FLAIR	59.28%	29	0:32:06
T2+FLAIR	55.63%	30	0:49:29

Tablica 5.3. Prikaz rezultata DeepLabV3+ arhitekture.

	DSC	Trajanje epohe [s]	Trajanje treninga [h:min:s]
T1	13.08%	12	0:11:00
T2	32.93%	11	0:07:03
FLAIR	58.53%	12	0:11:59
Ukupno	47.79%	46	0:32:04
T1+T2	31.73%	23	0:16:54
T1+FLAIR	52.42%	27	0:32:04
T2+FLAIR	47.12%	25	0:30:11

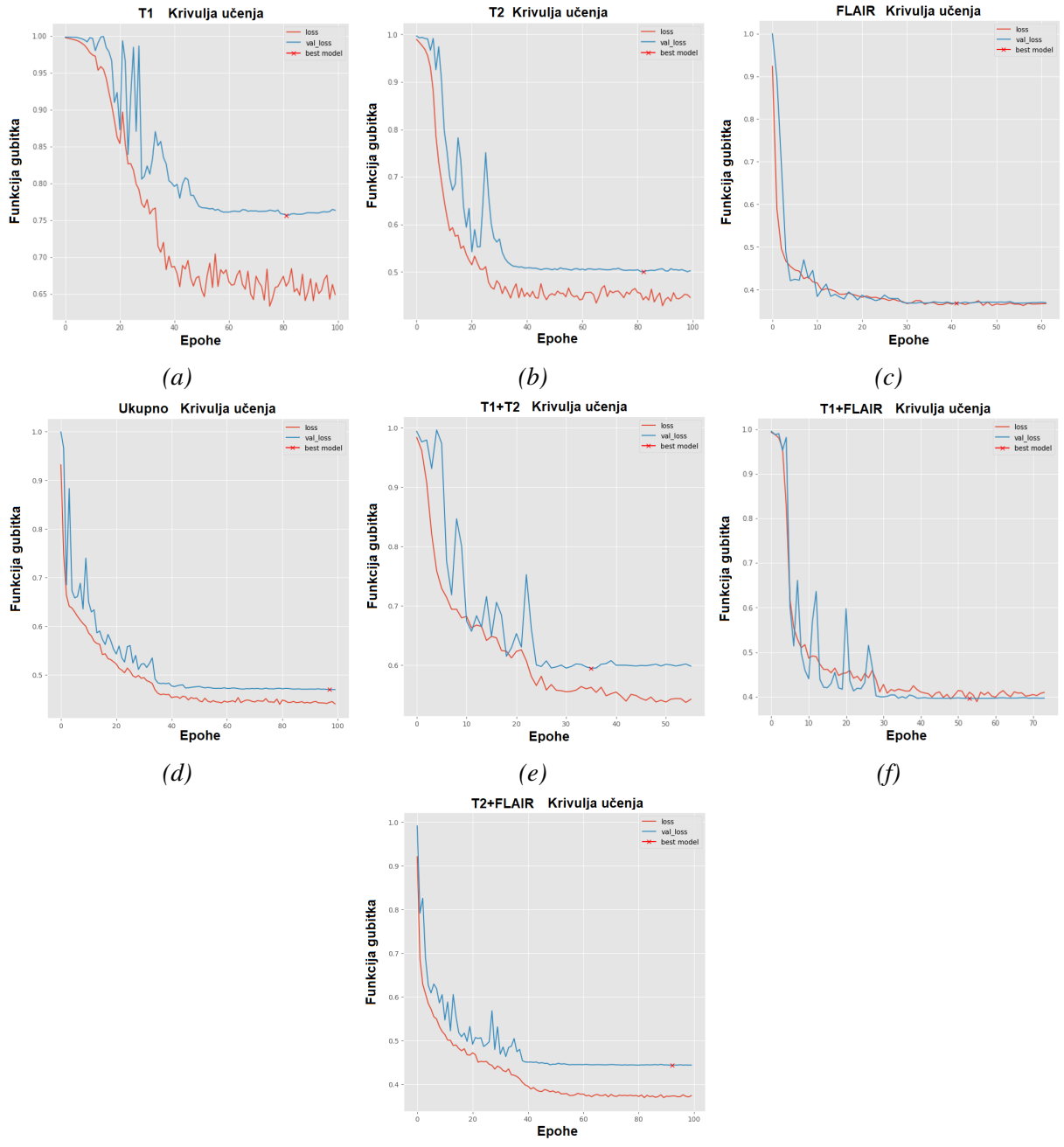
Rezultati ukazuju da su najbolje performanse segmentacije postignute na FLAIR skupu podataka, s postotkom preciznosti između 55.39% i 63.89% po DCS-u. Ovaj rezultat je očekivan, s obzirom na to da je FLAIR vrsta snimke na kojoj se lezije najviše razlikuju od okolnog tkiva. Najlošiji rezultati postignuti su na T1 skupu podataka (9.27%-24.32%). Pretpostavlja se da je uzrok

toga činjenica da se lezije na T1 slikama uopće ne razlikuju od okolnog tkiva, te što T1 skup podataka sadrži najmanju količinu segmentiranih lezija, odnosno najveći broj maski koje odgovaraju tom skupu su potpuno crne. Kao i kod prvotnih promatranja u prošlom poglavlju, sve arhitekture modela trenirane na skupovima podataka koji uključuju više vrsta snimki, za T1 i T2, pružaju bolje rezultate u odnosu na modele trenirane na pojedinačnim vrstama. S druge strane, za FLAIR podatke, model koji je treniran samo na FLAIR skupu podataka postiže najvišu preciznost. Povećanjem broja slika korištenih tijekom strojnog učenja i optimizacijom hiperparametara ostvareni su poboljšani rezultati u rasponu od 2 do 15%, ovisno o korištenoj arhitekturi i podskupu podataka. Ipak, primijećeno je značajno povećanje vremena potrebnog za treniranje modela. Među ispitivanim arhitekturama, U-net se ističe kao najpreciznija, ali istovremeno i kao najsporije trenirajuća, dok SegNet pruža najlošije rezultate. DeepLab i SegNet segmentiraju s približno jednakom točnošću, no DeepLab arhitektura se izdvaja kao najbrže trenirajuća od svih. To je zbog činjenice da se u DeepLab modelima koristi unaprijed istrenirana ResNet50 mreža za ekstrakciju lokalnih značajki. Kao rezultat toga, tijekom proces učenja je potrebno prilagođavati težinske faktore manjeg broja filtara koji se koriste u posljednjim slojevima i slojevima atrozne konvolucije. Osim toga, tijekom optimizacije hiperparametara uočava se da veći broj filtara značajno utječe na vrijeme potrebno za treniranje U-net arhitekture, dok se ovaj utjecaj manje primjećuje kod SegNet arhitekture, a gotovo zanemaruje kod DeepLab arhitekture. Proučavanjem krivulja učenja za sve tri arhitekture, zaključuje se da je povećanje korištenih slika uklonilo problem sposobnosti generalizacije modela u slučaju za FLAIR, ukupne i T1+FLAIR skupove podataka. Međutim, za T1 i T2, modeli i dalje bolje segmentiraju podatke za treniranje nego validaciju. To su ujedno i skupovi koji se karakteriziraju najvećim oscilacijama funkcije gubitka tijekom procesa učenja, što ukazuje na teškoće u konvergiranju rješenja. Na slici 5.2(a) do (g) prikazane su krivulje učenja SegNet arhitekture za sve skupine podataka. Plava boja označava funkcije gubitka za validacijski, a crvena za trenirajući skup.



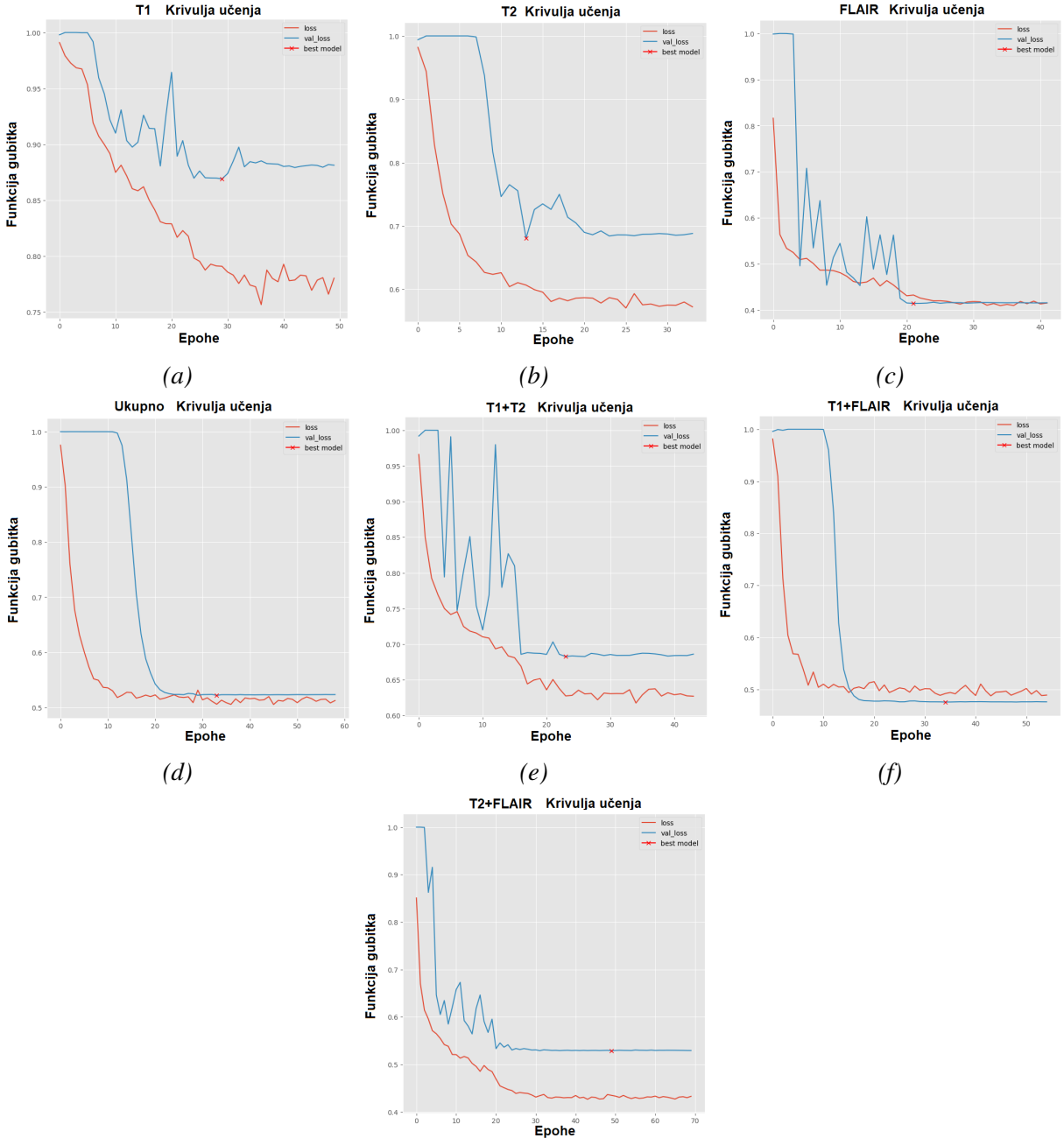
Slika 5.2. Krivulje učenja za (a) T1, (b) T2, (c) FLAIR, (d) Ukupni, (e) T1 i T2, (f) T1 i FLAIR, (g) T2 i FLAIR skupove podataka SegNet arhitekture.

Slike 5.3(a) do (g) prikazuju krivulje učenja U-net arhitekture. Vidljivo je da ovi modeli imaju bolju sposobnost generalizacije podataka u usporedbi sa SegNet arhitekturom.



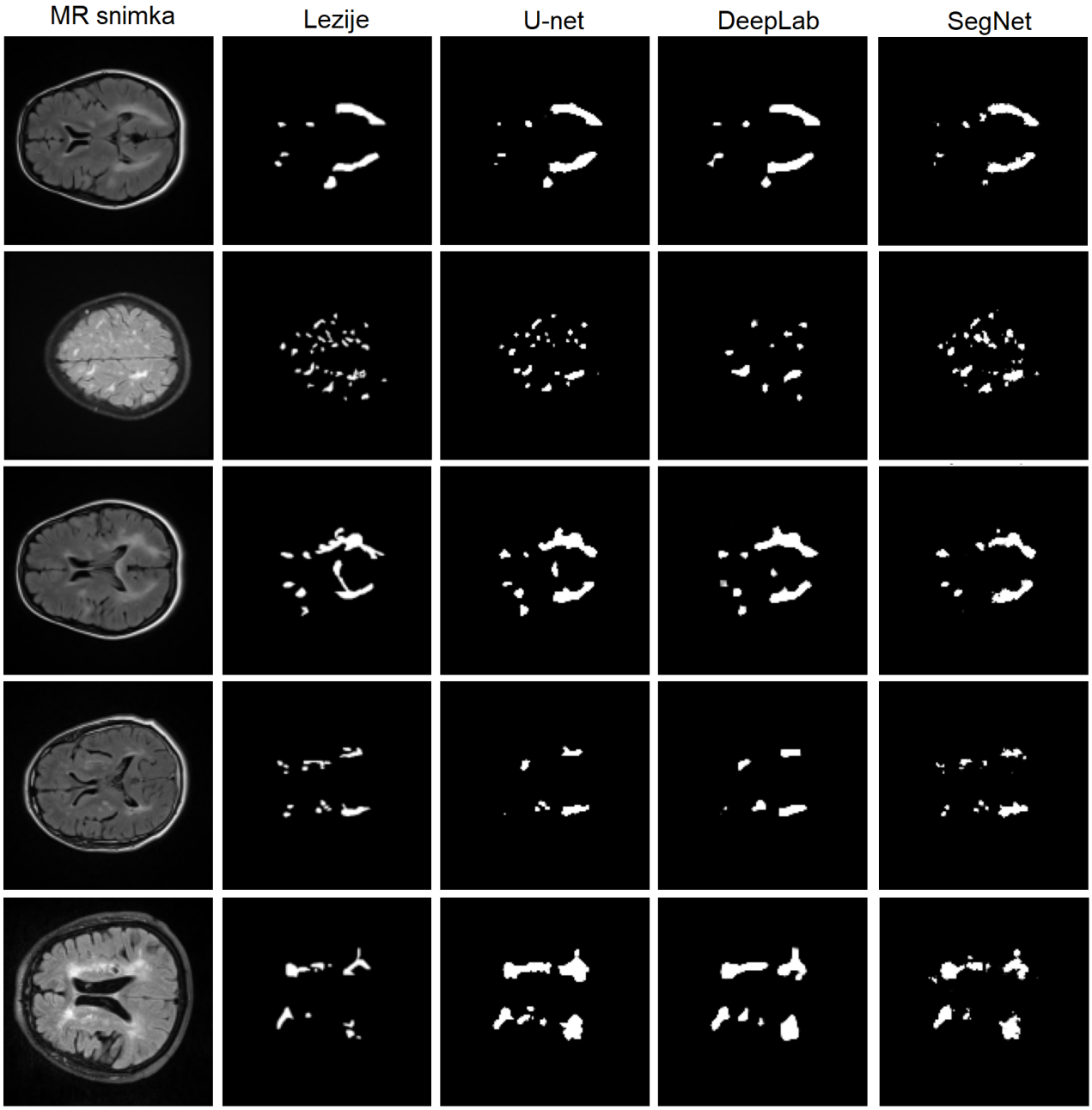
Slika 5.3. Krivulje učenja za (a) T1, (b) T2, (c) FLAIR, (d) Ukupni, (e) T1 i T2, (f) T1 i FLAIR, (g) T2 i FLAIR skupove podataka U-net arhitekture.

Na slici 5.4(a) do (g) prikazane su krivulje učenja DeepLabV3+ arhitekture.



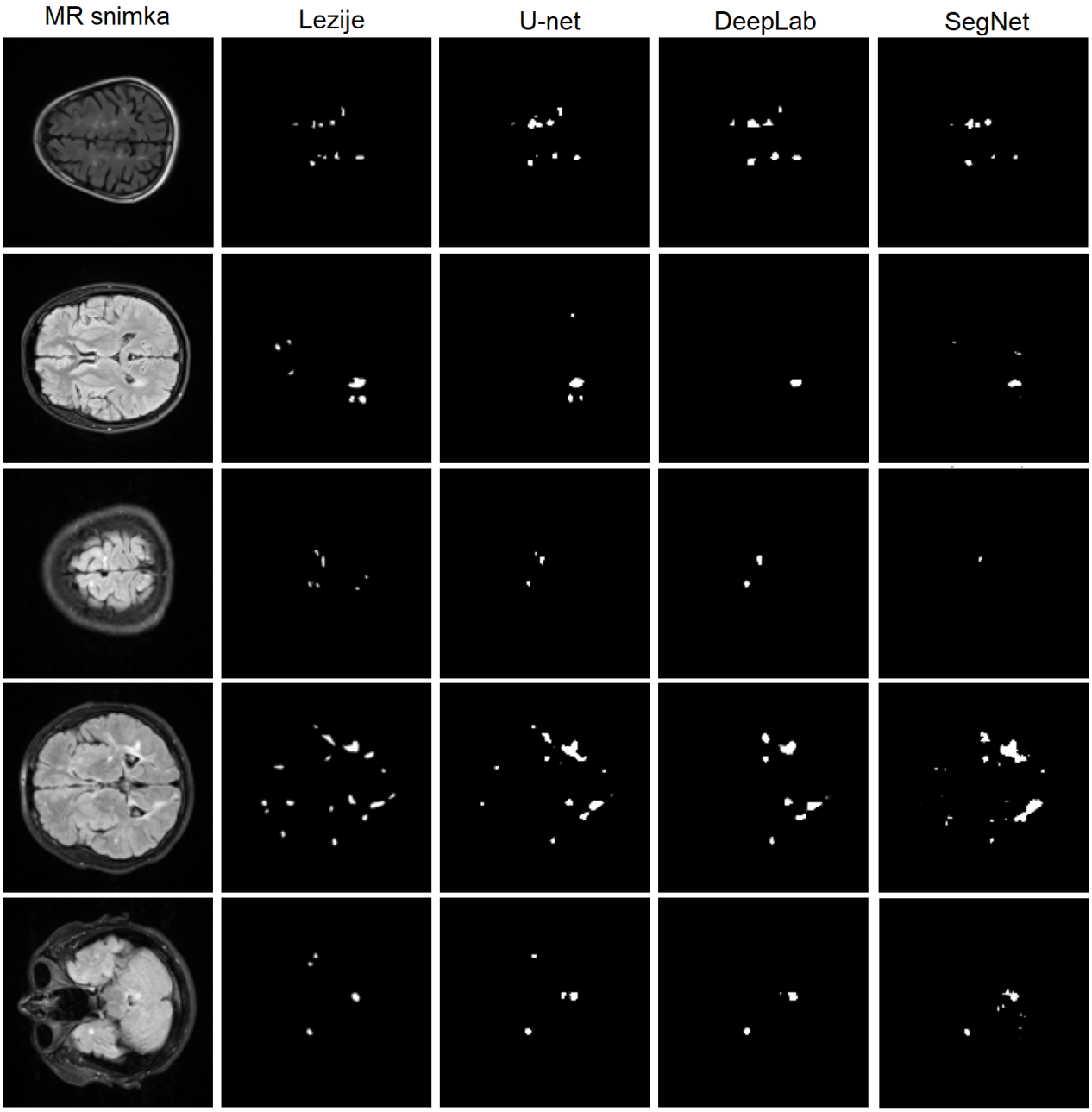
Slika 5.4. Krivulje učenja za (a) T1, (b) T2, (c) FLAIR, (d) Ukupni, (e) T1 i T2, (f) T1 i FLAIR, (g) T2 i FLAIR skupove podataka DeepLabV3+ arhitekture.

U nastavku su dani primjeri segmentacija dobivenih korištenjem ostvarenih modela. Slika 5.5 prikazuje segmentacije za FLAIR tip snimki. Vidljivo je da modeli uspješno izlučuju veće lezije, ali njihov oblik nije uvijek savršeno segmentiran. Kod površina koje sadrže više manjih lezija, modeli dobro lociraju regiju, ali pojedinačni oblici i točne pozicije često se ne podudaraju s istinitim vrijednostima.



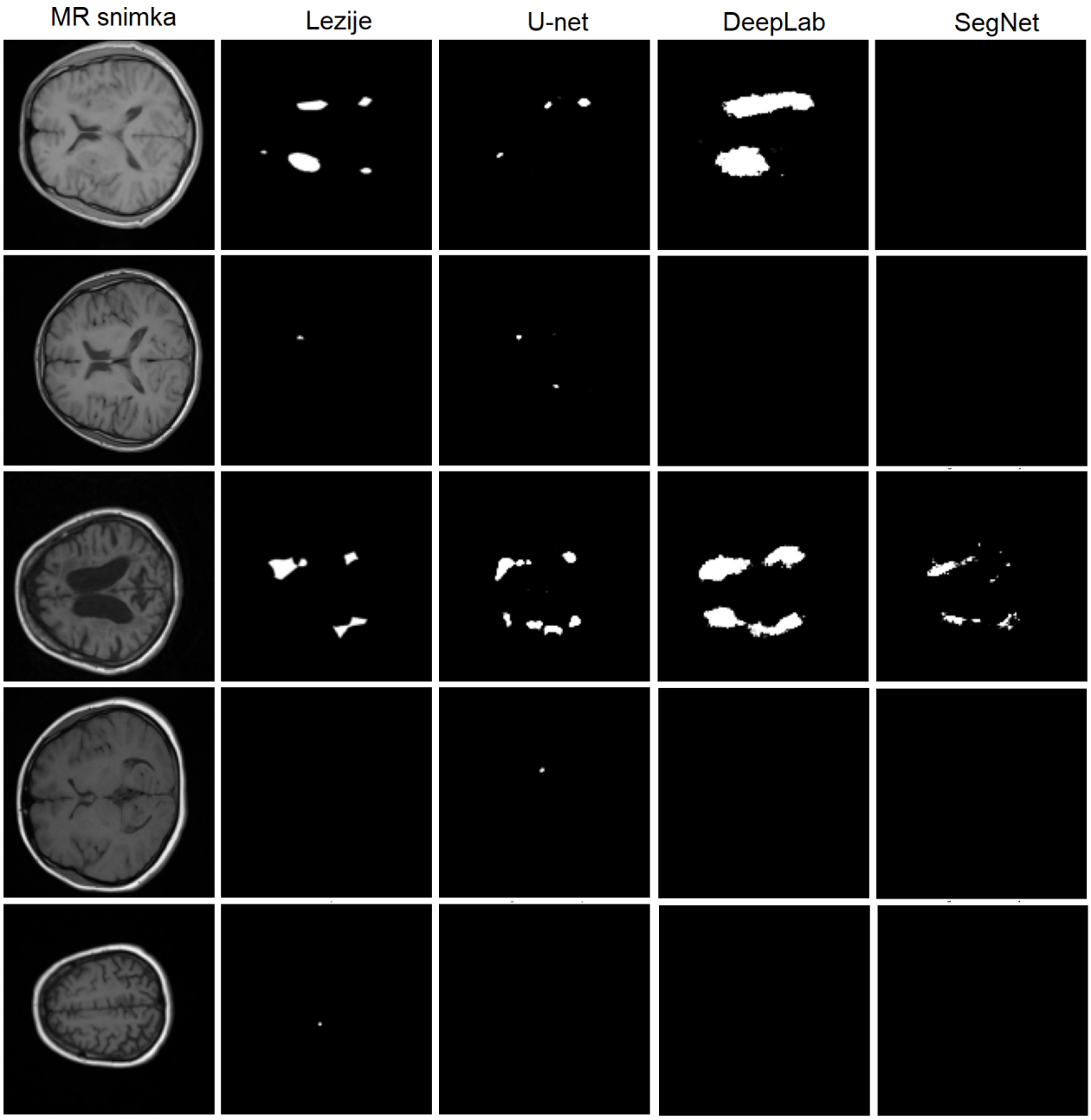
Slika 5.5. Primjer izvršenih segmentacija lezija na FLAIR vrsti snimaka.

Slika 5.6 prikazuje još primjera segmentacije za FLAIR tip snimki. Uočljivo je da U-net model pruža najpreciznije rezultate. Za ovaj skup podataka, može se zaključiti da su postignuti zadovoljavajući rezultati koji bi mogli biti od koristi stručnjacima za efikasniju i precizniju segmentaciju.



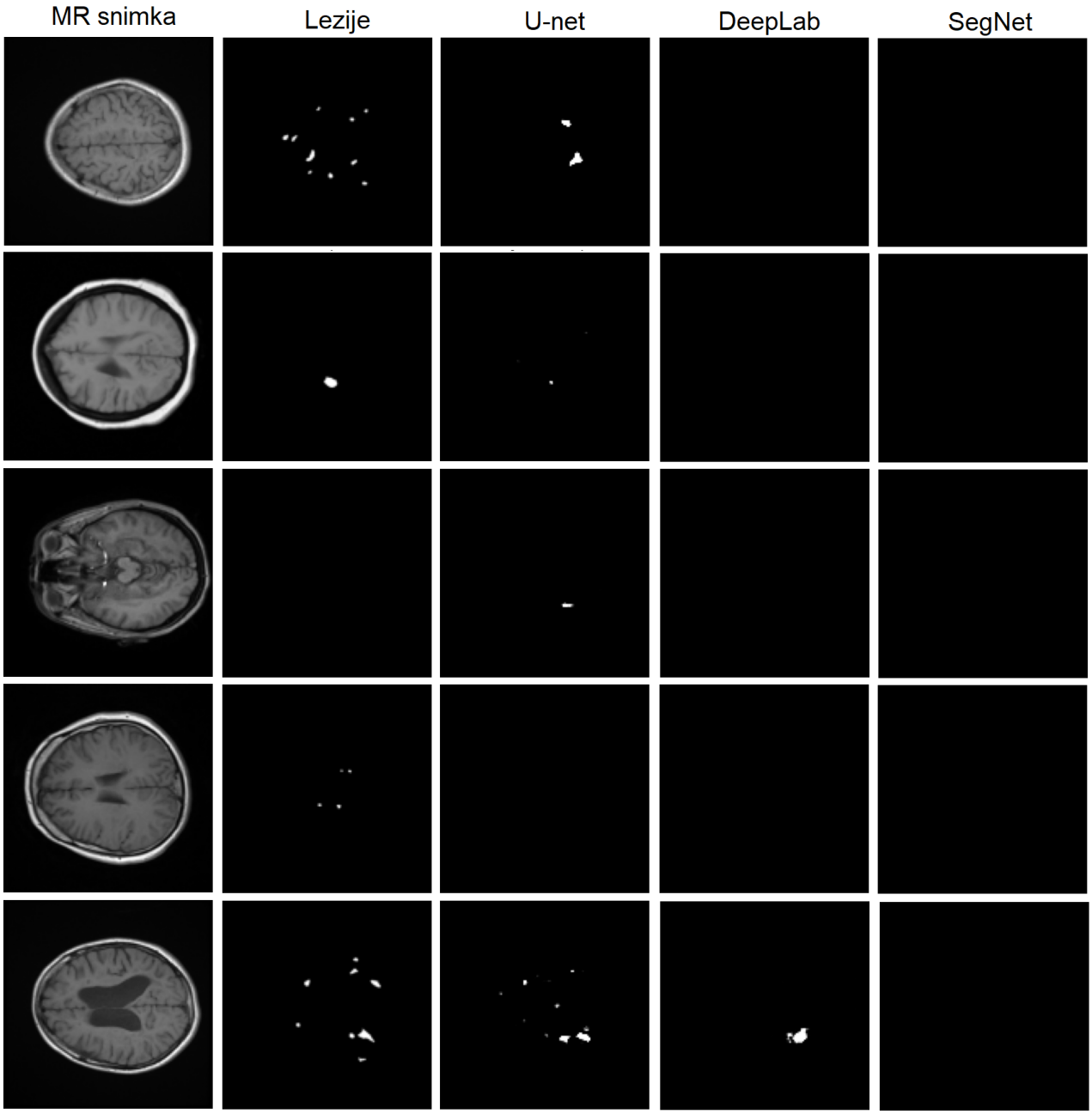
Slika 5.6. Primjer izvršenih segmentacija lezija na FLAIR vrsti snimaka.

Slike 5.7 prikazuje ostvarene segmentacije s najnižom preciznošću za T1 skup podataka. Vidljivo je da se velike lezije veoma loše prepoznaju, dok se manje gotovo potpuno neuspješno segmentiraju.



Slika 5.7. Primjer izvršenih segmentacija lezija na T1 vrsti snimaka.

Slika 5.8 također prikazuje primjere segmentacija za T1 skup podataka. Čak i za najpreciznij U-net model, rezultati nisu dovoljno dobri za praktičnu primjenu.



Slika 5.8. Primjer izvršenih segmentacija lezija na T1 vrsti snimaka.

6. Zaključak

Svrha ovog rada je bila upoznavanje s osnovama strojnog učenja i moguće primjene za segmentaciju lezija na mozgu u svrhu dijagnosticiranja multiple skleroze. U drugom poglavlju se iznosi teorijska podloga o magnetskoj rezonanciji s ciljem razumijevanja korištenog skupa podataka, odnosno snimaka ljudskog mozga. Osnovna svojstva umjetnih neuronskih mreža su dana u trećem poglavlju, pri čemu se opisuje struktura i način rada osnovnih oblika neuronskih mreža. Nadalje se opisuju temelji strojnog učenja, gdje se promatraju različite metode učenja. Također se opisuje razlika između generalizacije i učenja kako bi se mogli shvatiti pojmovi prekomjernog ili nedovoljnog uklapanja, te se uvodi optimiziranja hiperparametara. S obzirom na to da su konvolucijske neuronske mreže pogodne i najčešće primjenjivane za obradu slika, one se također detaljnije opisuju. Zatim se definiraju SegNet, U-net i DeepLabV3+ arhitekture umjetnih neuronskih mreža pogodnih za promatrani zadatak. Tema četvrtog poglavlja je implementacija opisanih algoritama za segmentaciju lezija na mozgu u Python programskom jeziku, posebno za svaki tip MR snimki. Konačno, u posljednjem poglavlju se pokušavaju poboljšati prvotni dobiveni rezultati korištenjem tehnika za povećanje skupa podataka i optimizaciju hiperparametara.

Postignuti rezultati pokazuju da su najpreciznije segmentacije dobivene na FLAIR skupu podataka, što se može objasniti činjenicom da se na toj vrsti snimki lezije najviše razlikuju od okolnog tkiva. Postotak preciznosti po DCS-u iznosi 55.39% do 63.89%, ovisno o vrsti korištene arhitekture mreže. Modeli istrenirani na T2 skupu podataka ostvarili su preciznost od 32.6% do 51.66%. Za T1 skup podataka su ostvareni najlošiji rezultati u rasponu od 9.27% do 24.32%. Vjerojatni uzrok tome je činjenica da se lezije na T1 slikama najmanje razlikuju od zdravog tkiva. Osim toga, T1 skup podataka sadrži najmanju količinu segmentiranih lezija, odnosno najveći broj maski koje odgovaraju tom skupu su potpuno crne. Sve navedene arhitekture modela pokazale su se učinkovitima kada su trenirane na skupovima podataka koji uključuju više vrsta snimki. U usporedbi s modelima treniranima na pojedinačnim vrstama, za T1 i T2 vrstu snimki, kombinirani modeli pružaju bolje rezultate. Za FLAIR podatke, model koji je treniran samo na FLAIR skupu podataka postiže najvišu preciznost u segmentaciji lezija na mozgu, što ukazuje na važnost korištenja odgovarajućih skupova podataka za treniranje modela. Od promatranih arhitektura, U-net se iskazuje kao najpreciznija, dok se DeepLab arhitektura ističe bržim procesom učenja uslijed činjenice da koristi unaprijed istreniranu ResNet50 mrežu za ekstrakciju lokalnih značajki. Vizualna evaluacija dobivenih segmentacija pokazuje da za FLAIR tip podataka modeli uspješno izlučuju veće lezije, ali uz nesavršen oblik. Kod površina koje sadrže više manjih lezija, modeli dobro lociraju regiju, no pojedinačni oblici i točne pozicije često se ne podudaraju s istinitim vrijednostima. Unatoč tome, rezultati su zadovoljavajući i mogu biti od velike koristi medicinskim stručnjacima kao dijagnostički alat za efikasniju i precizniju segmentaciju. Međutim, ostvareni modeli nisu dovoljno precizni da bi se koristili za oblikovanje percepcije robotskih manipulatora koji se koriste u medicinskim

zahvatima. S druge strane, za T1 skup podataka, vizualnom procjenom dobivenih rezultata može se zaključiti da se velike lezije veoma loše prepoznaju, dok se manje gotovo potpuno neuspješno segmentiraju. Odnosno, čak i najprecizniji U-net model nije uspio postići dovoljno dobre rezultate za praktičnu primjenu u ovom skupu podataka. Važno je napomenuti da su rezolucije korištenih slika u ovom istraživanju vrlo male (128x128 piksela) i da se za postizanje boljih rezultata u budućnosti preporučuje korištenje većih rezolucija slika. Također, jedan od načina za poboljšanje rezultata je proširenje korištenog skupa podataka, ne samo većim brojem slika, nego i uključivanjem trodimenzionalnih snimki, što bi moglo rezultirati poboljšanom preciznošću segmentacije. Naime, treba imati na umu da bi takvo proširenje skupa podataka također zakompliciralo korištene algoritme.

Bibliografija

[1] Yang Z.H., Feng F., Wang X.Y.: "MRI Guide-Check Specification, Clinical Strategies and Application of New Technologies", People's Medical Publishing House, Beijing, 2010.

[2] Huang J.Y. and Liang X.Y.: "Principles of Magnetic Resonance Imaging", Shaanxi Science and Technology Press, Shaanxi, 1998.

[3] Car Z., Baressi Šegota S., Anđelić N., Lorencin, I., Mrzljak V.: "Modeling the Spread of COVID-19 Infection Using a Multilayer Perceptron.", Computational and Mathematical Methods in Medicine, 2020.

[4] Du K., Swamy M. N. S.: "Neural Networks and Statistical Learning", Springer, Berlin, 2013.

[5] Barto A.G., Suton R.S., Anderson C.W.: "Neuronlike adaptive elements that can solve difficult learning control problems", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, No. 5, pp. 834-846, 1983.

[6] Feurer M., Hutter F.: "Hyperparameter Optimization." iz Hutter F., Kotthoff L., Vanschoren J.: "Automated Machine Learning. The Springer Series on Challenges in Machine Learning." Springer, Cham, 2019.

[7] Simard P.Y., Steinkraus D., Platt J.C.: "Best practices for convolutional neural networks applied to visual document analysis", Seventh International Conference on Document Analysis and Recognition, pp. 958-963, 2003.

[8] Nakazawa T., Kulkarni D. V.: "Anomaly Detection and Segmentation for Wafer Defect Patterns Using Deep Convolutional Encoder–Decoder Neural Network Architectures in Semiconductor Manufacturing", IEEE Transactions on Semiconductor Manufacturing, Vol. 32, No. 2, pp. 250-256, 2019.

[9] Li Y., Xu S., Luo X., Lin S.: "A new algorithm for product image search based on salient edge characterization.", Journal of the Association for Information Science and Technology, Vol. 65, No. 12, pp. 2534-2551.

[10] Badrinarayanan V., Cipolla R., Kendall A.: "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation", in IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39, No. 12, pp. 2481-2495, 2017.

[11] Baressi Šegota S., Lorencin I., Smolić K., Anđelić N., Markić D., Mrzljak V., Štifanić D., Musulin J., Španjol J., Car Z.: "Semantic Segmentation of Urinary Bladder Cancer Masses from CT Images: A Transfer Learning Approach.", Biology, Vol. 10, No. 11, EAN 1134, 2021.

[12] Ronneberg O., Fischer P., Brox T.: "U-Net: Convolutional networks for Biomedical Image Segmentation", ArXiv, 2015.

[13] Lamba H.: "Understanding Semantic Segmentation with UNET", s interneta, https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47, 17.2.2019.

[14] Chen L.C., Papandreou G., Schroff F., Adam H.: "Rethinking Atrous Convolution for Semantic Image Segmentation", ArXiv, 2017.

[15] Almutairi A.D., Alnuaimi O., Gawwam G.A., Hanafi M.B., Mahmud R., Mashohor S., Muslim A.,M.: "Brain MRI Dataset of Multiple Sclerosis with Consensus Manual Lesion Segmentation and Patient Meta Information", National Center for Biotechnology Information, Bethesda, Maryland, 2022.

[16] Laurence A.: "NIfTI Image Converter (nii2png) for Python and Matlab" , s interneta: https://github.com/alexlaurence/NIfTI-Image-Converter/, 28.7.2021.

[17] S interneta: https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50, 23.3.2023.

Sažetak i ključne riječi

Ovaj diplomski rad istražuje segmentaciju moždanih lezija uzrokovanih multiplom sklerozom. Opisuju se temelja strojnog učenja i predstavljaju tri arhitekture umjetnih neuronskih mreža prikladnih za promatrani zadatak. Zatim se implementiraju opisani algoritmi, te se provodi evaluacija rezultata. Najbolji rezultati ostvaruju se za FLAIR vrstu snimki magnetske rezonance korištenjem U-net arhitekture, s postignutom preciznošću od 63.89% po DCS-u. DeepLabV3+ pruža točnost segmentacije od 58.53%, a SegNet 55.39%. Ovakvi rezultati su zadovoljavajući i mogu biti korisni medicinskim stručnjacima kao dijagnostički alat za efikasniju i precizniju segmentaciju.

Ključne riječi: Umjetna inteligencija, segmentacija, U-net, SegNet, DeepLabV3+, Python, snimke magnetske rezonance

Summary and key words

This paper explores the segmentation of brain lesions caused by multiple sclerosis. It describes the basics of machine learning and presents three artificial neural network architectures suitable for the task at hand. The described algorithms are then implemented, and the results are evaluated. The best results are achieved for the FLAIR type of magnetic resonance imaging using the U-Net architecture, with achieved segmentation accuracy of 63.89% based on DCS. DeepLabV3+ provides segmentation accuracy of 58.53%, while SegNet achieves 55.39%. These results are satisfactory and can be useful for medical professionals as a diagnostic tool for more efficient and precise segmentation.

Keywords: Artificial intelligence, segmentation, U-net, SegNet, DeepLabV3+, Python, magnetic resonance imaging

A Funkcija za izgradnju SegNet modela

```
# -*- coding: utf-8 -*-

#Konvolucijski blokovi
def Conv2D_Blok(tensor, n_filters, k_size, batchnorm=True):
    #Prvi sloj
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(tensor)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    return x

def Conv2D_Blok2(tensor, n_filters, k_size, batchnorm=True):
    #Prvi sloj
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(tensor)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    #Drugi sloj
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    return x

def Conv2D_Blok3(tensor, n_filters, k_size, batchnorm=True):
    #Prvi sloj
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(tensor)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)
```

```
    #Drugi sloj
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    #Treci sloj
    x = Conv2D(filters=n_filters, kernel_size=(k_size, k_size),\
              kernel_initializer='he_normal', padding='same')(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    return x

#SEGNET
def segnet(ulaz_oblik, n_filters, dropout, k_size, batchnorm = True):
    #Enkoder
    ulaz = Input(shape=ulaz_oblik)

    c_1 = Conv2D_Blok2(ulaz, n_filters * 1, k_size = k_size,
                          batchnorm = batchnorm)
    p_1, mask_1 = MaxPoolingWithArgmax2D((2,2))(c_1)
    p_1 = Dropout(dropout)(p_1)

    c_2 = Conv2D_Blok2(p_1, n_filters * 2, k_size = k_size,
                          batchnorm = batchnorm)
    p_2, mask_2 = MaxPoolingWithArgmax2D((2,2))(c_2)
    p_2 = Dropout(dropout)(p_2)

    c_3 = Conv2D_Blok3(p_2, n_filters * 4, k_size = k_size,
                          batchnorm = batchnorm)
    p_3, mask_3 = MaxPoolingWithArgmax2D((2,2))(c_3)
    p_3 = Dropout(dropout)(p_3)

    c_4 = Conv2D_Blok3(p_3, n_filters * 8, k_size = k_size,
                          batchnorm = batchnorm)
    p_4, mask_4 = MaxPoolingWithArgmax2D((2,2))(c_4)
    p_4 = Dropout(dropout)(p_4)
```

```
c_5 = Conv2D_Blok3(p_4, n_filters * 8, k_size = k_size,
                       batchnorm = batchnorm)
p_5, mask_5 = MaxPoolingWithArgmax2D((2,2))(c_5)
p_5 = Dropout(dropout)(p_5)
print("Enkoder je izgraden")

#Dekoder

up_1 = tfa.layers.MaxUnpooling2D()(p_5, mask_5)

c_6 = Conv2D_Blok3(up_1, n_filters * 8, k_size = k_size,
                       batchnorm = batchnorm)
up_2 = tfa.layers.MaxUnpooling2D()(c_6, mask_4)
up_2 = Dropout(dropout)(up_2)

c_7 = Conv2D_Blok2(up_2, n_filters * 8, k_size = k_size,
                       batchnorm = batchnorm)
c_8 = Conv2D_Blok(c_7, n_filters * 4, k_size = k_size,
                      batchnorm = batchnorm)
up_3 = tfa.layers.MaxUnpooling2D()(c_8, mask_3)
up_3 = Dropout(dropout)(up_3)

c_9 = Conv2D_Blok2(up_3, n_filters * 4, k_size = k_size,
                       batchnorm = batchnorm)
c_10 = Conv2D_Blok(c_9, n_filters * 2, k_size = k_size,
                       batchnorm = batchnorm)
up_4 = tfa.layers.MaxUnpooling2D()(c_10, mask_2)
up_4 = Dropout(dropout)(up_4)

c_11 = Conv2D_Blok(up_4, n_filters * 2, k_size = k_size,
                       batchnorm = batchnorm)
c_12 = Conv2D_Blok(c_11, n_filters * 1, k_size = k_size,
                       batchnorm = batchnorm)
up_5 = tfa.layers.MaxUnpooling2D()(c_12, mask_1)
up_5 = Dropout(dropout)(up_5)

c_13 = Conv2D_Blok(up_5, n_filters * 1, k_size = k_size,
                       batchnorm = batchnorm)

izlaz = Conv2D(1, (1, 1), activation='sigmoid')(c_13)
```

```
    print("Dekoder je izgraden")

    model = Model(inputs=ulaz, outputs=izlaz, name="SegNet")

    return model
```

A1. U-net

```
#UNET
def unet(ulaz_oblik, n_filters, dropout, k_size, batchnorm = True):
    #Enkoder
    ulaz = Input(shape=ulaz_oblik)

    c_1 = Conv2D_Blok2(ulaz, n_filters * 1, k_size = k_size,
                       batchnorm = batchnorm)
    p_1 = MaxPooling2D((2, 2))(c_1)
    p_1 = Dropout(dropout)(p_1)

    c_2 = Conv2D_Blok2(p_1, n_filters * 2, k_size = k_size,
                       batchnorm = batchnorm)
    p_2 = MaxPooling2D((2, 2))(c_2)
    p_2 = Dropout(dropout)(p_2)

    c_3 = Conv2D_Blok2(p_2, n_filters * 4, k_size = k_size,
                       batchnorm = batchnorm)
    p_3 = MaxPooling2D((2, 2))(c_3)
    p_3 = Dropout(dropout)(p_3)

    c_4 = Conv2D_Blok2(p_3, n_filters * 8, k_size = k_size,
                       batchnorm = batchnorm)
    p_4 = MaxPooling2D((2, 2))(c_4)
    p_4 = Dropout(dropout)(p_4)

    c_5 = Conv2D_Blok2(p_4, n_filters = n_filters * 16, k_size = k_size,
                       batchnorm = batchnorm)
    print("Enkoder je izgraden")
    #Dekoder
    u_6 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2),
                          padding = 'same')(c_5)
    u_6 = concatenate([u_6, c_4])
    u_6 = Dropout(dropout)(u_6)
```

```
    c_6 = Conv2D_Blok2(u_6, n_filters * 8, k_size = k_size,
                       batchnorm = batchnorm)

    u_7 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2),
                          padding = 'same')(c_6)
    u_7 = concatenate([u_7, c_3])
    u_7 = Dropout(dropout)(u_7)
    c_7 = Conv2D_Blok2(u_7, n_filters * 4, k_size = k_size,
                       batchnorm = batchnorm)

    u_8 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2),
                          padding = 'same')(c_7)
    u_8 = concatenate([u_8, c_2])
    u_8 = Dropout(dropout)(u_8)
    c_8 = Conv2D_Blok2(u_8, n_filters * 2, k_size = k_size,
                       batchnorm = batchnorm)

    u_9 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2),
                          padding = 'same')(c_8)
    u_9 = concatenate([u_9, c_1])
    u_9 = Dropout(dropout)(u_9)
    c_9 = Conv2D_Blok2(u_9, n_filters * 1, k_size = k_size,
                       batchnorm = batchnorm)

    izlaz = Conv2D(1, (1, 1), activation='sigmoid')(c_9)
    print("Dekoder je izgraden")
    model = Model(inputs=ulaz, outputs=izlaz, name="Unet")

    return model
```

A2. DeepLabV3+

```
def ASPP(inputs, n_filters):
    shape = inputs.shape

    y_pool = AveragePooling2D(pool_size=(shape[1], shape[2]),
                              name='average_pooling')(inputs)
    y_pool = Conv2D(filters=n_filters, kernel_size=1,
                    padding='same', use_bias=False)(y_pool)
    y_pool = BatchNormalization()(y_pool)
    y_pool = Activation('relu')(y_pool)
```

```
    y_pool = UpSampling2D((shape[1], shape[2]),
                          interpolation="bilinear")(y_pool)

    y_1 = Conv2D(filters=n_filters, kernel_size=1, dilation_rate=1,
                 padding='same', use_bias=False)(inputs)
    y_1 = BatchNormalization()(y_1)
    y_1 = Activation('relu')(y_1)

    y_6 = Conv2D(filters=n_filters, kernel_size=3, dilation_rate=6,
                 padding='same', use_bias=False)(inputs)
    y_6 = BatchNormalization()(y_6)
    y_6 = Activation('relu')(y_6)

    y_12 = Conv2D(filters=n_filters, kernel_size=3, dilation_rate=12,
                  padding='same', use_bias=False)(inputs)
    y_12 = BatchNormalization()(y_12)
    y_12 = Activation('relu')(y_12)

    y_18 = Conv2D(filters=n_filters, kernel_size=3, dilation_rate=18,
                  padding='same', use_bias=False)(inputs)
    y_18 = BatchNormalization()(y_18)
    y_18 = Activation('relu')(y_18)

    y = concatenate([y_pool, y_1, y_6, y_12, y_18])

    y = Conv2D(filters=n_filters, kernel_size=1, dilation_rate=1,
               padding='same', use_bias=False)(y)
    y = BatchNormalization()(y)
    y = Activation('relu')(y)
    return y

def DeepLabV3Plus(ulaz_oblik, n_filters, dropout):
    #Enkoder
    ulaz = Input(ulaz_oblik)

    #ResNet
    base_model = ResNet50(weights='imagenet', include_top=False,
                          input_tensor=ulaz)

    #Izlaz iz Resneta u ASPP
    image_features = base_model.get_layer('conv4_block6_out').output
```

```
x_en = ASPP(image_features, n_filters)
x_en = UpSampling2D((4, 4), interpolation="bilinear")(x_en)
#Lokalne znacajke iz Resneta
x_de = base_model.get_layer('conv2_block2_out').output
x_de = Conv2D(filters=8, kernel_size=1, padding='same',
              use_bias=False)(x_de)
x_de = BatchNormalization()(x_de)
x_de = Activation('relu')(x_de)
x_de = Dropout(dropout)(x_de)

x = concatenate([x_en, x_de])

x = Conv2D(filters=n_filters, kernel_size=3, padding='same',
           use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(dropout)(x)

x = Conv2D(filters=n_filters, kernel_size=3, padding='same',
           use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((4, 4), interpolation="bilinear")(x)
x = Dropout(dropout)(x)

izlaz = Conv2D(1, (1, 1), activation='sigmoid')(x)

model = Model(inputs=ulaz, outputs=izlaz)
return model
```

B Ostatak korištenog Python koda

```
# -*- coding: utf-8 -*-

import os
import random
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")

from tqdm import tqdm
from skimage.transform import resize
from sklearn.model_selection import train_test_split

import tensorflow as tf

from keras.models import Model
from keras.layers import Input, BatchNormalization, Activation, Dropout
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.layers import concatenate
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import img_to_array, load_img, save_img

from datetime import datetime

from tensorflow.keras.layers import UpSampling2D
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.applications import ResNet50
from layyers import MaxPoolingWithArgmax2D

import tensorflow_addons as tfa



os.chdir(r'C:\Users\josip\Desktop\faks\DIPLOMSKI\projekt\pajtn')
cwd = os.getcwd()
```

```
print(cwd)

#Obrada skupa podataka
ids = next(os.walk("T1_images"))[2]
print("Broj slika = ", len(ids))
ids2 = next(os.walk("T1_masks"))[2]
print("Broj maski = ", len(ids2))

#Parametri velicine ulaznih slika
img_width = 128
img_height = 128

X = np.zeros((len(ids), img_height, img_width, 1), dtype=np.float32)
y = np.zeros((len(ids), img_height, img_width, 1), dtype=np.float32)


for n, id_ in tqdm(enumerate(ids), total=len(ids)):
   #Slike
    img = load_img("T1_images/"+id_, color_mode = "grayscale")
    X_img = img_to_array(img)
    X_img = resize(X_img, (img_height, img_width, 1),
                   mode = 'constant', preserve_range = True)
    X_img_max = X_img.max(axis=(0, 1))
    X_img_min = X_img.min(axis=(0, 1))
    XRange = X_img_max - X_img_min
    #Maske
    maska = img_to_array(load_img("T1_masks/"+id_, color_mode = "grayscale"))
    maska = resize(maska, (img_height, img_width, 1),
                   mode = 'constant', preserve_range = True)
    maska_max = maska.max(axis=(0,1))
    maska_min = maska.min(axis=(0,1))
    maskaRange = maska_max - maska_min
    #Normalizacija
    X[n] = (X_img-X_img_min)/XRange
    y[n] = (maska-maska_min)/maskaRange


#za deeplab
X = np.zeros((len(ids), img_height, img_width, 3), dtype=np.float32)
y = np.zeros((len(ids), img_height, img_width, 1), dtype=np.float32)
for n, id_ in tqdm(enumerate(ids), total=len(ids)):
```

```
    #Slike
     img = load_img("UKUPNO_images/"+id_, color_mode = "grayscale")
     x_img = img_to_array(img)
     x_img = resize(x_img, (128, 128, 1), mode = 'constant', preserve_range = Tru
     x_img = np.stack([x_img.squeeze()] * 3, axis=-1)  # Convert to 3 channels
     x_img_max = x_img.max(axis=(0, 1))
     x_img_min = x_img.min(axis=(0, 1))
     xRange = x_img_max - x_img_min
     #Maske
     mask = img_to_array(load_img("UKUPNO_masks/"+id_, color_mode = "grayscale"))
     mask = resize(mask, (128, 128, 1), mode = 'constant', preserve_range = True)
     mask_max = mask.max(axis=(0,1))
     mask_min = mask.min(axis=(0,1))
     maskRange = mask_max - mask_min
     #Normalizacija
     X[n] = (x_img-x_img_min)/xRange
     y[n] = (mask-mask_min)/maskRange

X = np.nan_to_num(X)
y = np.nan_to_num(y)

#Train-valid
X_train,X_valid,y_train,y_valid=train_test_split(X, y, test_size=0.1,
                                                 random_state=42)
#Test
X_train,X_test,y_train,y_test = train_test_split(X_train, y_train,
                                                 test_size=0.11111111)



#Prikaz nasumicne slike i  odgovarajuce maske
ix = random.randint(0, len(X_train))
has_mask = y_train[ix].max() > 0

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (20, 15))

ax1.imshow(X_train[ix, ..., 0], cmap='gray', interpolation='bilinear')
if has_mask:
    # Kontura koja odvaja lezije
    ax1.contour(y_train[ix].squeeze(), colors = 'red', linewidths=0.7,
                levels = [0.5])
ax1.set_title('MRI␣slice')
```

```
ax2.imshow(y_train[ix].squeeze(), cmap='gray', interpolation='bilinear')
ax2.set_title('Lezije')


#DCS
smooth = 1e-15
def dice_coef(y_true, y_pred):
    y_true = tf.keras.layers.Flatten()(y_true)
    y_pred = tf.keras.layers.Flatten()(y_pred)
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2. * intersection + smooth) / (tf.reduce_sum(y_true)
                                           + tf.reduce_sum(y_pred)
                                           + smooth)

def dice_coef_loss(y_true, y_pred):
    return 1 - dice_coef(y_true, y_pred)

#Compile
ulaz_oblik= (128,128,1)
model = segnet(ulaz_oblik, n_filters=16, dropout=0.05,k_size=3)
#model = unet(ulaz_oblik, n_filters=32, dropout=0.05,k_size=3)
#model = DeepLabV3Plus(ulaz_oblik, n_filters=32, dropout=0.1)
model.compile(optimizer=Adam(), loss=dice_coef_loss,
              metrics=["accuracy"])
model.summary()

#Pracenje i kriteriji zaustavljanja
callbacks = [
    EarlyStopping(patience=20, verbose=1),
    ReduceLROnPlateau(factor=0.1,patience=5,min_lr=0.00001,verbose=1),
    ModelCheckpoint('model-segnet.h5', verbose=1, save_best_only=True,
                    save_weights_only=True)
]

#Treniranje
train_start_time = datetime.now()

results = model.fit(X_train, y_train, batch_size=32, epochs=100,\
                    callbacks=callbacks,\
                    validation_data=(X_valid, y_valid))
```

```
train_end_time = datetime.now()

print('Trajanje treninga:', train_end_time-train_start_time)


#Krivulja ucenja
plt.figure(figsize=(8, 8))
plt.title("Learning curve")
plt.plot(results.history["loss"], label="loss")
plt.plot(results.history["val_loss"], label="val_loss")
plt.plot( np.argmin(results.history["val_loss"]),
         np.min(results.history["val_loss"]),
         marker="x", color="r", label="Najbolji model")
plt.xlabel("Epohe")
plt.ylabel("Funkcija gubitka")
plt.legend();

#Ucitavanje najboljeg modela
model.load_weights('model-segnet-T1-16f.h5')


#Evaluacija modela
model.evaluate(X_train, y_train, verbose=1)
model.evaluate(X_valid, y_valid, verbose=1)
model.evaluate(X_test, y_test, verbose=1)


#Predikcije za train, val, test
train_start_time1 = datetime.now()

preds_train = model.predict(X_train, verbose=1)

train_end_time1 = datetime.now()

print('Trajanje predikcije:', train_end_time1-train_start_time1)


train_start_time2 = datetime.now()

preds_val = model.predict(X_valid, verbose=1)
```

```
train_end_time2 = datetime.now()

print('Trajanje predikcije:', train_end_time2-train_start_time2)

train_start_time3 = datetime.now()

preds_test = model.predict(X_test, verbose=1)

train_end_time3 = datetime.now()

print('Trajanje predikcije:', train_end_time3-train_start_time3)


#Threshold predictions - tesko namjestiti pogodnu vrijednost
#obzirom da su sve velicine u slikama malene
preds_train_t = (preds_train > 0.1).astype(np.uint8)
preds_val_t = (preds_val > 0.1).astype(np.uint8)
preds_test_t =(preds_test>0.1).astype(np.uint8)


#Prikaz rezultata
def plot_sample(X, y, preds, binary_preds, ix=None):
    """Function to plot the results"""
    if ix is None:
        ix = random.randint(0, len(X))

    has_mask = y[ix].max() > 0

    fig, ax = plt.subplots(1, 3, figsize=(20, 10))
    ax[0].imshow(X[ix, ..., 0], cmap='gray')
    if has_mask:
        ax[0].contour(y[ix].squeeze(), colors='red', levels=[0.5],
                      linewidths = 0.7)
    ax[0].set_title('MRI slice')
    ax[0].set_axis_off()
    ax[0].grid(False)


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

```
    ax[1].set_axis_off()
    ax[1].grid(False)


    ax[2].imshow(preds[ix].squeeze(), vmin=0, vmax=1, cmap='gray')
    #if has_mask:
        #ax[2].contour(y[ix].squeeze(), colors='red',
                      #levels=[0.5],linewidths = 0.7)
    ax[2].set_title('Segmentirane lezije')
    ax[2].set_axis_off()
    ax[2].grid(False)

i=0
for i in range(len(X_test)):
    plot_sample(X_test, y_test, preds_test, preds_test_t, ix=i)

#Rezultati
DSCtest = dice_coef(y_test, preds_test)
DSCval = dice_coef(y_valid, preds_val)
DSCtrain = dice_coef(y_train, preds_train)
print('DSCtest:', DSCtest)
print('DSCval:', DSCval)
print('DSCtrain:', DSCtrain)

#Augmentacija dataseta
SEED = 7
#Generatori za slike i maske
image_data_generator = ImageDataGenerator(
    rotation_range=90,
    zoom_range=0.2,
    horizontal_flip=True,
    shear_range=0.2,
    fill_mode='constant')

mask_data_generator = ImageDataGenerator(
    rotation_range=90,
    zoom_range=0.2,
    horizontal_flip=True,
    shear_range=0.2,
    fill_mode='constant')
```

```
image_mask_generator = zip(
    image_data_generator.flow(X, batch_size=1, seed=SEED,
                              shuffle = False),
    mask_data_generator.flow(y, batch_size=1, seed=SEED,
                             shuffle = False))

i = 0
for i, (augmented_images, augmented_masks) in tqdm(
        enumerate(image_mask_generator), total=len(ids)):
    if i >= len(ids):
        break

    img_name = ids[i]

    img_name_aug = img_name.split('.')[0] + '_augg.png'

    img_path = os.path.join("aug_n_im", img_name_aug)
    save_img(img_path, augmented_images[0])

    mask_path = os.path.join("aug_n_mask", img_name_aug)
    save_img(mask_path, augmented_masks[0])

#GRID SEARCH
filters = [8, 16, 32]

batch = [32, 64, 128]

parameters = []
for i in filters:
    for j in batch:
        parameters.append((i, j))

print("Moguce kombinacije parametara : ", parameters)

print(parameters[2][0])
print(parameters[2][1])

print(len(parameters))

callbacks = [
    EarlyStopping(patience=20, verbose=1),
```

```
    ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.00001,
                      verbose=1)]

input_img = Input((img_height, img_width, 1))
dsc_values = []
for k in range (len(parameters)):
    model = unet(input_img, n_filters=parameters[k][0], dropout=0.05,
                 batchnorm=True)
    model.compile(optimizer=Adam(), loss=dice_coef_loss,
                  metrics=["accuracy"])
    results = model.fit(X_train, y_train, batch_size=parameters[k][1],
                        epochs=10, callbacks=callbacks,
                        validation_data=(X_valid, y_valid))

    plt.figure(figsize=(8, 8))
    plt.title(f"Parametri: {parameters[k][0]}, {parameters[k][1]}")
    plt.plot(results.history["loss"], label="loss")
    plt.plot(results.history["val_loss"], label="val_loss")
    plt.plot( np.argmin(results.history["val_loss"]),
              np.min(results.history["val_loss"]),
              marker="x", color="r", label="Najbolji model")
    plt.xlabel("Epohe")
    plt.ylabel("Funkcija gubitka")
    plt.legend();
    plt.savefig(f"figure_grid{k}.png")

    preds_val = model.predict(X_valid, verbose=1)
    DSCval = dice_coef(y_valid, preds_val)
    dsc_values.append(DSCval)


numpy_values = [tensor.numpy() for tensor in dsc_values]
print(numpy_values)

##Nii2png
import numpy, shutil, os, nibabel
import argparse
import imageio


base_path=os.path.abspath(os.path.dirname(__file__))
```

```
parser = argparse.ArgumentParser(description=
                                  'Arguments␣for␣input␣and␣output␣files')
parser.add_argument('--input_path', type=str,
                    default = base_path, help='Path␣of␣the␣input␣files')
parser.add_argument('--rotation_angle', type=int,
                    default = 90,
                    help='Rotation␣degree,␣default␣value␣is␣90Â°')
args = parser.parse_args()
input_path = args.input_path
rotation_angle = args.rotation_angle
source_files = os.listdir(input_path)
slice_counter = 0

source_ids = [files[0:8] for files in source_files if
              files.endswith('.nii')]

sample_ids = list(set(source_ids))

for file in sample_ids:
    fname = os.path.basename(file)
    image_array = nibabel.load(fname+'.nii').get_data()
    print(len(image_array.shape))
    # set destination folder
    if not os.path.exists(base_path+'/'+fname):
        os.makedirs(base_path+'/'+fname)
        print("Created␣ouput␣directory:␣" + base_path+'/'+fname)


    if len(image_array.shape) == 3:
        nx, ny, nz = image_array.shape
        total_slices = image_array.shape[2]
        # iterate through slices
        for current_slice in range(0, total_slices):
            # alternate slices
            if (slice_counter % 1) == 0:
                # rotate or no rotate
                if rotation_angle == 90:
                    data = numpy.rot90(image_array[:, :, current_slice])
                elif rotation_angle == 180:
                    data = numpy.rot90(
                        numpy.rot90(image_array[:, :, current_slice]))
```

```
            elif rotation_angle == 270:
                data = numpy.rot90(numpy.rot90(numpy.rot90(
                    image_array[:, :, current_slice])))

            if (slice_counter % 1) == 0:
                print('Saving␣image...')
                image_name = fname[:-4] + "_z" + "{:0>3}".format(
                    str(current_slice+1))+ ".png"
                imageio.imwrite(image_name, data)
                print('Saved.')
                #move images to folder
                print('Moving␣image...')
                src = image_name
                shutil.move(src, base_path+'/'+fname)
                slice_counter += 1
                print('Moved.')
                print('Finished␣converting␣images')
elif len(image_array.shape) == 4:
    nx, ny, nz, nw = image_array.shape
    total_volumes = image_array.shape[3]
    total_slices = image_array.shape[2]
    for current_volume in range(0, total_volumes):
        # iterate through slices
        for current_slice in range(0, total_slices):
            if (slice_counter % 1) == 0:
                if rotation_angle == 90:
                    data = numpy.rot90(image_array[
                        :, :, current_slice, current_volume])
                elif rotation_angle == 180:
                    data = numpy.rot90(numpy.rot90(image_array[
                        :, :, current_slice, current_volume]))
                elif rotation_angle == 270:
                    data = numpy.rot90(numpy.rot90(
                        numpy.rot90(image_array[
                            :, :, current_slice, current_volume])))


                if (slice_counter % 1) == 0:
                    print('Saving␣image...')
                    image_name = fname[:-4] + "_z"
                    + "{:0>3}".format(str(current_slice+1))+ ".png"
```

```
                        imageio.imwrite(image_name, data)
                        print('Saved.')
                        #move images to folder
                        print('Moving image...')
                        src = image_name
                        shutil.move(src, base_path+'/'+fname)
                        slice_counter += 1
                        print('Moved.')
                        print('Finished converting images')

#Maxpool s indeksima
from keras import backend as K
from keras.layers import Layer


class MaxPoolingWithArgmax2D(Layer):
    def __init__(self, pool_size=(2, 2), strides=(2, 2),
                 padding="same",
                 **kwargs):
        super(MaxPoolingWithArgmax2D, self).__init__(**kwargs)
        self.padding = padding
        self.pool_size = pool_size
        self.strides = strides

    def call(self, inputs, **kwargs):
        padding = self.padding
        pool_size = self.pool_size
        strides = self.strides
        if K.backend() == "tensorflow":
            ksize = [1, pool_size[0], pool_size[1], 1]
            padding = padding.upper()
            strides = [1, strides[0], strides[1], 1]
            output, argmax = K.tf.nn.max_pool_with_argmax(
                inputs, ksize=ksize, strides=strides, padding=padding
            )
        else:
            errmsg = "{} backend is not supported for layer {}".format(
                K.backend(), type(self).__name__
            )
            raise NotImplementedError(errmsg)
        argmax = K.cast(argmax, K.floatx())
```

```
    return [output, argmax]

def compute_output_shape(self, input_shape):
    ratio = (1, 2, 2, 1)
    output_shape = [
        dim // ratio[idx] if dim is not None else None
        for idx, dim in enumerate(input_shape)
    ]
    output_shape = tuple(output_shape)
    return [output_shape, output_shape]

def compute_mask(self, inputs, mask=None):
    return 2 * [None]
```