SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 2005

# Semantička segmentacija osoba konvolucijskim modelima

Lucija Ivković

Zagreb, rujan 2019.

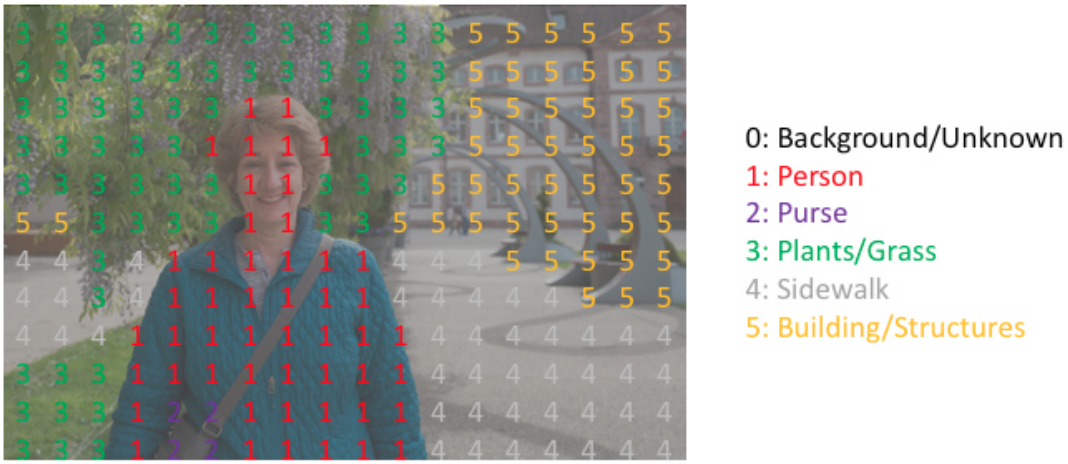
*Zahvaljujem se svom mentoru prof. dr. sc. Siniši Šegviću za svu pomoć koju mi je pružio tijekom pisanja ovog rada.*

# SADRŽAJ

# 1. Uvod

Semantička segmentacija u računalnom vidu je problem koji ima za cilj dodijeliti klasu svakom pikselu na slici. Duboka konvolucijska mreža za segmentaciju je stoga nešto drugačija od tipične mreže za klasifikaciju. Prije svega, za pojedinu ulaznu sliku izlaz mreže nije one-hot encoding jedinstvene odabrane klase, već tzv. segmentacijska mapa, koja ima jednaku rezoluciju kao i ulazna slika, a svaki piksel ima cjelobrojnu vrijednost koja predstavlja klasu kojoj piksel pripada (slika 1.1).

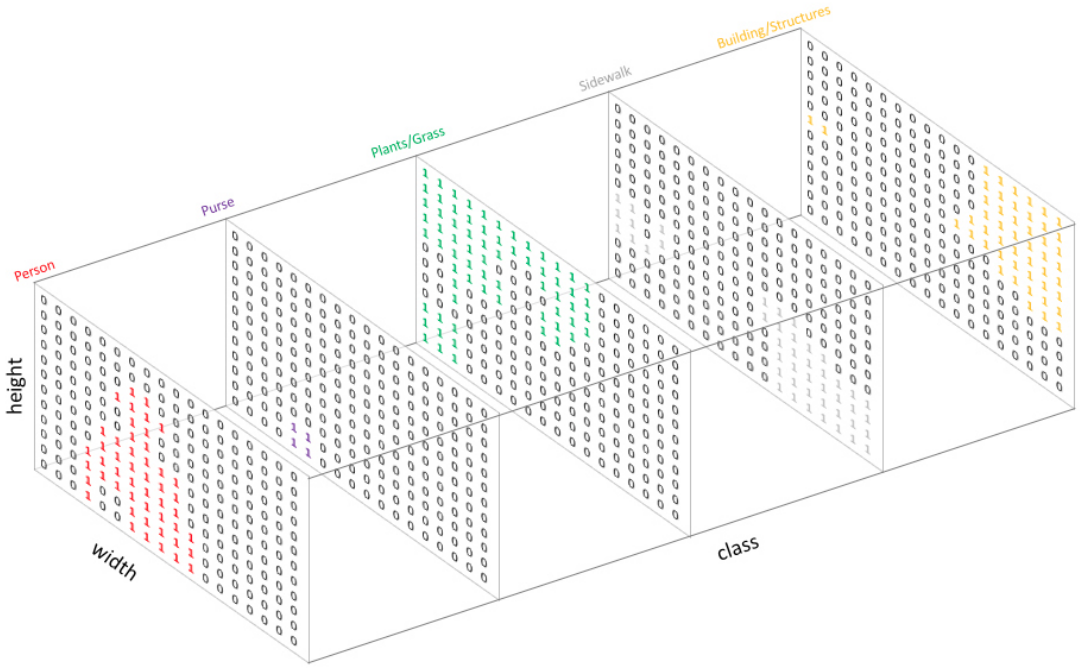
**Slika 1.1:** Primjer fotografije i odgovarajuće segmentacijske mape u vrlo niskoj rezoluciji radi jasnije vizualizacije (u stvarnosti, mapa i slika trebaju biti jednake rezolucije) [1].




Dodatna klasa može se uvesti za nepoznate piksele, tj. piksele za koje nije sasvim jasno kojoj klasi pripadaju. Ova klasa ne ulazi u izračun gubitka prilikom učenja mreže. Segmentacijsku mapu, slično kao i obične klasifikacijske oznake, potrebno je transformirati u one-hot encoding (slika 1.2), čime se kategorički oblik izlaza mreže transformira u oblik generalno pogodniji za algoritme strojnog učenja.

Nadalje, arhitektura mreže za segmentaciju sastoji se od dva dijela. Prvi dio odgovara klasičnoj konvolucijskoj mreži i obavlja ekstrakciju značajki iz ulazne slike.

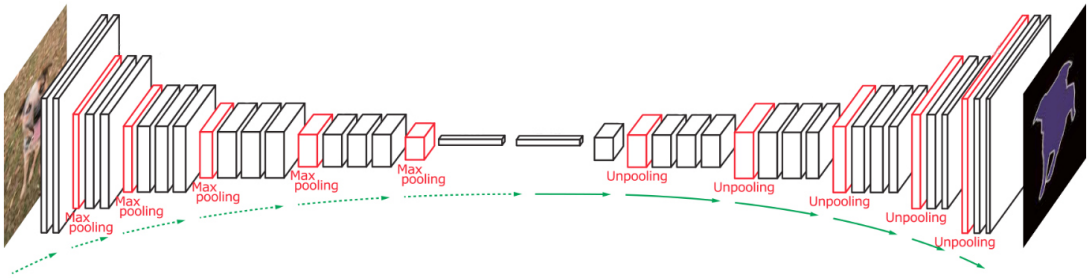
**Slika 1.2:** One-hot encoding segmentacijske mape sa slike 1.1 [1].




Drugi dio ima svrhu naduzorkovanja tih značajki da bi se na izlazu dobila odgovarajuća rezolucija mape, tj. veličina jednaka ulaznoj slici. Slika 1.3 prikazuje opisanu arhitekturu mreže za semantičku segmentaciju. Ovaj pristup rješavanja problema semantičke segmentacije baziran je na klasifikacijskim, konvolucijskim modelima. Razvijena su i druga rješenja za ovaj problem računalnog vida, primjerice ona po uzoru na algoritme konsenzusnih susjedstava (eng. neighbourhood consensus) u kojima se uspoređivanjem parova slika nalaze ekvivalentnosti na razini piksela. Jedno takvo rješenje predstavljeno je u [2], gdje se tzv. mreža konsenzusnih značajki (eng. consensus feature network) uči raspoznavati značajke koje se ne razlikuju među pikselima koji pripadaju istoj instanci ili kategoriji klase. Tipično su takve značajke u jednoj instanci translatirane ili rotirane, u odnosu na drugu. Nedostatak konvolucijskih modela u odnosu na ovakav pristup je u tome što oni pretežno uče značajke na razini slike, što može dovesti do dominacije velikih objekata ili objekata u fokusu u predikciji mreže.

U ovom radu problem jest segmentacija ljudi od pozadine, u svrhu uklanjanja pozadine sa slike. Dakle, problem se sastoji od dvije klase, čovjek i pozadina. Za ovaj zadatak koristila sam dvije različite arhitekture duboke segmentacijske mreže, te sam provela analizu i usporedbu njihovog rada. Prva arhitektura je ljestvičasto povezani DenseNet [4], a druga potpuni konvolucijski DenseNet, tzv. Tiramisu [5]. Za treniranje navedenih mreža koristila sam dva skupa slika za učenje. Cityscapes, [6] jedan

**Slika 1.3:** Arhitektura duboke konvolucijske mreže za zadatak semantičke segmentacije [3].



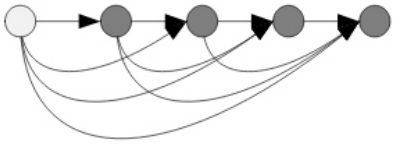

od najpopularnijih skupova podataka u području računalnog vida, upotrijebljen je za učenje isključivo u svrhu reprodukcije već postignutih rezultata, što je poslužilo za provjeru ispravnosti implementacije modela. Osim navedenih DenseNet modela, na Cityscapes skupu treniran je i jednostavni model kojeg koristimo kao osnovicu usporedbe (eng. baseline), Taj model se sastoji od DenseNet mreže za klasifikaciju, na koju se nadovezuje obična bilinearna interpolacija kao tehnika naduzorkovanja mape značajki do rezolucije ulazne slike. U svrhu rješenja zadanog problema primijenila sam skup LIP (Look Into Person), [7] koji se sastoji od oko 40000 označenih slika za učenje i validaciju modela, te 10000 neoznačenih za testiranje naučenog modela.

# 2. Konvolucijski modeli za semantičku segmentaciju

U ovom poglavlju predstavljena su tri modela za semantičku segmentaciju slika, svi bazirani na dubokoj gustoj konvolucijskoj arhitekturi DenseNet. Prvi, jednostavni model je najlošiji u predikciji, no služi samo kao referentni model ispod kojeg kvaliteta ne bi smjela padati. Ljestvičasti Densenet ima veću razinu složenosti i bolje rezultate od jednostavnog modela, te je vremenski i memorijski efikasniji od Tiramisu DenseNeta, koji ipak ima najveći kapacitet od sva tri navedena modela.

## 2.1. Gusto povezani modeli za klasifikaciju slika

Duboka gusta konvolucijska mreža DenseNet sastoji se od gusto povezanih konvolucijskih blokova (eng. dense blocks), i prijelaznih blokova koji obavljaju poduzorkovanje (eng. downsampling blocks). Gusti blok naziva se tako zbog guste povezanosti slojeva unutar svoje strukture. Jedan sloj unutar bloka je definiran kao kompozitna funkcija triju uzastopnih operacija; prvo se obavlja normalizacija po grupi (eng. batch normalization), zatim slijedi ReLU aktivacija i konačno, 3x3 konvolucija kojoj opcionalno prethodi 1x1 konvolucija u funkciji smanjenja broja ulaznih mapa, a time i računskog ubrzanja. Sloj koji ima ovu dodatnu opciju zovemo sloj s uskim grlom (eng. bottleneck). Broj izlaznih mapa značajki u konvoluciji, označimo ga s k, hiperparametar je gustog bloka i naziva se stopa rasta (eng. growth rate). Svaki sljedeći sloj prima na ulaz konkatenaciju svih prethodnih slojeva po vanjskoj dimenziji (slika 2.1), stoga ima uvid u sve već ekstrahirane značajke, te pridonosi cijelom modelu s k novih. Intuicija nas može navesti na krivi zaključak da bi zbog ovog svojstva mreža mogla imati znatno



**Slika 2.1:** Gusta povezanost slojeva unutar gustog bloka [8].

više parametara od klasičnih konvolucijskih mreža, no istina je da ovakva povezanost potiče ponovnu upotrebu već naučenih značajki, umjesto da ponovno uči redundantne značajke. Također, za solidan kapacitet modela hiperparametar k ne treba biti velik (uobičajeno se uzima 16 ili 32). Osim prethodno navedenih prednosti, DenseNet arhitektura poboljšava protok informacija i gradijenata kroz slojeve, što za sobom povlači veću efikasnost učenja parametara. Svaki sloj ima izravni pristup gradijentima iz funkcije gubitka i iz ulazne slike, čime se ostvaruje implicitni duboki nadzor u mreži.

Između gustih blokova nalaze se prijelazni blokovi, koji smanjuju prostorne dimenzije mapa značajki, tipično nekom tehnikom sažimanja (eng. pooling). Slično kao i u gustom bloku, sloj ovoga bloka sastoji se od normalizacije po grupi, ReLU aktivacije, 1x1 konvolucije s brojem izlaznih značajki dva puta manjim od ulaznih, te na kraju sažimanja prosječnom vrijednosti veličine 2x2. Dakle, osnovna DenseNet arhitektura za običnu klasifikaciju sastoji se od gustih i prijelaznih blokova u alternirajućem redoslijedu. [8]

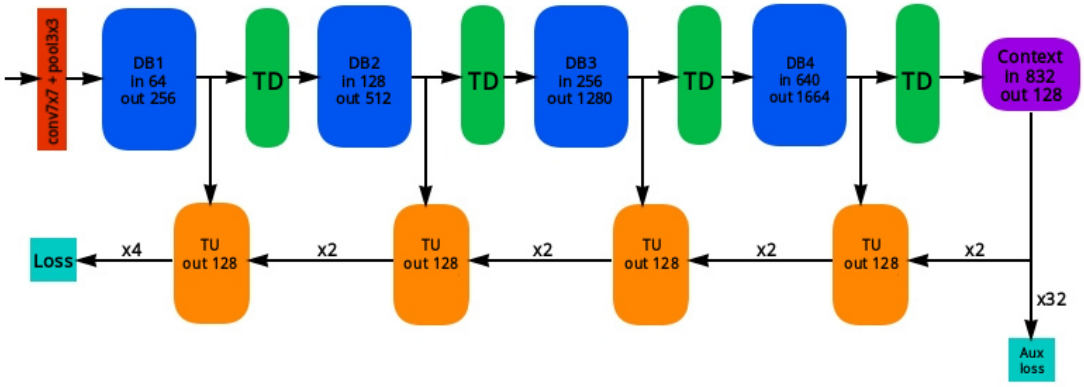
## 2.2. Osnovica usporedbe za semantičku segmentaciju

Kao osnovica usporedbe koristila sam model DenseNet121. Ova mreža ima 4 gusta bloka koji se sastoje od redom, 6, 12, 24 i 16 slojeva s uskim grlom, pri čemu je stopa rasta jednaka 32. Nakon svakog gustog bloka slijedi prijelazni blok, a prvom bloku prethodi inicijalna konvolucija s jezgrom veličine 7x7, korakom 2 i 64 izlazne mape značajki, te sažimanje po maksimumu veličine 3x3 uz korak veličine 2. Za problem segmentacije, u kojem izlaz mreže treba biti jednakih dimenzija kao ulaz, potrebno je povećati rezoluciju mapa značajki iz posljednjeg bloka na razinu ulazne rezolucije. U ovom, jednostavnom modelu uvećanje rezolucije provodi se bilinearnom interpolacijom uz faktor uvećanja jednak 32. Bilinearna interpolacija je jednostavna metoda koja se tipično koristi za uvećavanje rezolucije slika, no kako nema mogućnost treniranja, time ni ne doprinosi neuronskoj mreži u smislu složenosti i kapaciteta učenja.

## 2.3. Ljestvičasto povezana DenseNet arhitektura

Arhitektura ljestvičasto povezanog DenseNet modela preuzeta iz rada [4] prikazana je na slici 2.2. Preciznije, slika prikazuje inačicu DenseNet169 spomenute arhitekture. Mreža je podijeljena u dvije podatkovne putanje. Prva, na slici označena DB i TD blokovima, obavlja ekstrakciju značajki gustim blokovima (DB, eng. dense blocks)

**Slika 2.2:** Ilustracija arhitekture ljestvičasto povezanog DenseNet169 modela prema [4]. Gusti blokovi sadrže, redom 6, 12, 32, 32 sloja, uz stopu rasta jednaku 32. Oznake na strelicama prikazuju faktor prostornog uvećanja mapi značajki u pojedinim slojevima mreže. Na slici je također naznačen broj ulaznih i izlaznih značajki za guste te za kontekstualni blok, a za blokove naduzorkovanja samo broj izlaznih značajki (broj ulaznih je 2 puta veći zbog konkatenacije prethodnog sloja s preskočnom vezom).




i poduzorkovanje istih prijelaznim blokovima (TD, eng. transition down), opisanim u potpoglavlju 2.1. Prijelazni blokovi smanjuju rezoluciju značajki za faktor dva, dok inicijalna konvolucija zajedno sa slojem sažimanja (označeno crvenim pravokutnikom) na izlazu daje 4 puta manju mapu u odnosu na ulaznu sliku. Stoga je, nakon posljednjeg TD bloka mapa po prostornim dimenzijama 32 puta manja u odnosu na sliku s ulaza mreže. Nadalje slijedi tzv. kontekstualni blok (na slici označen ljubičastom bojom), koji se sastoji od jednog bottleneck sloja, pri čemu su filteri u 3x3 konvoluciji rastegnuti za faktor dilatacije 2 u svrhu ekstrakcije značajki širokog konteksta na slici. Kontekstualni blok služi tome da se iz slike izvuku okvirne informacije bez visoke razine detalja. Ovdje valja napomenuti kako su autori originalnog rada [4] u ulozi kontekstualnog bloka koristili prostorni, piramidalni modul za sažimanje (eng. spatial pyramid pooling module, SPP).

Druga putanja mrežne arhitekture sastavljena je od TU blokova sa slike 2.2. TU blokovi obavljaju postepeno naduzorkovanje ekstrahiranih mapa značajki. U ovoj putanji se primjenjuje ljestvičasta arhitektura preskočnih veza, koje su prikazane okomitim strelicama na slici, te povezuju mape značajki na istoj rezoluciji iz prve putanje s prethodno spomenutim značajkama širokog konteksta. Povezivanje se obavlja na sljedeći način: prvo se mape iz prve putanje (izlazi iz DB blokova) preslikaju 1x1 konvolucijom u manju dimenziju tako da oba ulaza u TU blok imaju jednaki broj značajki

(u ovom slučaju 128, pošto je to broj značajki na izlazu svakog TU bloka). Zatim se izlaz iz prethodnog TU bloka bilinearnom interpolacijom poveća za faktor 2 da oba ulaza imaju jednake prostorne dimenzije. Tada se oba transformirana ulaza konkateniraju (što je još jedan slučaj odstupanja od izvornog rada, u kojem su autori navedene ulaze zbrajali), nakon čega slijedi 3x3 konvolucija koja obavlja ekstrakciju značajki iz svih dosad skupljenih informacija na trenutnoj rezoluciji. Efektivno, to znači da u navedenoj putanji naduzorkovanja mreža ima uvid i u mape značajki ekstrahirane iz nižih slojeva mreže koje predstavljaju sitnu razinu detalja na slici (primjerice, rubovi i konture, manji dijelovi složenog objekta), i u mape značajki širokog konteksta koje obuhvaćaju krupne informacije o slici promatranoj u cijelosti. Na kraju zadnjeg TU bloka ponovno se obavlja bilinearna interpolacija da se zadnja skupina mapa značajki, koja je ovdje 4 puta manja u rezoluciji od izvorne, podigne na izvornu rezoluciju, te se obavi još jedna 1x1 konvolucija kojom se ekstrahirane mape značajki projiciraju u broj klasa zadanog problema. Iz tog izlaza se na kraju računa funkcija gubitka koja sačinjava 70% ukupnog gubitka. Preostalih 30% funkcije gubitka, tzv. pomoćni gubitak (na slici 2.2 označen s Aux loss) računa se direktno iz izlaza kontekstualnog bloka, naknadno projiciranog u broj klasa te uvećanog bilinearnom interpolacijom do originalne rezolucije za faktor 32, slično kao i kod prethodno spomenutog, glavnog gubitka. Pitanje koje se ovdje postavlja jest kojim redoslijedom postaviti ove dvije operacije, te je li redoslijed uopće bitan za konačan rezultat. U kontekstu vremenskog izvođenja redoslijed jest bitan. Slučaj u kojem se konvolucija izvodi na većoj rezoluciji, dakle nakon bilinearne interpolacije, je svakako sporiji. Stoga sam u potpoglavlju 2.3.1 provela matematički dokaz komutacije ovih dviju operacija, kojom sam pokazala da je za krajnji rezultat, redoslijed doista nebitan.

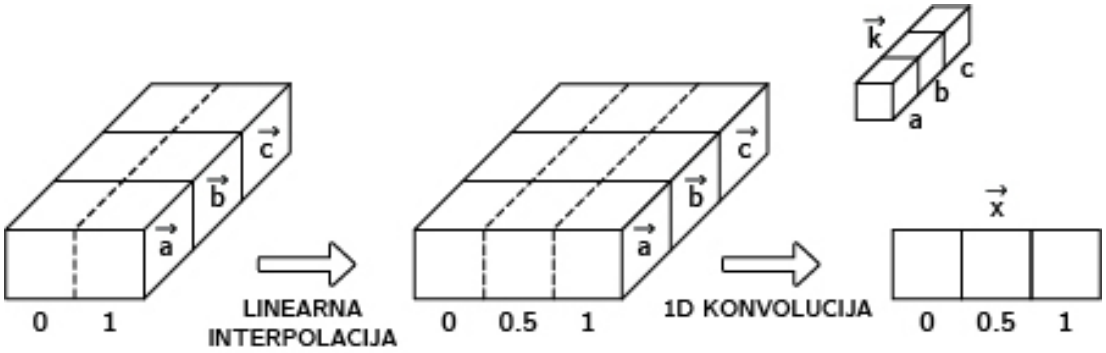
Opisana ljestvičasta arhitektura duboke neuronske mreže omogućuje da se u postupku učenja istovremeno uzimaju u obzir informacije visoke i niske razine iz slike. Kako se za naduzorkovanje značajki koristi poprilično jednostavna tehnika bilinearne interpolacije, ušteda na računalnim resursima je značajna u odnosu na složenije tehnike naduzorkovanja poput, primjerice, transponirane konvolucije (slika 2.3).

### 2.3.1. Komutativnost 1x1 konvolucije i bilinearne interpolacije

Dokaz komutativnosti 1x1 konvolucije i bilinearne interpolacije je, radi jednostavnosti, proveden nad prostorno 1D podatkom, nad kojim su primijenjene linearna interpolacija i 1D konvolucija s 3 ulazna i jednim izlaznim kanalom, no isti analogno vrijedi u višedimenzionalnom prostoru nad 2D podacima te s većim brojem izlaznih kanala.

Izvod (2.1), zajedno sa slikom 2.3 prikazuje transformaciju podatka prvo linearnom interpolacijom, a zatim 1x1 konvolucijom.

**Slika 2.3:** Transformacija vektorskog podatka s 3 kanala, označenim s $\vec{a}$, $\vec{b}$ i $\vec{c}$ prvo linearnom interpolacijom u vektor s jednim elementom više, bez promjene u broju kanala, a zatim 1D konvolucijom s 1x1 jezgrom $\vec{k}$ u vektor sa samo jednim kanalom. Konačan rezultat označen je s $\vec{x}$, a indeksi pojedinih elemenata naznačeni su ispod samih elemenata.




*Linearna interpolacija:*

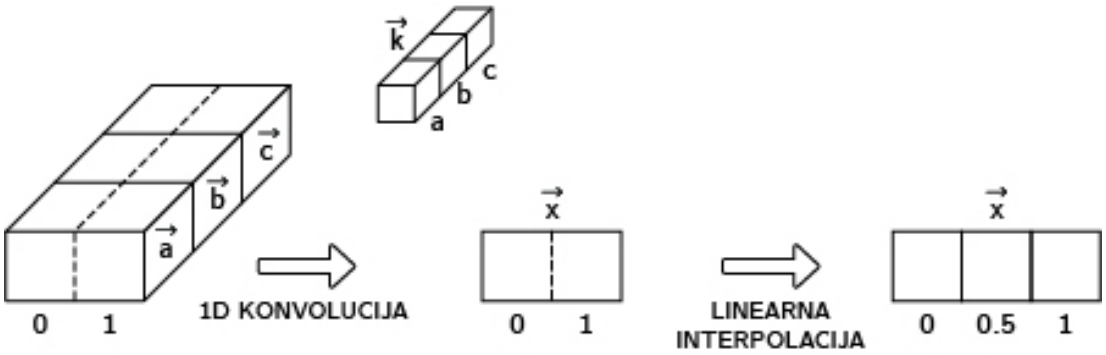
$$a_{0.5} = a_0 + 0.5(a_1 - a_0) = 0.5(a_0 + a_1)$$
$$b_{0.5} = b_0 + 0.5(b_1 - b_0) = 0.5(b_0 + b_1)$$
$$c_{0.5} = c_0 + 0.5(c_1 - c_0) = 0.5(c_0 + c_1)$$

*1x1 konvolucija:*

$$x_0 = a_0 k_a + b_0 k_b + c_0 k_c$$
$$x_1 = a_1 k_a + b_1 k_b + c_1 k_c$$
$$x_{0.5} = a_{0.5} k_a + b_{0.5} k_b + c_{0.5} k_c$$
$$x_{0.5} = 0.5((a_0 + a_1)k_a + (b_0 + b_1)k_b + (c_0 + c_1)k_c) \tag{2.1}$$

Nadalje, izvod (2.2), zajedno sa slikom 2.4 prikazuje transformaciju podatka prvo 1x1 konvolucijom, a zatim linearnom interpolacijom.

**Slika 2.4:** Transformacija vektorskog podatka s 3 kanala, označenim s $\vec{a}$, $\vec{b}$ i $\vec{c}$ prvo 1D konvolucijom s 1x1 jezgrom $\vec{k}$ u vektor sa samo jednim kanalom označen s $\vec{x}$, a zatim linearnom interpolacijom u vektor s jednim elementom više, bez promjene u broju kanala. Indeksi pojedinih elemenata naznačeni su ispod samih elemenata.




*1x1 konvolucija:*

$$x_0 = a_0 k_a + b_0 k_b + c_0 k_c$$
$$x_1 = a_1 k_a + b_1 k_b + c_1 k_c$$

*Linearna interpolacija:*

$$x_{0.5} = 0.5(x_0 + x_1)$$
$$x_{0.5} = 0.5(a_0 k_a + b_0 k_b + c_0 k_c + a_1 k_a + b_1 k_b + c_1 k_c)$$
$$x_{0.5} = 0.5((a_0 + a_1)k_a + (b_0 + b_1)k_b + (c_0 + c_1)k_c) \tag{2.2}$$

Primjećujemo da konačan rezultat $\vec{x}$ u oba slučaja ispadne jednak. S obzirom na ot da su ove dvije operacije komutativne, u modelu se odabire redoslijed izvođenja koji je vremenski i memorijski manje složen, a to je drugi predstavljeni slučaj.

## 2.4. Tiramisu - potpuna konvolucijska DenseNet arhitektura

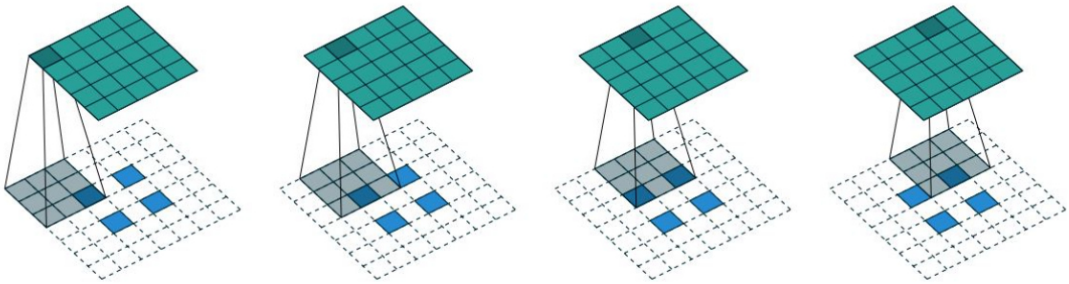
Ilustracija arhitekture potpunog konvolucijskog DenseNet modela Tiramisu prema radu [5] prikazana je na slici 2.6. Slično kao i ljestvičasti DenseNet iz poglavlja 2.3, ova

mreža je organizirana u dvije podatkovne putanje, prvu, prikazanu na lijevoj strani, za ekstrakciju značajki te drugu, na desnoj strani, za naduzorkovanje značajki do izvorne rezolucije. Prva putanja se sastoji od gustih blokova i blokova za poduzorkovanje, koji su konceptualno slični već opisanoj baznoj arhitekturi DenseNeta. Naravno, neke razlike postoje. Za početak, Tiramisu mreža nema slojeve s uskim grlom u gustim blokovima, što ju čini znatno masivnijom i računski zahtjevnijom, zbog čega broj slojeva u gustim blokovima i stopa rasta moraju biti manji u odnosu na klasični DenseNet. Blokovi za poduzorkovanje obavljaju sažimanje maksimalnom vrijednošću, za razliku od ljestvičastog koji obavlja sažimanje po prosjeku. Dodatno, u svakom sloju Tiramisu mreže dodano je nasumično ispuštanje značajki s vjerojatnosti 0.2 (eng. dropout), što je metoda regularizacije tijekom učenja. Temeljna razlika između ove dvije arhitekture je u broju i raspodjeli slojeva u mreži. Dok je Tiramisu strukturno simetričan model, što znači da se njegova druga putanja sastoji od strukturno jednakih gustih blokova kao i prva putanja, ljestvičasti model, s druge strane, ima asimetrično naduzorkovanje. Putanja naduzorkovanja ljestvičastog DenseNeta ima značajno manji kapacitet učenja od prve putanje za ekstrakciju značajki.

Putanja za naduzorkovanje se sastoji od gustih blokova (onih s prethodno istaknutim razlikama u odnosu na blokove ljestvičastog modela), te blokova za naduzorkovanje. Kao tehnika naduzorkovanja korištena je 3x3 transponirana konvolucija uz korak veličine 2. Transponiranom konvolucijom smatramo operaciju obrnutu od obične konvolucije. Ova operacija može se ostvariti koristeći običnu konvoluciju, ako se između elemenata ulaznog 2D podatka dodaju nule (korak transponirane konvolucije postavljen na vrijednost 2 znači dodavanje jedne nule između susjednih elemenata), te se rubovi također popune nulama tako da veličina izlaznog podatka odgovara željenoj veličini. Na takvom popunjenom ulaznom podatku nadalje se obavlja obična konvolucija uz odabranu veličinu jezgre. Prva četiri koraka opisanog postupka ilustrirana su na slici 2.5. Ova tehnika ima mogućnost učenja za razliku od bilinearne interpolacije, dakle još jedan element koji Tiramisu mrežu čini računski zahtjevnijom i složenijom od mreže iz prethodnog poglavlja. Nadalje, i ovdje su uvedene preskočne veze između dviju putanja na mjestima gdje mape značajki imaju jednake prostorne dimenzije. Ove veze stoga, prilikom naduzorkovanja obnavljaju informacije izgubljene tijekom poduzorkovanja, čime se u globalu poboljšava preciznost predikcije na sitnoj razini detalja. Dodatno, kako zbog spomenutih veza u putanji naduzorkovanja broj značajki linearno raste, te uz to još i dolazi do povećanja prostorne rezolucije, gusti blokovi na izlazu ne daju konkatenaciju ulaza i izlaza svih svojih slojeva, već daju samo nove naučene značajke.

Ovisno o broju i veličini gustih blokova, DenseNet Tiramisu može imati različite dubine slojeva. U ovom radu korištena je arhitektura s 57 slojeva, koja se sastoji od po 5 blokova za poduzorkovanje i naduzorkovanje, te 11 gustih blokova s po 4 konvolucijska sloja, od kojih prvih 5 blokova spadaju u putanju za poduzorkovanje te posljednjih 6 u putanju za naduzorkovanje. Izlazni broj mapa značajki iz jednog konvolucijskog sloja određen je stopom rasta, koja iznosi 12. Na početku mreže, prije prvog gustog bloka postavljena je inicijalna 3x3 konvolucija s 48 izlaznih mapa. Na kraju mreže, nakon posljednjeg gustog bloka, postavljena je 1x1 konvolucija koja trenutni broj mapa transformira u broj klasa, tj. računa konačni izlaz koji sudjeluje u izračunu funkcije gubitka.

**Slika 2.5:** Ilustracija transponirane konvolucije uz veličinu jezgre 3x3 i korak 2 [9].



## 2.5. Postupak optimizacije parametara mreža

Postupak treniranja svih mreža je isti i odvija se na sljedeći način. Za funkciju gubitka odabrana je unakrsna entropija, čija je formula za pojedini piksel i određenu klasu dana izrazom (2.3)

$$-(ylog(p) + (1 - y)log(1 - p)) \tag{2.3}$$

pri čemu $y$ predstavlja željeni izlaz (1 ako piksel pripada toj klasi, 0 u suprotnom), a $p$ predikciju modela koja je ograničena na interval $[0, 1]$ i stoga se može interpretirati kao vjerojatnost da piksel pripada upravo toj klasi. Vrijednost izraza (2.3) eksponencijalno raste što se izlaz modela $p$ više udaljava od stvarnog izlaza $y$.

Kao optimizacijski algoritam primjenjuje se Adam algoritam [10] uz propadajuću stopu učenja. Adam je modificirani algoritam gradijentnog spusta koji prati kretanje uprosječenih vrijednosti gradijenata (prvi moment) i kvadrata gradijenata (drugi moment), uz eksponencijalno propadanje starog znanja, te adaptira vrijednost inicijalne

stope učenja za svaki parametar zasebno u skladu s izračunatim prosjecima. Ovo znači da će stopa za parametre s velikim vrijednostima parcijalnih derivacija biti manja (i obrnuto), te da će, ako se tijekom procesa učenja ta vrijednost promijeni, algoritam detektirati promjenu i asimilirati se shodno tome. Dva hiperparametra diktiraju brzinu zaboravljanja uprosječenih starijih vrijednosti prvog i drugog momenta u odnosu na novije. Ovi parametri se tipično postavljaju na vrijednost 0.9 za prvi, te 0.999 za drugi moment.

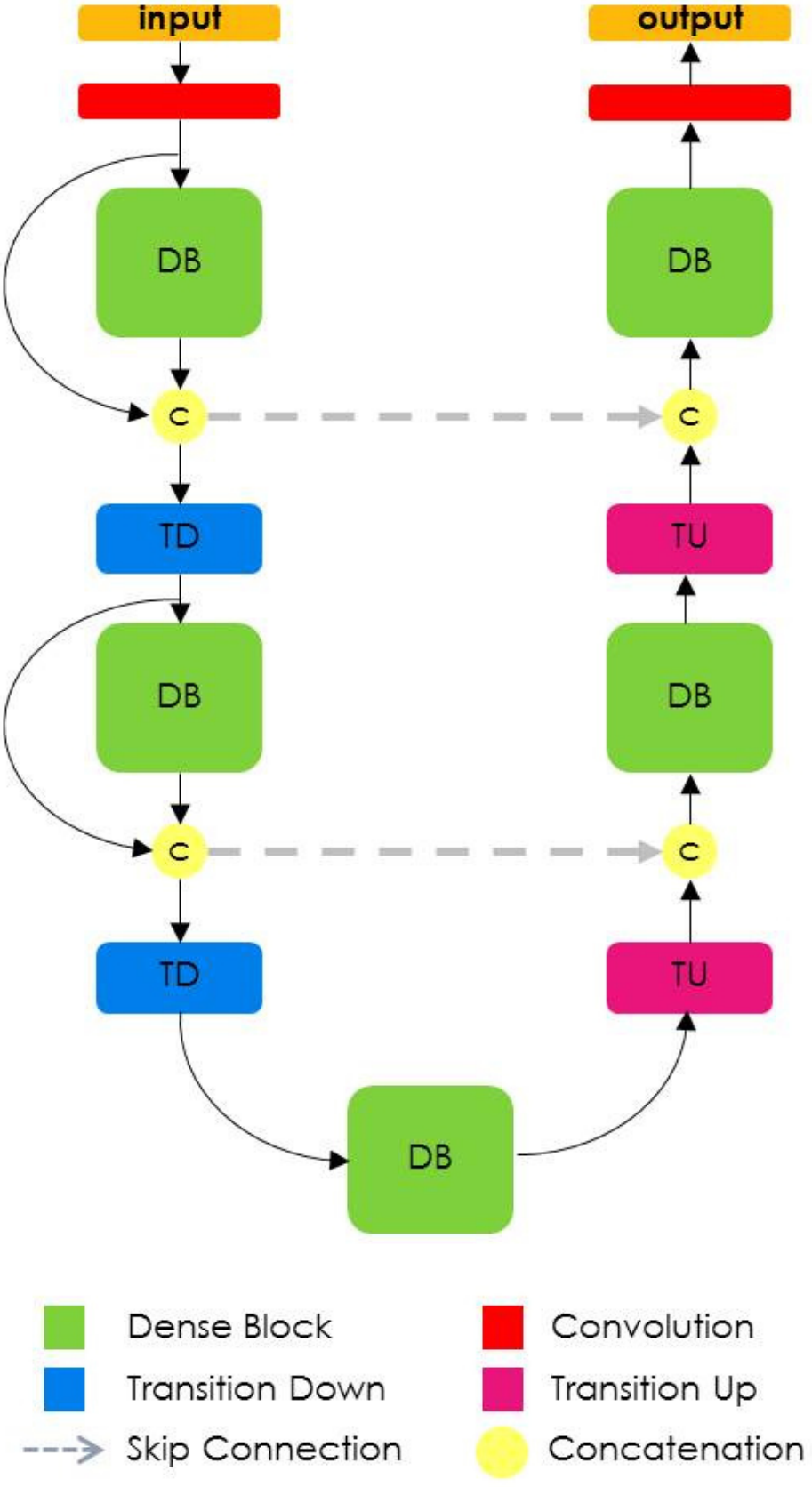
Osim ugrađene adaptacije stope učenja u opisanom optimizacijskom algoritmu, dobra je praksa inicijalnu stopu učenja ne postaviti na fiksnu vrijednost, već implementirati eksponencijalno propadanje kroz iteracije učenja prema formuli (2.4).

$$currentLR = initLR\left(1 - \frac{currentIter}{numIters}\right)^{1.5} \tag{2.4}$$

Pritom oznake $currentLR$ i $initLR$ predstavljaju trenutnu i početnu vrijednost stope učenja, a $currentIter$ i $nIters$ predstavljaju trenutnu iteraciju i ukupan broj iteracija cijelog procesa učenja. Iteracija u ovom kontekstu znači prolazak jedne grupice (eng. mini-batch) slika kroz mrežu, dok je ukupan broj iteracija jednak umnošku ukupnog broja epoha i broja iteracija po jednoj epohi učenja. Na ovaj način se osigurava da se cijeli proces učenja parametara mreže postepeno usporava.

Prve dvije mreže (osnovica usporedbe i ljestvičasti DenseNet) pretrenirane su na skupu ImageNet, dok su parametri u Tiramisu mreži nasumično inicijalizirani. Početna vrijednost stope učenja je $10^{-4}$ za pretrenirane parametre, te 5 puta veća za one koji nemaju ugrađenog predznanja.

**Slika 2.6:** Tiramisu arhitektura potpunog konvolucijskog DenseNet modela [5].



# 3. Implementacija modela i vanjske biblioteke

## 3.1. PyTorch biblioteka

PyTorch [11] je biblioteka za duboko učenje prilagođena programskom jeziku Python. Razvio ga je Facebookov tim za istraživanje. Biblioteka se sastoji od tri osnovna modula. Autograd je modul zadužen za efikasan izračun gradijenata visokodimenzionalnih podataka. Za danu duboku mrežu ovaj modul ispod haube gradi tzv. računski graf, koji je zapravo usmjereni, aciklički graf sastavljen od dvije vrste elemenata; varijabli, predstavljenih čvorovima grafa, i matematičkih operacija, predstavljenih bridovima grafa. Varijable su svi podaci koji teku mrežom, dakle ulazi, izlazi, međurezultati i parametri mreže. Tipično se ulazne podatke i parametre naziva listovima grafa, dok su izlazni podaci korijeni grafa. Matematičke operacije se vrše nad čvorovima, prvo u unaprijednom prolasku od ulaznih prema izlaznim podacima (eng. forward pass), pri čemu se međurezultati spremaju u predmemoriji (eng. cache), da bi zatim u obrnutom, odnosno unatražnom prolasku kroz graf (eng. backward pass) poslužili za izračun potrebnih gradijenata algoritmom propagacije unatrag (eng. backpropagation). U procesu učenja mreže, gradijenti se koriste za optimizaciju odabranog korijenskog čvora (tipično funkcije gubitka) iterativnim ažuriranjem mrežnih parametara. Brojne optimizacijske metode za učenje neuronskih mreža implementirane su u optim modulu, a sve komponente potrebne za definiciju same arhitekture neuronske mreže mogu se naći u nn modulu. Svi podaci spremaju se u tenzore, koji su funkcionalno vrlo slični Numpyjevim matricama, a PyTorch podržava i ubrzane implementacije mnogih operacija na GPU. Općenito, ova biblioteka je poprilično intuitivna za korištenje, a njen izvorni kod je vrlo jasan i čitak.

### 3.1.1. Checkpointing funkcija

Funkcija checkpoint je dio Pytorch utils modula, a primjenu nalazi kod mreža koje zauzimaju velike količine memorije prilikom treniranja, osobito u unatražnom prolazu gdje se, osim samih parametara trebaju čuvati međurezultati unaprijednog prolaza i vrijednosti gradijenata. Veliki zahtjevi za memorijskim prostorom na GPU postavljaju ograničenja na određene hiperparametre procesa učenja, kao što su veličina mini-batcha ili dimenzije ulaznih slika, zbog čega može doći do smanjenja kvalitete samog procesa učenja, a time i do lošije predikcije naučenog modela. Ova funkcionalnost rješava spomenuti problem na sljedeći način: umjesto da se svi međurezultati unaprijednog prolaza čuvaju u memoriji za izračun gradijenata, tijekom unatražnog prolaza potrebni se izlazi ponovno računaju. Ovako se olakšavaju zahtjevi za memorijom, naravno pod cijenu sporijeg vremenskog izvođenja programa.

Checkpointing je u ovom radu iskorišten na ljestvičastom i Tiramisu modelu. Kod ljestvičastog DenseNeta ova funkcionalnost je uvedena samo na gustim blokovima u putanji poduzorkovanja, pošto je to memorijski jedini kritičan dio te mreže, dok je u Tiramisu DenseNetu implementiran checkpoint na svakom gustom bloku (u objema putanjama) i na transponiranoj konvoluciji. U oba slučaja veličina mini-batcha je povećana 4 puta uz checkpointing.

## 3.2. Programska izvedba odabranih arhitektura

### 3.2.1. Ljestvičasti DenseNet

PyTorch implementacija ljestvičastog DenseNet169 modela opisanog u potpoglavlju 2.3 i ilustriranog na slici 2.2, priložena je u nastavku. Prva polovica mreže, tj. ekstraktor značajki (uz izuzeće posljednjeg sloja za klasifikaciju) preuzet je iz *torchvision.models* modula. To je modul s gotovim dubokim modelima unaprijed pretreniranim na skupu ImageNet. Ostatak mreže je implementiran samostalno u sklopu rada, te su parametri inicijalizirani nasumično. Kako ovi parametri nemaju nikakvog predznanja, za njih je postavljena stopa učenja 5 puta veća nego za pretrenirane parametre prilikom definicije objekta optimizatora Adam. Drugi dio mreže sastoji se od 3 različite komponente: kontekstualnog bloka `Context`, blokova za naduzorkovanje `TransUp` i krajnje konvolucije s bilinearnom interpolacijom. Kod za prve dvije spomenute komponente priložen je u nastavku.

```
1 import torch
2 import torch.nn as nn
```

```
3
4  class Context(nn.Sequential):
5      def __init__(self, in_channels, out_channels):
6          super().__init__()
7          self.add_module('norm1', nn.BatchNorm2d(in_channels))
8          self.add_module('relu1', nn.ReLU(inplace=True))
9          self.add_module('conv1', nn.Conv2d(in_channels, 4*
       out_channels, kernel_size=1))
10
11         self.add_module('norm2', nn.BatchNorm2d(4*out_channels))
12         self.add_module('relu2', nn.ReLU(inplace=True))
13         self.add_module('conv2', nn.Conv2d(4*out_channels,
       out_channels, kernel_size=3, dilation=2, padding=2))
14
15     def forward(self, x):
16         return super().forward(x)
17
18 class TransUp(nn.Module):
19     def __init__(self, in_channels, out_channels):
20         super().__init__()
21         self.norm1 = nn.BatchNorm2d(in_channels))
22         self.relu1 = nn.ReLU(inplace=True))
23         self.conv1x1 = nn.Conv2d(in_channels, out_channels,
       kernel_size=1)
24
25         self.norm2 = nn.BatchNorm2d(2*out_channels))
26         self.relu2 = nn.ReLU(inplace=True))
27         self.conv3x3 = nn.Conv2d(2*out_channels, out_channels,
       kernel_size=3, padding=1)
28
29     def forward(self, x, skip):
30         _, _, skip_h, skip_w = skip.size()
31
32         skip = self.conv1x1(self.relu1(self.norm1(skip)))
33         out = nn.functional.interpolate(x, size=(skip_h, skip_w),
       mode='bilinear', align_corners=True)
34
35         out = torch.cat([out, skip], 1)
36         out = self.conv3x3(self.relu2(self.norm2(out)))
37         return out
```

Kontekstualni blok se sastoji od dva konvolucijska sloja kojima prethodi normalizacija po grupi i ReLU (eng. rectified linear unit). Kompozitna funkcija koja se sastoji od

normalizacije popraćene ReLU-om pa konvolucijom, prisutna je u svakom konvolucijskom sloju mreže. Normalizacija po grupi dobra je tehnika regularizacije prilikom učenja, dok ReLU uvodi dozu nelinearnosti u model. Druga konvolucija ima faktor dilatacije `dilation` postavljen na 2, čime se poboljšava hvatanje širokog konteksta sa slike. Blok za naduzorkovanje prima dva ulazna tenzora, izlaz iz prethodnog sloja `x` i iz preskočne veze `skip`. Pritom `skip` ima dvostruko veću rezoluciju i znatno veći broj značajki, zbog čega ga je potrebno projicirati 1x1 konvolucijom u dimenziju koja je po broju značajki jednaka ulazu `x`, a `x` je potrebno bilinearnom interpolacijom povećati da po veličini odgovara ulazu `skip`. Konačno, transformirani se tenzori konkateniraju i šalju u posljednji konvolucijski sloj.

Posljednji konvolucijski sloj ima zadatak projicirati mapu značajki u dimenziju jednaku broju klasa, čiji se izlaz zatim šalje funkciji gubitka. Ova krajnja konvolucija prisutna je na dva mjesta u arhitekturi mreže (pošto su implementirana i dva gubitka), te na slici 2.2 odgovara blokovima AUX LOSS i LOSS. Osim transformacije u dimenziju broja klasa, potrebno je ove mape značajke podići na rezoluciju jednaku ulaznoj slici, što se postiže bilinearnom interpolacijom. U poglavlju 2.3 postavilo se pitanje ima li razlike u redoslijedu ovih dviju operacija, te je u potpoglavlju 2.3.1 proveden matematički dokaz koji je doveo da zaključka da su operacije komutativne. Eksperimentalno se odgovor na ovo pitanje može utvrditi sljedećim isječkom koda.

```
import sys, torch, time

size = int(sys.argv[1])
scale_factor = int(sys.argv[2])

feature_map = torch.randn(1,128,size,size)
w = torch.randn(2,128,1,1)

since = time.time()
h1 = torch.nn.functional.conv2d(feature_map, w)
r1 = torch.nn.functional.interpolate(h1, scale_factor=scale_factor,
    mode='bilinear', align_corners=True)
time_elapsed = time.time() - since
print("conv -> bilin: time - %d"%(time_elapsed))

since = time.time()
h2=torch.nn.functional.interpolate(feature_map, scale_factor=
    scale_factor, mode='bilinear', align_corners=True)
r2=torch.nn.functional.conv2d(h2, w)
time_elapsed = time.time() - since
```

```
print("bilin -> conv: time - %d"%(time_elapsed))

mean = (torch.norm(r1)+torch.norm(r2))/2
ratio = torch.norm(r1-r2)/mean

if ratio < 1e-4:
    print("1x1 convolution and bilinear interpolation are comutative")
else:
    print("1x1 convolution and bilinear interpolation are not comutative")
```

Pokretanjem koda s različitim vrijednostima faktora uvećanja `scale_factor` i veličine ulaznog podatka `size` zaključujemo da su ove dvije operacije komutativne, te da je jedina razlika u vremenu izvođenja. Drugi u gornjem kodu izloženi slučaj sporiji je od prvoga, ovisno o faktoru uvećanja kod interpolacije i prostornoj dimenziji mapi značajki.

### 3.2.2. Tiramisu DenseNet

Implementacija mreže Tiramisu, opisana u potpoglavlju 2.4. preuzeta je iz javnog GitHub repozitorija [12], te su svi njeni parametri inicijalizirani nasumično. Temeljni element mreže je gusti blok, čija je implementacija priložena u nastavku.

```
import torch
import torch.nn as nn

class DenseLayer(nn.Sequential):
    def __init__(self, in_channels, growth_rate, kernel_size=3, stride=1):
        super().__init__()
        self.add_module('norm', nn.BatchNorm2d(in_channels))
        self.add_module('relu', nn.ReLU(True))
        self.add_module('conv', nn.Conv2d(in_channels, growth_rate, kernel_size=kernel_size,
                                          stride=stride, padding=int(kernel_size/2), bias=True))
        self.add_module('drop', nn.Dropout2d(0.2))

    def forward(self, x):
        return super().forward(x)

class DenseBlock(nn.Module):
```

```
    def __init__(self, in_channels, growth_rate, n_layers, upsample=
    False):
        super().__init__()
        self.upsample = upsample
        self.layers = nn.ModuleList([DenseLayer(
            in_channels + i*growth_rate, growth_rate)
            for i in range(n_layers)])

    def forward(self, x):
        if self.upsample:
            new_features = []
            for layer in self.layers:
                out = layer(x)

                x = torch.cat([x, out], 1)
                new_features.append(out)
            return torch.cat(new_features,1)
        else:
            for layer in self.layers:
                out = layer(x)
                x = torch.cat([x, out], 1)
            return x
```

Kompozitna funkcija jednog sloja gustog bloka sastoji se od normalizacije po grupi, ReLU funkcije, 3x3 konvolucije i, konačno regularizacijske metode nasumičnog ispuštanja značajki. Gusti blok `DenseBlock` ima dvije inačice, ovisno o tome nalazi li se u putanji poduzorkovanja ili naduzorkovanja, što se regulira parametrom `upsample`. Prilikom poduzorkovanja, kad je ovaj parametar postavljen na `False`, na kraju se vraća konkatenacija ulaza u blok `x` i svih novih značajki ekstrahiranih u bloku `out`, kako bi preko skip veza u drugoj putanji mreža imala uvid u sve dostupne informacije na svakoj rezoluciji. Zahvaljujući ovome, kroz prvu putanju dolazi do eksplozije broja izlaznih značajki, stoga istu stvar nije pametno, a niti je potrebno, provoditi u drugoj putanji, već je dovoljno na izlaz poslati samo nove značajke `new_features`.

Blokovi za naduzorkovanje `TransitionUp` implementirani su u sljedećem isječku koda.

```
class TransitionUp(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.convTrans = nn.ConvTranspose2d(
            in_channels=in_channels, out_channels=out_channels,
            kernel_size=3, stride=2, padding=0, bias=True)
```

```
7
8       def forward(self, x, skip):
9           out = self.convTrans(x)
10          out = center_crop(out, skip.size(2), skip.size(3))
11          out = torch.cat([out, skip], 1)
12          return out
13
14  def center_crop(layer, max_height, max_width):
15      _, _, h, w = layer.size()
16      xy1 = (w - max_width) // 2
17      xy2 = (h - max_height) // 2
18      return layer[:, :, xy2:(xy2 + max_height), xy1:(xy1 + max_width)
        ]
```

Slično kao i kod ljestvičastog DenseNeta, ulaz u ove blokove čine neposredno prethodni izlaz `x` te izlaz iz preskočne veze `skip`, pri čemu je `skip` točno dva puta većih prostornih dimenzija. Transponirana konvolucija s veličinom jezgre 3 i korakom 2 te bez rubnog popunjavanja nulama na izlazu daje tenzor veličine $2k+1$, pri čemu je $k$ veličina ulaznog tenzora. Stoga je potrebno još povećani ulazni tenzor izrezati, tako da po veličini točno odgovara tenzoru iz preskočne veze. Centrirano izrezivanje implementirano je u funkciji `center_crop`.

### 3.2.3. Postavke hiperparametara treniranja

Svaka epoha procesa učenja mreže ima fazu treniranja i fazu validacije, pri čemu se nakon svake faze računa prosječna točnost (eng. accuracy) definirana formulom (3.1), i omjer presjeka i unije predikcije mreže sa stvarnim izlazom (eng. intersection over union, IoU), prema formuli (3.2). Priložene formule vrijede za LIP skup slika. Pritom je stvarna segmentacijska mapa, koja odgovara ulaznom podatku, označena s $GT$ (eng. ground truth), dok je mapa, dobivena prema predikciji mreže, označena s $PRED$. Obje mape su 2D matrice jednakih dimenzija kao i ulazna slika (visine $H$ i širine $W$), a svi elementi matrice čiji odgovarajući pikseli ulazne slike pripadaju čovjeku, jednaki su 1, te 0 u suprotnom.

$$Acc = \frac{\sum_{i,j}^{H,W} \mathbf{1}\{PRED_{ij} = GT_{ij}\}}{H \cdot W} \tag{3.1}$$

$$IoU = \frac{\sum_{i,j}^{H,W} \mathbf{1}\{PRED_{ij} = 1 \wedge GT_{ij} = 1\}}{\sum_{i,j}^{H,W} \mathbf{1}\{PRED_{ij} = 1 \vee GT_{ij} = 1\}} \tag{3.2}$$

Za izračun istih mjera točnosti kod skupa Cityscapes, iskorištene su službene skripte za evaluaciju [13].

Jedna epoha odgovara jednom prolasku kroz cijeli skup slika, a cijeli proces ima 100 epoha za svaku mrežu, radi adekvatne usporedbe rezultata. Ulazna slika je veličine 512x512 piksela za skup LIP, 512x1024 za skup Cityscapes. Shodno tome, odabrana je maksimalna veličina batcha koja može stati u memoriju GPU-a, prikazana u tablici 3.1. Ovdje valja napomenuti kako nisam provela svaku moguću kombinaciju evaluacije svakog modela nad svakim skupom. Prvo, nije bilo potrebe iskušati Tiramisu model na Cityscapesu, pošto u radu [5] autori nisu priložili evaluaciju svog modela na tom skupu, dakle službenih ocjena za usporedbu rezultata nema. Također, implementacijski kod za Tiramisu model preuzet je sa službenog GitHub repozitorija [12] autora originalnog rada, stoga možemo pretpostaviti da za provjerom ispravnosti implementacije modela nema potrebe. Drugo, osnovica usporedbe poslužila je svrsi na Cityscapes skupu, pa nisam vidjela poantu u treniranju tog jednostavnog modela nad LIP skupom, pošto je evidentno da rezultati ne bi mogli nadmašiti druga dva, složenija modela.

| Veličina batcha | Osnovica usporedbe | Ljestvičasti DenseNet | Tiramisu DenseNet |
|---|---|---|---|
| Cityscapes | 32 | 16 | - |
| LIP | - | 16 | 3 |

**Tablica 3.1:** Veličina batcha prilikom treniranja za svaki model i svaki skup slika. Polja označena znakom - ukazuju na to da se učenje u toj kombinaciji nije provodilo.

Što se tiče stope učenja, inicijalno je za sve parametre pretrenirane na ImageNetu postavljena na $10^{-4}$, dok je za parametre bez ugrađenog predznanja stopa učenja 5 puta veća. Razlog ovome jest u tome što je za pretrenirane parametre potrebno usporiti proces učenja, u odnosu na nasumično inicijalizirane parametre.

### 3.2.4. Zahtjevi za računalnim resursima

Učenje svih navedenih mreža je pokrenuto na GTX 1080 Ti grafičkoj kartici, koja ima 11GB memorije na raspolaganju. Prilikom treninga na LIP skupu izmjerena je količina alocirane memorije na GPU. Tu su ubrojeni tenzori koji drže podatke, parametre mreže, međurezultate i izračunate gradijente. Za ljestvičasti DenseNet uz odgovarajuću veličinu batcha alocirano je oko 10.18GB, a za Tiramisu (uz 4 puta manji batch) 10.67GB. Trajanje faza treniranja i validacije za svaki model i svaki skup slika priloženo je u tablicama 3.2 i 3.3.

| Faza treniranja | Osnovica usporedbe | Ljestvičasti DenseNet | Tiramisu DenseNet |
|---|---|---|---|
| Cityscapes | 14min | 18min | - |
| LIP | - | 33min | 135min |

**Tablica 3.2:** Trajanje faze treniranja u jednoj epohi za svaki model i svaki skup slika. Polja označena znakom - ukazuju na to da se učenje u toj kombinaciji nije provodilo.

| Faza validacije | Osnovica usporedbe | Ljestvičasti DenseNet | Tiramisu DenseNet |
|---|---|---|---|
| Cityscapes | 1min 45s | 2min 30s | - |
| LIP | - | 4min | 9min |

**Tablica 3.3:** Trajanje faze validacije u jednoj epohi za svaki model i svaki skup slika. Polja označena znakom - ukazuju na to da se učenje u toj kombinaciji nije provodilo.

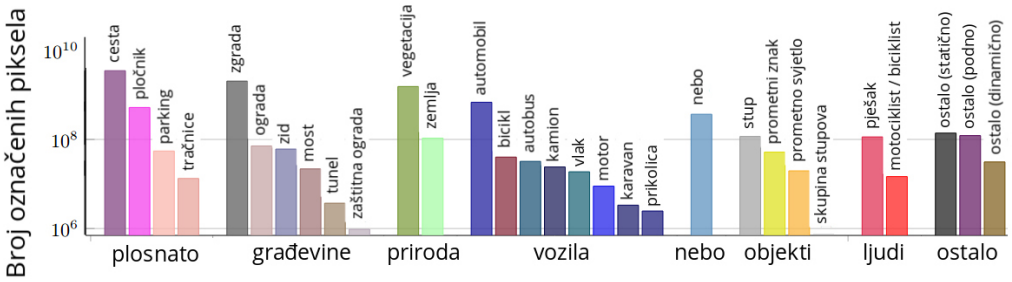
# 4. Skupovi slika za učenje

## 4.1. Cityscapes

Prilikom odabira i implementacije neuronske mreže, i kompletnog procesa učenja, zgodno bi bilo imati nekakvu garanciju da se u kodu nije potkrala nikakva greška. No programe takvog tipa vrlo je teško testirati i provjeriti. Stoga je uobičajeno vlastitu implementaciju programa provjeriti nad nekim klasičnim skupom podataka za kojeg su javno dostupni rezultati validacijskih mjera poput točnosti i IoU mjere, čime je potrebno reproducirati te rezultate. Tek nakon što se ustanovi da je kod ispravan, može se nastaviti s razvojem i učenjem odabranog modela nad vlastitim skupom podataka svojstvenom problemu. Skup podataka odabran za verifikaciju koda je Cityscapes, koji obuhvaća fotografije iz 50 različitih europskih gradova uslikane iz automobila u vožnji. 5000 slika iz skupa su precizno segmentirane na razini piksela, dok 20000 dodatnih ima grube anotacije za metode slabo-nadziranog učenja. Ovaj skup široko se koristi za razvoj programa strojnog učenja za autonomnu vožnju, zbog čega su i brojne evaluacije spomenutog skupa široko dostupne. Za potrebe ovog rada korišten je podskup od spomenutih precizno anotiranih slika, pri čemu ih je 2975 izdvojeno za treniranje, 500 za validaciju i, konačno, 1525 za testiranje. Primjeri nekoliko slika iz Cityscapes skupa prikazani su na slici 4.1. Slike su segmentirane u ukupno 30 klasa, no samo 19 ih je evaluirano zbog nedostatka dostupnih primjera iz preostalih 11 klasa (distribucija zastupljenosti pojedinih klasa po fino anotiranim slikama u skupu prikazana je na slici 4.2). Pikseli koji pripadaju tim ostalim klasama time se zanemaruju prilikom izračuna evaluacijskih mjera.

**Slika 4.1:** Cityscapes skup za učenje [6].

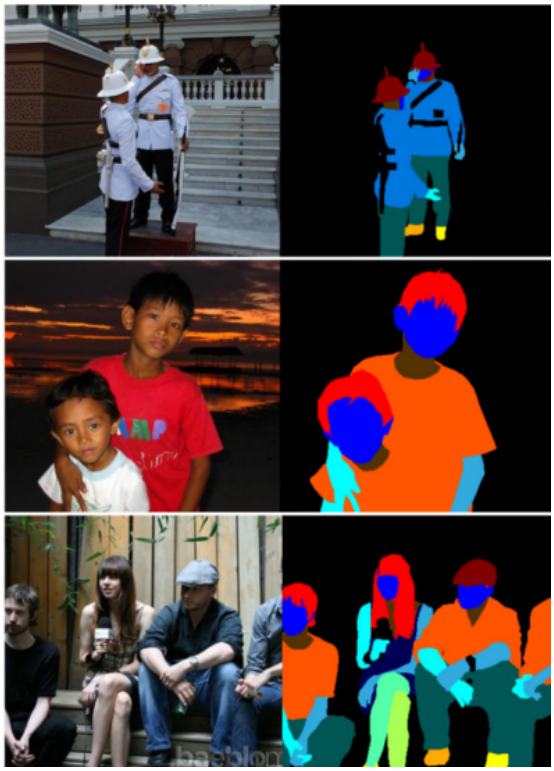
**Slika 4.2:** Ukupan broj fino označenih piksela za svaku pojedinu klasu u fino anotiranim slikama skupa Cityscapes [6].



## 4.2. LIP

Skup Look Into Person (LIP) ima preko 50000 slika u boji i crno-bijelih slika od kojih je otprilike jedna petina neoznačeno (slike za testiranje naučenog modela). Sve ostale su označene s 19 semantičnih dijelova tijela ili odjeće (lice, kosa, lijeva i desna ruka, noga i obuća, šešir, rukavice, sunčane naočale, majica, haljina, kaput, čarape, hlače, suknja, kombinezon, šal). Slike su uzete iz Microsoft COCO skupa slika, te su izrezani oni dijelovi slika na kojima se nalaze ljudi. [7] Na slici 4.3 prikazana su tri primjera slika i vizualizacija odgovarajućih segmentacijskih mapa. Podskup korišten u ovom radu sastoji se od 28210 slika za treniranje, 4987 slika za validaciju i 4991 slika za testiranje, pri čemu su crno-bijele slike izbačene iz originalnog skupa. Također, svih 19 klasa spojeno je u jednu semantičku klasu, čovjek, a svi neoznačeni pikseli (dijelovi crne boje na slici 4.3) svrstani su u drugu klasu, pozadina. Ovime se pojednostavljuje problem segmentacije i omogućuje se preciznija predikcija na rubovima između ljudi i pozadine, što je vrlo bitno za zadatak uklanjanja pozadine.

**Slika 4.3:** LIP skup za učenje [7].



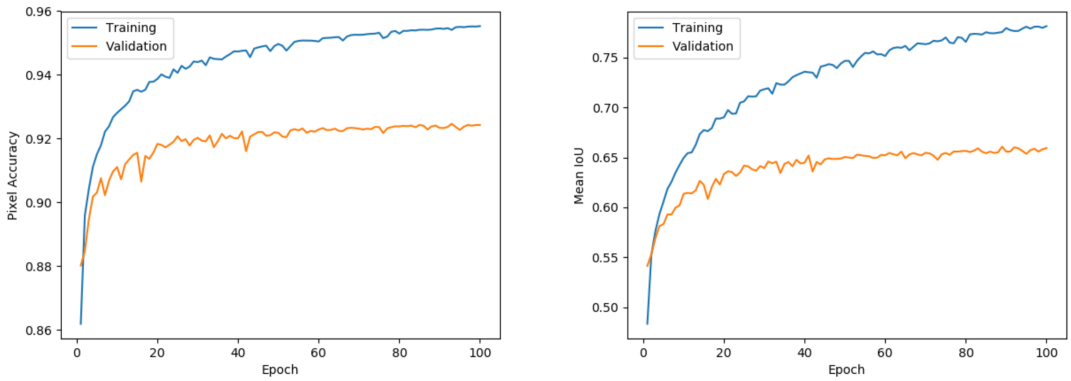
# 5. Analiza rezultata

Ovo poglavlje razdvojeno je u dva dijela, pri čemu su rezultati priloženi i analizirani odvojeno za eksperimente nad Cityscapes skupom i LIP skupom. Analiza se sastoji od grafičkog pregleda rasta mjera uspješnosti tijekom treninga dubokih modela, te konačnih rezultata istreniranih modela nad validacijskim i testnim podskupom slika.

## 5.1. Eksperimenti nad Cityscapes skupom

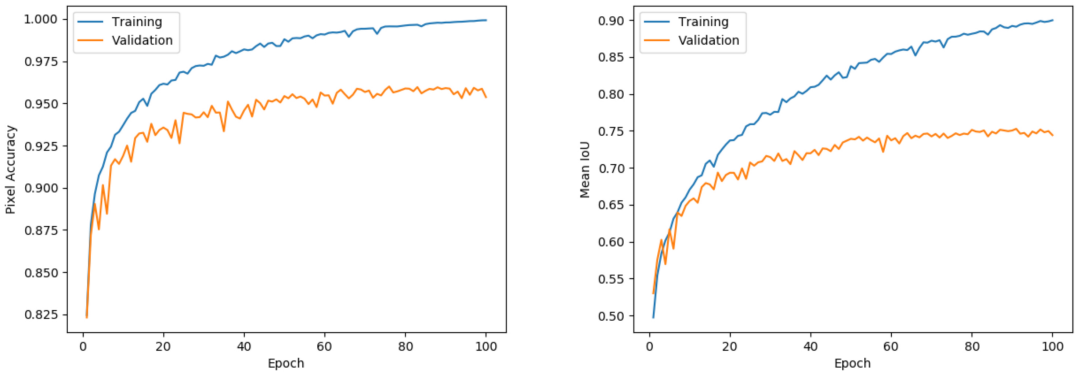
Slika 5.1 prikazuje razvojni proces učenja, uključivo s fazom validacije, jednostavne mreže, tj. osnovice usporedbe po epohama, praćen dvjema mjerama uspješnosti. Točnost piksela prikazana je na lijevom grafu, pri čemu maksimalna ostvarena vrijednost tijekom treninga iznosi 95.53%, a tijekom validacije 92.07%. Na drugom grafu može se vidjeti rast po klasama usrednjene IoU mjere, koja dostiže maksimalnu vrijednost u fazi treniranja u vrijednosti od 78.14%, a tijekom faze validacije 66.08%. U radu [4], koji je odabran za usporedbu rezultata, proveden je isti eksperiment i zabilježen je sličan rezultat za IoU mjeru nad validacijskim skupom iznosa 66.2%.

**Slika 5.1:** Grafovi rasta mjera točnosti (točnost piksela lijevo, IoU desno) prilikom treninga jednostavnog modela na Cityscapes skupu slika.



Što se tiče treniranja ljestvičastog DenseNeta, grafički prikaz istih mjera uspješnosti vidljiv je na slici 5.2. Maksimalne postignute vrijednosti točnosti piksela su redom 99.46% i 95.64% za fazu treninga i validacije, dok maksimum IoU mjere dostiže vrijednosti 89.98% i 75.28%. Usporedbe radi, rezultat IoU mjere nad validacijskim skupom prilikom treniranja iste mreže (ljestvičasti DenseNet169), kako je zapisano u radu [4], iznosi 75.8%.

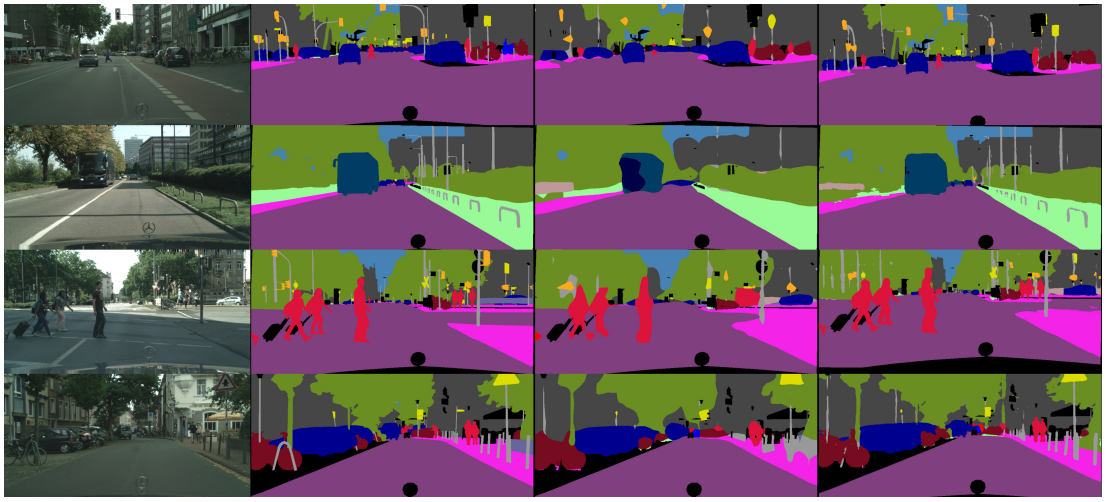
**Slika 5.2:** Grafovi rasta mjera točnosti (točnost piksela lijevo, IoU desno) prilikom treninga ljestvičastog modela na Cityscapes skupu slika.




Rezultati segmentacije nekoliko nasumično odabranih slika iz validacijskog skupa za obje mreže priloženi su na slici 5.3. Stupci su organizirani sljedećim redom: u prvom su originalne ulazne, u drugom je ispravna segmentacija u 19 klasa, uz posljednju 20. klasu označenu crnom bojom, koja se zanemaruje prilikom izračuna mjera uspješnosti, treći stupac odgovara rezultatima dobivenim od jednostavnog modela, a četvrti pripada ljestvičastom DenseNetu. Nedostatci jednostavnije mreže bez preskočnih veza, najbolje se primjećuju na rubovima segmentiranih objekata, gdje je učinak bilinearne interpolacije najuočljiviji. Također, slabiji kapacitet jednostavnog modela evidentira se na slici s autobusom u drugom redu, gdje je prednju stranu autobusa mreža svrstala u klasu automobil, dok je ljestvičasti DenseNet napravio zanemarivo sitnu pogrešku u tom području. Jednostavni model generalno ima problema s manjim objektima poput prometnih stupova, znakova, svjetala, zaštitne ograde isl., pošto se prilikom poduzorkovanja lako izgube svi sitniji detalji, a ova mreža, za razliku od ljestvičaste inačice, ipak nema mogućnost učenja u putanji naduzorkovanja.

Rezultati ostvareni u ovim eksperimentima, kao i naučene težine dubokih modela, neće se koristiti u daljnjim eksperimentima, već su samo poslužili kao osnovica usporedbe odabranog modela (ljestvičasti DenseNet169) s najjednostavnijom metodom

**Slika 5.3:** Usporedba rezultata naučenih mreža: jednostavnog (treći stupac) i ljestvičastog DenseNeta (četvrti stupac) na skupu Cityscapes. U prvom stupcu su odgovarajuće ulazne slike, a drugi stupac prikazuje stvarne, odnosno *ground truth* izlaze.



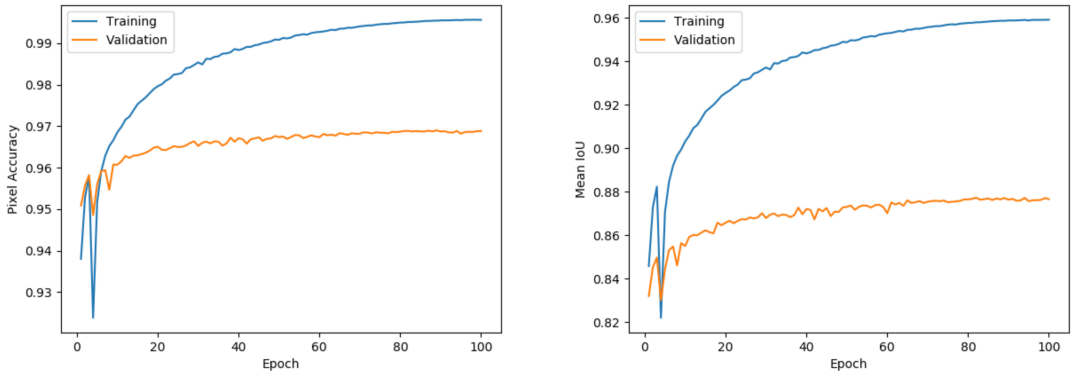
(jednostavni DenseNet121 uz bilinearnu interpolaciju) i provjera ispravnosti implementacije modela, cijelog procesa treninga i svih ostalih, pomoćnih funkcija.

## 5.2. Eksperimenti nad LIP skupom

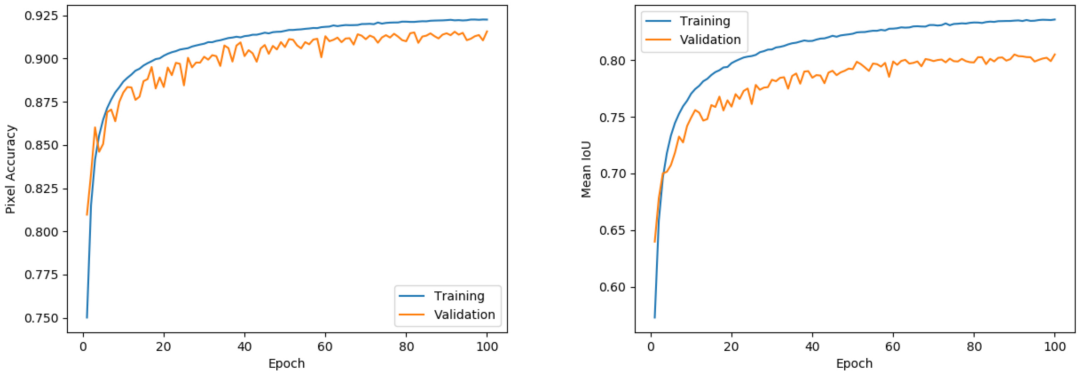
Eksperimenti predstavljeni i analizirani u ovom potpoglavlju, su konačno rješenje problema definiranog u uvodnom poglavlju, problema odvajanja ljudi od pozadine na slikama. Dvije odabrane mreže, ljestvičasti DenseNet169 i potpuni konvolucijski Tiramisu Densenet57 istrenirani su na LIP skupu, te je provedena usporedba uspješnosti i rezultata treninga obiju mreža, uz analizu i raspravu o tome koja od njih bi bila bolji izbor za dani zadatak, ili su možda u ansamblu najdjelotvornije.

Slike 5.4 i 5.5 prikazuju grafove ovisnosti mjera uspješnosti o broju epohe u procesima učenja, redom za ljestvičasti DenseNet i Tiramisu. Maksimalne vrijednosti točnosti piksela kod prve mreže su 99.68% u fazi učenja, i 96.94% u fazi validacije, dok je IoU mjera dosegla vrhunac na 95.90% i na 87.77%. Rezultati Tiramisua su nešto niži, s maksimalnom točnosti piksela od 92.42% i 91.56%, te maksimalnim IoU vrijednostima od 84.25% i 80.41%. Iako je temeljna arhitektura Tiramisu mreže znatno glomaznija od ljestvičaste arhitekture (prva je potpuna konvolucijska gusta mreža s transponiranom konvolucijom kao tehnikom naduzorkovanja, dok druga tek koristi bilinearnu interpolaciju i preskočne veze do prethodno ekstrahiranih značajki), ovi re-

**Slika 5.4:** Grafovi rasta mjera točnosti (točnost piksela lijevo, IoU desno) prilikom treninga ljestvičastog modela na LIP skupu slika.




**Slika 5.5:** Grafovi rasta mjera točnosti (točnost piksela lijevo, IoU desno) prilikom treninga Tiramisu modela na LIP skupu slika.




zultati nisu toliko iznenađujući. Odabrana Tiramisu inačica ipak sadrži samo 57 konvolucijskih slojeva, dok ih DenseNet s ljestvičastom arhitekturom ima 169. Odluka o veličini ovih dviju arhitektura ograničena je raspoloživim računalnim resursima, drugim riječima, odabrane su dvije najveće moguće inačice koje mogu stati u memoriju, te čija je vremenska složenost izvođenja unutar granica prihvatljivosti. Jedna pozitivna stvar koja se može primijetiti na grafovima Tiramisu mreže jest znatno manja razlika među izračunatim mjerama u fazi treninga i validacije, u odnosu na ljestvičastu mrežu, što ukazuje na činjenicu da je Tiramisu svakako bolji u generalizaciji danog problema, što je poželjna osobina za svaki naučeni model. Dok je normalizacija po grupi prisutna u obje mreže kao regularizacijska tehnika, u Tiramisu je dodatno još inkorporirana metoda nasumičnog ispuštanja značajki.

Na slici 5.6 prikazane su segmentacijske mape odabranih slika iz validacijskog skupa. Uslijed transformacije problema iz 19-klasnog u 2-klasni, crna boja je zadržana za pozadinu, dok su sve ostale združene klase, koje predstavljaju čovjeka, označene bijelom bojom. Priložene slike odabrane su kao najlošiji primjeri Tiramisu mreže (slika 5.6a) i ljestvičaste mreže (slika 5.6b), na kojima su očigledne određene pogreške koje naučene mreže rade. Prije svega, evidentno je da na većini slika na kojima ljestvičasti DenseNet značajno griješi, i Tiramisu griješi isto. Primjerice, na slici 5.6a u prvom redu Tiramisu je sjenu na pijesku zamijenio za dio ljudskog tijela, u drugom redu je vidljivo kako je segmentacija znatno lošija kad su ljudske figure mračnije, ili djelomično stopljene s pozadinom. U posljednja dva reda određeni objekti na fotografijama krivo su klasificirani kao ljudi. Na ovim primjerima ljestvičasta mreža je postigla solidne rezultate, no slika 5.6b prikazuje neke od neuspješnih primjera segmentacije ove mreže. U prvom redu vidimo kako je šarenu torbu s desna zabunom klasificirala kao komad odjeće, što ukazuje na slabiju ekstrakciju širokog konteksta sa slike. Poželjno bi bilo da mreža može raspoznati razliku između detalja koji podsjećaju na značajke odjeće od stvarnih odjevnih predmeta koji se tipično pojavljuju u obliku ljudske figure. Nadalje, u drugom redu ljestvičasti model nije uopće uspio prepoznati ljude u prtljažniku, što bi moglo biti uslijed problema skalabilnosti, pošto su figure poprilično sitne u odnosu na ostale objekte na slici. Tiramisu je ovdje prepoznao obje figure, no ovdje se javlja drugi problem velikog broja netočnih pozitivno klasificiranih piksela (eng. false positives, FP) za klasu čovjek. Čini se da Tiramisu donosi odluku često na temelju kontrasta intenziteta na rubovima pojedinih objekata, odnosno sklon je klasi čovjek pridijeliti sve ono što bojom, nijansom ili teksturom odskače od prevladavajućeg kolorita klasificiranog kao pozadina, i obrnuto, isključiti one piksele koji se stapaju s pozadinom. Ovo je vrlo nepoželjna karakteristika naučene mreže, jer obično fotografije široko variraju u količini, jakosti i položaju osvjetljenja, te bi svaka upotrebljiva mreža trebala biti dovoljno robusna da bude otporna na takve šumove u slikama. U trećem redu prikazan je primjer na kojem su obje mreže pale, pri čemu ih je zbunio vojnički uzorak na uniformi, vjerojatno u nedostatku odgovarajućih primjera u skupu za učenje. Slika u zadnjem redu poprilično je loše segmentirana od jedne i druge mreže, prvenstveno jer su krivo klasificirale autobusna sjedala kao ljude, izuzevši rukonaslone, dok im je, s druge strane, tamnija odjeća žene u fokusu izmakla u predikciji. Na ovom primjeru primjećujemo i jedan značajan nedostatak LIP skupa za učenje. Čovjek koji sjedi iza žene s bebom na stvarnoj izlaznoj maski (drugi stupac) svrstan je u pozadinu, no mreže su ga svejedno uspjele prepoznati i segmentirati.

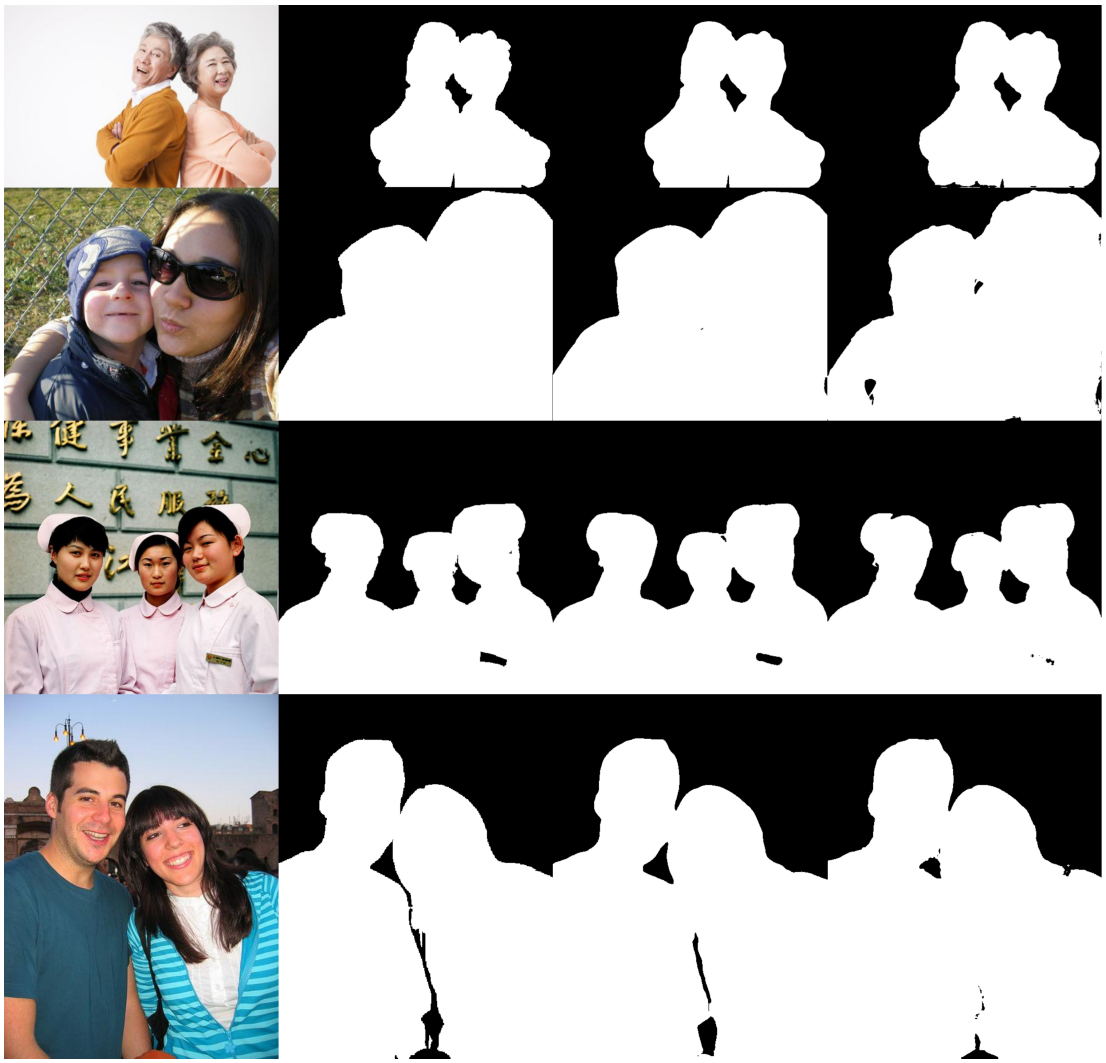
**Slika 5.6:** Raspored po stupcima, s lijeva na desno: izvorna ulazna slika, izvorna izlazna maska, segmentacijska maska dobivena iz predikcije ljestvičastog DenseNeta, segmentacijska maska dobivena iz predikcije Tiramisua. Bijela boja predstavlja klasu čovjek, a crna je pozadina. Odabrane slike su najlošiji primjeri slika iz validacijskog skupa segmentiranih prema predikcijama:

**(a)** Tiramisu mreže,

**(b)** ljestvičaste mreže.

Prethodne slike izdvojene su kao loši primjeri naučenih mreža, a u nastavku slijede neki od najbolje segmentiranih primjera, priloženih na slici 5.7. Vidimo kako najviše uspjeha mreže imaju sa slikama na kojima su ljudi uslikani en face, u krupnom planu, a pozadina je relativno jednolična.

**Slika 5.7:** Raspored po stupcima, s lijeva na desno: izvorna ulazna slika, izvorna izlazna maska, segmentacijska maska dobivena iz predikcije ljestvičastog DenseNeta, segmentacijska maska dobivena iz predikcije Tiramisua. Bijela boja predstavlja klasu čovjek, a crna je pozadina. Odabrane slike su primjeri najbolje segmentiranih slika iz validacijskog skupa u najuspješnijoj epohi treninga.



Koliko su naučene mreže uspješne u segmentaciji slika iz testnog skupa, koje uopće nisu vidjele tijekom procesa učenja, demonstrira slika 5.8. Ovdje nam stvarne segmentacijske maske nisu dostupne, stoga su u drugom i trećem stupcu priložene predikcije

redom ljestvičastog i Tiramisu DenseNeta, dok je prvi stupac ostao rezerviran za izvorne ulazne slike. U prvom redu vidimo nesigurnost Tiramisu modela s udaljenim figurama i dijelovima tijela koji se stapaju sa sjenama, dok ljestvičastu mrežu isto nije spriječilo u zadovoljavajućoj segmentaciji. Slika u drugom redu zadala je probleme objema mreža, pri čemu je ljestvičasti model promašio glavu lijeve plesačice i klasificirao dio pozadine kao dio tijela, a Tiramisu je odbacio sve tamne dijelove slike. Ipak ovaj primjer pokazatelj je bolje razine istreniranosti ljestvičaste mreže u odnosu na Tiramisu, pošto je, zahvaljujući bloku širokog konteksta savladala generalnu strukturu ljudskog tijela, pa je pokušala locirati i segmentirati nešto najbliže glavi lijeve plesačice, čije su konture slabo vidljive čak i za golo oko. Na trećem i četvrtom primjeru već se mogu primijetiti nedostatci ljestvičastog DenseNeta, tj. preciznije nedostatci samog skupa za učenje. Naime, originalno su u skupu svi objekti, koji ne spadaju u 19 predodređenih klasa, nabrojanih u poglavlju 4.2, označeni kao "pozadina", u što spada sav nakit, remenje, torbe, i slično (neki primjeri koji svjedoče o opisanim nedostatcima prikazani su na slici 6.3). Vidimo da je u oba slučaja mreža klasificirala remenje i nakit kao dio pozadine, dok Tiramisu, vjerojatno uslijed podnaučenosti nad danim skupom, to nije napravio. Nadalje, predzadnja slika prikazuje značajnu grešku koju je napravila Tiramisu mreža, gdje je kipić poznatog lika iz crtića zamijenila za čovjeka. Iako postoje evidentne sličnosti između čovjekoliko dizajniranog patka i generalne ljudske figure, pogreška ovakvog tipa mogla bi biti još jedna posljedica podnaučenosti Tiramisu mreže. Zadnji primjer sa slike 5.8 je pomalo varljiv zbog plakata koji se nalazi iza dječaka na pozornici. Trebamo li figuru dječaka s plakata svrstati u klasu čovjek ili pozadina, diskutabilno je i ovisi o konkretnoj primjeni rješenja za ovaj problem, a vidimo da su i dvije odabrane mreže u nesuglasnosti što se ove dileme tiče. Teško je odlučiti koja je od mreža u pravu u ovom slučaju.

Da sumiramo, nedvojbeno je da ljestvičasti DenseNet obavlja bolju segmentaciju od Tiramisua. U korist kvalitetne usporedbe modela, obje mreže podvrgnute su istim hiperparametrima u procesu učenja, što znači da su obje istrenirane nad istim skupom slika u istoj rezoluciji, uz isti broj epoha, isti optimizacijski algoritam te istu početnu stopu učenja. Pretpostavka je da postavljeni hiperparametri bolje odgovaraju ljestvičastoj mreži, dok bi Tiramisu vjerojatno dao bolje rezultate uz veći broj epoha ili prikladniji algoritam optimizacije.

**Slika 5.8:** Rezultati segmentacije na skupu za testiranje. U prvom stupcu su izvorne ulazne slike, a u drugom i trećem su redom rezultati ljestvičastog DenseNeta i Tiramisua.

# 6. Zaključak

LIP skup, iako na prvu djeluje kao odličan izbor za treniranje mreža u svrhu problema segmentacije ljudi od pozadine, ima dva značajno velika nedostatka. Prvi problem je u tome što je skup poprilično nekonzistentan u označavanju ljudi koji variraju po veličini. Nekoliko primjera priloženo je na slici 6.2. Promotrimo li svaku od ovih slika posebno, vidimo da među ljudima koji su skoro iste veličine, neki su označeni, a neki su pripojeni pozadini. Isto vrijedi i usporedimo li sadržaje dviju odvojenih slika. S obzirom na to da neuronske mreže nisu imune na skaliranje naučenih značajki, ovakva nekonzistentnost u podacima lako ih može zbuniti tijekom treninga. Doduše, ovaj problem odabrane mreže ipak su uspjele djelomično svladati, što ilustrira slika 6.1. Možemo primijetiti kako su mreže većinu ljudi na slikama (pa i one koji u originalnoj izlaznoj maski nisu uopće označeni) uspjele prepoznati, no nisu ih sve jako precizno segmentirale. Dodatni problem koji se javlja kao posljedica ovog nedostatka jest neadekvatan izračun mjera uspješnosti tijekom faze validacije, jer funkcija gubitka u procesu učenja slijepo vjeruje informacijama iz danih podataka, te kažnjava predikcije mreža poput ovih sa slike 6.1, iako su u principu duboki modeli u ovim primjerima dali rezultate koji su bolji od onih u skupu za učenje.

Drugi bitan problem jest da su određeni objekti koji se nađu na slikama, a koji ne pripadaju nijednoj od 19 izvornih klasa na koje je originalni LIP skup podijeljen, označeni kao pozadina. To se može vidjeti na slici 6.3. Stvari poput remena, nakita, ruksaka, pojaseva, itd. trebali bi biti pridijeljeni klasi čovjek ako se na slici nalaze na čovjeku, no shodno izvornoj namjeni ovog skupa (a to je segmentacija dijelova tijela i odjeće), prilikom transformacije skupa iz 19-klasnog u 2-klasni, svi navedeni objekti (i još mnogi drugi) pripojili su se pozadini. Kao posljedica ove anomalije u modificiranom skupu, na segmentiranim slikama ljestvičastog DenseNet modela (koji je bolje naučio dani skup od Tiramisua), možemo primijetiti mjestimične šupljine unutar ljudskih figura. Primjeri ove pojave mogu se vidjeti u trećem i četvrtom redu na slici 5.8. Tiramisu rjeđe radi ovakve greške, vjerojatno jer nije naučen na ulazne podatke jednako dobro kao što je ljestvičasti DenseNet. Konačno, uzevši navedene nedostatke

LIP skupa u obzir, možemo doći do zaključka kako bi originalni COCO skup (odnosno, slike iz COCO skupa na kojima se nalaze ljudi) možda ipak bio bolji izbor za primjenu opisanu u ovom radu.

Na kraju, valja napomenuti kako je u svrhu adekvatne usporedbe performansi ljestvičastog i Tiramisu DenseNeta, proveden trening s jednakim vrijednostima svih hiperparametara za jednu i drugu mrežu, uz iznimku veličine mini-grupe. Za svaku mrežu odabrana je maksimalna moguća vrijednost ovog hiperparametra unutar granica memorijskih resursa, pri čemu je ta vrijednost ispala 5 puta veća za ljestvičasti DenseNet. Ovo nas dovodi do zaključka kako je ljestvičasti DenseNet znatno kompaktniji, brži i jednostavniji model, uz dovoljno dobre konačne rezultate. Tiramisu bi se trebao trenirati na danom skupu možda i kroz dvostruko više epoha, što bi oduzelo ogromnu količinu vremena, osobito zato što jedna epoha kod treniranja Tiramisua traje 4 puta duže nego kod ljestvičastog. Samim time i evaluacija novih slika kod Tiramisua znatno je sporija, te ima veće zahtjeve za memorijske resurse. Sve u svemu, ljestvičasti DenseNet je brži, jednostavniji, memorijski manji, te ima sasvim dovoljan kapacitet da može solidno svladati dani problem segmentacije čovjeka od pozadine.

**Slika 6.1:** Primjeri slika na kojima su naučene mreže uspjele savladati problem LIP skupa ilustriran na slici 6.2. Redoslijed po stupcima je sljedeći: izvorna ulazna slika, izvorna izlazna maska, te predikcije ljestvičastog, a zatim Tiramisu modela.

**Slika 6.2:** Nekonzistentnost u označavanju ljudi u različitim relativnim veličinama. Pitanje koje se ovdje postavlja jest - u slikama prenapučenim ljudima, gdje postaviti granicu između osobe u fokusu interesa i osobe u pozadini?

**Slika 6.3:** Objekti koji ne ulaze u 19 predefiniranih klasa originalnog LIP skupa, prilikom transformacije u skup s dvije klase, postanu nepoželjne "rupe" u ljudskim figurama.

# LITERATURA


[1] J. Jordan, "An overview of semantic image segmentation." `https://www.jeremyjordan.me/semantic-segmentation/`, Svibanj 2018.

[2] T. Wu, S. Tang, R. Zhang, G. Guo, and Y. Zhang, "Consensus feature network for scene parsing," *CoRR*, vol. abs/1907.12411, 2019.

[3] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Computer Vision (ICCV), 2015 IEEE International Conference on*, 2015.

[4] I. Kreso, J. Krapac, and S. Segvic, "Efficient ladder-style densenets for semantic segmentation of large images," *CoRR*, vol. abs/1905.05661, 2019.

[5] S. Jégou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," *CoRR*, vol. abs/1611.09326, 2016.

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016.

[7] K. Gong, X. Liang, X. Shen, and L. Lin, "Look into person: Self-supervised structure-sensitive learning and A new benchmark for human parsing," *CoRR*, vol. abs/1703.05446, 2017.

[8] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[9] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv e-prints*, mar 2016.

[10] D. P. Kingma and J. L. Ba, "ADAM: A method for stochastic optimization," *ArXiv e-prints*, jan 2017.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[12] B. Fortuner, "One Hundred Layers Tiramisu - GitHub repository." `https://github.com/bfortuner/pytorch_tiramisu`, 2018. [Laatest commit: 15 March-2018].

[13] M. Cordts, "Cityscapes Scripts - GitHub repository." `https://github.com/mcordts/cityscapesScripts`, 2016. [Laatest commit: 28 April-2019].

## Semantička segmentacija osoba konvolucijskim modelima


### Sažetak

Semantička segmentacija prirodnih scena je važan zadatak računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolje rezultate na tom zadatku postižu konvolucijski modeli s ljestvičastim naduzorkovanjem. Zbog velikog broja komercijalnih primjena, posebno je zanimljiva binarna segmentacija osoba u prednjem planu. U okviru rada istražuju se postojeće pristupi za semantičku segmentaciju. Posebnu pažnju posvećuje se memorijskim zahtjevima naduzorkovanja latentnih reprezentacija. Cilj rada je izraditi izvedbu programskog sustava za učenje i primjenu segmentacijskog modela, uz prikaz i ocjenu ostvarenih rezultata. U rad se prilažu izvorni i izvršni kod razvijenih postupaka, ispitni sljedovi i rezultati, uz potrebna objašnjenja i dokumentaciju.



**Ključne riječi:** računalni vid, duboko učenje, semantička segmentacija, DenseNet arhitektura


## Semantic segmentation of humans using dense convolutional models


### Abstract

Semantic segmentation of natural scenes is an important task in computer vision with various diverting applications. Lately, the leaders in handling such tasks have been dense convolutional networks with ladder-style upsampling techniques. Because of its growing commercial applications, exeptionally interesting is a binary segmentation of humans in focus from the background. This work introduces a research of current segmentation methods, with an accent to memory consumption requirements of different upsampling techniques and their corresponding latent representations. The goal is to build and present a system for training and evaluation of chosen network architectures for semantic segmentation, along with the analysis of the obtained results and source code containing a required documentation.



**Keywords:** computer vision, deep learning, semantic segmentation, DenseNet architecture