Optimizacija segmentacije cestovnih scena u stvarnom vremenu

Barišić, Renata

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

Sveučilišni studij

OPTIMIZACIJA SEGMENTACIJE CESTOVNIH SCENA U STVARNOM VREMENU

Diplomski rad

Renata Barišić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Renata Barišić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1269R, 07.10.2022.
JMBAG:	0165081305
Mentor:	izv. prof. dr. sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 1:	izv. prof. dr. sc. Časlav Livada
Član Povjerenstva 2:	doc. dr. sc. Tomislav Galba
Naslov diplomskog rada:	Optimizacija segmentacije cestovnih scena u stvarnom vremenu
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno napraviti usporedbu U-Net, ENet i SegNet algoritama za segmentaciju cestovnih scena iz perspektive automobila. Implementacija se provodi u Pythonu uz korištenje PyTorch biblioteke. Nakon implementacije, potrebno je analizirati algoritme po preciznosti, brzini izvođenja, učinkovitosti u stvarnom vremenu i kompleksnosti implementacije. Cilj je pružiti duboki uvid u primjenjivost navedenih algoritama u kontekstu automobilske percepcije. Tema rezervirana za: Renata Barišić
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	19.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	26.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	26.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 26.09.2024.

Ime i prezime Pristupnika:	Renata Barišić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1269R, 07.10.2022.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Optimizacija segmentacije cestovnih scena u stvarnom vremenu**

izrađen pod vodstvom mentora izv. prof. dr. sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U ovom će se diplomskom radu obraditi problem semantičke segmentacije cestovnih scena usporedbom tri arhitekture konvolucijske neuronske mreže. Za početak, u drugom poglavlju, objasnit će se pojam segmentacije slike te navesti njezine vrste, a u prvom i jedinom potpoglavlju bit će navedene aktualne metode njezine provedbe. Zatim, u trećem poglavlju, bit će dana detaljna teorijska podloga triju korištenih arhitektura – *U-Net, SegNet* i *ENet*. Bit će objašnjene njihove sličnosti i razlike te će se argumentirati njihov odabir. Treće se poglavlje sastoji od tri potpoglavlja – po jedno za svaku od navedenih arhitektura. U četvrtom poglavlju bit će objašnjena implementacija rješenja – od pripreme skupa podataka u prvom potpoglavlju do opisa implementacije svake od arhitektura i opisa implementacije treninga u preostala četiri potpoglavlja. U petom će poglavlju biti prikazani rezultati implementacije te će se međusobno usporediti rezultati triju arhitektura.

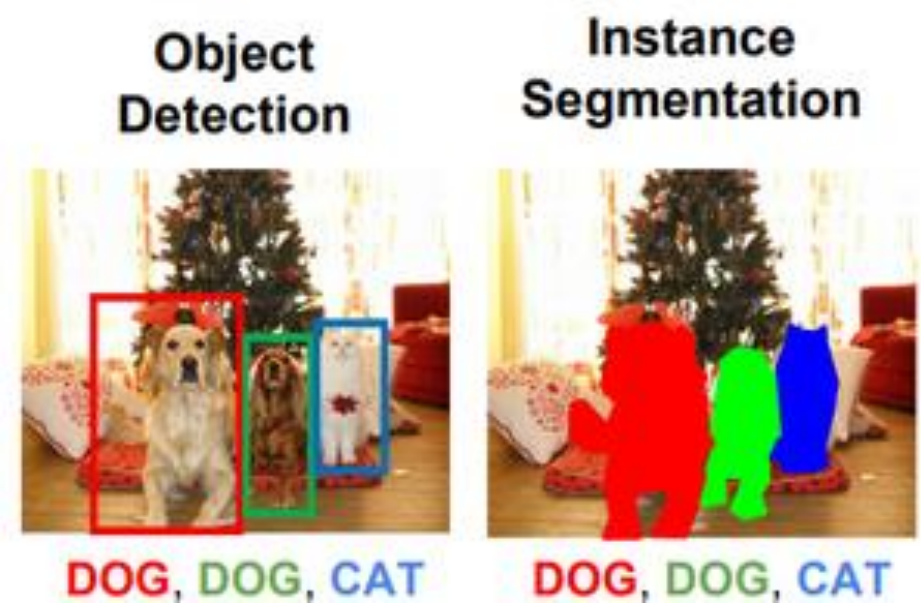
1.1. Zadatak diplomskog rada

U ovom je diplomskom radu potrebno napraviti usporedbu *U-Net, SegNet* i *ENet* algoritama za segmentaciju cestovnih scena iz perspektive automobila. Implementacija se provodi u *Python* programskom jeziku uz korištenje biblioteke *PyTorch*. Nakon implementacije, potrebno je analizirati navedene algoritme po točnosti, brzini izvođenja, učinkovitosti u stvarnom vremenu i kompleksnosti implementacije. Cilj je pružiti duboki uvid u primjenjivost navedenih algoritama u kontekstu automobilske percepcije.

2. SEGMENTACIJA SLIKE

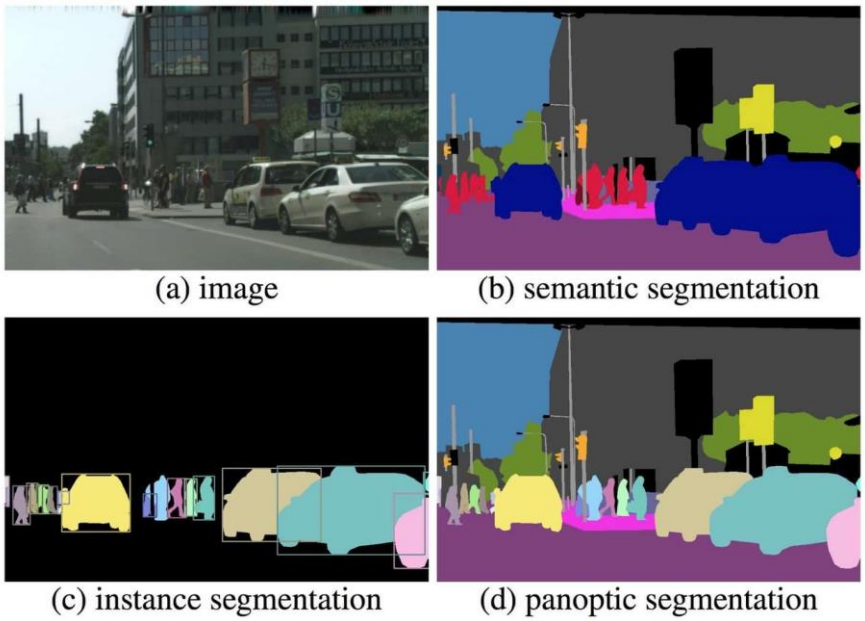
Segmentacija slike je tehnika računalnog vida kojoj je zadatak podjela digitalne slike u diskretne grupe piksela, odnosno segmente. Cilj segmentacije slike je pojednostaviti sliku i pretvoriti ju u format koji je značajniji za računalo i lakši za analizirati, prema [1]. Segmentacija slike koristi se u različite svrhe, poput analize medicinskih slika, prepoznavanja lica, videonadzora, robotske percepcije, proširene stvarnosti, kompresije i povećanja rezolucije slike te razumijevanja scene, kao što je navedeno u [2]. Za ovaj je rad najznačajnija upravo ta zadnja svrha – razumijevanje cestovne scene. Ona se fokusira na klasifikaciju i segmentiranje različitih elemenata scene iz prometa kao što su cesta, vozila, prometni znakovi i pješaci.

Važno je naglasiti razliku između segmentacije slike i detekcije objekta. Ova se dva pojma ponekad miješaju zbog svojih sličnosti – oba uključuju pronalaženje objekata na slici i njihovo klasificiranje, no ne radi se o istoj metodi. Detekcijom objekata pronalaze se svi objekti na slici te im se daje granični okvir (engl. *bounding box*) i klasa, dok segmentacija slike označava prisutnost objekta pomoću maske generirane po pikselima za svaki objekt na slici. Ova tehnika je preciznija od same detekcije jer može odrediti oblik svakog objekta prisutnog na slici - ona umjesto crtanja okvira, određuje koji pikseli čine neki objekt. Na slici 2.1. moguće je vidjeti razliku između ove dvije metode.



Sl. 2.1. Usporedba detekcije objekata i segmentacije slike [3].

Također je važno spomenuti kako postoje različite vrste segmentacije – prema [4], ona se dijeli na semantičku segmentaciju, segmentaciju instanci i panoptičku segmentaciju. Razlika između navedenih može se vidjeti na slici 2.2.



Sl. 2.2. Usporedba semantičke, segmentacije instanci i panoptičke segmentacije [5].

Semantička segmentacija je najjednostavnija vrsta segmentacije slike. Ona dodjeljuje klasu svakom pikselu slike, ali ne daje nikakav drugi kontekst ili informacije, poput objekata. Na konkretnom primjeru segmentacije cestovne scene, rezultat semantičke segmentacije bi bio slika na kojoj je svakom pikselu dodijeljena klasa, no nema razlika između više instanci iste klase – tako bi se automobili parkirani jedan ispred drugog jednostavno tretirali kao jedan dugačak segment klase *automobil*. Ova će se metoda zbog svoje jednostavnosti i dovoljno dobre primjenjivosti koristiti za implementaciju rješenja u nastavku ovog rada.

Nešto kompleksnija metoda je segmentacija instanci. Ona radi točno suprotno od semantičke segmentacije, odnosno ocrtava točan oblik svake zasebne instance objekta. Ona je zapravo naprednija metoda detekcije objekata jer pronalazi sve objekte u slici i svakom zasebno daje masku piksela. Segmentacija instanci bi dakle u nizu parkiranih auta označila svaki auto zasebno.

Kombinacija ove dvije metode je panoptička segmentacija. Ona određuje semantičku klasifikaciju svih piksela te razlikuje svaku instancu objekta na slici. Kod ove metode svaki piksel mora biti označen i semantičkom oznakom i identifikacijskim brojem instance, a pikseli koji dijele istu oznaku i identifikacijski broj pripadaju istom objektu. Ovo je dakle najkompleksnija od tri navedene metode.

2.1. Aktualne metode semantičke segmentacije cestovnih scena

Postoje razne efikasne metode segmentacije slike, no u svrhu semantičke segmentacije cestovnih scena najčešće se oslanja na metode dubokog učenja. U nastavku ovog potpoglavlja bit će navedene najsuvremenije arhitekture i metode rješavanja ovog zadatka.

PSPNet ili *Pyramid Scene Parsing Network,* opisana u [6], je arhitektura za semantičku segmentaciju koja koristi takozvani modul za piramidalno analiziranje. Taj modul dobiva informacije o globalnom kontekstu promatrajući različite regije zasebno. Ova arhitektura dakle koristi lokalni i globalni kontekst zajedno za konačno predviđanje.

FCN ili *Fully Convolutional Network,* predstavljena u [7], je arhitektura koja koristi isključivo lokalno povezane slojeve, zato što izbjegavanje upotrebe gustih slojeva znači manje parametara, a time i brže treniranje. To također znači da *FCN* može raditi za različite ulazne veličine slike baš zato što su sve veze lokalne.

DeepLab je poprilično kompleksna metoda čiji se rad fokusira na povećavanju vidnog polja filtera kako bi se uključio širi kontekst, ali bez povećanja broja parametara ili količine izračuna, kao što je opisano u [8].

HRNet ili *High-Resolution Network* je arhitektura koja u cijeloj mreži održava prikaze visoke rezolucije, kao što ime govori, što dovodi do preciznijih rezultata segmentacije, prema [9].

GSCNN ili *Gated Shape Convolutional Neural Network* je duboka neuronska mreža s dva toka – klasičnim tokom i tokom oblika, koji se obrađuju paralelno. Ovaj način rada daje preciznije rezultate, posebno kod malih i tankih objekata, tvrdi [10].

Osim navedenih, među najpoznatijim i najefikasnijim arhitekturama za semantičku segmentaciju cestovnih scena nalaze se *U-Net*, *SegNet* i *ENet* arhitekture koje će biti detaljno objašnjene u idućem poglavlju.

3. ANALIZA ARHITEKTURA U-NET, SEGNET I ENET

Prije ulaska u detaljnu analizu pojedinih arhitektura potrebno je argumentirati njihov odabir. Navedene tri arhitekture su odabrane zbog njihove široke primjene, različitih pristupa problemu segmentacije te uspješnosti u stvarnom vremenu, što ih čini idealnim kandidatima za usporedbu u kontekstu automobilske percepcije, no zašto baš one?

U-Net, kao vrhunac segmentacije slike u današnje vrijeme, iako ima impresivne rezultate u segmentaciji cestovnih scena, zapravo je kreirana za segmentaciju medicinskih slika. Njezina se primjena proširila i na brojne druge zadatke segmentacije, a i na zadatke van tog problema, kao što su povećanje rezolucije slike i kreiranje slike iz teksta.

Ova napredna i precizna metoda će se usporediti s metodom namijenjenom i specijaliziranom baš u svrhe segmentacije cestovnih scena i urbanih područja, a to je *SegNet*. Ovom će se usporedbom vidjeti igra li početna namjena arhitekture ulogu u konačnim rezultatima rješavanja problema.

Nadalje, navedene će se metode usporediti s arhitekturom dizajniranom za efikasnu segmentaciju koja koristi znatno manje računalnih resursa – *ENet*. Ova je arhitektura optimizirana za upotrebu u aplikacijama koje zahtijevaju visoke performanse i malo kašnjenje (engl. *latency*), kao što su autonomna vozila. Na ovaj će se način ustanoviti utječe li kompleksnija struktura i veći broj resursa na preciznije rezultate.

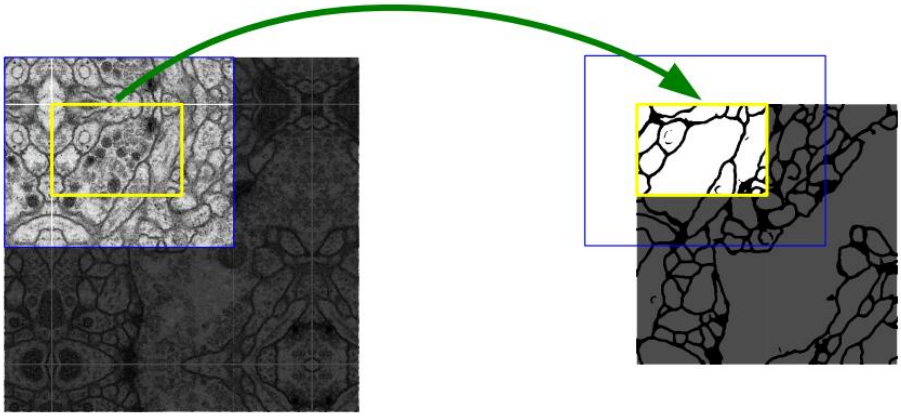
3.1. *U-Net* arhitektura

U-Net je arhitektura konvolucijske neuronske mreže koja je opisana u [11], a oslanja se na korištenje povećanja skupa podataka (engl. *data augmentation*) kako bi što učinkovitije koristila dostupne označene podatke. Povećanje skupa podataka je metoda stvaranja većeg broja podataka za modele strojnog učenja mijenjanjem postojećih ili generiranjem novih podataka. Ono se koristi jer je ova arhitektura kreirana za segmentaciju biomedicinskih slika, koje nisu dostupne u velikim količinama.

U-Net arhitektura se sastoji od sužavajućeg puta (engl. *contracting path*) za prikupljanje konteksta i od simetričnog proširivajućeg puta (engl. *expanding path*) koji omogućuje preciznu lokalizaciju. Autori ove arhitekture tvrde kako se takva mreža može istrenirati iz vrlo malog broja slika i kako nadmašuje u to vrijeme najbolju metodu, a to je bila konvolucijska mreža s kliznim prozorom.

Ova arhitektura je zapravo nadogradnja prethodno spomenute *FCN* arhitekture, ali je izmijenjena i proširena kako bi mogla dobro raditi i s malim brojem ulaznih fotografija. Ona nadopunjuje

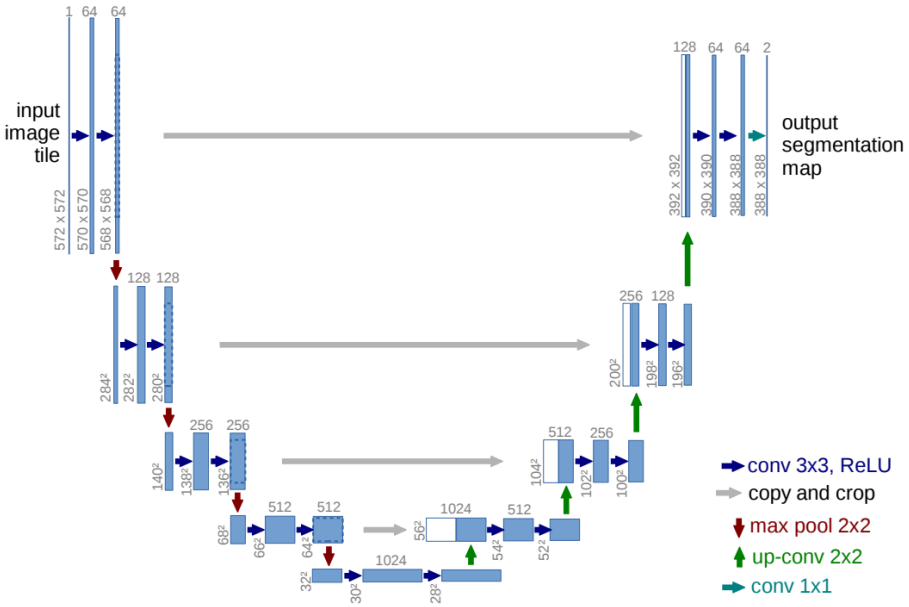
uobičajenu sužavajuću mrežu uzastopnim slojevima, a operatore sažimanja (engl. *pooling*) u proširivajućem dijelu mreže zamjenjuje operatorima povećanja uzorkovanja (engl. *upsampling*), kako bi ti operatori povećavali rezoluciju izlaza. U svrhu lokalizacije, značajke visoke rezolucije iz sužavajućeg puta kombiniraju se s izlazom povećanja uzorkovanja pa sloj uzastopne konvolucije tada može naučiti sastavljati preciznije izlaze na temelju tih informacija. Jedna važna izmjena u ovoj arhitekturi spram *FCN* arhitekture je u tome što u dijelu povećanja uzorkovanja postoji veliki broj kanala značajki (engl. *feature channels*), koji mreži omogućuju prosljeđivanje informacija o kontekstu slojevima više rezolucije. Na taj se način dobiva proširivajući put koji je više-manje simetričan sužavajućem. Također, ova mreža nema potpuno povezane slojeve i koristi samo važeći dio svake konvolucije, odnosno, izlazna mapa segmentacije sadrži samo one piksele za koje je dostupan potpuni kontekst na ulaznoj slici. Ovakav način rada omogućuje vrlo preciznu segmentaciju slika proizvoljne veličine, a to radi strategijom pločica preklapanja (engl. *overlap-tile*). Za predviđanje piksela u rubnom području slike, kontekst koji nedostaje dobiva se zrcaljenjem ulazne slike, što je moguće vidjeti na slici 3.1. Ova strategija takozvanog *popločavanja* je važna za segmentaciju velikih ulaznih slika, kako se ne bi dostupnom GPU memorijom ograničila maksimalna rezolucija slike.



Sl. 3.1. Zrcaljenje slike strategijom pločice preklapanja [11].

Ova je arhitektura dobila svoje ime po svojoj strukturi nalik slovu *U*, a prikazana je na slici 3.2. Ova vrsta arhitekture također se naziva i koder-dekoder arhitektura, jer se lijeva tj. sužavajuća strana ponaša kao koder, a desna tj. proširivajuća kao dekoder. Lijevi dio mreže slijedi tipičnu arhitekturu konvolucijske mreže, a sastoji se od 1) uzastopne primjene dviju konvolucija veličine 3x3 i to bez ruba (engl. *unpadded convolution*), nakon kojih slijedi *ReLU* (engl. *Rectified Linear Unit*) aktivacijska funkcija i 2) sažimanja po maksimalnoj vrijednosti (engl. *max pooling*) veličine

2x2 s korakom veličine 2 za smanjivanje uzorkovanja (engl. *downsampling*). U svakom se koraku smanjenja uzorkovanja udvostručuje broj kanala značajki. Svaki korak u desnom dijelu mreže se sastoji od 1) povećanja uzorkovanja mape značajki nakon kojeg slijedi konvolucija prema gore (engl. *up-convolution*) veličine 2x2 koja prepolovljuje broj kanala značajki, 2) ulančavanja s odgovarajućom izrezanom kartom značajki iz lijevog dijela mreže takozvanim prespojnim vezama (engl. *skip connections*) i 3) dvije konvolucije veličine 3x3, nakon kojih slijedi *ReLU* aktivacijska funkcija. Spomenuto izrezivanje karte značajki je neophodno zbog gubitka rubnih piksela u svakoj konvoluciji. Na posljednjem se sloju koristi konvolucija veličine 1x1 za mapiranje svakog vektora značajki, koji se sastoji od 64 komponente, u željeni broj klasa – u ovom primjeru su to 2 klase. Ukratko, može se reći da lijevi dio mreže služi za pronalaženje onoga što je na slici, a desni dio mreže služi za pronalaženje lokacije onoga što je na slici. Prespojne veze igraju ključnu ulogu u spajanju te dvije informacije za konačan izlaz.



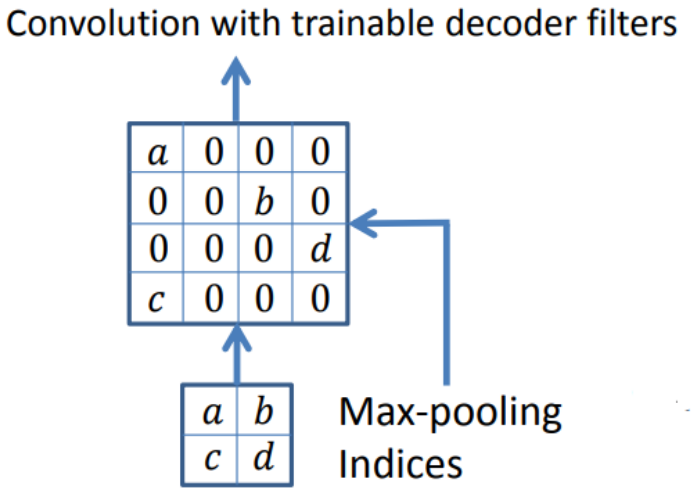
Sl. 3.2. Struktura U-Net arhitekture [11].

Važno je naglasiti da je za preciznu segmentaciju potrebno odabrati veličinu ulazne slike tako da se sva maksimalna sažimanja veličine 2x2 primjenjuju na sloj s jednakom visinom i širinom. Također, iz slike 3.2. je vidljivo kako ulazna slika i izlazna segmentacijska mapa nisu jednake veličine – već je spomenuto kako se to događa zbog gubitka rubnih piksela u svakoj konvoluciji, a način izbjegavanja tog gubitka bi bilo dodavanjem ruba (engl. *padding*).

3.2. *SegNet* arhitektura

Kao što je već spomenuto, *SegNet* je duboka potpuno konvolucijska arhitektura neuronske mreže kreirana specifično za zadatke razumijevanja scene. Zbog toga je, prema [12], dizajnirana da bude učinkovita i u smislu uporabe memorije i u smislu vremena računanja tijekom zaključivanja (engl. *inference*). Značajno je manja u broju parametara koji se mogu trenirati od drugih tada aktualnih arhitektura te pruža dobre performanse s konkurentnim vremenom zaključivanja i najučinkovitijom uporabom memorije.

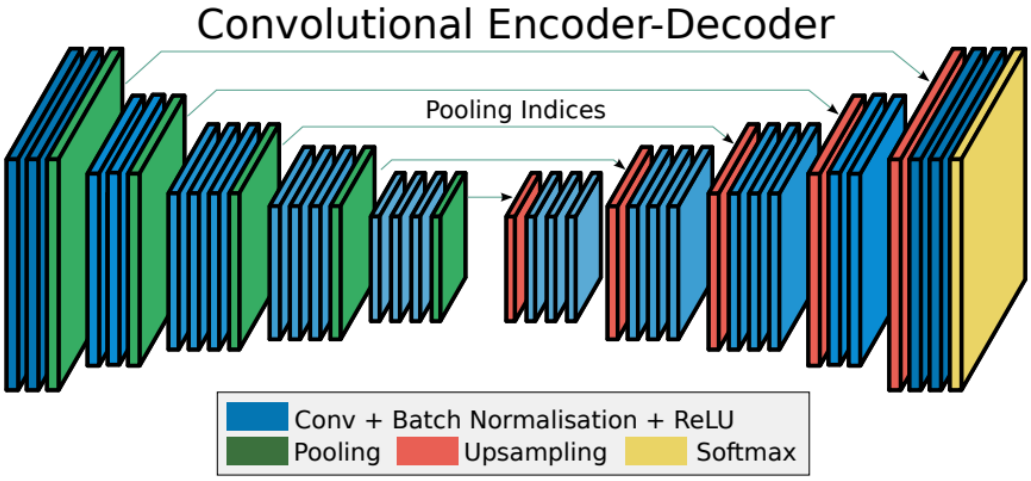
Struktura ove arhitekture sastoji se od kodera, dekodera i sloja klasifikacije po pikselima. Uloga dekodera je preslikati mape značajki niske rezolucije iz kodera u mape značajki pune ulazne rezolucije za klasifikaciju po pikselima na kraju mreže. Novost koju *SegNet* uvodi je način na koji dekoder povećava uzorkovanje – konkretno, dekoder koristi indekse iz slojeva maksimalnog sažimanja odgovarajućeg kodera za izvođenje nelinearnog povećanja uzorkovanja. Način kopiranja indeksa vidljiv je na slici. 3.3.



Sl. 3.3. Kopiranje indeksa iz kodera u dekoder [12].

Detaljna struktura ove arhitekture prikazana je na slici 3.4. Prva dva bloka kodera sastoje se od dva kovolucijska sloja s normalizacijom uzorka (engl. *batch normalization*) i *ReLU* aktivacijskom funkcijom nakon kojih slijedi sloj maksimalnog sažimanja veličine 2x2 s korakom veličine 2 – ovo je ujedno i sloj iz kojeg se pamte indeksi. Naredna tri bloka imaju istu strukturu, ali s tri kovolucijska sloja. Simetrično tome, prva tri bloka dekodera se sastoje od sloja povećanja uzorkovanja i tri kovolucijska sloja sa normalizacijom uzorka i ReLU aktivacijskom funkcijom, a zadnja sva bloka su iste strukture, ali s dva kovolucijska sloja. Na kraju mreže se nalazi višeklasni

soft-max klasifikator koji zasebno klasificira svaki piksel. Izlaz iz posljednjeg sloja mreže je slika s onoliko kanala koliko ima klasa u segmentaciji, a predviđena segmentacija odgovara klasi s najvećom vjerojatnošću za svaki piksel.



Sl. 3.4. Struktura *SegNet* arhitekture [12].

Iako su na prvu vrlo slične, razlika između *SegNet* i *U-Net* arhitekture je u tome što kod *U-Net*-a, veze između kodera i dekodera predstavljaju prespojne veze koje kopiraju izlaz iz koderskih slojeva u dekoderske – što troši više memorije, a kod *SegNet*-a se kopiraju samo indeksi na kojima se dogodilo sažimanje, a ne cijeli izlaz. Ova ušteda memorije kod *SegNet*-a rezultira malim gubitkom točnosti, ali je još uvijek prikladna za praktične primjene. Još jedna razlika spram *U-Net*-a je u tome što je *U-Net* više-manje simetrična, a *SegNet* potpuno simetrična arhitektura.

3.3. *ENet* arhitektura

ENet je zamišljena kao efikasna neuronska mreža, a [13] tvrdi kako je brža od tada aktualnih modela te kako ima jednaku ili veću preciznost, uz značajno manji broj parametara i operacija s pomičnim zarezom po sekundi (engl. *FLOPs*). Optimizirana je za brzo zaključivanje uz visoku točnost. Ona je kreirana za mobilne uređaje i uređaje na baterije te za zadatke koji zahtijevaju malo kašnjenje – konkretno brže od 10 FPS, i zato je idealna za segmentaciju cestovnih scena u stvarnom vremenu. Također, za razliku od tada aktualnih arhitektura, *ENet* ne koristi nikakve korake naknadne obrade (engl. *post-processing*) jer bi oni pogoršali brzinu izvođenja ovog pristupa.

ENet arhitektura je u [13] predstavljena tablicom prikazanom na slici 3.5. Dakle, radi se o kompaktnoj arhitekturi s jednom početnom, tri koderske i dvije dekoderske faze. Početna faza sadrži samo jedan blok. Prva faza kodera ima pet blokova, dok druga i treća imaju istu strukturu,

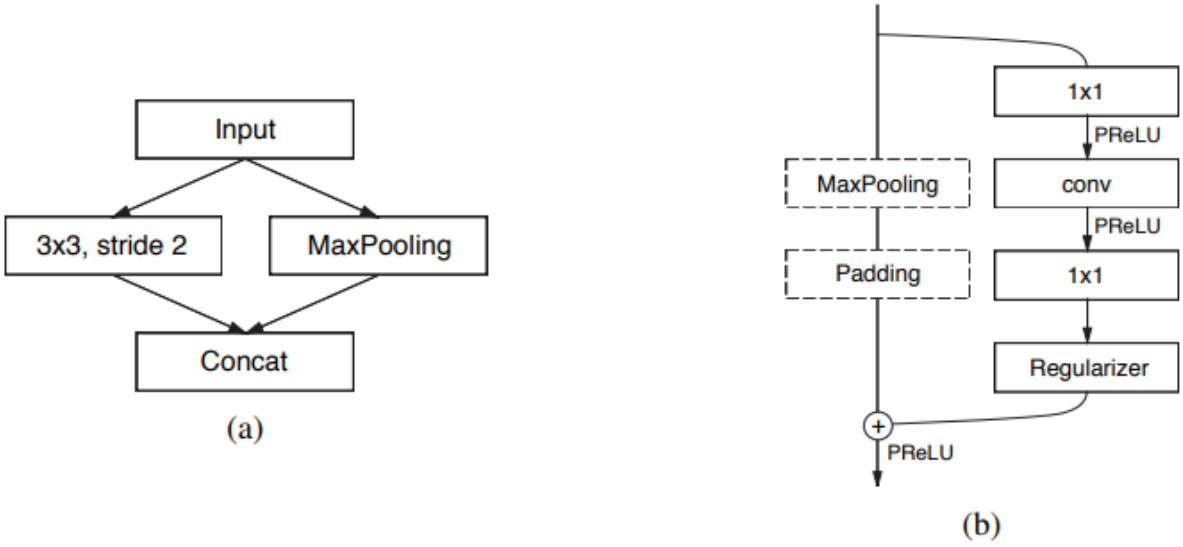
osim što treća ne smanjuje ulaz na početku izbacivanjem prvog bloka – dakle, druga faza ima 9, a treća 8 blokova – time završava koderski dio. Četvrta faza, ujedno i prva dekoderska faza, ima tri, a peta dva bloka. Posljednji blok mreže je jedna potpuna konvolucija, što značajno smanjuje vrijeme obrade dekodera.

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

Sl. 3.5. Struktura *ENet* arhitekture za ulaz veličine 512x512 [13].

Početni blok prikazan je na slici 3.6. – u njemu se na jednoj grani izvodi maksimalno sažimanje s nepreklapajućim prozorima veličine 2x2, a na drugoj se grani paralelno izvodi konvolucija veličine 3x3 s korakom veličine 2 koja ima 13 filtera, što daje 16 mapa značajki nakon spajanja ove dvije grane. Preostali blokovi ove mreže nazivaju se blokovi uskog grla (engl. *bottleneck*), a također su prikazani na slici 3.6. Svaki blok ima glavnu granu i granu s konvolucijskim filterima koje se razilaze i zatim spajaju sa zbrajanjem po elementu (engl. *element-wise addition*). Svaki blok uključuje tri konvolucijska sloja: 1) projekciju veličine 1x1 za smanjenje dimenzionalnosti, 2) glavni konvolucijski sloj i 3) proširenje veličine 1x1. Između svih konvolucija primjenjuju se normalizacija uzorka i PReLU (engl. *Parametric Rectified Linear Unit*) aktivacijska funkcija. Ako blok uskog grla izvršava smanjivanje uzorkovanja, glavnoj grani se dodaje sloj maksimalnog sažimanja, a prva projekcija veličine 1x1 zamjenjuje se konvolucijom veličine 2x2 s korakom veličine 2. Glavni konvolucijski sloj svakog bloka može biti pravilna, proširena ili potpuna konvolucija s filtrima veličine 3x3. Ponekad se ova konvolucija zamjenjuje asimetričnom

konvolucijom, odnosno nizom konvolucija veličine 5x1 i 1x5. Za regulator se koristi prostorno ispuštanje (engl. *Spatial Dropout*), sa stopom ispuštanja (engl. *dropout rate*) od 0,01 prije uskog grla broj 2.0 i 0,1 nakon njega.



Sl. 3.6. (a) Početni blok *ENet* mreže, (b) Struktura blokova uskog grla [13].

ENet vidljivo povlači dosta inspiracije iz *SegNet* arhitekture, no uz neke razlike. *ENet* nije simetrična mreža poput *SegNet*-a i *U-Net*-a, već ima dekoder manji od kodera. Također, *ENet* koristi *PReLU* aktivacijske funkcije za razliku od dosadašnjih *ReLU*, što je povećalo brzinu i točnost. Još jedan način kojim je *ENet* postigao veću brzinu je izvođenje konvolucije i sažimanja paralelno, a ne jedno nakon drugog.

4. IMPLEMENTACIJA RJEŠENJA

Implementacija opisanih arhitektura bit će provedena u programskom jeziku *Python* uz korištenje *PyTorch* bibilioteke. *PyTorch* je, prema [14], biblioteka za duboko učenje pomoću GPU-a i CPU-a, temeljena na biblioteci *Torch*. Koristi se, između ostalog, za računalni vid uz brojne druge primjene te je jedna od najpopularnijih biblioteka za strojno učenje jer nudi besplatan softver otvorenog koda.

Za treniranje i evaluaciju koristi se *Cityscapes* skup podataka, koji će biti detaljno opisan u idućem potpoglavlju.

4.1. Priprema skupa podataka

Cityscapes, opisan u [15] te dostupan na [16], je skup podataka koji se sastoji od 5000 precizno i 20000 grubo označenih slika urbanih uličnih scena – u ovom će se radu koristiti precizno označene slike. Skup se sastoji od 30 klasa, poput auta, ceste, pješaka i prometnih znakova. Uključuje slike iz 50 različitih europskih gradova, snimljenih tijekom proljeća, ljeta i jeseni za vrijeme dobrih vremenskih uvjeta i dnevnog svjetla. Segmentacije su ručno označene uz velik broj objekata u pokretu, različit raspored scene i promjenjivu pozadinu.

Prije korištenja skupa podataka, potrebno je poduzeti neke korake za pripremu. Kao prvo, već je spomenuto kako skup sadrži 30 klasa, no ne koriste se sve. Postoji 10 klasa koje se ne koriste u evaluaciji pa ih nikad nije ni potrebno predviđati. Ove će se klase u implementaciji smatrati jednom klasom imena *neoznačeno*. Zbog čitljivosti koda i lakšeg razumijevanja podataka, preostalim će se klasama promijeniti identifikacijski brojevi kako bi išle redom od 0 do 19, a ne uz preskakanja s nedostajućim brojevima tamo gdje su se nalazile nekorištene klase. Posljednja stvar koju je potrebno napraviti je pretvoriti crno-bijele slike u slike u boji s unaprijed definiranim bojama za svaku klasu.

Osim toga, potrebno je provesti neke korake obrade slika kako bi bile spremne za učinkovit trening u *PyTorch*u – ti koraci uključuju promjenu veličine, normalizaciju i pretvaranje podataka u format prikladan za trening. Ovi se koraci provode uz biblioteku albumentations (s aliasom *A*). Promjena veličine obavlja se funkcijom *A.Resize*(), koja kao parametre prima željenu visinu i širinu slike, a u ovom slučaju je to 320x640 piksela jer, prema [13], veličina ulazne slike dovoljna za analizu cestovne scene iznosi 360x640, no u ovom se slučaju ona reducira zbog održavanja omjera slika 1:2 iz skupa podataka. Promjena veličine uobičajeni je korak pripreme u zadacima računalnog vida kako bi se osiguralo da su sve slike jednake veličine prije nego što se unesu u neuronsku mrežu.

Normalizacija se obavlja funkcijom *A.Normalize*(), koja prima srednje vrijednosti i vrijednosti standardne devijacije. Vrijednosti korištene za normalizaciju - (0,485, 0,456, 0,406) za srednju vrijednost i (0,229, 0,224, 0,225) za standardnu devijaciju su već izračunate statističke vrijednosti izvedene iz skupa podataka *ImageNet*, dostupnog na [17], koje se često koriste u različitim zadacima računalnog vida. Normalizacija pomaže modelu da brže konvergira te pomaže u stabilizaciji procesa treninga jer osigurava da su ulazni podaci unutar standardiziranog raspona. Posljednji korak obrade pretvara slike u tenzore funkcijom *ToTensorV2*(). Tenzor je format koji očekuje većina okvira dubokog učenja, uključujući *PyTorch*, zato što *GPU*-ovi učinkovito upravljaju tenzorima, što čini računanja bržima tijekom treninga.

4.2. Implementacija *U-Net* algoritma

Za implementaciju ove i narednih arhitektura potrebno je uključiti biblioteke *torch* i *torch.nn* (s alisasom *nn*), a posebno za *U-Net* i *torchvision.transforms.functional* (s aliasom *TF*).

U-Net se u kodu implementira stvaranjem klase *UNet*() te njezinim kasnijim pozivanjem pri treningu. Pri pozivu klase bit će joj potrebno predati broj ulaznih i izlaznih kanala te listu značajki. Ovi se parametri postavljaju na zadane vrijednosti: 3 za broj ulaznih kanala – jer su slike u boji, 20 za broj izlaznih kanala – jer postoji 20 mogućih klasa i [64, 128, 256, 512] za listu značajki jer su to veličine kanala značajki redom u koracima *U-Net* arhitekture.

Za definiranje slojeva mreže prvo je definirana pomoćna klasa *DoubleConv*() kako bi se mogla višestruko pozivati uz manje dupliciranja koda. Ona kao parametre prima broj ulaznih i izlaznih kanala. Sastoji se od dva uzastopna konvolucijska sloja – *nn.Conv2d*(), nakon kojih slijedi normalizacija uzorka *nn.BatchNorm2d*() i *ReLU* aktivacijska funkcija *nn.ReLU*(). Ovi slojevi se nalaze u sekvencijalnom spremniku *nn.Sequential*() jer je to praktičan način za sekvencijalno organiziranje niza slojeva ili operacija unutar modula neuronske mreže.

Nadalje, definiraju se koder, dekoder i usko grlo mreže. Koder se sastoji od slijednih poziva klase *DoubleConv*() nakon kojih slijedi maksimalno sažimanje *nn.MaxPool2d*(). Dekoder ide u suprotnom smjeru – transponirane konvolucije *nn.ConvTranspose2d*(), koje služe u svrhu konvolucije prema gore, prethode slijednim pozivima klase *DoubleConv*(). Usko grlo, koje se nalazi između kodera i dekodera, definira se jednim pozivom klase *DoubleConv*() bez maksimalnog sažimanja. Na kraju mreže se nalazi završna konvolucija pozivom funkcije *nn.Conv2d*(). Također je definirana metoda prosljeđivanja (engl. *forward*), koja izvršava prolazak kroz mrežu, uz spremanje prespojnih veza iz kodera za kasnije spajanje s dekoderom. U ovu je svrhu definirana pomoćna funkcija *crop_to_match*() koja prilagođava veličinu mape značajki kako

bi odgovarala veličini prespojne veze prije spajanja, u slučaju da, zbog ulaznih dimenzija slike, već nisu jednake veličine.

4.3. Implementacija *SegNet* algoritma

Za *SegNet* je, osim prethodno spomenutih, potrebno uključiti i biblioteku *torch.nn.functional* (s alisaom *F*).

SegNet se također u kodu implementira stvaranjem klase *SegNet*() te njezinim kasnijim pozivanjem pri treningu. Kako se radi o vrlo sličnim arhitekturama, pri pozivu će joj također biti potrebno predati broj ulaznih i izlaznih kanala te listu značajki. Ovi se parametri postavljaju na jednake zadane vrijednost kao i kod *U-Net* arhitekture.

Za definiranje slojeva kodera koristi se već definirana pomoćna klasa *DoubleConv*() i nova klasa *TripleConv*() – ona je potpuno iste strukture, ali uz trostruko ponavljanje konvolucije, normalizacije uzorka i *ReLU* aktivacijske funkcije umjesto dvostrukog. Za dekoder se koriste klase *ReverseDoubleConv*() i *ReverseTripleConv*(), jer se *SegNet* drugačije nosi s prostornim dimenzijama spram *U-Net*a. Kod *SegNet*a u koderu prvi konvolucijski sloj smanjuje dimenzije kanala, a sljedeći slojevi zadržavaju tu dimenziju, dok u dekoderu prvi slojevi održavaju trenutni broj kanala, dok zadnji sloj prelazi na veću dimenziju.

Koder se definira dvostrukim pozivanjem klase *DoubleConv*() i trostrukim pozivanjem klase *TripleConv*() – iza svakog ovog poziva nalazi se *nn.MaxPool2d*() koji pamti indekse maksimalnog sažimanja. Nasuprot tome, dekoder prvo trostruko poziva klasu *ReverseTripleConv*(), zatim dvostruko *ReverseDoubleConv*(), a prije svakog poziva se nalazi *nn.MaxUnpool2d*() koji, osim smanjenja uzorkovanja, provodi spajanje sa spremljenim indeksima iz kodera. Na kraju mreže nalazi se jedan poziv *softmax* klasifikatora *F.softmax*().

4.4. Implementacija *ENet* algoritma

Za implementaciju *ENet* arhitekture potrebno je uključiti jednake biblioteke kao za *SegNet*.

Kao i ostale, *ENet* se implementira stvaranjem istoimene klase za kasnije pozivanje u kodu uz predaju broja ulaznih i izlaznih kanala.

Za definiciju slojeva ne mogu se koristiti prethodno definirane pomoćne klase, već se stvaraju nove – *InitialBock*() i *Bottleneck*(). *InitialBlock*(), kao što ime govori, definira početni blok mreže. U njemu se odvija spajanje dviju grana – grane maksimalnog sažimanja *nn.MaxPool2d*() i konvolucijske grane *nn.Conv2d*(). Nakon spajanja, primjenjuju se normalizacija uzorka

nn.BatchNorm2d() i *PReLU* aktivacijska funkcija *nn.PReLU*(). Klasa *Bottleneck*() opisuje sve preostale blokove mreže – blokove uskog grla. Ona prima brojne parametre koji joj govore je li riječ o povećavanju ili smanjivanju uzorkovanja, kakva se konvolucija koristi – pravilna, proširena ili potpuna te kolika je stopa ispuštanja. U blokovima uskog grla se također vrši spajanje dviju grana – u granama se vrše provjere o kojem je uzorkovanju i kojoj konvoluciji riječ te se one primjenjuju po potrebi. Bitno je naglasiti da *nn.MaxPool2d*() pamti indekse sažimanja te da se iza svake konvolucije primjenjuju normalizacija uzorka *nn.BatchNorm2d*() i *PReLU* aktivacijska funkcija *nn.PReLU*(). Na kraju grane vrši se regularizacija pozivom *nn.Dropout2d*() uz predaju parametra stope ispuštanja. Pri spajanju grana ponovno se poziva *PReLU* aktivacijska funkcija *nn.PReLU*().

ENet() klasa sastoji se od jednog poziva *InitialBlock*() klase i zatim slijednih poziva *Bottleneck*() klase kako bi se pratio redoslijed blokova sa slike 3.5, odnosno kako bi se stvorili koder i dekoder. Zadnji korak ove arhitekture je jedna potpuna konvolucija koja se ostvaruje pozivom *nn.ConvTranspose2d*().

4.5. Implementacija procesa treninga

Implementacija procesa treninga oslanja se na biblioteku *pytorch_lightning*, a osim nje korištene su i *segmentation_models_pytorch* (s aliasom *smp*), *torch.utils.data*, *multiprocessing*, *torchmetrics* i *time*.

Za trening je prvo potrebno definirati klasu *Model*() – u njoj se poziva *UNet*(), *SegNet*() ili *Enet*() klasa kako bi se odabrala arhitektura koja će se koristiti za trening. U njoj se također nalaze hiperparametri – stopa učenja, veličina serije i broj radnih niti – do ovih vrijednosti se dolazi empirijski za svaku arhitekturu. Zbog ograničenja hardvera i mogućnosti treniranja samo na malim veličinama serije, dodatno je implementirano i korištenje akumulacije gradijenta (engl. *gradient accumulation*). Prema [18], ono omogućava gomilanje gradijenata u više mini serija (engl. *mini-batch*) prije ažuriranja težina u modelu, a time se postiže simuliranje veće veličine serije dok stvarna veličina ostaje mala – u ovom slučaju maksimalna koliku hardver može podnijeti.

Nadalje, definiraju se funkcija gubitka i metrike procjene – za funkciju gubitka koristi se *smp.losses.DiceLoss*(), a za metriku procjene *torchmetrics.JaccardIndex*() i *torchmetrics.classification.MulticlassAccuracy*(). *Dice* koeficijent je često korištena metrika za procjenu rezultata segmentacije, a modificiran je da se koristi i kao funkcija gubitka, prema [19]. *Jaccard Index*, objašnjen u [20], je omjer presjeka i unije (engl. *Intersection over Union – IoU*), i on predstavlja primarnu metriku za procjenu točnosti modela segmentacije, prema [21]. *IoU* se

računa prema formuli (4.1.), gdje *TP* predstavlja istinito pozitivna, *FP* lažno pozitivna, a *FN* lažno negativna predviđanja. Ova se metrika najbolje razumije iz vizualizacije prikazane na slici 4.1. S obzirom na to da *IoU* zapravo računa vrijednost preklapanja za pojedinu klasu, a ovaj se rad bavi višeklasnom segmentacijom, potrebno je računati srednju vrijednost za sve klase pa se time dobiva *mIoU* (engl. *mean Intersection over Union*). *mIoU* će se koristiti kao glavna metrika za usporedbu rezultata, a uz nju se prati i točnost piksela (engl. *pixel accuracy*). Točnost piksela je omjer točno predviđenih piksela i ukupnog broja piksela, prema [22], a ona se neće koristiti kod evaluacije rezultata već samo za praćenje tijeka treninga.

Sl. 4.1. Vizualizacija *IoU* metrike [21].

$$IoU = \frac{TP}{TP + FP + FN} \tag{4-1}$$

Zatim, u *Model* klasi se definira spremanje kontrolnih točaka (engl. *checkpoint*) i rano zaustavljanje kako bi se omogućilo spremanje najboljih modela za evaluaciju ili nastavak treninga te kako bi se trening zaustavio na vrijeme, odnosno prije pretjeranog usklađivanja podacima za trening (engl. *overfitting*). Za optimizator se koristi *torch.optim.AdamW*() te uz njega i upravljač stopom učenja *torch.optim.lr_scheduler.ReduceLROnPlateau*(). U ovoj je klasi također definirano učitavanje skupova podataka za trening i validaciju te koraci treninga i validacije koji nakon svake epohe ispisuju svoj *mIoU*, točnost piksela i gubitak, što je korisno za praćenje treninga kako bi se mogli konfigurirati parametri po potrebi.

Nakon definicije klase stvara se njezina instanca pod imenom *model* te instanca klase *Trainer*() imena *trainer* te se poziva funkcija *trainer.fit(model)* za početak treninga. Prije i nakon poziva ove funkcije poziva se *time.perf_counter*() kako bi se izračunalo vrijeme trajanja treninga.

5. EKSPERIMENTALNI REZULTATI

Trening sva tri modela odvija se u potpuno istom okruženju, a ono se sastoji od *NVIDIA Quadro P2200* grafičke kartice s 5 GB *VRAM*a, *Intel Core i7-10700* procesora, 32 GB *RAM*a i *SSD* pohrane.

Pokretanjem naredbe *trainer.fit(model)* iz prethodnog poglavlja pokreće se proces treninga te se dobiva ispis sažetka arhitekture modela zajedno s veličinom i brojem parametara. Navedeni podaci za svaku od arhitektura vidljivi su u tablici 5.1.

Tablica 5.1. Usporedba veličine arhitektura.

	Broj parametara [u milijunima]	*Veličina modela [MB]*
U-Net	31	124.155
SegNet	29.4	117.787
ENet	0.358	1.434

Još jedna bitna stavka koja daje uvid u kompleksnost arhitektura je korištenje resursa – ono je prikazano u tablici 5.2. U tablici se, osim korištenja resursa, prikazuje i veličina serije korištena pri treniranju svake od arhitektura. Prema [23], veličina serije predstavlja broj uzoraka – u ovom slučaju slika, koje model istovremeno obrađuje u jednom prolazu kroz mrežu. Ona se navodi u tablici jer izravno utječe na korištenje resursa – što je ona veća, veći su zahtjevi za GPU. Prikazane veličine serije su najveće koje GPU može podnijeti za svaku arhitekturu, što pruža uvid u zahtjeve za resursima svakog modela pod istim hardverskim uvjetima.

Tablica 5.2. Usporedba korištenja resursa pri treningu.

	Veličina serije	*GPU [GB]*	*CPU*	*RAM [GB]*
U-Net	2	4.4 (88%)	33%	31.1 (97%)
SegNet	2	4.1 (82%)	39%	22.5 (70%)
ENet	4	3.2 (64%)	34%	22.6 (71%)

Rano zaustavljanje, objašnjeno i implementirano u prethodnom poglavlju, omogućava zaustavljanje treninga na vrijeme, odnosno nakon dovoljno treninga, a prije pretjeranog usklađivanja podacima za trening. Modeli završavaju trening s rezultatima prikazanim u tablici 5.3.

Tablica 5.3. Usporedba treninga.

	Trajanje treninga [sati:minute:sekunde]	*Brzina (po epohi) [minute:sekunde]*	*Broj epoha*	*mIoU treninga*	*Validacijski mIoU*	*Točnost piksela na treningu*	*Točnost piksela na validaciji*
U-Net	109:39:00	25:48	255	78%	60%	98%	95%
SegNet	27:36:00	20:42	80	64%	59%	96%	94%
ENet	15:14:15	2:39	345	59%	50%	96%	93%

Nakon svih završenih treninga, provodi se provjera rezultata na testnom skupu podataka. Kako je korišteni skup podataka zapravo kreiran za usporednu analizu (engl. *benchmarking*), testni dio skupa sadržava samo originalne slike, a ne i njihove segmentacije – zbog toga nije moguće kvantitativno odrediti točnost modela. Umjesto toga, mjeri se vrijeme procesa evaluacije testnog skupa kako bi se dobilo srednje vrijeme zaključivanja po slici iz kojeg se računa FPS koji dalje daje informaciju o tome je li arhitektura izvediva u stvarnom vremenu ili ne. Kao što je prethodno navedeno – model je izvediv u stvarnom vremenu ako se izvodi brzinom većom ili jednakom 10 FPS. Sve navedene metrike mogu se očitati u tablici 5.4.

Tablica 5.4. Usporedba rezultata.

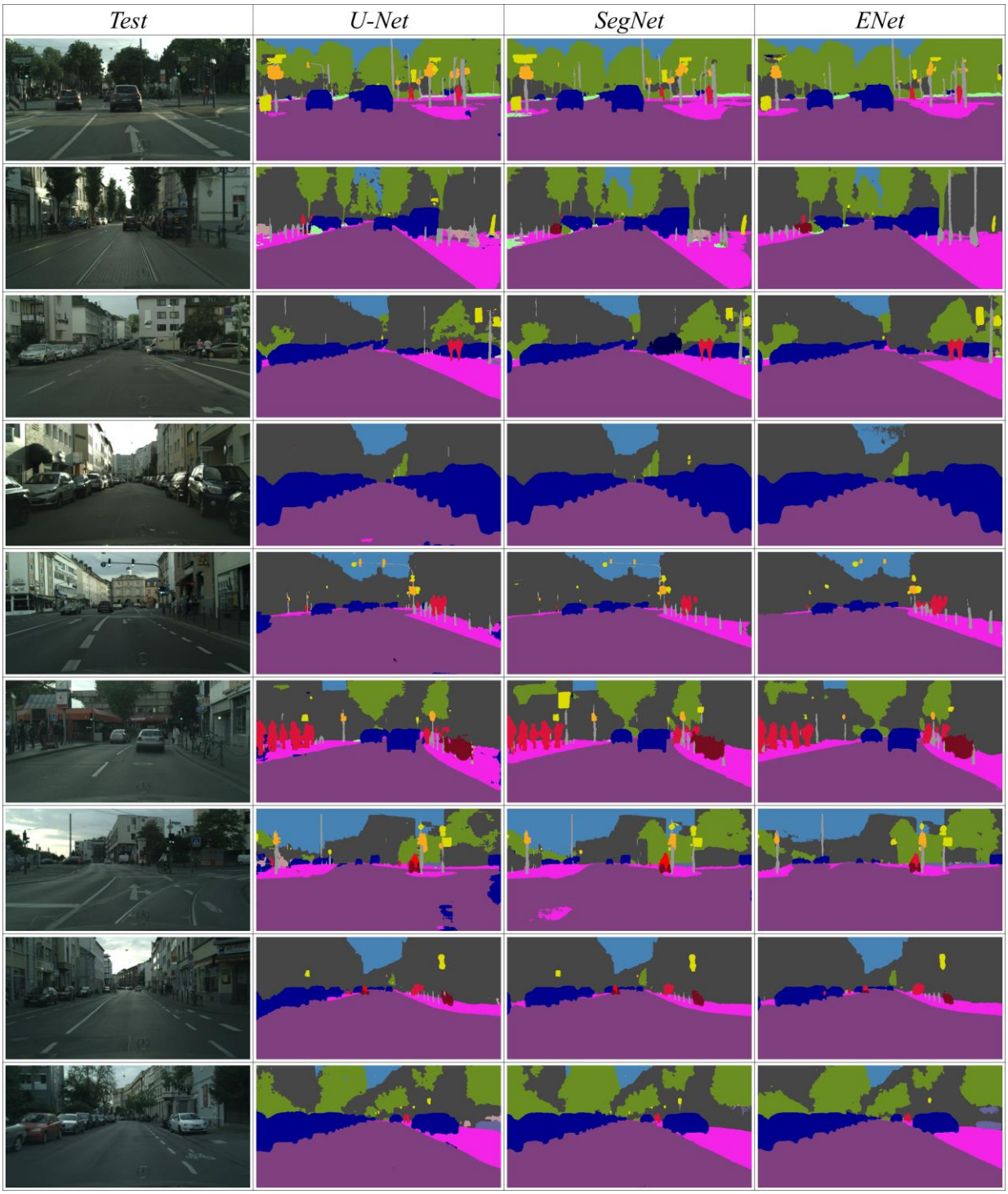
	Vrijeme zaključivanja [s/po slici]	*FPS*	*Izvedivo u stvarnom vremenu*
U-Net	0.32	3.1	Ne
SegNet	0.29	3.29	Ne
ENet	0.19	5.04	Ne

Kako se nijedna od arhitektura nije pokazala izvedivom u stvarnom vremenu na korištenom hardveru, nadalje se mjeri korištenje resursa pri zaključivanju. Ovo se mjerenje provodi kako bi se bolje razumjela njihova praktična učinkovitost te prikladnost za primjenu u okruženjima s ograničenim računalnim resursima. U ovom se slučaju koristi veličina serije 1 za svaku od arhitektura jer je to očekivano ponašanje u stvarnoj primjeni – obrađivanje jedne po jedne slike.

Tablica 5.2. Usporedba korištenja resursa pri zaključivanju.

	Veličina serije	*GPU [GB]*	*CPU*	*RAM [GB]*
U-Net	1	3.9 (78%)	51%	12.3 (38%)
SegNet	1	3.3 (66%)	23%	12.4 (39%)
ENet	1	1.3 (26%)	38%	12.4 (39%)

Na slici 5.1. prikazani su rezultati svake arhitekture u usporedbi s originalnim slikama iz testnog skupa podataka, a na slici 5.2. u usporedbi s točno označenim segmentacijama iz validacijskog skupa.



Sl. 5.1. Usporedba rezultata sa slikama iz testnog skupa.

Sl. 5.2. Usporedba rezultata sa segmentacijama iz validacijskog skupa.

Iz svih navedenih rezultata može se primijetiti sljedeće – *ENet* je, kao što je i bilo očekivano, fizički puno manja i kompaktnija arhitektura od *U-Net*a i *SegNet*a – i po fizičkoj veličini i po broju parametara, a uz to je i puno brža i pri treniranju i pri zaključivanju – ona se najviše bliži izvođenju u stvarnom vremenu. Ovdje je važno naglasiti da, iako na korištenom hardveru nijedna od

arhitektura nije postigla brzinu izvođenja u stvarnom vremenu, ne znači da to nije moguće za svaku od njih na sofisticiranijem hardveru. Nadalje, *ENet* troši znatno manje resursa od preostale dvije arhitekture, ali postiže nešto lošije rezultate. *U-Net* i *SegNet* pokazuju slične performanse u svim metrikama.

Za kraj, može se zaključiti kako odabir idealne arhitekture u konačnici ovisi o svrsi u kojoj će se koristiti. Za slučajeve u kojima su kompaktnost, brzina i učinkovitost resursa najvažniji, idealno bi bilo koristiti *ENet*, ali uz daljnja poboljšanja za veću točnost. Nasuprot, ako je održavanje visoke točnosti kritično, a ograničenja resursa su nešto što se može kompenzirati, *U-Net* ili *SegNet* bi bili bolji izbor. Ovo je posebno važno u primjenama kao što je autonomna vožnja, gdje je točnost ključna za situacije osjetljive na sigurnost i gdje bi ulaganje u skuplji hardver značilo veću sigurnost na cesti. U takvim slučajevima, davanje prioriteta točnosti segmentacije u odnosu na učinkovitost resursa ključno je za izbjegavanje ugrožavanja sigurnosti u prometu.

6. ZAKLJUČAK

U ovom radu napravljena je usporedba rada tri arhitekture konvolucijske neuronske mreže za segmentaciju cestovnih scena iz perspektive automobila – *U-Net, SegNet* i *ENet*. Na početku, objašnjeno je što je to segmentacija slike i koje su njezine vrste, te je semantička segmentacija odabrana kao ona koja će se koristiti u radu. Nadalje, predstavljene su najaktualnije metode rješenja problema segmentacije, uz detaljnu teorijsku i implementacijsku podlogu o tri korištene arhitekture. Opisana je i implementacija treninga provedena uz pomoć biblioteke *PyTorch* na skupu podataka *Cityscapes* – koji je detaljno opisan uz objašnjenja koraka koji se moraju poduzeti za obradu slika prije primjene na treningu. Modeli arhitektura su istrenirani u istom okruženju, uz mjerenje vremena treninga, brzine treniranja, broja epoha, srednje vrijednosti omjera presjeka i unije te točnosti piksela. Modeli su osim toga uspoređeni po broju parametara, fizičkoj veličini, korištenju resursa pri treningu, a nakon treninga uspoređeni su po brzini zaključivanja i mogućnosti izvođenja u stvarnom vremenu te po korištenju resursa pri zaključivanju. Modeli su istrenirani uz određena hardverska ograničenja, tako da ostavljaju prostora za poboljšanje – i u smislu točnosti, a pogotovo u smislu brzine izvođenja, no daju vizualno zadovoljavajuće rezultate. Nakon usporedbe, zaključeno je kako ne postoji najbolja od tri arhitekture, već izbor idealne ovisi o svrsi pa se tako *ENet* bira u slučajevima kada je najvažnije osigurati brzinu i učinkovitost resursa, a *U-Net* i *SegNet* su, zbog vrlo bliskih rezultata i performansi, odabrane za slučajeve kada je točnost ključna, a iskorištenost resursa nešto što se može žrtvovati.

LITERATURA

[1] H. Asqiriba, G. Sultan, „Image Segmentation Techniques“, *ResearchGate*, pro. 2020., https://www.researchgate.net/publication/347564769_IMAGE_SEGMENTATION_TECHNIQUES

[2] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, D. Terzopoulos, „Image Segmentation Using Deep Learning: A Survey“, stu. 2018., https://doi.org/10.48550/arXiv.2001.05566

[3] F. Li, R. Krishna, D. Xu, „Detection and Segmentation“, svi. 2021., dostupno na: https://cs231n.stanford.edu/slides/2021/lecture_15.pdf [5.6.2024.]

[4] „What is image segmentation“, *IBM*, ruj. 2023., dostupno na: https://www.ibm.com/topics/image-segmentation [5.6.2024.]

[5] A. Kirillov, K. He, R. Girshick, C. Rother, P. Dollár, „Panoptic Segmentation“, tra. 2019., https://doi.org/10.48550/arXiv.1801.00868

[6] H. Zhao, J. Shi, X. Qi, X. Wang, J. Jia, „Pyramid Scene Parsing Network“, tra. 2017., https://doi.org/10.48550/arXiv.1612.01105

[7] J. Long, E. Shelhamer, T. Darrell, „Fully Convolutional Networks for Semantic Segmentation“, ožu. 2015., https://doi.org/10.48550/arXiv.1411.4038

[8] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. L. Yuille, „DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs“, svi. 2017., https://doi.org/10.48550/arXiv.1606.00915

[9] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, B. Xiao, „Deep High-Resolution Representation Learning for Visual Recognition“, ožu. 2020., https://doi.org/10.48550/arXiv.1908.07919

[10] T. Takikawa, D. Acuna, V. Jampani, S. Fidler, „Gated-SCNN: Gated Shape CNNs for Semantic Segmentation“, srp. 2019., https://doi.org/10.48550/arXiv.1907.05740

[11] O. Ronneberger, P. Fischer, T. Brox, „U-Net: Convolutional Networks for Biomedical Image Segmentation“, svi. 2015., https://doi.org/10.48550/arXiv.1505.04597

[12] V. Badrinarayanan, A. Kendall, R. Cipolla, „SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation“, lis. 2016., https://doi.org/10.48550/arXiv.1511.00561

[13] A. Paszke, A. Chaurasia, S. Kim, E. Culurciello, „ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation“, lip. 2016., https://doi.org/10.48550/arXiv.1606.02147

[14] „PyTorch Documentation“, *PyTorch*, 2023., dostupno na: https://pytorch.org/docs/stable/index.html [8.6.2024.]

[15] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, „The Cityscapes Dataset for Semantic Urban Scene Understanding“, tra. 2016., https://doi.org/10.48550/arXiv.1604.01685

[16] „Dataset overview“, *Cityscapes Dataset*, lis. 2020., dostupno na: https://www.cityscapes-dataset.com/dataset-overview/ [8.6.2024.]

[17] „ImageNet“, *Stanford Vision Lab, Stanford University, Princeton University*, ožu. 2021., dostupno na: https://image-net.org/index.php [14.6.2024.]

[18] R. Rotenberg, „What is Gradient Accumulation in Deep Learning?“, *Towards Data Science*, sij. 2020., dostupno na: https://towardsdatascience.com/what-is-gradient-accumulation-in-deep-learning-ec034122cfa [13.9.2024.]

[19] S. Jadon, „A survey of loss functions for semantic segmentation“, *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, str. (1-7), Via del Mar, Čile, 2020., https://doi.org/10.48550/arXiv.2006.14822

[20] „Jaccard Index: Module Interface“, *Lightning AI*, dostupno na: https://lightning.ai/docs/torchmetrics/stable/classification/jaccard_index.html [20.6.2024.]

[21] Kukil, „Intersection over Union (IoU) in Object Detection & Segmentation: Intersection over Union in Image Segmentation“, *Big Vision LLC*, lip. 2022., dostupno na: https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/ [20.6.2024.]

[22] „Accuracy: Module Interface“, *Lightning AI*, dostupno na: https://lightning.ai/docs/torchmetrics/stable/classification/accuracy.html [13.9.2024.]

[23] Devansh, „How does Batch Size impact your model learning“, *Medium*, sij. 2022., dostupno na: https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa [17.9.2024.]

OPTIMIZACIJA SEGMENTACIJE CESTOVNIH SCENA U STVARNOM VREMENU

SAŽETAK

Cilj ovog diplomskog rada bio je implementirati *U-Net*, *SegNet* i *ENet* arhitekture te njihov trening uz pomoć *PyTorch* biblioteke, a dobivene rezultate koristiti za usporedbu navedenih arhitektura po pitanju točnosti, brzini izvođenja, učinkovitosti u stvarnom vremenu i kompleksnosti implementacije. Trening je proveden na *Cityscapes* skupu podataka, na kojem je provedena predobrada potrebna za korištenje pri treningu. Prije treninga, arhitekture su uspoređene po veličini i broju parametara; tijekom treninga, prate se srednja vrijednost unije i presjeka te točnost piksela; a nakon treninga uspoređuju se po trajanju i brzini treniranja, broju epoha te se provodi testiranje na testnom dijelu skupa podataka. Pri testiranju se mjere vrijeme zaključivanja i mogućnost izvođenja u stvarnom vremenu, a točnost se mjeri kvalitativno, pošto ne postoji izvor istine za segmentacije testnog skupa. Navedenim usporedbama pružen je dubok uvid u primjenjivost navedenih arhitektura u kontekstu automobilske percepcije te je svakoj arhitekturi pronađena prikladna primjena na temelju njezinih rezultata i performansi – *ENet* je izabrana za zadatke u kojima su ključne brzina i učinkovitost resursa, a *U-Net* i *SegNet* se, zbog svojih sličnih rezultata, predlažu za istu primjenu, a to su situacije u kojima je točnost bitnija od iskorištenosti resursa.

Ključne riječi: duboko učenje, računalni vid, segmentacija cestovnih scena, segmentacija slike

OPTIMIZATION OF SEGMENTATION OF ROAD SCENES IN REAL TIME

ABSTRACT

The aim of this thesis was to implement U-Net, SegNet and ENet architectures along with their training with the help of the PyTorch library, and using the obtained results to compare the mentioned architectures in terms of accuracy, execution speed, real-time efficiency and implementation complexity. The training was carried out on the Cityscapes dataset, which was pre-processed for use in training. Before the training, the architectures were compared according to their size and number of parameters; during the training the mean Intersection over Union and the pixel accuracy were monitored; after the training, they were compared according to the duration and speed of the training, the number of epochs, and testing was carried out on the test part of the data set. During testing, inference time and real-time-execution capabilities were measured quantitatively, and accuracy was measured qualitatively, since there is no ground truth for the test set segmentations. The aforementioned comparisons provided deep insight into the applicability of the architectures in the context of automotive perception, and a suitable task was found for each architecture based on its results and performance - ENet was chosen for tasks where speed and resource efficiency are key, while U-Net and SegNet are suggested for the same tasks due to their similar results – tasks in which the accuracy is more important than resource utilization.

Key words: deep learning, computer vision, road scene segmentation, image segmentation

ŽIVOTOPIS

Autor ovog rada, bacc. ing. comp. Renata Barišić, rođena je 7.3.2001. u Osijeku. U Osijeku je pohađala osnovnu školu „Mladost“, srednju školu – III. gimnaziju Osijek i preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Za vrijeme pisanja ovog rada, na istom fakultetu studira na drugoj godini diplomskog studija računarstva na izbornom bloku Informacijske i podatkovne znanosti.

_____

Potpis autora