Segmentacija voznog područja iz slike dobivene kamerom montiranom na prednjoj strani vozila

Ćaleta, Dunja

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

SEGMENTACIJA VOZNOG PODRUČJA IZ SLIKE DOBIVENE KAMEROM MONTIRANOM NA PREDNJOJ STRANI VOZILA

Diplomski rad

Dunja Ćaleta

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:	Dunja Ćaleta
Studij, smjer:	Sveučilišni diplomski studij Automobilsko računarstvo i
Mat. br. pristupnika, god.	D-67 ARK, 07.10.2022.
JMBAG:	0165082713
Mentor:	prof. dr. sc. Mario Vranješ
Sumentor:	doc. dr. sc. Denis Vranješ
Sumentor iz tvrtke:	Zvonimir Kaprocki
Predsjednik Povjerenstva:	izv. prof. dr. sc. Ratko Grbić
Član Povjerenstva 1:	prof. dr. sc. Mario Vranješ
Član Povjerenstva 2:	prof. dr. sc. Marijan Herceg
Naslov diplomskog rada:	Segmentacija voznog područja iz slike dobivene kamerom montiranom na prednjoj strani vozila
Znanstvena grana diplomskog rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Segmentacija voznog područja je osnovni zadatak u okviru autonomne vožnje, pri čemu se u slici dobivenoj s kamere montirane na prednjoj strani vozila lokalizira područje koje je pogodno za vožnju. Ovo je ključno za autonomna vozila jer se dobivene informacije koriste za navigaciju i izbjegavanje prepreka. Lokalizacija voznog područja u slici je problem semantičke segmentacije, za što su danas na raspolaganju moderni algoritmi zasnovani na dubokim neuronskim mrežama. U okviru ovog rada potrebno je analizirati postojeća suvremena riešenja za navedeni
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	26.08.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	30.09.2024.
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:	30.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Dunja Ćaleta
Studij:	Sveučilišni diplomski studij Automobilsko računarstvo i komunikacije
Mat. br. Pristupnika, godina upisa:	D-67 ARK, 07.10.2022.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom**: Segmentacija voznog područja iz slike dobivene kamerom montiranom na prednjoj strani vozila**

izrađen pod vodstvom mentora prof. dr. sc. Mario Vranješ

i sumentora doc. dr. sc. Denis Vranješ

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

Iz Nacionalnog plana sigurnosti cestovnog prometa Republike Hrvatske [1] vidljivo je kako je čovjek potencijalni uzrok 57% teških prometnih nesreća u Republici Hrvatskoj te da je u čak 59% teških prometnih nesreća neoprezna vožnja jedan od potencijalnih uzroka, dok je u 38% teških prometnih nesreća neoprezna vožnja glavni uzrok. Osobna vozila i dalje u najvećem broju sudjeluju u prometnim nesrećama. Prema nacionalnom planu koji obuhvaća razdoblje od 2021. do 2030. godine cilj je povećati udio novih osobnih vozila s EuroNCAP sigurnosnom ocjenom jednakom ili većom od utvrđenog praga te smanjiti udio tehnički neispravnih vozila tijekom redovitih tehničkih pregleda.

Radi smanjenja broja prometnih nesreća, automobilska industrija usmjerila se na razvoj sustava za pomoć vozačima (engl. *Advanced Driver-Assistance Systems* - ADAS) te samim time i na poboljšanje autonomne vožnje. ADAS osigurava povećanu sigurnost sudionika prometa brzim prepoznavanjem opasnosti korištenjem brojnih podsustava, od kojih su neki od njih zaduženi za npr. upozorenje na sudar, automatsko kočenje, praćenje prometa i sl. Određeni ADAS mogu pridonijeti udobnosti vožnje poput npr. adaptivnog tempomata ili olakšati vožnju asistencijom prilikom parkiranja te asistencijom pri održavanju vozne trake.

Uz detekciju, klasifikaciju i praćenje objekata, osnovu autonomne vožnje čini i segmentacija voznog područja koja lokalizira pogodno vozno područje iz slika snimljenih kamerom montiranom na prednjoj strani vozila. Lokalizacija voznog područja u današnjim se vozilima rješava primjenom semantičke segmentacije koja svakom pikselu slike dodjeljuje određenu oznaku, odnosno klasu. Ukoliko na ulaznoj slici postoji desetak pješaka i jednako toliko vozila, svi pikseli koji pripadaju pješacima bit će grupirani u jednu klasu, a svi pikseli koji pripadaju vozilima u drugu. Rezultat semantičke segmentacije je primijenjena maska na ulaznu sliku kod koje se svaka klasa objekta označava odgovarajućom bojom. Potencijalni problemi koji se mogu pojaviti kod razvoja ovog sustava su nepovoljni vremenski uvjeti, slaba vidljivost, noćna vožnja te ostali uvjeti koji otežavaju proces učenja algoritma semantičke segmentacije. Nakon prikupljanja slika s kamere montirane na prednjoj strani vozila iste se obrađuju različitim tehnikama i algoritmima koji rezultiraju segmentiranom voznom trakom. Takve algoritme moguće je podijeliti na one zasnovane na klasičnim tehnikama računalnog vida i one zasnovane na neuronskim mrežama. Korištenjem klasičnih tehnika računalnog vida cijeli proces se dijeli na manje podzadatke te se svaki od njih zasebno rješava. Jedan od načina obrade kod problema

semantičke segmentacije započinje određivanjem područja od interesa na ulaznoj slici po x i y osi. Nakon što je definirano područje interesa, slika se prebacuje u jednobojnu sliku tonova sive boje (engl. *grayscale*). Optimalnim se često pokazalo i smanjenje šuma te zamagljivanje primjenom Gaussovog filtra. Slijedi obično primjena Canny operatora za detekciju rubova (iako postoje i druge metode). Nakon detekcije rubova, određeno područje interesa kopira se na crnu masku koja je jednakih dimenzija kao i ulazna slika. Crna maska služi kao podloga koja će omogućiti izolaciju i analizu specifičnih dijelova slike, dok ostali dijelovi ostaju crni. Proces se privodi kraju korištenjem Houghove transformacije i određivanjem praga koji se koristi za filtriranje linija otkrivenih Houghovom transformacijom, rezultirajući prikazom samo onih linija koje zadovoljavaju određeni kriterij. Kao rezultat dobiva se prikaz linija na crnoj masci te prikaz linija na ulaznoj slici [2]. Kod algoritama zasnovanih na neuronskim mrežama odvija se takozvani *end-to-end* pristup gdje je cijeli, ranije navedeni proces provođenja semantičke segmentacije, sadržan unutar neuronske mreže.

Ovim diplomskim radom predložen je poboljšani *end-to-end* algoritam semantičke segmentacije vozne trake koji procjenjuje voznu traku vozila uporabom duboke neuronske mreže korištenjem ulaznih slika dobivenih s kamerom montiranom na prednjoj strani vozila. Algoritam je napravljen pomoću *Python* programskog jezika uporabom *TensorFlow* i *Keras* biblioteka. Nakon dobivanja zadovoljavajućih performansi, algoritam je implementiran na ugradbenu računalnu platformu – Android pametni mobitel, na kojem se u stvarnom vremenu procjenjuje vozno područje iz video signala i prikazuje ga se na ekranu mobitela.

Drugo poglavlje rada sadrži pojašnjenje problema semantičke segmentacije i prikazuje pregled postojećih rješenja za semantičku segmentaciju voznog područja. Trećim poglavljem predstavljaju se biblioteke i programsko okruženje korišteno za razvoj vlastitog algoritma semantičke segmentacije. Analiziraju se dostupni skupovi podatka, predstavljaju se evaluacijske metrike te se odabrani algoritmi evaluiraju prema prethodno opisanim metodama. Četvrto poglavlje rada detaljno opisuje implementaciju odabranog algoritma na ugradbenu računalnu platformu (Android telefon) i evaluaciju istog. Na kraju rada iznesen je zaključak zasnovan na rezultatima testiranja.

2. PREGLED POSTOJEĆIH RJEŠENJA ZA SEGMENTACIJU VOZNOG PODRUČJA IZ SLIKE DOBIVENE KAMEROM MONTIRANOM NA PREDNJOJ STRANI VOZILA

2.1. Problem segmentacije slike voznog područja iz slike dobivene kamerom montiranom na prednjoj strani vozila

Problem segmentacije digitalne slike podrazumijeva razlaganje slike na nekoliko dijelova, odnosno skupova piksela koji formiraju određeni objekt slike. Najčešća uporaba segmentacije je u svrhu pronalaska objekata i granica od interesa [3]. Proces semantičke segmentacije slike podrazumijeva dodjeljivanje oznake (klase) svakom pikselu slike. Ovaj postupak može se podijeliti u nekoliko kategorija ovisno o metodi segmentacije.

Semantička segmentacija zasnovana na regiji započinje s prepoznavanjem regija na slici, nakon čega se izvodi postupak segmentacije. Tijekom faze testiranja, predikcije na razini kategorija postaju predikcije na razini piksela. Landrieu i Simonovsky [4] predložili su kombinaciju PointNet-a s grafičkim modelima i nenadziranom segmentacijom. Cijeli oblak točaka podijeljen je na geometrijski homogene regije. Uloga PointNet-a je dobivanje globalnih značajki za svaku od pojedinih regija. Izgradnja grafičkog modela vrši se modeliranjem regija u ulozi čvorova grafa na koje je primjenjiva grafička konvolucija. Po uzoru na Landrieu i Simonovsky, Contreras i Sickert [5] predlažu korištenje nenadzirane segmentacije radi dijeljenja scene u početnom koraku. Međutim, prikupljaju lokalne informacije o točkama u svakoj regiji putem dodatne podmreže. Također, koriste obuhvatno lokalno znanje koje je zasnovano na geometrijskim atributima susjednih regija putem rubnih konvolucija.

Kod semantičke segmentacije zasnovane na CNN (engl. *convolutional neural network*) ulazni podatak je slika proizvoljnih dimenzija, dok je rezultantna izlazna slika dimenzija jednakih ulaznoj. Propagiranjem kroz više konvolucijskih slojeva i slojeva združivanja (engl. *pooling layer*) moguće je dobiti izlazne mape značajki niske kvalitete, što dovodi do nejasnih granica objekata. Ideja proširivanja konvolucijske mreže na ulaze proizvoljne veličine prvi put se pojavila u radu Matan i sur. [6], koji su odlučili proširili klasični LeNet [7] za prepoznavanje nizova brojeva. Wolf i Platt [8] predstavili su širenje izlaza konvolucijskih mreža na 2-dimenzionalne mape. Shelhamer i Long [9] predstavljaju potpuno konvolucijske mreže (FCNs) koje se treniraju od početka do kraja (*end-to-end*), tj. od piksela do piksela, za semantičku segmentaciju. Ovi modeli predviđaju gusto označene izlaze iz proizvoljnih ulaza, omogućavajući

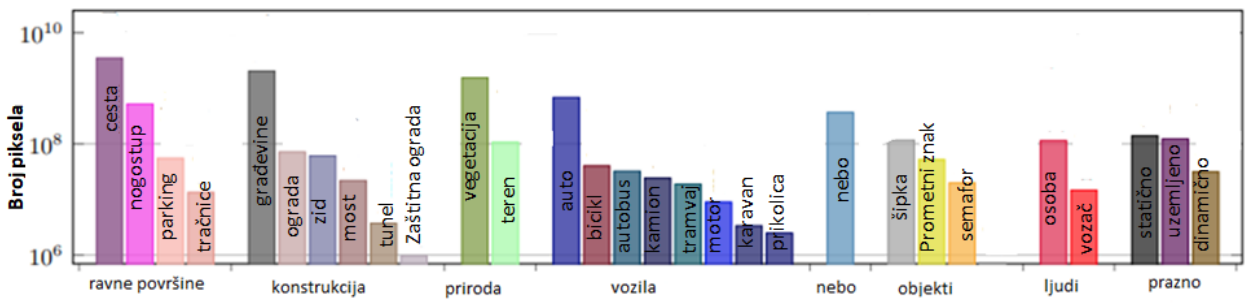
učenje i inferenciju za cijelu sliku odjednom. Također se koriste slojevi povećanja dimenzija (engl. *upsampling layers*) unutar mreže, koji omogućuju predviđanje na razini piksela i učenje u mrežama s poduzorkovanjem. Neke od arhitektura koje su zasnovane na potpuno povezanoj mreži su FPN, DeepLab, U-Net. FPN *(*engl. *Feature pyramid network*) [10] koristi piramidalnu hijerarhiju konvolucijskih mreža za stvaranje piramida značajki koje su u mogućnosti detektirati objekte različitih skala koristeći *top-down* arhitekturu za izradu kvalitetnih mapa značajki. Ulazni podatak je proizvoljne veličine, a izlaz čine proporcionalne mape značajki na svim skalama. DeepLab [11] je serija mreža kreirana od strane *Google*-a, dok je DeepLab3plus jedna od tih mreža koja sadrži kompleksnu arhitekturu mreže u kojoj su sadržani enkoder i dekoder. Enkoderski dio koristi veći konvolucijski sloj radi maksimiziranja receptivnog područja i povećanja sposobnosti izvlačenja značajki. Značajke niže razine ulaze u dekoder, a značajke više razine spojene su s ASPP modulom (engl. *Atrous Spatial Pyramid Pooling*) radi prikupljanja i povećanja semantičkih informacija slike. Značajke visoke i niske razine spajaju se u dekoderu te prolaze kroz niz konvolucijskih slojeva kako bi izlazna slika imala istu rezoluciju kao i ulazna. U-Net [12] se sastoji od kontrakcijskog i ekspanzivnog dijela. Kontrakcijski dio sastoji se od opetovane primjene konvolucije 3x3 nakon čega slijedi funkcija ReLU (engl. *Rectified Linear Unit*) i operacija maksimalnog združivanja (engl. *Max pooling*) 2x2 s korakom 2 za poduzorkovanje. Svakim korakom udvostručuje se broj značajki kanala. Koraci u ekspanzijskom putu uključuju uzorkovanje prema gore mapi značajki, uporabu konvolucije 2x2, kombiniranje s odgovarajućom skraćenom mapom značajki iz kontrakcijskog dijela te dvije konvolucije 3x3 od kojih je svaka praćena ReLU funkcijom. Skraćivanje se radi kako bi se spriječio gubitak rubnih piksela svake konvolucije. Završni sloj koristi konvoluciju 1x1 za dodjelu svakog vektora značajki od 64 komponente željenom broju klasa.

Slabo nadzirana semantička segmentacija (engl. *Weakly supervised semantic segmentation*) koristi oznake na razini slika te CAM (engl. *class activation map*) za pronalaženje značajki slike koje imaju ključnu ulogu za predikciju određene klase. Slike trening skupa podataka sadrže klase kojima pripadaju, ali ne i lokaciju objekata na slici. Aktivacijske mape obuhvaćaju gotovo cijele ciljane regije, pri čemu se neki dijelovi s nižim aktivacijskim vrijednostima lako zanemaruju. Kao rješenje takvog problema predstavljana je adaptivna mreža aktivacija s dvjema granama koje rekalibriraju regije s niskim stupnjem povjerenja na aktivacijskim mapama [13]. Grana za poboljšanje aktivacije dizajnirana je za redistribuciju vrijednosti aktivacije koristeći mehanizam pažnje za prilagodbu. S obzirom na to da slike višestruke skale pružaju dopunske informacije, grana skale paralelno nadzire granu poboljšanja aktivacije. Međusobno nadgledanje i fuzija

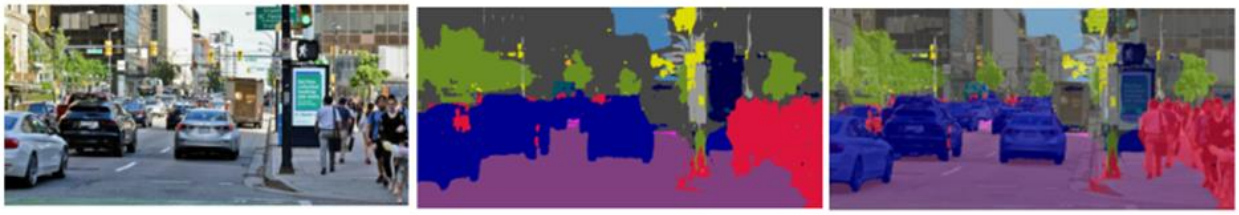
spomenutih dviju grana potiču manje diskriminativne dijelove i deaktiviraju pozadinske regije. Na temelju toga, predložen je jednostavan, ali učinkovit modul za smanjenje šuma kako bi se dodatno poboljšala kvaliteta lažnih maski koji koristi velike predikcije trenirane mreže segmentacije.

2.2. Analiza dostupnih skupova podataka za semantičku segmantaciju voznog područja

Cityscapes [14] je skup podataka koji se sastoji od stereo video sekvenci uličnih scena 50 različitih gradova. Sadrži visokokvalitetne anotacije na razini piksela i instance za 30 različitih klasa. Namijenjen je procjeni performansi algoritama vizije za semantička, instancijska i panoramska označavanja. Služi istraživanjima koja su usmjerena na iskorištavanje velike količine (slabo) anotiranih podataka kao što je treniranje dubokih neuronskih mreža. Prednosti Cityscapes skupa podataka su raznolik i reprezentativan uzorak urbanih scena prikupljen iz različitih gradova i različitih doba dana. Također, pruža detaljne i precizne anotacije na razini piksela i instanci za 30 klasa grupiranih u 8 kategorija te dodatni skup od 20 000 slika s grubim anotacijama, koji se može koristiti za prethodno treniranje ili nadzirano učenje. Omogućuje i stereo parove slika koji se mogu koristiti za procjenu dubine, 3D rekonstrukciju scene ili 3D detekciju objekata. Nedostatak Cityscapes skupa podataka je što su slike snimljene samo u urbanim sredinama i to jedino u europskim gradovima, što dovodi do smanjenja performansi algoritama koji se primjenjuju na slikama snimljenim u ruralnim ili otvorenim područjima. Ujedno, sadrži samo slike snimljene u dobrim vremenskim uvjetima, što može smanjiti generalizaciju algoritama u drugim scenarijima te zahtijeva ručno preuzimanje i registraciju za pristup datotekama, što može biti nezgodno ili ograničavajuće pojedinim korisnicima. Slika 2.1 prikazuje broj precizno označenih piksela po klasi s pridruženim kategorijama, dok su na slici 2.2 dani primjeri originalnih slika s pripadnim slikama s oznakama.

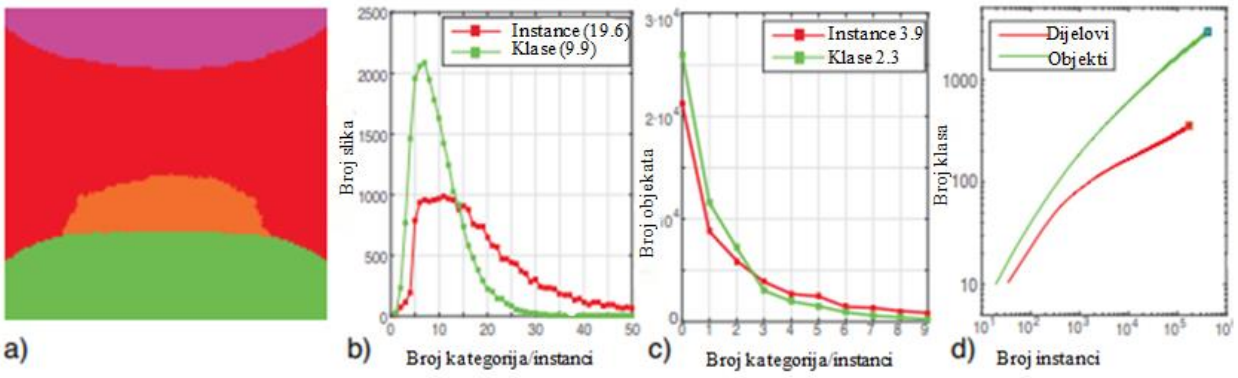


***Sl. 2.1**. Broj precizno označenih piksela (y-os) po klasi i njima pridružene kategorije (x-os) [14]*



***Sl. 2.2**. Primjer iz Cityscapes skupa podatka. S lijeva na desno: originalna slika, segmentacijska mapa, preklopljena segmentacijska mapa preko originalne slike [14]*

ADE20K [15] skup podataka sadrži više od 20 tisuća slika scena iz baza podataka SUN [16] i Places [17] koje su potpuno anotirane s objektima, obuhvaćajući preko 3 tisuće klasa. Mnoge slike uz objekte sadrže i dijelove objekata. Također, pruža originalne anotirane poligone i instance objekata za amodalnu segmentaciju. Sve slike su anonimizirane zamagljujući lica i registarske tablice kako bi osigurale privatnost prikazanih osoba. Glavne prednosti ADE20K skupa podataka su raznolik i detaljan skup anotacija za prepoznavanje i segmentaciju objekata u slici, velika količina klasa, objekata i dijelova objekata, kao i njihove definicije i hijerarhije iz WordNeta [18] te informacije o vremenu anotiranja, atributima, dubinskom poretku i poligonima za svaki objekt. S druge strane, nedostaci su zahtijevana registracija i obavezna prijava za pristup datotekama, snimljene slike samo u pogodnim vremenskim uvjetima i u visokoj rezoluciji, što može otežati obradu i pohranu podataka te smanjiti primjenjivost algoritama koji koriste ADE20K u drugačijim vrstama okruženja. Slika 2.3 daje primjer segmentacije s pripadnim histogramima, a slika 2.4 daje primjer originalne slike s odgovarajućom slikom temeljne istine (engl. *ground truth*).



***Sl. 2.3.** a) Segmentacija 'nebo', 'zid', 'zgradu' i 'pod'. b) Histogram broja segmentiranih instanci objekata i klasa po slici. c) Histogram broja instanci i klasa segmentiranih dijelova po objektu. d) Broj klase kao funkcija segmentiranih instanci [15]*

***Sl. 2.4**. Primjer iz ADE20K skupa podatka. S lijeva na desno: originalna slika, segmentacijska mapa, preklopljena segmentacijska mapa preko originalne slike [15]*

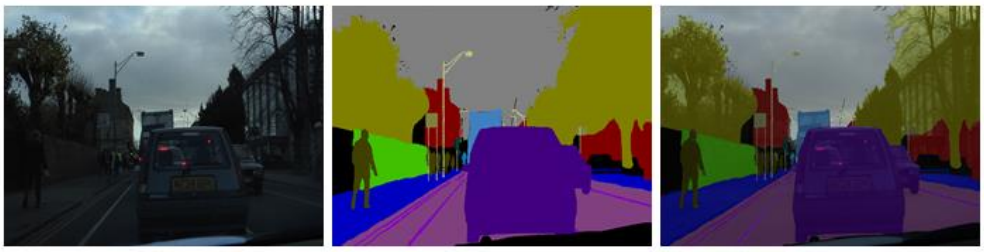
Skup podataka COCO-Stuff [19] koristi se za razumijevanje scena, poput zadataka semantičke segmentacije, detekcije objekata i generiranja opisa slika. Predstavlja proširenje originalnog COCO [20] skupa podataka koji je anotirao samo objekte koji se mogu pomicati i manipulirati, a zanemario pozadinske elemente koji nemaju jasne granice. COCO-Stuff sadrži 163 tisuće slika iz COCO skupa podataka; 118 tisuća za obuku, 5 tisuća za provjeru ispravnosti i po 20 tisuća za test-dev i 20 tisuća za *test-challenge* koje su anotirane na razini piksela sa 172 klase. COCO-Stuff pruža bogate i raznolike informacije o kontekstu scene koje pomažu boljem razumijevanju i opisivanju slika. Omogućuje istraživanje složenih odnosa između različitih objekata, što može dovesti do novih uvida i boljih primjena. Sadrži 5 opisa po slici iz COCO skupa podataka, što omogućuje evaluaciju kvalitete generiranih opisa slika na temelju ljudskih ocjena. Neki od nedostataka su zahtijevana velika količina memorije i računalne snage za obradu i analizu slika visoke rezolucije s gustim anotacijama te potencijalno teški pronalazak odgovarajućeg modela i metode za rješavanje različitih zadataka na istom skupu podataka. Različiti zadaci mogu se fokusirati na različite aspekte scene. Poprilično je otežano usklađivanje i uspoređivanje rezultata različitih istraživanja na COCO-Stuff skupu podataka zbog mogućnosti korištenja različitih verzija, podskupova i metrika za evaluaciju. Slika 2.5 daje primjer originalne slike s odgovarajućom slikom temeljne istine; segmentacijska mapa ima drugačije dimenzije u odnosu na originalnu i preklopljenu sliku zbog načina na koji su slike generirane ili kombinirane.

***Sl. 2.5**. Primjer iz COCO-Stuff skupa podatka. S lijeva na desno: originalna slika, segmentacijska mapa, preklopljena segmentacijska mapa preko originalne slike [19]*

Kako bi se postigao *overlay* prikaz, segmentacijska mapa je prvotno skalirana na jednake dimenzije kao i originalna slika. Nakon skaliranja, segmentacijska mapa je preklopljena preko originalne slike radi vizualnog prikaza rezultata segmentacije na originalnoj slici. Navedeni postupak omogućava jasan prikaz slaganja predviđanja modela s originalnim objektima slike, usprkos početnim različitim dimenzijama slika.

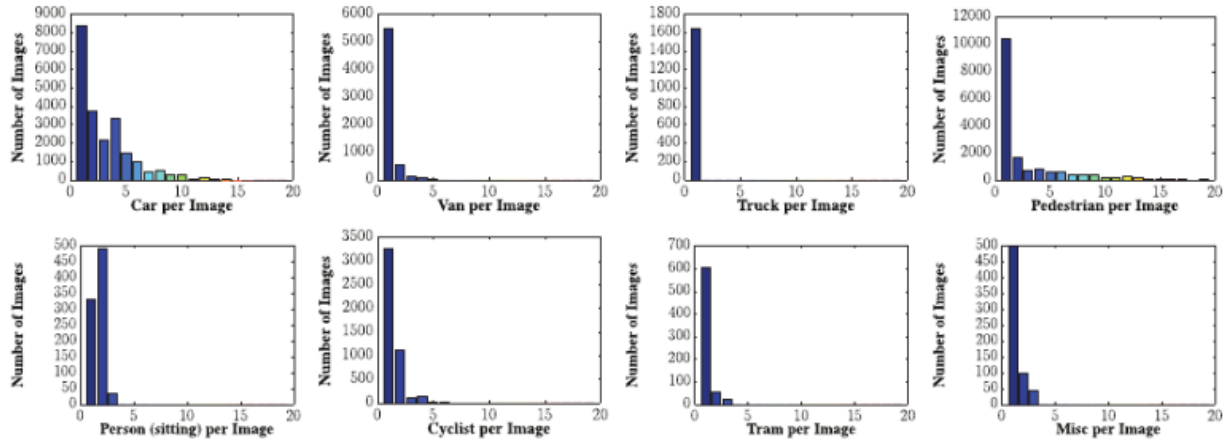
CamVid [21] je skup podataka koji sadrži videozapise semantičkih oznaka objekata tijekom scena vožnje automobila. Sadrži oznake koje povezuju svaki piksel s jednom od 32 semantičke klase. Koristi se kod evaluacije algoritama za segmentaciju i prepoznavanje objekata u videu. Visoke je kvalitete i sadrži slike velike rezolucije (960×720). Uključuje ručno specificiranu semantičku segmentaciju od 701 slike pregledanu i potvrđenu od strane druge osobe. Snimljene su kalibracijske sekvence za boju kamere i intrinzične parametre te je izračunata 3D pozicija kamere za svaki kadar u sekvencama. Također, pruža prilagođeni softver za označavanje koji može pomoći korisnicima crtati precizne oznake klasa za druge slike i videozapise. CamVid sadrži mali broj videozapisa (samo 5) i slika (701) koji možda nisu dovoljni za obučavanje dubokih neuronskih mreža. Nudi ograničen broj semantičkih klasa (samo 32) koji u nekim slučajevima ne pokriva sve moguće objekte u scenama vožnje. Neujednačena raspodjela klasa u skupu podataka može dovesti do pristranosti modela češćim klasama (slika 2.6). Na slici 2.7 vidljiv je primjer originalne slike s pripadajućom slikom temeljne istine.

Sl. 2.6. *Prikaz 32 klase CamVid skupa podataka [21]*



Sl. 2.7*. Primjer iz CamVid skupa podatka. S lijeva na desno: originalna slika, segmentacijska mapa, preklopljena segmentacijska mapa preko originalne slike [21]*

Skup podataka KITTI [22] sadrži videozapise prikupljene sa senzora različitih modaliteta, uključujući RGB, stereo kamere i 3D laserski skener, koji prikazuju scenarije vožnje automobila. Koristan je za evaluaciju algoritama za stereo, optički tok, vizualnu odometriju, 3D detekciju te praćenje objekata. Visoke je kvalitete i sadrži veliku rezoluciju video slika (1242×375). Ima preciznu referentnu vrijednost koja je dobivena pomoću Velodyne laserskog skenera i GPS sustava lokalizacije. Sadrži raznolike prikaze scena i objekata, uključujući gradska, ruralna i autocestovna okruženja s najviše do 15 automobila i 30 pješaka po slici. Koristi različite metrike za evaluaciju performansi algoritama na skupu podataka, kao i web stranice za rangiranje rezultata (slika 2.8). Nedostatak je mali broj videozapisa (samo 22) i slika (15 000) koji mogu biti nedovoljni za obučavanje dubokih neuronskih mreža. KITTI sadrži ograničen broj klasa objekata (8) te ima neujednačenu raspodjelu klasa u skupu podataka. Nedostatak mu je i manjak semantičke segmentacije za većinu slika u skupu podataka, što otežava zadatke semantičkog razumijevanja scene. Na slici 2.9 moguće je vidjeti primjer originalne slike KITTI skupa podataka s pripadajućom segmentacijskom maskom. Nakon toga, u tablici 2.1 sažeto su uspoređeni svi spomenuti skupovi podataka s obzirom na broj slika, vrstu, rezoluciju, izvor i sadržaj slika pri nepovoljnim uvjetima te broj klasa i svrhu primjene.



Sl. 2.8. *Broj oznaka objekata po klasi i slici. Pokazatelj učestalosti objekta na slici kod KITTI skupa podataka [22]*

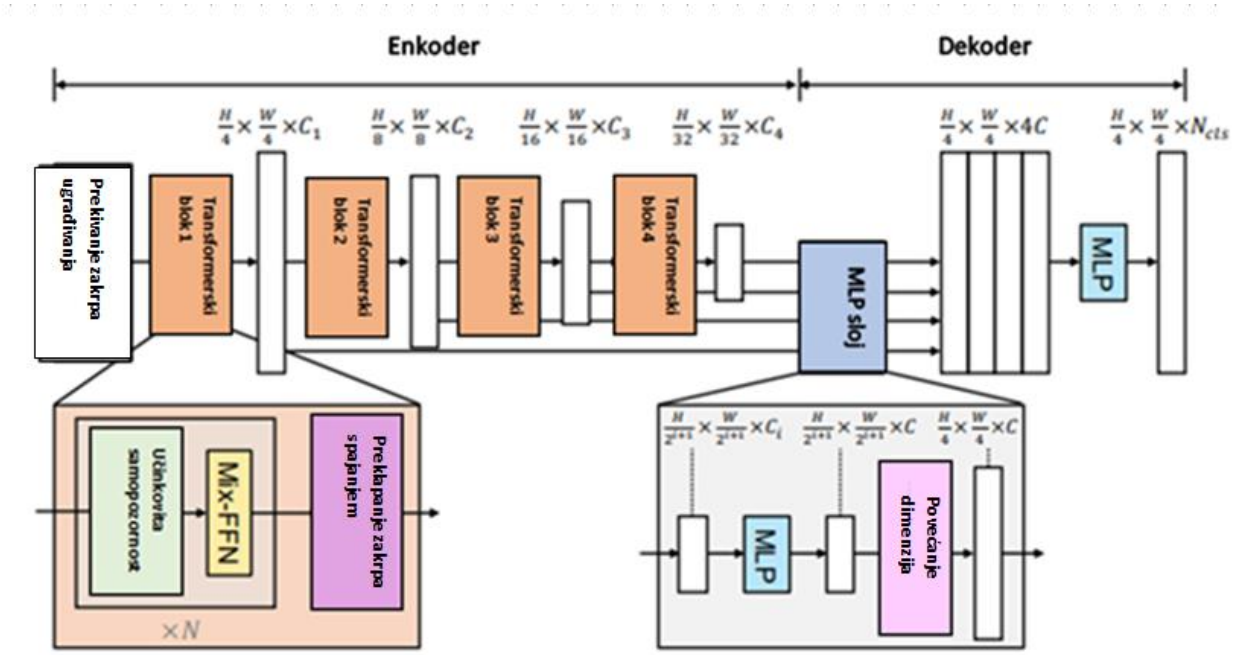
Sl. 2.9*. Primjer iz KITTI skupa podatka. S lijeva na desno: originalna slika, segmentacijska mapa, preklopljena segmentacijska mapa preko originalne slike [22]*

Tab. 2.1. *Usporedba različitih skupova podataka analiziranih u ovom radu [1*

Skup podataka	Broj slika za trening	Broj slika za validaciju	Broj slika za test	Vrsta slika	Rezolucija slika u pikselima	Sadrži slike pri nepovoljnim vremenskim uvjetima
Cityscapes [14]	2975	500	1525	Scene voznog područja urbanih mjesta	2048x1024	NE
ADE20K [15]	25574	2000	3000	Razne scene: urbano, suburbano, unutarnji prostori, prirodne scene	Različita; većina 512x512	NE
COCO-Stuff [19]	118000	5000	20000	Razne scene: slike iz različitih okoliša (urbano, suburbano, prirodno, otvoreno) i različitih vremenskih uvjeta, snimljene u različitim dijelovima dana (dan/noć)	640x480	DA
CamVid [21]	701	-	-	slike iz urbanih i suburbanih područja, snimljene iz vozila. Snimke uključuju različite vremenske uvjete (dnevno svjetlo, sumrak)	960x720	DA
KITTI [22]	7481	423	711	slike snimljene iz vozila u urbanim i otvorenim područjima. Snimke uključuju različite vremenske uvjete (dnevno svjetlo, oblačno) i različite dijelove dana (dan).	1242x375	DA

2.3. Pregled postojećih algoritama za segmentaciju voznog područja iz slike dobivene kamerom montiranom na prednjoj strani vozila

Jedan od novijih algoritama za segmentaciju voznog područja razvijen od strane NVIDIAe [23] je SegFormer [24] koji uključuje hijerarhijski strukturirani *transformer* enkoder čija je uloga generiranje višeklasnih značajki, pri čemu ne zahtjeva kodiranje položaja te jednostavni dekoder (slika 2.10).



Sl. 2.10. *Prikaz glavnih dijelova SegFormer okvira [24]*

Predloženi enkoder izbjegava interpolaciju položajnih kodova prilikom izvođenja inferencije na slikama s razlučivošću različitom od one korištene tijekom obuke, što rezultira prilagodbi enkodera proizvoljnim rezolucijama testiranja bez utjecaja na performanse. Hijerarhijski dio omogućuje enkoderu generiranje karakteristika visoke rezolucije i grubih karakteristika niske rezolucije, što je suprotno od ViT-a [25] koji može proizvesti samo pojedinačne karakteristike niske rezolucije s fiksnim rezolucijama. Dizajniran je niz *Mix Transformer* enkodera (MiT), MiT-B0 do MiT-B5, s istom arhitekturom, ali različitih veličina. MiT-B0 je lagani model za brzo zaključivanje, dok je MiT-B5 najveći model s najboljim performansama. Cilj ovog modela je, s obzirom na ulaznu sliku, generirati značajke na više razina poput CNN-a. Glavni dio enkodera je sloj samopažnje. U izvornom procesu samopažnje s više glava, svaka od glava Q, K, V ima iste dimenzije N × C, gdje je N jednak umnošku visine i širine slike te predstavlja duljina niza. Prema formuli (2-1) samopažnja se procjenjuje kao:

$$Samopažnja(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_{glava}}}\right)V, \quad (2\text{-}1)$$

s razinom kompleksnosti O(N^2). Radi smanjenja kompleksnost koristi se proces redukcije niza koji rezultira smanjenom razinom kompleksnosti O ($\frac{N^2}{R}$).

Prikupljanjem informacija iz različitih slojeva, MLP dekoder kombinira lokalnu i globalnu pažnju što rezultira jednostavnim i jasnim dekoderom koji generira snažne reprezentacije. Hijerarhijski transformer koder ima veće efektivno receptivno polje (ERF) od tradicionalnih CNN kodera te omogućuje relativno jednostavni dekoder. Predloženi All-MLP dekoder sastoji se od četiriju glavnih koraka. Prema formulama od (2-2) do (2-5) vidljivo je kako u prvom koraku višerazinske značajke F_i iz MiT kodera prolaze kroz MLP sloj radi unifikacije dimenzija kanala. U drugom koraku, značajke se uzorkuju do četvrtine svoje izvorne veličine te se spajaju. Treći korak podrazumijeva usvajanje MLP sloja za spajanje značajki *F*. Posljednji korak odnosi se na drugi MLP sloj koji uzima spojene značajke za predviđanje maska segmentacije *M* sa *H/4*W/4*Ncls* rezolucijom, gdje *Ncls* predstavlja broj klasa.

$$\hat{F}_i = Linear(C_i, C)(F_i), \forall i, \quad (2\text{-}2)$$

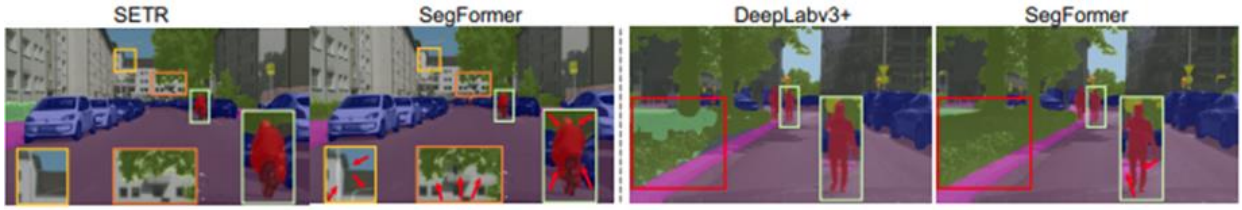
$$\hat{F}_i = Upsample\left(\frac{W}{4}\right) * \frac{W}{4}(F_i), \forall i, \quad (2\text{-}3)$$

$$F = Linear(4C, C)(Concat(F_i)), \forall i, \quad (2\text{-}4)$$

$$M = Linear(C_i, N_{cls})(F). \quad (2\text{-}5)$$

U radu su demonstrirane prednosti SegFormera po pitanju veličine modela, vremena izvođenja i točnosti na trima javno dostupnima skupovima podataka: ADE20K [15], Cityscapes [14] i COCO-Stuff [19]. Korištena je *mmsegmentation1* kodna baza koja je trenirana na serveru s 8 Tesla V100. Enkoder je prethodno treniran na Imagenet-1K [26] skupu podataka, a dekoder je nasumično inicijaliziran. Tijekom procesa treniranja primijenjena je augmentacija podataka pomoću nasumične promjenjive veličine s omjerom 0.5-2.0, nasumičnog horizontalnog preklapanja i nasumičnog obrezivanje na 512 × 512, 1024 × 1024, 512 × 512 za ADE20K, Cityscapes i COCO-Stuff, redom. Modeli su trenirani koristeći AdamW optimizator tijekom 160 tisuća iteracija na ADE20K, Cityscapes i 80 tisuća iteracija na COCO-Stuff skupu podataka. Korištena je veličina grupe (engl. *batch size*) 16 za ADE20K i COCO-Stuff, te 8 za Cityscapes. Stopa učenja postavljena je na početnu vrijednost od 0.00006, a zatim je korišten "*poly*" raspored

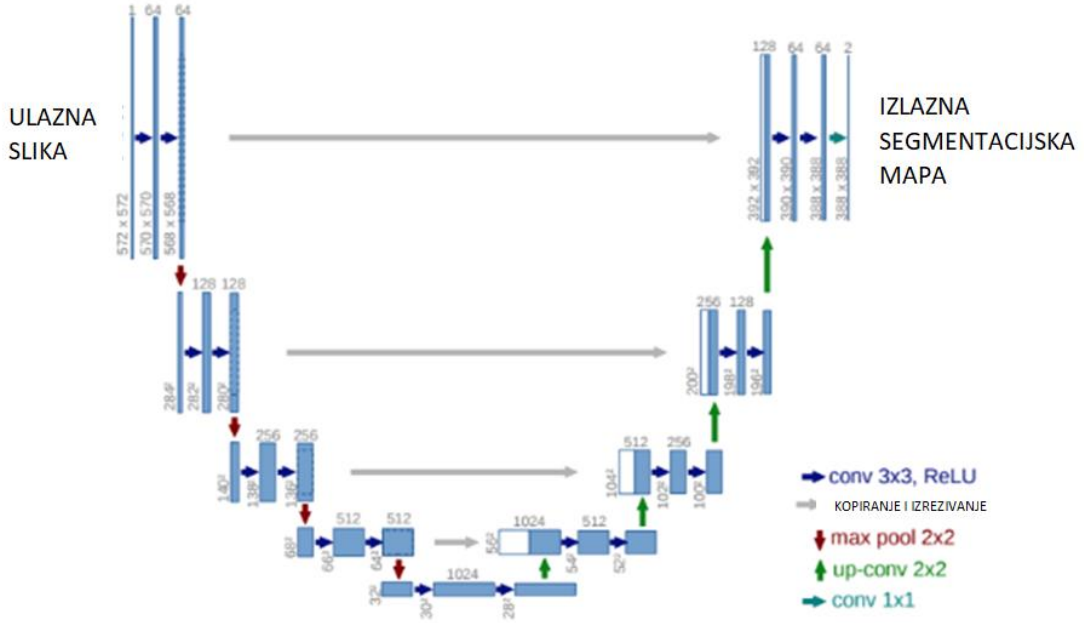
stope učenja s faktorom 1.0 prema zadanim postavkama. Na Cityscapesu SegFormer-B0, bez ubrzanja poput TensorRT-a, ostvaruje 71.9% mIoU (engl. *Mean Intersection over Union*) pri 48 FPS (engl. *Frames Per Second*), što predstavlja relativno poboljšanje od 60% u latenciji (brzini) i 4.2% u preciznosti predikcija u usporedbi s ICNet [27]. SegFormer-B5, ostvaruje 84.0% mIoU, što predstavlja relativno poboljšanje od 1.8% mIoU dok je 5× brži od SETR [28]. Na ADE20K, ovaj model ostvaruje 51.8% mIoU dok je 4× manji od SETR-a. Opisani pristup znatno je otporniji na proboje i potencijalne napade, čime je prikladan za aplikacije kod kojih je ključna sigurnost. Slika 2.11 prikazuje rezultate treniranja na Cityscapes skupu podataka.



Sl. 2.11. *Rezultati treniranja na Cityscape skupu podatak [28]*

Autori rada [29] koriste besplatan i otvoreni skup podataka od 5000 snimljenih slika iz CARLA simulatora s pripadajućim slikama temeljnih istina za semantičku segmentaciju. Segmentirane slike podijeljene su u 13 klasa, gdje svaka klasa predstavlja različite objekte. Veličina skupa podataka je oko 5 GB. Predstavljeni model (slika 2.12) zasniva se na U-Net strukturi. Lijeva strana modela predstavlja nedefinirani model za izdvajanje značajki koji je treniran za klasifikaciju slika. Nakon izdvajanja željenih značajki, slijedi daljnja obrada na različitim skalama kako bi se omogućio zadovoljavajući rad modela nad objektima različitih veličina. Također, potrebna je obrada visoko-razinskih značajki iz kasnijih dijelova mreže koje sadrže korisne semantičke informacije. Nakon ulaznog sloja s više konvolucijskih grupa, slojeva normalizacije po grupama i slojeva aktivacije tijekom cijele strukture, model slijedi određeni uzorak. Nakon prvog sloja konvolucije (*Conv2D*), struktura se dijeli na dva odvojena toka kako bi se usredotočilo i na glavne značajke i na rezidualne dijelove objekta radi zadovoljavajuće lokalizacije. Nakon svakog takvog sloja dodaju se slojevi obrade. Nakon drugog sloja zbrajanja (*add_2)* veličina izlazne slike je (16, 16, 256). Stoga, nakon izdvajanja značajki iz malih dijelova slike, jedan tok prolazi kroz isto izdvajanje značajki dok se drugi *up-sampla* i koristi za rezidualno izdvajanje. Nakon šestog sloja zbrajanja (*add_6*) slijedi sloj konvolucije s 13 različitih izlaza te *Softmax* aktivacijom za segmentaciju slika u više kanala. Njegov izlaz ima oblik (*x*, 256, 256, 13) gdje *x* označava broj ulaznih slika, dok su sljedeće dvije dimenzije,

dimenzije izlaza, a posljednja dimenzija označava broj kanala, pri čemu svaki kanal predstavlja različitu klasu (npr. automobili, cesta, vegetacija, zgrade itd.).



Sl. 2.12. *Izgled modela [29]*

Od hiperparametara modela potrebno je spomenuti kako je broj epoha 40, korišten je *rmsprop* optimizator, veličina grupe jednaka je 32, što znači da se 32 primjera koriste u svakoj iteraciji treniranja modela. Iz tablice 2.2 moguće je iščitati performanse modela na validacijskom i testnom skupu podataka.

Tab. 2.2. *Prikaz rezultata modela [29].*

Točnost na validacijskom skupu podataka	Točnost na testnom skupu podataka	Gubitak na skupovima podataka za validaciju i testiranje
91.92%	93.80%	< 0.25

Rad [30] predstavlja implementaciju arhitekture U-Net za semantičku segmentaciju slika korištenjem *Kerasa* i *TensorFlowa*. U prvom dijelu učitavaju se potrebne biblioteke poput *NumPy*, *scikit-image* i *Matplotlib* za obradu slika i vizualizaciju. Nakon toga slijedi faza pripreme podataka. Podaci (slike s odgovarajućim maskama) se učitavaju iz odgovarajućih datoteka i obrađuju na željenu veličinu od 128x128 piksela kako bi odgovarali formatu potrebnom za treniranje modela. Također, vrši se vizualizacija nekoliko trening slika zajedno s pripadajućim maskama kako bi se prikazali podaci koje koristi model.

U središnjem dijelu koda definira se arhitektura U-Net modela. U-Net se sastoji od kontrakcijskog i ekspanzijskog dijela, koji omogućavaju učenje i usklađivanje nivoa detalja slika. Kontrakcijski dio obuhvaća nekoliko slojeva konvolucije, aktivacije ReLU i *dropout* slojeva koji služe za izbjegavanje prevelike naučenosti modela, tj. prevelike prilagodbe modela trening podacima. Ekspanzijski dio koristi slojeve konvolucije i transponirane konvolucije kako bi povećao prostornu rezoluciju izlaza. Izlazni sloj koristi aktivaciju sigmoida kako bi generirao binarne maske. Nakon definiranja arhitekture, model se trenira na trening podacima koristeći optimizator Adam i funkciju gubitka binarne unakrsne entropije (engl. *binary crossentropy loss*). Treniranje se odvija tijekom 25 epoha s veličinom grupe 16. Tijekom treniranja koriste se pozivne funkcije (engl. *callbacks*) kao što su *ModelCheckpoint* za spremanje najboljeg modela, *EarlyStopping* za sprječavanje prenaučenosti te *TensorBoard* za praćenje napretka učenja. Nakon završetka treniranja, model se koristi za predikciju na skupovima za treniranje, validaciju i test. Dobivene predikcije binariziraju se s pragom od 0.5 radi dobivanja binarnih maski. Konačno, radi provjere, nekoliko nasumično odabranih slika iz trening skupa podataka prikazuju se zajedno s pravim maskama i predikcijama kako bi se ocijenila kvaliteta segmentacije.

U radu [31] predstavljane su mreže dvostruke rezolucije s dubokim reprezentacijama visoke rezolucije za zadatak semantičke segmentacije visoko-rezolucijskih slika, posebno slika ceste u realnom vremenu. DDRNets započinju s jednim stablom, zatim se dijele na dva usporedno duboka ogranka različitih rezolucija. Jedan od ogranaka generira relativno visoko-rezolucijske mape značajki, dok drugi ekstrahira bogate semantičke informacije kroz višestruke operacije uzorkovanja prema dolje. Višestruke bilateralne veze povezuju ogranke kako bi se postiglo što učinkovitije spajanje dobivenih informacija. Rad predlaže i novi modul DAPPM koji prima značajne mape niske rezolucije, izvlači kontekstualne informacije višestrukih razmjera i spaja ih na kaskadni način.

Svi modeli trenirani su s ulaznom rezolucijom 224×224 piksela, veličinom grupe 256 sa 100 epoha na četiri NVIDIA RTX 2080 Ti grafičkom procesoru (engl. *Graphics processing unit*) GPU. Početna stopa učenja postavljena je na 0,1 te se smanjuje s faktorom 10 u epohama 30, 60 i 90. Sve mreže obučene su korištenjem SGD-a (engl. *Stochastic Gradient Descent*) sa smanjenjem težine od 0.0001 i Nesterova momenta od 0.9. DDRNet testiran je na tri skupa podataka: Cityscapes, CamVid i COCOStuff gdje postiže kompromis između preciznosti i brzine. 77.4% mIoU pri 102 FPS na Cityscapes testnom skupu, 80.4% mIoU pri 22 FPS i 74.7% mIoU pri 230 FPS na CamVid testnom skupu.

Radi postizanja ravnoteže između razlučivosti i brzine zaključivanja, visoko-rezolucijska grana stvara značajke čija je razlučivost jednaka 1/8 razlučivosti ulazne slike, pri tome ne sadržavajući nikakve operacije smanjenja razlučivosti. Radi formiranja duboke reprezentacije visoke razlučivosti, odnos s granom niske rezolucije joj je jedan naprema jedan. Slijedi višestruko bilateralno spajanje značajki na različitim fazama kako bi se prostorne informacije spojile sa semantičkim informacijama. Kod ulaznog dijela ResNeta [32] jedan 7×7 konvolucijski sloj zamijenjen je s dvama uzastopnim 3×3 konvolucijska sloja. Osnovni rezidualni blokovi koriste se za izgradnju trupa i narednih dviju grana. Za proširenje dimenzija izlaza, na kraj svake grane dodan je blok s uskim grlom. Bilateralno spajanje uključuje spajanje visoko-rezolucijske grane s nisko-rezolucijskom granom (spajanje od visokog do niskog) i obrnuto (spajanje od niskog do visokog). Za spajanje od visokog do niskog, značajke visoke razlučivosti smanjuju se sekvencom 3×3 konvolucijskih slojeva. Za spajanje od niskog do visokog razlučivanja, značajke niske razlučivosti prvo se komprimiraju pomoću 1×1 konvolucijskog sloja, a zatim se povećavaju uzorkovanjem bilinearnom interpolacijom. Formule (2-6) i (2-7) prikazuje implementaciju bilateralnog spajanja gdje su i-te značajke visoke razlučivosti X_{Hi} i niske razlučivosti X_{Li} predstavljene kao:

$$X_{Hi} = R\left(F_H\left(X_{H(i-1)}\right) + T_{L-H}\left(F_L\left(X_{L(i-1)}\right)\right)\right), \tag{2-6}$$

$$X_{Li} = R\left(F_L\left(X_{L(i-1)}\right) + T_{H-L}\left(F_H\left(X_{H(i-1)}\right)\right)\right), \tag{2-7}$$

gdje F_H i F_L odgovaraju sekvenci rezidualnih osnovnih blokova s visokom razlučivošću i niskom razlučivošću, $T_{(L-H)}$ i $T_{(H-L)}$ odnose se na „niski-do-visoki“ i „visoki-do-niski“ transformator, a R označava ReLU aktivacijsku funkciju. Konstruirane su četiri mreže s dvostrukom razlučivošću različitih dubina i širina. DDRNet-23 je dvostruko širi od DDRNet-23-slim, a DDRNet-39 1,5× je također šira verzija DDRNet-39 (tablica 2.3).

Po uzoru na Res2Net [33] radi se povećanje značajki, nakon čega se koristi više 3×3 konvolucija radi integriranja kontekstualnih informacija različitih razlučivanja na hijerarhijski-rezidualan način. Prema formuli (2-8) razmatrajući ulaz x, svaki y_i se može zapisati kao:

$$y = \begin{cases} C_{1*1}(x), \ i = 1; \\ C_{3*3}\left(U\left(C_{1*1}\left(P_{2^i+1,2^{i-1}}(x)\right)\right) + y_{i-1}\right), \ 1 < i1n; \\ C_{3*3}\left(U\left(C_{1*1}\left(P_{globalno}(x)\right)\right) + y_{i-1}\right), \ i = n; \end{cases} \tag{2-8}$$

gdje je C_{1*1} konvolucija veličine 1×1, C_{3*3} je konvolucija veličine 3×3, $P_{j,k}$ označava sloj bazena čija je veličina jezgre j, a korak je k, P_{global} označava globalno prosječno grupiranje.

***Tab. 2.3**. Arhitekture DDRNet-23-slim i DDRNet-39 za ImageNet; „conv4*r" označava da se conv4 ponavlja r puta (za DDRNet-23-slim r = 1 i za DDRNet-39 r = 2).[31]*

sloj	izlaz	DDRNet-23-*slim*	DDRNet-39
conv1	112x112	3x3, 32, *stride* 2	3x3, 64, *stride* 2
conv2	56x56	3x3, 32, *stride* 2	3x3, 64, *stride* 2
		$\begin{bmatrix} 3*3 & 32 \\ 3*3 & 32 \end{bmatrix} * 2$	$\begin{bmatrix} 3*3 & 64 \\ 3*3 & 64 \end{bmatrix} * 3$
conv3	28x28	$\begin{bmatrix} 3*3 & 64 \\ 3*3 & 64 \end{bmatrix} * 2$	$\begin{bmatrix} 3*3 & 128 \\ 3*3 & 128 \end{bmatrix} * 4$
conv4*r	14x14,28x28	$\begin{bmatrix} 3*3 & 128 \\ 3*3 & 128 \end{bmatrix} * 2$ $\quad$ $\begin{bmatrix} 3*3 & 64 \\ 3*3 & 64 \end{bmatrix} * 2$	$\begin{bmatrix} 3*3 & 256 \\ 3*3 & 256 \end{bmatrix} * 3$ $\quad$ $\begin{bmatrix} 3*3 & 128 \\ 3*3 & 128 \end{bmatrix} * 3$
		Bilateralna fuzija	Bilateralna fuzija
conv5_1	7x7, 28x28	$\begin{bmatrix} 3*3 & 256 \\ 3*3 & 256 \end{bmatrix} * 2$ $\quad$ $\begin{bmatrix} 3*3 & 64 \\ 3*3 & 64 \end{bmatrix} * 2$	$\begin{bmatrix} 3*3 & 512 \\ 3*3 & 512 \end{bmatrix} * 3$ $\quad$ $\begin{bmatrix} 3*3 & 128 \\ 3*3 & 128 \end{bmatrix} * 3$
		Bilateralna fuzija	Bilateralna fuzija
		$\begin{bmatrix} 1*1 & 256 \\ 3*3 & 256 \\ 1*1 & 512 \end{bmatrix} * 1$ $\quad$ $\begin{bmatrix} 1*1 & 64 \\ 3*3 & 64 \\ 1*1 & 128 \end{bmatrix} * 1$	$\begin{bmatrix} 1*1 & 512 \\ 3*3 & 512 \\ 1*1 & 1024 \end{bmatrix} * 1$ $\quad$ $\begin{bmatrix} 1*1 & 128 \\ 3*3 & 128 \\ 1*1 & 256 \end{bmatrix} * 1$
conv5_2	7x7	Visoka-do-niska fuzija	Visoka-do-niska fuzija
		1x1, 1024	1x1, 2048
	1x1	7x7 združivanje po srednjoj vrijednosti	7x7 združivanje po srednjoj vrijednosti
		1000-d fc, softmax	1000-d fc, softmax

Zatim se sve mape značajki spajaju i komprimiraju pomoću konvolucije veličine 1×1. Osim toga, dodaje se 1×1 projekcijski prečac za jednostavnu optimizaciju. DAPPM je implementiran sa sekvencom BN-ReLU-Conv. Unutar DAPPM-a, konteksti izdvojeni većim jezgrama bazena integriraju se s dubljim tokom informacija, a više-skalna priroda oblikuje se integriranjem različitih dubina s različitim veličinama jezgara bazena. Iako DAPPM sadrži više konvolucijskih slojeva i složeniju strategiju fuzije, rijetko utječe na brzinu zaključivanja jer je razlučivost ulaza samo 1/64 razlučivosti slike. Primjerice, uz sliku od 1024×1024, maksimalna razlučivost značajki je 16×16.

Neki od mogućih nedostataka predloženog rješenja su što mreže dvostruke rezolucije zahtijevaju više memorije i računalnih resursa od mreža jednostruke rezolucije. Također, bilateralne fuzije između dviju grana mogu uzrokovati gubitak informacija ili neusklađenost značajki ako se ne usklade pažljivo. Modul DAPPM može biti osjetljiv na promjene u raspodjeli

podataka ili različitim razinama šuma, jer se oslanja na globalno prosječno grupiranje i adaptivno grupiranje. Slika 2.13. daje vizualni prikaz rezultata segmentacije na Cityscapes skupu podataka.

Sl. 2.13. *Vizualizacija rezultata segmentacije na Cityscapes skupu podatka. S lijeva na desno; ulazna slika, temeljna istina, izlaz DDRNet-23-slim, izlaz DDRNet-23 [31].*

Rad [34] predstavlja implementaciju UNET arhitekture za semantičku segmentaciju slika unutar *Pythona* koristeći biblioteke poput *PyTorch*, *Albumentations* i *PIL*. Radi se o sustavu za učenje koji koristi Carvana [35] skup podataka. Implementacija rješenja organizirana je unutar četiriju odvojenih datoteka čije su funkcionalnosti međusobno povezane.

Unutar *dataset.py* datoteke definirana je *CarvanaDataset* klasa koja predstavlja prilagođeni skup podataka za obradu slika i maski. Koristi se za učitavanje slika i maski iz odgovarajućih datoteka te primjenu transformacija. Carvana skup podataka sadrži slike vozila snimljenih iz različitih kutova snimanja i pod različitim uvjetima osvjetljenja te s različitim pozadinskim okolinama. Svaka ulazna slika ima pripadajuću masku koja označava vozilo na slici. Ulazne slike su u RGB formatu, dok su maske binarne slike kod kojih su područja na kojima se nalaze vozila označena bijelom bojom (binarno 1), a ostatak slike je crne boje (binarno 0).

Datoteka *utils.py* sadrži pomoćne funkcije koje se koriste tijekom treninga i evaluacije modela. To uključuje funkcije za spremanje i učitavanje stanja modela (engl. *checkpoints*), generiranje *DataLoader* objekata za učitavanje podataka, provjeru točnosti modela te spremanje predikcija u obliku slika radi vizualizacije performansi. Funkcija *'save_checkpoint'* omogućava spremanje trenutnog stanja modela i pripadajućeg optimizatora u datoteku kako bi se mogli koristiti prema potrebi. Druga funkcija, *'load_checkpoint'*, služi za učitavanje prethodno spremljenog stanja modela iz datoteke, što je korisno za daljnje treniranje modela od zadnjeg definiranog trenutka. Zatim, funkcija *'get_loaders'* generira *DataLoader* objekte za trening i validaciju modela koristeći prilagođeni skup podatka definiran u *dataset.py* datoteci. Ova funkcija omogućava efikasno učitavanje podataka za treniranje i evaluaciju. Funkcija *'check_accuracy'* koristi se za provjeru točnosti modela na zadanom DataLoaderu, što je potrebno radi praćenja performansi modela tijekom treniranja. Posljednja funkcija, *'save_predictions_as_imgs'* sprema predikcije modela kao slike radi vizualne provjere

performansi. Postupak omogućava usporedbu predikcija modela s originalnim maskama i olakšava analizu performansi modela.

Model.py pruža informacije o arhitekturi modela koja se sastoji od enkodera i dekodera. Enkoderi su niz blokova nazvanih *'DoubleConv'*, koji se smanjuju koristeći *max pooling* operaciju. Dekoderi se, s druge strane, sastoje od konvolucijskih slojeva i *upscaling* operacija. Metoda *'forward'* definira propagaciju kroz mrežu. Ulazni podaci prvo prolaze kroz enkodere, a zatim i kroz dekodere. Dobiveni izlaz prolazi kroz posljednji konvolucijski sloj kako bi se dobio konačni rezultat. Također, u datoteci se nalazi i klasa *'DoubleConv'* koja definira blok dvaju konvolucijskih slojeva s ReLU aktivacijom i *batch* normalizacijom između njih. Ovaj se blok koristi unutar UNet arhitekture za obavljanje operacija konvolucije. Dodatak datoteci je testna funkcija pod nazivom *'test'* koja provjerava ispravnost implementacije modela. Funkcija *'test'* generira slučajne ulazne podatke i provodi propagaciju provjeravajući pri tome je li izlazni oblik ispravan.

Posljednja datoteka *train.py* predstavlja skriptu za treniranje UNet modela za semantičku segmentaciju slika. Uključuje logiku za pripremu podataka, definiranje modela, postavljanje hiperparametara, izvođenje treniranja i evaluacije te spremanje rezultata i stanja modela. U početnom dijelu datoteke navode se potrebne biblioteke i klase, uključujući *PyTorch*, *Albumentations* za augmentaciju slika, *tqdm* za prikaz napretka u petljama te definicije modela i pomoćnih funkcija iz prethodno navedenih datoteka. Zatim se definiraju hiperparametri kao što su stopa učenja, uređaj na kojemu će se izvršiti kod (centralna procesorska jedinica (engl. *Central processing unit*, CPU) ili GPU), veličina grupe, broj epoha, veličina slika i putanje do direktorija s podacima. Nakon toga, slijedi funkcija za treniranje i evaluaciju modela. Funkcija *'train_fn'* izvodi jednu epohu treninga, dok funkcija *'check_accuracy'* provjerava točnost modela na validacijskom skupu. Nakon definiranja funkcija, slijedi priprema *DataLoader* objekata za trening i validaciju pomoću funkcije *'get_loaders'* iz *utils.py* te inicijalizacija UNet modela. Glavna petlja izvršava treniranje i evaluaciju modela kroz zadani broj epoha. Za svaku se epohu prolazi kroz trening podatke, izvršava se trening i evaluacija modela te se sprema stanje modela nakon završene epohe kako bi se omogućilo kasnije učitavanje i korištenje treniranog modela.

3. EVALUACIJA I POJEDNOSTAVLJENJE ALGORITMA ZA SEGMENTACIJU VOZNOG PODRUČJA IZ SLIKE DOBIVENE KAMEROM MONTIRANOM NA PREDNJOJ STRANI VOZILA

3.1. Programsko okruženje i biblioteke korištene za razvoj algoritma

3.1.1. Programsko okruženje korišteno za razvoj algoritma

Razvoj *Python* programa omogućen je korištenjem integriranog razvojnim okruženjem *Visual Studio Code* (*VS Code*). *VS Code* [36] je alat koji pruža napredne značajke za pisanje koda te omogućava integraciju raznih alata za olakšan razvoj softvera. Radno okruženje konfigurirano je na način da koristi *Python* verziju 3.11.1, što znači da su svi programi pisani u *Python*u izvršavani pomoću navedene verzije jezika. Osim toga, u okviru radnog okruženja integrirani su razni alati za razvoj i obrazovanje, kao i upravitelj paketa naziva *PIP* koji omogućava instaliranje dodatnih *Python* paketa. *Python* tumač (engl. *interpreter*) je nadograđen s nekoliko dodatnih paketa biblioteka kako bi se omogućila upotreba određenih funkcionalnosti. Na primjer, *OpenCV*, *NumPy*, *PIL* (engl. *Python Imaging Library*) i *tqdm* su neki od dodatnih paketa koji su instalirani putem *PIP*-a.

U svrhu prikupljanja trening i testnih skupova podataka koriste se simulatori autonomne vožnje koji omogućavaju promjenu vremenskih uvjeta i scenarija virtualne okoline vozila. *CARLA* (engl. *Car Learning to Act)* [37], besplatni simulator otvorenog koda, namijenjen je razvitku autonomne vožnje te omogućava razvoj, treniranje i testiranje algoritama za autonomnu vožnju. Sadrži veliki broj implementiranih senzora od kojih su neki: RGB i dubinske kamere, kamere za semantičku segmentaciju, senzori kolizije, lidar i radar. *CARLA* simulator koristi klijent-poslužitelj arhitekturu gdje se virtualna okolina simulatora pokreće zasebno pomoću izvršne datoteke koja je u ulozi poslužitelja, a klijentske skripte šalju zahtjeve za konfiguraciju okruženja, stvaranje vozila i ostalih objekata te upravljanje istim objektima.

Za implementaciju algoritma na *Android* mobilnom uređaju korišten je *Android Studio* [27]; integrirano razvojno okruženje (IDE) namijenjeno razvoju aplikacija za *Android* platformu. To je službeno *Google IDE* za razvoj *Android* aplikacija koje se temelji na *IntelliJ IDEA IDE*-u. *Android Studio* omogućuje razvoj različitih vrsta *Android* aplikacija, kao što su mobilne aplikacije, aplikacije za TV, nosive uređaje i još mnogo toga. Sadrži brojne integrirane alate i

funkcije čiji je zadatak olakšanje razvoja aplikacija, poput XML dizajner sučelja, izgradnja alata, dubinsko pretraživanje koda te analiza performansi. *Android Studio* koristi *Gradle* kao alat za izgradnju projekata i dolazi s ugrađenim *Android* emulatorima za testiranje aplikacija na različitim virtualnim uređajima. *Google* redovito ažurira *Android Studio* s novim značajkama, poboljšanjima performansi i ispravcima grešaka (engl. *bugs*). *IDE* se integrira s različitim *Google* uslugama kao što su *Firebase, Google Play Services, Google Maps* i druge, čime je olakšana integracija navedenih usluga unutar *Android* aplikacije.

3.1.2. Biblioteke korištene za razvoj algoritma

TensorFlow [38], razvijen od strane *Google*a, predstavlja biblioteku otvorenog pristupa (engl. *open-source)* koja omogućava izgradnju, treniranje i razvoj brojnih modela dubokog učenja. *Tensorflow* koristi grafički model izračuna s tenzorima kao glavnom jedinicom podataka, što omogućuje efikasno izvršavanje operacija na GPU i TPU (engl. *Tensor Processing Unit*). Pruža alate poput *TensorBoard*-a za vizualizaciju rezultata i *TensorFlow Extended* (TFX) za razvoj *end-to-end* rješenja u strojnom učenju. Prilikom treniranja modela korištena je verzija *TensorFlow* 2.15.0.

Keras [39] je aplikacijsko programsko sučelje (engl. *Application Programming Interface,* API) visoke razine koji služi za izgradnju neuronskih mreža te je uključen u *TensorFlow* od verzije 2.0. Namijenjen je olakšanju razvoja neuronskih mreža putem jednostavnog i intuitivnog sučelja. *Keras* omogućuje brzo definiranje, prevođenje (engl. *compilation*) i treniranje modela, pri tome podržavajući razne arhitekture modela, slojeva i funkcija gubitaka. Prilikom treniranja modela korištena je verzija *Keras* 2.15.0.

NumPy [40] je projekt otvorenog koda koji omogućuje numeričke izračune u *Pythonu*. Nastao je 2005. godine temeljeći se na dotadašnjim bibliotekama *Numeric* i *Numarray*.

Tqdm [41] je *Python* paket koji omogućuje prikazivanje napretka u petljama tijekom izvršavanja određenog procesa. Koristan je kod izvršavanja dugotrajnih procesa kao što su treniranje modela ili obrada velikih skupova podataka. Pruža korisnicima vizualni prikaz napretka izvršavanja pomoću jednostavne trake napretka s informacijama poput vremena izvršavanja, prosječne brzine i preostalog vremena treniranja.

OpenCV (engl. *Open Source Computer Vision Library*) [42] je *Python* biblioteka koja pruža širok spektar alata i funkcija za obradu slika i računalni vid. Omogućava učitavanje,

prikazivanje, spremanje i manipulaciju slikama različitih formata te izvođenje operacija poput promjene veličine, rotacije, izrezivanja i promjene boje. Nudi alate za detekciju i praćenje objekata u slikama i videozapisima, uključujući detekciju lica, rubova, linija i objekata. *OpenCV* uključuje implementacije različitih algoritama strojnog učenja i dubokog učenja za zadatke računalnog vida poput klasifikacije, segmentacije i prepoznavanja.

Python biblioteka matplotlib [43] omogućuje vizualizaciju podataka i crtanje grafova. Pruža alate za stvaranje raznih vrsta grafikona, uključujući linije, trake, histograme, raspršene grafikone te druge vrste grafova. *Matplotlib* omogućuje jednostavno crtanje grafova pomoću *Python* koda, pružajući kontrolu nad svakim aspektom grafa, uključujući osi, oznake, boje i stilove. Također, podržava interaktivne grafove, što omogućuje korisnicima interakciju s grafom putem miša ili tipkovnice (zumiranje, pomicanje, označavanje točaka). Široko se koristi u raznim područjima, uključujući znanstveno istraživanje, analizu podataka, financije, inženjerstvo, medicinu, geografiju i mnoge druge discipline. Na slici 3.1 prikazane su naredbe za instalaciju *Tensorflow* 2.15.0, *Keras* 2.15.0 te paketa *numpy*, *opencv-python*, *matplotlib* i *tqdm*.

```
------------------------------------------------
pip install tensorflow==2.15.0
pip install keras==2.15.0
pip install numpy opencv-python matplotlib tqdm
------------------------------------------------
```

Sl. 3.1. *Naredbe za instalaciju potrebnih biblioteka i paketa*

3.2.Pregled metrika za evaluaciju modela strojnog učenja

Evaluacijske metrike ključne su za ocjenjivanje performansi modela strojnog učenja. Osim što pružaju objektivnu procjenu ispravnosti rada modela, pružaju i uvid u dublje razumijevanje njihovih sposobnosti i nedostataka. Korištenje metrika omogućava praćenje ponašanja modela tijekom treninga, čime je moguće identificirati poboljšanja ili pogoršanja rada modela te jednostavnije identificirati uzroke smanjenja performansi. Evaluacijske metrike moguće je primijeniti na različite arhitekture modela te ih međusobno uspoređivati. Također, pomažu pri donošenju odluke koji model ili verzija modela je najprikladnija za konkretne primjene.

3.2.1. Matrica zabune

Matrica zabune koriste se za procjenu pogrešaka kod problema klasifikacije najčešće s nadziranim strojnim učenjem. Kao rezultat matrica zabune može dati jedno od četiriju stanja

(tablica 3.1): istinito pozitivni (engl. *true positive*), istinito negativni (engl. *true negative*), lažno pozitivni (engl. *false positive*) i lažno negativni rezultati (engl. *false negative*). Broj istinito pozitivnih rezultati označava točno klasificirane slučajeve, istinito negativni rezultati označavaju ispravno neklasificirane objekte, lažno pozitivni označavaju pogrešno klasificirane slučajeve, a lažno negativni označavaju propuštenu klasifikaciju [44]. Parametri matrice zabune mogu predstavljati brojeve točno/netočno klasificiranih slika ili piksela cijelog trening skupa podataka.

Tab.3.1. *Prikaz matrice zabune: TP – istinito pozitivno, FN – lažno negativni, FP – lažno pozitivni, TN – istinito pozitivni. [44]*

		Zadatak z	
		+	-
Oznaka l	+	TP	FN
	-	FP	TN

3.2.2. Preciznost

Preciznost (engl. *Precision*) ili povjerenje (engl. *Confidence*) prema formuli (3-1) definira se kao udio ispravno klasificiranih slučajeva u sveukupnom skupu pozitivno klasificiranih slučajeva [45]. Preciznost odražava sposobnost modela za identificiranjem relevantnih slučajeva. Visoka preciznost upućuje na to da model u rijetkim slučajevima daje lažno pozitivne rezultate, odnosno da rijetko označava negativne primjere kao pozitivne. Preciznost je preporučljivo koristiti u kombinaciji s drugim evaluacijskim metodama kako bi se dobila potpunija slika performansi evaluiranog modela, jer kod modela s mnogo negativnih primjera (neujednačeni, tj. nebalansirani skup podataka), visoka preciznost može biti obmanjujuća jer model može prepoznati nekolicinu TP primjera te prividno rezultirati visokim postotkom preciznosti, no ne znači nužno da model dobro prepoznaje sve TP primjere, jer većina TP primjera može ostati nepravilno klasificirana kao FN. Preciznost je definirana kao:

$$p = \frac{true\ positive}{true\ positive + false\ positive} = \frac{TP}{TP+FP} \tag{3-1}$$

3.2.3. Odziv

Prema formuli (3-2) odziv (engl. *Recall*) ili osjetljivost (engl. *Sensitivity*) definira se kao udio ispravno klasificiranih slučajeva u sveukupnom skupu stvarno pozitivnih slučajeva [45]. Visok odziv ukazuje kako je model sposoban prepoznati većinu piksela koji pripadaju ciljanoj klasi.

Kod nebalansiranih klasa, odziv može biti visok za dominantnu klasu, a niži za slabije zastupljene klase. Mogući su i slučajevi u kojima model ima visoki odziv, ali istovremeno i veliki broj lažno pozitivnih piksela što rezultira niskom preciznošću. Formula za određivanje odziva glasi:

$$r = \frac{true\ positive}{true\ positive + false\ negative} = \frac{TP}{TP+FN} \tag{3-2}$$

3.2.4. F1 ocjena i *Dice* koeficijent

F1 ocjena i *Dice* koeficijent (engl. *dice score*) su evaluacijske metrike koje su često korištene u području strojnog učenja, pogotovo kod problema segmentacije ili detekcije objekata. Prema formuli (3-3), F1 ocjena predstavlja harmonijsku sredinu između preciznosti i odziva. Kombiniranjem prethodno navedenih metrika dobiva se broj koji stvara balans između preciznosti i odziva. Visoka F1 ocjena ukazuje kako model zadovoljavajuće klasificira pozitivne primjere (preciznost) te ne propušta mnogo stvarno pozitivnih primjera (odziv). *Dice* koeficijent, poznat i pod nazivom Sørensen-Dice koeficijent, je metrika za mjerenje sličnosti između stvarnih i predviđenih segmentacija (3-4). Visoka vrijednost *Dice* koeficijenta ukazuje na veliku sličnost stvarne i predviđene segmentacije. Iako se i F1 ocjena i *Dice* koeficijent jednako izračunavaju, interpretiraju se na drugačije načine. Dok se F1 ocjena odnosi na prikazivanje balansa između preciznosti i odziva, *Dice* koeficijent prikazuje sličnost između stvarne i predviđene segmentacije koja se temelji na preklapanju tih dvaju skupova [46]. Formule za određivanje F1 ocjene te *Dice* koeficijenta su:

$$F1\ ocjena = \frac{2*preciznost*odziv}{preciznost+odziv}, \tag{3-3}$$

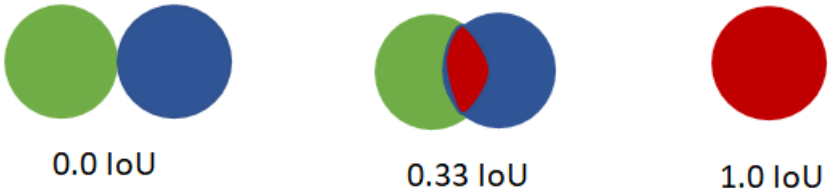
$$Dice\ koeficijent = \frac{2*TP}{2*TP+FN+FP} \tag{3-4}$$

3.2.5. IoU (Intersection over Union)

IoU je metrika koja je invarijantna na skalu, što podrazumijeva kako sličnost između dvaju proizvoljnih oblika A i B ne ovisi o mjerilu njihovog prostora. Navedena metrika se koristi za procjenu različitih zadataka u računalnom vidu poput segmentacije slika na razini piksela ili instanci te detekcije objekata u 2D ili 3D prostoru [46]. IoU metrika prikazuje koliko se dobro predviđena regija preklapa sa stvarnom regijom na slici. Pogledom na formulu (3-5), vidljivo je

kako se ova metrika izračunava kao omjer površine preklapanja predviđene i stvarne regije te njihove unije. U slučajevima kod kojih IoU iznosi skoro pa jedan, predviđena regija skoro pa u cijelosti pokriva stvarnu regiju (slika 3.2). Predstavljena metrika osjetljiva je na manje površine te površine nepravilnih i složenih oblika. Formula za izračun IoU metrike:

$$IoU\ (A, B) = \frac{A \cap B}{A \cup B} = \frac{TP}{TP+FP+FN} \tag{3-5}$$



Sl. 3.2. *Vizualni prikaz IoU metrike.*

Srednja vrijednost preklapanja unije (mIoU) predstavlja prosječni IoU svih klasa koji pruža brzi pregled učinkovitosti modela [45]. Definiran je izrazom (3-6):

$$mIoU\ (A, B) = \sum_i^N \frac{A_i \cap B_i}{A_i \cup B_i} = \frac{1}{N} \sum_i^N IoU_i \tag{3-6}$$

gdje su A_i *i* B_i skupovi predviđanja i oznaka (labela) za i-tu klasu, a N je ukupan broj klasa.

3.2.6. Točnost

Točnost (engl. *accuracy*) je jedna od najosnovnijih evaluacijskih metrika u području strojnog učenja. Formulom (3-7) prikazano je kako se točnost računa kao omjer ispravno klasificiranih primjera naprema ukupnog broja primjera unutar odabranog skupa podataka. Ukoliko su svi A_i i svi B_i jednaki, točnost će iznositi jedan. Riječ je o jednostavnoj i intuitivnoj metrici, no preporučljivo je koristiti ju u kombinaciji s ostalim metrikama radi dobivanja jasnijeg uvida u performanse modela. Ukoliko je većina primjera u skupu podataka negativna, model koji predviđa samo negativne primjere može imati izrazito veliku točnost, a u praksi loše funkcionirati [46]. Točnost se definira kao:

$$Točnost(A, B) = \frac{1}{N} \sum_{i=0}^{N} 1_{(A_i = B_i)} = \frac{TP + TN}{N} \tag{3-7}$$

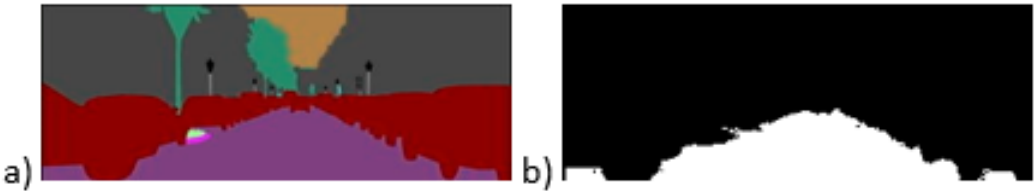
3.3. Evaluacija odabranih algoritama za semantičku segmentaciju voznog područja

U nastavku ovoga poglavlja bit će analizirane performanse triju modela [29], [30], [31] koji su ranije opisani unutar podpoglavlja 2.3. Svi odabrani algoritmi trenirani su na trima različitim skupovima podataka: KITTI, Cityscapes i CamVid. Na istrenirane modele primijenjene su evaluacijske metrike opisane podpoglavljem 3.3., radi pronalaska modela koji najbolje rješava problem semantičke segmentacije voznog područja.

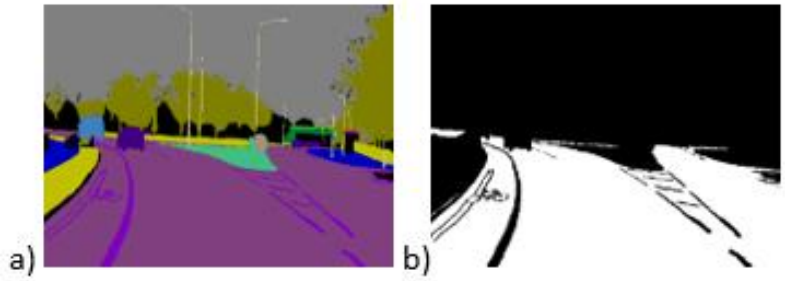
3.3.1. Prilagodba skupova podataka za rješenje problema semantičke segmentacije voznog područja

U svrhu korištenja skupova podataka za treniranje i testiranje odabranih algoritama, potrebno je odraditi prilagodbu slika, tj. maski pojedinog skupa podataka. Skup podataka KITTI prvotno je sadržavao maske kod kojih je vozno područje označeno ljubičastom bojom, okolina crvenom, a nedefinirana područja crnom bojom. Kod Cityscapes skupa podataka klasa od interesa, od njih 30, bila je klasa označena tamnoljubičastom bojom koja predstavlja kolnik. CamVid skup podataka sadrži dvije različite klase za područje kolnika, jedna klasa, označena ružičastom bojom, predstavlja kolnik, dok su oznake na kolniku označene ljubičastom bojom. Kao što je vidljivo na slikama 3.3., 3.4. i 3.5., maske svih triju skupova preoblikovane su u binarne slike kod kojih binarne jedinice predstavljaju vozno područje (kod CamVid skupa ono ne uključuje kolničke oznake), a pozadina je predstavljena s nulama.

Sl. 3.3. *Primjer binarizacije KITTI maske. a) originalna maska, b) binarna maska*

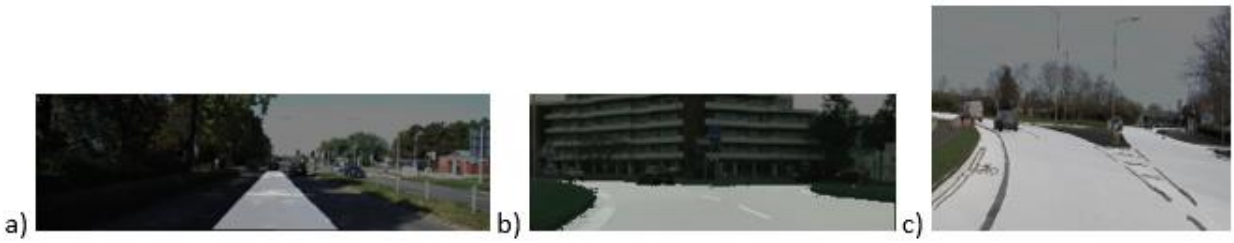


Sl. 3.4. *Primjer binarizacije Cityscapes maske. a) originalna maska, b) binarna maska*



Sl. 3.5. *Primjer binarizacije CamVid maske. a) originalna maska, b) binarna maska*

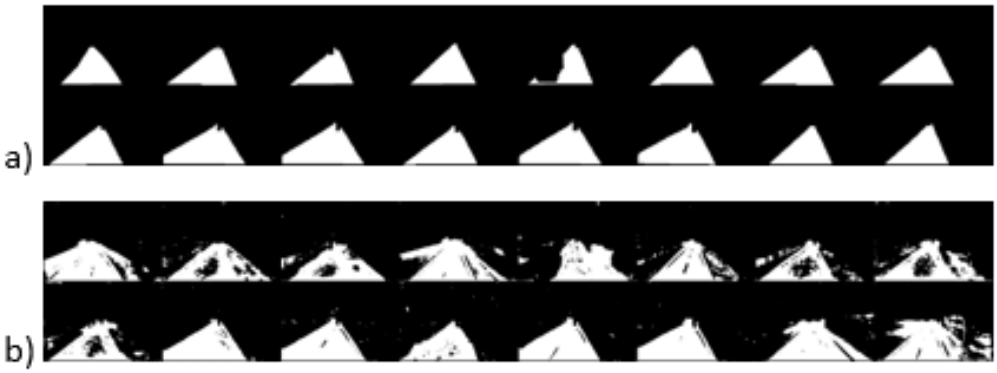
Nakon prilagodbe skupova podataka, odrađena je provjera ispravnosti preklapanja područja od interesa binarnih maski s područjima od interesa ulaznih slika (engl. *overlay*) (sl.3.6.).



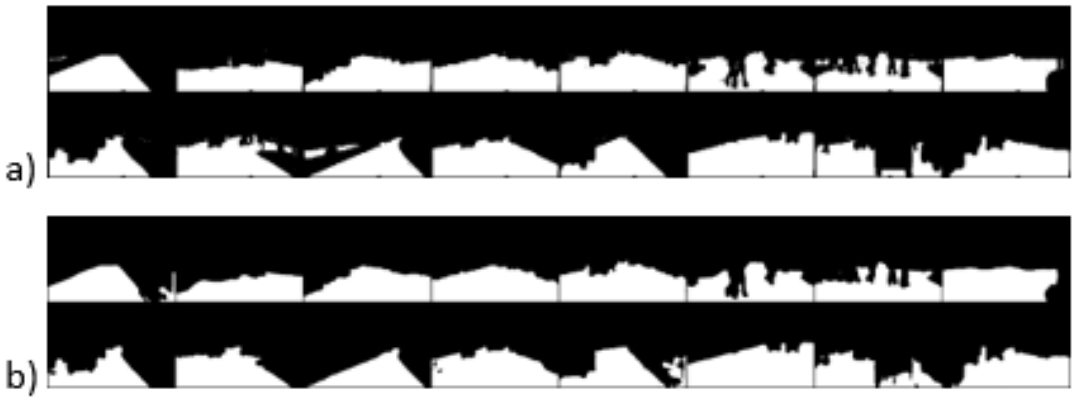
Sl. 3.6*. Prikaz preklapanja ulazne slike i binarne maske na: a) KITTI, b) Cityscapes, c) CamVid skupu podataka*

3.3.2. Evaluacija i analiza performansi odabranih postojećih algoritama

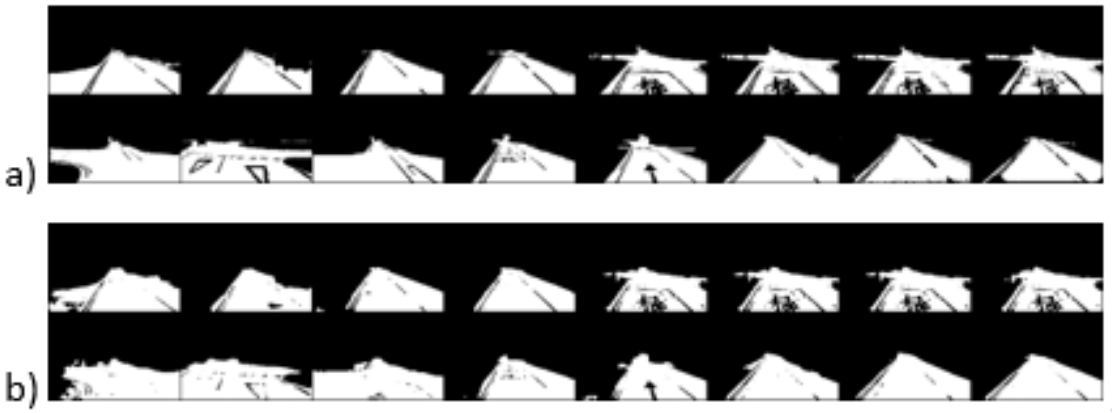
Prvi evaluirani algoritam [31], zasnovan na arhitekturi UNET, obučen je tijekom 20 epoha na različitim skupovima podataka, uključujući KITTI, Cityscapes i CamVid skup podatka. S 31.036.481 parametara za treniranje, veličina modela iznosi 483.78 megabajta, dok je veličina grupe postavljena na 16. Ulazne slike su skalirane na dimenzije 128x512x3 kako bi bile prilagođene modelu. Tijekom treniranja korišten je optimizacijski algoritam Adam sa stopom učenja od 0.0001. Treniranje se izvodi na GPU-u ukoliko je isti dostupan, u suprotnome se koristi CPU. Model koristi tehnike augmentacije podataka kao što su rotacija, horizontalno i vertikalno zrcaljenje radi poboljšanja robusnosti modela. Tijekom treniranja, praćenje gubitka izvodi se pomoću funkcije za gubitak binarne križne entropije s težinama za ravnotežu klasa. Nakon svake epohe, model se evaluira na validacijskom skupu podataka kako bi se omogućilo praćenje točnosti. Također, predviđanja modela spremaju se u obliku slike radi vizualizacije performansi (sl.3.7. - sl.3.9.).



Sl. 3.7. *Vizualizacija rezultata treniranja na KITTI skupu podataka; a) temeljna istina, b) predviđanje semantic_model modela*



Sl. 3.8. *Vizualizacija rezultata treniranja na Cityscapes skupu podataka; a) temeljna istina, b) predviđanje semantic_model modela*



Sl. 3.9. *Vizualizacija rezultata treniranja na CamVid skupu podataka; a) temeljna istina, b) predviđanje semantic_model modela*

Iz tablice 3.2. vidljivo je kako model postiže najbolje rezultate na CamVid skupu podataka, s visokim vrijednostima preciznosti, F1 ocjene, *Dice* koeficijenta i IoU-a. Na Cityscapes skupu podataka model postiže visoku preciznost, dok najbolju točnost pokazuje na validacijskom skupu KITTI. Vrijeme inferencije odnosi se na vrijeme potrebno modelu za obradu jedne ulazne slike.

Tab.3.2. *Rezultati evaluacije algoritma iz [31] na KITTI, Cityscapes i CamVid skupu podatka.*

Skup podataka	Broj trening slika	Broj slika za validaciju	Broj testnih slika	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost	Vrijeme inferencije [s]
KITII	261	28	290	$\begin{bmatrix} 714225 & 252955 \\ 57226 & 810602 \end{bmatrix}$	0.76216	0.93405	0.83939	0.83939	0.72324	0.8536	0.06106
Cityscapes	2975	500	-	$\begin{bmatrix} 21711210 & 1557360 \\ 3513856 & 5985574 \end{bmatrix}$	0.79354	0.63009	0.70244	0.70243	0.54135	0.7123	0.03996
CamVid	369	100	232	$\begin{bmatrix} 2939560 & 428005 \\ 513936 & 2672099 \end{bmatrix}$	0.86124	0.83869	0.85016	0.85016	0.73936	0.7525	0.02564

Mjerenje vremena inferencije za svaki od skupova podataka pokazuje da su vremena znatno ispod jedne sekunde, što ukazuje na prilično brz rad modela (podaci mjereni na CPU).

Algoritam [30] predstavlja algoritam semantičke segmentacije koji je također treniran na skupovima podataka KITTI, Cityscapes i CamVid. Broji 1.941.105 parametara za treniranje s veličinom modela od 7.40 megabajta. Model je treniran tijekom 25 epoha, a veličina ulazne slike iznosi 128x128x3. Tijekom treniranja modela korištena je funkcija gubitka binarne unakrsne entropije koja se smanjuje tijekom postupka optimizacije. Osim toga, praćenje performansi modela i sprječavanje prenaučenosti ostvareno je praćenjem promijene metrike točnosti tijekom procesa treniranja.

Prema tablici 3.3., najveći odziv, F1 ocjenu, Dice koeficijent i točnost na validacijskom skupu podataka model postiže na Cityscapes skupu podataka. Najmanje vrijeme inferencije ima na CamVid skupu podataka, ali je za razliku od prethodnog modela vrijeme obrade podjednako za CamVid i Cityscapes skupove podataka.

Tab. 3.3. *Rezultati evaluacije algoritma [30] na KITTI, Cityscapes i CamVid skupu podatka.*

Skup podataka	Broj slika za trening	Validacija	Broj testnih slika	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost	Vrijeme inferencije [s]
KITII	261	28	290	$\begin{bmatrix} 364798 & 4717 \\ 9650 & 79587 \end{bmatrix}$	0.94405	0.89187	0.91721	0.91721	0.84708	0.9613	0.11145
Cityscapes	2975	500	-	$\begin{bmatrix} 5947838 & 170230 \\ 148802 & 1925130 \end{bmatrix}$	0.91876	0.92825	0.92348	0.92348	0.85784	0.9690	0.04611
CamVid	369	100	232	$\begin{bmatrix} 1049723 & 32404 \\ 66388 & 489885 \end{bmatrix}$	0.93795	0.88065	0.90841	0.90841	0.83218	0.94366	0.04199

Algoritam koristi optimizacijski algoritam Adam, koji se prilagođava tijekom procesa učenja radi postizanja boljih performanse. Adam optimizator koristi informacije o prvom i drugom momentu gradijenta koje mu služe za ažuriranje parametara modela.

Radi vizualizacije performansi modela korištena je petlja za prikaz rezultata na validacijskom skupu, pri čemu se prikazuju originalne slike, stvarne maske i predviđanje modela za nekoliko odabranih primjera (sl.3.10. – sl.3.12.). Takav način očitavanja performansi nije u potpunosti pouzdan, no omogućuje korisniku vizualizaciju rada modela na odvojenom skupu podataka.

***Sl. 3.10**. Vizualizacija rezultata treniranja modela iz [30] na KITTI skupu podataka*

***Sl. 3.11**. Vizualizacija rezultata treniranja modela iz [30] na Cityscapes skupu podataka*

***Sl. 3.12**. Vizualizacija rezultata treniranja modela iz [30] na CamVid skupu podataka.*

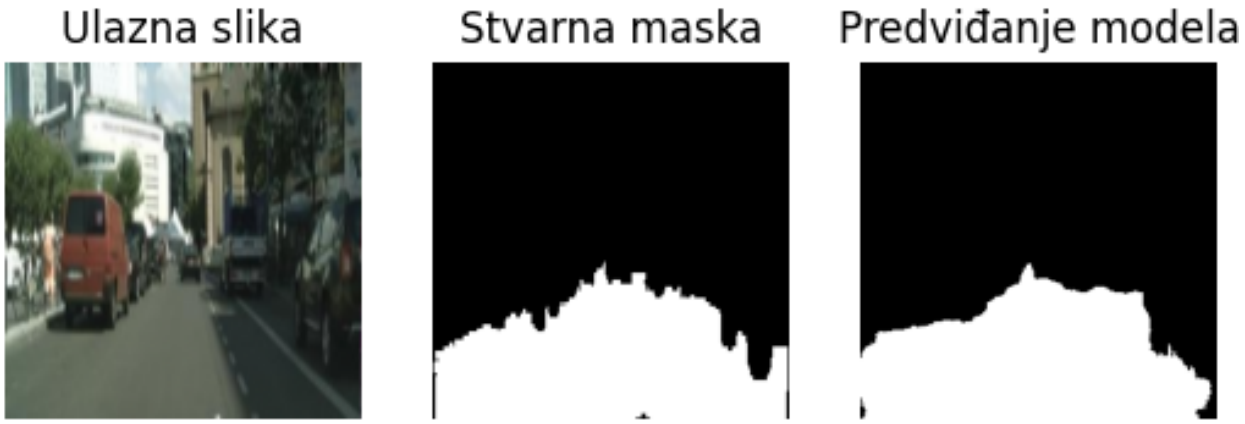
Kao i prethodno opisana dva algoritma, i algoritam [29] je treniran na skupovima podataka KITTI, Cityscapes i CamVid te koristi konvolucijsku neuronsku mrežu za semantičku segmentaciju slika. Ovaj model sadrži ukupno 2.058.690 parametara za treniranje, a veličina modela iznosi 7.85 megabajta. Veličina ulazne slike u model je 256x256 piksela. Sve slike podijeljene su u odvojene skupove za treniranje i validaciju. Slike su normalizirane na raspon [0, 1] kako bi se poboljšala konvergencija modela za vrijeme treniranja. Tijekom treniranja modela korištena je binarna unakrsna entropija kao gubitak, optimizator *RMSprop* i evaluacijska metrika točnosti. *RMSprop* (engl. *Root Mean Square Propagation*) služi za prilagodljivo podešavanje stope učenja svakog parametra modela, ovisno o povijesti gradijenta tih parametara. Ujedno se primjenjuju i pozivne funkcije kao što su smanjenje stope učenja i spremanje najboljeg modela na temelju gubitka nad validacijskom skupu. Broj epoha postavljen je na 50. Veličina grupe, koja predstavlja broj uzoraka korištenih za izračun gubitka i ažuriranje parametara modela u svakoj iteraciji, iznosi 16. Povećani broj epoha može poboljšati performanse modela, ali isto tako može dovesti i do prenaučenosti modela. Veličina grupe utječe na stabilnost treninga i brzinu konvergencije modela. Veći veličina grupe može ubrzati trening, ali i zahtijevati više memorije, dok manja može rezultirati nestabilnim gradijentima.

Performanse modela evaluirane su na validacijskom skupu koristeći metrike preciznosti, odziva, F1 ocjene, *Dice* koeficijenta, IoU i točnosti. Iz tablice 3.4. može se uočiti kako su najbolji rezultati u svim ovim metrikama postignuti na Cityscapes skupu podataka. Vrijeme inferencije modela najmanje je na Cityscapes skupu podataka, dok je najveće vrijeme inferencije zabilježeno na KITTI skupu podataka u iznosu od 0.07203 sekundi. Radi vizualizacije performansi modela korištena je petlja za prikaz rezultata na validacijskom skupu, pri čemu se prikazuju originalne slike, stvarne maske i predviđanje modela za nekoliko odabranih primjera (sl.3.13. – sl.3.15.).

Tab. 3.4. *Rezultati evaluacije algoritma iz [29] na KITTI, Cityscapes i CamVid skupu podatka.*

Skup podataka	Broj slika za trening	Validacija	Broj testnih slika	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost	Vrijeme inferencije [s]
KITII	261	28	290	$\begin{bmatrix} 1526154 & 61 \\ 308111 & 682 \end{bmatrix}$	0.91790	0.00220	0.004406	0.00440	0.00220	0.8320	0.07203
Cityscapes	2975	500	-	$\begin{bmatrix} 24334691 & 548472 \\ 511770 & 7373067 \end{bmatrix}$	0.930762	0.93594	0.932923	0.93292	0.87428	0.96764	0.05411
CamVid	369	100	232	$\begin{bmatrix} 4346472 & 258946 \\ 74995 & 1873187 \end{bmatrix}$	0.87855	0.96151	0.91816	0.91816	0.84869	0.96934	0.05959

***Sl. 3.13**. Vizualizacija rezultata treniranja modela iz [29] na KITTI skupu podataka*

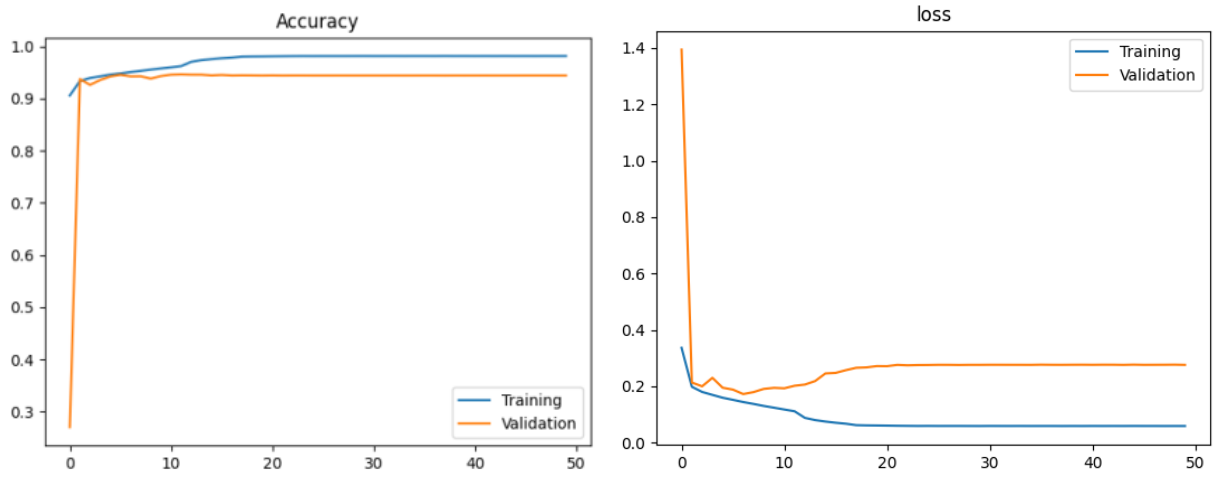


***Sl. 3.14**. Vizualizacija rezultata treniranja modela iz [29] na Cityscapes skupu podataka*

***Sl. 3.15**. Vizualizacija rezultata treniranja modela iz [29] na CamVid skupu podataka*

3.3.3. Pojednostavljenje odabranog modela i njegova evaluacija

Između treniranih i evaluiranih algoritam [29], [30], [31] najboljim algoritmom pokazao se algoritam [29] treniran na Cityscapes skupu podatka s preciznošću od 93.07%, odzivom 93.59%, F1 ocjenom i Dice koeficijentom od 93.29% te s IoU od 87.43%. Točnost modela na validacijskom skupu podataka, mjerena zajedno s gubitkom tijekom procesa treninga (sl. 3.20), iznosi 96.76%. Spomenuti model odabran je za daljnju obradu te je u sklopu ovog diplomskog rada podvrgnut procesu pojednostavljenja modela kako bi postao optimalniji za implementaciju na ugradbeni računalni sustav. Proces pojednostavljenja obuhvaća promjenu različitih hiperparametara i kombiniranje dimenzija ulaznih slika, broja i dimenzija korištenih filtera te varijacije korištenih slojeva neuronske mreže. U nastavku rada, prikazane su sve varijante odabranog modela istražene u sklopu diplomskog rada, pojašnjene su promjene koje su izvedene, prikazani su rezultati evaluacije te je dan po jedan vizualni primjer izlaznog podatka svakog od modela na validacijskom skupu podataka. Validacijski skup podataka sadrži 500 slika, dok trening skup podatka nakon augmentacije broji 8925 slika. Slika 3.16 prikazuje točnosti i gubitak prilikom treniranja izvorne verzije modela [29].



Sl. 3.16. *Prikaz točnosti i gubitka prilikom treniranja izvorne verzije modela [29] tijekom 50 epoha*

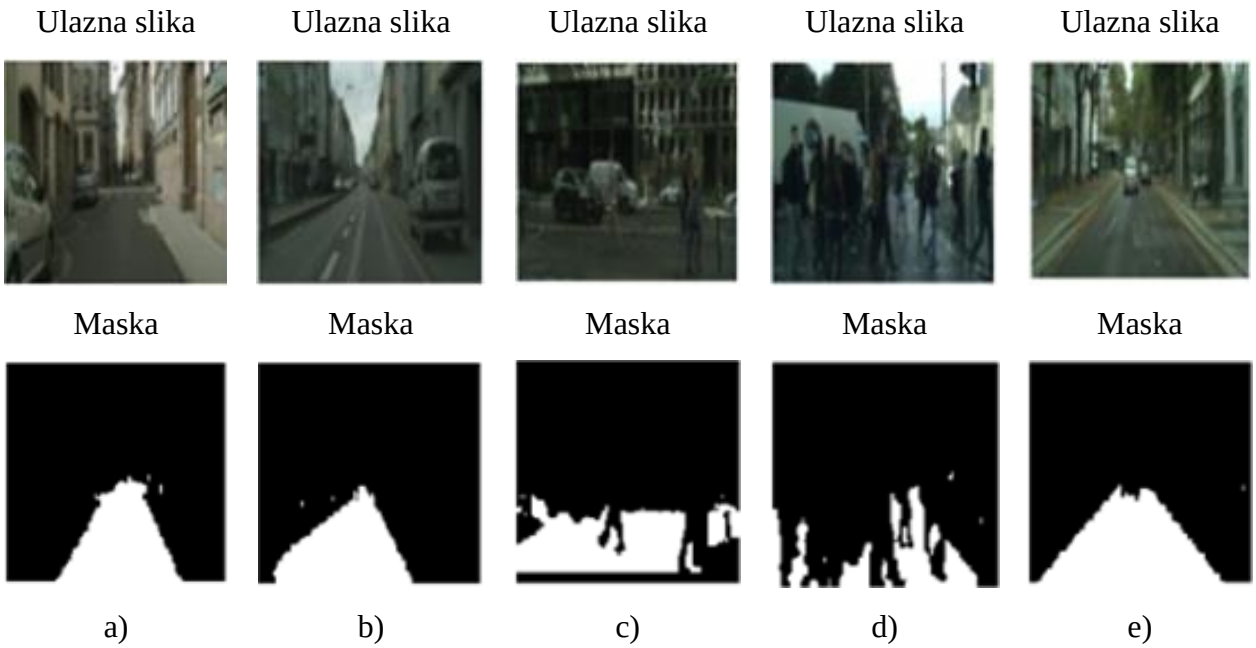
Originalni model iz [29], u kodu označen s *get_model()* funkcijom, dizajniran je s ulaznim dimenzijama od 256x256 piksela i 3 kanala (RGB format). Glavno je obilježje modela njegova uporaba standardnih *Conv2D* (konvolucijskih) slojeva na početku mreže, koja omogućava izvlačenje osnovnih značajki iz slika. Nakon početnih slojeva, model koristi *SeparableConv2D* slojeve, koji su zaslužni za efikasnost pri obradi, smanjujući potrebne resurse i vrijeme izvođenja

bez značajnog gubitka u performansama. Tijekom faze smanjivanja dimenzija (engl. *downsamplinga),* model koristi *MaxPooling2D* slojeve kako bi smanjio dimenzije izlaznih značajki, čime je dodatno smanjena kompleksnost modela. U svrhu ponovnog povećanja dimenzije izlaza i detaljnije rekonstrukcije slike, implementirano je povećanje dimenzija (engl. *upsampling*) uporabom *UpSampling2D* i *Conv2DTranspose* slojeva. Model prolazi kroz tri razine dubine, gdje se "razine dubine" odnose na različite slojeve konvolucijske neuronske mreže s povećanjem broja filtera. Konkretno, model započinje s brojem filtera postavljenih na 64, zatim se povećava na 128, a potom na 256 filtera prije početka povećanja dimenzija. Tijekom faze povećanja dimenzija, model završava s konfiguracijom filtera postavljenom na 256, 128, 64 te 32, čime se omogućuje precizna rekonstrukcija željene maske.

Izlazni sloj modela koristi *softmax* aktivacijsku funkciju, što ukazuje na to da je model prvotno korišten za rješavanje problema klasifikacije više klasa. Međutim, u ovom kontekstu, model je prilagođen za semantičku segmentaciju gdje se koristi samo jedan izlazni kanal, s maskama tipa podataka *float32* i jedinstvenim vrijednostima [0, 1], koje predstavljaju binarnu klasifikaciju svakog piksela slike.

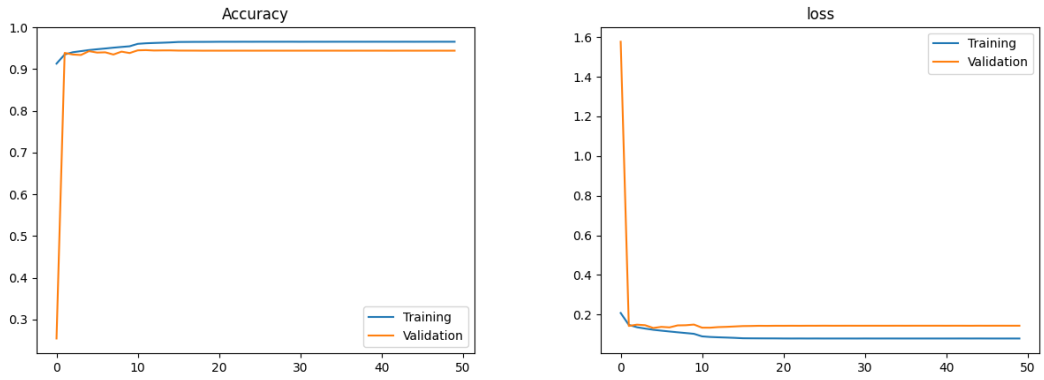
Za poboljšanje modela i njegove sposobnosti generalizacije, kako na izvornom modelu tako i na svim pojednostavljenim verzijama, primijenjena je augmentacija na skupu podataka za trening. Trening skup, koji je prvotno sadržavao 2975 slika, nakon augmentacije sadrži 8925 slika s dimenzijama (256, 256, 3) za ulazne slike i (256, 256, 1) za maske. Augmentacija uključuje različiti stupanj zumiranja te promjene svjetline slika, čime se simuliraju stvarni uvjeti pri kojima bi model mogao biti primijenjen, poput noćne vožnje, segmentacije kolnika na manjoj ili većoj udaljenosti od vozila. Augmentacija zumiranja nasumično skalira sliku s faktorom skaliranja u rasponu od 0.8 do 1.2 za oba skupa (ulazne slike s odgovarajućim maskama), simulirajući promjene u udaljenosti kolnika od kamere. Korištenjem *iaa.Multiply* s rasponom [0.7, 1.3] nastaje promjenjivo osvjetljenje slika, čime je slike moguće potamniti ili posvijetliti, simulirajući različite uvjete osvjetljenja. Množenjem s vrijednostima manjim od 1 smanjuje se osvjetljenje (do 30%), dok množenjem s vrijednostima koje su veće od 1 dolazi do povećanja osvjetljenja (do 30%). Navedene tehnike augmentacije omogućuju modelu efikasniju prilagodbu različitim scenarijima tijekom vožnje te poboljšanje vlastitih performansi (programski kod prikazan u prilogu P.4.1). Nakon dodavanja augmentacije, obavljena je provjera ispravnosti trening skupa podataka. Slika 3.17 prikazuje nekoliko primjera ulaznih slika s odgovarajućim maskama koje su nastale nakon primjene augmentacije na originalni trening skup podataka.

Prilagodba modela za semantičku segmentaciju za implementaciju na ugradbeni sustav ostvarena je iterativnim postupkom poboljšanja i promjene arhitekture modela, čime je ostvareno osam različitih verzija modela. Prve četiri verzije modela koriste konvolucijske slojeve s filtrima dimenzija 3x3, ali se razlikuju po dimenziji ulazne slike, broju filtera te strukturi i dubini slojeva.



Sl. 3.17. *Primjeri podataka koji su nastali nakon augmentacije: a) povećanje svjetline, b)smanjenje svjetline, c) smanjenje svjetline, d) augmentacija skaliranja prema gore (simulacija približavanja), e)augmentacija skaliranja prema dolje (simulacija udaljavanja)*

Prva verzija modela pod nazivom *Model_1* koristi ulazne slike dimenzija 128x128x3. Na početku se nalaze dva konvolucijska sloja sa 64 i 128 filtera. Nakon toga, slijedi faza *upsamplinga* koja završava s konvolucijskim slojevima koji imaju 128, 64 i 32 filtera, omogućujući time preciznu rekonstrukciju izlazne slike. Ovaj model uvodi mehanizam za rano prekidanje dodavanja novih slojeva u slučaju kada dimenzije značajki postanu manje ili jednake 16x16. *Model_1* koristi sigmoid aktivaciju u izlaznom sloju, implicirajući na njegovu primjenu za binarne segmentacijske zadatke. Veličina *Modela_1* je 1.94 megabajta s ukupno 509217 parametara, od čega je 507489 parametara za treniranje, a ostalih 1728 parametara čine parametri koji se ne ažuriraju tijekom treninga, ali su dio modela (parametri iz slojeva poput *BatchNormalization* ili parametri slojeva koji su zamrznuti). Slika 3.18. prikazuje graf promjene točnosti i gubitka tijekom procesa učenja, dok se na slici 3.19. prikazuju ulazne slike sa stvarnim maskama te predviđanjima modela *Model_1* za nekoliko odabranih primjera iz validacijskog skupa podataka.



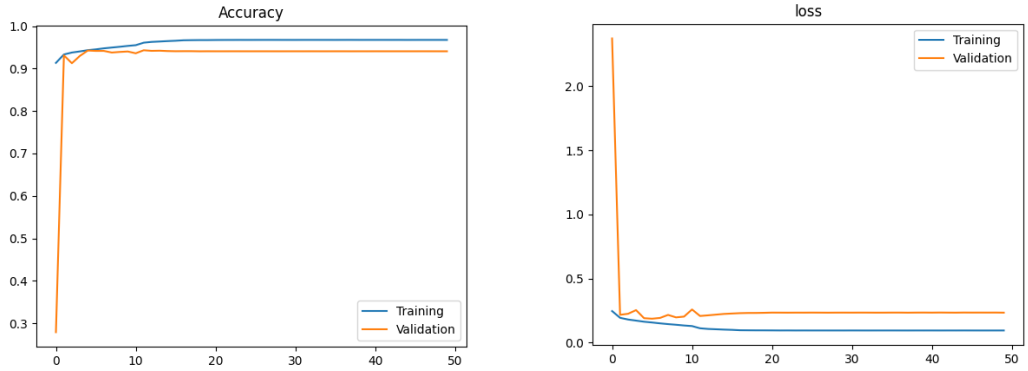
Sl. 3.18. *Prikaz točnosti i gubitka prilikom treniranja modela Model_1 tijekom 50 epoha*

Sl. 3.19. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_1*

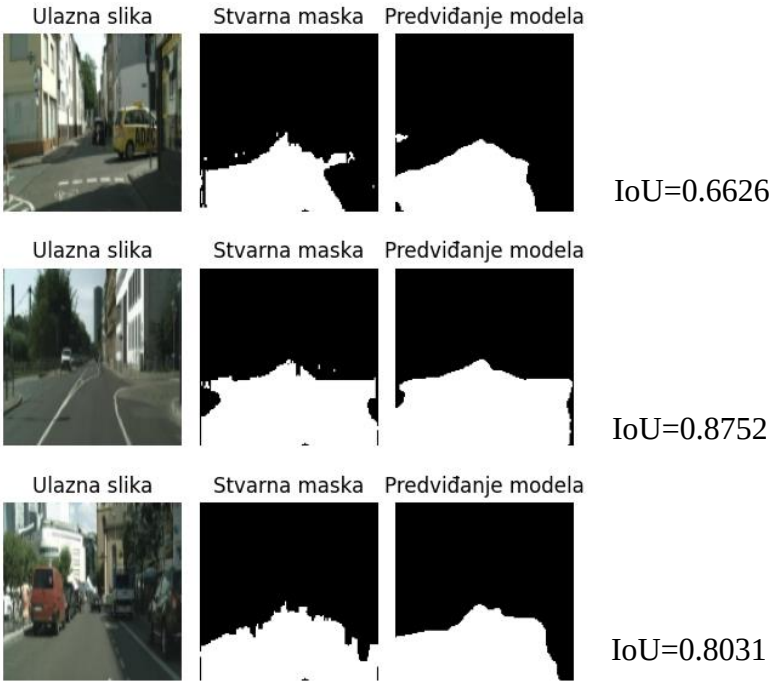
Detaljniji rezultati o performansama modela *Model_1* nalaze se u tablici 3.6.

Sljedeća verzija modela, s fokusom na efikasnost, je *Model_2*. Slično kao *Model_1*, i *Model_2* obrađuje slike dimenzija 128x128x3. Međutim, kod *Modela_2* smanjen je broj filtera u prvom *Conv2D* sloju s 32 na 16. Ovaj model koristi različite brojeve filtera u slojevima *SeparableConv2D* i *Conv2DTranspose*, prateći strukturu [64, 128] u prvom dijelu i [128, 64, 32] tijekom *upsamplinga*. Dodatno, *Model_2* uvodi dodatni *Conv2D* sloj sa 64 filtra neposredno

prije izlaznog sloja. Veličina *Modela_2* je 2 megabajta, s ukupnim brojem parametara 525553, od kojih su 523729 parametara namijenjena treniranju uz 1824 netrenirajućih parametara. Slike 3.20. i 3.21. prikazuju graf promjene točnosti i gubitka tijekom procesa učenja na *Modelu_2* te vizualni prikaz ulaznih slika sa stvarnim maskama te predviđanjima modela *Model_2* za nekoliko odabranih primjera iz validacijskog skupa podataka.



Sl. 3.20. *Prikaz točnosti i gubitka prilikom treniranja modela Model_2 tijekom 50 epoha*

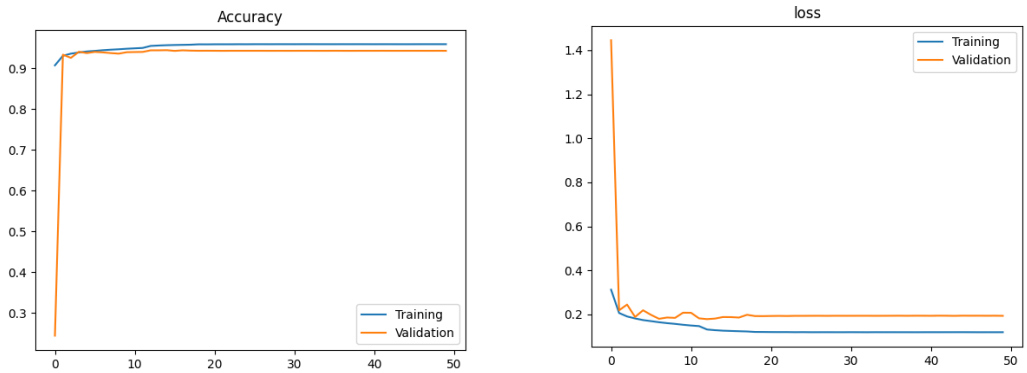


Sl. 3.21. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_2*

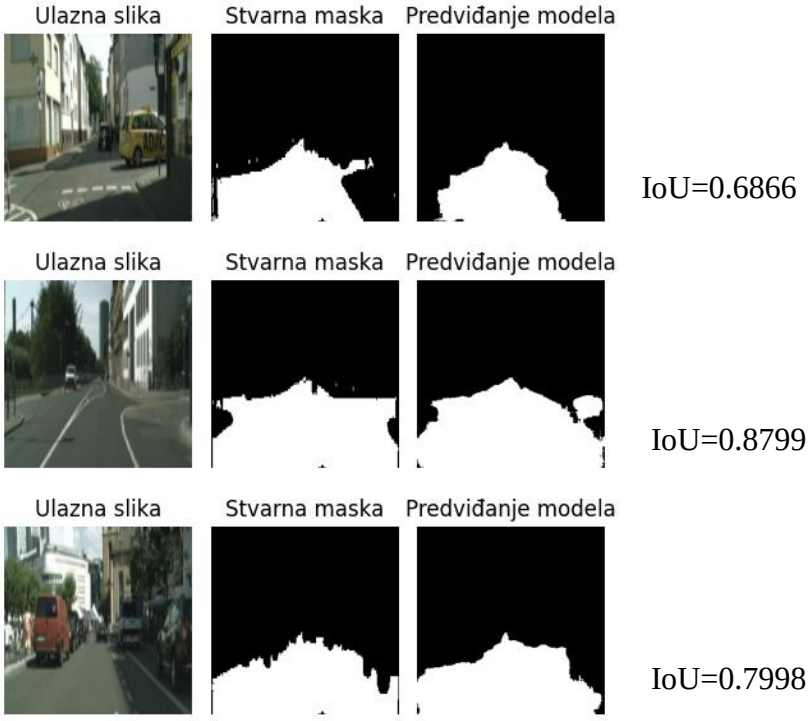
Detaljniji rezultati o performansama modela *Model_2* nalaze se u tablici 3.6.

Model_3 mijenja arhitekturu povećanjem dimenzija ulaznih slika na 256x256x3, zadržavajući istu strukturu smanjenja i povećanja broja filtera kao *Model_2*. Ovaj pristup omogućava detaljniju analizu slika većih rezolucija. *Model_3* je optimiziran u smislu veličine, s 1.93 megabajta i ukupno 506513 parametara, od čega 504817 parametara zahtijeva treniranje, a

ostalih 1696 parametara ne zahtjeva. *Model_3* ne sadrži dodatni *Conv2D* sloj koji je prisutan u *Modelu_2*, što doprinosi manjem broju parametara. Slika 3.22. prikazuje graf promjene točnosti i gubitka tijekom procesa učenja, dok se na slici 3.23. prikazuju ulazne slike sa stvarnim maskama te predviđanjima modela *Model_3* za nekoliko odabranih primjera iz validacijskog skupa podataka.



Sl. 3.22. *Prikaz točnosti i gubitka prilikom treniranja modela Model_3 tijekom 50 epoha*

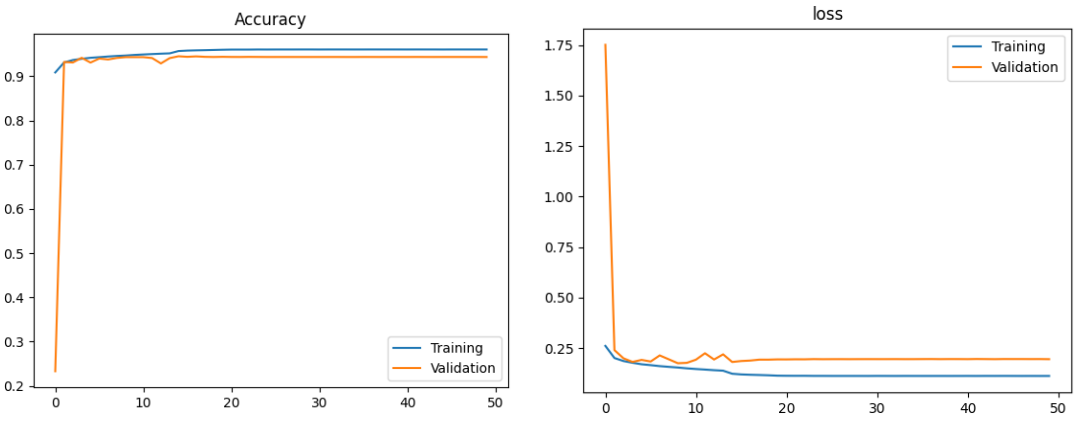


Sl. 3.23. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_3*

Detaljniji rezultati o performansama modela *Model_3* nalaze se u tablici 3.5.

Model_4 dijeli iste ulazne dimenzije i strategiju povećanja i smanjenja broja filtera s *Modelom_3*, ali se na njegovu arhitekturu dodaje još jedan *Conv2D* sloj s 64 filtra prije samog izlaznog sloja, povećavajući time njegovu sposobnost učenja složenijih značajki slika. Veličina *Modela_4* je 2 megabajta s ukupnim brojem parametara od 525553, od kojih 523729 ulazi u

trening parametre, a 1824 ne ulazi u trening parametre. Slika 3.24. prikazuje graf promjene točnosti i gubitka tijekom procesa učenja, dok se na slici 3.25. prikazuju ulazne slike sa stvarnim maskama te predviđanjima modela *Model_4* za nekoliko odabranih primjera iz validacijskog skupa podataka.



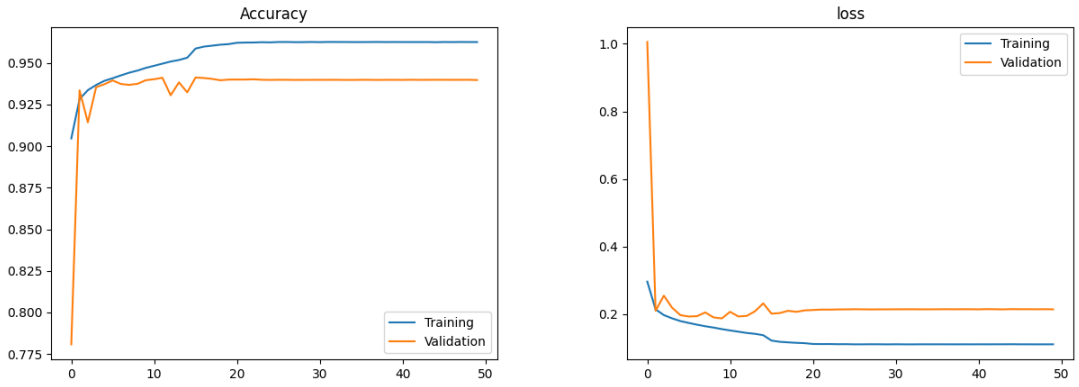
Sl. 3.24. *Prikaz točnosti i gubitka prilikom treniranja modela Model_4 tijekom 50 epoha*

Sl. 3.25. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_4*

Detaljniji rezultati o performansama modela *Model_4* nalaze se u tablici 3.5.

Preostala četiri dodatna pristupa unose nove varijacije u dizajn mreže. Ovi modeli eksperimentiraju s manjim veličinama filtera i povećanjem broja slojeva te različitim dimenzijama ulaznih slika.

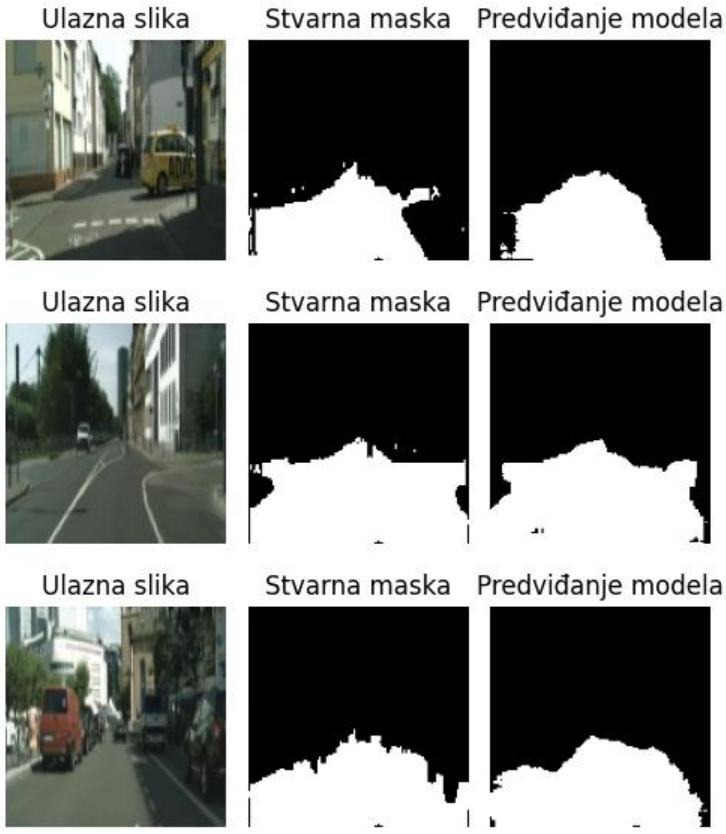
Prvi od tih modela je model pod nazivom *Model_5* koji stavlja fokus na manje dimenzije filtera. Model koristi ulazne slike dimenzija 128x128 piksela i uvodi manje filtere dimenzija 2x2, za razliku od prethodno korištenih 3x3 filtera (programski kod arhitekture modela nalazi se u prilogu P.4.2). Takva promjena rezultira modelom značajno manje veličine; od 1.02 megabajta s ukupnim brojem parametara 266497, od kojih je 264769 parametara dostupno za treniranje, a ostalih 1728 parametara ne ulazi u proces treniranja. Smanjenje veličine filtra stavlja fokus na finije značajke slike u manjem mjerilu. Slika 3.26. prikazuje graf promjene točnosti i gubitka tijekom procesa učenja (programski kod obrade trening podataka prikazan u prilogu P.4.3), dok se na slici 3.27. prikazuju ulazne slike sa stvarnim maskama te predviđanjima modela *Model_5* za nekoliko odabranih primjera iz validacijskog skupa podataka.



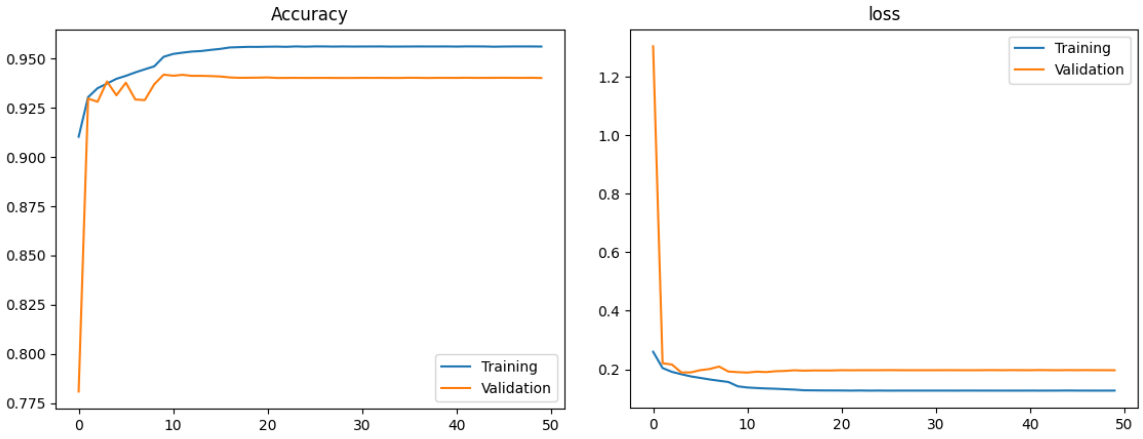
Sl. 3.26. *Prikaz točnosti i gubitka prilikom treniranja modela Model_5 tijekom 50 epoha*

Detaljniji rezultati o performansama modela *Model_5* nalaze se u tablici 3.6.

Model_6, slično kao i *Model_5*, koristi 2x2 filtere na ulaznim slikama 128x128, no ovaj model povećava složenost uvođenjem dodatnog konvolucijskog sloja sa 64 filtra veličine 2x2 neposredno prije izlaznog sloja. Takva promjena uzrokuje povećanje veličine modela na 1.05 megabajta te povećanje ukupnog broja parametara na 275137, od kojih je 273281 parametara dostupno za treniranje, a 1856 parametara nije dostupno. Ova dodatna složenost omogućava modelu bolje učenje kompleksnih značajki. Slika 3.28. prikazuje graf promjene točnosti i gubitka tijekom procesa učenja, dok se na slici 3.29. prikazuju ulazne slike sa stvarnim maskama te predviđanjima modela *Model_6* za nekoliko odabranih primjera iz validacijskog skupa podataka.

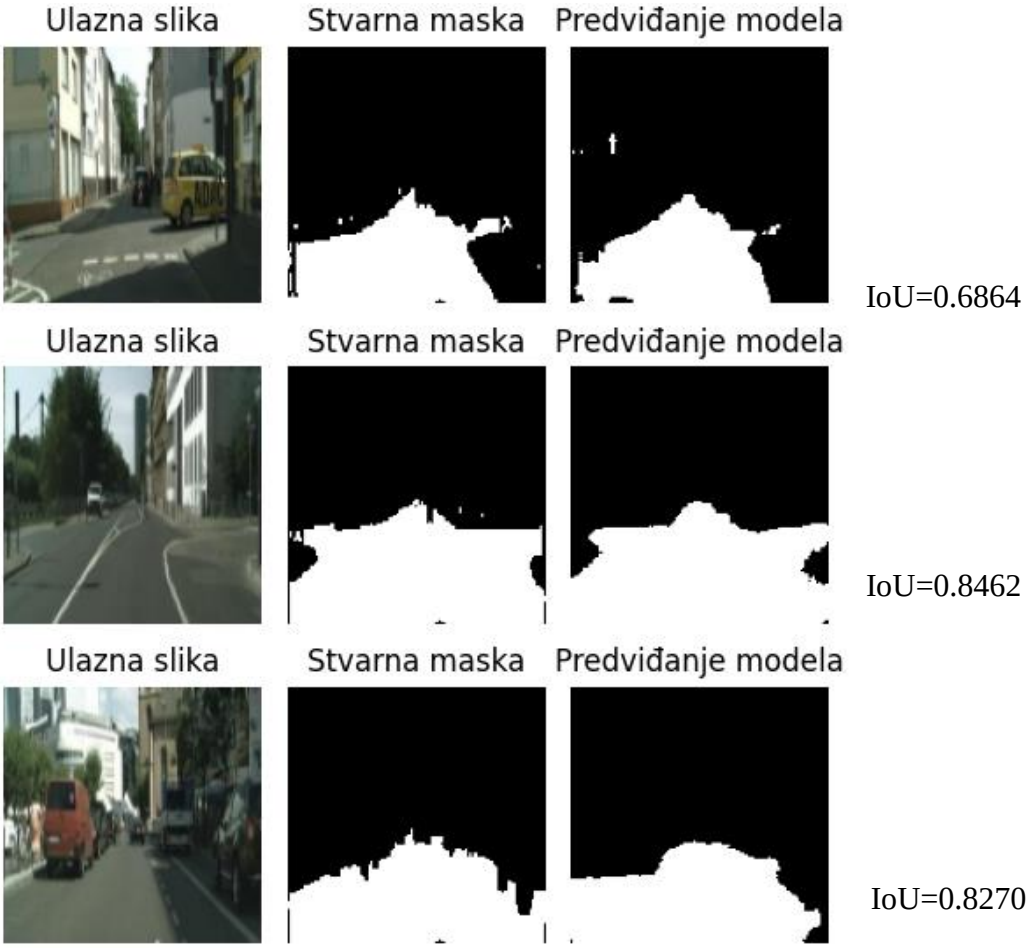


Sl. 3.27. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_5*



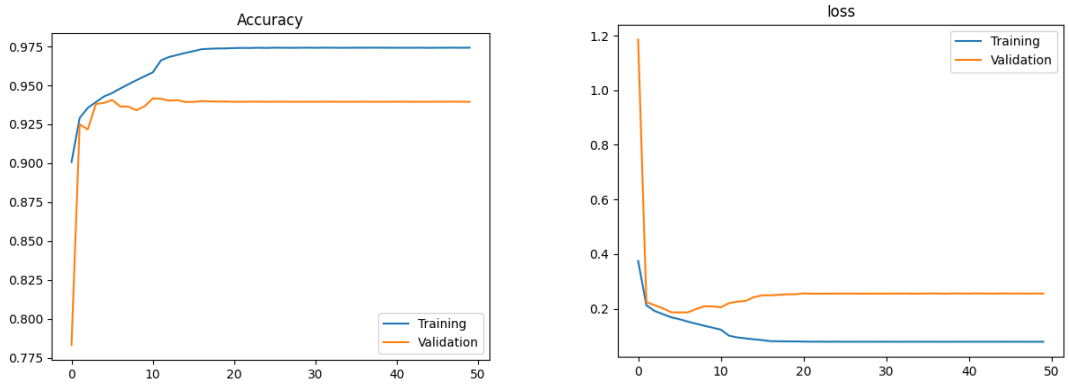
Sl. 3.28. *Prikaz točnosti i gubitka prilikom treniranja modela Model_6 tijekom 50 epoha*

Detaljniji rezultati o performansama modela *Model_6* nalaze se u tablici 3.6.

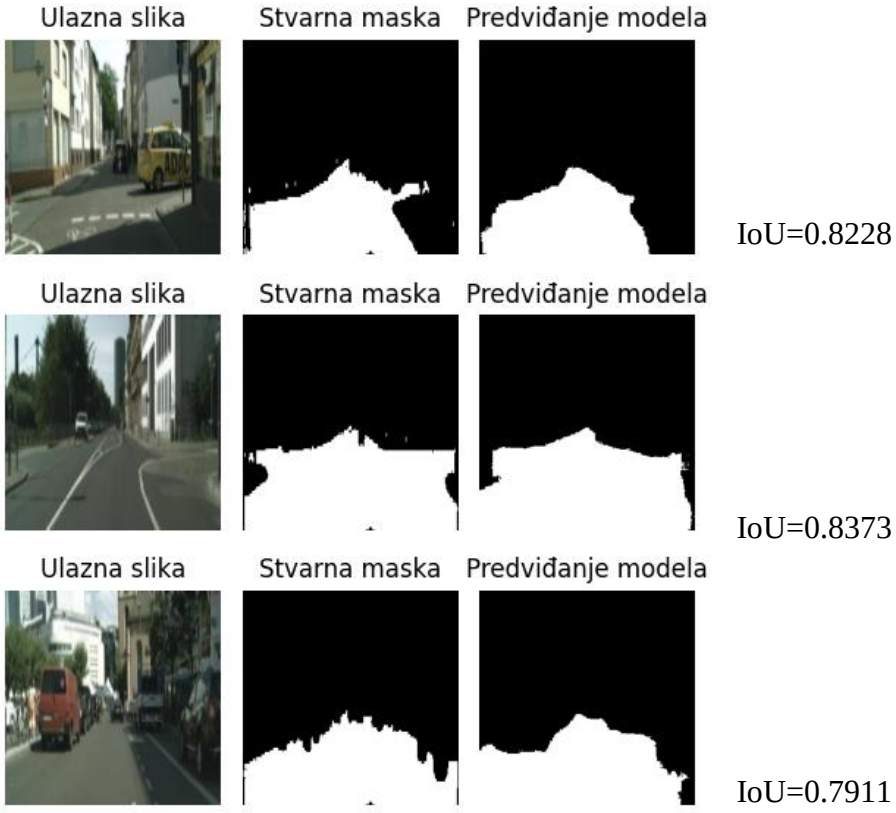


Sl. 3.29. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_6*

Model_7 se razlikuje od modela *Model_5* i *Model_6* dimenzijom ulaznih slika koje iznose 256x256 piksela, uz održavanje manjih 2x2 filtera. Veće dimenzije slike omogućavaju modelu obradu većeg broja informacija, dok manji filtri omogućuju detaljniju analizu. Ukupna veličina modela je 4.11 megabajta s ukupnim brojem parametara od 1076481, od kojih 1072705 čine parametri korišteni u procesu treniranja. *Model_7* ima znatno veći broj parametara zbog veće ulazne dimenzije slike (256x256), što rezultira većim izlaznim tenzorima iz svakog sloja i samim time povećanim brojem parametara u cijeloj mreži. *Model_5* i *Model_6* koriste manje ulazne slike (128x128), što smanjuje dimenzije tenzora kroz mrežu i rezultira manjim brojem parametara, unatoč dodatnom sloju u *Modelu_6*.Slikom 3.30 prikazana je promijena točnosti i gubitka u svakoj od 50 epoha treniranja *Modela_7*, dok se na slici 3.31. prikazuju ulazne slike sa stvarnim maskama te predviđanjima modela *Model_7* za nekoliko odabranih primjera iz validacijskog skupa podataka.



Sl. 3.30. *Prikaz točnosti i gubitka prilikom treniranja modela Model_7 tijekom 50 epoha*

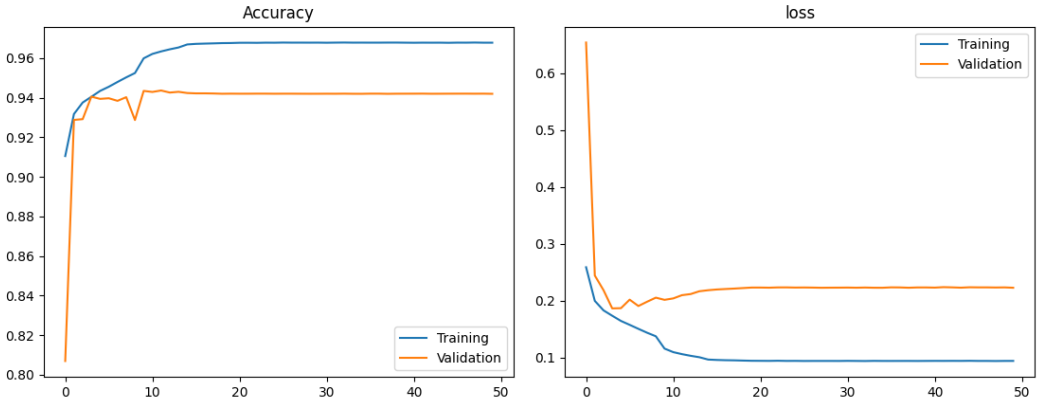


Sl. 3.31. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_7*

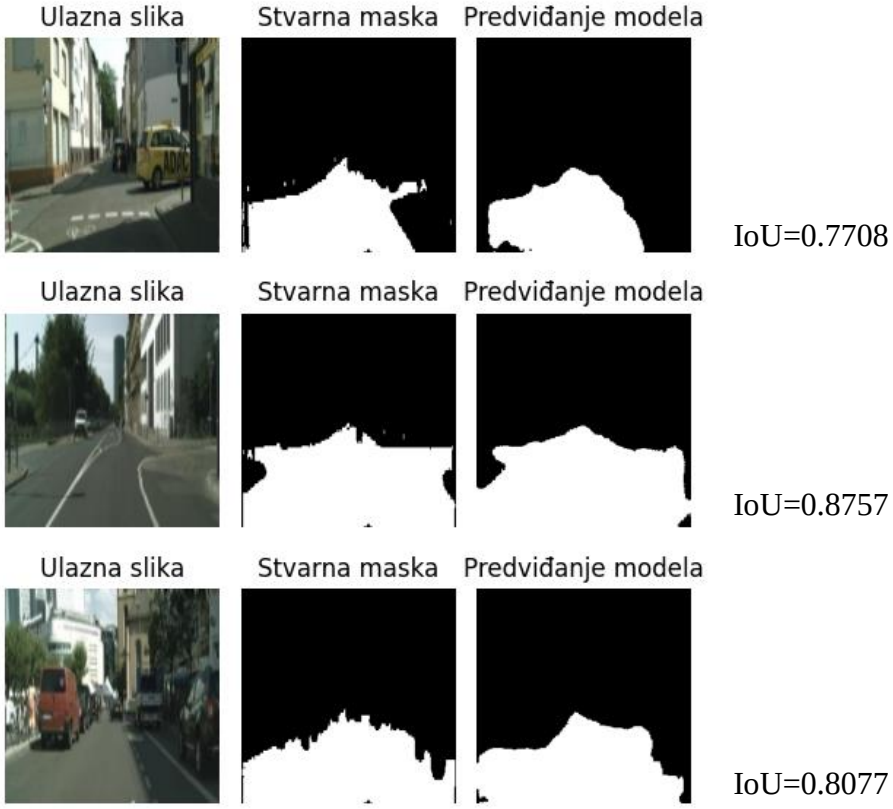
Detaljniji rezultati o performansama modela *Model_7* nalaze se u tablici 3.5.

Model_8 je nadogradnja *Modela_7* u smislu povećanja složenosti dodavanjem još jednog konvolucijskog sloja sa 64 filtra veličine 2x2 prije izlaznog sloja. *Model_8* koristi iste ulazne dimenzije slike kao i *Model_7* od 256x256 piksela. Takva izmjena povećava ukupnu veličinu modela na 4.14 megabajta i broj parametara na 1085121, s 1081217 parametara dostupnih za

treniranje. Dodatni sloj omogućava bolje izdvajanje složenih značajki i poboljšava potencijal modela za precizniju semantičku segmentaciju. Slikom 3.32 prikazana je promijena točnosti i gubitka u svakoj od 50 epoha treniranja *Modela_8*, dok se na slici 3.33. prikazuju ulazne slike sa stvarnim maskama te predviđanjima modela *Model_8* za nekoliko odabranih primjera iz validacijskog skupa podataka.



Sl. 3.32. *Prikaz točnosti i gubitka prilikom treniranja modela Model_8 tijekom 50 epoha*



Sl. 3.33. *Primjeri izlaznih maski za zadane ulazne slike iz validacijskog skupa Cityscapes skupa podataka za najbolji dobiveni model prilikom treniranja Modela_8*

Detaljniji rezultati o performansama modela *Model_8* nalaze se u tablici 3.5.

Predstavljeni modeli omogućuju dublje razumijevanje kako promjene u veličini filtera, broju filtera i dodavanje konvolucijskih slojeva mogu utjecati na performanse i složenost modela u zadatku semantičke segmentacije. Takve prilagodbe omogućuju preciznu kontrolu nad kapacitetom modela za učenje, kao i njegovom sposobnošću za generalizacijom na temelju različitih ulaznih podataka. Osim toga, omogućuju balansiranje između točnosti i učinkovitosti izvršavanja, što je ključno za postizanje optimalnih rezultata u praktičnim primjenama. Integracija ovih prilagodbi u proces dizajniranja modela za semantičku segmentaciju omogućuje stvaranje fleksibilnih i efikasnih arhitektura koje su prilagođene specifičnim zahtjevima.

Sve opisane verzije modela grupirane se u dvije skupine s obzirom na veličinu ulazne slike (128x128 ili 256x256), kako bi se odgovarajuće verzije mogle međusobno uspoređivati. Grupaciju te rezultate evaluacije moguće je iščitati iz tablica 3.5. i 3.6. Iz tablice 3.5. vidljivo je kako je originalni model prepoznao najviše TN piksela te ujedno ima i najmanje FP detektiranih piksela (metrika matrice zabune). *Model_8* uspio je najbolje prepoznati TP piksele (metrika matrice zabune), a *Model_3* ima najnižu stopu FN detektiranih piksela (metrika matrice zabune). Svi modeli iz tablice 3.5. pokazali su približno jednaku učinkovitost s F1 ocjenom u rasponu od 0.9072 do 0.9159, dok se *Model_3* pokazao najbržim, uz vrijeme potrebno za obradu jedne ulazne slike od 0.04618 sekundi.

Tablica 3.6. pruža uvid u rezultate modela kod kojih je dimenzija ulazne slike 128x128. Kod takvih verzija modela *Model_1* identificirao je najveći broj TP piksela, dok istovremeno ima najmanji broj FP piksela (metrika metrice zabune). *Model_2* se istaknuo po najboljem prepoznavanju TN piksela, dok *Model_6* pokazuje najnižu stopu FN piksela (metrika matrice zabune). Također, svi modeli iz tablice 3.6. pokazali su približno jednak učinak glede F1 ocjene, u rasponu od 0.8993 do 0.9081, dok je *Model_5* pokazao najbrže vrijeme obrade jedne ulazne slike od 0.04256 sekundi. Svi modeli trenirani su na istom trening skupu Cityscapes skupa podataka koji nakon augmentacije broji 8925 slika, a validirani su na skupu za validaciju od 500 slika.

Uzimajući u obzir F1 ocjenu, vrijeme inferencije te veličinu modela, *Model_5* se pokazao kao najpogodniji za implementaciju na ugradbenu računalnu platformu koja će procjenjivati vozno područje iz video signala i prikazivati ga na korisničkom ekranu. *Model_5* ima veličinu od samo 1.01 megabajta i najniže vrijeme inferencije (0.04256 sekundi). Iako je prema F1 ocjeni slabiji od *Modela_8*, koji je najbolji po toj mjeri za 0.01014, njegove druge karakteristike ga čine

praktičnijim za ovu primjenu. Iako *Model_5* ima najniže vrijeme inferencije među razmatranim modelima, dodatno testiranje i optimizacija su potrebni kako bi se osiguralo da može raditi u stvarnom vremenu.

Tab. 3.5. *Prikaz rezultata evaluacije originalnog modela i modela čija je ulazna slika dimenzija 256x256 piksela.*

Model	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost	Vrijeme inferencije [s]	Broj parametara modela	Veličina modela u memoriji (megabajti)
Original [29]	[23847003 1036160 435315 7449522]	0.87789	0.94479	0.91011	0.91011	0.83505	0.95509	0.05411	2058401	7.85
Model_3	[24000075 883088 510630 7374207]	0.89305	0.93529	0.91366	0.91366	0.84104	0.95747	0.04618	506513	1.93
Model_4	[23984821 898342 465077 7419760]	0.89200	0.94102	0.91585	0.91585	0.84477	0.95839	0.06416	525553	2.00
Model_7	[23948504 934659 563040 7321797]	0.88679	0.92859	0.90721	0.90721	0.83018	0.95429	0.04763	1076481	4.11
Model_8	[24094956 788207 643810 7241027]	0.90183	0.91835	0.91002	0.9100	0.83489	0.95629	0.06289	1085121	4.14

Tab. 3.6. *Prikaz rezultata evaluacije modela čija je ulazna slika dimenzija 128x128 piksela.*

Model	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost	Vrijeme inferencije [s]	Broj parametara modela	Veličina modela u memoriji (megabajti)
Original [29]	[23847003 1036160 435315 7449522]	0.87789	0.94479	0.91011	0.91011	0.83505	0.95509	0.05411	2058401	7.85
Model_1	[6017930 180284 217602 1776184]	0.90785	0.89086	0.89928	0.89928	0.81699	0.95143	0.04436	509217	1.94
Model_2	[5937725 260489 119056 1874730]	0.8780	0.94029	0.908079	0.90801	0.83163	0.953669	0.05040	525553	2.00
Model_5	[5938161 260053 138187 1855599]	0.87708	0.93069	0.90309	0.90309	0.82331	0.95139	0.04256	266497	1.02
Model_6	[5962337 235877 149008 1844778]	0.88663	0.92526	0.90554	0.90554	0.82738	0.95302	0.04753	275137	1.05

3.4.CARLA simulator

U cilju procjene sposobnosti generalizacije i robusnosti razvijenog algoritma za semantičku segmentaciju provedena je evaluacija nad podacima snimljenim u različitim vremenskim uvjetima. Primarna je svrha ovog potpoglavlja prikazati rezultate treniranja i evaluacije algoritma na sintetičkim podacima kreiranim u CARLA simulatoru te stvarnim podacima iz Cityscapes skupa podataka. Korištenje podataka iz simulatora omogućuje generiranje velikih skupova podataka s precizno anotiranim segmentacijama. S druge strane, evaluacija na stvarnim podacima je ključna za procjenu funkcionalnosti algoritma u stvarnim uvjetima.

Proces treniranja na sintetički dobivenim podacima odvijao se u jednakim uvjetima kao i proces treniranja na podacima iz Cityscapes skupa podataka. Tijekom treniranja korišten je *Model_5* te je dobiven *semantic_model_cs1.h5* model. Treniranje je izvršeno tijekom 50 epoha, uz prilagođenu funkciju gubitaka *weighted_binary_crossentropy* i veličinu grupe od 24. Nakon uspješnog završetka procesa treniranja, dobiveni algoritam prvotno je evaluiran na testnom skupu prikupljenom iz CARLA simulatora, a nakon toga i na testnom skupu Cityscapes skupa podataka kako bi se uvidjele performanse algoritma i na slikama snimljenim u stvarnome svijetu.

3.4.1. Stvaranje skupova podataka iz simulatora

Za dodatnu provjeru i obradu modela *semantic_model_cs1.h5* unutar CARLA simulatora kreirano je 10000 slika s odgovarajućim binarnim maskama za trening skup te dodatnih 1000 slika, također s odgovarajućim maskama, za testni skup podataka. Skupovi se sastoje od slika dobivenih s RGB kamerom i semantičkom kamerom, koje su montirane na prednjoj strani vozila. Implementirano je dohvaćanje slike s prednje kamere te spremanje u liste. Korištenje spomenutih kamera omogućava snimanje visokokvalitetnih slika i semantički označenih slika koje se kasnije koriste za obuku modela. U rada s CARLA simulatorom korištena je skripta pod nazivom *collect_data.py* koja započinje postavljanjem veze s CARLA simulatorom te postavljanjem početnih parametara, uključujući broj vozila i pješaka te različite opcije kamere. Skripta uključuje funkcije za dodavanje različitih senzora na vozila. Za segmentacijsku kameru, koristi se *sensor.camera.semantic_segmentation blueprint* koji generira semantički označene slike. Važna funkcija u skripti je *process_image3* (slika 3.34), koja se koristi za snimanje slika s RGB kamere i segmentacijske kamere te njihovo spremanje na disk. Ova funkcija također pohranjuje podatke o kontrolama vozila (npr. gas, kočenje, upravljanje) u CSV datoteku za kasniju analizu. Funkcija *save_image* koristi *OpenCV* za spremanje snimljenih slika u odgovarajuće direktorije.

Svaka snimljena slika ima jedinstveno ime koje uključuje broj okvira kako bi se olakšala organizacija i pretraga. Generirani skupovi podataka koriste se za obuku te testiranje modela za semantičku segmentaciju.

```
def process_image3(rgb_image_105, data, semantic_image):
    global frame_count
    save_path_105 = f"/home/tech/CARLA/PythonAPI/carla/train_images/rgb_mask_frames/frame_{frame_count}.jpg"
    save_path_semantic = f"/home/tech/CARLA/PythonAPI/carla/train_images/semantic/frame_{frame_count}.png"

    data1 = data
    data1.append(save_path_105)
    data1.append(save_path_semantic)

    file_name = "Data_cameras_t3.csv"
    folder_path = "/home/tech/CARLA/PythonAPI/carla/train_images"

    file_path = os.path.join(folder_path, file_name)
    with open(file_path, 'a', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(data1)

    save_image(rgb_image_105, save_path_105)
    save_image(semantic_image, save_path_semantic)

    frame_count += 1
```

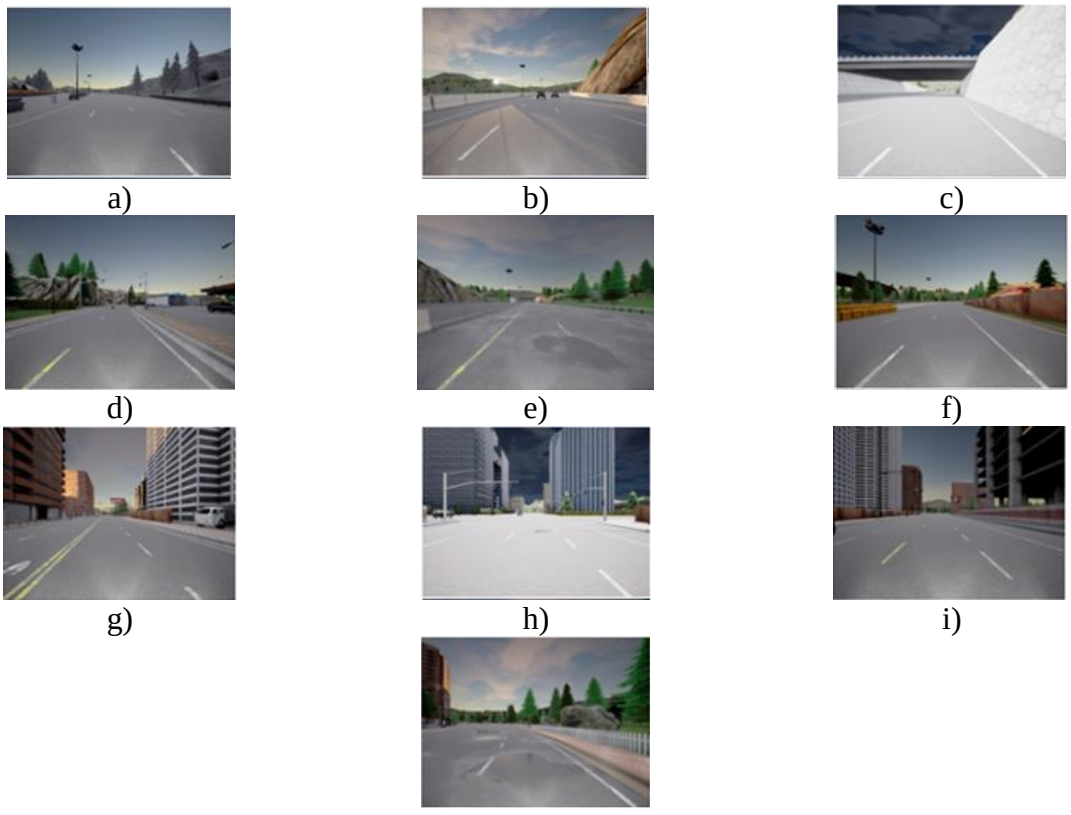
Sl. 3.34. *Prikaz funkcije process_image3 koja se koristi za snimanje slika s RGB kamere i segmentacijske kamere te njihovo spremanje na disk*

Podaci testnog skupa prikupljeni su unutar grada *Town01*, dok su podaci skupa podatka za treniranje prikupljeni iz gradova *Town04* i *Town05*. Svi podaci, uključujući i test i trening podatke, prikupljeni su pri različitim vremenskim uvjetima podešavajući parametre oblačnosti, padalina, naslaga padalina na kolniku, intenziteta vjetra, azimutnog kuta Sunca te kuta visine Sunca. Tablica 3.7 prikazuju raspone promjena parametara vremenskih uvjeta tijekom procesa prikupljanja podataka iz simulatora, a slika 3.35. prikazuje deset primjera slika iz testnog skupa podataka kreiranog CARLA simulatorom. Svaka od spomenutih deset slika (a) do (j) predstavlja sliku kreiranu uz različite parametre koji su opisani tablicom 3.7.

Slike prikupljene semantičkom kamerom obrađene su na način da označavaju samo voznu traku automobila, a ne više objekata (kolnik, oznake na kolniku, pješake, vozila, vegetaciju…). Navedeno je postignuto korištenjem *Canny* operatora za detekciju rubova te *Hughove* transormacije za pronalazak linija. Pronađene linije su zatim sortirane po udaljenosti od vertikalne simetrale slike kako bi se odredile dvije linije koje su najbliže spomenutoj simetrali (jedna lijevo od i jedna desno od nje). Po odabranim linijama povučeni su pravci te je pronađeno sjecište istih. Na kraju, određen je poligon definiran dvama pravcima i točkom sjecišta koji predstavlja voznu traku vozila (slika 3.36). Dok su ostale slike zadržale svoje izvorne dimenzije, slika (d) je skalirana kako bi se prilagodila specifičnim zahtjevima prikaza, odnosno daljnjoj obradi unutar sustava, što može rezultirati različitim proporcijama u prikazu.

Tab. 3.7. *Prikaz promjena parametara vremenskih uvjeta tijekom prikupljanja podataka.*

Trening skup	Test skup	Parametri
1-2000	1-100	Vedro nebo, bez oblaka, padalina i vjetra. Sunce na sjevoroistoku nisko iznad horizonta.
	101-200	Niska oblačnost, bez padalina i vjetra. Sunce na sjevoroistoku nisko iznad horizonta.
2001-4000	201-300	Srednja oblačnost, bez trenutnih padalina, umjerenih naslaga padalina na kolniku, blagi vjetar. Sunce na sjevoroistoku nisko iznad horizonta.
	301-400	Srednja oblačnost, bez padalina uz blagi vjetar. Sunce na sjeveroistoku visoko na nebu.
4001-6000	401-500	Srednja do visoka oblačnost, bez padalina uz umjeren vjetar. Sunce na sjeveroistoku, visoko na nebu.
	501-600	Niska oblačnost, bez padalina uz umjeren vjetar. Sunce na zapadu nisko iznad horizonta.
6001-8000	601-700	Niska oblačnost, umjerene padaline s umjerenim naslagama na kolniku. pojačan vjetar, Sunce na sjeveroistoku vrlo nisko iznad horizonta.
	701-800	Niska oblačnost, umjerene padaline s umjerenim naslagama na kolniku. pojačan vjetar, Sunce na sjeveroistoku vrlo nisko iznad horizonta.
8001-10000	801-900	Niska oblačnost, intenzivne padaline s naslagama na kolniku. pojačan vjetar, Sunce na sjeveroistoku vrlo nisko iznad horizonta.
	901-1000	Niska oblačnost, intenzivne padaline s naslagama na kolniku. pojačan vjetar, Sunce na sjeveroistoku vrlo nisko iznad horizonta.



Sl. 3.35. *Primjeri kreiranih podataka uz različite parametre iz CARLA simulatora prema tablici 3.7 za testni skup: (a) 1-100, (b) 101-200, (c) 201-300, (d) 301-400, (e) 401-500, (f) 501-600, (g) 601-700, (h) 701-800, (i) 801-900, (j) 901-1000*

a) b) c) d)

Sl. 3.36. *a) Slika prikupljena segmentacijskom kamerom, b) binarna slika, c) oznaka vozne trake, d) overlay slike pod b i pod c.*

3.4.2. Ispitivanje performansi modela *semantic_model_cs1.h5* u CARLA simulatoru

Model *semantic_model_cs1.h5* (treniran samo na trening skupu Cityscapes skupa podataka) testiran je na testnom skupu podataka prikupljenom iz CARLA simulatora te su dobiveni rezultati evaluacije uspoređeni s rezultatima evaluacije algoritma na validacijskom skupu Cityscapes skupa podataka. Iz tablice 3.8 moguće je iščitati kako je preciznost nešto viša kod testiranja algoritma na sintetičkim podacima iz CARLA simulatora, što ukazuje na to kako algoritam ima manji broj FP uzoraka kod CARLA skupa podataka. Odziv je značajno viši na Cityscapes podacima, što znači da algoritam bolje prepoznaje pozitivne uzorke u Cityscapes skupu. Kada je u pitanju F1 ocjena, koja predstavlja harmonijsku sredinu preciznosti i odziva, viša je na Cityscapes podacima, ukazujući na bolju ukupnu performansu algoritma na ovom skupu. *Dice* score prati F1 ocjenu i također pokazuje bolje rezultate na Cityscapes skupu podatka. IoU je viši za Cityscapes skup, što rezultira boljim prostornim preklapanjem predikcija sa stvarnim oznakama. Također, na Cityscapes skupu podatka točnost je značajno viša nego na CARLA testnom skupu, sugerirajući bolju prilagodbu modela za rad sa stvarnim slikama iz ovog skupa podataka. Rezultati pokazuju kako model daje bolje performanse na validacijskom skupu Cityscapes podataka u odnosu na testne slike prikupljene iz CARLA simulatora. Dobiveni rezultati mogu biti posljedica razlika u kompleksnosti i karakteristikama slika između dvaju skupova podataka. Dok je preciznost nešto viša na podacima kreiranim CARLA simulatorom, sve ostale metrike (odziv, F1 ocjena, *Dice* score, IoU i točnost) pokazuju bolje rezultate na Cityscapes skupu podataka.

Nakon dobivanja rezultata evaluacije modela *semantic_model_cs1.h5* na testnom skupu od 1000 slika prikupljenih iz CARLA simulatora, model je podlegnut procesu dotreniravanja

korištenjem ranije opisanog trening skupa podataka (dio 3.4.1.) prikupljenog iz CARLA simulatora.

Tab. 3.8. *Rezultati evaluacije semantic_model_cs1.h5 modela treniranog na Cityscapes trening skupu podataka*

Skup podataka	Model	Broj slika za validaciju/ testiranje	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost
Cityscapes	*semantic_model_cs1.h5*	500	[5938161 260053 138187 1855599]	0.87708	0.93069	0.90309	0.90309	0.82331	0.95139
CARLA simulator	*semantic_model_cs1.h5*	1000	[8931252 882198 1009045 5561505]	0.88457	0.84643	0.85468	0.85468	0.74624	0.88457

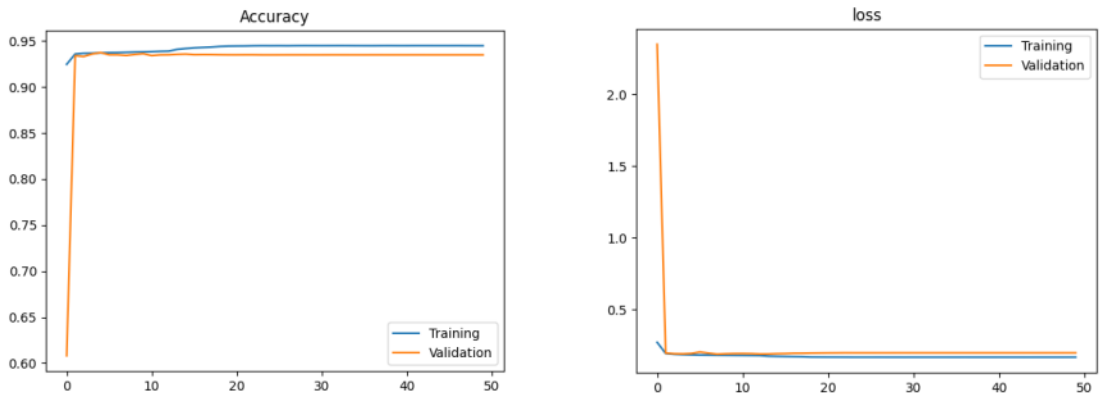
Performanse dotreniranog modela provjerene su ponovljenim procesom evaluacije na testnom skupu CARLA simulatora te validacijskom skupu Cityscapes skupa podataka. Tablica 3.9 pokazuje kako je preciznost značajno viša nakon dodatnog treniranja na CARLA trening skupu podataka, što rezultira manjim brojem lažno pozitivnih (FP) predikcija. Odziv je također značajno viši nakon dotreniranja, što znači da model bolje prepoznaje pozitivne uzorke u CARLA testnom skupu podataka. F1 ocjena je porasla za približno 10% , ukazujući na bolje ukupne performanse modela. *Dice* score prati F1 ocjenu i pokazuje bolje rezultate nakon dotreniranja na CARLA skupu podataka. IoU i točnost također su porasli, ukazujući na bolje prostorno preklapanje predikcija sa stvarnim oznakama te bolju prilagođenost modela. Rezultati pokazuju značajno poboljšanje performansi modela nakon dotreniranja na CARLA skupu podataka. Model treniran samo na Cityscapes podacima pokazao je dobre rezultate, ali dotreniranjem na CARLA podacima značajno su se poboljšale sve evaluacijske metrike. Preciznost, odziv, F1 ocjena, *Dice* score, IoU i točnost su svi viši nakon dotreniranja, ukazujući na bolju sposobnost modela za prepoznavanjem vozne trake u slikama iz CARLA simulatora. Usporedbom rezultata na Cityscapes skupu podataka prije i nakon dotreniravanja *semantic_model_cs1.h5* algoritma moguće je uočiti kako su sve evaluacijske metrike (preciznost, odziv, F1 ocjena, *Dice score*, IoU i točnost) značajno bolje prije dotreniravanja na CARLA skupu podataka. Preciznost je smanjena s 0.87708 na 0.85367, što ukazuje na povećan broj FP predikcija nakon dotreniravanja. Odziv se smanjio s 0.93069 na 0.68145, što znači da je model postao manje sposoban prepoznati pozitivne uzorke. F1 ocjena i *Dice score* te IoU i točnost, također su znatno niži nakon dotreniravanja. Ovi rezultati ukazuju na to da je dotreniranje na

CARLA podacima, iako korisno za poboljšanje performansi na simuliranim podacima, dovelo do pogoršanja performansi modela na stvarnim slikama iz Cityscapes skupa. Ovi rezultati sugeriraju da, iako dotreniranje na CARLA skupu značajno poboljšava performanse na sličnim simuliranim podacima, generalizacija modela na slikama iz stvarnog svijeta, poput onih iz Cityscapes skupa, može biti izazovnija.

Tab. 3.9. *Rezultati evaluacije dotreniranog semantic_model_cs1.h5 algoritma.*

Skup podataka	Model	Broj slika za testiranje	Matrica konfuzije	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost
CARLA simulator	*semantic_model_cs1.h5*	1000	[8931252 882198 1009045 5561505]	0.88457	0.84643	0.85468	0.85468	0.74624	0.88457
CARLA simulator	*semantic_model_cs1.h5* dotrenirani	1000	[9490164 323286 244394 6326156]	0.95138	0.96281	0.95706	0.95706	0.917654	0.96535
Cityscapes	*semantic_model_cs1.h5* dotrenirani	500	[9490164 323286 244394 6326156]	0.85367	0.68145	0.75791	0.75791	0.61018	0.89404

Proces obrade nastavljen je korištenjem jednake arhitekture modela kao i kod treniranja modela *semantic_model_cs1.h5*. Spomenuta arhitektura ponovno je podlegnuta procesu treniranja, ali ne na Cityscapes skupu podataka, već samo na prikupljenom trening skupu podataka iz CARLA simulatora (ne radi se o dotreniravanju modela, nego o ponovnom treniranju). Slika 3.37 prikazuje promjenu točnosti i gubitka tijekom procesa treniranja.



Sl. 3.37. *Prikaz točnosti i gubitka prilikom treniranja modela semantic_model_carla.h5 tijekom 50 epoha samo na trening skupu kreiranom u CARLA simulatoru*

Proces treniranja odvijao se u jednakim uvjetima kao i za dobivanje *semantic_model_cs1.h5* modela. Treniranje je izvršeno tijekom 50 epoha, uz prilagođenu funkciju gubitaka: *weighted_binary_crossentropy* i veličinu grupe od 24. Nakon uspješnog završetka procesa treniranja, dobiveni model prvotno je evaluiran na testnom skupu prikupljenom iz CARLA simulatora, a nakon toga i na validacijskom skupu Cityscapes skupa podatka kako bi se uvidjele performanse modela na slikama snimljenim u stvarnome svijetu (Tablica 3.10). Očekivano, model je imao bolje performanse na podacima prikupljenim iz simulatora negoli na stvarnim podacima. Najveće razlike vidljive su kod mjerenja preciznosti te IoU. Model evaluiran na stvarnim podacima pokazuje 12% manju preciznost te za čak 17% slabiji IoU.

Model treniran samo na skupu podataka CARLA simulatora pokazuje veću preciznost na testnom skupu iz CARLA simulatora (0.83420) nego na validacijskom skupu Cityscapes podataka (0.70723). S druge strane, model treniran na Cityscapes trening skupu i dotreniran na trening skupu iz CARLA simulatora pokazuje značajno veću preciznost na CARLA podacima (0.95138) u odnosu na osnovni model treniran samo na CARLA skupu (0.88457). Što se tiče odziva, model treniran samo na trening skupu iz CARLA simulatora pokazuje vrlo visok odziv na testnom skupu CARLA simulatora (0.97439), ali manji odziv na validacijskom skupu Cityscapes podataka (0.87969). Dotrenirani model pokazuje poboljšani odziv na CARLA podacima (0.96281) u odnosu na osnovni model semantic_model_cs1.h5 (0.84643). F1 ocjena također je viša za model treniran samo na na trening skupu iz CARLA simulatora na CARLA podacima (0.89887) nego na Cityscapes podacima (0.78408). Međutim, dotrenirani model ima višu F1 ocjenu na CARLA podacima (0.95706) u odnosu na osnovni model (0.85468). *Dice* koeficijent prati sličan obrazac, gdje model treniran samo na CARLA skupu ima viši *Dice* koeficijent na CARLA podacima (0.898865) nego na Cityscapes podacima (0.78408), dok dotrenirani model ima viši *Dice* koeficijent na CARLA podacima (0.95706) u odnosu na osnovni model (0.85468). IoU je također viši za model treniran samo na CARLA skupu na CARLA podacima (0.81631) nego na Cityscapes podacima (0.64486). Dotrenirani model ima značajno viši IoU na CARLA podacima (0.917654) u odnosu na osnovni model (0.74624). Na kraju, točnost je viša za model treniran samo na CARLA skupu na CARLA podacima (0.91207) nego na Cityscapes podacima (0.88209). Dotrenirani model ima višu točnost na CARLA podacima (0.96535) u odnosu na osnovni model (0.88457).

Iz rezultata je jasno da modeli trenirani samo na trening skupu iz CARLA simulatora postižu bolje performanse na CARLA testnom skupu u odnosu na Cityscapes skup, što je očekivano zbog sličnosti između trening i testnih podataka. Međutim, model koji je treniran na Cityscapes

skupu i dotreniran na CARLA skupu pokazuje znatno bolje performanse na CARLA testnom skupu u gotovo svim metrikama u odnosu na model treniran samo na CARLA skupu. To ukazuje na važnost korištenja raznolikih skupova podataka i dodatnog treniranja kako bi se poboljšala generalizacija modela i njegovo prilagođavanje na različite scenarije. Ovaj dotrenirani model (*semantic_model_cs1.h5* dotrenirani) pokazuje se kao robustniji za zadatak semantičke segmentacije, s obzirom na njegovu visoku preciznost, odziv, F1 ocjenu, Dice koeficijent, IoU i točnost na CARLA skupu podataka.

Tab. 3.10. *Rezultati evaluacije semantic_model_carla.h5 algoritma.*

Skup podataka	Model	Broj slika za test/validaciju	Matrica konfuzije	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost
CARLA simulator	*semantic_model_carla.h5*	1000	$\begin{bmatrix} 8541014 & 1272436 \\ 168253 & 6402297 \end{bmatrix}$	0.83420	0.97439	0.89887	0.898865	0.81631	0.91207
Cityscapes	*semantic_model_carla.h5*	500	$\begin{bmatrix} 5472133 & 726081 \\ 239864 & 1753922 \end{bmatrix}$	0.70723	0.87969	0.78408	0.78408	0.64486	0.88209

4. IMPLEMENTACIJA ODABRANOG MODELA NA UGRADBENU RAČUNALNU PLATFORMU

U ovom poglavlju razmatra se proces implementacije odabranog modela dubokog učenja na ugradbenu računalnu platformu. Cilj je omogućiti modelu učinkovito funkcioniranje na uređajima s ograničenim resursima, poput pametnih telefona, mikrokontrolera i drugih ugradbenih sustava. Ovaj korak je ključan za primjenu razvijenog modela u stvarnim uvjetima, kao što je procjena voznog područja u stvarnom vremenu. Prvi korak u procesu implementacije je pretvorba *Keras* modela u *TensorFlow Lite* format, što je objašnjeno u odjeljku 4.1. *TensorFlow Lite* je verzija *TensorFlowa* optimizirana za rad na uređajima ograničenih resursa, omogućujući brzu i učinkovitu inferenciju modela. Detaljno su opisani postupci kvantizacije i konverzije modela, te korištenje hardverskih akceleratora za ubrzanje izvođenja. Odjeljak 4.2 bavi se evaluacijom dobivenog *TFLite* modela kako bi se provjerile njegova ispravnost i performanse nakon konverzije. Uspoređuju se rezultati s izvornim *Keras* modelom, s posebnim naglaskom na vrijeme inferencije, koje je značajno smanjeno korištenjem optimizacija specifičnih za mobilne uređaje. U odjeljku 4.3 opisana je izrada *Android* aplikacije koja koristi razvijeni model za segmentaciju voznog područja iz slike dobivene kamerom. Aplikacija je razvijena korištenjem *Android Studia* i programskog jezika *Kotlin,* a detaljno su opisani koraci potrebni za integraciju modela i rad s kamerom putem *Camera2* API-ja. Odjeljak 4.4 opisuje postupak postprocesiranja predviđanja modela kako bi se poboljšala vizualizacija rezultata unutar aplikacije. Korištenjem *OpenCV* biblioteke, obavlja se dodatna obrada izlaza modela, čime se osigurava precizniji prikaz segmentiranog voznog područja.

4.1. Pretvorba *Keras* modela u *Tensorflow Lite* format

TensorFlow Lite (*TFLite*) verzija *TensorFlowa* optimizirana je za korištenje modela na uređajima ograničenih resursa, poput pametnih mobitela, mikrokontrolera, *Raspberry Pi-a* i drugih. Omogućuje implementaciju strojnog učenja na uređajima koji ne sadrže dovoljno resursa za izvršavanje složenih modela u punom *TensorFlow* obliku. Postupkom kvantizacije smanjuje se veličina modela s ciljem minimalnog gubitka točnosti. Kvantizacija je ključna za optimizaciju memorije i brzine izvođenja na uređajima s ograničenim resursima. *TensorFlow Lite* podržava različite platforme, od kojih su neke *Android, iOS, Raspberry Pi*. Također, *TensorFlow Lite* omogućuje korištenje hardverskih akceleratora poput GPU-a i DSP-a za ubrzanje izvođenja

modela na spomenutim platformama. Spomenuta verzija podržava CNN te rekurentne neuronske mreže (RNN) [47].

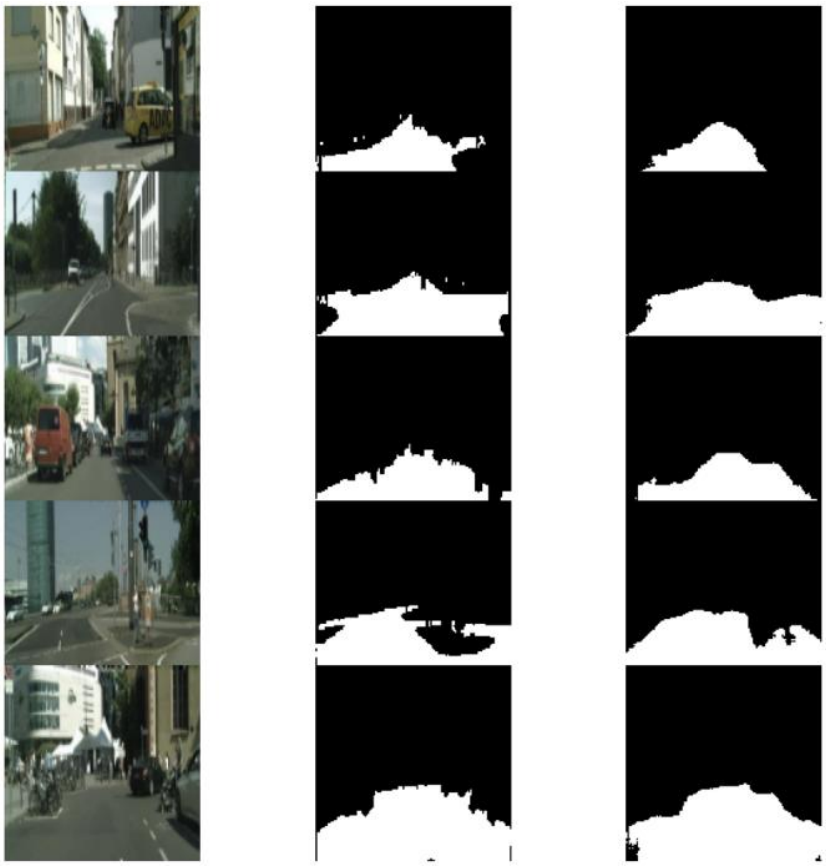
Konverzija *Keras* modela u *Tensorflow Lite* format započela je učitavanjem prethodno treniranog *Modela_5* (`*semantic_model_cs1.h5*`) pomoću `*tf.keras.models.load_model*` funkcije. Učitavanju modela prethodila je definicija prilagođene funkcije gubitaka za binarnu klasifikaciju (*weighted_binary_crossentropy*) koja pridružuje veće težine pozitivnim primjerima (klasa 1) korištenjem faktora 2.0, dok negativni primjeri (klasa 0) zadržavaju težinu 1.0 s ciljem postizanja ravnoteže među klasama. Radi uključenja spomenute prilagođene funkcije gubitaka, nakon učitanja modela korištena je *'tf.keras.utils.custom_object_scope'* funkcija. Idući je korak kreiranje pretvarača *TFLite* verzije iz učitanog *Keras* modela pomoću ugrađene funkcije `*tf.lite.TFLiteConverter.from_keras_model*`. `*Converter.allow_custom_ops*` postavljen je na *False,* što upućuje na to kako se ne dopuštaju prilagođene operacije tijekom konverzije modela, čime se osigurava minimalno smanjenje točnosti modela. Pozivom `*converter.convert()*` pretvarač se koristi za pretvaranje *Keras* modela u *TFLite* format. Konvertirani model zapisuje se u binarnu datoteku čija je ekstenzija `*.tflite*` koristeći `*with open(...)*` funkciju. Naposljetku, učitava se *TFLite* model koristeći `*tf.lite.Interpreter*`, a zatim se provjerava je li sve u redu alociranjem tenzora. Ukoliko je konverzija uspješno obavljena, unutar konzole ispisuje se poruka: *„INFO: Created TensorFlow Lite XNNPACK delegate for CPU.“* (Programski kod pretvorbe *Keras* modela u *Tensorflow Lite* model vidljiv je u prilogu P.4.4).

4.2.Evaluacija dobivenog *TFLite* formata modela

Nakon uspješno odrađene konverzije *Keras* modela u *TFLite format* odrađena je evaluacija dobivenog formata modela radi dodatne provjere ispravnosti procesa konverzije te prikaz ulazne slike sa stvarnim maskama te predviđanjima modela *'semantic_model_cs1.tflite'* za nekoliko odabranih primjera testiranih na istom validacijskom skupu na kojemu je testiran i sami *Kerasov* model *'semantic_model_cs1.h5'* (slika 4.1.).

Model je ispitan jednakim evaluacijskim metrikama kao i svi ranije opisani *Keras* modeli. Iz tablice 4.1. vidljivo je kako su rezultati evaluacije modela ostali nepromijenjeni. Vrijeme inferencije ostalo je približno na istoj razini kao i na CPU, odnosno oko 0,04 sekunde, što je izuzetno povoljno jer omogućava obradu od 25 slika u sekundi. *TensorFlow Lite* i slične biblioteke dizajnirane su za iskorištavanje hardverskih akceleratora dostupnih na mobilnim uređajima, što omogućuje znatno bržu obradu podataka u usporedbi s klasičnim PC procesorima. Mobilni su uređaji optimizirani za specifične zadatke uz pomoć hardverskih akceleratora koji

nisu dostupni ili nisu korišteni na PC-u prilikom inicijalne evaluacije. Stoga, optimizacija modela za mobilne uređaje može rezultirati značajnim povećanjem performansi u vidu brzine rada, zadržavajući pritom istu razinu točnosti i drugih evaluacijskih metrika.



Sl. 4.1. *Primjeri izlaznih maski za zadane ulazne slike iz Cityscapes validacijskog skupa podataka za dobiveni model prilikom treniranja semantic_model_cs1.tflite. S lijeva na desno: ulazna RGB slika, temeljna istina, predviđanje modela*

Tab. 4.1. *Prikaz rezultata evaluacije modela semantic_model_cs1.tflite i usporedba s formatom semantic_model_cs1.h5*

Model	Broj slika za validaciju	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost na validacijskom skupu podataka	Vrijeme inferencije [s]
semantic_model_cs1.h5	500	$\begin{bmatrix} 5938161 & 260053 \\ 138187 & 1855599 \end{bmatrix}$	0.87708	0.93069	0.90309	0.90309	0.82331	0.95139	0.04256
semantic_model_cs1.tflite	500	$\begin{bmatrix} 5938161 & 260053 \\ 138187 & 1855599 \end{bmatrix}$	0.87708	0.93069	0.90309	0.90309	0.82331	0.95139	0.04000

4.3.Izrada Android aplikacije

Aplikacija za segmentaciju voznog područja iz slike dobivene kamerom montiranom na prednjoj strani vozila razvijena je unutar ranije predstavljenog *Android Studia* korištenjem programskog jezika *Kotlin*. Ovaj programski jezik, koji se razvio kao alternativna verzija *Jave*, koristi se u svrhu razvoja Android, *web* aplikacija i aplikacija za radnu površinu. *Kotlin* je dizajniran s fokusom na poboljšanje jasnoće koda, čime je programerima omogućeno pisanje koda jednake funkcionalnosti u manje linija koda u odnosu na funkcionalnost napisanu unutar programskog jezika *Java*. *Kotlin* se također može koristiti i zajedno s *Javom*. Postojeće se *Java* biblioteke i kod mogu integrirati unutar *Kotlin* projekta. Riječ je o statički tipiziranom jeziku koji koristi *"type inference"* kako bi se smanjila potreba za eksplicitnim tipovima podataka. Ovaj je jezik službeno podržan od strane *Google-a* kao prvi izbor za razvoj *Android* aplikacija. Podržava funkcionalno programiranje, što uključuje lambda izraze, funkcije prvog reda te funkcije viših redova. *Kotlin* je programski jezik potpuno otvorenog koda [48].

Za rad unutar *Android studia* korišten je Androidov standardni API *(*engl. *Application Programming Interface)* za rad s kamerama koji je uveden u Android 5.0 (API razina 21) *Camera2* API. Spomenuti API predstavlja značajno unaprjeđenje u odnosu na prethodni *Camera* API. *Camera2* API omogućuje programski pristup različitim postavkama kamere kao što su ekspozicija i ISO, balans bijele boje, čime je razvojnim programerima omogućeno prilagoditi postavke kamere potrebama aplikacije u razvoju. *Camera2* API podržava istovremeni rad više kamera, omogućujući aplikacijama korištenje različitih kamera uređaja (prednje i stražnje kamere, višestruke leće na uređajima s više kamera). Podržava visoku brzinu snimanja okvira, što omogućuje snimanje videozapisa visoke brzine. Također, podržava i RAW slikovne formate, što omogućuje aplikacijama dobivanje sirovih slikovnih podataka izravno s kamere, bez potrebe za obradom [49].

Prvotni zadatak kod izrade aplikacije bio je omogućiti primanje slike s kamere. Kako bi se to omogućilo, unutar *Android* manifesta dodane su dvije linije (slika 4.2). Prva linija aplikaciji omogućuje zahtjev za pristup kameri uređaja od strane korisnika. U slučaju nedobivanja odobrenja, aplikacija neće imati mogućnost korištenja kamere. Ovo će odobrenje biti zatraženo prilikom instalacije aplikacije, osiguravajući transparentnost i kontrolu korisnika nad njihovim podacima. Druga linija jasno označava zahtjev aplikacije za hardverskom značajkom kamere na uređaju. Postavljanje atributa *'android:required="true"'* naglašava neophodnost kamere za ispravan rad aplikacije. U slučaju nedostupnosti kamere na uređaju ili ako je ista onemogućena,

korisnik neće moći instalirati aplikaciju, čime se osigurava rad aplikacije samo na uređajima koji zadovoljavaju navedene zahtjeve.

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera" android:required="true"/>
```

***Sl. 4.2**. Naredbe za omogućavanje korištenja kamere*

XML kod za opis izgleda sučelja aktivnosti *Android* aplikacije prikazan je slikom 4.3. Riječ je o *XML* datoteci verzije 1.0 koja koristi UTF-8 kodiranje. Korijenski element *layout*-a je *RelativeLayout*. Atributi *android:layout_width* i *android:layout_height* postavljeni su na veličinu *'match_parent'* što označava korištenje cijelog zaslona prilikom rada aplikacije. Prvi postavljeni element je *<TextureView>* koji se koristi za prikazivanje površina izvan *OpenGL-a*, odnosno, u ovome slučaju riječ je o korištenju *TextureView* u svrhu prikazivanja slike dobivene s kamerom uređaja. *<ImageView>* element koristi se za prikazivanje slika ili grafika na zaslonu uređaja. Tijekom rada aplikacije *<ImageView>* koristi se za prikaz obrađenih slika dobivenih s kamere uređaja. Atributom *android:id* dodjeljuju se jedinstveni identifikatori čime je omogućen programski pristup elementima iz *Kotlin* koda. Oba elementa zauzimaju cijeli prostor dostupan unutar roditeljskog *RelativeLayout-a.*

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextureView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/textureView"/>

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imageView"
        android:contentDescription="@+string/image_description"/>

</RelativeLayout>
```

***Sl. 4.3.** Xml kod izgleda korisničkog sučelja (layout-a) unutar Android aplikacije.*

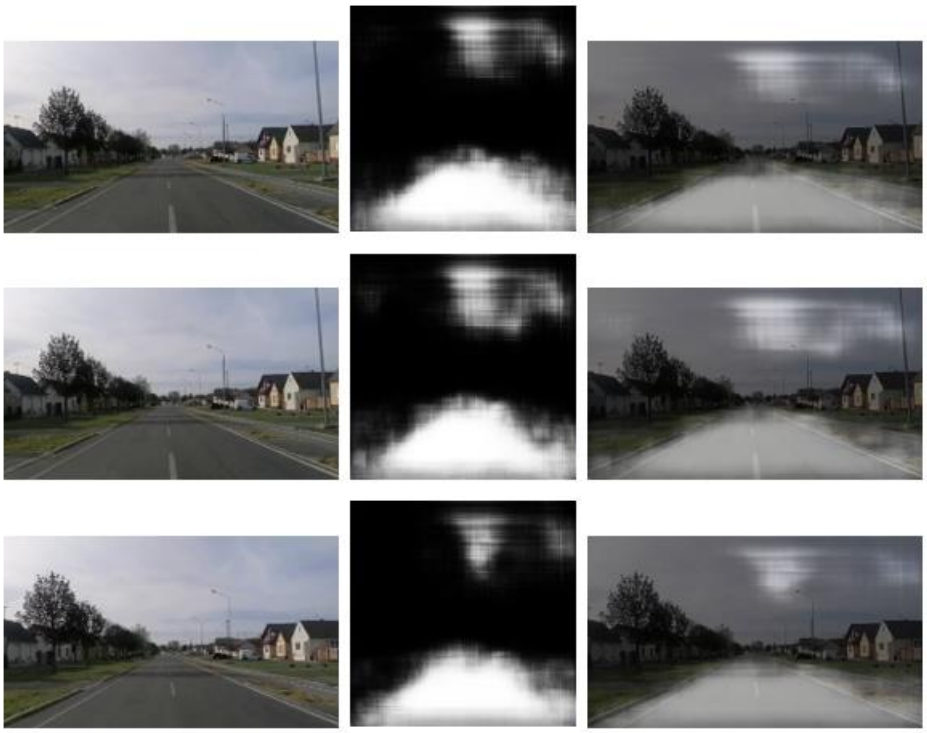
Unutar *mainActivity.kt* prilikom pokretanja aplikacije poziva se *onCreate()* funkcija koja unutar sebe poziva *getPermission()* funkciju. Funkcija *getPermission()* koristi se za provjeru dopuštenja pristupa kameri i, po potrebi, zahtijeva od korisnika odobrenje pristupa kameri. Funkcija koristi *ContextCompat.checkSelfPermission()* metodu kako bi provjerila je li aplikacija već dobila dozvolu za pristup kameri. Argumenti metode uključuju trenutni kontekst aktivnosti ('*this*') i naziv dozvole koja se provjerava ('*android.Manifest.permission.CAMERA*'). Također, provjerava se je li rezultat provjere dopuštenja različit od *'PackageManager.PERMISSION_GRANTED'* (znači da dopuštenje za korištenjem kamere nije već odobreno). Ukoliko prethodna provjera otkrije kako aplikacija nema dopuštenje za pristup kameri, koristi se *ActivityCompat.requestPermissions()* metoda kako bi se od korisnika zatražilo odobrenje pristupa kameri. Prvi argument je trenutna aktivnost ('*this*'), drugi argument je polje koje sadrži naziv dopuštenja koje se traži (u ovom slučaju, samo dopuštenje za pristup kameri), a treći argument je zahtjevni kod (101), koji se koristi za identifikaciju zahtjeva za dopuštenjem unutar *onRequestPermissionsResult()* metode.

Za praćenje promjena na *'TextureView'*-u (praćenje slike s kamere) koristi se *SurfaceTextureListener* čija se implementacija sastoji od četiriju funkcija. Prva od tih funkcija je *onSurfaceTextureAvailable(surface: SurfaceTexture, width: Int, height: Int)* koja se poziva prilikom stvaranja *'SurfaceTexture'*. U radu ove aplikacije ova funkcija poziva funkciju *CameraManager-a openCamera()* koja služi za otvaranje kamere, postavljanje prikaza kamere na '*TextureView*' te neprestano prikazivanje slike s kamere na zaslonu. Unutar funkcije *OpenCamera()* odabire se željena kamera s popisa dostupnih kamera pomoću argumenta *cameraIdList*. Za praćenje stanja otvaranja kamere koristi se anonimna klasa *CameraDevice.StateCallback(),* s trima funkcijama koje se pozivaju u različitim fazama procesa otvaranja kamere. Funkcija ' *override fun onOpened(camera: CameraDevice) { ... }* ' poziva se nakon što je kamera uspješno otvorena. Argument *cameraDevice* postavlja se na otvorenu kameru nakon čega se stvara *'Surface'* pomoću *'TextureView'* koji omogućuje prikaz slika s kamere. *session.setRepeatingRequest (captureRequest.build(), null, null)* postavlja ponavljajući zahtjev za prikaz preko kamere koristeći *setRepeatingRequest()* metodu sesije. U ovome se slučaju koristi prethodno stvoren *captureRequest*, koji omogućuje neprestano prikazivanje slika s kamere na odgovarajućem *Surface-u*. Argumenti *onSurfaceTextureAvailable* metode uključuju '*surface*', što je '*SurfaceTexture*' stvoreni objekt te '*width*' i '*height*', koji predstavljaju dimenzije prikazane površine (dimenzije zaslona). Druga se funkcija *onSurfaceTextureSizeChanged(surface: SurfaceTexture, width: Int, height: Int)* poziva prilikom

promjene veličine *'SurfaceTexture'*. Sljedeća funkcija: *onSurfaceTextureDestroyed(surface: SurfaceTexture): Boolean* poziva se prilikom uništenja *'SurfaceTexture'*. Unutar ove aplikacije implementacija spomenute funkcije vraća *'false'*, čime se osigurava opstanak *'TextureView'* nakon uništenja *`SurfaceTexture'*. Zadnja funkcija zadužena za implementaciju *SurfaceTextureListener* je *onSurfaceTextureUpdated(surface: SurfaceTexture)* (programski kod vidljiv u prilogu P.4.5.) koja se poziva prilikom ažuriranja *'SurfaceTexture'*. Ažuriranje se događa prilikom prikazivanja novog okvira s kamere unutar *'TextureView'*-a. Unutar ove funkcije provodi se obrada slike koja se trenutno prikazuje. Navedene funkcije omogućuju aplikaciji reagiranje na promjene vezane uz *`SurfaceTexture`* unutar *`TextureView`*-a, kao što su stvaranje, promjena veličine, uništavanje i ažuriranje te omogućuju izvršavanje odgovarajućih akcija prilikom spomenutih promjena.

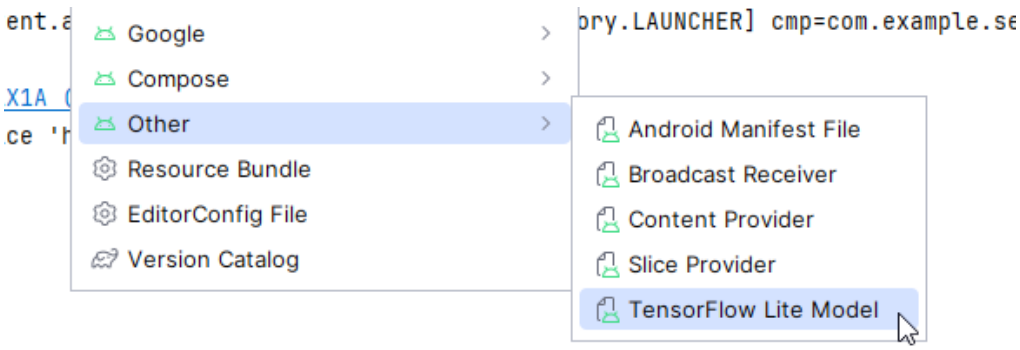
Nakon uspješno dobivenog prikaza slike s kamere obavljeno je prikupljanje podataka s kamere iz aplikacije u obliku video snimki kolnika. Video snimke su zatim razložene na okvire koji su poslužili kao testni skup za *Keras* i *Tensorflow Lite* verzije modela *semantic_model_cs1*. Na slikama 4.4 i 4.5. vidljivi su rezultati predviđanja obaju modela te je zaključeno kako su ulazne slike s kamere kompatibilne s ulaznim slikama modela. Pošto su na slici 4.5 ulazne slike i predviđanja modela različitih dimenzija, *overlay* slike se dobije skaliranjem obiju slika na jednake dimenzije te prikazivanju rezultantne slike trećeg stupca slike 4.5.

Sl. 4.4. *Prikaz rezulata predviđanja modela semantic_model_cs1.h5. Prvi stupac: ulazna slika, drugi stupac: predviđanje modela, treći stupac: overlay ulazne slike i predviđanja modela.*



Sl. 4.5. *Prikaz rezultata predviđanja modela semantic_model_cs1.tflite. Prvi stupac: ulazna slika, drugi stupac: predviđanje modela, treći stupac: overlay ulazne slike i predviđanja modela.*

Idući korak kod razvoja željene aplikacije je ubacivanje modela strojnog učenja unutar Android aplikacije. Odabrani model (*semantic_model_cs1.tflite*) uključen je unutar Android Studia desnim klikom unutar projektnog stabla te odabirom opcije *New -> Other -> TensorFlow Lite Model* (Slika 4.6.).



Sl. 4.6. *Prijenos semantic_model_cs1.tflite modela unutar Android aplikacije.*

Korištenje modela *semantic_model_cs1.tflite* unutar Android aplikacije omogućeno je unutar *onSurfaceTextureUpdate* funkcije (slika 4.7.). Navedena funkcija se aktivira svakim ažuriranjem površine unutar *'TextureView'*-a pristizanjem novih slika s kamere uređaja. Na početku funkcije

provjerava se je li trenutna slika dostupna na `*TextureView*`-*u*. Ukoliko je slika dostupna, pretvara se u `*Bitmap*` objekt za daljnju obradu. *Bitmap* objekt je objekt klase unutar Androida koja predstavlja sliku ili grafiku u obliku piksela. Objekt može sadržavati sliku u 2D formatu s informacijama o boji svakog piksela na slici. Navedena klasa omogućuje manipulaciju slikom (promjena veličine, rotacija, primjena filtera). *Bitmap* objekt sadrži podatke o slikovnom sadržaju koji se nalazi u memoriji. Sadržaji *Bitmap* objekta mogu biti slike u bilo kojem formatu podržanom od strane Androida (JPEG, PNG, BMP). Također, može se stvoriti prazan objekt koji se može koristiti za crtanje ili generiranje slike programski. *Bitmap* objekti se često koriste za prikazivanje slika na zaslonu (npr. u *ImageView*), manipulaciju slikama pri obradi ili analizi, te za pohranu slika u lokalnoj memoriji uređaja ili prijenos putem mreže.

```
override fun onSurfaceTextureUpdated(surface: SurfaceTexture) {
    textureView.bitmap?.let { currentFrameBitmap ->
        val scaledBitmap = Bitmap.createScaledBitmap(currentFrameBitmap, dstWidth: 128, dstHeight: 128, filter: true)
        currentFrameBitmap.recycle()

        val tensorImage = TensorImage(DataType.FLOAT32)
        tensorImage.load(scaledBitmap)
        val processedImage = imageProcessor.process(tensorImage)

        val inputFeature0 = TensorBuffer.createFixedSize(intArrayOf(1, 128, 128, 3), DataType.FLOAT32)
        inputFeature0.loadBuffer(processedImage.buffer)
        val outputs = model.process(inputFeature0)
        val outputFeature0 = outputs.outputFeature0AsTensorBuffer
        Log.d( tag: "ModelOutput", msg: "Output values: ${outputFeature0.floatArray.joinToString( separator: ", ")}")
        val maskBitmap = Bitmap.createBitmap( width: 128, height: 128, Bitmap.Config.ARGB_8888)
        val pixels = IntArray( size: 128 * 128)
        for (i in outputFeature0.floatArray.indices) {
            val prediction = outputFeature0.floatArray[i]
            if (prediction > 0.5) {
                pixels[i] = Color.argb( alpha: 128, red: 0, green: 255, blue: 0) // Segmentirana područja
            } else {
                pixels[i] = Color.TRANSPARENT // Ostala područja
            }
        }
        maskBitmap.setPixels(pixels, offset: 0, stride: 128, x: 0, y: 0, width: 128, height: 128)
        val scaledMaskBitmap = Bitmap.createScaledBitmap(maskBitmap, textureView.width, textureView.height, filter: true)
        runOnUiThread {
            imageView.setImageBitmap(scaledMaskBitmap)
        }
    }
}
```

Sl. 4.7. *Kod onSurfaceTextureUpdate funkcije.*

Nakon pretvorbe u *Bitmap* objekt, odvija se skaliranje trenutne slike na dimenzije 128x128 piksela jer su tih dimenzija ulazne slike u *semantic_model_cs1.tflite* model. Nakon skaliranja, slika se učitava u *TensorImage* objekt, koji je potreban za daljnju obradu s *TensorFlow Lite* modelom. *TensorImage* objekt je dio *TensorFlow Lite* biblioteke koji služi za rad sa slikama u

kontekstu strojnog učenja na ugradbenim računalnim sustavima. Omogućuje učitavanje slika iz različitih izvora, kao što su *Bitmap* objekti ili izvori slika u memoriji. Također, omogućuje obradu i pripremu slika za ulaz u *TensorFlow Lite* modele. *TensorImage* olakšava proces pretvaranje slike u tenzor, koji je osnovna struktura podataka u *TensorFlow Lite* modelima. Tenzor predstavlja višedimenzionalni niz vrijednosti.

Iza učitavanja *Bitmap* objekta u *TensorImage*, slika se obrađuje pomoću *ImageProcessor* objekta. `*ImageProcessor.Builder()*` stvara novi *Builder* objekt za kreiranje *ImageProcessor* objekta. U procesu obrade slike dodana je operacija za promjenu veličine objekta na dimenzije 128x128 piksela pomoću *'.add(ResizeOp(128,128,ResizeOp.ResizeMethod.BILINEAR))'*. *BILINEAR* metoda koristi bilinearnu interpolaciju piksela kako bi se stvorila glatka slika. Na kraju se dodaje i operacija normalizacije slike. Normalizacija pomaže u održavanju vrijednosti piksela unutar određenog raspona, što poboljšava performanse modela. *NormalizeOp* objekt postavlja raspon piksela na vrijednosti od 0 do 255 (za RGB slike).

Unutar koda dodana je i provjera spremnosti spremnika (engl. *buffer*) za čitanje te se ovisno o rezultatu provjere ispisuje poruka u *LogCat* konzoli unutar Android Studio okruženja. Nakon odrađene provjere stvara se *TensorBuffer* objekt koji će sadržavati ulazne podatke za model. Definiraju se dimenzije tenzora koje odgovaraju dimenzijama slika očekivanih za ulaz u model (1x128x128x3). Prvi broj označava koliko se slika istovremeno obrađuje, dok ostala tri predstavljaju dimenziju slike. Sve slike su *float32* tipa. Zatim se obrađena slika učitava u prethodno stvoreni *TensorBuffer* objekt. Slijedi izvršavanje *TensorFlow Lite* modela na pripremljenim ulaznim podacima dobivenim s kamere uređaja. Rezultat izvođenja modela pohranjuje se u odgovarajući objekt. Izlazni podaci modela pretvaraju se u *TensorBuffer* radi lakšeg daljnjeg rukovanja rezultatima. Nakon pretvorbe izlaznih podataka, podaci se ispisuju unutar *LogCat* konzole radi boljeg praćenja rezultate modela tijekom izvođenja aplikacije (slika 4.8.). Prvi redak slike prikazuje vrijednost *Byte buffera*, dok drugi redak prikazuje izlazne vrijednosti.

```
Byte buffer values: 0, 0, 0, 0, 0, 0, -128, 63, 0, 0, -128, 63, 0, 0,
Output values: 0.026737323, 0.020253845, 0.019298749, 0.0150599405, 0.
```

Sl. 4.8. *Primjer ispisa vrijednosti byte buffera i izlaznih vrijednosti unutar konzole.*

Nakon ispisa izlaznih podataka obavlja se pretvaranje izlaznih podataka modela u sliku koja se može prikazati na korisničkom sučelju (unutar *ImageView*). Stvara se nova *Bitmapa*

odgovarajućih dimenzija (128x128x3) i postavljaju se boje piksela ovisno o predikcijama modela. Područja slike koja predstavljaju kolnik i koje je model prepoznao označena su zelenom bojom s prozirnošću postavljenom na 128, dok su ostala područja izlazne slike transparentna. Pikseli se postavljaju unutar *Bitmape* te se stvara nova *Bitmapa* koja je prilagođena dimenzijama korisničkog sučelja. Dretva (engl. *thread*) je osnovna jedinica izvršavanja u računalnom programu koja omogućuje paralelno izvršavanje zadataka. Postavljanjem ažuriranja sučelja na glavnu dretvu osigurava se da se promjene na korisničkom sučelju odvijaju u konzistentnom i kontroliranom okruženju, čime se izbjegavaju problemi s više dretvi kao što su sudari ili nepredvidivo ponašanje aplikacije. Unutar aplikacije dodan je i kod za mjerenje vremena potrebnog za obradu jednog okvira te kod za broj okvira koji se obradi unutar jedne sekunde (FPS). Vrijeme potrebno za obradu jednog okvira iznosi u prosjeku 0.1686 sekundi, a FPS: 5.9312.

Aplikacija je testirana u stvarnim uvjetima vožnje pri brzinama od 20 km/h, 30 km/h te 50 km/h. Rezultati prikaza predviđanja modela unutar aplikacije vidljivi su na slici 4.9.

a) b) c)

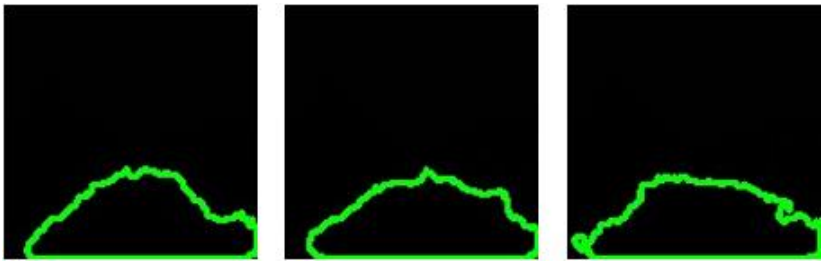
Sl. 4.9. *Prikaz predviđanja modela unutar aplikacije u stvarnim uvjetima a) pri brzini od 20km/h, b) pri brzini od 30km/h, c) pri brzini od 50km/h*

4.4. Postprocesiranje predviđanja modela

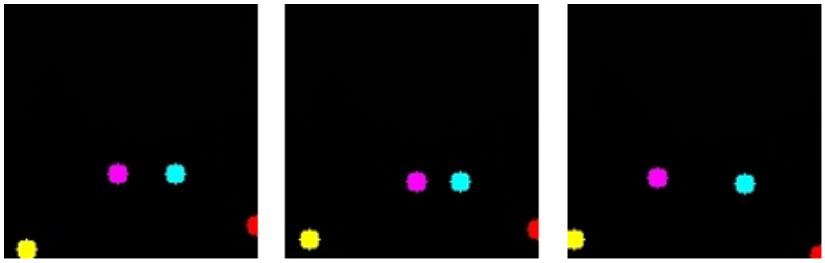
U svrhu poboljšanja prikaza predviđanja modela unutar aplikacije na izlaz modela primijenjen je dodatni postupak obrade. Postupak obrade izlaza modela prvotno je testiran izvan aplikacije korištenjem *OpenCV* biblioteke. Na izlaz modela (sl. 4.10.) primijenjena je funkcija *cv2.findContours()* čija je uloga prikazivanje svih kontura objekata maske. Parametrom *cv2.RETR_EXTERNAL* osigurava se pronalaženje samo vanjskih kontura, dok parametar *cv2.CHAIN_APPROX_SIMPLE* reducira broj točaka u konturi, zadržavajući samo krajnje točke. Pretpostavljajući da je objekt od interesa najveći objekt na slici, odabire se kontura s najvećom površinom (sl. 4.11.). Nakon identifikacije najveće konture unutar maske, slijedi pronalazak točaka ekstrema (sl. 4.12.) pronađene konture. Cilj je pronaći točke ekstrema koje predstavljaju krajnje granice objekta na slici. Očekivan je pronalazak dviju točaka koje su među najvišim točkama pronađene konture, pri čemu je jedna od točaka, točka koja je najviše u lijevo, dok druga predstavlja najdesniju točku pronađene konture. Također, potrebno je pronaći i par točaka koji se odnosni na najniže točke pronađene konture, od kojih se jedna odnosi na točku konture koja je najviše u lijevo, a druga u desno. Točke unutar konture prvo se sortiraju po y-koordinati čime je omogućeno razdvajanje gornjih i donjih točaka identificirane konture. Zatim, na osnovi sortiranih točaka, izračunavaju se točke koje pripadaju grupi gornjih 10% točaka i točke koje pripadaju grupi donjih 10% točaka. Gornje se točke zatim sortiraju po x-koordinati kako bi se pronašla točka koja je najviše u lijevo i ona najviše u desno te pronađene točke postaju gornje lijeva i gornje desna točka ekstrema. Slično, donje točke se sortiraju radi pronalaska točke koja je najviše ulijevo i one najviše u desno te one postaju donje lijeva i donje desna točka ekstrema. Dobivene točke ekstrema spremaju se unutar liste koja se koristi za crtanje poligona (sl. 4.13.). Iz originalne slike stvara se kopija radi postupka crtanje polinoma. Traženi poligon definiran je ekstremnim točkama koje se povezuju redoslijedom koji formira zatvoreni oblik (povezivanje započinje od gornje lijevo točke, preko donje lijevo i donje desno, sve do gornje desno i na kraju se vraća do gornje lijevo točku). Dodatno se provjerava koja od gornje ekstremnih točaka ima veću vrijednost po y osi te koja od donje ekstremnih točaka ima manju vrijednost po y osi kako bi se osiguralo da su gornje točke i donje točke međusobno horizontalno poravnate. Funkcijom *cv2.fillPoly()* omogućeno je crtanje i popunjavanje polinoma kopije slike. Za kraj se modificirane slike s iscrtanim polinomom spremaju u odredišnu mapu.

Nakon uspješno dobivenog prikaza izvan aplikacije, jednaka logika primijenjena je i unutar Android aplikacije. Proces postprocesiranja izlaza modela odrađen je uporabom triju funkcija.

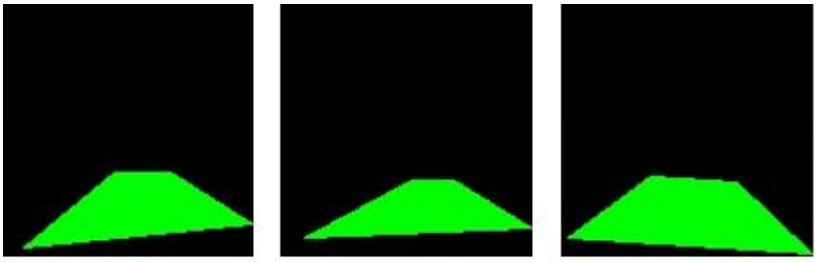
Sl. 4.10. *Predviđanja modela.*



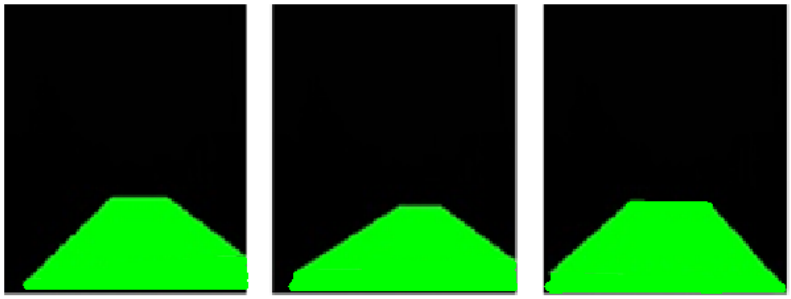
Sl. 4.11. *Najveće konture predviđanja modela.*



Sl. 4.12. *Točke ekstrema pronađenih kontura.*



Sl. 4.13. *Poligoni definirani s 4 pronađene točke.*



Sl. 4.14. *Poligoni s poravnatim horizontalnim stranicama.*

Prva od navedenih funkcija je *postProcessOutput* (sl. 4.15.) koja prima izlazni *TensorBuffer* modela i pretvara ga u *Bitmapu* koja predstavlja masku. Nakon pretvorbe, funkcija na masci pronalazi najveću konturu, točke ekstrema pronađene konture, iscrtava poligon na *Bitmapu* te vraća *Bitmapu* s iscrtanim poligonom.

```
private fun postProcessOutput(outputFeature0: TensorBuffer): Bitmap {
    val mask = floatArrayToMat(outputFeature0.floatArray, rows: 128, cols: 128)
    if (mask.type() != CvType.CV_8UC1 && mask.type() != CvType.CV_32SC1) {
        if (mask.channels() != 1) {
            Imgproc.cvtColor(mask, mask, Imgproc.COLOR_BGR2GRAY) }
        mask.convertTo(mask, CvType.CV_8UC1) }
    val contours = ArrayList<MatOfPoint>()
    val hierarchy = Mat()
    Imgproc.findContours(mask, contours, hierarchy, Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_SIMPLE)
    if (contours.isNotEmpty()) {
        val largestContour = contours.maxByOrNull { Imgproc.contourArea(it) }!!
        if (largestContour != null) {
            val extremePoints = findExtremePoints(largestContour)
            val maskBitmap = Bitmap.createBitmap( width: 128, height: 128, Bitmap.Config.ARGB_8888)
            val canvas = Canvas(maskBitmap)
            val paint = Paint().apply { this: Paint
                color = Color.GREEN // Zelena boja
                alpha = 128
                style = Paint.Style.FILL
                isAntiAlias = true
            }
            val sortedPoints = extremePoints?.sortedBy { it.x }
            if (extremePoints != null && extremePoints.size == 4) {
                val path = Path().apply { this: Path
                    moveTo(extremePoints[0].x.toFloat(), extremePoints[0].y.toFloat()) // Gornji lijevi
                    lineTo(extremePoints[1].x.toFloat(), extremePoints[1].y.toFloat()) // Gornji desni
                    lineTo(extremePoints[3].x.toFloat(), extremePoints[3].y.toFloat()) // Donji lijevi
                    lineTo(extremePoints[2].x.toFloat(), extremePoints[2].y.toFloat()) // Donji desni
                    close()
                }
                canvas.drawPath(path, paint)
            } else {return Bitmap.createBitmap( width: 128, height: 128, Bitmap.Config.ARGB_8888) }
            return maskBitmap } }
    return Bitmap.createBitmap( width: 128, height: 128, Bitmap.Config.ARGB_8888) }
```

Sl. 4.15. *Funkcija postProcessOutput.*

Pronalazak ekstremnih točaka odvojen je u zasebnu funkciju pod nazivom *findExtremePoints* (slika 4.16.). Unutar spomenute funkcije pronađena kontura pretvara se u listu točaka koje se zatim sortiraju prema y koordinati. Zatim se izračunava indeks za gornje i donje postotke točaka te se gornje i donje točke sortiraju prema x koordinati. Za kraj se odabiru točke ekstrema (gornja lijeva, gornja desna, donja lijeva i donja desna). Dodatno se provjerava koja od gornje ekstremnih točaka ima veću vrijednost po y osi te koja od donje ekstremnih točaka ima manju vrijednost po y osi kako bi se osiguralo da su gornje točke i donje točke međusobno horizontalno poravnate.

```
private fun findExtremePoints(contour: MatOfPoint, percentage: Double = 0.1): List<Point>? {
    val points = contour.toList()
    if (points.isEmpty()) {
        return null}
    val sortedByY = points.sortedBy { it.y }
    val cutoffIndex = (sortedByY.size * percentage).toInt()
    val topPoints = sortedByY.subList(0, cutoffIndex)
    val bottomPoints = sortedByY.subList(sortedByY.size - cutoffIndex, sortedByY.size)
    val sortedTopPoints = topPoints.sortedBy { it.x }
    val sortedBottomPoints = bottomPoints.sortedBy { it.x }
    val upperLeftmost = sortedTopPoints.firstOrNull()
    val upperRightmost = sortedTopPoints.lastOrNull()
    val lowerLeftmost = sortedBottomPoints.firstOrNull()
    val lowerRightmost = sortedBottomPoints.lastOrNull()
    if (upperLeftmost != null && upperRightmost != null && lowerLeftmost != null && lowerRightmost != null) {
        val upperY = maxOf(upperLeftmost.y, upperRightmost.y)
        val lowerY = minOf(lowerLeftmost.y, lowerRightmost.y)
        val adjustedExtremePoints = listOf(
            Point(upperLeftmost.x.toDouble(), upperY.toDouble()),
            Point(upperRightmost.x.toDouble(), upperY.toDouble()),
            Point(lowerLeftmost.x.toDouble(), lowerY.toDouble()),
            Point(lowerRightmost.x.toDouble(), lowerY.toDouble()))
        return adjustedExtremePoints
    }
    return null
}
```

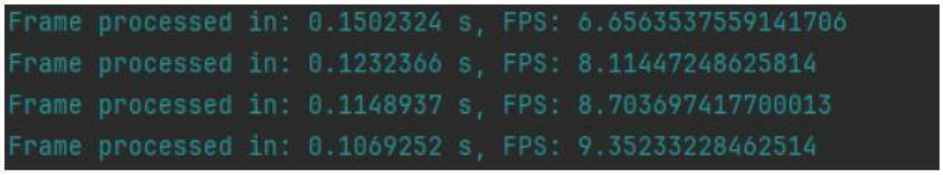
Sl. 4.16. *Funkcija findExtremePoints.*

Zadnja funkcija za obradu izlazne vrijednosti modela je pomoćna funkcija *floatArrayToMat* (slika 4.17.) koja služi za pretvaranje *FloatArray* u *Mat* objekt koji se koristi unutar *OpenCV* biblioteke.

```
fun floatArrayToMat(floatArray: FloatArray, rows: Int, cols: Int): Mat {
    val mat = Mat(rows, cols, CvType.CV_32F)
    mat.put( row: 0,  col: 0, floatArray)
    return mat
}
```

Sl. 4.17. *Pomoćna funkcija floatArrayToMat.*

Za vrijeme rada aplikacije, mjereno je koliko se FPS može obraditi, a rezultati su prikazani na slici 4.18. Prosječno vrijeme potrebno za obradu jednog okvira iznosilo je 0.1069252 sekundi, što rezultira FPS-om od približno 9.35, pokazujući da aplikacija može obraditi preko 9 slika u sekundi. Ova brzina obrade osigurava zadovoljavajuću izvedbu u stvarnom vremenu.



Sl. 4.18. *Prikaz broja obrađenih FPS tijekom rada aplikacije*

Nadalje, evaluacija rada algoritma unutar aplikacije provedena je pomoću odabranih evaluacijskih metrika na testnom skupu od 1000 slika izdvojenih iz deset različitih video snimki (ravna područja ceste te zavoji pri različitim uvjetima (sunce, zalazak, sumrak, kiša) obuhvaćajući državne i lokalne ceste). Iz tablice 4.2 vidljivo je kako model postiže visoku točnost od 0.9460, preciznost od 0.9280, odziv od 0.8325, F1 ocjenu od 0.8776 te prosječni IoU od 0.7786. Ovi rezultati potvrđuju da model učinkovito prepoznaje i segmentira vozne površine, iako postoji prostor za daljnje poboljšanje. Ukupni rezultati pokazuju da je model dobro prilagođen za rad u stvarnim uvjetima te da može pružiti prilično pouzdane predikcije uz visoku brzinu obrade, čineći ga praktičnim za primjenu u sustavima autonomne vožnje i sličnim aplikacijama. Na slici 4.19. moguće je vidjeti primjere rada aplikacije nakon postupka postprocesiranja u stvarnim uvjetima pri radu u realnom vremenu.

Tab. 4.2. *Prikaz rezultata evaluacije modela semantic_model_cs1.tflite unutar aplikacije na kreniranom testnom skupu podataka dobivenom iz video snimki vožnje.*

Model	Broj slika za test	Matrica zabune	Preciznost	Odziv	F1 ocjena	Dice score	IoU	Točnost	Vrijeme inferencije
semantic_model_cs1.tflite	1000	[12328881 245981 638143 3170995]	0.9280	0.8325	0.8776	0.90309	0.7786	0.9460	0.0243

Sl. 4.19. *Primjeri predviđanja modela unutar aplikacije u stvarnim uvjetima.*

Na prikazanim slikama rezultata rada aplikacije za segmentaciju voznog područja vidljivo je kako područja ceste ispod i iznad trapezoidnog oblika nisu označena. Korištenjem funkcija poput *cv2.findContours(),* moguće je da se konture ceste ne prepoznaju ispravno izvan trapezoidnog oblika, posebno ako su rubovi ceste nejasni ili manje vidljivi (većinom kod lokalnih cesta). Kako bi se rezultati segmentacije poboljšali potrebno je osigurati što jasnije ulazne slike s minimalnim šumom te nastaviti treniranje modela s dodatnim podacima koji uključuju različite uvjete ceste i rubne slučajeve za bolje prepoznavanje ceste izvan trenutnih granica. Ujedno, korištenje dubljih modela s više slojeva koji mogu bolje prepoznati složenije strukture može poboljšati performanse segmentacije ceste izvan trenutnih granica. Poboljšanje algoritma za pronalazak kontura kako bi bolje prepoznavao rubove ceste i interpolirao segmente između poznatih točaka također može poboljšati rezultate. Povećanje rezolucije, dulje treniranje modela i naprednije tehnike postprocesiranja mogli bi biti ključni za postizanje boljih rezultata.

5. ZAKLJUČAK

Diplomskim je radom detaljno istražena tema semantičke segmentacije voznog područja koristeći slike dobivene kamerom montiranom na prednjoj strani vozila. Nadalje, razvijena je metoda za semantičku segmentaciju vozne trake, zbog postojeće potrebe za poboljšanjem sigurnosti u autonomnoj vožnji, pri čemu precizna segmentacija voznog područja predstavlja jedan od glavnih problema.

Izborom skupova podataka kao što su KITTI, Cityscapes i ADE20K, omogućena je detaljna evaluacija različitih verzija modela semantičke segmentacije. Evaluacija je uključivala usporedbu performansi modela koji su se pokazali kao iznimno važni u kontekstu računalnog vida i semantičke segmentacije. Pristupi zasnovani na dubokom učenju istaknuti su kao najperspektivniji zbog njihove sposobnosti da precizno identificiraju i klasificiraju različite elemente scene u realnom vremenu.

Nakon temeljite analize i usporedbe modela, model koji je pokazao najbolje performanse, prilagođen je za upotrebu na mobilnim platformama. Konverzija modela iz *Kerasa* u *TensorFlow Lite* format ključan je korak koji je omogućio implementaciju algoritma na Android uređajima. Spomenuta transformacija omogućuje modelu učinkoviti rad uz ograničene resurse mobilnih uređaja. Praktična implementacija algoritma na *Android* platformi pokazala je da model može uspješno funkcionirati u stvarnom vremenu, što dokazuje njegovu primjenjivost u stvarnim situacijama na kolniku. Postprocesiranjem je omogućeno brzo i precizno dodatno obraditi slike prikupljene stražnjom kamerom mobilnog uređaja koja snima scene kolnika ispred vozila.

Ovaj rad ističe tehnička dostignuća kao i važnost kontinuiranog istraživanja i razvoja u području autonomne vožnje. Semantička segmentacija samo je jedan od mnogih problema koje je potrebno razviti i integrirati kako bi se ostvarila potpuno autonomna vožnja. Predloženi model pruža snažan okvir za daljnje inovacije, no istraživanje bi trebalo proširiti i na druga područja autonomne vožnje, poput predviđanja ponašanja drugih sudionika u prometu i prilagodbe uvjetima u prometu.

U konačnici, ova tema pruža osnovu za modernu tehnologiju autonomne vožnje i njezinu implementaciju na mobilnim platformama, nudeći put do inovacije koja bi mogla promijeniti način na koji razumijemo i koristimo autonomna vozila. Daljnji rad trebao bi biti usmjeren na poboljšanje ovih tehnologija, njihovu integraciju s drugim tehnološkim rješenjima i njihovo testiranje u različitim radnim okruženjima. To će dokazati da je potpun i pouzdan.

LITERATURA

[1] Nacionalni plan sigurnosti cestovnog prometa Republike Hrvatske za razdoblje od 2021. do 2023., Vlada Republike Hrvatske, NN 86/2021:8

[2] Aakanksha, A. Seth, S.Sharma, Semantic Segmentation: A Systematic Analysis From State-of-the-Art Techniques to Advance Deep Networks, Journal of Information Technology Research, br.15, sv.1, 1-28, siječanj 2022.

[3] K. S. N. Ripon, S. Newaz, E. A. Ali, J. Ma, Bi-Level Multi-Objective Image Segmentation Using Texture-Based Color Features, 20th International Conference of Computer and Information Technology (ICCIT), 22-24, prosinac 2017.

[4] L. Landrieu, M. Simonovsky: Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs, IEEE/CVF Conference on Computer Vision and Pattern Recognition, ožujak 2018

[5] J. Contreras , S. Sickert , J. Denzler: Region-Based Edge Convolutions With Geometric Attributes for the Semantic Segmentation of Large-Scale 3-D Point Clouds, IEEE journal of selected topics in applied earth observations and remote sensing, vol. 13, 2020

[6] O. Matan, C. J. Burges, Y. LeCun, and J. S. Denker, Multi-digit recognition using a space displacement neural network, in NIPS, 1991, pp. 488–495.

[7] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Backpropagation applied to hand-written zip code recognition, in Neural Computation, 1989.

[8] R. Wolf and J. C. Platt, Postal address block location using a convolutional locator network, in NIPS, 1994, pp. 745–745.

[9] E.Shelhamer, J. Long, T. Darell: Fully Convolutional Networks for Semantic Segmentation, 2016

[10] Seferbekov, S., Iglovikov, V., Buslaev, A. i Shvets, A., 2018. Feature pyramid network for multi-class land segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (str. 272-275).

[11] Z.Wang, Y. Zhao, Y. Tian, Y. Zhang, L. Gao; „The Improved Deeplabv3plus Based Fast Lane Detection Method“, . Actuators 2022, 11, 197. https://doi.org/10.3390/act11070197

[12] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A.

(eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science(), vol 9351. Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28svibanj 2015

[13] H. Rezatofighi, N. Tsoi, J. Y. Gwak, A. Sadeghian, I. Reid, and S. Savarese, Generalized intersection over union: A metric and a loss for bounding box regression. arXiv preprint arXiv:1902.09630, 2019

[14] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). The Cityscapes Dataset for Semantic Urban Scene Understanding (Version 2). arXiv. https://doi.org/10.48550/ARXIV.1604.01685

[15] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso and A. Torralba, "Scene Parsing through ADE20K Dataset," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 5122-5130, doi: 10.1109/CVPR.2017.544.

[16] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In Proc. CVPR, 2010

[17] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In Proc. CVPR, 2014.

[18] WordNet, A Lexical Database for English, s Interneta, https://wordnet.princeton.edu/ [7.12.2023]

[19] Caesar, H., Uijlings, J., & Ferrari, V. (2016). COCO-Stuff: Thing and Stuff Classes in Context (Version 4). arXiv. https://doi.org/10.48550/ARXIV.1612.03716

[20] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. Zitnick. Microsoft COCO: Common objects in context. In ECCV, 2014.

[21] G. J. Brostow, J. Fauqueur, and R. Cipolla, “Semantic object classes in video: A high-definition ground truth database,” Pattern Recognition Letters, 2009, vol. 30, no. 2, pp. 88–97.

[22] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset, 2013., IJRR, 32(11).

[23] NVIDIA, „NVIDIA DRIVE videos“, s Interneta, DRIVE Video Series | Experience the Latest AV Innovations | NVIDIA [6.12.2023]

[24] E.Xie, W.Wang, Z.Yu, A.Anandkumar, J.M.. Alvarez, P. Luo; „SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers“, 2023, The University of Hong Kong, Nanjing University, NVIDIA, Caltech

[25] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. „An image is worth 16x16 words: Transformers for image recognition at scale.“,2020, arXiv

[26] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

[27] H. Zhao, X. Qi, X. Shen, J. Shi, J. Jia; „Icnet for real-time semantic segmentation on high-resolution images.“,2018, In ECCV

[28] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, Philip HS Torr, et al.; „Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers.“,2021, CVPR

[29] AI road semantic segmentation, s Interneta, https://github.com/PatelVatsalB21/AI-Road-Semantic-Segmentation [11.12.2023]

[30] Image segmentation using U-Net, s Interneta, https://github.com/bnsreenu/python_for_microscopists/blob/master/076-077-078-Unet_nuclei_tutorial.py [18.3.2024]

[31] Hong, Y., Pan, H., Sun, W., & Jia, Y. (2021). Deep Dual-resolution Networks for Real-time and Accurate Semantic Segmentation of Road Scenes (Version 2). arXiv. https://doi.org/10.48550/ARXIV.2101.06085 + kod: s Interneta, https://github.com/chenjun2hao/DDRNet.pytorch.git [19.3.2024]

[32] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,”, 2016, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778.

[33] S. Gao, M.-M. Cheng, K. Zhao, X.-Y. Zhang, M.-H. Yang, and P. H. Torr, “Res2net: A new multi-scale backbone architecture,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019.

[34] PyTorch Image Segmentation Tutorial with U-NET, s Interneta, https://github.com/aladdinpersson/Machine-Learning-Collection.git [19.3.2024]

[35] Carvana Image Masking Challenge, s Interneta, https://www.kaggle.com/c/carvana-image-masking-challenge/data [19.3.2024]

[36] Learn to code with Visual Studio Code, s Interneta, https://code.visualstudio.com/learn [19.3.2024]

[37] CARLA simulator, s Interneta, https://carla.org// [26.6.2024]

[38] Android Studio, s Interneta, https://developer.android.com/studio [11.4.2024]

[39] TensorFlow basics and Keras: The high-level API for TensorFlow, s Interneta, https://www.tensorflow.org/guide/keras [19.3.2024]

[40] NumPy, s Interneta, https://numpy.org/about/ [19.3.2024]

[41] Tqdm, s Interneta, https://pypi.org/project/tqdm/ [19.3.2024]

[42] OpenCV, s Interneta, https://pypi.org/project/opencv-python/ [19.3.2024]

[43] Matplotlib, s Interneta, https://matplotlib.org/ [19.3.2024]

[44] E.Beauxis-Aussalet, L. Hardman: „Visualization of Confusion Matrix for Non-Expert Users“, 2014, In IEEE Conference on Visual Analytics Science and Technology (VAST)-Poster Proceedings

[45] Powers, D.M., „Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.“, 2020., arXiv preprint arXiv:2010.16061

[46] All the segmentation metrics!, s Interneta, https://www.kaggle.com/code/yassinealouini/all-the-segmentation-metrics [20.3.2024]

[47] Tensorflow Lite (TFLite), s Interneta, https://www.tensorflow.org/lite [12.4.2024]

[48] Kotlin Programming Language, s Interneta, https://kotlinlang.org/ [15.4.2024]

[49] Camera2 API, s Interneta, https://developer.android.com/media/camera/camera2 [15.4.2024]

SAŽETAK

U sklopu ovog rada opisuje se razvoj i evaluaciju odabranih algoritama za semantičku segmentaciju voznog područja. Rad započinje pregledom područja te ističe potrebu za razvojem naprednih sustava za pomoć vozačima (*ADAS*). Detaljno se analiziraju postojeći pristupi semantičke segmentacije, uključujući segmentaciju zasnovanu na regijama i metode zasnovane na potpuno konvolucijskim mrežama (FCNs), s fokusom na modele kao što su SegFormer i U-Net. Također, istražuje se i evaluira prilagodba odabranih modela s ciljem poboljšanja učinkovitosti i jednostavnosti implementacije. Tijekom rada, odabrani algoritmi testirani su na skupovima podataka kao što su KITTI, Cityscapes i ADE20K, s ciljem identifikacije najučinkovitijeg modela za semantičku segmentaciju voznog područja. Opisana je implementacija modela, koji se pokazao kao najoptimalniji izbor, na mobilnim uređajima pomoću Android platforme. Rad se zaključuje potvrđivanjem uspješnosti implementacije, gdje odabrani model pokazuje zadovoljavajuće performanse u realnom vremenu. Ističe se značaj i potencijal semantičke segmentacije u području autonomne vožnje. Ovo istraživanje potvrđuje važnost daljnjeg razvoja i poboljšanja algoritma segmentacije, ne samo za poboljšanje sigurnosti i performansi autonomnih vozila, već i za primjenu u različitim inteligentnim transportnim sustavima.

Ključne riječi: autonomna vozila, semantička segmentacija, Keras modeli, U-Net, Android aplikacija, stvarno vrijeme obrade

SEGMENTATION OF THE DRIVING AREA FROM THE IMAGE OBTAINED BY A CAMERA MOUNTED ON THE FRONT OF THE VEHICLE

ABSTRACT

Within this work, the development and evaluation of selected algorithms for semantic segmentation of the driving area are described.The paper begins with an overview of the area and highlights the need for the development of advanced driver assistance systems (ADAS). Existing approaches to semantic segmentation are analyzed in detail, including region-based segmentation and methods based on fully convolutional networks (FCNs), with a focus on models such as SegFormer and U-Net. Also, the adaptation of the selected models is investigated and evaluated with the aim of improving the efficiency and ease of implementation. During the work, the selected algorithms were tested on datasets such as KITTI, Cityscapes and ADE20K, with the aim of identifying the most effective model for semantic segmentation of the driving area. The implementation of the model, which proved to be the most optimal choice, on mobile devices using the Android platform is described. The paper concludes by confirming the success of the implementation, where the selected model shows satisfactory performance in real time. The importance and potential of semantic segmentation in the field of autonomous driving is highlighted. This research confirms the importance of further development and improvement of the segmentation algorithm, not only for improving the safety and performance of autonomous vehicles, but also for application in various intelligent transportation systems.

Keywords: autonomous vehicles, semantic segmentation, Keras models, U-Net, Android application, real time processing

ŽIVOTOPIS

Dunja Ćaleta rođena je 20. srpnja 2000. u Osijeku. Završila je srednjoškolsko obrazovanje u I. gimnaziji u Osijeku s odličnim uspjehom te nakon završene srednje škole upisuje preddiplomski sveučilišni studij elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Akademski naziv sveučilišne prvostupnice (*baccalaureus*) inženjerke elektrotehnike stječe 2023. godine. Iste godine upisuje sveučilišni diplomski studij Automobilskog računarstvo i komunikacije.

PRILOZI

Prilog P.4.1: Programski kod obrade trening podatka (*data_pipeline.py*)

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
from tqdm import notebook
import tensorflow as tf
import imgaug.augmenters as iaa
from imgaug.augmentables.segmaps import SegmentationMapsOnImage

np.random.seed(42)
tf.random.set_seed(42)
print("started pipeline")

def visualize_data(input_images, mask_images, num_samples=5):
    fig, ax = plt.subplots(num_samples, 2, figsize=(10, num_samples*2))
    for i in range(num_samples):
        img_idx = np.random.randint(0, len(input_images))
        ax[i, 0].imshow(input_images[img_idx])
        ax[i, 0].set_title("Ulazna slika")
        ax[i, 1].imshow(mask_images[img_idx].squeeze(), cmap='gray')
        ax[i, 1].set_title("Maska")
        for a in ax[i]:
            a.axis('off')
    plt.tight_layout()
    plt.show()

def process_data():
    print("process_data")
    input_images = []
    mask_images = []
    image_path = "model/eval/data/Cityscapes/train_images/"
    mask_path = "model/eval/data/Cityscapes/train_masks/"
    image_list = os.listdir(image_path)
    mask_list = os.listdir(mask_path)
    image_list = [image_path + i for i in image_list]
    mask_list = [mask_path + i for i in mask_list]
    augmenter_zoom = iaa.Affine(scale={"x": (0.8, 1.2), "y": (0.8, 1.2)})
    augmenter_brightness = iaa.Multiply((0.7, 1.3))
    augmenter_darkness = iaa.Multiply((0.7, 1.0))

    for i in notebook.tqdm(range(len(image_list))):
        img, mask = cv2.imread(image_list[i]), cv2.imread(mask_list[i], 0)
        img = cv2.resize(img, (128, 128))
        mask = cv2.resize(mask, (128, 128))
```

```
            mask = np.where(mask > 0, 1, 0)
            mask = SegmentationMapsOnImage(mask, shape=img.shape[:2])
            augmented = augmenter_zoom(image=img, segmentation_maps=mask)
            img_augmented_zoom = augmented[0]
            mask_augmented_zoom = augmented[1].get_arr()

            input_images.append(img_augmented_zoom)
            mask_images.append(mask_augmented_zoom)

            img_augmented_brightness = augmenter_brightness(image=img)
            mask_augmented_brightness = mask_augmented_zoom

            input_images.append(img_augmented_brightness)
            mask_images.append(mask_augmented_brightness)

            img_augmented_darkness = augmenter_darkness(image=img)
            mask_augmented_darkness = mask_augmented_zoom

            input_images.append(img_augmented_darkness)
            mask_images.append(mask_augmented_darkness)

    input_images = np.array(input_images)
    input_images = input_images / 255.

    mask_images = np.array(mask_images)

    input_images = input_images.astype('float32')
    mask_images = mask_images.astype('float32')

    mask_images = mask_images.reshape((mask_images.shape[0],
    mask_images.shape[1], mask_images.shape[2], 1))
    print(mask_images.shape)
    input_img = input_images[0]
    h = input_img.shape[0]
    w = input_img.shape[1]
    l = input_img.shape[2]

    print(f"Tip podataka ulaznih slika: {input_images.dtype}, Min-Max
    vrijednosti: {input_images.min()}-{input_images.max()}")
    print(f"Tip podataka maski: {mask_images.dtype}, Jedinstvene vrijednosti:
    {np.unique(mask_images)}")
    visualize_data(input_images, mask_images, num_samples=5)
    return input_images, mask_images, h, w, l
if __name__ == "__main__":
    process_data()
```

Prilog P.4.2: Programski kod arhitekture *Modela_5* (*model5.py*)

```
import tensorflow as tf
from keras.layers import Input, Conv2D, BatchNormalization, Activation,
SeparableConv2D, MaxPooling2D, Conv2DTranspose, UpSampling2D
from keras.models import Model

def get_model():
    inputs = Input(shape=(128, 128, 3), name="Input_Image")
    x = Conv2D(32, (2, 2), strides=2, padding="same")(inputs)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    previous_block_activation = x

    for filters in [64, 128]:
        x = Activation("relu")(x)
        x = SeparableConv2D(filters, (2, 2), padding="same")(x)
        x = BatchNormalization()(x)
        x = Activation("relu")(x)
        x = SeparableConv2D(filters, (2, 2), padding="same")(x)
        x = BatchNormalization()(x)
        if x.shape[1] <= 16:
            break
        x = MaxPooling2D(3, strides=2, padding="same")(x)
        residual = Conv2D(filters, (1, 1), strides=2, padding="same")
        (previous_block_activation)
        x = tf.keras.layers.add([x, residual])
        previous_block_activation = x
    for filters in [128, 64, 32]:
        x = Activation("relu")(x)
        x = Conv2DTranspose(filters, (2, 2), padding="same")(x)
        x = BatchNormalization()(x)
        x = Activation("relu")(x)
        x = Conv2DTranspose(filters, (2, 2), padding="same")(x)
        x = BatchNormalization()(x)
        x = UpSampling2D(2)(x)
        residual = UpSampling2D(2)(previous_block_activation)
        residual = Conv2D(filters, (1, 1), padding="same")(residual)
        x = tf.keras.layers.add([x, residual])
        previous_block_activation = x
    outputs = Conv2D(1, (2, 2), activation="sigmoid", padding="same")(x)
    return Model(inputs=[inputs], outputs=[outputs])
if __name__ == "__main__":
    model = get_model()
    model.summary()
```

Prilog P.4.3: Programski kod za treniranje *Modela_5* (*model_train.py*)

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.python.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
import sys
from keras.losses import BinaryCrossentropy
sys.path.insert(0, 'C:/Users/caleta/Documents/Diplomski/Algoritmi/alg3/AI-Road-
Semantic-Segmentation/model/pipeline')
sys.path.insert(0, '../../model/model')
from data_pipeline import process_data
from model import get_model
input_images, mask_images, h, y, l = process_data()
tf.keras.backend.clear_session()
model = get_model()

def weighted_binary_crossentropy(y_true, y_pred):
    weights = tf.where(tf.equal(y_true, 1), 2.0, 1.0)
    bce = BinaryCrossentropy()
    return bce(y_true, y_pred, sample_weight=weights)


model.compile(optimizer="rmsprop", loss=weighted_binary_crossentropy,
metrics=["accuracy"])
model_checkpoint_callback = ModelCheckpoint(
    filepath="models/semantic_model_cs_origin_augmented.h5",
    save_weights_only=False,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

callbacks = [
    ReduceLROnPlateau(monitor="val_loss", patience=5, factor=0.1, verbose=1,
min_lr=1e-12),
    model_checkpoint_callback
]
history = model.fit(input_images, mask_images, batch_size=24, epochs=50,
callbacks=callbacks, validation_split=0.1)
```

Prilog P.4.4: Programski kod pretvorbe *Kerasovog* model u *Tensorflow Lite* model (*tflite_transform.py*)

```
import tensorflow as tf
from keras.losses import BinaryCrossentropy

def weighted_binary_crossentropy(y_true, y_pred):
    weights = tf.where(tf.equal(y_true, 1), 2.0, 1.0)
    bce = BinaryCrossentropy()
    return bce(y_true, y_pred, sample_weight=weights)

with tf.keras.utils.custom_object_scope({'weighted_binary_crossentropy':
weighted_binary_crossentropy}):
    model = tf.keras.models.load_model('models/semantic_model_cs1.h5')

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = []

converter.allow_custom_ops=False
converter.experimental_new_converter = True
tflite_model = converter.convert()

with open('models/semantic_model_cs1.tflite', 'wb') as f:
    f.write(tflite_model)

interpreter = tf.lite.Interpreter(model_path="models/semantic_model_cs1.tflite")
interpreter.allocate_tensors()
```

Prilog P.4.5: Programski kod *onSurfaceTextureUpdated* funkcije

```
override fun onSurfaceTextureUpdated(surface: SurfaceTexture) {
    frameCount += 1
    // Procesiraj samo svaki 5. frejm
    if (frameCount % 5 == 0) {
        val startTime = System.nanoTime()

        // Provjeri je li textureView inicijaliziran i nije null
        if (textureView != null && textureView.bitmap != null) {
            val currentFrameBitmap = textureView.bitmap

            val scaledBitmap =
                Bitmap.createScaledBitmap(currentFrameBitmap!!, 128, 128,
true)
            currentFrameBitmap.recycle()

            val tensorImage = TensorImage(DataType.FLOAT32)
            tensorImage.load(scaledBitmap)
            val processedImage = imageProcessor.process(tensorImage)
            val inputFeature0 = TensorBuffer.createFixedSize(
                intArrayOf(1, 128, 128, 3),
                DataType.FLOAT32
            )
            inputFeature0.loadBuffer(processedImage.buffer)
            val outputs = model.process(inputFeature0)
            val outputFeature0 = outputs.outputFeature0AsTensorBuffer

            // Postprocesiranje izlaza modela
            val processedBitmap = postProcessOutput(outputFeature0)

            // Provjeri je li imageView inicijaliziran i nije null
            if (imageView != null) {
                // Prikazivanje slike na ekranu mobitela
                runOnUiThread {
                    imageView.setImageBitmap(processedBitmap)
                }
            }

            val endTime = System.nanoTime()
            val duration = (endTime - startTime) / 1_000_000_000.0
            val fps = 1.0 / duration
            Log.d("Performance", "Frame processed in: $duration s, FPS:
$fps")
        }
    }
}
```