SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 8

SEMANTIČKA SEGMENTACIJA HARMONIČKIM GUSTO POVEZANIM MODELIMA

Anja Delić

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 8

SEMANTIČKA SEGMENTACIJA HARMONIČKIM GUSTO POVEZANIM MODELIMA

Anja Delić

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

ZAVRŠNI ZADATAK br. 8

Pristupnica:	**Anja Delić (0036515733)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Siniša Šegvić

Zadatak: **Semantička segmentacija harmoničkim gusto povezanim modelima**

Opis zadatka:

Semantička segmentacija važan je zadatak računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme vrlo zanimljive rezultate postižu modeli s gusto povezanim konvolucijskim slojevima. Nedavno se pojavila poboljšana organizacija koja postiže veću učinkovitost obrade izbjegavanjem memorijski intenzivnih konvolucija. U okviru rada, potrebno je odabrati okvir za automatsku diferencijaciju te upoznati biblioteke za rukovanje matricama i slikama. Proučiti i ukratko opisati postojeće pristupe za semantičlu segmentaciju slike. Odabrati slobodno dostupni skup slika te oblikovati podskupove za učenje, validaciju i testiranje. Implementirati najmanje složenu inačicu razmatrane arhitekture. Uhodati postupke učenja modela i validiranja hiperparametara. Primijeniti naučene modele te prikazati i ocijeniti postignutu točnost. Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 11. lipnja 2021.

Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću na prenesenom znanju, svim savjetima i strpljenju. Zahvaljujem svojoj majci i prijateljima na podršci.

SADRŽAJ

1. Uvod

Osjetilo vida glavno je ljudsko osjetilo. Vidjeti i razumjeti svijet oko sebe ljudima je trivijalno, no za računala je to vrlo težak zadatak. Računalni vid područje je umjetne inteligencije čiji je cilj ekstrakcija korisnih informacija iz slike[19]. Sukladno napretku u umjetnoj inteligenciji i inovacijama u dubokom učenju, dolazi i do napretka u području računalnog vida, a ubrzanom razvoju dodatno pridonosi povećanje i pojeftinjenje računske snage što omogućuje efikasno treniranje dubljih i kompleksnijih modela.

Kako bi računalo moglo prepoznati i interpretirati korisne značajke iz slike razvijeno je više postupaka razumijevanja slike koji su različite složenosi i imaju različite fokuse. Jedan od jednostavnijih postupaka razumijevanja slike je klasifikacija, dodjeljivanje oznake slici. Detekcija objekta pronalazi i klasificira objekte prisutne na slici, a lokalizacija određuje položaj objekta na slici. Želimo li postići to da računalo može prepoznati i označiti vrste objekata na slici trebamo kombinirati zadatke detekcije, lokalizacije i određivanja granica vrsta objekata. Taj problem poznat je pod nazivom semantička segmentacija slike i predmet je istraživanja ovog rada. Zadatak semantičke segmentacije je pridjeljivanje oznake svakom pikselu slike, gdje oznaka predstavlja značenje piksela, odnosno, pripadnost određenoj kategoriji. Semantička segmentacija ima široku primjenu od prepoznavanja rukom pisanog teksta, analize biomedicinskih slika, prepoznavanja lica, do uporabe u autonomnom vozilima.

Duboke konvolucijske mreže pokazuju se vrlo pogodnima za rješavanje problema segmentacije i postoji puno različitih arhitektura dubokih konvolucijskih mreža koje nastoje postići što bolje performanse. Konvolucijske mreže s gusto povezanim konvolucijskim slojevima pokazuju vrlo dobre rezultate, a pojavile su se i nadogradnje takvih mreža koje pomnim odabirom načina povezivanja konvolucijskih slojeva utječu na smanjenje memorijski zahtjevnih operacija, a da pritom ne dođe do smanjenje točnosti modela. U ovom radu opisan je harmonički gusto povezani model, njegova motivacija i značaj. Provedena je evaluacija i učenje modela FCHarDNet 70. Također, provedeno je učenje modela SwiftNetRN-18 s kojim sam uspoređivala model HarDNet 70.

U poglavljima 2, 3 i 4 opisani su algoritmi i matematički aparat korišten u sus-

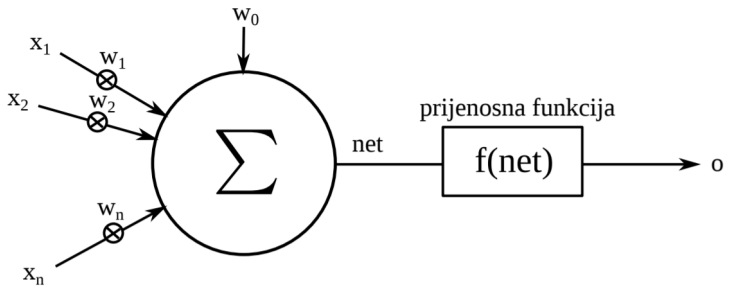
tavu za semantičku segmetnaciju slike. U poglavlju 2 dan je kratak opis neuronskih mreža, njihova motivacija i primjeri prijenosnih funkcija. U poglavlju 3 opisan je postupak učenja neuronskih mreža. Ukratko su opisane funkcije gubitka, optimizacijski algoritmi, postupci regularizacije te algoritam propagacije unatrag. Konvolucijske neuronske mreže opisane su u poglavlju 4. Zadatak semantičke segmentacije detaljno je opisan u poglavlju 5. Fokus ovog rada, harmonički gusto povezani modeli i FCHarDNet70 kao konkretna impplementacija, predstavljeni su u pogavlju 6.1. Opis korištenih skupova podataka dan je u poglavlju 7. Detalji programske izvedbe opisani su u poglavlju 8, a detaljan opis eksperimenta, rezultati i diskusija rezultata izneseni su u poglavlju 9. Kratka rekapitulacija, zaključak rada i prijedlozi budućeg napretka dani su u poglavlju 10.

2. Umjetna neuronska mreža

Umjetne neuronske mreže nastale su po uzoru na biološki središnji živčani sustav i njima se nastoji simulirati postupak učenja i obrade podataka. Umjetna neuronska mreža skup je međusobno povezanih jednostavnih procesnih elemenata, neurona, čija se funkcionalnost temelji na biološkom neuronu i koji služe distribuiranoj paralelnoj obradi podataka. Temelje se na konektivističkom pristupu razvoju inteligentnih sustava pa mogu učiti samostalno na temelju iskustva čak uz manjkave ili nejasne podatke[30]. Neuronske mreže jedan su od trenutno najpopularnijih pristupa strojnom učenju zbog dobrih performansi pri rješavanju raznih problema pa tako i problema semantičke segmentacije slika. Neuronske mreže sastoje se od međusobno povezanih neurona opisanih u poglavlju 2.1.

2.1. Umjetni neuron

Umjetni neuron temeljna je jedinica umjetnih neuronskih mreža i prikazan je na slici 2.1.



Slika 2.1: Model umjetnog neurona (preuzeto iz [30])

Umjetan neuron nastao je po uzoru na biološki neuron. Dendriti umjetnog neurona modelirani su ulazima $x_1, x_2, ..., x_n$ i preko njih umjetni neuron prima signal od prethodnih neurona. Težinama $w_1, w_2, ..., w_n$ je modelirana jakost sinapse, odnosno u kojoj mjeri pojedini primljeni signal utječe na neuron. Tijelo neurona modelira integriranje

svih ulaza neurona u zajednički potencijal net prema izrazu 2.1. Ulazne vrijednosti množe se s pripadnim težinama i umnošku se dodaje prag koji je u formuli i na slici označen s w_0. Akson je ostvaren kao prijenosna funkcija koja određuje konačan izlaz neurona[30].

$$net = \sum_{i=1}^{n} w_i \cdot x_i + w_0 \tag{2.1}$$

2.2. Prijenosna funkcija

Prijenosne funkcije određuju izlaz neurona, povećavaju ekspresivnost mreže i omogućuju modeliranje nelinearnih odnosa. Dobra prijenosna funkcija trebala bi biti nelinearna i diferencijabilna. Nelinearnost omogućuje da model uči nelinearne odnose u podatcima, a diferencijabilnost omogućuje propagaciju unatrag pri optimizaciji težina[30]. Primjeri prijenosnih funkcija navedeni su u nastavku.

Najjednostavnija prijenosna funkcija je funkcija identiteta koja koja ne modificira izlaz tijela neurona:

$$idn(x) = x. \tag{2.2}$$

Funkcija skoka definirana je izrazom 2.3, a može biti i definirana na način da se izlaz umjesto između 0 i 1 mijenja između -1 i 1 (2.4). Funkcija skoka nije diferencijabilna u točki $x = 0$, a u ostalim točkama je njen gradijent 0 pa se ne može koristiti za postupke učenja koji se temelje na gradijentima.

$$step(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.3}$$

$$step(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.4}$$

Sigmoidalna funkcija, poznata i kao logistička prijenosna funkcija, definirana je izrazom (2.5) i zapravo je poopćenje funkcije skoka definirane izrazom 2.3 čija se vrijednost postupno mijenja između 0 i 1. Funkcija tangens hiperbolni (2.6) poopćenje je funkcije skoka definirane izrazom 2.4 čija se vrijednost postupno mijenja između -1 i 1. Tangens hiperbolni se lako može izraziti preko sigmoidalne funkcije (2.6).

$$sigm(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = 2 \cdot sigm(2 \cdot x) - 1 \tag{2.6}$$

Za razliku od funkcije skoka, sigmoidalna funkcija i funkcija tangens hiperbolni su derivabilne pa omogućuju učenje postupcima temeljenim na gradijentnom spustu. Njihove derivacije su:

$$\frac{d}{dx}sigm(x) = sigm(x) \cdot (1 - sigm(x)) \tag{2.7}$$

$$\frac{d}{dx}tanh(x) = 2 \cdot sigm(2 \cdot x) - 1 \tag{2.8}$$

Za klasifikaciju se često koristi funkcija soft-max (2.9) koja je poopćenje sigmoidalne funkcije na slučaj kada je broj klasa veći od 2.

$$softmax(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_i}} \tag{2.9}$$

Prethodno navedene funkcije imaju problem nestajućeg gradijenta (engl. *vanishing gradient*) što znači da prestaje propuštati gradijent unatrag kada uđe u zasićenje. Stoga se danas sigmoidane funkcije i njezine nadogradnje više ne koriste u unaprijednim mrežama, već se najčešće koriste zglobnica i njezina poboljšanja. Zglobnica (engl. *Rectified Linear Unit*) (2.10) sve pozitivne vrijednosti propušta bez prigušenja, dok negativne vrijednsoti uopće ne propušta. Zbog toga što je za sve negativne vrijednosti iznos zglobnice konstantan, njezina derivacija (2.11) na tom području će biti nula što predstavlja problem za metode učenja koje koriste gradijente.

$$relu(x) = max(0, x) \tag{2.10}$$

$$\frac{d}{dx}relu(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{2.11}$$

Propusna zglobnica (engl. *Leaky Rectified Linear Unit*) rješava taj problem i određena je izrazom 2.12 gdje je parametar alfa mali pozitivan broj. Propusna zglobnica propušta i pozitivne i negativne vrijednosti: pozitivne bez prigušenja, a negativne množi faktorom alfa. Takva funkcija ima korisnu derivaciju (2.13) na cijelom području definicije i može se koristiti u postupcima učenja koji se temelje na uporabi gradijenata.

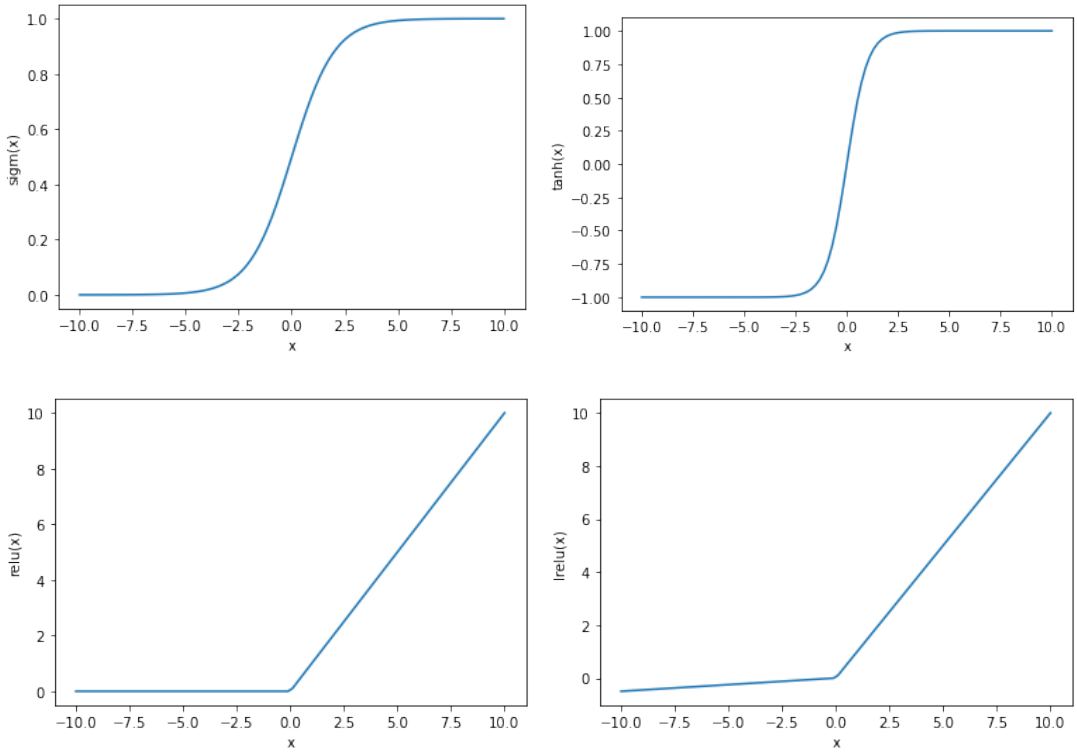
$$lrelu(x) = \begin{cases} x, & x < 0 \\ \alpha \cdot x, & x \geq 0 \end{cases} \tag{2.12}$$

$$\frac{d}{dx}lrelu(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases} \tag{2.13}$$

Još jedna nadogradnja zglobnice je njezina glatka aproksimacija, soft-plus:

$$softplus(x) = ln(1 + e^x). \tag{2.14}$$

Grafički prikaz sigmoide, funkcije tangens hiperbolni, zglobnice i propusne zglobnice dan je na slici 2.2.



Slika 2.2: Grafički prikaz funkcija prijelaza sigmoida, tangens hiperbolni, zglobnica i propusna zglobnica

3. Učenje neuronske mreže

Bitno svojstvo umjetnih neuronskih mreža je sposobnost učenja. Učenje neuronske mreže može se podijeliti u dvije faze: fazu učenja (treniranja) i fazu iskorištavanja (eksploatacije). Učenje je iterativan postupak predočavanja uzoraka neuronskoj mreži tijekom kojeg dolazi do postupnog prilagođavanja težina neurona viđenim podatcima[1]. Težine se mogu podešavati nakon svakog predočenog uzorka, nakon više predočenih uzoraka ili nakon svih predočenih uzoraka, a navedene metode su redom poznate pod nazivima pojedinačno učenje, učenje u minigrupama i grupno učenje. Jedna iteracija predstavlja predočavanje jednog uzorka ili jedne minigrupe neuronskoj mreži, a predočavanje cijelog skupa uzorka naziva se epohom. Učenje s obzirom na vrstu dostupnih podataka možemo podijeliti na nadzirano i nenadzirano. U nadziranom učenju za svaki ulaz predočava se i očekivani izlaz i zadatak mreže je obično klasifikacija ili regresija, dok se kod nenadziranog učenja neuronskoj mreži predočavaju samo ulazi te se najčešće od mreže očekuje grupiranje podataka. Postoje i varijante polunadziranog učenja koje kombiniraju dva prethodno navedena pristupa. U svrhu praćenja postupka učenja, uobičajena praksa je skup podataka podijeliti na tri podskupa: skup za učenje, skup za provjeru i skup za testiranje. Neuronska mreža uči na skupu za učenje, a postupak učenja se povremeno kontrolira nad skupom za provjeru. Tijekom provjere neuronska mreža se ne prilagođava viđenim podatcima. Uz pretpostavku da su globalni trendovi podataka prisutni i u skupu za učenje i u skupu za provjeru, kako učenje napreduje tako će se poboljšavati performanse na skupu za učenje i na skupu za provjeru. U jednom trenutku će se mreža početi prilagođavati specifičnim podatcima i eventualnom šumu u skupu podataka za učenje te će se performanse na skupu za provjeru početi padati, tj. pogreška na skupu za učenje nastavit će padati dok će pogreška na skupu za provjeru početi rasti. To znači da mreža počinje gubiti svojstvo generalizacije i potrebno je prekinuti učenje kako ne bi došlo do prenaučenosti. Kada je postupak učenja gotov, slijedi konačna provjera preformansi mreže koje se mjere na skupu za testiranje. Ovaj rad bavi se rješavanjem problema segmentacije nadziranim učenjem u mini grupama kroz više epoha.

Nadzirano učenje modela sastoji se od nekoliko osnovnih koraka. Prvo je potrebno izračunati izlaz modela i gubitak, a taj postupak naziva se prolaz unaprijed. Slijedi prolaz unatrag, odnosno računanje gradijenta funkcije gubitka s obzirom na parametre modela. Na kraju se provodi optimizacijeki postupak koji je obično varijanta gradijentnog spusta[9].

3.1. Funkcija gubitka

Bitan dio procesa učenja i prilagođavanja težina je funkcija gubitka (pogreške). Funkcija gubitka mjeri razliku između predikcije modela i stvarne vrijednosti. Funkciju gubitka treba minimizirati kako bi se dobila najbolja performansa modela. Najjednostavnija funkcija gubitka je 0-1 gubitak definiran izrazom 3.1. Takav gubitak nije derivabilan što otežava njegovu minimizaciju [9].

$$l_{01}(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} 0, & \mathbf{y} = \hat{\mathbf{y}} \\ 1, & \text{inače} \end{cases} \tag{3.1}$$

Kod semantičke segmentacije, i generalno klasifikacije, kao funkcija gubitka najčešće se koristi unakrsna entropija ako pretpostavimo da na izlazu imamo kategoričku razdiobu preko svih razreda i da se kao prijenosna funkcije izlaznog sloja koristi funkcija softmax. Funkcija sofmax definirana je formulom 2.9 i detaljnije opisana u poglavlju 2.2. Gubitak unakrsne entropije za jedan primjer definiran je kao

$$l = -\sum_{j=1}^{C} y_j log(p_j(\mathbf{x})) \tag{3.2}$$

gdje je C broj razreda, a $\mathbf{x}$ ulaz funkcije softmax. Vektor $\mathbf{x}$ predstavlja vektor logaritama nenormaliziranih vrijednosti ili logit. Često ga zovemo i klasifikacijska mjera. Treba napomenuti da je prethodna formula definira gubitak unakrsne entropije za jedan primjer, dok se u praksi gubitak najčešće računa nad minigrupom. Vektor $\mathbf{y}$ sadrži točnu distribuciju preko svih razreda za dani primjer i najčešće je zadan jednojediničnim vektorom, a p_j je izlaz funkcije softmax za razred j [9].

3.2. Optimizacijski algoritmi

Optimizacija parametara kod neuronskih mreža najčešće se temelji na iterativnom korigiranju parametara oslanjajući se na gradijent [29].

3.2.1. Gradijentni spust

Duboke mreže najčešće se uče algoritmom gradijentnog spusta koji optimizira izlednost predviđanja modela. Gradijentni spust način je pronalaska minimuma diferencijabilne funkcije. Ideja je raditi korake u u najstrmijem smjeru funkcije, to jest, u smjeru suprotnom od smjera gradijenta funkcije u trenutnoj točki i pridruživati novu vrijednost parametrima. Ako je definirana diferencijabilna funkcija gubitka parametrizirana po parametrima modela $J(\Theta)$, iterativnim ponavljenjem sljedećeg postupka dolazi se do minimuma

$$\Theta' = \Theta - \eta \cdot \nabla_\Theta J(\Theta) \tag{3.3}$$

gdje je hiperparametar η korak učenja (eng. learning rate) [21]. Budući da funkcija gubitka dubokih modela nije konveksna, ne postoji garancija da ćemo naći globalni minimum. Pridruživanje nove vrijednosti parametrima može se događati nakon što se modelu predoči čitav skup podataka što može biti vremenski i računski skupo ako se radi o velikom skupu podataka, pojedinačan podatak ili minigrupa. Praksa pokazuje da se najbolji rezultati postižu u posljednjem slučaju [29].

3.2.2. Stohastični gradijentni spust

Stohastični gradijentni spust (eng. Stochastic gradient descent, SGD) je optimizacijski algoritam kod kojeg pridjeljivanje nove vrijednosti parametara događa nakon predočenog pojedinačnog podatka ili minigrupe. Budući da SGD procjenjuje iznos gradijenta u svakom koraku, postoji šum pa u minimumu procijenjeni gradijent neće biti iznosa 0. Potrebno je koristiti promjenjivu stopu učenja koja se postupno smanjuje i prigušuje šum kako bi opitmizacija konvergirala.

Postoje metode koje dodatno ubrzavaju učenje. Jedna od takvih metoda je učenje momentom koje se koristi kod algoritma SGD kod malih konzistentnih gradijentana i šumovite procjene gradijenata za ublažavanje oscilacija promjene smjera gradijenata i vodi k bržoj konvegenciji. Postupak se temelji na računanju prosjeka prethodnih gradijenata:

$$\begin{aligned} v &\leftarrow \alpha \cdot v - \eta \cdot \nabla_\Theta J(\Theta) \\ \Theta &\leftarrow \Theta + v \end{aligned} \tag{3.4}$$

gdje v predstavlja eksponencijalno umanjujuć prosjek prethodnih gradijenata, a $\alpha \in [0, 1)$ koliko je relevantan prethodno izračunati prosjek. Tipične vrijednosti za α su 0.5, 0.9 ili 0.99 i mogu se mijenjati tijekom učenja [29].

3.2.3. Adam

Uz SGD se danas često koriste noviji algoritmi s adaptivnim stopama učenja. Primjeri takvih algoritama su AdaGrad, RMSProp i Adam. U nastavku je opisan algoritam Adam koji je korišten u eksprimentima opisanima u poglavlju **??**.

Adam [12] (od engl. Adaptive Moment Estimation) robusan je algoritam koji radi sa širokim skupom parametara i danas ima široku primjenu. Iako je često brži od SGD-a, u nekim slučajevima ipak lošije generalizira od SDG-a. Adam pamti gradijente i kvadrate gradijenata prethodnih koraka uz eksponencijalno zaboravljanje. Moment se implicitno ugrađen u algoritam jer se korekcija parametara radi prema prosječnom gradijentu.

Uz osnovni algoritam, postoji i niz modifikacija kao što su AdaMax, Nadam i AMSGrad.

3.3. Regularizacija

Tehnike regularizacije sprječavaju prenaučenost modela, odnosno poboljšavaju generalizaciju bez da se nužno poboljšava i točnost na skupu za učenje. Jedna od metoda regularizacije je modificirati funkciju gubitka dodavanjem norme vektora parametara

$$\tilde{J}(\boldsymbol{\Theta}; \mathbf{X}, \mathbf{y}) = J(\boldsymbol{\Theta}; \mathbf{X}, \mathbf{y}) + \alpha\Omega(\boldsymbol{\Theta}). \tag{3.5}$$

gdje je $\alpha \in [0, \infty)$ regularizacijski hiperparametar koji određuje relativan doprinos regularizatora Ω, $\tilde{J}(\boldsymbol{\Theta}; \mathbf{X}, \mathbf{y})$ regularizirani gubitak, $J(\boldsymbol{\Theta}; \mathbf{X}, \mathbf{y})$ podatkovni član.

Regularziacija L_2 normom vektora parametara modela (u literaturi poznato i engl. weight decay) je metoda regularizacije gdje je $\Omega(\boldsymbol{\Theta}) = \frac{1}{2} \parallel \omega \parallel_2^2$. Često se koristi i regularizacije L_1 normom vektora parametara modela gdje je $\Omega(\boldsymbol{\Theta}) = \sum_i \mid \omega_i \mid = \parallel \omega \parallel_1$ [11].

Biblioteka PyTorch, detaljnije opisana u poglavlju 8, nudi mogućnost provođenja L_2 regularizacije izravno u optimizatoru[1] preko argumenta konstruktora `weight_decay`.

3.3.1. Normalizacija po grupi

Normalizacije po grupi [23] je metoda regularizacije koja normalizira izlaz sloja mreže prije primjene prijenosne funkcije oduzimanjem srednje vrijednosti minigrupe i dije-

[1] https://pytorch.org/docs/stable/optim.html

ljenjem sa standardnom devijacijom minigrupe. Normalizacije po grupi poboljšava generalizaciju i ubrzava učenje.

3.4. Algoritam propagacije pogreške unatrag

Algoritam propagacije pogreške unatrag (engl. *backpropagation algorithm*) osnovni je algoritam učenja unaprijednih neuronskih mreža. To je jednostavan i računski nezahtjevan način računanja gradijenta kompozicije funkcija. Izračunava kako svaki primjer iz skupa za učenje utječe na promjenu težina pojedinog neurona. Nakon propagacije unaprijed, računaju se parcijalne derivacije funkcije pogreške s obzirom na težine izlaznog sloja. Zatim se vraćamo sloj po sloj i računamo parcijalnu derivaciju funkcije pogreške s obzirom na težine sloj kojima trenutni sloj šalje svoj izlaz. Za to se koristi pravilo ulančavanja. Originalni algoritam podrazumijeva da se prethodno opisani postupak provodi nad cijelim skupom za učenje, ali to je računski vrlo zahtjevno pa se u praksi češće koristi verzija algoritma propagacije pogreške unatrag nad minigrupama. Takav algoritam poznat je pod nazivom stohastički algoritam propagacije unatrag[9][30].

4. Konvolucijske neuronske mreže

Mreže koje se koriste u mnogim problemima računalnog vida pa tako i za rješavanje problema semantičke segmentacije slike su duboke konvolucijske neuronske mreže. Potpuno povezani modeli nisu prikladni za obradu RGB slika u kojima se neki objekt može pojaviti na više lokacija jer svaka aktivacija ovisi o svim pikselima što znači da bi potpuno povezani model morao posebno učiti svaki objekt u različitim dijelovima slike. To vodi k lošoj generalizaciji. Također, slike često sadrže velik broj piksela što povlači velik broj parametara u svakoj aktivaciji, a broj parametara ograničen je memorijom GPU-a. Duboki konvolucijski modeli imaju prednost nad potpuno povezanim modelima jer u obzir uzimaju lokalne ovisnosti. Konvolucijska mreža obično sarži konvolucijske slojeve, slojeve sažimanja i potpuno povezane slojeve.

4.1. Konvolucijski sloj

Konvolucijski slojevi služe kao izlučivači značajki (engl. *feature extractors*). Početni slojevi služe za izlučivanje jednostavnijih značajki, dok dublji slojevi služe za izlučivanje kompleksinijih značajki. Stoga se izlazi slojeva nazivaju mape značajki (engl. feature maps). Svaki sljedeći sloj koristi mape značajki prethodnog sloja za prepoznavanje značajki više razine, a u svrhu očuvanja ekspresivnosti, dubljim slojevima mreže se povećava broj mapa značajki.

Konvolucijski slojevi na ulaz primaju tenzore trećeg reda. Izlazi konvolucijskog sloja također su tenzori trećeg reda i nazivamo ih mapama značajki. Težine, odnosno jezgra, su tenzori četvrtog reda, imaju dvije prostorne dimenzije, jednu dimenziju po mapama ulaznog tenzora i jednu dimenziju po mapama izlaznog tenzora. Prostorne dimenzije jezgre obično su male u odnosu na sliku: 3, 5 ili 7. Pomak (engl. *stride*) određuje način pomicanja jezgre i nadopunjavanje (engl. *padding*). Što je korak veći to će izazna dimenzija biti manja. Kako na krajevima ulaza ne bi došlo do gubitka informacije, dodaje se nadopuna (engl *padding*) kao dodatni stupci i redci na rubovima ulazne matrice čiji su elementi najčešće popunjeni nulama. Konvolucijski slojevi nad

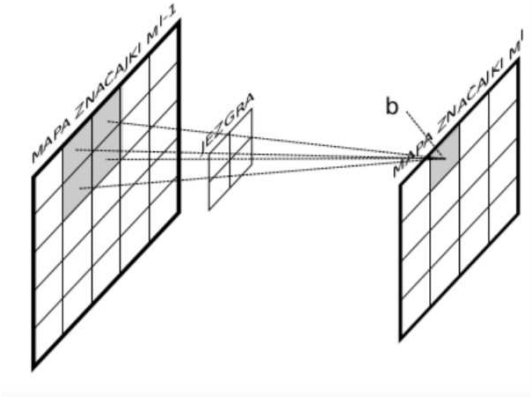
ulazom i jezgrom provode operaciju konvolucije. U domeni stojnog učenja konvolucija je ekvivalnta korelaciji. Konvolucija u kontinuiranom slučaju definirana je izrazom

$$h(t) = (w * x)(t) = \int_{D(w)} w(\tau)x(t-\tau)d\tau \tag{4.1}$$

gdje je x ulazni tenzor, w jezgra, odnosno tenzor slobodnih parametara, a h mapa značajki. U diskretnom slučaju definicija konvolucije je

$$S(i,j) = (K * I)(i,j) = \sum_{m=m_{min}}^{m=m_{max}} \sum_{n=n_{min}}^{n=n_{max}} K(m,n) \cdot X(i+m, j+n) \tag{4.2}$$

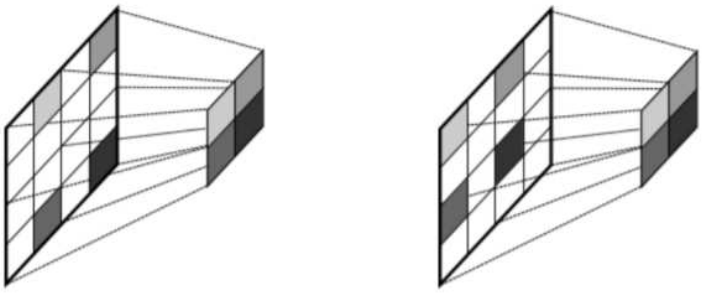
gdje je I ulazni tenzor, K jezgra, a S izlazna mapa značajki. Operacijom konvolucije smanjuje se prostorna dimenzija mape značajki. Osnovna svojstva konvolucije su da modelira lokalne interakcije i dijeli parametre. To znači da elementi izlazne mape značajki ovise o lokalnom susjedstvu elemenata ulazne mape značajki, a dijeljene parametara se očituje u tome što se svi elementi mape značajki računaju uz pomoć istog skupa parametara. Dakle, izlazna reprezentacija će biti ekvivarijantna na pomak. Zbog toga što u konvolucijskom sloju svaki izlaz ovisi o manjem dijelom ulaza, konvolucijski sloj imat će puno manje parametara od potpuno povezanog sloja [10]. Ilustracija konvolucijakog sloja prikazana je na slici 4.1.



Slika 4.1: Ilustracija konvolucije (preuzeto iz [10])

4.2. Sloj sažimanja

Funkcija sažimanja (engl. *pooling*) mapira skup prostorno bliskih značajki na ulazu u jednu značajku na izlazu. Sažimanje se obično provodi nakon konvolucije. Primjer korištenja sloja sažimanja je kada sliku želimo prevesti u razdiobu po klasama kada rješavamo problem klasifikacije. Vrste sažimanja koje se najčešće koriste su sažimanje maksimalnom vrijednosti, sažimanje srednjom vrijednosti i sažimanje L2 normom. Sažimanje pridonosi invarijantnosti na male pomake, a to je vidljivo na ilustraciji 4.2. [10]



Slika 4.2: Ilustracija sloja sažimanja i invarijantnosti sažimanja na male pomake (preuzeto iz [10])

5. Semantička segmentacija

Jedan od zadataka računalnog vida je segmentacija slike. Segmentacija slike postupak je podjele slike na segmente temeljem karakteristika piksela. Cilj je identificirati objekte i granice objekata na slici kako bi se mogla efikasno analizirati [27]. Za zadatke segmentacije u literaturi se često koristi i izraz gusta predikcija jer je njihov cilj razumijevanje slike na razini piksela. Vrste segmentacije slike su segmentacija instanci, semantička segmentacija i panoptička segmentacija. Segmentacija instanci podrazumijeva pridjeljivanje oznake konkretnog objekta svakom pikselu. Zadatak semantičke segmentacije je pridjeljivanje semantičke oznake klase, tj. oznake kategorije kojima pripadaju objekti na slici, svakom pikselu slike, pri čemu se pojedinačni objekti ne razlikuju nego se samo određuje kojoj kategoriji pojedini objekt pripada. Panoptička segmentacija kombinacija je semantičke segmentacije i segmentacije instanci. Ovaj rad bavi se zadatkom semantičke segmentacije koji je detaljnije opisan u nastavku.

Semantička segmentacija ima široku primjenu. Koristi se za analizu medicinkih slika, navigaciju robota, prepoznavanje lica, analizu rukom pisanog teksta, autonomnu vožnju i dr. Metode za semantičku segmentaciju koje su se koristile u prošlosti temeljile su se na uporabi uvjetnih slučajnih polja (engl. *conditional random fileds*), usporedbi piksela i superpiksela, potencijala piksela, njihovih lokacija i sličnim metodama [26]. Danas se u tu svrhu koriste potpuno konvolucijske neuronske mreže (engl. *fully convolutional neural networks*). To su konvolucijske mreže koje ne sadrže potpuno povezane slojeve već samo konvolucijske slojeve i slojeve sažimanja.

Problem semantičke segmentacije može se shvatiti kao problem klasifikacije na razini piksela, međutim, izlaz modela koji se koristi za klasifikaciju je vektor dimenzije jednake broju klasa, dok izlaz modela za segmentaciju treba očuvati prostorne dimenzije ulaza. Kao što je opisano u poglavlju 4, primjenom konvolucije dolazi do poduzorkovanja (engl. *downsampling*) i smanjenja rezolucije slike što nikako nije pogodno za problem segmentacije jer na izlazu želimo dobiti mapu značajki čije su dimenzije jednake ulaznim. Taj problem rješava se primjenom postupaka naduzorkovanja (engl. *upsampling*) koji vraćaju sliku na originalnu rezoluciju. Jednostavan način kako

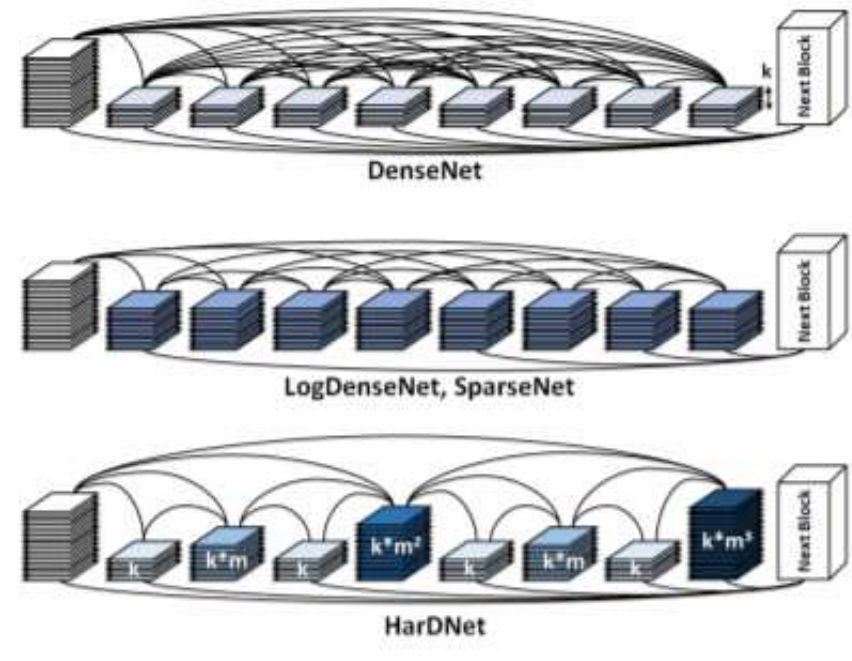
se modeli za klasifikaciju mogu pretvoriti u modele prikladne za semantičku segmentaciju, tj. potpuno konvolucijske modele, opisan je u radu [14]. Postupak se temelji na izbacivanju zadnjeg sloja klasifikacijskog modela, zamjeni potpuno povezanih slojeva konvolucijskim slojevima i bilinearnom naduzorkovanju postupkom unatražne konvolucije (engl. *backwards convolution*) koji se naziva i dekonvolucija. Naduzorkovanje se događa postupno korištenjem prethodnih slojeva (engl. *skip connections*). Bilinearno naduzokovanje njačešće se koristi za semantičku segmentaciju, a računa se uzimajući u obzir težine najbližih piksela. Moderne arhitekutre modela za semantičku segmentaciju temelje se na potpuno konvolcijskim mrežama, a jedna takva arhitektura, FC-HarDNet70, opisana je u poglavlju 6.1.1 i koristi se u eksperimentu opisanom u poglavlju 9.

6. Arhitekture modela za semantičku segmentaciju

Danas se u praksi koriste složenije arhitekture konvolucijskih mreža koje pažljivim kombiniranjem konvolucijskih slojeva, slojeva sažimanja i potpuno povezanih slojeva postižu bolje performanse. Primjeri takvih arhitektura koje se, između ostalog, koriste i za semantičku segmentaciju slika su AlexNet [13], VGG-net [24], Inception [25], U-Net [20], MobileNet [22], ResNet [6] i DenseNet [8]. Da bi prethodno navedeni modeli mogli biti korisni za masovnu uporabu, bitno da omogućiti zaključivanje na temelju neuronskih mreža na uređajima s ograničenim računskim resursima. Takvi modeli imaju manji kapacitet kako bi se smanjio broj računski zahtjevnih operacija i time omogućuju predikciju u stvarnom vremenu. Jedna od primjena koja zahtijeva predikciju u stvarnom vremenu je analiza scena iz persepktive vozača u autonomnim vozilima. U nastavku su opisani modeli HarDNet[4] i SwiftNet[17] koji postižu visoku točnost uz smanjenje kapaciteta modela, a eksperimenti provedeni nam tim modelima opisani su u poglavlju 9.

6.1. Harmonički gusto povezani modeli

Kako bi se omogučilo zaključivanje na temelju neuronskih mreža na uređajima s ograničenim računskim resursima treba smanjiti broj parametara modela jer model s manjim brojem parametara povlači manje računski zahtjevnih operacija. Primjeri računski zahtjevnih operacija su operacija množenja, akumuliranja i operacije nad podatcima s pomičnom točkom (engl. *multiply-accumulate operations or floating point operations*, MAC). Također, treba smanjiti broj pristupa dinamičkoj memoriji s nasumičnim pristupom (engl. *dynamic random access memory*, DRAM) pri čitanju i pohrani parametara modela i mapi značajki. Smanjenje kapaciteta modela obično povlači i smanjenje točnosti, ali to se može spriječiti pažljivim dizajnom mreža koji maksimizira omjer točnosti i parametara mreže.



Slika 6.1: Usporedba gusto povezanih blokova DenseNeta, HardNeta te LogDenseNeta i SparseNeta (preuzeto iz [4])

U radu [4] predstavljena je harmonička gusto povezana mreža (engl. *harmonic densely conncected network*, HarDNet). HarDNet je dizajniran po uzoru na DenseNet iz [8]. DenseNet se sastoji od gusto povezanih blokova (engl. *dense blocks*) koji su međusobno povezani prijelaznim slojem koji se sastoji konvolcijskog sloja i sloja sažimanja. U gusto povezanom bloku se mapa značajki svakog sloja konkatenirana na svaki ulaz svakog sloja koji slijedi. Budući da svaki sloj u DenseNetu u gustom bloku referencira prethodne, broj parametara mreže je značajno smanjen u odnosu na druge arhitekture. Razlike arhitektura gusto povezanih blokova DenseNeta, HarDNeta i sličnih poboljšanja DenseNeta, LogDenseNeta [7] i SparseNeta [28], vidi se na ilustraciji 6.1.

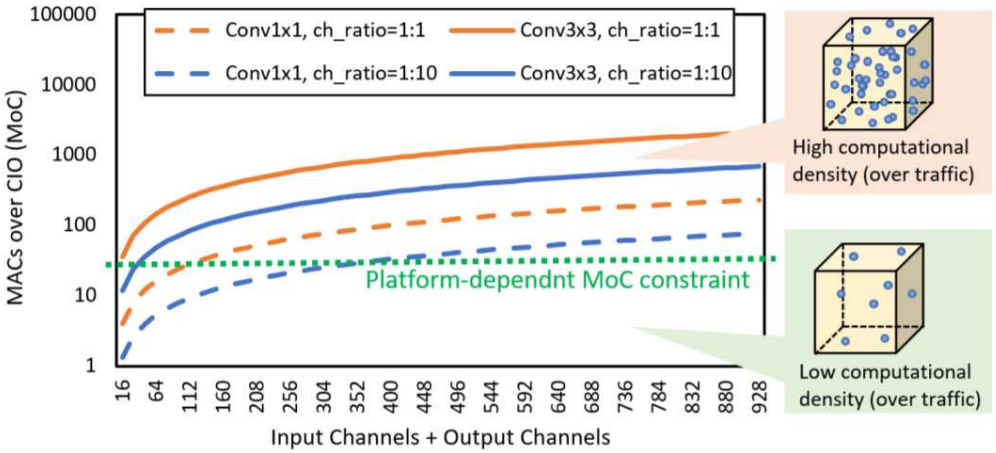
HarDNet je dizajniran tako da reducira broj poveznica među slojevima gusto povezanog bloka i na taj način smanjuje cijenu konkatenacije mapi značajki. Harmonička gusto povezana mreža sastoji se od harmoničkih gusto povezanih blokova (engl. *harmonic dense blocks*, HDB) nakon kojih slijedi konvolucijski sloj s jezgrom dimenzija 1×1 kao prijelazni sloj. HDB blok sastoji je od konvolucijskih slojeva s jezgrom dimenzija 3×3 međusobno povezanih prema sljedećem pravilu: sloj k će biti povezan sa slojem $k - 2^n$ ako 2^n dijeli k, gdje je n nenegativan cijeli broj i $k - 2^n \geq 0$, a sloj s indeksom 0 je ulazni sloj. Ovakav način povezivanja slojeva koristan je jer jednom kada

je sloj s indeksom 2^n obrađen, slojevi od 1 do $2^n - 1$ više nisu potrebni u memoriji. HarDNet daje vrlo dobre rezultate za rješavanje problema semantičke segmentacije i detekcije objekata. Eksperimenti iz [4] fokusirani su na smanjenje vremena predikcije modela (engl. *inference time*) i smanjenje DRAM prometa i pokazuju da HarDNet daje najbolji omjer točnosti i DRAM prometa u odnosu na druge *state-of-the-art* modele. U odnosu na redom FC-DenseNet-103, DenseNet-264, ResNet-50, ResNet-152 i SSD-VGG, HarDNet postiže za 35%, 36%, 30%, 32% i 45% kraće vrijeme predikcije. Ideja HarDNeta je manja memorijska propustnost u odnosu na DenseNet zbog koje će model HarDNet imati manje operacija visoke memorijske intenzivosti. U radu je predložena mjera CIO (engl. *Convolutional Input/Output*) dana izrazom 6.1. To je sumacija dimenzija ulaznih i izlaznih tenzora za svaki konvolucijski sloj. CIO je aproksimacija DRAM prometa i proporcionalna je stvarnoj vrijednosti DRAM prometa.

$$\text{CIO} = \sum_{l} (c_{in}^{(l)} \times w_{in}^{(l)} \times h_{in}^{(l)} + c_{out}^{(l)} \times w_{out}^{(l)} \times h_{out}^{(l)}) \tag{6.1}$$

CIO se lako može minimizirati korištenjem velikih konvolucijskih jezgri, međutim to će utjecati računsku efikasnost i u konačnici na vrijeme predikcije modela. Pokazano je da CIO dominira vremenom predikcije samo kad je računska gustoća (engl. *computational density*) ispod određene granice koja je određena karakteristikama platforme na kojoj se program izvodi. Računska gustoća zapravo je omjer vrijednosti MAC i CIO. Slika 6.2 ilustrira ovakav pristup.

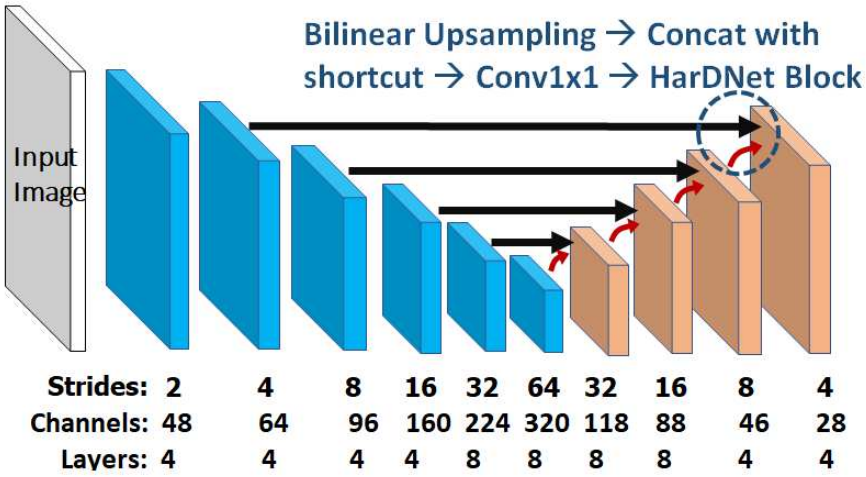
U nastavku je opisana jedna konkretna implementacija HarDNeta za problem semantičke segmentacije.



Slika 6.2: Ilustracija koncepta ograničenja omjera računske i memorijske intenzivnosti (preuzeto s [4])

6.1.1. FC-HarDNet 70

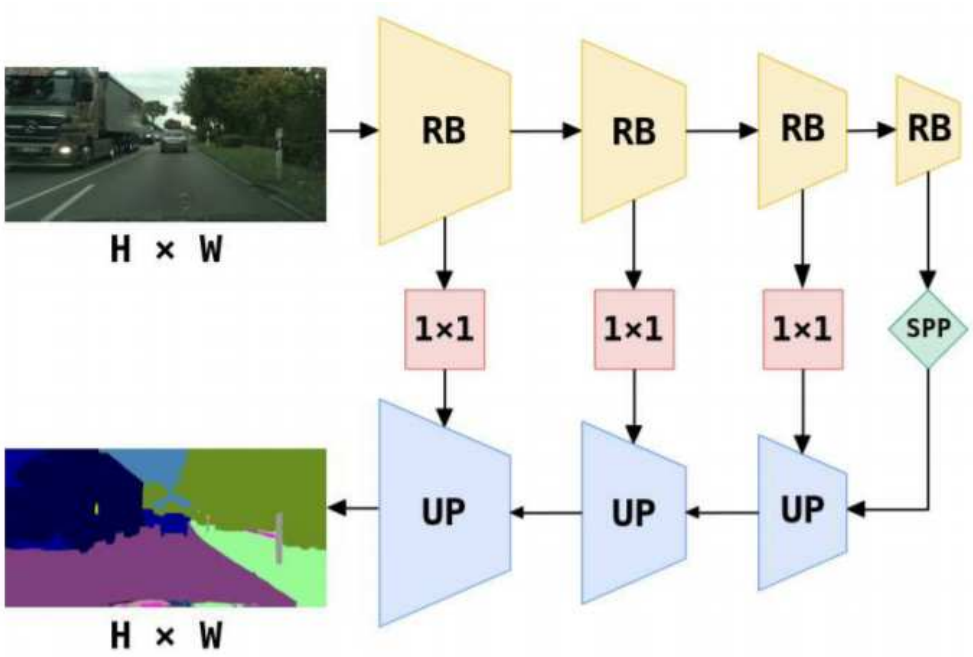
FC-HarDNet 70 [3] je potpuno konvolucijska mreža (engl. *fully convolutional network*, FCN) dizajnirana za problem semantičke segmentacije po uzoru na [4]. Arhitektura modela HarDNet prikazana je na slici 6.3. Model ima jednostavnu enkoder-dekoder U-strukturu. Enkoder dio je sastoji se od gusto povezanih blokova opisanih u 6.1, a na 6.3 označenih plavom bojom. Prvi blok modela sastoji se od 4 konvolucijska sloja s jezgrom 3×3 koji redom na izlazima daju 16, 24, 32 i 48 kanala. Zatim slijedi pet gusto povezanih blokova, a nakon svakog gusto povezanog bloka slijedi prijelazni konvolucijski sloj jezgre 1×1. Broj kanala na izlazu tih prijelaznih konvolucijskih slojeva nakon svakog bloka dan je na slici 6.3. U enkoder dijelu dolazi do poduzorkovanja i smanjenja dimenzija mapi značajki. Dekoder dio, na slici 6.3 blokovi označeni narančastom bojom, provodi postupno naduzorkovanje bilinearnom interpolacijom. Svaki blok za naduzorkovanje je gusto povezani blok i ima poveznicu na odgovarajući gusto povezani blok iz enkoder dijela, a prethodi mu konvolucijski sloj jezgre 1×1. Broj kanala na izlazu svakog gusto povezanog bloka u dijelu za naduzorkovanje dani su na slici 6.3. Na slici 6.3 je također naznačen broj konvolucijskih slojeva u svakom bloku.



Slika 6.3: Arhitektura modela FC-Hardnet 70 s naznačenim pomacima, brojevima kanala na izlazu bloka i brojevima konvolucijskih slojeva u svakom bloku (preuzeto s [3] uz izmjene)

6.2. SwiftNet

SwiftNet[18][17] je model koji postiže vrlo dobre rezultate za semantičku segmentaciju i dizajniran je kako bi se mogao koristiti predikciju u stvarnom vremenu. SwiftNet se sastoji od sastoji od segmentacijskog enkodera, dekodera za naduzokovanje i modula za povećanje receptivnog polja. Arhitektura modela SwiftNet prikazana je na slici 6.4. Kao segmentacijski enkoder koriste se ResNet-18 ili MobileNet V2 koji su predtrenirani na ImageNetu. Dekoder za naduzokovanje povećava dimenzije mape zančajki koristeći lateralne veze s rezidualnim blokovima. Postoje dvije implementacije modula za povećanje receptivnog polja, a to su prostorno piramidalno sažimanje (engl. *spatial pyramid pooling*) i piramidalna fuzija (engl. *pyramid fusion*). U eksperimentima opisanim u 9 koristi se najjednostavnija inačica modela SwiftNetRN-18 bez piramidalne fuzije koja kao enkoder ima model SwiftNet-18.

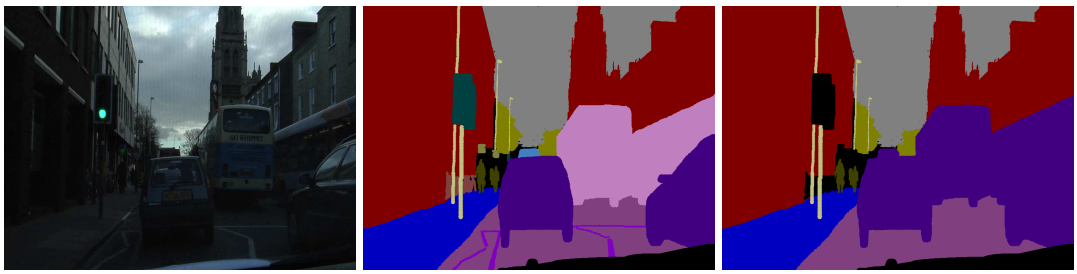


Slika 6.4: Arhitektura modela SwiftNe (preuzeto iz [17])

7. Korišteni skupovi podataka

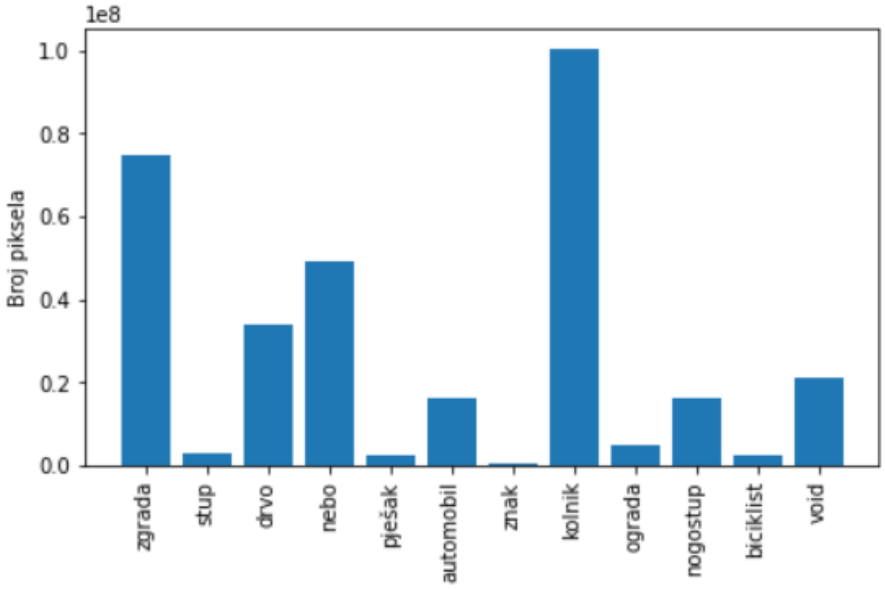
7.1. CamVid

Skup podataka CamVid[1] originalno je nastao kao pet video sekvenci snimljenih tijekom gradske voženje automobilom iz perspektive vozača [2]. Tretiraju li se kao zasebne slike, skup podataka sastoji se od 701 slike dimenzija 960×720. Skup podataka podijeljen je u skup za učenje, skup za provjeru i skup za testiranje koji redom sadrže 367, 101 i 233 slike. Slike originalno sadrže 32 klase od kojih se u eksperimentima opisanima u poglavlju 9 koristi 11 osnovnih klasa, a to su: zgrada, drvo, nebo, automobil, znak, kolnik, pješak, ograda, stup, nogostup i biciklist uz dodatnu klasu *void* koja obuhvaća ostale klase. Na slici 7.1 prikazan je primjer originalne slike, njezine originalne labele s 32 klase i modificirane labele s 11 klasa. U eksperimentima se umjesto trokanalnih (RGB) označenih slika korištene su jednokanalne slike čiji svaki piksel ima vrijednost indeksa klase kojoj pripada. U skupu slika nisu sve klase jednako zastupljene, a odnos zastupljenosti među pojedinim klasama može se vidjeti na stupičastom dijagramu 7.2. Stupičasti dijagram prikazuje ukupan broj piksela svake klase pristunih na slikama iz unije skupova za učenje i provjeru na kojima je provedeno učenje u eksperimentima opisanima u poglavlju 9.



Slika 7.1: Primjer slike iz skupa CamVid, njezine originalne oznake s 32 klase i njezine modificirane oznake s 11 osnovnih klasa

[1] http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/



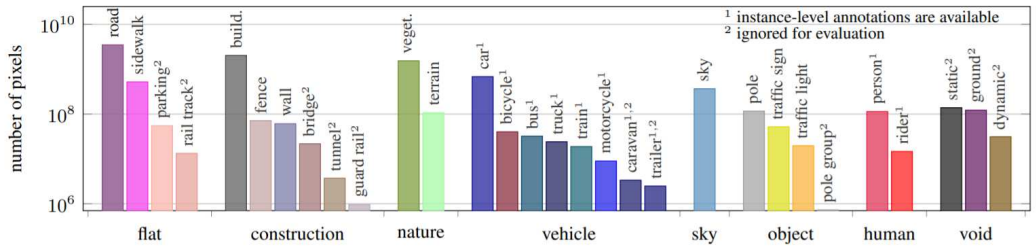
Slika 7.2: Primjer fino označene slike iz skupa Cityscapes

7.2. Cityscapes

Skup podataka Cityscapes[2] nastao je iz video sekvenci snimljenih tijekom gradske vožnje automobilom iz perspektive vozača u 50 gradova. Video sekvence su snimljene tijekom dana pri povoljnim vremenskim uvjetima kroz različita godišnja doba. Skup podataka sadrži 5000 fino označenih slika i 20000 grubo označenih slika rezolucije 2048×1024. Pri validaciji modela FCHarDNet70 (poglavlje 9.2) korištene su samo fino označene slike. Slike sadrže 30 klasa od kojih sam koristila 19 osnovnih. Klase su pomno odabrane i mogu se grupirati u sljedeće kategorije: ravne površine, ljudi, vozila, konstrukcije, objekti, priroda, nebo i praznu kategoriju (engl. *void*). Tipična podjela skupa podataka je podjela na skup za treniranje, skup za provjeru i skup za testiranje koji redom sadrže 2975, 500 i 1525 slika [5]. Broj piksela fino označenih slika po klasama prikazan je na stupičastom dijagramu 7.4.

[2]https://www.cityscapes-dataset.com

Slika 7.3: Primjer fino označene slike iz skupa Cityscapes



Slika 7.4: Broj fino označenih piksela po klasama s navedenim pripadajućim kategorijama (preuzeto iz [5])

8. Programska izvedba

Zadatak ovog rada je evaluirati model FCHarDNet70 koji je naučen na skupu podataka Cityscapes te naučiti isti model na skupu podataka CamVid. Rezultati učenja modela HarDNet uspoređeni su s rezultatima učenja modela SwiftNet. Korišteni modeli opisani su u poglavljima 6.1 i 6.2, a korišteni skupovi podataka u poglavlju 7. U nastavku su opisane korištene tehnologije i detalji programske izvedbe.

Programska izvedba sustava za semantičku segmentaciju slika ostvarena je u programskom jeziku Python po uzoru na [3] uz potrebne izmjene. Neke ideje korištene u implementaciji uzete su iz završnog rada [16]. Programe sam izvodila na platformi Goolge Colab[1] koja omogućava rad s udaljenim repozitorijima te učitavanje i spremanje podataka na Google Disk. Također, Google Colab omogućava korištenje grafičke procesne jedinice (GPU), a na raspolaganju su Nvidia K80s, T4s, P4s i P100s, međutim, nije ih moguće birati.

Za implementaciju koristila sam biblioteke PyTorch[2] i NumPy[3]. Pytorch je biblioteka otvorenog koda najčešće korištena pri implementaciji zadataka strojnog učenja. Temelji se na biblioteci Tocrh programskog jezika Lua. U ovoj programskoj izvedbi korisna je jer omogućava lak i ekfikasan rad s tenzorima uz mogućnost integracije s grafičkim procesorom te je pogodna za modelirnje dubokih konvolucijskih modela. Važna mogućnost PyTorcha je automatska diferencijacija i to mi je posebno korisno jer sam u eksperimentima koristila metode učenje temeljene na izračunu gradijenata. NumPy je biblioteka vrlo efikasna za izvođenje operacija s vekotrima i matricama u Pythonu.

Konvolucijska mreža implementirana je korištenjem već gotovih implementacija slojeva iz `torch.nn`. Svaki konkretan sloj nasljeđuje osnovni razred `torch.nn.Module`. Slojevi se mogu i grupirati korištenjem razreda `torch.nn.Sequential` koji čuva poredak slojeva. Razred `nn.Conv2d` predstavlja konvolucijski sloj koji provodi dvo-

[1] https://colab.research.google.com

[2] https://pytorch.org/

[3] https://numpy.org/

dimenzionalnu konvoluciju. Kao sloj sažimanja korišten je razred `nn.AvgPool2d`, `nn.ReLU` predstavlja zglobnicu, a razred `nn.BatchNorm2d` je korišten za normalizaciju grupe.

Za učitavanje i obradu slika koristila sam razrede `torch.utils.data.DataLoder` i `torch.utils.data.DataSet` biblioteke PyTorch. Oni omogućuju jednostavno učitavanje i korištenje već postojećih skupova podataka u PyTorchu kao i vlastitih skupova podataka. Razred `DataSet` je apstraktan razred koji predstavlja skup podataka. Služi za učitavanje i spremanje podataka i pripadajućih labela. Svi izvedeni razredi trebaju implementirati metodu `__getitem__()` za dohvaćanje podataka i opcionalno metodu `__len__()` za dohvaćanje duljine podataka. Razred `DataLoder` omata primjerak razreda `DataSet` objektom koji može vratiti iterator (engl. *iterable*) s metodama `__iter__()` i `_ next__()` za jednostavno dohvaćanje podataka. Također, `DataLoder` nudi mogućnost konfiguriranja redoslijeda učitavanja podataka, automatskog grupiranja podataka u grupe (engl. *batch*) i višedretveno porcesiranje. Implementacija razreda `DataSet` korištena za učitavanje i obradu skupova CamVid i Cityscapes nalazi se u modulima `camvid_loader.py` i `cityscapes_loader.py`. PyTorch također omogućuje jednostavnu obradu podataka prije korištenja kroz modul `transforms` biblioteke `torchvision`. Modul `transforms` nudi mnogo razreda za obradu slike, a neki primjeri su `Normalize` za normalizaciju slike, `ToTensor` za prevođenje slike ili polja u tenzor i `RandomHorizontalFlip` za slučajno zrcaljenje slike. Takve transformacije se mogu kombinirati korištenjem razreda `Compose`. U radu nisam koristila modul `torchvision.transforms` već sam koristila implementaciju nastalu po uzoru na taj modul iz [3] uz potrebne izmjene. Korištena implementacija prilagođena je za provođenje jednakih transformacija i na slikama i na pripadnim labelama.

Kao strukturu podataka za pohranu i rad s ulazima, izlazima i parametrima modela koristila sam razred `torch.Tensor`. Razred `Tensor` predstavlja višedimenzionalnu matricu i sličan je razredu `ndarray` iz biblioteke `NumPy`. Međutim, `Tensor` je optimiziran za automatsku diferencijaciju i operacije nad primjercima razreda `Tensor` je moguće izvoditi na grafičkim procesnim jedinicama (GPU) i sličnim akceleratorima. To je korisno ako je potrebno koristiti matrice velikih dimenzija i značajno smanjuje vrijeme učenja.

U Pytorchu su dostupne i gotove implementacije mnogih optimizatora, funkcija gubitka i schedulera. Kao optimizator koristila sam `torch.optim.Adam` , za funkciju gubitka `torch.nn.CrossEntropyLoss` i `torch.optim.lr_scheduler.CosineAnnealingLR` kao raspoređivač.

9. Eksperimenti

9.1. Metrike

Za evaluaciju modela korištene su mjere točnost piksela, srednja točnost piksela i omjer presjeka i unije [14] [32].

$$OverallAcc = \frac{\sum_{i=0}^{C} n_{ii}}{\sum_{i=0}^{C} t_i} \tag{9.1}$$

$$MeanAcc = \frac{1}{C} \sum_{i=0}^{C} \frac{n_{ii}}{t_i} \tag{9.2}$$

$$IoU_x = \frac{|A \cap B|}{|A \cup B|} \tag{9.3}$$

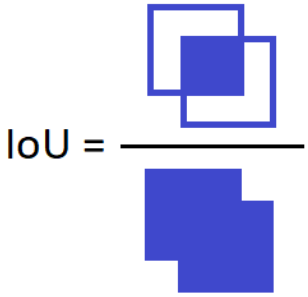
$$mIoU = \frac{1}{C} \sum_{i=0}^{C} \frac{n_{ii}}{t_i + \sum_{j=0}^{C} n_{ji} - n_{ii}} \tag{9.4}$$

Formule 9.1, 9.2 i 9.4 su objašnjene u nastavku. Vrijednost n_{ij} označava broj piksela klase indeksa i za koje je procijenjeno da pripadaju razredu indeksa j (netočno pozitivno klasificirani pikseli), t_i je definiran kao $t_i = \sum_{j=0}^{C} n_{ij}$ (netočno negativno klasificirani pikseli), a C je broj klasa.

Točnost piksela, u literaturi poznata i kao globalna točnost (eng. Overall Accuracy, Pixel Accuracy) je jednostavna mjera koja predstavlja omjer točno klasificiranih piksela i ukupan broj piksela, a računa se prema izrazu 9.1. Točnost piksela često ne odražava stvarno stanje jer ne uzima u obzir zastupljenost pojedinih klasa. Može se desiti da je točnost piksela visoka iako model nema dobre performanse nad klasama koje su slabo zastupljene.

Srednja točnost piksela računa se kao suma omjera točno klasificiranih piksela i ukupnog broja piksela za svaki razred podijeljena s ukupnim brojem razreda, a računa se prema izrazu 9.2. Ova mjera je preciznija od prethodno opisane jer uzima u obzir zastupljenost klasa, međutim, ne penalizira netočno pozitivno klasificirane piksele.

Omjer presjeka i unije (engl. *Intersection over Union, IoU*) [31] široko je korištena metrika za validaciju modela za segmentaciju. Mjera IoU za klasu x definirana je izrazom 9.3 u notaciji skupova gdje je A skup piksela koji pripadaju klasi x, a B skup piksela klasificiranih u klasu x. Slikovni prikaz mjere IoU u kontekstu skupova dan je na slici 9.1. Uspješnost modela izražava se preko srednjeg omjera presjeka i unije (engl. *mean intersection over union*, mIoU), a to je suma omjera presjeka i unije po svakoj klasi podijeljena s brojem klasa (formula 9.4).



Slika 9.1: Slikovni prikaz mjere omjer presjeka i unije (IoU) u kontekstu skupova

Također, u eksperimentima sam mjerila vrijeme predikcije modela (engl. *inference time*). Vrijeme predikcije ovisi o performasama grafičkog porcesora na kojemu model izvodi predikciju i izražava se brojem okvira po sekundi (engl. *frames per second*, fps).

9.2. Rezultati za model HarDNet

Kao praktičan dio ovog rada provela sam eksperiment po uzoru na [4] i [3].

Prvi dio eksperimenta je validacija modela FCHarDNet 70 opisanog u poglavlju 6.1.1. Validirala sam već naučen model na skupu podataka Cityscapes (poglavlje 7.2) [3] prema uputama autora. Rezultati validacije na skupu za provjeru (engl. *validation set*) prikazani su u tablici 9.1. Izmjerila sam vrijeme predikcije na grafičkom procesoru Tesla T4 na platformi Google Colab u iznosu od 20.03 okvira po sekundi.

Drugi dio eksperimenta je učenje modela FCHarDNet70 na skupu CamVid (poglavlje 7.1). Detalji programske izvedbe opisani su u poglavlju 8, a u nastavku su opisani detalji učenja. Koristila sam optimizator Adam[12] s početnim korakom učenja veličine 0.001. Korak učenja se smanjuje nakon svake epohe prema pravilima kosinusnog kaljenja (engl. *cosine annealing scheduler*)[15] do minimalne vrijednosti od 10^{-6} u zadnjoj epohi učenja. Prigušenje težina (engl. *weight decay*) je vrijednsoti $2.5 \cdot 10^{-5}$. Kao funkciju gubitka koristila sam gubitak unakrsnom entropijom. Para-

metri modela su inicijalizirani iz parametara naučenih na skupu podataka ImageNet[1]. Budući da je skup podataka CamVid relativno mali, nakon postupka validiranja hiperparametara i pronalaska onih koji daju najbolje rezultate na skupu za validaciju, model sam trenirala na kombinaciji skupova za treniranje i provjeru, ukupno 468 slika, što je uobičajena praksa kod malih skupova podataka kako bi se dobili bolji rezultati. Model sam testirala na standardnom skupu za testiranje. Treniranje se provodi nad slučajnim izrescima slika dimenzije 448×448 uz veličinu grupe 16 kroz 400 epoha. Prije izrezivanja slike se modificiraju na način da se slučajno horizontalno zrcale i slučajno uvećavaju za faktor od 0.5 do 2. Učenje na slučajnim izrescima poboljšava generalizaciju jer model nauči raspoznavati djelomično zaklonjene objekte, a slučajno izrezivanje iz prethodno slučajno povećanih slika daje dodatno poboljšanje generalizacije. Također, svaka ulazna slika se normalizira na vrijednosti: mean $= [0.406, 0.456, 0.485]$ i std $= [0.225, 0.224, 0.229]$. Testiranje modela provedeno je na slikama pune rezolucije (960×720). Dobiveni rezultati na skupu za testiranje prikazani su u tablici 9.2. Na slici 9.2 prikazani su primjeri slika za testnog skupa, njihove labele i predikcije. Na grafičkom procesoru Tesla T4 izmjerila sam vrijeme predikcije modela od 48.34 okvira po sekundi.

Analizom slike 9.2 mogu se vidjeti razlike labela i predikcija. Vidi se da model radi dobre predikcije za većinu klasa, pogotovo za one koje su u većoj mjeri zastupljene na slici, dok za klase kao što su stupovi i znakovi dolazi do manjih odstupanja. Također, manja odstupanja pristuna su i na rubovima neravnih gradnica klasa, npr. kod klase drvo.

U radu [4] opisani su rezultati učenja modela HarDNet na skupu CamVid na slikama polovične rezolucije, 360×480. Učenje je provedeno na konkretnim implementacijama modela FCHarDNet68, FCHarDNet76 i FCHarDNet84. Rezultate iz 9.2 usporedit ću s rezultatima modela FCHarDNet76 jer je, od prethodno navedenih modela, po broju parametara najsličniji modelu FCHarDNet70. Model FCHarDNet70 ima 4.1 milijun parametara, a model FCHarDNet76 ima 3.5 milijuna parametara. Modeli FCHarDNet68 i FCHarDNet84 imaju redom 1.4 i 8.4 milijuna parametara. Za očekivati je da će se model bolje naučiti na punoj rezoluciji nego na polovičnoj jer ima na raspolaganju više piksela, a to se vidi i na rezultatima. Model FCHarDNet76 ima točnost piksela 91.2% i mIoU 65.8% što znači da FCHarDNet70 naučen na punoj rezoluciji postiže za bolju točnost piksela za 2.5 postotna boda i bolji mIoU za 6.5 postotnih bodova od modela FCHarDNet76 naučenog na pola rezolucije.

[1] https://www.image-net.org

Iz tablica 9.1 i 9.2 vidi se da model FCHarDNet70 naučen na skupu podataka Cityscapes daje bolje rezultate nego kada je naučen na skupu CamVid. Jedan od razloga za takvo ponašanje je što je skup podataka Cityscapes puno veći od skupa podataka CamVid. Cityscapes ima 2975 slika za učenje, a CamVid je treniran na 468 slika iz kombinacije skupa za provjeru i skupa za učenje. Također, slike iz Cityscapesa (2048×1024) su originalno veće rezolucije od onih iz Camvida (960×720). U oba slučaja model je treniran na slučajnim izrescima, na Cityscapesu su korišteni veći izresci dimenzija (1024×1024), a na CamVidu manji izresci dimenzija (448×448).

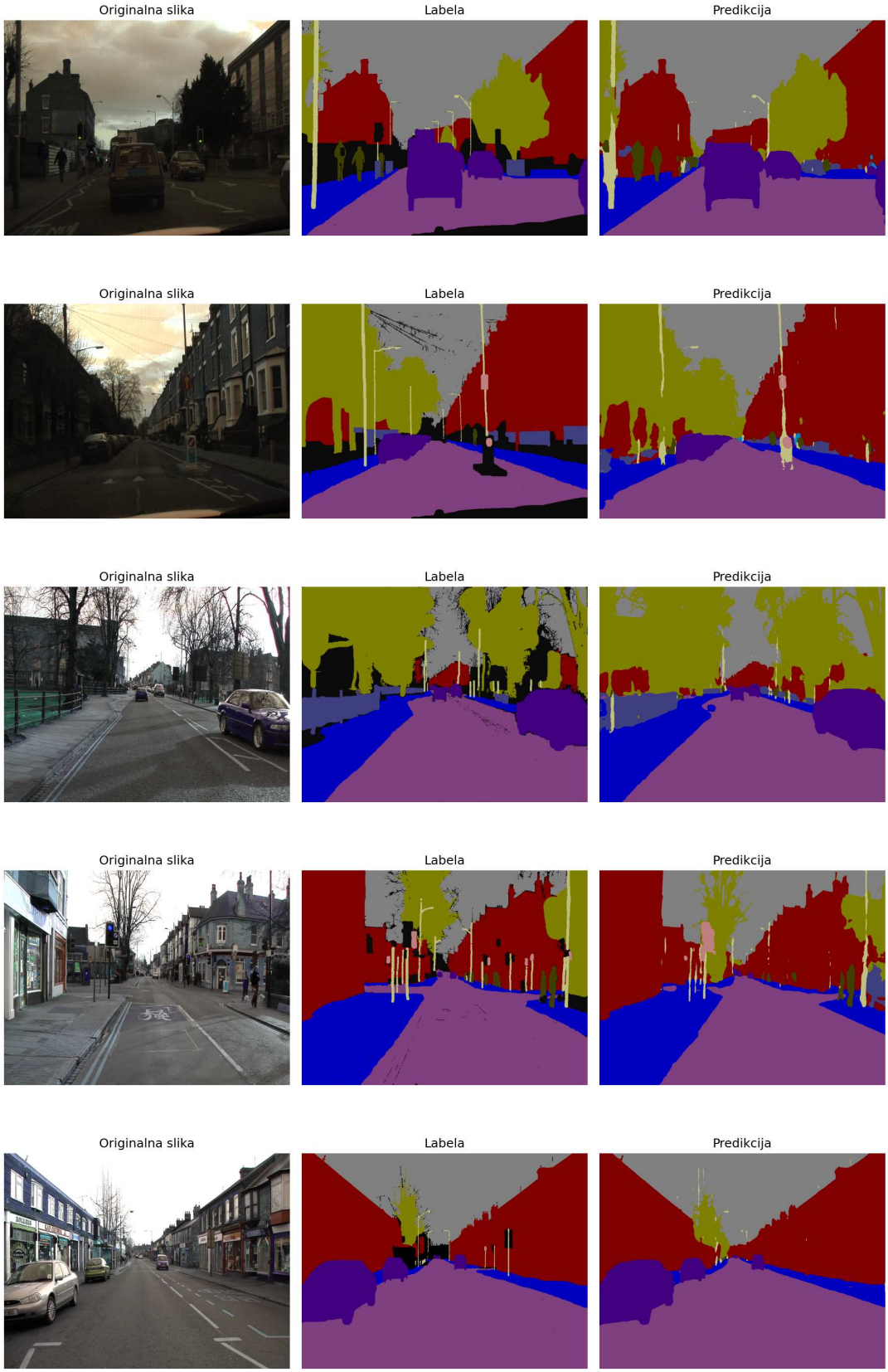
Rezultati iz tablice 9.2 pokazuju da je IoU na nekim klasama znatno lošiji nego na drugima. Radi se o klasama znak i stup koje imaju IoU manji od 40% u odnosu na npr. klasu kolnik koja ima IoU veći od 95%. Na stupičastom dijagramu 7.2 vidi se da su klase znak i stup značajno slabije zastupljene od kolnika. Dijagram pokazuje da i su i klase pješak i biciklist slabo zastupljene, ali se svejedno postiže prihvatljiv IoU od preko 60%. Jedan od razloga za takvo ponašanje je to što su biciklisti i pješaci veći objekti, a stupovi i znakovi su manji objekti. Na skupu Cityscapes nema takvog ponašanja jer se radi o većem skupu podataka i nema tolike razlike u zastupljenosti pojedinih klasa kao kod skupa CamVid[5].

Tablica 9.1: Rezultati validacije na skupu za provjeru modela HarDNet [3] naučenog na skupu podataka Cityscapes

Mjera		Rezultat [%]
Točnost piksela		96.09
Srednja točnost piksela		85.65
mIoU		77.74
IoU za svaku klasu	kolnik	98.07
	nogostup	84.40
	zgrada	92.70
	zid	56.73
	ograda	63.52
	stup	64.88
	semafor	69.89
	prometni znak	78.38
	vegetacija	92.59
	zemljište	64.41
	nebo	94.94
	čovjek	81.61
	vozač	59.72
	automobil	95.11
	kamion	78.31
	autobus	85.98
	vlak	79.68
	motocikl	59.94
	bicikl	76.34

Tablica 9.2: Rezultati na skupu za testiranje modela HarDNet [3] naučenog na skupu podataka CamVid

Mjera		Rezultat [%]
Točnost piksela		93.77
Srednja točnost piksela		79.45
mIoU		72.28
IoU za svaku klasu	zgrada	87.81
	drvo	78.93
	nebo	93.14
	automobil	88.48
	znak	36.84
	kolnik	95.77
	pješak	65.60
	ograda	61.22
	stup	36.30
	nogostup	87.27
	biciklist	63.67



Slika 9.2: Prikaz originalne slike iz testnog skupa, njezine labele i predikcije modela FCHarD-Net70

9.3. Rezultati za model SwiftNet

Radi usporedbe rezultata iz poglavlja 9.2, provela sam eksperimente opisane u nastavku. Model SwiftNet opisan je u poglavlju 6.2, a u eksperimentima sam koristila najjednostavniju inačicu modela SwiftNetRN-18. Detalji učenja isti su kao i za model HarDNet koji su navedeni u poglavlju 9.2 osim što je inicijalna stopa učenja 0.0001. Rezultati su prikazani u tablici 9.3. Na grafičkom procesoru Tesla T4 izmjerila sam vrijeme predikcije od 34.89 okvira po sekundi.

Tablica 9.3: Rezultati na skupu za testiranje modela SwiftNet naučenog na skupu podataka CamVid uz korak učenja 0.0001 i rotiranje kanala

Mjera		Rezultat [%]
Točnost piksela		93.26
Srednja točnost piksela		79.05
mIoU		71.28
IoU za svaku klasu	zgrada	87.57
	drvo	78.61
	nebo	92.36
	automobil	84.15
	znak	35.66
	kolnik	94.87
	pješak	66.07
	ograda	58.45
	stup	35.98
	nogostup	84.85
	biciklist	65.54

Kako bih poboljšala rezultate iz 9.3 pokušala sam učiti model bez rotiranja kanala te sam parametre predtrenirane na ImageNetu ažurirala korakom učenja 0.0001, a slučajno inicijalizirane parametre korakom učenja 0.0004. Za predtrenirane parametre koristila sam propadanje težina iznosa $2.5 \cdot 10^{-5}$, a iznos 0.0001 za sličajno inicijalizirane parametre. U tablici 9.4 prikazani rezultati učenja bez rotiranja kanala uz različite korake učenja za predtrenirane i slučajno inicijalizirane parametre, a u tablici 9.5 prikazani su rezultati učenja s rotiranjem kanala uz različite korake učenja za predtrenirane i slučajno inicijalizirane parametre.

Ovakvi rezultati su zadovoljavajući jer je u radu [18] navedeno da je postignut

Tablica 9.4: Rezultati na skupu za testiranje modela SwiftNet naučenog na skupu podataka CamVid uz korak učenja 0.0001 za predtrenirane parametre i korak učenja 0.0004 za slučajno inicijalizirane parametre bez rotiranja kanala

Mjera		Rezultat [%]
Točnost piksela		93.56
Srednja točnost piksela		79.47
mIoU		72.20
IoU za svaku klasu	zgrada	88.15
	drvo	78.34
	nebo	92.18
	automobil	89.33
	znak	40.30
	kolnik	95.14
	pješak	67.57
	ograda	58.65
	stup	36.98
	nogostup	85.86
	biciklist	61.96

mIoU 72.6%. Usporedbom rezultata iz 9.4 i 9.5 vidi se da učenje bez rotiranja kanala daje za jedan postotni bod bolji rezultat. Takav rezultat je očekivan jer učenje bez rotiranja kanala korišteno pri predtreniranju na ImageNetu pa se bolje može iskoristiti inicijalizacija. Očekivano je da će ažuriranje predtreniranih parametara četiri puta manjim korakom učenja u odnosu na korak učenja kojim se ažuriraju slučajno inicijalizirani parametri dati bolje rezultate, međutim usporedbom rezultata iz tablica 9.3 i 9.5 vidi se da to nije tako. Ažuriranje i predtreniranih i slučajno inicijaliziranih parametara istim korakom učenja daje čak nešto bolje rezultate. Ažuriranje predtreniranih parametara manjim korakom učenja otežava zaboravljanje parametara naučenih na ImageNetu i poboljšava generalizaciju. Istraživanje zašto u ovom eksperimentu nisu postignuti očekivani rezultati jedna je od ideja za budući napredak.

Usporedimo li rezultate dobivene učenjem HarDNeta i SwiftNeta vidimo da modeli postižu sličan mIoU. HarDNet ima veću brzinu zaključivanja (48.34 fps) u odnosu na SwiftNet (34.89 fps) i manje memorijsko zauzeće. Na Colabu, na GPU Tesla T4, najveća veličina minigrupe koja se može zadati za HarDNet je 85, a za SwiftNet 79.

Tablica 9.5: Rezultati na skupu za testiranje modela SwiftNet naučenog na skupu podataka CamVid uz korak učenja 0.0001 za predtrenirane parametre i korak učenja 0.0004 za slučajno inicijalizirane parametre s rotiranjem kanala

Mjera		Rezultat [%]
Točnost piksela		93.44
Srednja točnost piksela		78.08
mIoU		71.11
IoU za svaku klasu	zgrada	87.24
	drvo	78.85
	nebo	92.44
	automobil	87.30
	znak	29.33
	kolnik	95.15
	pješak	68.30
	ograda	56.90
	stup	35.67
	nogostup	86.22
	biciklist	64.79

10. Zaključak

Ovaj rad bavi se zanimljivim problemom računalnog vida, semantičkom segmentacijom. Opisana su osnovna načela umjetnih neuronskih mreža i dubokih modela te njihov postupak učenja. Opisane su funkcije gubitka, optimizacijski algoritam, metode regularizacije i algoritam propagacije unatrag. Naglasak je stavljen na konvolucijske neuronske mreže.

Predstavljen je harmonički gusto povezani model, HarDNet [4], koji postiže vrlo dobre rezultate u rješavanju problema semantičke segmentacije. HarDNet je nastao kao poboljšanje modela DenseNet, a napredak se očituje u tome što ima manju memorijsku propusnost i manje memorijski intenzivnih operacija. Opisan je model FCHarDNet 70 [3] kao jedna konkretna implementacija harmonički gusto povezanog modela. Također, opisan je model SwiftNet [17][18].

Praktični dio rada sastoji se od validacije modela FCHarDNet 70 naučenog na skupu Cityscapes i učenja istog modela na skupu CamVid. Provedeno učenje modela SwiftNetRN-18 na skupu CamVid. Rezultati su izneseni i detaljno objašnjeni. U programskoj izvedbi korištena je biblioteka PyTorch koja nudi mogućnost automatske difrenecijacije, mogućnost izvođenja koda na grafičkim procesnim jedinicama i module za efikasno korištenje matrica, učitavanje i procesiranje slika. Učenje i testiranje modela provedeno je na platformi Google Colab. Postignuti su zadovoljavajući rezultati. FCHarDNet 70 na CamVidu postiže mIoU 72.28%, a SwiftNetRN-18 na CamVidu postiže mIoU 72.20%.

Postoji još mjesta za napredak ovog rada. Moglo bi se detaljnom analizom utvrditi zašto korištenje četiri puta manjeg koraka učenja za parametre predtrenirane na ImageNetu u odnosu na slučajno inicijalizirane parametre u modelu SwiftNet nije dalo bolje rezultate. Također, mogli bi se provesti dodatni eksperimenti i vidjeti kakve rezultate takav pristup daje na modelu HarDNet.

LITERATURA

[1] Jan Šnajder Bojana Dalbelo Bašić, Marko Čupić. Umjetne neuronske mreže. https://www.fer.unizg.hr/_download/repository/UI_12_UmjetneNeuronskeMreze[1].pdf. Pristupljeno: 1. lipnja 2021.

[2] Gabriel J Brostow, Julien Fauqueur, i Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[3] Ping Chao. Fully convolutional hardnet for segmentation in pytorch. https://github.com/PingoLH/FCHarDNet. Pristupljeno: 6. lipnja 2021.

[4] Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, i Youn-Long Lin. Hardnet: A low memory traffic network, 2019.

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, i Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. U *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. U *Proceedings of the IEEE conference on computer vision and pattern recognition*, stranice 770–778, 2016.

[7] Hanzhang Hu, Debadeepta Dey, Allison Del Giorno, Martial Hebert, i J Andrew Bagnell. Log-densenet: How to sparsify a densenet. *arXiv preprint arXiv:1711.00002*, 2017.

[8] Gao Huanga, Zhuang Liu, Laurens van der Maaten, i Kilian Q. Weinberger. Densely connected convolutional networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, stranice 2261–2269, 2017.

[9] Siniša Šegvić Josip Krapac. Duboke unaprijedne mreže. http://www.zemris.fer.hr/~ssegvic/du/du1feedforward.pdf, . Pristupljeno: 1. lipnja 2021.

[10] Siniša Šegvić Josip Krapac. Konvolucijski modeli. http://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf, . Pristupljeno: 6. lipnja 2021.

[11] Sinša Šegvić Josip Krapac. Regularizacija dubokih modela. http://www.zemris.fer.hr/~ssegvic/du/du3optimization.pdf, . Pristupljeno: 2. lipnja 2021.

[12] Diederik P Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Alex Krizhevsky, Ilya Sutskever, i Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[14] Jonathan Long, Evan Shelhamer, i Trevor Darrell. Fully convolutional networks for semantic segmentation. U *Proceedings of the IEEE conference on computer vision and pattern recognition*, stranice 3431–3440, 2015.

[15] Ilya Loshchilov i Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[16] Kara Marijan. Razumijevanje scena iz perspektive vozača semantičkom segmentacijom, 2020.

[17] Marin Oršić, Ivan Krešo, Petra Bevandić, i Siniša Šegvić. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images, 2019.

[18] Marin Oršić i Siniša Šegvić. Efficient semantic segmentation with pyramidal fusion. *Pattern Recognition*, 110:107611, 2021.

[19] S.J.D. Prince. *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012.

[20] Olaf Ronneberger, Philipp Fischer, i Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. U *International Conference on Medical image computing and computer-assisted intervention*, stranice 234–241. Springer, 2015.

[21] Sebastian Ruder. An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descen. Pristupljeno: 2. lipnja 2021.

[22] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, i Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. U *Proceedings of the IEEE conference on computer vision and pattern recognition*, stranice 4510–4520, 2018.

[23] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, i Aleksander Madry. How does batch normalization help optimization? *arXiv preprint arXiv:1805.11604*, 2018.

[24] Karen Simonyan i Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, i Zbigniew Wojna. Rethinking the inception architecture for computer vision. U *Proceedings of the IEEE conference on computer vision and pattern recognition*, stranice 2818–2826, 2016.

[26] Richard Szeliski. *Computer vision: algorithms and applications, 2nd edition*. electronic draft, 2021.

[27] Serena Yeung. Tutorial3: Image segmentation. https://ai.stanford.edu/~syyeung/cvweb/tutorial3.html. Pristupljeno: 6. lipnja 2021.

[28] Ligeng Zhu, Ruizhi Deng, Michael Maire, Zhiwei Deng, Greg Mori, i Ping Tan. Sparsely aggregated convolutional networks. U *Proceedings of the European Conference on Computer Vision (ECCV)*, stranice 186–201, 2018.

[29] Marko Čuipić. Optimizacija parametara modela. http://www.zemris.fer.hr/~ssegvic/du/du3optimization.pdf. Pristupljeno: 1. lipnja 2021.

[30] M. Čupić. Umjetne neuronske mreže, 2016. URL http://java.zemris.fer.hr/nastava/ui/ann/ann-20180604.pdf.

[31] Siniša Šegvić. Konvolucijski modeli za računalni vid: stanje tehnike. http://www.zemris.fer.hr/~ssegvic/du/du12stanje.pdf. Pristupljeno: 6. lipnja 2021.

[32] Jan Šnajder. Strojno učenje: Vrednovanje modela. https://www.fer.unizg.hr/_download/repository/SU-2019-21-VrednovanjeModela.pdf. Pristupljeno: 6. lipnja 2021.

Semantička segmentacija harmoničkim gusto povezanim modelima

Sažetak

Jedan od mnogih zanimljivih zadataka računalnog vida je semantička segmentacija. Modeli s gusto povezanim konvolucijskim slojevima postižu vrlo dobre rezultate pri semantičkoj segmentaciji, a u posljednje vrijeme pojavila su se poboljšanja koja postižu veću učinkovitost obrade izbjegavanjem memorijski intenzivnih operacija. U ovom radu predstavljen je harmonički gusto povezani model te je provedeno učenje i evaluacija modela FCHarDNet 70. Opisan je model SwiftNet i provedeno je učenje modela SwiftNetRN-18. Rezultati su prikazani i detaljno objašnjeni. Dodatno, ukratko su opisane osnove dubokog učenja i konvolucijskih mreža. Opisan je i postupak učenja, korišteni skupovi podataka, detalji programske implementacije i korištene biblioteke.

Ključne riječi: računalni vid, semantička segmentacija, konvolucijske neuronske mreže, HarDNet, SwiftNet

Semantic segmentation with harmonic densely connected models

Abstract

One of the many interesting tasks of computer vision is semantic segmentation. Models with densely connected convolutional layers achieve very good results in semantic segmentation, and recently there have been improvements thatachieve greater processing efficiency by avoiding memory-intensive operations. In this paper, harmonic densely connected models are presented and the learning and evaluation of the FCHarDNet70 model is performed. The model SwiftNet is described and the learning of the SwiftNetRN-18 model is performed. The results are presented and explained in detail. Additionally, the basics of deep learning and convolutional networks are briefly described. The learning process, used data sets, details of program implementation and used libraries are also described.

Keywords: computer vision, semantic segmentation, convolutional neural networks, HarDNet, SwiftNet