Zahvaljujem svome mentoru, prof. dr. sc. Siniši Šegviću na strpljenju i savjetima kojima mi je pomogao tijekom pisanja ovog rada.

Sadržaj

Uvod

Računalni vid interdisciplinarno je znanstveno područje koje proučava i opisuje kako računala mogu procesirati slike ili videa. S obzirom da je za bilo kakvo razumijevanje sadržaja slike potrebna velika količina podataka za učenje (slika), kroz posljednjih petnaestak godina, područje računalnog vida počelo se razvijati mnogo brže, zajedno s područjem znanosti o podatcima (eng. *Data science*).

Jedan od načina na koji možemo postići razumijevanje sadržaja slike ili videa jest klasifikacijom objekata na slici. U praksi, to radimo tako da svakom pikselu na slici pridjeljujemo semantičku oznaku. Klasifikacija promatrana u kontekstu guste predikcije zadatak je semantičke segmentacije. Rješavanje tog problema ima široki raspon područja primjene, od kojih je vrlo zanimljivo područje autonomnih vozila. Još uvijek najbolje rezultate za zadatak semantičke segmentacije pokazuju duboki konvolucijski modeli.

Duboki konvolucijski modeli su prikladni za rješavanje semantičke segmentacije i bilo kojih drugih zadataka u kojima podatci imaju topologiju rešetke, upravo zato jer konvolucijski slojevi mogu kvalitetno modelirati lokalne interakcije, te imaju puno manje parametara od potpuno povezanih slojeva, zbog čega je učenje konvolucijskih modela mnogo kraće.

Duboke modele najčešće učimo nadziranim postupkom, što znači da model uči na temelju ručno označenih podataka. Međutim, ručno označavanje je skupo i vremenski zahtjevno, pa je jedna od tehnika kojoj se u tim slučajevima pribjegava polunadzirano učenje. To je vrsta učenja u kojoj je jedan dio podataka označen, a drugi, obično veći dio nije.

U ovom diplomskom radu opisat ću osnove dubokih neuronskih i konvolucijskih modela. Zatim ću objasniti što je to semantička segmentacija te nabrojati neke od njenih primjena, s posebnim osvrtom na autonomnu vožnju. Nakon toga, usredotočit ću se na implementaciju modela SwiftNet, te na polunadzirano učenje semantičke segmentacije tog modela. Slijedi opis skupova podataka CIFAR-10 i CamVid. Na njima ću testirati model i postupak polunadziranog učenja generiranjem pseudooznaka. Naposljetku ću prikazati rezultate testiranja i dati kratki osvrt na model.

1. Uvod u duboko učenje

U ovom poglavlju opisane su osnove dubokog učenja, neuronskih mreža i konvolucijskih modela. Duboko učenje (eng. *Deep learning*) podskup je strojnog učenja (eng. *Machine learning*), koje spada u široko područje umjetne inteligencije (eng. *Artificial intelligence*). Duboki modeli mogu se promatrati kao modeli strojnog učenja koji primjenjuju više uzastopnih nelinearnih transformacija nad ulaznim podatcima. Često ih zovemo umjetnim ili naprednim neuronskim mrežama.

1.1. Uvod u strojno učenje

Strojno učenje možemo definirati kao programiranje računala na način da optimiziraju neki kriterij uspješnosti temeljem podatkovnih primjera [1]. U strojnom učenju raspolaže se modelom koji je definiran do na neke parametre:

$$H = \{ h(\boldsymbol{x};\ \boldsymbol{\theta}) \}_{\boldsymbol{\theta}} \tag{1.1}$$

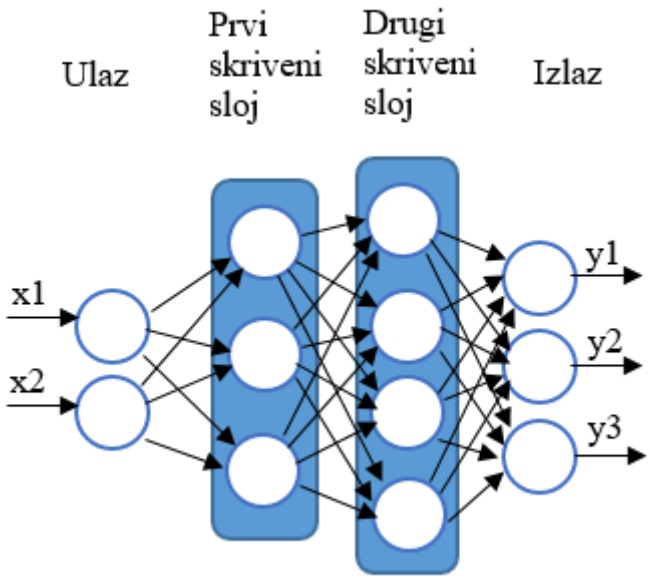
Učenje smatramo optimizacijom parametara modela, $\boldsymbol{\theta}$.

Model zapravo predstavlja skup hipoteza H, odnosno funkcija $h: X \rightarrow Y$, koje svakom podatkovnom primjeru iz skupa X pridjeljuju oznake iz skupa Y. U tom kontekstu, strojno učenje možemo promatrati kao pretraživanje skupa H u potrazi za najboljom hipotezom $h \in H$. Najbolja hipoteza je ona koja daje najtočnije predikcije na primjerima za učenje. U potrazi za tom hipotezom koristimo druge dvije komponente algoritma strojnog učenja, a to su funkcija gubitka i optimizacijski postupak.

Strojno učenje doživjelo je procvat kroz posljednjih 15-ak godina s obzirom na ubrzani razvoj računalne moći, kao i široku dostupnost podataka. Algoritme strojnog učenja primjenjujemo u složenim problemima gdje ne postoji ljudsko znanje o procesu (poput računalnog vida i obrade prirodnog jezika), zatim u dinamički prilagodljivim sustavima (poput robotike i višeagentskih sustava), te u pokušajima otkrivanja znanja u skupovima podataka (eng. *data mining*) [2].

1.2. Neuronske mreže

Neuronska mreža je algoritam strojnog učenja u kojem je model gotovo proizvoljna, nelinearna i diferencijabilna funkcija. „Znanje“ neuronske mreže pohranjena je u vrijednostima težina koje se prilagođavaju tijekom postupka učenja. Neuronske mreže prikladne su za rješavanje problema klasifikacije i regresije, tj. za sve probleme kod kojih postoji kompliciran odnos između ulaznih i izlaznih varijabli. Primjenjuju se za zadatke poput raspoznavanja uzoraka, obrade slika i govora, optimizacijskih problema, obrade podataka, simulacija i sl [3]. Slika 1.1 prikazuje arhitekturu jedne umjetne neuronske mreže.



Slika 1.1 Prikazana neuronska mreža sastoji se od četiri sloja. Njenu arhitekturu čine ulazni sloj od 2 neurona, dva skrivena sloja od 3 i 4 neurona te izlazni sloj od 3 neurona. Arhitekturu kraće zapisujemo i na ovaj način: 2x3x4x3

Neuronska mreža može se sastojati od više slojeva. S obzirom da su korisniku obično vidljivi samo ulazni i izlazni sloj, slojeve između njih nazivamo skrivenim slojevima. Ako su svi neuroni sloja k povezani sa svim neuronima sloja $k + 1$, tada sloj k nazivamo potpuno povezanim slojem.

Važno je također napomenuti da je mreža sa slike 1.2 unaprijedna (eng. *feed-forward*) što znači da ne postoji ciklus između pojedinih slojeva. Postoje i vrste neuronskih mreža koje

imaju povratne veze, te mreže koje su lateralno povezane, a onda i razne vrste hibridnih mreža [1].

Glavni cilj unaprijednih neuronskih mreža je aproksimirati funkciju f^*. Funkcija $y = f^*(x)$ opisuje stvaran odnos između ulaza x i izlaza y, dok za aproksimaciju koristimo funkciju $\hat{y} = f(x, \theta)$, gdje θ predstavlja parametre koje želimo prilagoditi (naučiti) da što bolje aproksimiraju funkciju f^*, koja nije poznata za svaki x, nego samo za skup od N podataka za učenje. Taj skup podataka nazivamo skupom za učenje. Da bi model bio uspješan, mora imati dobru sposobnost generalizacije, odnosno mora dobro aproksimirati podatke koje nikad prije nije vidio (podatke kojih nema u skupu za učenje).

Nakon što podatci uđu u model putem ulaznog sloja, prolaze kroz skrivene slojeve pa sve do izlaznog sloja. Ako uvedemo pomoćne varijable za slojeve $h_L, h_{L-1}, \dots, h_1$, gdje L označava dubinu modela, tj. ukupan broj slojeva, tada model možemo izraziti na ovaj način:

$$h_1 = f_1(x, \theta_1) \tag{1.2}$$

$$h_i = f_i(h_{i-1}, \theta_i) \tag{1.3}$$

$$f(x, \theta) = o(h_L) \tag{1.4}$$

$o(h_L)$ označava izlaz iz zadnjeg sloja. Dakle, za svaki sloj računa se težinska suma koju možemo prikazati kao matrično množenje $W^T x$, kojoj se zatim dodaje pomak zbrajanjem po elementima. Izlaz iz sloja još prolazi kroz nelinearnu (aktivacijsku) funkciju. Funkcija mora biti nelinearna ako želimo riješiti nelinearni problem, odnosno problem klasifikacije linearno-nezavisnih razreda.

1.2.1. Aktivacijske funkcije

Aktivacijske funkcije nazivamo tim imenom jer one imaju ulogu aktivacije pojedinih neurona. Osim što su nelinearne, dobre aktivacijske funkcije često su i monotono rastuće[1], a važno je i da se mogu izvršiti na svakom elementu. Neke od aktivacijskih funkcija koje se često koriste su sigmoida, tangens hiperbolni i zglobnica (ReLU, eng. *Rectified Linear Unit*).

Sigmoida preslikava ulaz na interval $[0, 1]$, a definiramo je kao

[1] Ipak, ovo nije pravilo. Jedna od aktivacijskih funkcija koja nije monotono rastuća je tzv. *Swish* funkcija koju je predložio Google 2017. godine [22].

$$f(x) = \frac{1}{1 - e^{-x}} \tag{1.5}$$

Sigmoida je dugo vremena bila vrlo popularna u dubokim modelima, no u posljednje vrijeme se ne koristi u latentnim slojevima zbog problema nestajućih gradijenata prilikom učenja, te zbog toga što središnje vrijednosti nisu centrirane oko nule. Ipak, još se uvijek koristi u izlaznim slojevima, posebno u povratnim neuronskim mrežama.

Funkcija **tangens hiperbolni** slična je sigmoidi, iako je skalirana po ordinati na interval $[-1, 1]$, što znači da će joj središnje vrijednosti biti centrirane oko nule, za razliku od sigmoide. Međutim, i dalje ostaje problem nestajućih gradijenata. Funkciju definiramo kao:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{1.6}$$

Svakako jedna od najpopularnijih aktivacijskih funkcija u dubokim modelima, u posljednjih nekoliko godina je **zglobnica**, koja se definira vrlo jednostavno:

$$f(x) = \max(0, x) \tag{1.7}$$

Ona propušta pozitivne vrijednosti, dok sve negativne guši. S obzirom da je jednostavna i efikasna operacija koja postiže bolju propagaciju gradijenata od sigmoide i tangensa hiperbolnog, od 2012. je postala najpopularnija aktivacijska funkcija za duboke modele [21]. Ipak, ima nekoliko potencijalnih problema poput toga što središnje vrijednosti nisu centrirane oko nule. Također, pojavljuje se i problem „umirućih neurona“, u slučaju kad izlaz poprima vrijednost 0 za bilo koji primljeni ulaz. Ti neuroni tada blokiraju tok podataka pa nije moguće provesti korekciju prilikom unazadnog prolaska[2].

Problem umirućih neurona može ublažiti propusna zglobnica (eng. *Leaky ReLU*) koju definiramo na sljedeći način:

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & inače \end{cases} \tag{1.8}$$

[2] Ovaj problem rješava se i normalizacijom po grupama (eng. *batchnorm*) [27], koja je ukratko opisana u trećem poglavlju.

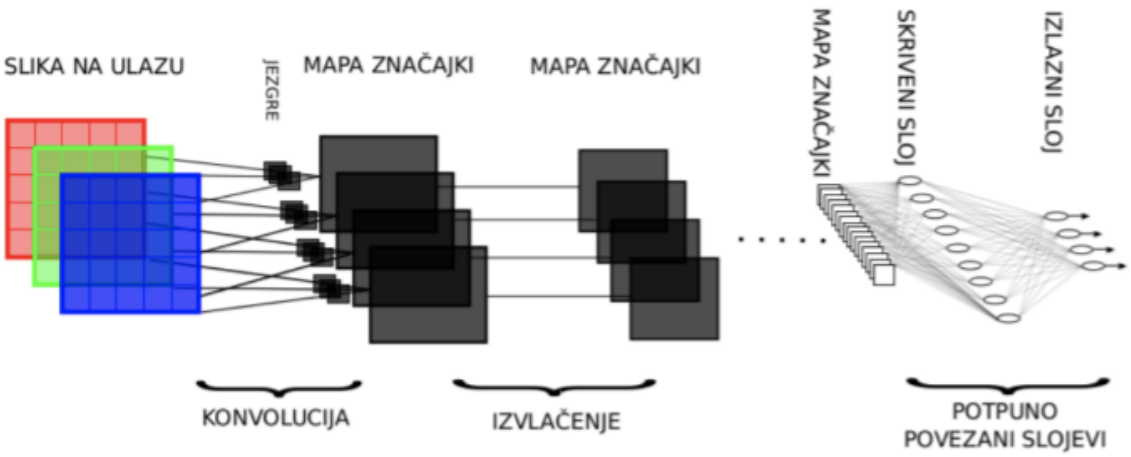
Propusna zglobnica ne guši negativne vrijednosti u potpunosti, nego ih množi s konstantom jako malog iznosa (reda veličine 0.01). Koristi se u modulima koji zahtijevaju bijektivnu odnosno invertibilnu aktivaciju. Primjeri su radovi [24] i [25].

1.3. Konvolucijske neuronske mreže

U neuronskim mrežama iz prethodnog poglavlja pretpostavljali smo da su svi skriveni slojevi potpuno povezani. Broj parametara, tj. ukupan broj težina i pragova mreže koje trebamo ažurirati prilikom učenja mreže, u takvim modelima može biti jako velik, zbog čega učenje može biti sporo, a moguće je i da se model prenauči (eng. *overfitting*), te da izgubi svojstvo generalizacije za nikad viđene podatke. Potpuno povezani modeli mogu naučiti prepoznavati pojedine značajke objekata na slici, ali nailaze na problem već kod translacije slika. Naime, takvim modelima će translatirana slika biti potpuno različita od originala, što znači da će trebati naučiti svaku translaciju odvojeno. Ovdje je važno primijetiti da su ključne značajke objekata sa slike određene lokalnim susjedstvima.

Konvolucijski modeli rješavaju taj problem jer su specijalizirani za podatke s topologijom rešetke, te izrazito dobro modeliraju lokalne interakcije. Osim toga, konvolucijski modeli se mnogo brže mogu učiti zbog puno manjeg broja parametara od potpuno povezanih modela.

Konvolucijske neuronske mreže definiramo kao neuronske mreže koje imaju najmanje jedan konvolucijski sloj umjesto potpuno povezanog sloja. Pored konvolucijskih slojeva u pravilu se korite slojevi sažimanja (eng. *pooling*) i aktivacijske funkcije, od kojih najčešće zglobnica (ReLU). Obično se konvolucijske mreže sastoje od konvolucijskih slojeva, slojeva sažimanja, te od potpuno povezanih slojeva. Slika 1.2 prikazuje kostur konvolucijske neuronske mreže.



Slika 1.2: Primjer konvolucijske neuronske mreže [4]. Mape značajki (eng. *features map*) su izlazi konvolucijskog sloja, tj. rezultati primjene operacije konvolucije pomičnih jezgri na ulaz (što mogu biti ili ulazni podatci, ili druga mapa značajki). Kod slika su to tenzori trećeg reda oblika H x W x C, gdje su H i W prostorne dimenzije, a C semantička dimenzija mape značajki. Obično se višestrukim izvlačenjem povećava semantička dimenzija, dok se prostorne dimenzije smanjuju.

1.3.1. Konvolucijski sloj

Konvolucijski sloj koristi operaciju konvolucije. Konvolucija je matematička operacija čiji su operandi dvije funkcije, a rezultat nova funkcija koja izražava kako oblik jedne funkcije modificira drugu. Operaciju konvolucije u strojnom učenju poistovjećujemo s unakrsnom korelacijom. Definiramo je na sljedeći način [4]:

$$h(t) = (w * x)(t) = \int_{D(w)} w(\tau)x(t+\tau)\,d\tau \tag{1.9}$$

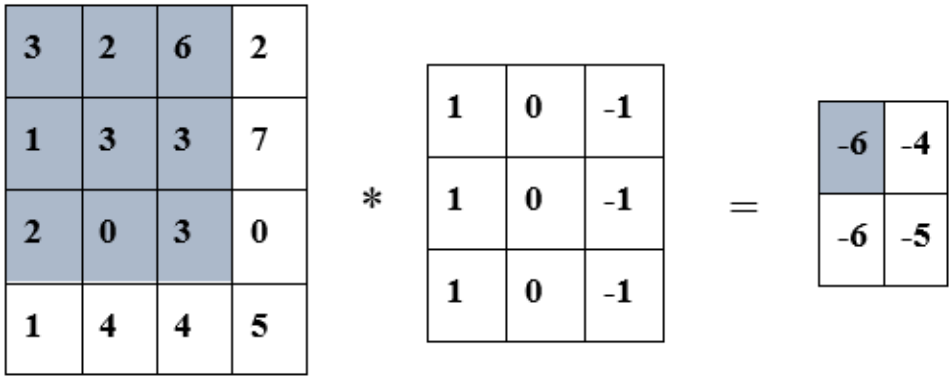
U kontekstu strojnog učenja, funkcija x predstavlja ulaz, funkcija w predstavlja pomičnu jezgru, odnosno filtre koje primjenjujemo na ulaz, a funkcija h je rezultat operacije kojeg nazivamo mapom značajki. Konvolucijski sloj provodi diskretni oblik operacije konvolucije kojeg možemo definirati i za primjenu kroz više dimenzija, na sljedeći način:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) \cdot K(i-m, j-n) \tag{1.10}$$

I predstavlja ulaz koji može biti slika, ili druga mapa značajki, dok K predstavlja jezgru (eng. *kernel*) konvolucijskog sloja. Izlaz konvolucijskog sloja je nova mapa značajki koja se

dobije pomicanjem jezgre (kvadratna matrica manjih dimenzija od ulazne mape značajki) po vrijednostima ulaza, te množenjem matrice jezgre s pojedinim dijelovima ulazne mape značajki. Jezgra klizi po svim kombinacijama ulaza, dajući pritom rezultate izlazne mape značajki, koji će biti jednaki sumi produkata ulaznih i izlaznih elemenata.

Kao što jasno možemo vidjeti na slici 1.3, rezultat ove operacije je mapa značajki manjih dimenzija nego ulaz. Preciznije, uz ulaz dimenzija (W, H), i jezgru dimenzija (F, F), dimenzija izlazne mape značajki bit će $((1 + W - F), (1 + H - F))$.



Slika 1.3. Primjer operacije konvolucije nad ulaznom matricom dimenzije 4x4 jezgrom dimenzija 3x3. Rezultat je matrica dimenzija 2x2. Osjenčan je samo jedan element izlazne matrice koji se dobije nakon primjene filtra nad osjenčanim dijelom ulazne matrice. Ostali elementi izlazne matrice mogu se dobiti na jednak način.

U jezgri konvolucijskog sloja se nalaze vrijednosti koje želimo naučiti. Za njene dimenzije najčešće se uzima relativno mali neparni broj (3, 5 ili 7). Ostali parametri jezgre su korak (eng. *stride*) i nadopunjavanje nulama (eng. *zero-padding*). Korak definira duljinu pomaka posmičnog prozora pri svakom sljedećem pomaku po ulaznoj mapi značajki. U primjeru sa slike 1.4 korak je bio duljine 1. Veličina koraka također utječe na dimenziju izlazne mape značajki, na sljedeći način: Ako je korak duljine S, tada će dimenzija izlazne mape značajki biti jednaka: $((1 + \frac{W-F}{S}), (1 + \frac{H-F}{S}))$. Naposljetku, ako želimo zadržati prostornu dimenziju mape značajki, možemo provesti nadopunjavanje nulama ulazne mape značajki za proizvoljnu vrijednost koju reguliramo hiperparametrom.

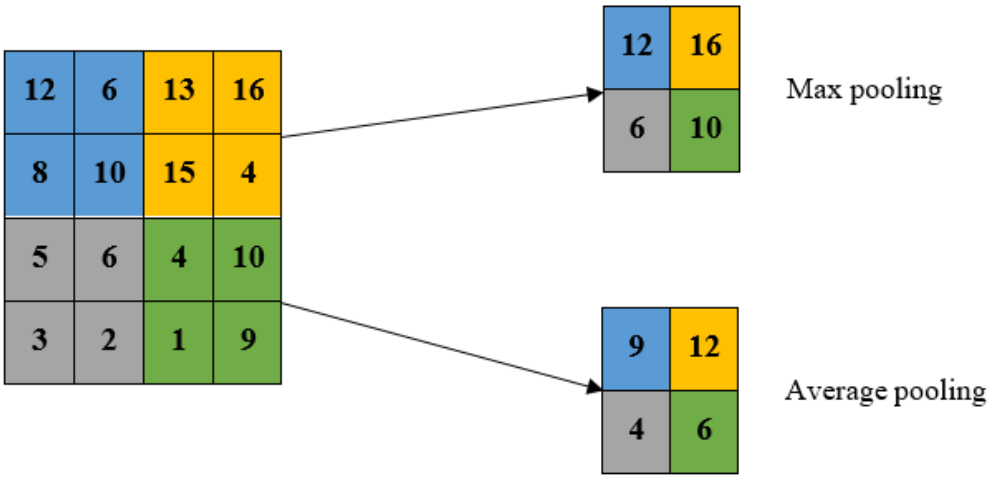
Sada se jasno može vidjeti zašto ovakvi slojevi imaju manje parametara od potpuno povezanih slojeva. Naime, svi elementi mape značajki se računaju uz pomoć istog skupa parametara (konvolucija dijeli parametre), dok u potpuno povezanom sloju sve veze imaju svoje parametre. Nadalje, konvolucija modelira lokalne interakcije jer elementi izlazne mape značajki ovise o lokalnom susjedstvu elemenata ulazne mape značajki. Iz ovoga proizlazi da konvolucijskim modelima lakše možemo naučiti značajke pojedinih objekata bez obzira na lokalitet tog objekta na slici.

1.3.2. Sloj sažimanja

Sloj sažimanja (eng. *Pooling*) koristimo za dodatno smanjenje dimenzija te ubrzavanje računanja. Također, ovaj sloj nam služi za izbjegavanje prenaučenosti modela.

Sažimanje podataka postiže se primjenom funkcija sažimanja. Postupak je sličan konvoluciji jer svaki sloj sažimanja ima definiranu veličinu jezgre (najčešće dimenzije jezgre su 2x2) koja primjenjuje funkciju sažimanja nad područjem ulazne mape značajki, te izračuna vrijednost koju upiše na pripadajuće mjesto u izlaznoj mapi značajki. Obično se sloj sažimanja postavlja nakon jednog ili više konvolucijskih slojeva.

Funkcije sažimanja za koje se pokazalo da daju najbolje rezultate su sažimanje maksimalnom vrijednosti (eng. *Max pooling*) i sažimanje srednjom vrijednosti (eng. *Average pooling*). U pravilu, sažimanje se provodi tako da se mapa značajki podijeli na područja koja se ne preklapaju. Slika 1.4 prikazuje operaciju sažimanja maksimalnom i srednjom vrijednošću nad jednostavnoj 2D mapi značajki malih dimenzija.



Slika 1.4: Primjer sažimanja maksimalnom i srednjom vrijednošću nad mapom značajki. Sažimanje koristi jezgru dimenzija 2x2.

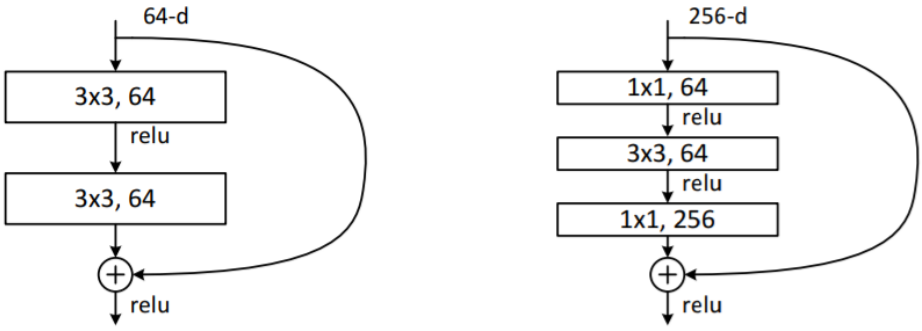
1.3.3. Rezidualni modeli

Suprotno početnim pretpostavkama, s vremenom se pokazalo da veća dubina i složenost modela negativno utječe na njegove generalizacijske sposobnosti, koja je rasla sve do određenog broja slojeva, da bi tada počela opadati.

Kaiming He i ostali su u svom radu [5] pokušali izbjeći navedeni problem tako što su dodali rezidualne (preskočne) veze u konvolucijske slojeve. Rezidualna veza ne provodi nikakvu transformaciju nad ulaznim podatcima (funkcija identiteta), te povezuje ulaz prvog konvolucijskog sloja s izlazom posljednjeg konvolucijskog sloja, prije prolaska kroz aktivacijsku funkciju u bloku[3]. Zbog ovakvih veza, gradijent se prilikom učenja može propagirati u ranije slojeve mreže što uvelike olakšava učenje, a istovremeno, računalna složenost ne raste, kao ni broj parametara. Zbog toga možemo imati veću dubinu modela. Za izrazito duboke modele koristi se rezidualna jedinica s uskim grlom koje smanjuje broj parametara i računsku složenost.

Slika 1.5 prikazuje primjer osnovne rezidualne jedinice, te jedinice s uskim grlom.

[3] Postoje i predaktivacijski rezidualni modeli [23].



Slika 1.5. Osnovna rezidualna jedinica (s lijeve strane) i rezidualna jedinica s uskim grlom (s desne strane). Jedinica s uskim grlom (eng. *bottleneck*) obično se koristi za modele s 50 ili više slojeva. Slika je preuzeta iz [5].

Otkriće rezidualnih modela iznimno je značajno jer se pokazuje da su takvi modeli mnogo uspješniji od ostalih modela. Rezidualne arhitekture se i danas koriste za rješavanje mnogih problema računalnog vida.

1.4. Učenje modela

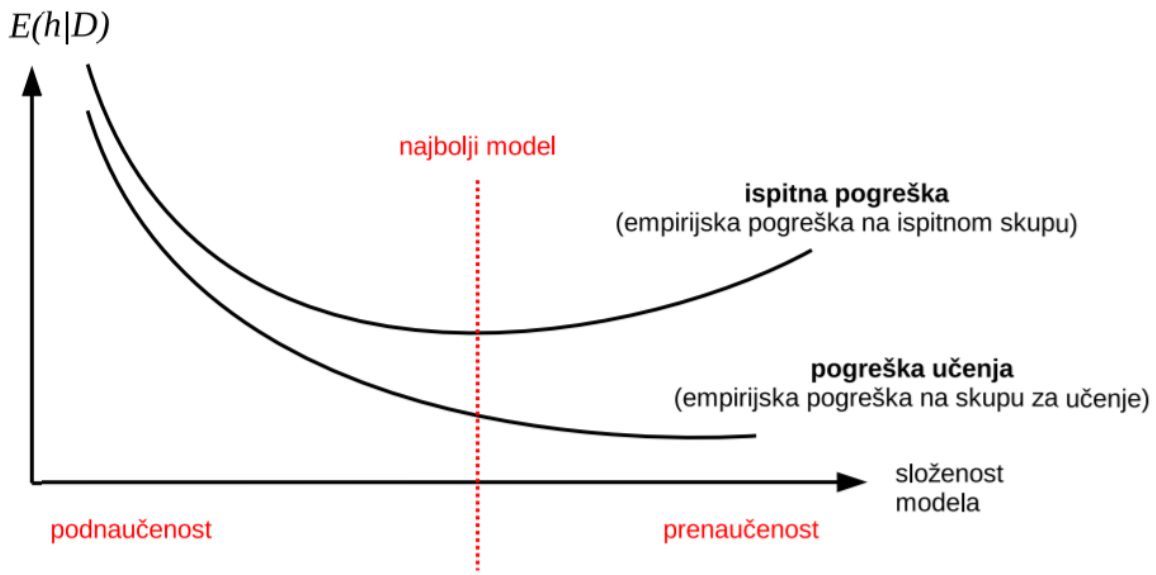
Najvažnije svojstvo neuronskih mreža jest njihova sposobnost generalizacije koju stječu učenjem iz podataka. Postoje razne vrste učenja: nadzirano učenje, nenadzirano učenje, polunadzirano učenje i podržano učenje. Iako se ovaj rad bavi polunadziranim učenjem (koje je opisano u daljnjem poglavlju), najprije je bitno razumjeti koncepte nadziranog učenja.

Nadzirano učenje (eng. *supervised learning*) podrazumijeva da su podatci za učenje označeni, te pretpostavlja da postoji veza između ulaznih i izlaznih podataka. Tijekom procesa učenja model nastoji prilagoditi težine svakog sloja kako bi u sljedećem prolazu kroz sve slojeve modela izlaz bio što sličniji očekivanoj vrijednosti. Prilagođavanje, odnosno neprestano ažuriranje težina provodi se optimizacijskim postupkom koji nastoji minimizirati funkciju gubitka (pogrešku) na skupu podataka za učenje.

Nadzirano učenje se provodi tako da se modelu predočavaju podatci za koje su poznate vrijednosti izlaza. Predočavanje jedne mini-grupe nazivamo iteracijom, a jedan ciklus predočavanja svih uzoraka nazivamo epohom učenja. Neuronska mreža obično se trenira u više epoha, sve dok se njena moć generalizacije povećava.

Skup podataka obično dijelimo na skup za treniranje, skup za validaciju i skup za testiranje. Učenje se provodi na skupu za treniranje (obično 60-80% od ukupnog broja podataka) uz povremenu provjeru kvalitete modela na skupu za validaciju. Hiperparamatre koji osiguravaju najbolju performansu na validacijskom skupu koristimo kako bismo naučili konačni model na skupovima za učenje i validaciju. Nakon učenja, na skupu za testiranje se provjerava generalizacijska točnost modela. Najboljim modelom smatramo upravo onaj model koji postiže najmanju generalizacijsku pogrešku.

Slika 1.6 prikazuje odnos između složenosti modela te pogrešaka na skupu za učenje i na skupu za testiranje. Najbolji model je onaj kojem je ispitna pogreška najmanja.



Slika 1.6. Grafički prikaz odnosa između složenosti i pogreške modela. Slika je preuzeta iz [2].

Prenaučenost modela je stanje u kojem model ima vrlo malu empirijsku pogrešku (pogrešku na skupu za treniranje), ali visoku pogrešku generalizacije (pogrešku na skupu za testiranje). Prenaučenost je nešto što svakako želimo izbjeći, jer je glavni cilj dobiti model koji će znati dobro generalizirati. Model može postati prenaučen u raznim situacijama, a jedan od primjera je prevelika složenost. S druge strane, podnaučenost modela je također nepoželjna jer rezultira visokom pogreškom generalizacije, kao i visokom empirijskom pogreškom. Podnaučenost je najčešće rezultat premale složenosti modela. U traženju balansa složenosti modela možemo pronaći model koji će najbolje generalizirati.

1.4.1. Funkcija gubitka

Gubitak nazivamo mjerom pogreške određivanja predikcije. Funkcija gubitka određuje na koji način se mjeri odstupanje između izlaza iz mreže i očekivane vrijednosti. Postoje različite funkcije gubitka, ali one koje se najčešće koriste su srednja kvadratna pogreška za problem regresije, te unakrsna entropija za probleme klasifikacije. S obzirom da se ovaj rad bavi klasifikacijom, u nastavku je opisana samo unakrsna entropija.

Operacija unakrsne entropije definira se uz vjerojatnosne funkcije p i q, na sljedeći način:

$$E(p, q) = -\sum_{x} p(x) \log q(x) \tag{1.11}$$

U slučaju binarne klasifikacije možemo definirati binarnu unakrsnu entropiju kao:

$$E = \frac{-1}{n} \sum_{i}^{n} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \tag{1.12}$$

gdje je n ukupan broj uzoraka, y_i stvarna oznaka klase koja odgovara ulaznom podatku i, a $\hat{y}_i$ procjena izlaza za taj podatak.

Naposljetku, unakrsnu entropiju možemo svesti na negativnu logaritamsku izglednost (eng. *negative log likelihood*) u slučaju kad imamo više razreda, također uz uvjet da izlazna aktivacijska funkcija (najčešće *softmax* funkcija) daje izlaz s razdiobom vjerojatnosti pojedinih razreda [6].

1.4.2. Optimizacijski algoritmi

Neuronske mreže gotovo uvijek optimiramo gradijentnim postupcima prvog reda, iz čega slijedi da je potrebno izračunati gradijent funkcije pogreške. Algoritam kojim se to može napraviti učinkovito naziva se propagacija pogreške unatrag (eng. *Backpropagation*). Optimizacijski algoritmi koriste gradijente dobivene tim postupkom da bi pronašli minimum funkcije pogreške. Najpopularniji i najčešće korišteni optimizacijski algoritmi danas su stohastički gradijentni spust i ADAM.

Stohastički gradijentni spust (eng. *Stochastic gradient descent, SGD*) izvršava ažuriranje parametara u svakom koraku učenja. Za svaki primjer $x^{(i)}$ i njegovu pripadajuću oznaku $y^{(i)}$ izračunavaju se novi parametri po sljedećoj formuli [7]:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{1.13}$$

Pri tome J označava ciljnu funkciju koju želimo minimizirati ažuriranjem parametara u smjeru suprotnom od gradijenata ciljne funkcije $\nabla_\theta J(\theta)$. Stopa učenja η određuje veličinu koraka kojim se krećemo prema minimumu funkcije.

Kako bi se smanjila računska složenost, najčešće se radi korekcija nad manjom grupom od n podataka (eng. *mini-batch*) [7]:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{1.14}$$

Konvergencija je na ovaj način stabilnija jer se smanjuje varijanca osvježavanja parametara. Osim toga, mogu se koristiti optimizirani algoritmi za rješavanje matričnih operacija što ubrzava postupak i smanjuje računsku složenost algoritma. Algoritam stohastičkog gradijentnog spusta prikazan je na slici 1.7.

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Slika 1.7. Pseudokod za stohastički gradijentni spust, preuzet iz [8].

Adam (eng. *Adaptive Moment Estimation*) je također jedan od popularnih optimizatora koji se koriste prilikom traženja minimuma ciljne funkcije pomoću gradijentnog spusta. Algoritam prati eksponencijalni pomični prosjek gradijenta i kvadrata gradijenta uz eksponencijalno zaboravljanje. Korekcija se ne obavlja prema izračunatom već prema uprosječenom gradijentu. Algoritam radi s mnogo parametara, a možemo ga opisati sljedećim formulama:

$$s = \beta_1 s + (1 - \beta_1) dx \tag{1.15}$$

$$r = \beta_2 r + (1 - \beta_2) dx^2 \tag{1.16}$$

Pritom su s i r procjene prvog, odnosno drugog momenta, β_1 i β_2 su stope eksponencijalnog prigušenja, a dx je gradijent. Procjena prvog momenta je srednja vrijednost, a procjena drugog momenta je necentrirana varijanca koja se dobije ako se ne oduzima od srednje vrijednosti prilikom izračuna. Ažuriranje parametara možemo opisati izrazom

$$\theta \leftarrow \theta - \eta \cdot \frac{s}{\sqrt{r} + \delta}, \tag{1.17}$$

pri čemu je δ vrlo mala konstanta reda veličine 10^{-8} koja se koristi za numeričku stabilizaciju.

Jedan od izazova je odrediti pogodnu stopu učenja η. Premalena stopa učenja dovodi do spore konvergencije, dok prevelika stopa učenja može dovesti do divergencije ili do zanemarivanja smjera u kojem bismo mogli naći globalni minimum. Općenito, najbolji rezultati dobivaju se korištenjem različitih metoda koje mijenjaju stopu učenja tijekom samog učenja. Jedna od takvih metoda je **kosinusno kaljenje** (eng. *cosine annealing scheduling*). U toj metodi učenje započinje visokom stopom učenja, da bi se tijekom učenja stopa postupno smanjivala prema sljedećoj funkciji [9]:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \cdot \left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), \tag{1.18}$$

gdje je T_{cur} trenutna epoha učenja, a T_{max} posljednja epoha učenja u kojoj primjenjujemo postupak. η_{max} je početna, odnosno najveća stopa učenja, a η_{min} hiperparametar kojim unaprijed određujemo do koje vrijednosti će stopa učenja padati.

1.4.3. Polunadzirano učenje korištenjem pseudooznaka

Ručno označavanje podataka je skupo i vremenski zahtjevno. Jedna od tehnika kojoj se pribjegava u tim slučajevima je polunadzirano učenje. To je vrsta učenja u kojem su neki podatci označeni, a neki (često puta puno više njih) nisu. Pseudooznake su jednostavno

predikcije koje smo dobili za neoznačene podatke, te ih tretiramo kao da su prave oznake (eng. *ground truth*) [10]:

$$y'_i = \begin{cases} 1, & ako\ je\ i = argmax_{i'} f_{i'}(x) \\ 0, & inače \end{cases} \tag{1.19}$$

Glavna ideja koju predlažu autori izvornog rada [10] je da se mreža najprije uči na označenim i neoznačenim podatcima istovremeno. Za neoznačene podatke, pseudooznake se ponovno izračunavaju nakon svakog ažuriranja parametara.

Ukupan broj označenih i neoznačenih podataka često varira pa je jako bitno održavati ravnotežu prilikom treniranja, zbog čega se ukupna funkcija gubitka računa kao:

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L\left(y'^m_i, f'^m_i\right) \tag{1.20}$$

n predstavlja broj mini-grupa označenih podataka, n' broj mini-grupa neoznačenih podataka, f_i^m je izlaz modela za m-tu grupu, y_i^m oznaka, f'^m_i izlaz za neoznačene podatke, te y'^m_i pseudooznake za neoznačene podatke. $L(y_i^m, f_i^m)$ predstavlja funkciju gubitka (u ovom slučaju gubitak unakrsne entropije) nad označenim podatcima, a $L(y'^m_i, f'^m_i)$ gubitak nad neoznačenim podatcima. Na kraju, $\alpha(t)$ je koeficijent kojim balansiramo nadzirani i nenadzirani gubitak. Taj koeficijent je od iznimne važnosti za performanse modela. Ako je prevelik, može poremetiti treniranje čak i za označene podatke. Suprotno tome, ako je premali, ne koristimo neoznačene podatke za učenje koliko bismo mogli. Pri računanju tog koeficijenta autori rada [10] predlažu sljedeću formulu:

$$\alpha(t) = \begin{cases} 0, & t < T_1 \\ \dfrac{t - T_1}{T_2 - T_1} \cdot \alpha_f, & T_1 \le t < T_2 \\ \alpha_f, & T_2 \le t \end{cases} \tag{1.21}$$

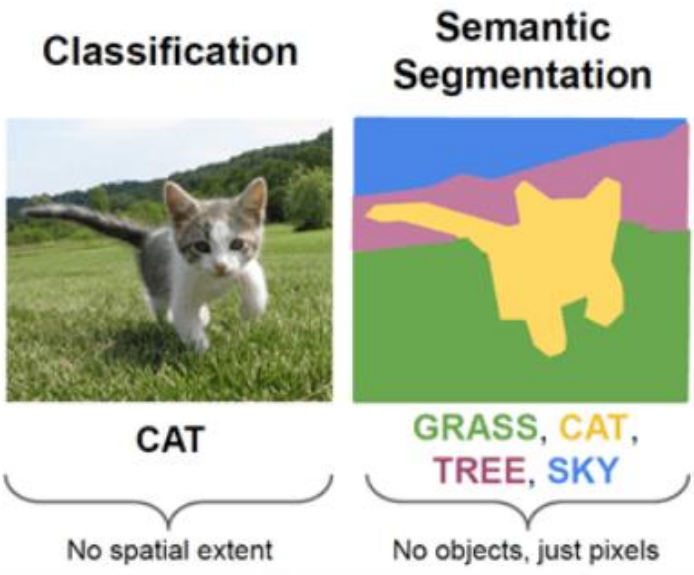
Preporučene vrijednosti parametara su: $\alpha_f = 3, T_1{=}100, T_2{=}600$.

2. Semantička segmentacija

Analiza scene je veliko područje u okviru računalnog vida. Semantička segmentacija bavi se razumijevanjem scene na razini piksela, te ima široku primjenu u autonomnoj vožnji, klasifikaciji područja na temelju satelitskih snimki, robotici te medicinskoj dijagnostici. Semantička segmentacija je metoda čiji je cilj dodjeljivanje semantičke oznake (klase) svakom pikselu na slici.

Postoje razne metode kojima se pokušavalo riješiti problem segmentacije. Neke od tih metoda su promatranje bliskih područja, promjena boje, usporedba piksela i superpiksela [11]. Te metode ispale su iz upotrebe kroz posljednjih desetak godina jer su duboki konvolucijski modeli uvelike nadmašili njihove performanse.

Izlaz iz konvolucijskih modela za rješavanje zadatka semantičke segmentacije naziva se segmentacijska mapa, što je obično matrica s cjelobrojnim vrijednostima, dimenzija jednakih dimenzijama ulaza u model, odnosno slike. Elementi ove mape odgovaraju nekoj od oznaka klasa koja se nalazi na slici. Ove oznake klasa su predefinirane, te različite za svaki problem koji želimo riješiti.



Slika 2.1. Razlika između obične klasifikacije i semantičke segmentacije [12]. Lijeva slika prikazuje primjer ulaza u model, dok desna prikazuje masku s označenim labelama ili izlaz iz naučenog modela koji bi mogao dati tu masku.

Pojedini objekt na slici može biti vrlo malen i zauzimati svega nekoliko piksela, a opet vrlo važan za prepoznavanje (npr. čovjek ili životinja na cesti). Zato je potreban model koji

neće gubiti informacije o takvim objektima. Modeli s ljestvičastim naduzorkovanjem pokazali su se najboljim rješenjem za ovakve probleme zato jer sprječavaju gubitak informacije do kojeg dolazi zbog korištenja rezidualnih blokova. Da bi se ovaj problem spriječio, koristi se ljestvičasto naduzorkovanje [13] koje se ostvaruje uzastopnim pribrajanjem aktivacijskih značajki nižeg i višeg sloja, čije je značajke potrebno bilinearno naduzorkovati, da bi se mapa značajki vratila na početnu rezoluciju.

2.1. Metrike

Postoji više različitih metrika za evaluaciju modela koji rješavaju problem semantičke segmentacije. U nastavku će biti opisane neke od tih metrika, kao i razlozi zašto nam je potrebno više njih.

Popularni alat za računanje raznih metrika prilikom bilo kakvog zadatka klasifikacije je **matrica zabune** (eng. *Confusion matrix*). Matrica zabune je matrica dimenzija $C \times C$, gdje je C broj klasa. Ona svaki klasificirani primjer (u slučaju semantičke segmentacije to bi bio svaki piksel) stavlja u jednu od četiri mogućih kategorija:

- TP (eng. *True Positive*) – točno pozitivno klasificiran primjer
- FP (eng. *False Positive*) – netočno pozitivno klasificiran primjer
- FN (eng. *False Negative*) – netočno negativno klasificiran primjer
- TN (eng. *True Negative*) – točno negativno klasificiran primjer

Prva metrika do koje intuitivno dolazimo jest metrika **točnosti piksela** (eng. *Accuracy*), koja prikazuje udio točno klasificiranih piksela u ukupnom broju piksela:

$$Acc = \frac{TP + TN}{TP + TN + FN + FP} \tag{2.1}$$

U realnom svijetu česta je pojava neuravnoteženosti u zastupljenostima pojedinih klasa. Ova pojava jasno je vidljiva u zadatku semantičke segmentacije za problem analize prirodne scene vozača. Primjerice, pješaci zauzimaju mnogo manje piksela na slici u odnosu na cestu,

nebo, zgrade, ili druge aute, a znamo da je vrlo važno prepoznati pješake na slici. Mjera točnosti piksela u ovom slučaju može biti varljiva, jer model može davati točnost veću od 95%, a da uopće ne prepozna tako ključnu stvar poput pješaka na cesti. Stoga se uvode i neke druge metrike za evaluaciju modela za semantičku segmentaciju.

Mjera **preciznosti** (eng. *Precision*) prikazuje koliko precizno model predviđa podatke kojima je stvarna vrijednost pozitivna. Ovu mjeru pogodno je koristiti kada lažno pozitivne predikcije daju visok trošak. Koristeći prethodno opisane pojmove TP, TN, FP, i FN, preciznost možemo ovako definirati:

$$P = \frac{TP}{TP + FP} \tag{2.2}$$

Mjera **odziva** (eng. *Recall*) daje udio ispravno pozitivnih predikcija u cijelom skupu ispravnih predikcija. Definiramo ga kao:

$$R = \frac{TP}{TP + FN} \tag{2.3}$$

Konačno, možda i najpopularnija metrika za semantičku segmentaciju je **Jaccardov indeks**. Često puta naziva se i **IoU** (eng. *Intersection over Union*). Ova metrika formalno se definira kao omjer presjeka i unije predikcija i oznaka. Obično se računa posebno za svaku klasu, da bi se dobio prosjek **mIoU** (eng. *mean Intersection over Union*). Jaccardov indeks možemo računati na sljedeći način:

$$IoU = \frac{TP}{TP + FN + FP} \tag{2.4}$$

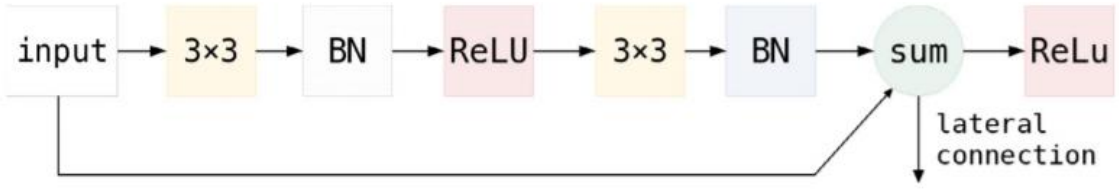
3. Opis korištene metode

U ovom poglavlju opisana je arhitektura korištenog modela. Korištene su dvije inačice modela: model za semantičku segmentaciju, prilagođen za skup podataka CamVid, te model za klasifikaciju slika, prilagođen za skup podataka CIFAR.

Obje inačice modela zasnivaju se na modulu za ekstrakciju značajki koji je predtreniran na skupu podataka ImageNet da bi što bolje iskoristio regularizaciju od prijenosnog učenja. Važno je napomenuti da mreža koristi regularizacijski efekt **normalizacije po grupama** (eng. ***batch normalization***) [27] koja normalizira izlaz prije primjene nelinearne aktivacije u distribuciju sa srednjom vrijednošću 0 i varijancom 1. Ovo rješava problem promjene distribucije podataka po slojevima dubljih modela. Da bi regularizacijski efekt došao do izražaja, važno je da se model trenira na grupama podataka (eng. *Batch learning*).

Kod modela za semantičku segmentaciju, metoda segmentacije se zasniva na tri osnovna gradivna bloka. To su koder za raspoznavanje, dekoder za povećanje mape značajki, i modul za povećanje receptivnog polja. Ovi blokovi detaljnije su opisani u nastavku.

Koder za raspoznavanje. Ovaj rad koristi ResNet-18 kao segmentacijski koder, iako izvorni rad [14] razmatra i MobileNet V2. Za ova oba modela su javno dostupne težine predtrenirane na ImageNetu. ResNet-18 je ovdje dobar odabir jer, zbog svoje umjerene dubine, može raditi u stvarnom vremenu, a rezidualna struktura uvelike olakšava učenje i bez predtreniranih težina. Koder se sastoji od četiri blokova koji produciraju semantički sve bogatije mape značajki (x4, x8, x16, x32). Struktura posljednje rezidualne jedinice bloka kodera prikazana je na slici 3.1.



Slika 3.1. Strukturalni dijagram zadnje rezidualne jedinice bloka kodera. [14]

Dekoder za povećanje dimenzija mape značajki. Mape značajki dobivene iz kodera imaju manju rezoluciju što nam štedi memoriju i vrijeme. Glavna svrha dekodera je povećati rezoluciju značajki sve do rezolucije ulaza, odnosno slike. Koristi se jednostavan dekoder organiziran kao sekvenca modula za povećanje mape značajki, koji su povezani lateralnim

vezama [14]. Takvi moduli imaju dva ulaza, gdje je prvi ulaz mapa značajki niske rezolucije, a drugi ulaz su lateralne značajke dobivene iz ranijih slojeva kodera. Mape značajki niske rezolucije najprije se povećavaju bilinearnom interpolacijom na jednaku rezoluciju kao lateralne značajke. Tada se taj povećani prvi ulaz sumira po elementima s drugim ulazom, da bi se zatim promiješao s konvolucijom 3×3. Lateralne značajke s izlaza zbrajanja po elementima unutar zadnjeg rezidualnog bloka prosljeđuju se na odgovarajuću razinu poduzorkovanja.

Modul za povećanje receptivnog polja. Rad predlaže dva načina kojima se postiže povećanje receptivnog polja, uz moguću predikciju u stvarnom vremenu. To su modul za prostorno piramidalno sažimanje (eng. *spatial pyramid pooling, SPP*) i piramidalna fuzija (eng. *pyramid fusion*). S obzirom da je u ovom radu implementiran i testiran samo jednorezolucijski model (eng. *single scale model*), piramidalna fuzija nije korištena. Modul za prostorno piramidalno sažimanje sakuplja značajke iz kodera na različitim razinama sloja sa sažimanje. Mapu značajki koju smo dobili iz posljednje grupe kodera najprije provodimo kroz sloj BN-ReLU-Conv (1x1), smanjujući semantičku dimenziju za faktor 4. Dobiveni tenzor zatim provlačimo kroz tri razine piramide. Svaka razina provodi tenzor kroz sljedeće slojeve:

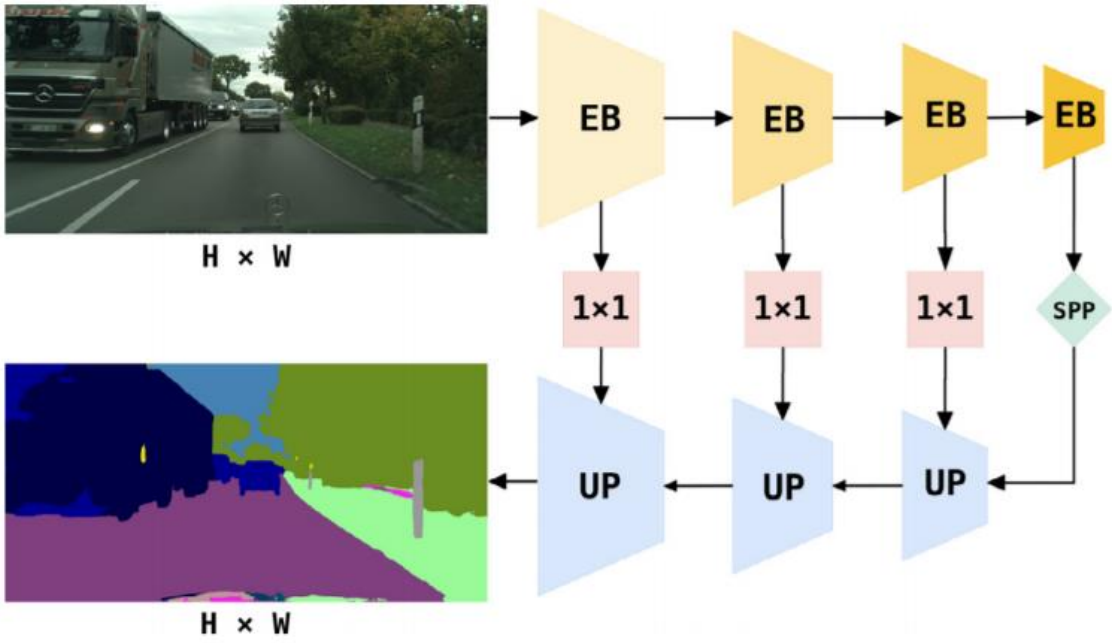
1. Prilagodljivi sloj za sažimanje srednjom vrijednošću (eng. *Adaptive Average Pooling*)[4]. Visina h na izlazu iz ovog sloja bit će jednaka širini w. Te rezolucije bit će različite na svakoj razini piramide (vrijednosti će im biti jednake vrijednostima: 8, 4 i 2).
2. BN-ReLU-Conv (1×1). Ovaj sloj smanjuje semantičku dimenziju mape značajki za faktor $d \,//\, N$, gdje je d jednak semantičkoj dimenziji dobivenoj nakon prvog BN-ReLU-Conv sloja, a N je jednak broju razina piramide.
3. Sloj za bilinearno naduzorkovanje na početnu rezoluciju (rezoluciju prije ulaska u modul za piramidalno sažimanje).

Nadalje, tako dobivene tenzore na svim razinama konkateniramo u jedan tenzor. Na kraju taj tenzor provodimo kroz još jedan sloj BN-ReLU-Conv (1×1) čime dobivamo tenzor jednakih dimenzija kao što je i tenzor kojeg smo dobili nakon prvog BN-ReLU-Conv sloja u modulu za piramidalno sažimanje.

[4] Pri korištenju prilagodljivog sloja za sažimanje (eng. *Adaptive Pooling*) ne trebamo specificirati veličinu koraka ni veličinu jezgre za sažimanje. Umjesto toga specificiramo veličinu izlaza pa se veličine koraka i jezgre automatski prilagode veličini ulaza i željenoj veličini izlaza.

3.1. Jednorezolucijski SwiftNet model

Jednorezolucijski model transformira ulaznu sliku u guste semantičke predikcije putem gore opisanih gradivnih blokova. Strukturalni dijagram modela prikazan je na slici 3.2.



Slika 3.2. Strukturalni dijagram jednorezolucijskog SwiftNet modela [14]. Žuti trapezi prikazuju blokove kodera za raspoznavanje, odnosno konvolucijske grupe, koje se mogu predtrenirati na ImageNet skupu. Zeleni romb prikazuje modul za prostorno piramidalno sažimanje, crveni kvadrati označuju lateralne veze, dok su dekoderi za povećanje rezolucije prikazani plavim trapezima.

Blokovi kodera za raspoznavanje čine četiri konvolucijske grupe. Izlaz iz prve konvolucijske grupe daje značajke rezolucije $H/4 \times W/4$, gdje je H visina, a W širina ulazne slike. Svaka sljedeća grupa prepolavlja rezoluciju značajki, što znači da su dimenzije značajki $H/32 \times W/32$ na izlazu zadnje konvolucijske grupe. Te značajke zatim prolaze kroz modul za piramidalno sažimanje da bi se povećalo receptivno polje. Dobiveni tenzori usmjeravaju se na dekoder koji je asimetričan koderu, jer se koder sastoji od više konvolucija u svakoj konvolucijskoj grupi, za razliku od dekodera, koji sadrži samo jednu konvoluciju po modulu. Kao što je već opisano ranije, moduli za povećanje dimenzija mapi značajki izvršavaju se u tri koraka:

1. Uvećavaju se značajke niske rezolucije na dimenzije lateralnih značajki, postupkom bilinearne interpolacije.
2. Dobivene reprezentacije sumiraju se s lateralnom vezom.
3. Rezultat se miješa konvolucijom 3×3.

3.2. Model prilagođen za klasifikaciju skupa CIFAR

S obzirom na samu prirodu skupa podataka CIFAR, te razliku između kompleksnosti problema semantičke segmentacije i klasifikacije slika, model implementiran za ovaj zadatak ipak je nešto jednostavniji. Za početak, ne koristi se modul za prostorno piramidalno sažimanje, kao što se ne koristi (jer nije ni potrebno) povećanje dimenzionalnosti na početnu rezoluciju slike.

Nakon ekstrakcije značajki se, umjesto modula za povećanje receptivnog polja i posljednjeg BatchNorm-ReLU-Conv sloja, koristi prilagodljivi sloj za sažimanje srednjom vrijednosti izlazne rezolucije 1×1, te potpuno povezani sloj za konačnu klasifikaciju slika. Osim toga, s obzirom da su slike iz skupa CIFAR mnogo manjih dimenzija (32x32) za razliku od slika iz skupa CamVid (960x720), početna konvolucija koristi manju jezgru (3x3, umjesto 7x7), manji korak (1 umjesto 2), te manji *padding* (1 umjesto 3). Konačno, ne koristi se sažimanje nakon prve dvije konvolucije, što dosta poboljšava rezultat.

4. Korištene tehnologije i programska izvedba

4.1. Korištene tehnologije

Programska izvedba cijelog rada, dakle model, učenje i evaluacija modela, kao i enkapsulacija skupova podataka, ostvarena je u programskom jeziku Python. Korištene su biblioteke *NumPy* za efikasno baratanje s višedimenzionalnim poljima, zatim *PIL* za procesiranje slika, te *PyTorch* za implementaciju dubokih modela, te za duboko učenje.

Nadalje, za baratanje skupovima podataka, bili su mi potrebni moduli `Dataset` i `DataLoader`, a važno je spomenuti i da sam koristio dostupne metode optimizacije (optimizatori *ADAM* i *SGD*, uz kosinusno kaljenje) i funkcije gubitka (`CrossEntropyLoss`) iz *PyTorcha*. S obzirom da izračun matrice zabune preko Cythona nije bio podržan, morao sam implementirati vlastite metode u Pythonu, što je prilično usporilo sami postupak evaluacije.

4.2. Programska izvedba

Što se tiče programske izvedbe samog modela, bilo je potrebno prilagoditi originalni programski kod, posebno za klasifikaciju skupa CIFAR. Napisana je metoda koja razdvaja skup podataka na označeni i umjetno neoznačeni skup zato da bi se mogla testirati metoda s polunadziranim učenjem. Slike iz skupa podataka CIFAR-10 preuzete su na uobičajeni način, preko biblioteke *torchvision*. Za učenje se koristi poznata i prokušana tehnika kojom se ti skupovi podataka enkapsuliraju u razred `DataLoader`, dok se za optimizaciju parametara koriste metode razreda iz modula `torch.nn` i `torch.optim`. To su metode `zero_grad()` za poništavanje gradijenata (u svakom koraku trebamo posebno akumulirati gradijente), `backward()` za izračunavanje gradijenata, te `step()` za ažuriranje parametara koristeći izračunate gradijente. Evaluacija se provodi u kontekstu `torch.no_grad()` što znači da se svim parametrima računskog grafa postavlja vrijednost varijable `requires_grad` u `False` [15]. U tom kontekstu se deaktivira PyTorch-ev *autograd engine* za računanje gradijenata. Ovo je važno jer oslobađa dosta memorije (možemo koristiti veće grupe za evaluaciju) te ubrzava postupak evaluacije s obzirom da nam gradijenti tada nisu potrebni. Nakon svake epohe učenja, novo-ažurirani model se evaluira, te sprema stanje cijelog

modela (`state_dict()`) ukoliko je novi rezultat bolji od trenutnog najboljeg. Nakon svake iteracije polunadziranog postupka, model će preuzeti stanje parametara koje daje najbolji rezultat. Bilo je važno apstrahirati klasu za skup podataka CamVid na način da je moguće stvoriti objekt koji se sastoji samo od slika, ne i od oznaka. Takav objekt koristit će se za generacije pseudooznaka. Tijekom generacije pseudooznaka, iscrtat će se proizvoljan broj pseudooznaka koje je generirao trenutni najbolji model.

4.3. Polunadzirana metoda

Polunadzirana metoda koja se koristi preuzeta je iz rada [18]. Postupak je sljedeći:

1. Model učitelja uči na označenom dijelu skupa podataka.
2. Model učitelja generira grube pseudooznake na neoznačenom skupu podataka.
3. Novi model studenta uči na neoznačenom skupu koristeći pseudooznake kao točne klase.
4. Model studenta se fino podešava na označenom dijelu skupa podataka.
5. Proizvoljno mnogo puta se postupak ponavlja od koraka 2, na način da model studenta iz koraka 4 postaje model učitelja u koraku 2.

Algoritam 4.1 prikazuje polunadzirani postupak napisan u Pythonu. Svi detalji oko implementacije modela te baratanja podatcima su skriveni.

Algoritam 4.1. Polunadzirana metoda za semantičku segmentaciju temeljena na pseudooznakama.

Ulazi:

- broj iteracija polunadziranog postupka: *N_ITER*,
- brojevi epoha za nadzirano/nenadzirano treniranje: *sup_epochs, unsup_epochs*
- broj epoha koje moraju proći da bi se model evaluirao prilikom treniranja: *eval_each*,
- putanja u koju se sprema najbolji model: *CHECKPOINT_PATH*,
- putanja u koju se spremaju pseudooznake: *PSEUDO_PATH*,

- skupovi podataka enkapsulirani u razrede `DataLoader`: *sup_dataloaders*, *unsup_loader*
- model za semantičku segmentaciju: *SemsegModel*,
- klasa koja enkapsulira gubitak unakrsne entropije: *SemsegCrossEntropy*

```
1  for i in range(1, N_ITER + 1):
2      teacher_network = SemsegModel(…)
3      teacher_network.criterion = SemsegCrossEntropy(…)
4      teacher_network.load_state_dict(torch.load(CHECKPOINT_PATH))
5      generate_pseudo(teacher_network, unsup_loader, iter=i,
                       save=PSEUDO_PATH)
6      pseudo_loader = create_pseudoloader(…)
7      student_network = SemsegModel(…)
8      student_network.criterion = SemsegCrossEntropy(…)
9      student_optim = optim.Adam(…)
10     scheduler = CosineAnnealingLR(student_optim, …)
11     pseudo_train(student_network, unsup_epochs, pseudo_loader,
                    student_optim)
12     train_and_evaluate(student_network, sup_epochs,
                          sup_dataloaders, student_optim, scheduler,
                          eval_each, iter=i, save=CHECKPOINT_PATH)
```

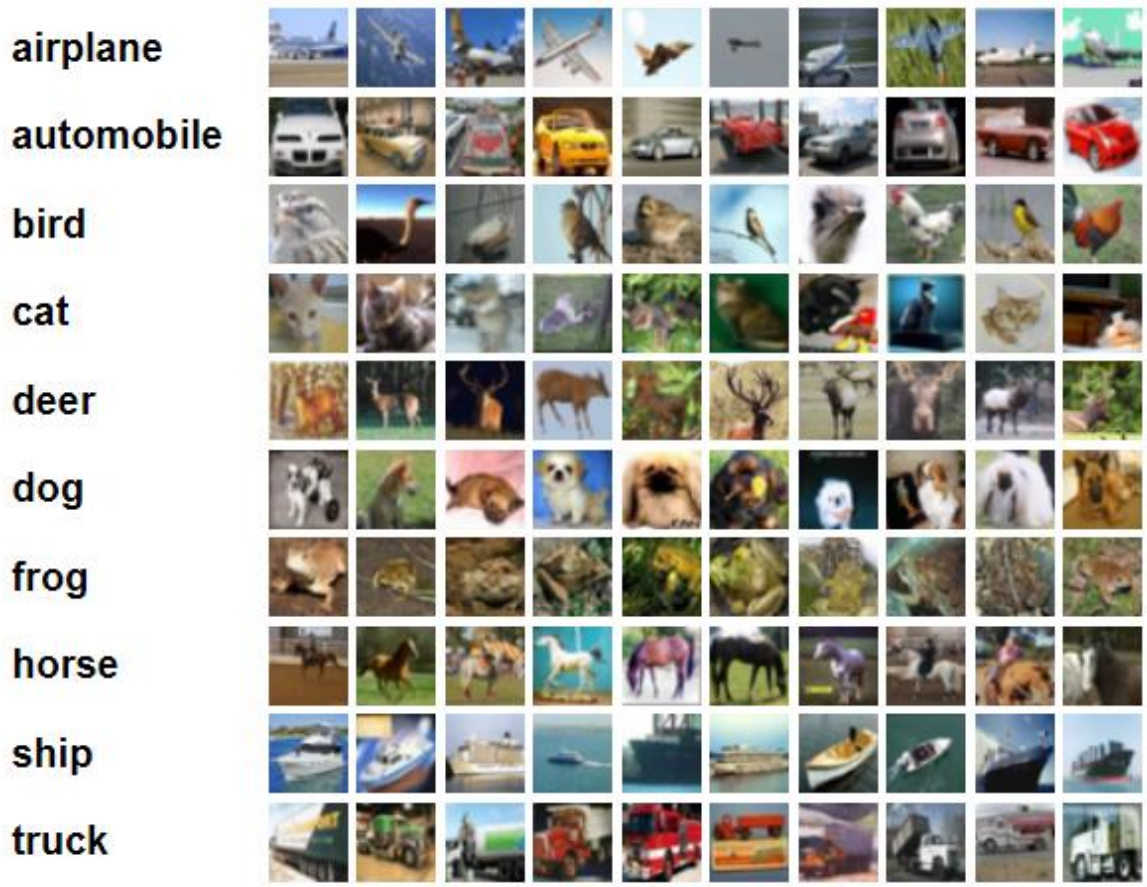
Riječi koje predstavljaju ulaze napisane su kosim (*italic*) stilom. Riječi koje su podebljane predstavljaju ključne korake polunadzirane metode:

- Linija 2: inicijaliziramo model učitelja.
- Linije 5 i 6: generiramo pseudooznake na neoznačenim podatcima pomoću modela učitelja.
- Linija 7: inicijaliziramo novi model učenika te ga treniramo na pseudooznakama u liniji 11.
- Linija 12: fino podešavamo model učenika na označenim podatcima, te ga evaluiramo (na validacijskom skupu tijekom treniranja, i na skupu za testiranje nakon treniranja). Najbolji model spremamo da bismo mogli njegovo stanje vratiti u sljedećoj iteraciji (linija 4).

5. Skupovi podataka

5.1. CIFAR-10

Skup podataka CIFAR-10 (Canadian Institute For Advanced Research) je kolekcija od ukupno 60.000 ručno označenih slika u boji, rezolucije 32x32. Skup podataka je vrlo popularan te se često koristi za strojno učenje te isprobavanje različitih modela koji rješavaju zadatke računalnog vida. Skup čine slike od 10 jednako raspoređenih klasa (po 6000 slika svake klase). Klase su sljedeće: avion, auto, ptica, mačka, jelen, pas, žaba, konj, brod i kamion. 50.000 slika skupa čine skup za treniranje, dok ostalih 10.000 čine skup za testiranje. Ne postoje nikakva preklapanja između pojedinih klasa. Slika 5.1 prikazuje primjere slika iz svake klase.



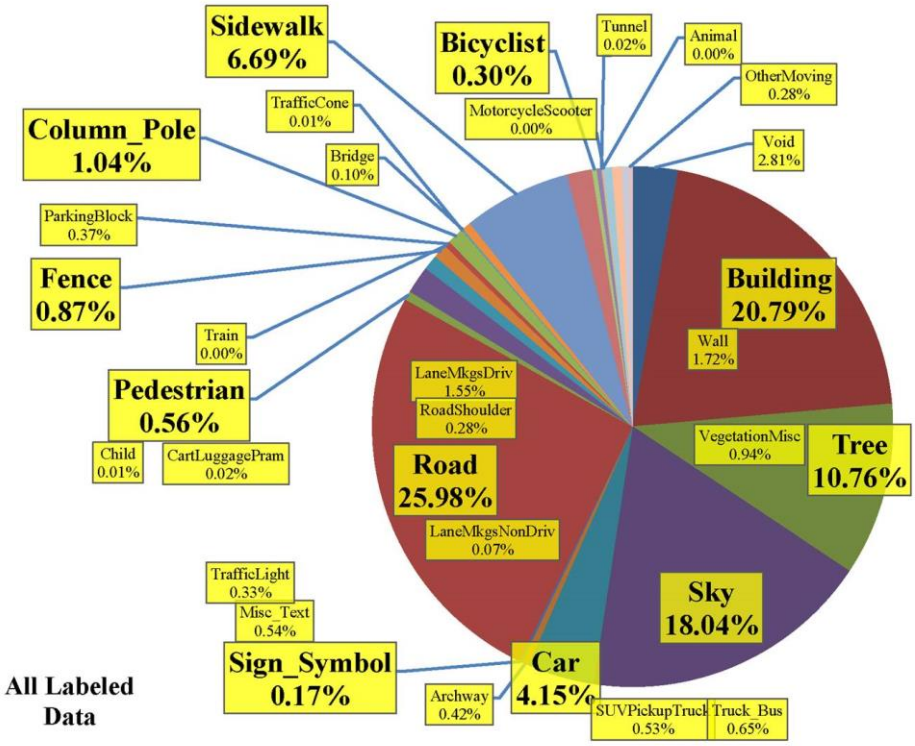
Slika 5.1. Primjeri slika iz skupa podataka CIFAR-10, s označenim klasama. Slika je preuzeta iz [16].

5.2. CamVid

Cambridge-driving Labeled Video Database (CamVid) prva je kolekcija videa s označenim pikselima. Videozapisi su snimljeni iz perspektive vozača tijekom vožnje. Skup sadrži sveukupno preko 10 minuta videozapisa visoke kvalitete. Slike su preuzete iz videa te ručno označene, tako da je svakom pikselu na slici pridijeljena jedna od 32 semantičke labele. Ukupno je označena 701 slika. Taj skup podijeljen je na skup za treniranje koji se sastoji od 367 slika, skup za validaciju u kojem je 101 slika, te skup za testiranje kojeg čini 233 slike. Na slici 5.2 prikazan je jedan par slike sa svojom ručno označenom, obojanom oznakom.

Slika 5.2. Jedan par slike iz skupa podataka CamVid s obojanom oznakom [17]

Rezolucija slika u izvornom formatu je 960 x 720 piksela. Popularno je koristiti samo 11 razreda prilikom učenja, svi ostali razredi se ignoriraju (oznaka *void*). Razredi koji se koriste su: nebo, cesta, zgrada, stablo, auto, biciklist, pješak, pločnik i kolnički trak. Slika 5.3 prikazuje kružni graf s postotcima ukupne količine označenih piksela po razredima.



Slika 5.3. Postotci ukupne količine piksela po razredima. Vidljivo je da 11 razreda koji se koriste čine većinu relevantnih u odnosu na sve razrede. Slika je preuzeta iz [17].

6. Rezultati

Provedeni su eksperimenti treninga i evaluacije na skupovima podataka navedenima u prethodnom poglavlju. U ovom poglavlju prikazani su rezultati postignuti na skupovima za testiranje, za skupove podataka CIFAR-10 i CamVid. Za CIFAR-10, jedina metrika koja se koristi pri evaluaciji je točnost (eng. *accuracy*). Što se tiče CamVid-a, uvedene su metrike koje se najčešće koriste u zadatcima semantičke segmentacije.

6.1. Detalji treniranja

Model je treniran i testiran na grafičkoj procesnoj jedinici vlastitog računala, NVIDIA GeForce GTX 1650 Ti. Postoji nekoliko razlika u detaljima treniranja modela na skupu podataka CIFAR-10 i CamVid.

Model treniran na skupu podataka CIFAR-10 koristi veličinu grupe 128 za treniranje, te veličinu grupe 100 za evaluaciju. Na skupu za učenje su provedene transformacije slučajnog podrezivanja (eng. *Random Cropping*) na originalnu veličinu slika (32x32) i slučajnog horizontalnog zrcaljenja (eng. *Random Horizontal Flip*). Nadalje, koristi se optimizacijski algoritam SGD uz početnu stopu učenja 0.01 koja se smanjuje kosinusnim kaljenjem do stope učenja $1 \cdot 10^{-4}$ u posljednjoj epohi. Propadanje težina postavljeno je na $5 \cdot 10^{-4}$. Što se tiče polunadzirane metode, prvi model učitelja treniran je na 120 epoha, prvi model studenta na 60 epoha, te konačno fino podešavan na 150 epoha.

Model treniran na skupu podataka CamVid koristi veličinu grupe 14, te Adam optimizator umjesto SGD-a. Detalji treniranja preuzeti su iz originalnog rada [14]. Početna stopa učenja iznosi $4 \cdot 10^{-3}$, a krajnja je postavljena na $1 \cdot 10^{-5}$.

6.2. Rezultati na skupu CIFAR-10

Proveo sam četiri eksperimenta na skupu CIFAR-10. Najprije sam testirao model bez polunadzirane metode u 150 epoha. Rezultat tog prvog eksperimenta prikazan je u usporedbi s mnogim drugim modelima, u tablici 6.1. Zatim sam isprobao polunadziranu metodu

odvajanjem podataka iz skupa za učenje na označeni i umjetno neoznačeni skup, po uzoru na [26]. Metodu sam proveo razdvajanjem na skupove od n označenih, te $50.000 - n$ neoznačenih slika. Za n su uzete vrijednosti 250, 1000 i 4000. Jedina metrika koja se koristi pri evaluaciji je klasifikacijska točnost na skupu za testiranje.

Tablica 6.1 Usporedba točnosti raznih modela na skupu podataka CIFAR-10. Rezultati za sve ostale modele preuzeti su iz [19].

Model	**Točnost (%)**
VGG16	92.64
ResNet18	93.02
ResNet50	93.62
ResNet101	93.75
RegNetX_200MF	94.24
RegNetY_400MF	94.29
MobileNetV2	94.43
ResNeXt29 (32x4d)	94.73
SimpleDLA	94.89
DenseNet121	95.04
PreActResNet18	95.11
DPN92	95.16
DLA	95.47
Modif. RN-18 (moj)	**94.43**

Što se tiče polunadziranih eksperimenata, oni su provedeni u 3 iteracije. U svakoj iteraciji, model je učio na pseudooznakama kroz 60 epoha, a zatim je fino podešavan na označenim primjerima kroz 120 epoha. Tablica 6.2 prikazuje najbolje rezultate u svakoj iteraciji, za sva tri polunadzirana eksperimenta.

Tablica 6.2. Rezultati polunadziranih eksperimenata za CIFAR-10. Prvi broj u prvom redu predstavlja broj slučajno odabranih označenih slika.

250 / 50000		1000 / 50000		4000 / 50000	
Iteracija	Točnost (%)	Iteracija	Točnost (%)	Iteracija	Točnost (%)
0	35.53	0	52.89	0	76.05
1	36.69	1	54.91	1	79.34
2	37.12	2	55.79	2	79.64
3	**37.5**	**3**	**56.5**	**3**	**80.71**

6.3. Rezultati na skupu CamVid

Kao i za skup CIFAR-10, jednorezolucijska inačica SwiftNetRN-18 modela najprije je isprobana bez dodatnih slika za polunadziranu metodu, kroz 400 epoha, po uzoru na [14]. Treniranje je provedeno na kombiniranom skupu podskupova za učenje i za validaciju (*trainval*). Tablica 6.3 prikazuje IoU točnosti po klasama na skupovima za validaciju i testiranje, dok tablica 6.4 prikazuje srednje vrijednosti Jaccardova indeksa, odziva i preciznosti, te ukupnu točnost klasificiranih piksela. Slika 6.1 prikazuje tri slike iz skupa CamVid uz njihove točne oznake te izlaze koje je model dao.

Tablica 6.3. Prikazane su IoU točnosti po klasama na skupovima za validaciju i testiranje. Najmanja je točnost za klase stupa, znakova, pješaka i ograde, a najveća za klase ceste, neba i zgrada.

KLASA	Valid IoU (%)	Test IoU (%)
Building	96.81	85.85
Tree	95.18	76.62
Sky	95.08	93.03
Car	94.09	86.72
Sign	73.77	15.43
Road	98.32	94.06
Pedestrian	77.94	54.76
Fence	90.21	45.66
Column pole	53.94	30.40
Sidewalk	94.66	84.51
Bicyclist	90.81	39.96

Tablica 6.4. Prikaz rezultata po mjerama točnosti na evaluacijskim skupovima podataka za CamVid.

METRIKA	Valid	Test
IoU mean class accuracy	87.35%	64.27%
Mean class recall	91.82%	73.39%
Mean class precision	93.67%	79.04%
Pixel accuracy	97.80%	92.42%

Slika 6.1. U prvom stupcu prikazane su originalne slike iz evaluacijskih skupova podataka iz skupa CamVid. Drugi stupac prikazuje točne oznake, a zadnji stupac prikazuje izlaze modela. Napomena: izlazi modela zanemaruju razred *void* koji je u točnim oznakama prikazan žutom bojom. Ovo je razlog zbog kojeg je distribucija boja nešto različita.

6.3.1. Polunadzirani eksperiment za CamVid

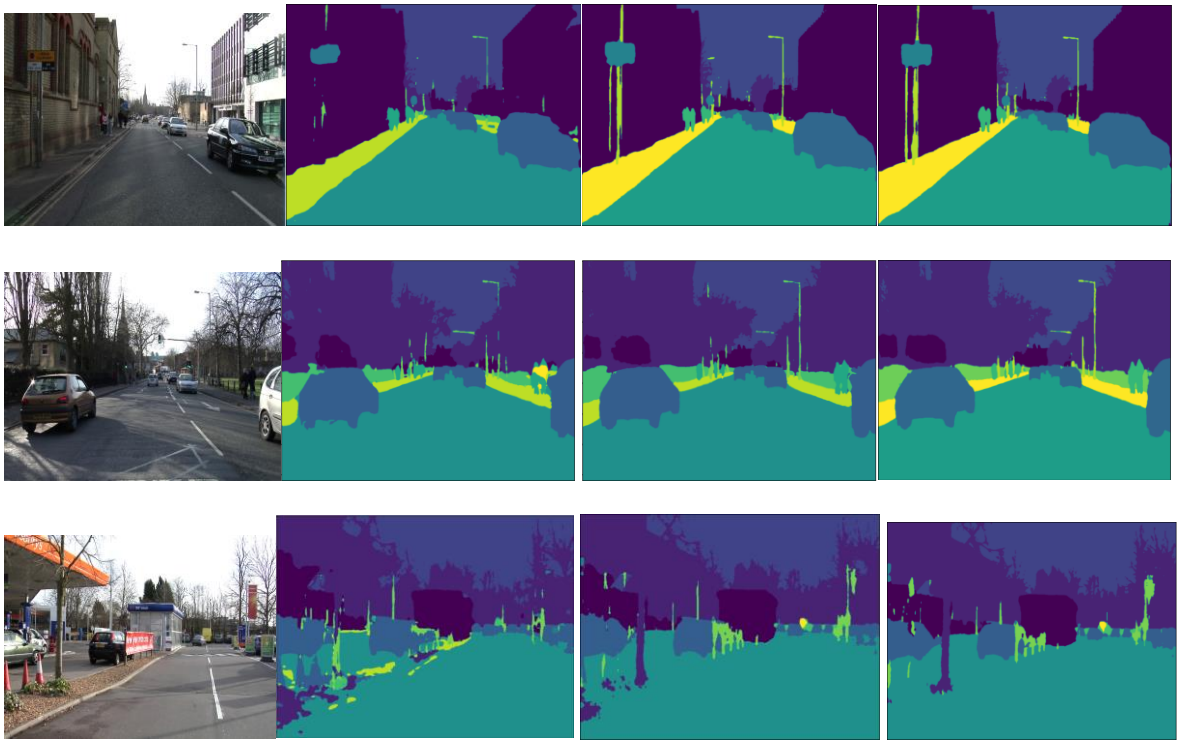
Proveo sam polunadziranu metodu opisanu u prethodnom poglavlju, kroz tri iteracije. Za neoznačene podatke uzeo sam ukupno 762 jednoliko raspoređene slike iz sekvenci 06R0 i 16E5. Za model učitelja u prvoj iteraciji preuzete su težine iz modela dobivene prethodno opisanim nadziranim postupkom. Model studenta učio je na pseudooznakama kroz 100 epoha, a fino se podešavao nad označenim podatcima kroz 200 epoha. Koristio sam optimizator Adam uz početnu stopu učenja jednaku $1 \cdot 10^{-4}$. Po uzoru na [18], početna stopa za polunadzirano učenje je manja nego za inicijalno učenje na označenom skupu podataka. Propadanje težina se ne koristi pri polunadziranom učenju. Tablica 6.5 prikazuje dobivene vrijednosti Jaccardova indeksa po klasama, kroz sve tri iteracije. Posljednja četiri retka prikazuju srednju vrijednost Jaccardova indeksa, srednji odziv i srednju preciznost po klasama, te točnost piksela na skupu za testiranje nakon svake iteracije.

Tablica 6.5. Prikaz rezultata po klasama na skupu za testiranje, nakon svake iteracije polunadziranog postupka.

KLASA	**Iteracija 1** **IoU (%)**	**Iteracija 2** **IoU (%)**	**Iteracija 3** **IoU (%)**
Building	88.18	88.75	88.76
Tree	79.62	79.98	79.31
Sky	93.00	93.16	93.26
Car	92.24	91.80	93.17
Sign	37.69	36.15	40.86
Road	94.79	94.71	95.25
Pedestrian	69.35	68.32	67.71
Fence	44.73	53.24	59.17
Column pole	37.25	36.99	38.21
Sidewalk	85.44	85.55	86.49
Bicyclist	65.30	64.09	67.37

meanIoU	**71.60**	**72.07**	**73.60**
Mean class recall	**78.36**	**78.93**	**81.05**
Mean class precision	**87.22**	**87.64**	**87.15**
Pixel accuracy	**93.52**	**93.69**	**93.86**

Naposljetku, slika 6.2 prikazuje originalne slike iz neoznačenog skupa podataka, te njihove pseudooznake koje je model generirao u različitim iteracijama polunadziranog postupka.



Slika 6.2. Prvi stupac s lijeve strane prikazuje slike iz neoznačenog skupa podataka. Stupac nakon prikazuje pseudooznake generirane nakon nadziranog postupka. Sljedeći stupci prikazuju pseudooznake generirane nakon prve i treće iteracije polunadziranog postupka učenja (uz fino podešavanje na nadziranom skupu). Vidljiv je značajan napredak, posebno između prvog i drugog stupca. Razlika se najbolje vidi u jasnijim obrisima stupova na prvoj slici, glađim rubovima automobila na drugoj slici, te preciznije prikazanoj ogradi na trećoj slici.

7. Zaključak

U ovome radu najprije su opisane osnove strojnog učenja, dubokog učenja, te svojstva konvolucijskih neuronskih mreža. Također, ukratko su opisane osnove polunadziranog učenja temeljenog na pseudooznakama, kao i zadatak semantičke segmentacije, te razne metrike koje se koriste za evaluaciju modela koji rješavaju taj zadatak.

Implementiran je jednorezolucijski SwiftNet model koji koristi ResNet-18 kao segmentacijski koder. Nakon kodera slijedi dekoder za povećanje dimenzija mape značajki pomoću metode bilinearne interpolacije. Naposljetku, model koristi modul za prostorno piramidalno sažimanje kojim se povećava receptivno polje. Model je implementiran u programskom jeziku Python, korištenjem biblioteke PyTorch.

Iterativna polunadzirana metoda uključuje inicijalno učenje na nadziranom skupu podataka. Nakon toga, model generira grube pseudooznake za neoznačeni skup podataka. Zatim se inicijalizira novi model koji najprije uči na neoznačenom skupu podataka koristeći dobivene pseudooznake. Na kraju slijedi fino podešavanje na označenom skupu podataka. Ovaj postupak ponavlja se proizvoljno mnogo puta (tri puta u slučaju ovog rada).

Metoda je najprije testirana na jednostavnijem skupu podataka CIFAR-10, uz jednostavniji model koji se također bazira na ResNet-18 arhitekturi. Metoda je isprobana kroz 3 iteracije, sa 250, 1000, 4000, te 50000 označenih podataka, te postiže zadovoljavajuće rezultate.

Naposljetku, metoda je isprobana i na skupu podataka CamVid, koristeći jednorezolucijski SwiftNet model, te 1370 dodatnih, neoznačenih slika. Korištenjem polunadzirane metode ostvaren je značajan napredak u rezultatima, kad ih se usporedi s rezultatima koji se dobiju običnim nadziranim postupkom.

Zbog prilično spore evaluacije koja ne koristi Cython, i samog treniranja modela, validacija svih hiperparametara bila je prilično izazovna. Bilo bi uputno isprobati različite brojeve epoha za polunadzirano i nadzirano učenje, kao i drugačije početne stope učenja, te različite vrijednosti hiperparametara za promjenu stope učenja.

Literatura

[1] Alpaydin, E., *Introduction to Machine Learning*, 4. izdanje. MIT Press, 2014.

[2] Bojana Dalbelo Bašić, Jan Šnajder. *Strojno učenje*. FER, 2020.

[3] Bojana Dalbelo Bašic, Marko Čupić, Jan Šnajder. *Umjetne neuronske mreže*. 2008.

[4] Josip Krapac, Siniša Šegvić. *Duboko učenje, konvolucijski modeli*. 2020.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. *Deep residual learning for image recognition*. CVPR, 2015. Poveznica: https://arxiv.org/pdf/1512.03385.pdf

[6] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, 2006.

[7] Ruder, S., *An Overview of gradient descent optimization algorithms* (2016, siječanj). Poveznica: https://ruder.io/optimizing-gradient-descent/

[8] Čupić, M. *Duboko učenje, optimizacija parametara modela*. 2019.

[9] Torch Contributors, *CosineAnnealingLR*, 2019. Poveznica: https://pytorch.org/docs/master/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html

[10] D. Lee, *Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks*. ICMLW, 2013. Poveznica: https://www.semanticscholar.org/paper/Pseudo-Label-%3A-The-Simple-and-Efficient-Learning-Lee/798d9840d2439a0e5d47bcf5d164aa46d5e7dc26

[11] Tingwu Wang. *Semantic Segmentation*. Poveznica: https://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf

[12] Kurtanović, J. *Deep Learning – Semantic Segmentation*. 2021. Poveznica: https://serengetitech.com/tech/deep-learning-semantic-segmentation/

[13] Krešo, I., Krapac, J., Šegvić, S. *Efficient Ladder-style DenseNets for Semantic Segmentation of Large Images*. IEEE T-ITS, 2019. Poveznica: https://arxiv.org/abs/1905.05661

[14] M. Oršić, I.Krešo, P. Bevandić, S. Šegvić, *In Defense of Pre-trained ImageNet Architectures for Real-time Semantic Segmentation of Road-driving Images*. CVPR, 2019. Poveznica: https://openaccess.thecvf.com/content_CVPR_2019/html/Orsic_In_Defense_of_Pre-Trained_ImageNet_Architectures_for_Real-Time_Semantic_Segmentation_CVPR_2019_paper.html

[15] Torch Contributors, *NO_GRAD*, 2019. Poveznica: https://pytorch.org/docs/stable/generated/torch.no_grad.html

[16] Krizhevsky, A., *The CIFAR-10 dataset*, 2009. Poveznica: https://www.cs.toronto.edu/~kriz/cifar.html

[17] Brostow, Shotton, Fauqueur, Cipolla, *Segmentation and Recognition Using Structure from Motion Point Clouds*, ECCV 2008, Poveznica: http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

[18] Chen L., et al. *Naive-Student: Leveraging Semi-Supervised Learning in Video Sequences for Urban Scene Segmentation*. ECCV 2020. Poveznica: https://arxiv.org/pdf/2005.10266.pdf

[19] Kuangliu, *Train CIFAR10 with PyTorch*, 2021. Poveznica: https://github.com/kuangliu/pytorch-cifar

[20] Krivošić, M. *Vrednovanje modela SwiftNet za semantičku segmentaciju*. Diplomski rad. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2020.

[21] Krizhevsky et al. *ImageNet Classification with Deep Convolutional Neural Networks*. NIPS, 2012. Poveznica: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[22] Ramachandran et al. *Swish: a Self-Gated Activation Function*, 2017. Poveznica: https://arxiv.org/pdf/1710.05941v1.pdf?source=post_page

[23] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. ECCV, 2016. Poveznica: https://arxiv.org/abs/1603.05027

[24] Samuel Rota Bulò et al. *In-Place Activated BatchNorm for Memory-Optimized Training of DNNs*. CVPR, 2018. Poveznica: https://arxiv.org/pdf/1712.02616.pdf

[25] Song, Meng, Ermon, *MintNet: Building Invertible Neural Networks with Masked Convolutions*. NeurIPS, 2019. Poveznica: https://arxiv.org/pdf/1907.07945.pdf

[26] Grubišić, Oršić, Šegvić, *A baseline for semi-supervised learning of efficient semantic segmentation models*. CoRR, 2021. Poveznica: https://arxiv.org/pdf/2106.07075.pdf

[27] Ioffe, Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ICML, 2015. Poveznica: https://arxiv.org/pdf/1502.03167.pdf

Polunadzirana semantička segmentacija utemeljena na pseudooznačavanju

Sažetak

Semantička segmentacija prirodnih scena važan je zadatak računalnog vida s mnogim zanimljivim primjenama. Jako dobre rezultate na tom zadatku postižu konvolucijski modeli s lakom klasifikacijskom osnovom i ljestvičastim naduzorkovanjem. Ovaj rad bavi se polunadziranim učenjem takvih modela, što znači da model uči na grupama koje se sastoje od označenih i neoznačenih slika. Koristi se polunadzirana metoda pseudooznačavanja neoznačenih slika, najprije na manjem i jednostavnijem skupu podataka, a zatim na nešto kompleksnijem. Provedeno je mnogo iscrpnih eksperimenata koji su pokazali da metoda doista daje bolje rezultate nego uobičajeni postupak nadziranog učenja.

Ključne riječi: računalni vid, konvolucijski modeli, ljestvičasto naduzorkovanje, polunadzirano učenje, semantička segmentacija, pseudooznake, CIFAR-10, CamVid

Semi-supervised semantic segmentation based on pseudolabelling

Abstract

The segmentation of natural scenes is an important task of computer vision with many interesting applications. Very good results on this task are achieved by convolutional models with a light-weight general purpose architecture as the main recognition engine, and upsampling with lateral connections. This paper deals with the semi-supervised learning of such models, which means that the model learns in groups consisting of both labeled and unlabeled images. A semi-supervised method of pseudolabelling is used, first on a smaller and simpler data set, and then on a somewhat more complex one. Many exhaustive experiments have been conducted which have shown that the method really gives better results than the usual supervised learning procedure.

Keywords: computer vision, convolutional neural networks, ladder-style upsampling, semi-supervised learning, pseudolabelling, semantic segmentation, CIFAR-10, CamVid