Hvala mentoru prof. dr. sc. Siniši Šegviću na strpljenju, pristupačnosti i pomoći pri izradi ovog rada.

SADRŽAJ

1. Uvod

Računalni vid danas je široko prepoznato područje s brojnim primjenama. Prije pojave računalnog vida, bilo je teško ili nemoguće automatizirati određene poslove koji zahtijevaju ljudsku intuiciju i percepciju. Primjeri takvih poslova su vožnja automobila, analiza medicinskih snimki, prepoznavanje neispravne robe na proizvodnoj traci... Razvitak računalnog vida uvelike je olakšao takve poslove te otvorio nove tehnološke mogućnosti.

Semantička segmentacija vrlo je bitan dio računalnog vida, a zadaća joj je prepoznati dijelove slike različitog značenja te ih svrstati u klase. U ovom radu rješavat ću problem semantičke segmentacije zgrada u satelitskim snimkama, odnosno određivanja dijelova slike na kojima se nalaze zgrade. To može biti korisno za procjenu naseljenosti, stupnja urbanizacije ili praćenje izgradnje novih zgrada na nekom području. Kao arhitekturu modela uzet ću SwiftNet koji radi učinkovitu semantičku segmentaciju pomoću piramidalne fuzije. Imajući na umu potrebu brzog raspoznavanja objekata u slikama iz vožnje, taj je model napravljen za semantičku segmentaciju u stvarnom vremenu, a pritom zadržava dobru generalizacijsku moć.

Neki prijašnji radovi koji se bave bliskim temama su [12] i [2].

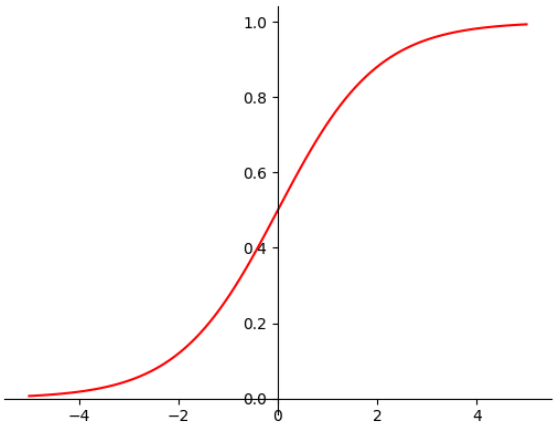
2. Duboko učenje

2.1. Logistička regresija

Logistička regresija povijesno prethodi dubokom učenju. To je model koji služi za predviđanje izlazne varijable (najčešće binarne) na temelju ulaznih podataka. Složenija je od linearne regresije jer koristi logističku prijenosnu funkciju, poznatu i kao sigmoida. To joj omogućuje klasifikaciju podataka čije klase ne podliježu linearnoj ovisnosti o ulazu. Standardan oblik logističke funkcije:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

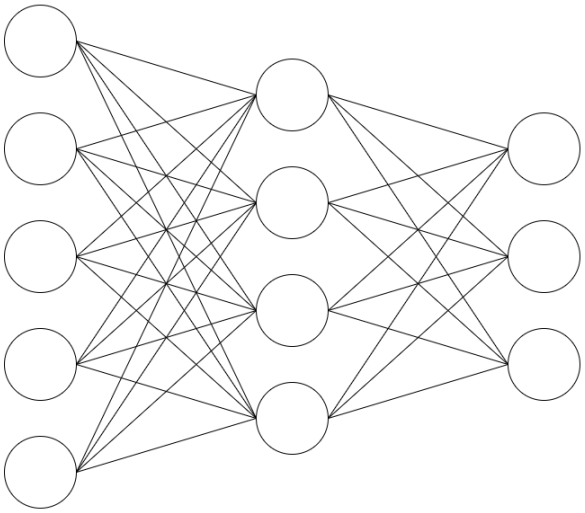
Logistička funkcija, iako na početku vrlo popularna, danas se malo koristi u latentnim slojevima dubokih modela zbog svojih nedostataka koji su opisani u 2.2.1.



Slika 2.1: Izgled logističke funkcije. Vidimo da je vrijednost funkcije ograničena na interval od 0 do 1.

2.2. Duboki modeli

Neuronska mreža je konstrukcija koja se sastoji od više čvorova (odnosno neurona) često poslaganih u slojeve. Čvorovi na ulaz dobivaju podatke iz ulaznog podatka ili od izlaza drugih čvorova. Duboke modele možemo promatrati kao poopćenje neuronskih mreža, oni su zapravo bilo kakav slijed parametriziranih nelinearnih transformacija.



Slika 2.2: Primjer neuronske mreže. Prvi sloj koji se sastoji od pet čvorova služi za prijenos ulaznih podataka. Oni se prolaskom kroz mrežu obrađuju nakon čega u trećem sloju od tri čvora dobivamo izlaze mreže.

Pojedini čvor sve svoje ulaze ne tretira jednako, neki više utječu na njegov izlaz, a neki manje. To je postignuto tako da se svaki ulaz najprije pomnoži s odgovarajućom težinom za tu vezu te se tako dobiveni umnošci zbroje. U tu sumu obično se dodaje i tzv. prag koji omogućuje promjenu izlaza čvora neovisno o ulazima. Konačno se na sumu primjenjuje prijenosna funkcija g.

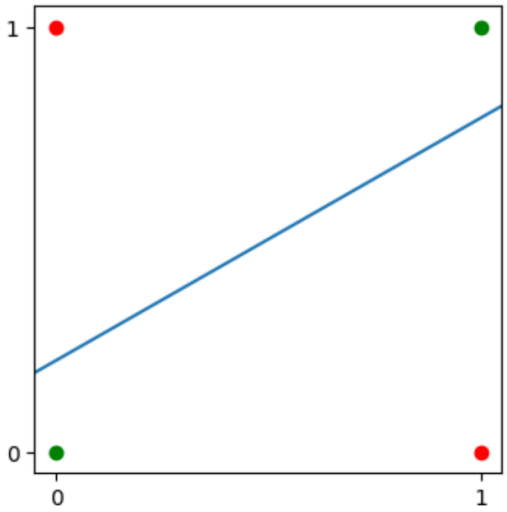
$$y = g(\sum_{i=1}^{n} w_i x_i + b) \tag{2.2}$$

S x je označen vektor ulaznih vrijednosti u čvor, a s y izlazna vrijednost. Vektor w sadrži težine koje pripadaju pojedinom ulazu, a vrijednost b je prag.

2.2.1. Prijenosne funkcije

Prijenosne ili aktivacijske funkcije služe za uvođenje nelinearnosti u model. Kada bismo maknuli prijenosnu funkciju, izlaz modela bio bi linearna kombinacija njegovih ulaza što uvelike ograničava njegove sposobnosti. Jednostavan primjer koji se koristi

da bi se to pokazalo je funkcija XOR. Model radi s kontinuiranim vrijednostima, ali ovdje na ulaz dobiva vrijednosti dva bita koji su diskretni. Domena ulaza će zato biti ograničena na vrijednosti 0 i 1 iako bi model mogao raditi i s drugim vrijednostima. Mi linearnom kombinacijom dva bita ne možemo odijeliti slučajeve kada su ti bitovi isti i onaj kada su različiti. To je najbolje prikazati grafički:



Slika 2.3: Problem učenja funkcije XOR. Zelene i crvene točke pripadaju različitim klasama koje linearan model ne može odijeliti.

Ulaze [0 0] i [1 1] prikazane zelenim točkama trebamo odijeliti od ulaza [0 1] i [1 0] koji su prikazani crvenim točkama. Budući da model koristi samo linearne kombinacije zadanih ulaza, on na izlazu dobiva funkciju $\alpha x_1 + \beta x_2$ gdje su α i β neki realni brojevi, a x_1 i x_2 vrijednosti bitova na ulazu u model. Sada možemo odabrati vrijednost izlaza iznad koje ćemo ulaz klasificirati u jednu klasu, a ispod te vrijednosti u drugu klasu. Jasno je da svi ulazi koji postižu upravo tu graničnu vrijednost leže na pravcu. Različite strane tog pravca pripadaju različitim klasama. Iz tog razloga linearan model točke može odijeliti samo pravcem kojim je nemoguće odijeliti zelene i crvene točke sa slike.

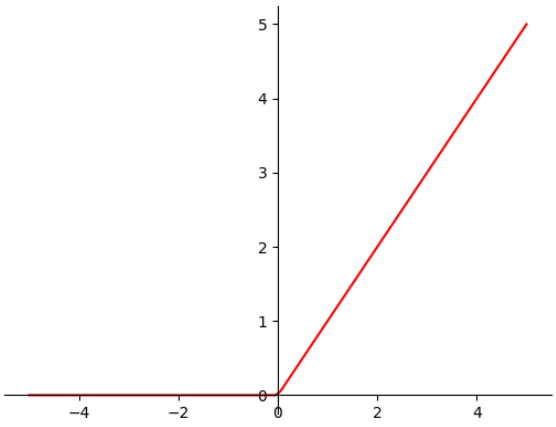
Logistička funkcija

S logističkom funkcijom već smo se upoznali u 2.1. Budući da ona uvodi nelinearnost, omogućila je učenje složenijih modela. Glavni problem logističke funkcije jest smanjenje njezine derivacije za vrijednosti udaljene od nule. Tijekom učenja, ako neka prijenosna funkcija na ulaz dobiva vrlo veliki ili mali broj, bit će teško značajno promijeniti izlaz te funkcije, odnosno trebali bismo dosta promijeniti ulaz kako bismo malo

promijenili izlaz. Strojno učenje zbog stabilnosti radi male pomake parametara i aktivacija pa učenje modela s logističkom funkcijom može jako usporiti ili čak stagnirati. Možemo na to gledati i ovako: Parametri se tijekom učenja mijenjaju ovisno o izračunatom lancu parcijalnih derivacija (algoritam gradijentnog spusta opisan u 2.2.4). To znači da male parcijalne derivacije povlače male pomake parametara.

Zglobnica

Zglobnica je danas vrlo raširena prijenosna funkcija, a računa se kao $g(x) = max(0, x)$.



Slika 2.4: Izgled zlobnice. S donje strane je ograničena nulom kao i logistička funkcija, ali u pozitivnom dijelu ima konstantan linearan rast.

Zglobnica poništava sve negativne vrijednosti te propušta isključivo pozitivne ulaze. Vidimo da je derivacija konstantna za sve pozitivne vrijednosti što rješava problem sporog učenja logističke funkcije.

Negiranje utjecaja negativnih vrijednosti kao posljedicu ima zapuštanje dijela mreže. Kad zglobnica na ulaz dobije negativnu vrijednost, tad će njezina derivacija biti 0 što onemogućuje učenje određenog dijela parametara. Zbog toga je izmišljena propusna zglobnica čija je formula $g(x) = max(\alpha x, x)$ gdje je α realan broj obično postavljen na 0.01. Ona u negativnom dijelu ima blagi rast koji omogućuje učenje unatoč negativnom ulazu.

2.2.2. Računanje izlaza mreže

Ako su čvorovi raspoređeni u slojeve te ako se radi o potpuno povezanoj mreži kao što je to slučaj na slici 2.2, možemo pojednostaviti računanje izlaza tako da ih računamo

sloj po sloj. Neka je vektor x vektor ulaznih podataka iz sloja $L-1$ u sloj L te neka je težina koja množi j-ti ulaz i-tog čvora u sloju L označena s w_{ij}. Tada izlaze čvorova L-tog sloja možemo predstaviti sljedećim vektorom:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = g\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \right) \tag{2.3}$$

Usredotočimo se na unutarnji dio jednadžbe, on računa izlaze čvorova prije djelovanja prijenosne funkcije, a ona pak djeluje na cijeli dobiveni vektor. Vidimo da se u i-tom retku nalaze težine vezane za i-ti čvor L-tog sloja. Te težine množe se s odgovarajućim ulazima te se zbrajaju, a na njih se još dodaje prag za taj čvor.

Ovakav zapis uz kompaktnost opisa izračuna izlaza cijelog sloja sa sobom nosi i implementacijsku efikasnost. Naime, optimirano matrično množenje je za red veličine brže od naivnih algoritama.

2.2.3. Funkcije gubitka

Funkcija gubitka je funkcija koja na temelju predviđanja modela te očekivane, odnosno ispravne vrijednosti daje mjeru pogreške. Što je predviđanje bliže stvarnoj vrijednosti, to je funkcija gubitka manja. Učenje modela bazira se na tome da pokušava minimizirati tu funkciju.

Unakrsna entropija

Unakrsna entropija često je korištena funkcija gubitka za semantičku segmentaciju. Ona je definirana na dvije razdiobe vjerojatnosti p i q te glasi:

$$H(p,q) = -\sum_{x \in X} p(x) \log q(x) \tag{2.4}$$

X je skup svih mogućih događaja, p je stvarna distribucija vjerojatnosti, a q je procijenjena. Najčešće korišteni logaritmi imaju baze 2, e i 10. U našem slučaju, p će imati sve nule osim jedne jedinice jer za ulazni podatak postoji samo jedna moguća klasa koja ima vjerojatnost 1. q je distribucija vjerojatnosti određena na temelju izlaza modela. Kada pojednostavimo formulu, dobivamo da je vrijednost unakrsne entropije u našem slučaju $-\log q(t)$ gdje je t oznaka točne klase ulaznog podatka.

2.2.4. Propagacija pogreške unatrag i gradijentni spust

Propagacija pogreške unatrag je algoritam čija je zadaća određivanje gradijenta funkcije gubitka, odnosno parcijalne derivacije funkcije gubitka po svakom parametru modela. Derivacije parametara nekog sloja određuju se pomoću derivacija sljedećeg sloja, primjenom pravila ulančavanja. Zbog toga se njihovo računanje provodi od zadnjeg sloja prema prvom te se postupno gradi lanac parcijalnih derivacija. Nakon propagacije pogreške unatrag gradijentni spust koristi izračunati gradijent te mijenja parametre modela kako bi smanjio funkciju gubitka te tako uči model. On će parametre pomaknuti u smjeru suprotnom od gradijenta jer je to smjer u kojem funkcija gubitka najbrže pada. Od parametara se ne oduzima čisti gradijent jer bi to radilo prevelike skokove te bismo se teško pozicionirali blizu minimuma funkcije gubitka. Zato se gradijent prvo množi s relativno malim brojem μ zvanim stopa učenja koji omogućava finije postavke parametara.

2.2.5. Optimizatori

Običan gradijentni spust mijenja parametre nakon što izračuna gradijent na temelju svih dostupnih ulaznih podataka. Kako učenje ne bi bilo presporo može se koristiti stohastički gradijentni spust koji računa gradijent nakon svakog ulaznog podatka ili nakon obrade manjih grupa podataka. Na taj način se parametri modela češće mijenjaju te je učenje brže.

Često je problem odabrati pravu stopu učenja. Ako odaberemo preveliku stopu učenja onemogućit ćemo manje pomake parametara, a ako odaberemo premalu stopu učenja učenje će trajati predugo. Zbog toga su razvijeni brojni optimizatori gradijentnog spusta koji ubrzavaju proces učenja te ga čine učinkovitijim. Jedan od tih optimizatora je Adam izvorno opisan u [8]. On pamti prijašnje vrijednosti gradijenata i kvadrata gradijenata, a nove vrijednosti računa kao kombinaciju prijašnjih i trenutne, tzv. eksponencijalni pomični prosjek:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L(w_t) \tag{2.5}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla^2 L(w_t) \tag{2.6}$$

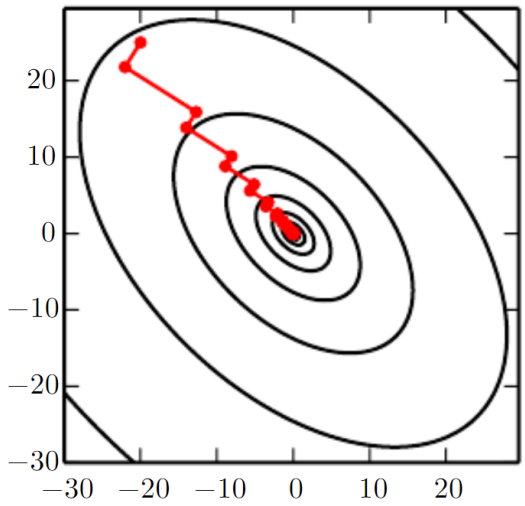
Niz m pamti težinski prosjek gradijenata, a njihove kvadrate pamti niz v. $\nabla L(w_t)$ je gradijent funkcije gubitka za trenutne težine. Uobičajene vrijednosti parametara β_1 i

β_2 su 0,9 i 0,999. Težine modela tada se mijenjaju po sljedećoj formuli:

$$w_{t+1} = w_t - \hat{m}_t(\frac{\mu}{\sqrt{\hat{v}_t} + \epsilon}) \tag{2.7}$$

Vrijednosti $\hat{m}_t$ i $\hat{v}_t$ računaju se kao $\frac{m_t}{1-\beta_1^t}$ i $\frac{v_t}{1-\beta_2^t}$, a služe za odstranjivanje početne pristranosti zbog postavljanja m_0 i v_0 na 0.

Na ovaj način uvedeno je zaboravljanje gradijenata. Što je gradijent stariji, to ga više faktora β množi pa manje utječe na novu vrijednost. Pamćenje nedavnih gradijenata omogućuje djelomično zadržavanje njihovog smjera i veličine zbog čega optimizacija brže napreduje kroz nepovoljne dijelove funkcije gubitka.



Slika 2.5: Primjer optimiziranog gradijentnog spusta. Crne linije prikazuju dijelove funkcije gubitka iste vrijednosti, a vrijednost joj opada prema sredini. Crvena linija predstavlja kretanje parametara tijekom učenja. Izvor: [4]

2.2.6. Regularizacija

Regularizacija je naziv za metodu koja služi kao pomoć kod učenja te sprječava česte probleme poput prenaučenosti. Cilj joj je povećati generalizacijsku moć i smanjiti pogrešku na podacima za validaciju, uz moguće povećanje pogreške na podacima za učenje.

Kažnjavanje norme parametara modela

Kažnjavanje norme parametara modela je metoda regularizacije koja ne dozvoljava modelu da postane pretjerano složen jer to dovodi do prenaučenosti. To ostvaruje tako

da kažnjava velike težine povećanjem funkcije gubitka. Velike težine često su pokazatelj prenaučenosti jer to znači da se model jako fokusira na neke dijelove ulaza što možda radi na podacima za treniranje, ali vjerojatno neće i na neviđenim podacima. Funkciji gubitka dodaje se suma kvadrata težina pomnožena s faktorom λ. Tim faktorom možemo kontrolirati koliko jako želimo ograničiti težine. Češće korištene vrijednosti su 0,1 i 0,01.

Normalizacija mini-grupe

Normalizacija mini-grupe još je jedna od tehnika regularizacije, ali ona se umjesto parametrima modela bavi ulaznim podacima. Ideja je normalizirati mini-grupu prije ulaza u pojedini sloj mreže kako slijedi:

$$x_{norm} = \frac{x - \mu}{\sigma + \epsilon} \tag{2.8}$$

μ predstavlja aritmetičku sredinu mini-grupe, σ predstavlja standardnu devijaciju, a ϵ malu konstantu koja sprječava dijeljenje s jako malim vrijednostima. Intuitivno se postavlja da je lakše učiti na normaliziranim podacima. Kada bi npr. podaci bili jako odmaknuti od nule, težine modela morale bi biti velike kako bi promijenile izlaz te bismo imali problema s preciznošću tih težina.

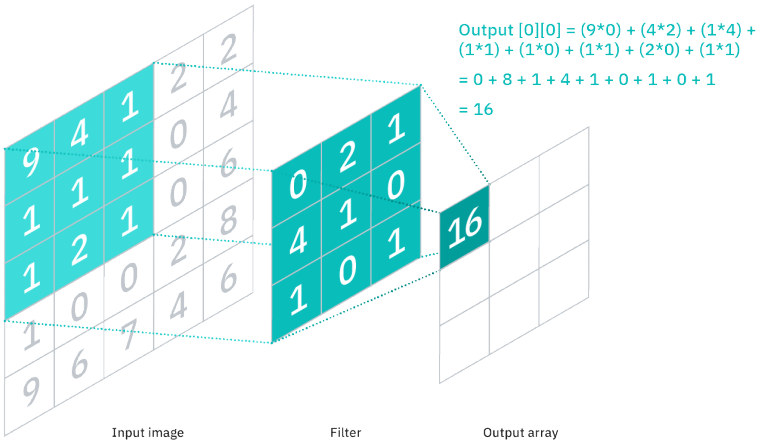
2.3. Konvolucijske neuronske mreže

Kako se područje umjetne inteligencije više razvijalo, potreba za modelima koji rade sa slikama sve je više rasla. Obične neuronske mreže nisu zadovoljile zahtjeve zbog nemogućnosti generalizacije s obzirom na položaj objekata na slici. Glavna konstrukcija dubokih modela za računalni vid upravo je konvolucijska mreža. U srži tih mreža leži operacija konvolucije.

2.3.1. Konvolucija

Operacija konvolucije služi za izlučivanje podataka iz slike. Ona ima mogućnost prepoznavanja jednostavnih uzoraka, a njihovim kombiniranjem možemo dobiti vrlo sofisticiran model. Za operaciju konvolucije potrebna nam je matrica koja se zove jezgra. Predstavimo ulaznu sliku matricom u kojoj svako polje označava intenzitet piksela. Slike u boji su predstavljene kanalima (npr. RGB), ali zbog jednostavnosti ćemo

promatrati samo jedan kanal. Elemente izlazne matrice nazvane mapom značajki dobivamo tako da preklopimo sliku i jezgru, preklopljene elemente pomnožimo te dobivene umnoške zbrojimo. Za izračun sljedećeg elementa pomaknemo jezgru za određeni pomak.



Slika 2.6: Primjer konvolucije. Matrica u sredini je jezgra veličine 3x3. Na lijevoj matrici koja predstavlja sliku osjenčani su elementi koje jezgra preklapa. Izvor: [7]

Primijetimo da dimenzije dobivene mape značajki ovise o veličini jezgre te o odabranom pomaku. Čak i kada bismo postavili pomak na 1 još uvijek bismo obje dimenzije smanjili za $K - 1$ gdje je K dimenzija jezgre. Zbog toga se originalnu matricu može nadopuniti prije konvolucije tako da se sa svake strane matrice doda određen broj elemenata te privremeno povećaju dimenzije. Nakon operacije konvolucije ponekad se primjenjuje i prijenosna funkcija kako bi se u model uvela nelinearnost.

2.3.2. Sažimanje

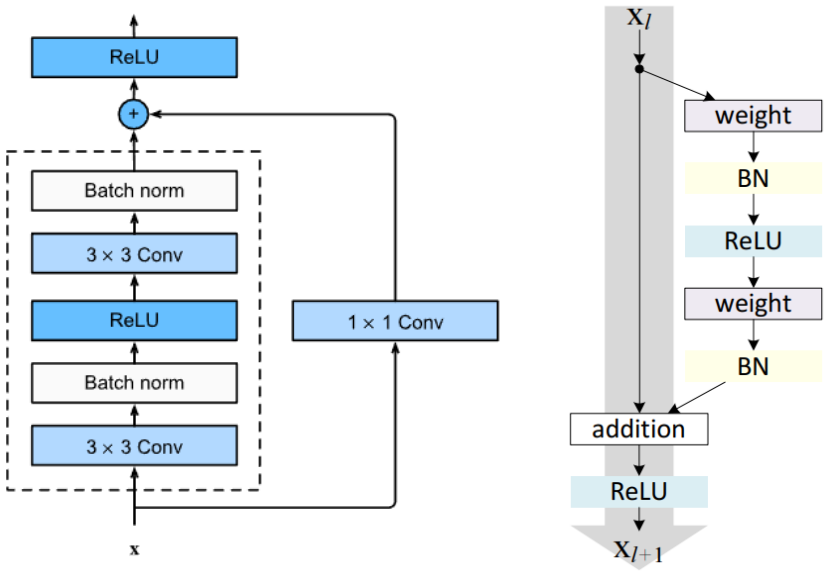
Operacija sažimanja obično slijedi nakon operacije konvolucije, a služi za smanjenje dimenzionalnosti. Slično kao i kod konvolucije, prolazimo ulaznom matricom te promatramo podmatrice zadane veličine. Pomoću određene funkcije tu podmatricu predstavimo jednim brojem te se pomičemo na sljedeću podmatricu odmaknutu za zadani pomak. Ako kao pomak uzmemo vrijednost 2, dobit ćemo mapu značajki s dvostruko manjim dimenzijama. To nam omogućava da zadržimo razumne količine podataka unatoč povećanju dubine, odnosno broja mapa značajki.

Postoje dvije glavne vrste sažimanja, a razlikuju se po funkciji koju primjenjujemo na podmatrice. Jedna vrsta uzima najveću vrijednost podmatrice, a druga uzima prosječnu vrijednost. Pokazalo se da prva vrsta daje bolje rezultate jer se izražene značajke ne mogu zagušiti tako lako kao kod druge vrste.

2.4. Rezidualne mreže

Ako želimo dobiti konvolucijski model veće složenosti, prva stvar koju bismo vjerojatno probali napraviti bila bi povećanje dubine modela. Eksperimentalno se pokazalo da takvi modeli vrlo sporo uče. Naime, za učenje se koristi propagacija pogreške unatrag koja određuje utjecaj pojedinih parametara na funkciju gubitka pomoću parcijalnih derivacija. Što je model dublji, to se više derivacija množi te to može postati problem u početnim slojevima mreže. Ako su parcijalne derivacije relativno male, dolazi do problema zvanog "nestajući gradijent". Tada se prvi slojevi jedva mijenjaju pa skoro da i ne sudjeluju u procesu učenja. Pažljiv odabir prijenosne funkcije, npr. zglobnica, može pomoći, ali ona nije jedini faktor u izračunu funkcije gubitka. Nestajući gradijent se ne pojavljuje u modelima koji koriste normalizaciju po grupi kao što je objašnjeno u [5], no ta operacija svejedno ne omogućuje korištenje dubokih modela zbog nepogodnosti optimizacije funkcije gubitka.

Rezidualne mreže [5] su jedan od načina da se riješi ovaj problem. One se sastoje od rezidualnih jedinica koji se pak sastoje od operacija konvolucije, normalizacije po grupi, prijenosne funkcije te preskočne veze. Ideja iza rezidualnih mreža je zbrajanje ulaza konvolucijske jedinice s njenim izlazom. Tako konačan izlaz ovisi o originalnom izlazu iz jedinice, ali i o ulazu u jedinicu. Tu preskočnu vezu možemo gledati kao prečicu koja omogućuje prosljeđivanje gradijenta do ulaza u jedinicu što rješava prethodno opisani problem.



Slika 2.7: Primjeri rezidualne jedinice. Nakon operacija konvolucije i normalizacije prikazanih unutar isprekidanog okvira rezultat se zbraja s originalnim ulazom. Izvori: [1], [6]

Budući da se unutar obične jedinice rade konvolucije, dubina izlazne mape značajki ne mora odgovarati dubini one ulazne. To se rješava tako da se na preskočnoj vezi primjenjuje konvolucija s jezgrom dimenzija 1x1 čija je zadaća promijeniti dimenzije mape značajki.

2.4.1. ResNet-18

ResNet18 jedan je od popularnih modela rezidualne mreže. ResNet-18 obavlja normalizaciju mini-grupe nakon svake operacije konvolucije te to neću dalje navoditi. Započinje konvolucijom s veličinom jezgre 7x7 i pomakom 2 koja daje 64 kanala. Nakon toga slijedi sažimanje maksimalnom vrijednošću, veličinom jezgre 3x3 te pomakom 2. Slijedi osam rezidualnih jedinica koji izgledaju kao na slici 2.7. Sve neparne jedinice osim prve udvostručuju broj kanala, ali zato raspolavljaju visinu i širinu mapa značajki. Nakon toga se obavlja globalno sažimanje prosjekom, što znači da se svaki kanal konačne mape značajki predstavlja prosjekom njegovih vrijednosti te dobivamo niz od 512 brojeva. Konačno, taj niz se provlači kroz potpuno povezani sloj.

2.5. Semantička segmentacija

Semantička segmentacija je problem podjele slike na smislene cjeline te njihovo klasificiranje. To je danas vrlo važan zadatak s brojnim primjenama, npr. raspoznavanje elemenata važnih za autonomnu vožnju, prepoznavanje opasnih struktura u medicinskim snimkama, automatsko izrezivanje željenog dijela slike itd.

Semantička segmentacija svodi se na pridjeljivanje klase svakom pikselu. Tada susjedni pikseli koji imaju istu klase čine cjelinu iste semantike.

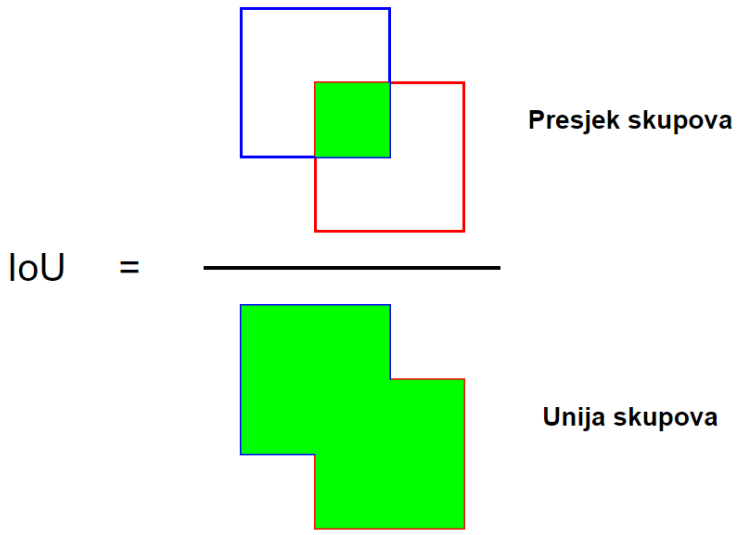
2.5.1. Točnost semantičke segmentacije

Kako bismo objektivno znali koliko je neki model dobar, trebamo kvantitativan način određivanja točnosti. Prva metrika koja bi većini ljudi pala na pamet je udio točno klasificiranih piksela u cijeloj slici. Što je veći taj udio, to je model bolji. Problem s tim sustavom je taj što ne razmatra učestalost pojavljivanja pojedinih klasa. Uzmimo da nam je priroda skupa podataka takva da velika većina piksela pripada istoj klasi, npr. slike šuma u kojima želimo segmentirati stabla. Tada možemo napraviti jako loš model koji ne primjenjuje postupke učenja, nego cijelu sliku klasificira kao stablo. Ako stabla zauzimaju oko 75% slike, tada će tolika biti i točnost modela iz kojeg nismo

dobili nikakve informacije.

Omjer presjeka i unije

Kako bismo riješili ovaj problem, točnost semantičke segmentacije najčešće mjerimo omjerom presjeka i unije (engl. *IoU, Intersection over Union*). Ta metrika za svaku klasu računa IoU točnost, a ukupna IoU točnost modela je srednja vrijednost točnosti svih klasa. IoU pojedine klase računa se tako da se broj piksela koji su točno klasificirani kao ta klasa podijeli s ukupnim brojem piksela koji stvarno pripadaju toj klasi ili za koje je model predvidio da pripadaju toj klasi. Iskazano drugačije, IoU klase je omjer kardinalnog broja presjeka i kardinalnog broja unije dva skupa gdje je prvi skup skup piksela koji stvarno pripadaju toj klasi, a drugi skup piksela za koje je model odredio da pripadaju toj klasi.



Slika 2.8: Grafički prikaz metrike IoU. Omjer kardinalnih brojeva presjeka i unije ovdje je prikazan omjerom zelenih površina.

Najveći IoU postižemo onda kada su spomenuti skupovi jednaki. Klasificiranjem manjeg broja ispravnih piksela smanjit će se brojnik, a klasificiranjem dodatnih piksela povećava se nazivnik. Vratimo li se na primjer modela koji sve klasificira u istu klasu, vidjet ćemo da je IoU za tu klasu isti kao i udio točno klasificiranih piksela, ali je za ostale klase ta vrijednost 0 što smanjuje prosječnu vrijednost IoU-a.

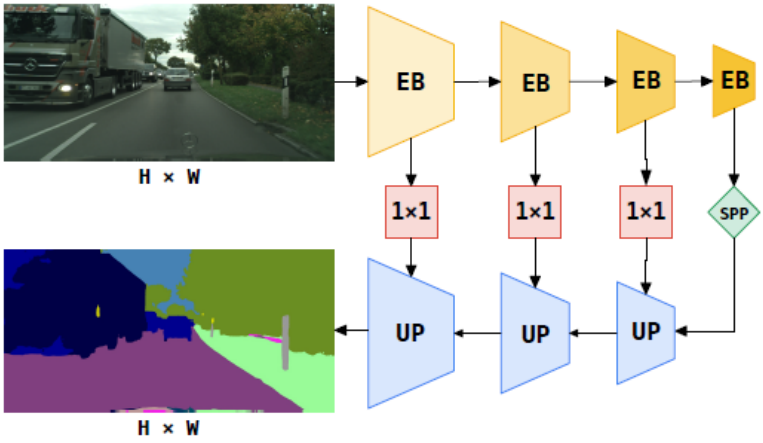
Preciznost i odziv

Preciznost i odziv također su metrike koje možemo koristiti za procjenu kvalitete modela. Preciznost za neku klasu računamo kao omjer točno klasificiranih piksela za tu klasu i ukupnog broja piksela kojima je model pridijelio tu klasu. On zapravo daje udio točnih klasifikacija u svim pikselima klasificiranim u tu klasu.

Odziv klase računamo kao omjer točno klasificiranih piksela za tu klasu i ukupnog broja piksela koji su stvarno u toj klasi, odnosno on daje udio točnih klasifikacija u svim pikselima koji pripadaju toj klasi.

2.6. SwiftNet

SwiftNet je arhitektura specijalizirana za semantičku segmentaciju u stvarnom vremenu. Jednorazinska inačica arhitekture sastoji se od konvolucijske okosnice (engl. *backbone*) i jednostavnih modula za naduzorkovanje koji su spojeni lateralnim vezama. Također su ugrađene i metode za povećanje receptivnog polja. Veličina receptivnog polja predstavlja broj piksela ulazne slike koji utječu na predikciju, odnosno veličinu promatranog konteksta oko pojedinog piksela tijekom njegove klasifikacije.



Slika 2.9: Prikaz arhitekture SwiftNet. Narančasto osjenčani trapezi predstavljaju procesne blokove, a plavo osjenčani module za naduzorkovanje. Crveni kvadrati predstavljaju konvolucije na lateralnim vezama, a zeleni kvadrat je modul za prostorno piramidalno sažimanje. Izvor: [13]

Niz procesnih blokova koji su na slici 2.9 obojeni narančastom bojom služi za izvlačenje mapa značajki iz ulazne slike. Cilj tih blokova je napraviti dobru diskriminaciju između različitih klasa. Oni se mogu trenirati ispočetka postavljanjem početnih parametara na neke slučajne vrijednosti, ali se mogu koristiti i već istrenirani modeli

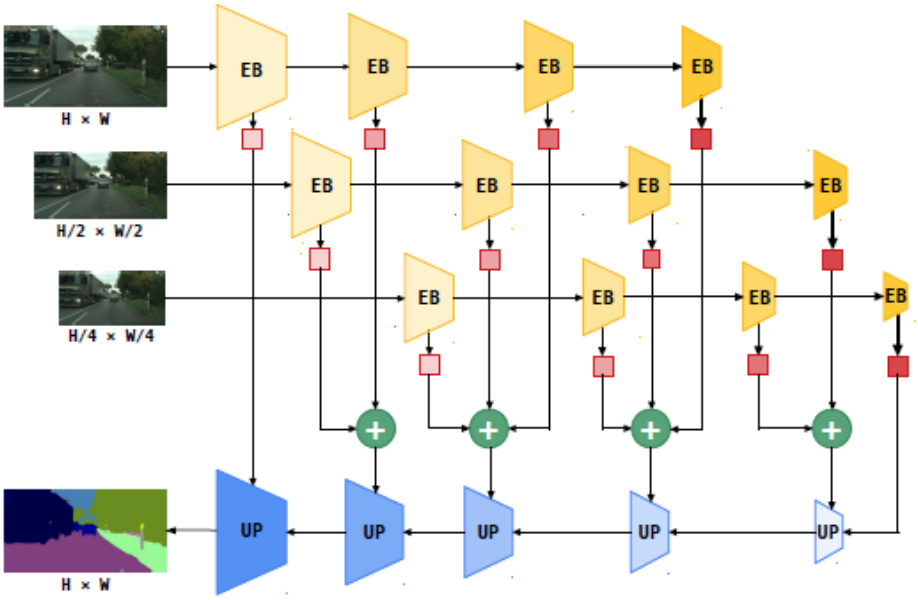
koji dobro diskriminiraju na nekom skupu podataka za klasifikaciju slika. Konvolucijske okosnice tipično se sastoje od četiri procesna bloka koji se pak sastoje od rezidualnih konvolucijskih jedinica. Prvi procesni blok na izlazu daje mape značajki s četiri puta manjom visinom i širinom od ulazne slike, dok svaki sljedeći rezultira dvostruko manjom visinom i širinom.

Nakon prolaska kroz modul za prostorno piramidalno sažimanje koji služi za povećanje receptivnog polja dolazimo do niza modula za naduzrokovanje. Oni služe za povećanje dimenzionalnosti izdvojenih značajki kako bismo ih na kraju mogli klasificirati. Budući da su visine i širine mapa značajki na izlazu iz zadnjeg procesnog bloka 32 puta manje od originalne slike, treba nam način da vratimo originalnu rezoluciju. Zato oni provode naduzorkovanje pomoću bilinearne interpolacije koja zapravo uzima težinski prosjek susjednih piksela kako bi "proizvela" nove piksele. Time zapravo nismo dobili nikakve nove informacije pa još uvijek ne možemo napraviti finiju segmentaciju. Tu u igru dolaze lateralne veze koje iz procesnog dijela šalju mape značajki koje se onda zbrajaju s naduzorkovanim podacima. Svi moduli za naduzorkovanje imaju istu dubinu (odnosno broj kanala) mapa značajki, ono što se mijenja su dimenzije pojedine mape. To predstavlja problem kada želimo zbrojiti mape značajki iz procesnog bloka koje imaju manju dubinu. Zato se na lateralnim vezama primjenjuju 1x1 konvolucije koje namještaju tu dubinu. Nakon zbrajanja se dobivene mape značajki obrade jednom 3x3 konvolucijom. Eksperimentalno je pokazano da se dobivaju bolji rezultati kada iz procesnog bloka uzmemo mape značajki prije primjenjivanja prijenosne funkcije, zglobnice.

Osim prostornog piramidalnog sažimanja, razvijen je još jedan način povećanja receptivnog polja, a to je piramidalna fuzija. Ideja je da primijenimo niz procesnih blokova na više istih slika, ali različitih rezolucija. Smanjenjem rezolucije dobivamo veće receptivno polje jer jednak broj piksela prikazuje veću fizičku udaljenost. Promotrimo model na slici 2.10 koji koristi slike pune, dvostruko te četverostruko manje rezolucije.

Na svaku sliku opet se primjenjuje niz od četiri procesna bloka. Glavna promjena očituje se u dijelu za naduzorkovanje gdje se u pojedini modul za naduzorkovanje lateralnim vezama dodaju sve mape značajki iz procesnih blokova koje imaju odgovarajuću rezoluciju. Ovaj model bolje prepoznaje objekte različitih veličina jer dobiva različite početne rezolucije.

Kod ovakvih modela primijećen je problem težeg raspoznavanja manjih objekata na slici. Razlog tome je taj što prvi modul za naduzorkovanje na ulaz dobiva mapu značajki malih dimenzija u kojoj se gube manji objekti, a bilinearna interpolacija tu



Slika 2.10: Prikaz SwiftNeta s piramidalnom fuzijom. Tri slike na ulazu različitih dimenzija model zasebno provlači kroz niz procesnih blokova. Lateralne veze dobivene mape značajki vode do odgovarajućeg modula za naduzorkovanje gdje se one zbrajaju. Izvor: [13]

neće pomoći. Kao dobro rješenje pokazalo se pojačavanje funkcije gubitka za piksele koji su blizu semantičke granice. Tako će piksel koji je jako blizu granice imati dosta veći doprinos funkciji gubitka nego onaj u sredini semantičke grupe. Udaljenost pojedinog piksela od granica računa se pomoću maski pravih klasa. Na ovaj način model više "pazi" da točno klasificira piksele na granicama.

3. Skup podataka

U ovom radu za semantičku segmentaciju zgrada korišten je skup podataka Inria uveden u [11]. Sastoji se od 360 satelitskih snimki s deset različitih područja koje ukupno prekrivanju površinu od 810 km^2. Pet od tih deset područja dolazi i s maskama u kojima su segmentirane zgrade od ostatka slike. Svaka slika je rezolucije 5000x5000 piksela, a svaki piksel pokriva 9 dm^2 stvarne površine.

Slika 3.1: Primjer satelitske snimke dijela grada Beča i njezine maske. Bijeli pikseli u masci znače da se na tom mjestu u originalnoj slici nalazi zgrada. Izvor: [10]

Ovaj skup podataka dobiven je pomoću javno dostupnih satelitskih snimki i službenih otisaka zgrada. Za originalne slike koristi se RGB zapis, a za maske se koristi jedan kanal s vrijednostima 0 i 255 gdje vrijednost 255 označava da piksel sadržava zgradu.

Za potrebe učenja, dio skupa podataka koji sadrži maske rastavljen je u dva podskupa, skup za treniranje i skup za validaciju. Skup za treniranje služi za namještavanje parametara modela na temelju predviđanja i stvarne segmentacije slike. Skup za validaciju služi za provjeru kvalitete modela, na njima model ne uči. Kada bismo model trenirali na istom skupu na kojem ga provjeravamo, dolazi do problema prenaučenosti. Tada model prividno ima odlične rezultate jer je naučio podatke na kojima ga testiramo, ali može imati problema kada mu damo neke još neviđene podatke. U tom slučaju model nije uspio generalizirati naučeno pa nije koristan. Kao validacijski skup odvojio sam sve slike iz jednog područja (Austin) jer je tada validacija dobar pokazatelj gene-

ralizacije. Različita područja mogu imati različite gustoće, oblike i razmještaje zgrada te mogu imati različite tonove boja i smjerove sjena. Zbog toga je bolje ne trenirati nad istim područjima nad kojima validiramo.

4. Programska izvedba

Kao model za učenje koristio sam implementaciju SwiftNeta s piramidalnom fuzijom [13]. Implementacija je razvijena u najpopularnijem programskom jeziku za strojno učenje, Pythonu, a korištene su i neke biblioteke koje implementiraju potrebne alate.

PyTorch pruža radni okvir za duboko učenje. Omogućuje jednostavno oblikovanje, učenje i vrednovanje modela pomoću već gotovih klasa u kojima su implementirane potrebne metode. Također koristi posebnu strukturu podataka, tenzore, čije su operacije vrlo brze zahvaljujući korištenju grafičke jedinice. Jedan od često korištenih i vrlo korisnih alata je automatska diferencijacija koja samostalno računa gradijente što bi često bio problem napraviti ručno.

Torchvision nadograđuje PyTorch, a fokusira se na domenu slika i videa. Sadrži korisne transformacije te neke često korištene skupove podataka i arhitekture modela.

Numpy je biblioteka prikladna za operacije s nizovima. Pažljivo upravlja memorijskim prostorom pa su dobiveni nizovi puno brži od lista u Pythonu. Uz to su i računski zahtjevni dijelovi implementirani u brzim jezicima kao što su C i C++. Za rad s matricama koristi biblioteke s operacijama koje se optimirano izvode u strojnom kodu.

Korišten je i modul Pillow koji je nadopuna modula PIL i služi za rad sa slikama, a za iscrtavanje grafova i prikazivanje slika pogodna je biblioteka Matplotlib.

Sav kod osim pretprocesiranja podataka izvršavao sam na Google Colabu. Google Colab je usluga koja omogućuje izvršavanje Python koda u oblaku. Pruža dobre računalne resurse pa je pogodna za duboko učenje. Osim toga omogućuje povezivanje s Google Diskom što pojednostavljuje pristup kodu i podacima.

Pretprocesiranje podataka

Zbog memorijske ograničenosti te brzine provođenja eksperimenata, odlučio sam se za učenje na skupu podataka polovične rezolucije. Slike smanjenih dimenzija dobio sam izvršavajući jednostavnu skriptu koja koristi funkciju resize modula PIL. Novodobiveni skup podataka postavio sam na Google Drive iz kojega je Google Colab vukao

podatke.

Računanje konfuzijske matrice

Konfuzijska matrica je matrica koja bilježi učestalost svake kombinacije predviđene klase i prave klase prilikom klasifikacije na promatranom podatkovnom skupu. U našem slučaju, svaki od piksela jest jedan podatak kojeg klasificiramo pa će zbroj svih vrijednosti konfuzijske matrice odgovarati broju piksela u promatranom podatkovnom skupu.

Neka se u matrici u i-tom retku i j-tom stupcu nalazi broj piksela kojima je model predvidio klasu i, a pripadaju klasi j. Usporedbom predviđanja i oznaka točnih razreda trebamo popuniti konfuzijsku matricu. Najlakši način je iteriranje for petljom po svim pikselima podatkovnog skupa te dodavanje jedinice na odgovarajuće mjesto u konfuzijskoj matrici. Problem s ovim pristupom je taj što je Python općenito spor te se često pristupa istim elementima matrice. Ivan Krešo je zbog toga u svom radu [9] koristio programski jezik Cython koji omogućuje izvršavanje koda pisanog u C-u unutar Pythona kako bi ubrzao računski zahtjevne dijelove programa.

Budući da mi imamo samo dvije različite klase, možemo se poslužiti malim trikom u određivanju elemenata konfuzijske matrice. Kako je već prije spomenuto, Numpy ima implementirane vrlo optimizirane metode za rad s nizovima. Ono što ćemo mi koristiti je metoda koja vraća broj elemenata niza različitih od nule, *count_nonzero()*.

Želimo odrediti četiri broja konfuzijske matrice M, a to su $M_{1,1}$, $M_{1,2}$, $M_{2,1}$ i $M_{2,2}$. Označimo niz predviđanja oznaka klasa piksela s p, a niz pravih oznaka klasa s t. Jedinica na i-toj poziciji niza označava da je pikselu i dodijeljena klasa zgrade, a inače nije. Prebrojimo li elemente različite od nule u nizu p (suma niza isto radi, ali je sporija), dobit ćemo vrijednost $M_{2,1} + M_{2,2}$. Brojimo koliko piksela je naš model klasificirao kao zgradu, a svaki od njih može stvarno pripadati klasi zgrade ($M_{2,2}$) ili ne ($M_{2,1}$). Slično, brojanjem jedinica u nizu t dobivamo $M_{1,2} + M_{2,2}$. Sada određivanjem bilo kojeg od elemenata matrice M možemo lako odrediti sve ostale.

Odredit ćemo element $M_{1,1}$. Prvo možemo spojiti nizove p i t u jednu matricu dimenzija 2xn gdje je n broj piksela slike:

$$\begin{bmatrix} p_1 & p_2 & \cdots & p_n \\ t_1 & t_2 & \cdots & t_n \end{bmatrix} \tag{4.1}$$

Ako u rezultantnoj matrici prebrojimo elemente različite od nule po stupcima, dobit ćemo niz koji na i-toj poziciji ima broj elemenata različitih od nule u i-tom stupcu matrice. To je ekvivalentno zbroju nizova p i t jer su sve vrijednosti ili 0 ili 1, ali

se zbroj nizova eksperimentalno pokazao sporijim pa ga zato nisam koristio. Dalje možemo prebrojiti elemente različite od nule u novodobivenom nizu. To nam je korisno jer smo brojali sve kombinacije vrijednosti nizova p i t osim one kada su obje vrijednosti nula, tj. dobili smo vrijednost $M_{1,2} + M_{2,1} + M_{2,2}$. Naime, da bi element na poziciji i niza $p + t$ bio nula, mora vrijediti $p_i = 0$ i $t_i = 0$, a to su elementi koji pripadaju $M_{1,1}$ koje jedino nismo brojali. Sada lako možemo odrediti $M_{1,1}$ jer znamo n, a budući da znamo i $M_{2,1} + M_{2,2}$ i $M_{1,2} + M_{2,2}$ možemo dobiti i $M_{2,2}$ kao $(M_{2,1} + M_{2,2}) + (M_{1,2} + M_{2,2}) + M_{1,1} - n$.

Ovaj način računanja konfuzijske matrice daje značajno ubrzanje u odnosu na običnu for petlju. Na slikama dimenzija 2500x2500 obična for petlja trebala je preko 20 sekundi po iteraciji dok je optimiziranom načinu trebalo ispod dvije. Isječak programskog koda koji računa konfuzijsku matricu:

```
def collect_confusion_matrix(pred, true, conf_mat):
    a = np.count_nonzero(pred)
    b = np.count_nonzero(true)
    c = pred.size - np.count_nonzero(
    np.count_nonzero(np.array((pred, true)), axis=0))
    d = (a + b + c) - pred.size
    conf_mat[0][0] = c
    conf_mat[0][1] = b - d
    conf_mat[1][0] = a - d
    conf_mat[1][1] = d
```

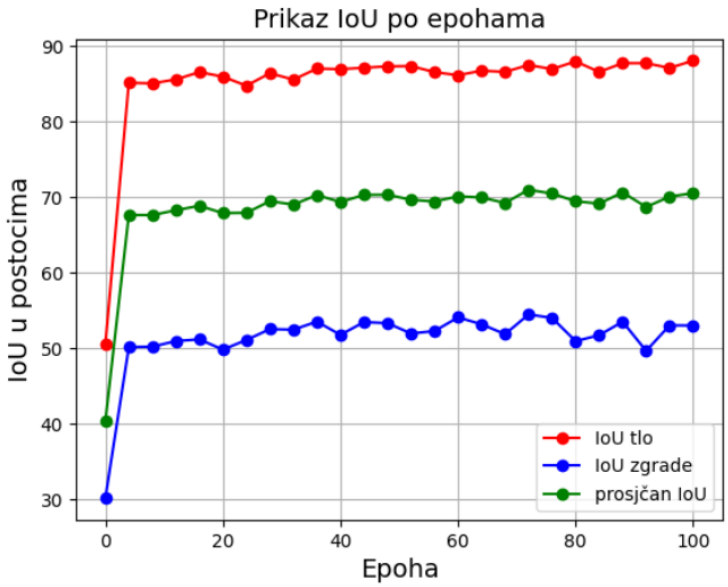
5. Eksperimentalni rezultati

Za učenje su korišteni slučajni isječci veličine 512x512 piksela s mogućim zrcaljenjem. Time prividno dobivamo više ulaznih podataka te omogućujemo modelu učenje različitih varijanti iste slike. Sve su slike snimljene s iste visine te zbog toga nije korišteno slučajno uvećanje slika. Početna stopa učenja je $4 \cdot 10^{-4}$, a korišteni optimizator je Adam. Parametar za kažnjavanje norme parametara modela postavljen je na 10^{-4}. Kao veličinu mini-grupe uzeo sam najveći broj za koji se eksperiment izvršavao stabilno, odnosno za koji je bilo dovoljno mjesta u radnoj memoriji. Eksperimentalno sam odredio da je taj broj osam. Kao okosnica modela korišten je ResNet-18 model predtreniran na skupu podataka ImageNet [3].

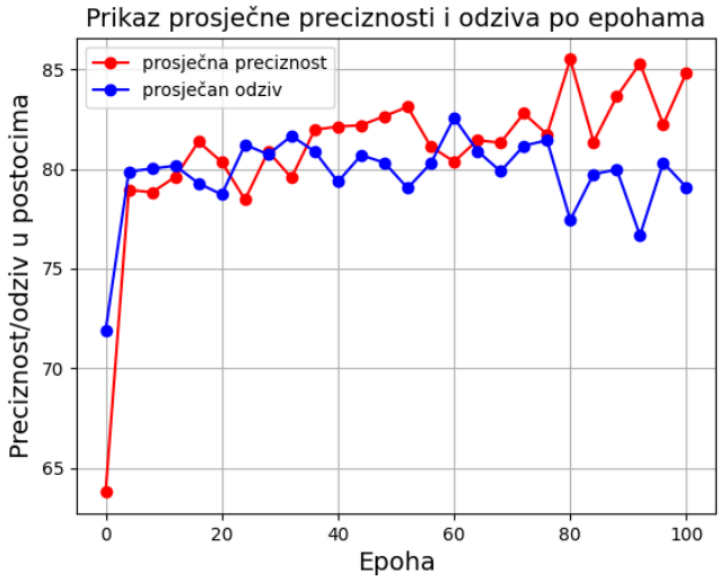
Model sam učio 165 epoha, a tada sam stao jer već dugo nije bilo poboljšanja točnosti. Validacija modela provođena je svake četiri epohe. Model koji ima najbolje rezultate na validacijskom skupu polovične rezolucije nastao je u 72. epohi kada je IoU tla bio 87,5%, IoU zgrade 54,5%, a srednji IoU 71,0%. U toj epohi na skupu za treniranje IoU tla bio je 79,1%, IoU zgrade 73,4%, a srednji IoU 76,2%. Na natjecanju koje se može naći na [10] najbolji modeli postigli su srednji IoU od oko 81%, no oni su trenirani na skupu pune rezolucije. Na slici 5.1 prikazan je IoU po klasama te srednja vrijednost IoU-a kroz epohe.

Možemo vidjeti da prosječni IoU raste do oko 50. epohe nakon čega stagnira uz manje oscilacije. Također možemo vidjeti da je IoU tla puno veći od IoU-a zgrada. Jedan od razloga za to je zastupljenost pojedine klase na slikama. Tlo se kao klasa dosta češće pojavljuje od zgrada pa je zbog toga robusnije na manje pogreške u segmentaciji dok za zgrade vrijedi obrnuto. Iako možda na prvi pogled ne izgleda tako, izvršavanje jednostavne skripte pokazalo je da je prosječan udio piksela koji sadrže zgradu u skupu za validaciju svega 15,0%.

Na slici 5.2 dan je grafički prikaz prosječne preciznosti i odziva po epohama. Kao i kod IoU-a, primjećujemo generalan rast u prvih par desetaka epoha. Nakon toga dolazi do povećanja preciznosti, ali opadanja odziva te takvi ostaju do kraja učenja. Nakon 160 epoha preciznost oscilira oko 84%, a odziv oko 79%. Ova inverzna ovisnost



Slika 5.1: Prikaz IoU-a na skupu za validaciju kroz prvih 100 epoha.



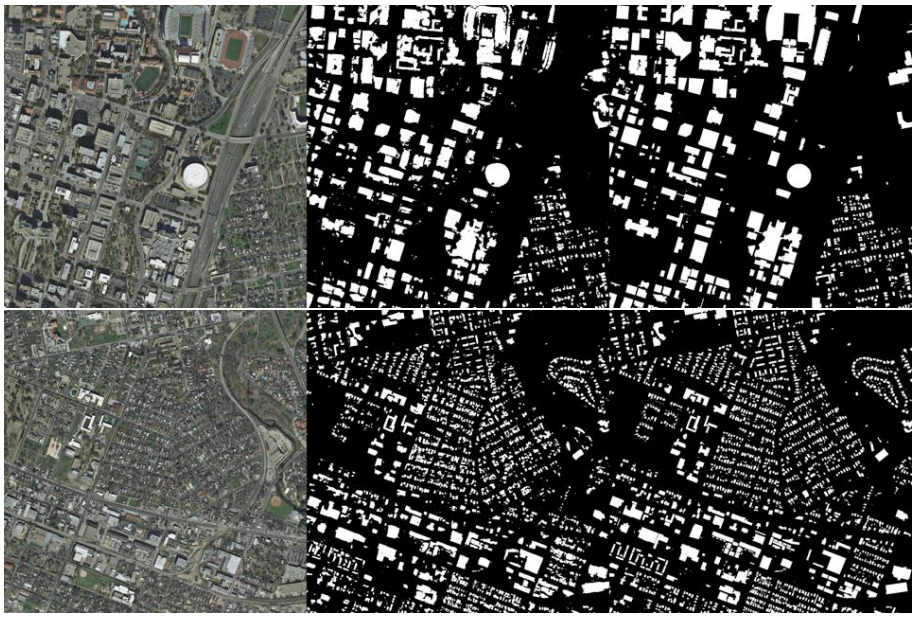
Slika 5.2: Prikaz preciznosti i odziva na skupu za validaciju kroz prvih 100 epoha.

preciznosti i odziva je nešto za očekivati. Naime, preciznost će za neku klasu biti visoka čak i ako klasificiramo mali dio piksela te klase, važno je samo da broj piksela kojima je predviđena ta klasa, a ne pripadaju toj klasi bude što manji. Odzivu s druge strane odgovara klasifikacija velikog broja piksela, za njega je važno da pokrijemo što više piksela koji stvarno pripadaju toj klasi.

Usporedbom li ovaj graf s IoU grafom zamijetit ćemo još jednu ovisnost. Crvena linija grafa preciznosti i odziva prati trend crvene linije IoU grafa, a isto vrijedi i za

plave linije. Povećanje preciznosti i smanjenje odziva u nekoj epohi odgovara većem IoU-u tla i manjem IoU-u zgrade i obrnuto. Pokušaj objašnjenja ove pojave je taj da niska preciznost, ali visok odziv neke klase povlači puno piksela klasificiranih u tu klasu, bilo točnih, bilo krivih. Budući da je klasa zgrade manje zastupljena ona ima jači utjecaj na prosječnu preciznost i odziv. Jednak broj pogrešnih piksela za zgrade više će spustiti metriku od onih za tlo. Zbog toga visok odziv znači puno piksela klasificiranih kao zgrada, a metrici IoU više paše veći broj piksela od kojih su neki krivo klasificirani, ali je također pokriven velik dio pravih zgrada u odnosu na manji broj piksela gdje nije pokriven dobar dio onih koji sadrže zgradu. U prvom slučaju za izračun IoU-a imamo veći nazivnik, a u drugom manji brojnik. Ako je brojnik manji ili jednak nazivniku, smanjenje brojnika za neki pozitivan broj a dat će manji rezultat od onog u kojem povećamo nazivnik za isti broj a. Promatranjem grafova također možemo zaključiti da se IoU ponaša stabilnije, odnosno manje oscilira.

Slika 5.3 prikazuje dva primjera originalne snimke, predviđanja modela te ispravne segmentacije za tu sliku.



Slika 5.3: Prikaz satelitske snimke, predviđanja modela i ispravne segmentacije tim redom. Snimke su dio validacijskog skupa na kojima model nije učio, a sadrže grad Austin.

Model zna imati problema sa segmentacijom manjih zgrada. Kako bih pokušao to popraviti, smanjio sam udaljenosti u optimizaciji pojačavanja funkcije gubitka na granicama kako bih još više potaknuo model da točno segmentira granice, ali to nije rezultiralo poboljšanjem. Ono što ostaje je trenirati slike na većoj rezoluciji kako bi model dobio više detalja na granicama, no to je ostavljeno za buduća poboljšanja jer

zahtijeva dodatne resurse.

6. Zaključak

U ovom radu bavio sam se problemom semantičke segmentacije, danas vrlo popularnom temom u domeni računalnog vida. Konkretno, zadatak ovog rada je provesti semantičku segmentaciju zgrada u satelitskim snimkama. Korištena arhitektura modela za učenje je SwiftNet s piramidalnom fuzijom. Iako zamišljen kao arhitektura koja omogućuje semantičku segmentaciju slika u stvarnom vremenu, SwiftNet održava visoku točnost, a uz to i brzo uči što ga čini dobrim kandidatom za semantičku segmentaciju u ovom radu. Slike i njihove maske koje sam koristio dolaze iz skupa podataka Inria.

Model sam učio na isječcima veličine 512x512 iz skupa podataka polovične rezolucije, a kao okosnica korišten je ResNet-18 model predtreniran na skupu podataka ImageNet. Učenje je pokazivalo rast točnosti modela do negdje 50. epohe gdje je točnost stagnirala uz manje oscilacije. Najveći prosječni IoU dobiven je nakon 72. epohe, a iznosi 71,0%.

Za daljnja poboljšanja točnosti prvi korak bio bi učenje na slikama pune rezolucije uz veću cijenu memorije i računalne snage. Nadalje, mogli bismo povećati veličinu isječka te veličinu mini-grupa što također zahtijeva više računalnih resursa. Osim toga, kao okosnica modela može se postaviti jači model, npr. ResNet-34.

LITERATURA

[1] Residual networks (ResNet). URL https://d2l.ai/chapter_convolutional-modern/resnet.html. Datum pristupa: 2022-06-02.

[2] Jelena Bratulić. Semantička segmentacija kolničkih trakova. 2020. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/bratulic20bs.pdf.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, i Li Fei-Fei. Imagenet: A large-scale hierarchical image database. U *2009 IEEE conference on computer vision and pattern recognition*, stranice 248–255. Ieee, 2009.

[4] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Identity mappings in deep residual networks, 2016. URL https://arxiv.org/abs/1603.05027.

[7] IBM. Convolutional neural networks, 2020. URL https://www.ibm.com/cloud/learn/convolutional-neural-networks. Datum pristupa: 2022-06-01.

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

[9] Ivan Krešo, Denis Čaušević, Josip Krapac, i Siniša Šegvić. Convolutional scale invariance for semantic segmentation. stranice 64–75, 2016.

[10] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, i Pierre Alliez. Inria aerial image labeling dataset, 2016. URL `https://project.inria.fr/aerialimagelabeling/`. Datum pristupa: 2022-06-01.

[11] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, i Pierre Alliez. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. U *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2017.

[12] Hrvoje Maligec. Semantička segmentacija zračnih slika. 2020. URL `http://www.zemris.fer.hr/~ssegvic/project/pubs/maligec20bs.pdf`.

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

Semantička segmentacija zgrada u satelitskim snimkama

Sažetak

Ovaj rad bavi se problemom semantičke segmentacije zgrada u satelitskim snimkama. Rezidualne mreže općenito su se pokazale kao dobro rješenje semantičke segmentacije. Zbog toga je za ovaj rad korištena arhitektura SwiftNet s piramidalnom fuzijom, inače stvorena za semantičku segmentaciju u stvarnom vremenu. Ta arhitektura pokazala je vrlo dobre rezultate za segmentaciju snimki u prometu, a jedna od prednosti joj je brzo učenje. Slike satelitskih snimki dolaze iz skupa podataka Inria, od čega je dio iskorišten za treniranje, a dio za validaciju modela. Sve slike smanjene su na pola rezolucije kako bih zadovoljio resursna ograničenja. U radu su objašnjeni osnovni pojmovi potrebni za shvaćanje korištenih metoda nakon čega su izneseni dobiveni rezultati.

Ključne riječi: semantička segmentacija, SwiftNet, Inria, piramidalna fuzija

Semantic segmentation of buildings in satellite imagery

Abstract

This paper deals with the problem of semantic segmentation of buildings in satellite imagery. Residual networks have generally proven to be a good solution for semantic segmentation. Therefore, the SwiftNet architecture with pyramidal fusion, otherwise created for real-time semantic segmentation, was used for this paper. This architecture has shown very good results for segmentation of traffic images, and one of its advantages is fast learning. Satellite images come from the Inria data set, part of which was used for training, and part for validating the model. All images are reduced to half resolution in order to meet computing resource constraints. The paper explains the basic terms needed to understand the methods used after which the obtained results are presented.

Keywords: semantic segmentation, SwiftNet, Inria, pyramidal fusion