SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5696

Semantička segmentacija slika dubokim konvolucijskim mrežama

Antonio Anđelić

Zagreb, lipanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 16. ožujka 2018.

ZAVRŠNI ZADATAK br. 5696

Pristupnik: **Antonio Anđelić (0036492930)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Semantička segmentacija slika dubokim konvolucijskim modelima**

Opis zadatka:

Semantička segmentacija prirodnih scena je neriješen problem računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolji rezultati u tom području postižu se pristupima utemeljenima na dubokim konvolucijskim modelima. Za ovaj rad posebno su zanimljivi strogo nadzirani pristupi gdje u svakoj piknji skupa za učenje na raspolaganju imamo informaciju o pripadnom semantičkom razredu.
U okviru rada, potrebno je istražiti postojeće pristupe iz literature za ostvarenje sementičke segmentacije. Proučiti dokumentacije programskih okvira Tensorflow i PyTorch te biblioteke programskog jezika Python za rukovanje matricama i slikama. Izraditi izvedbu programskog sustava za učenje i primjenu segmentacijskog modela. Evaluirati utjecaj rezidualnih veza na točnost modela. Prikazati i ocijeniti ostvarene rezultate.
Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:
Prof. dr. sc. Siniša Šegvić

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću na stručnim savjetima i ukazanom povjerenju.

Zahvaljujem kolegama Antoniju Borcu, Bruni Kovaču i Ivanu Šegi na diskusijama i pomoći kod izrade rada.

Hvala Nikoli, Tomislavu, Ariani i Dorotei na podršci i poticaju.

SADRŽAJ

1. Uvod **1**

2. Duboko učenje **2**

2.1. Umjetne neuronske mreže 2

2.1.1. Poveznice s biološkim neuronskim mrežama 2

2.1.2. Perceptron 3

2.1.3. Prijenosne funkcije 4

2.1.4. Arhitektura neuronske mreže 6

2.1.5. Univerzalni aproksimator 7

2.2. Učenje neuronske mreže 7

2.2.1. Funkcija gubitka 7

2.2.2. Gradijentni spust 8

2.2.3. Optimizacija i regularizacija 9

2.2.4. Propagacija pogreške unazad 11

2.3. Konvolucijska mreža 12

3. Semantička segmentacija koristeći rezidualne neuronske mreže **15**

3.1. Rezidualne neuronske mreže 15

3.2. Semantička segmentacija 16

4. Implementacija i rezultati **18**

4.1. Korištene tehnologije i detalji implementacije 18

4.2. Podaci 19

4.3. Priprema podataka 19

4.4. Arhitektura mreže 20

4.5. Rezultati 20

4.5.1. Grafički prikaz rezultata 22

4.5.2. Eksperimenti 23

5. Zaključak **27**

Literatura **28**

1. Uvod

Područje računalnog vida se sastoji od raznih problema. Za neke od tih problema rješenje je pronađeno, međutim postoje razni problemi koja još uvijek imaju prostora za znanstveni napredak. Dva osnovna problema koja su još uvijek neriješena, su klasifikacija i semantička segmentacija. Iako u suštini jednostavni problemi, već dugo godina se pokušavaju riješiti. Klasifikacija bi bila jednostavno razlučivanje što slika točno prikazuje, dok semantička segmentacija pokušava dokučiti što svaki piksel na slici predstavlja.

Prije par godina, područje računalnog vida je doseglo granicu uspješnosti u rješavanju takvih problema s tada poznatim metodama. Međutim, ono što je ponovno poguralo računalni vid je povratak neuronskih mreža, a pogotovo i njihova nadogradnja, duboke konvolucijske mreže koje su stvorile novo područje strojnog učenja koje se naziva duboko učenje. Područje računalnog vida je tim napredkom doživjelo preporod. Iz dana u dan očitovao se sve veći napredak. Svaka ideja se bazirala na već navedenoj dubokoj konvolucijskoj mreži i kroz godine su se pojavljivale razne implementacije i nove ideje kako bi se takve mreže još više poboljšale. Trenutno titulu najbolje mreže drže rezidualne mreže koje su proizvod Microsoftovog istraživačkog tima (He et al., 2015). S malom promjenom su omogućili veće dubine dubokih konvolucijskih mreža, a time se i uspješnost jako povećala za mnoge probleme računalnog vida, među kojima je i semantička segmentacija. U sklopu ovog rada, detaljnije ću obraditi navedenu mrežu, točnije njegovu iteraciju ResNet50. Uz to sam i implementirao navedenu mrežu i pokušao ju primjeniti na problem semantičke segmentacije.

Prvo poglavlje se bavi općenito područjem dubokog učenja, počevši od njegove suštine i najosnovnijih dijelova. Također, mnogi dijelovi imaju veći izbor metoda pa ću opisati neke od njih iako ih moja implementacija neće koristiti. U drugom poglavlju opisat ću dijelove ResNet mreže koju ću koristiti za svoj zadatak. Uz to, opisat ću problem koji pokušavam riješiti uz pomoć ResNet-a koristeći Tensorflow. U zadnjem poglavlju opisat ću detalje moje implementacije i prokomentirat rezultate i moguća mjesta na kojima bi se mogao napraviti napredak.

2. Duboko učenje

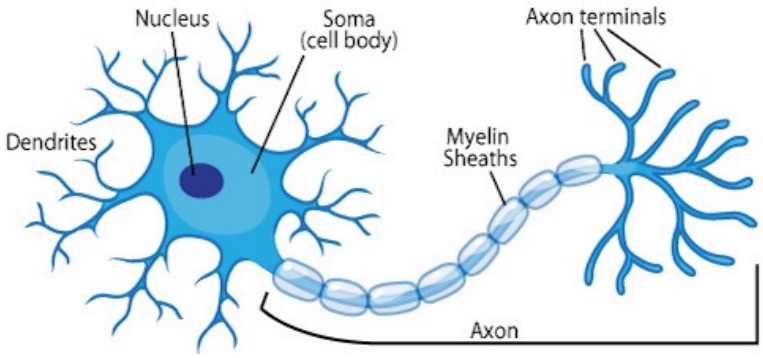
2.1. Umjetne neuronske mreže

Umjetne neuronske mreže su trenutno jedna od glavnih metoda primjenjenih na području strojnog učenja. Inspiracija za njih je bila biološka neuronska mreža, tj. umjetne neuronske mreže pokušavaju simulirati njihov način rada. Ideja za njih se pojavila već 1943. kada su Warren McCulloch i Walter Pitts definirali prvi model neuronske mreže. Međutim, bili su ograničeni tadašnjom tehnologijom pa je moć umjetnih neuronskih mreža došla do izražaja tek kada je tehnologija postigla adekvatan razvoj, a pojavili su se i mnogi napreci vezani za samu mrežu, pogotovo otkrivanjem metode propagacije pogreške unazad.

2.1.1. Poveznice s biološkim neuronskim mrežama

Iako nam mehanizam rada mozga nije u potpunosti poznat, pojavile su se razne teorije na koji način mozak obrađuje informacije. Ljudski mozak se sastoji od neurona. Neuron prikuplja signale iz susjednih, povezanih neurona kroz dendrite, a svoje neuronske impulse šalje kroz tanke, izdužene aksone koje se dijele na veći broj manjih grana. Na kraju svake grane nalazi se sinapsa s kojima neuroni ostvaruju kontakt. Neuron će prosljediti signal kroz svoje aksone u slučaju da je primljeni signale dovoljno velik. Time možemo reći da sinapse određuju koliko jedan neuron utječe na drugi.

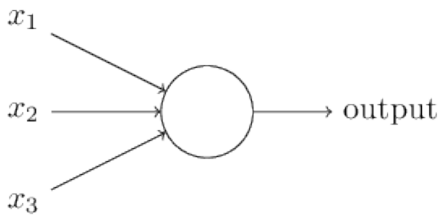
Umjetne neuronske mreže u velikoj mjeri pojednostavljuju opisani biološki proces. Također se sastoji od neurona koji su međusobno povezani. Umjetni neuron može primiti ulazne signale, a njegov izlaz je definiran raznim matematičkim funkcijama. Tako najosnovniji neuroni jednostavne sumiraju sve ulaze i ovisno o pragu, šalju izlazni signal. Jedan od opisanih, jednostavnih neurona je perceptron.



Slika 2.1: Biološki neuron

2.1.2. Perceptron

Perceptroni su razvijeni od strane Franka Rosenblatta. Iako danas postoje bolji modeli umjetnih neurona, zanimljivo je vidjeti koje mane perceptrona su noviji modeli pokušali ispraviti.



Slika 2.2: Perceptron

Perceptron prima više binarnih ulaza $x_1, x_2, \ldots$, i proizvodi binarni izlaz. Kako bi se odredio izlaz, Rosenblatt je uveo i težine $w_1, w_2, \ldots$ kojima je definirano odnos između svakog neurona. Težine su realni brojevi koji nam govore koliko jedan neuron utječe na drugog. Tako je izlaz svakog neurona definiran po tome je li suma $\sum_{k=1}^{n} w_k x_k$ manja od nekog zadanog praga. Algebarski zapis izlaza perceptrona, uz to da je prag također realan broj, bio bi:

$$\text{izlaz} = \begin{cases} 0 & \text{ako } \sum_j w_j x_j \leq \text{prag} \\ 1 & \text{ako } \sum_j w_j x_j > \text{prag} \end{cases} \tag{2.1}$$

Perceptroni koji se nalaze u istom stupcu neuronske mreže stvaraju jedan sloj. Svaki perceptron donosi odluku na temelju izlaza prethodnog sloja i definiranih težina za svaki od tih perceptrona.

Gore izvedenu formulu možemo još malo urediti tako da prag prebacimo na lijevu stranu i definiramo novu varijablu $b = -\text{prag}$, koja označava pristranost. U kontekstu

neuronske mreže, pristranost označava koliko je lako natjerati perceptron da pošalje 1 kao izlaz. Ako je pristranost neki ogroman broj, perceptron će skoro uvijek na izlazu imati jedan, dok za jako negativan broj vrijedi obrnuto. Time smo napravili prvu izmjenu perceptrona i trenutno je izlaz definiran kao:

$$\text{izlaz} = \begin{cases} 0 & \text{ako } \sum_j w_j x_j + b \leq 0 \\ 1 & \text{ako } \sum_j w_j x_j + b > 0 \end{cases} \tag{2.2}$$

Nadalje, linearnu funkciju izlaza koja je trenutno zapisana sumom, možemo prikazati u matričnom obliku: $f = w^T x + b$.

2.1.3. Prijenosne funkcije

Kako bi bolje modelirali izlaz neurona, koristimo razne prijenosne funkcije (ili nelinearnosti). Ako bi koristili gore definiranu linearnu funkciju izlaza, sitna promjena u ulazu bi mogla potpuno promjeniti izlaz. Jedan od ciljeva prijenosne funkcije je omogućiti primjećivanje sitnih promjena na ulazu, tj. u kojoj mjeri novi ulaz utječe na izlaz neurona. Aktivacijske funkcije na ulazu primaju jedan broj nad kojim se primnjuju razne matematičke operacije. Tako je svaki neuron definiran kao:

$$a^k = \sigma_k(W_k^T a^{k-1} + b) \tag{2.3}$$

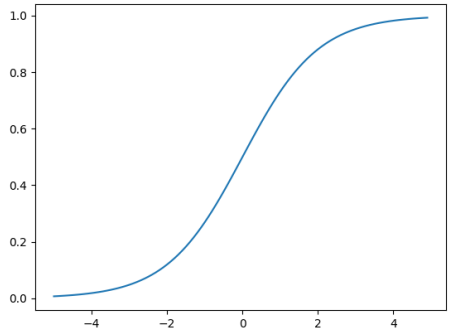
gdje σ predstavlja aktivacijsku funkciju, a k sloj kojem neuron pripada.

Postoje više prijenosnih funkcija koja su trenutno u uporabi. Prva od njih je sigmoidalna funkcija.

Sigmoidalna funkcija je prijenosna funkcija oblika $\sigma(x) = \frac{1}{1+e^{-x}}$. Glavno obilježje sigmoidalne funkcije je njeno prebacivanje bilo kojeg broja na interval od 0 do 1. Tako jako veliki pozitivni brojevi poprimaju vrijednost 1, dok jako negativni brojevi poprimaju vrijednost 0. U početku je bila često korištena prijenosna funkcija zbog dobre interpretacije izlaza neurona koji je prikazan intervalom od 0, kada neuron nikad ne vraća izlaz, do 1, kada neuron daje izlaz njegovom pretpostavljenom maksimalnom frekvencijom. Međutim, kasnije se sigmoidalna funkcija sve manje koristila zbog dva velika nedostatka:

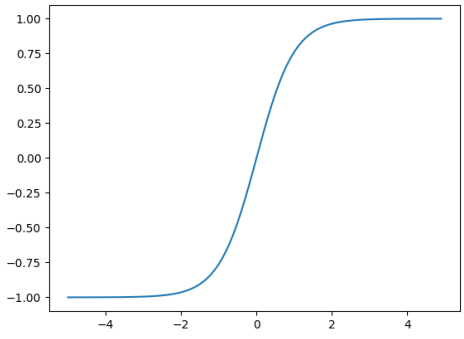
- Prvi nedostatak je u slučaju zasićenja izlaza neurona na krajevima (0 ili 1) gdje gradijent poprima vrijednost koja je jako blizu nuli. Pošto, mreža uči uz pomoć lokalnih gradijenata, takvi slučajevi u potpunosti prekinu protok podataka. Zbog toga se ni težine ne smiju inicijalizirati na jako velike brojeve.

– Izlazi sigmoidalne funkcije također nisu centrirani oko 0. To svojstvo također utječe na računanje gradijenta jer će u slučaju svih pozitivnih ulaza, gradijent težina w biti potpuno pozitivan ili potpuno negativan, ovisno o gradijentu cijelog izraza.



Slika 2.3: Sigmoidalna funkcija

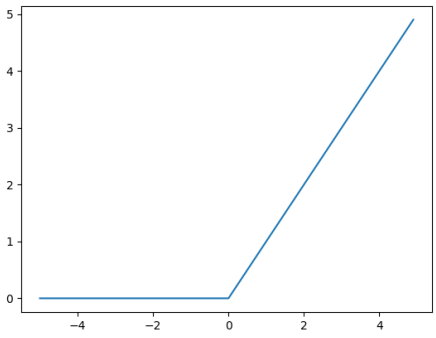
Tanh je sljedeća prijenosna funkcija koja također bilo koji ulaz smjesti na određen interval, u ovom slučaju na interval $[-1, 1]$. Prisutan je i prvi nedostatak sigmoidalne funkcije kod zasićenja neurona, međutim izlazi $tanh$ nelinearnosti su centrirani oko 0. Zbog tog je uvijek poželjnije koristiti $tanh$ nego sigmoidalnu funkciju. Također je pristutan i odnos između tih dviju prijenosnih funkcija u obliku: $tanh(x) = 2\sigma(2x) - 1$.



Slika 2.4: Tanh

ReLU (*Recitified Linear Unit*) je u zadnjih par godina najpopularnija prijenosna funkcija. Na temelju ulaza računa funkciju $f(x) = max(0, x)$. Drugim riječima,

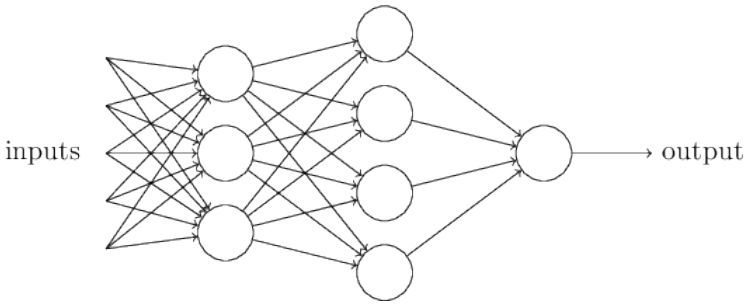
propušta ulaze koji su veći od 0. Glavna prednost ReLU funkcije je njena brzina računanja i uz to ubrazava konvergenciju stohastičkog gradijenta. Međutim, neuroni koji koriste ReLU prijenosnu funkciju mogu "odumrijeti". Prevelik gradijent može izmjeniti težine na takav način da se neuron više nikad ne aktivira. Uz pravilno postavljanje hiperparametara, pogotovo korak učenja, taj problem se može izbjeći.



Slika 2.5: ReLU

2.1.4. Arhitektura neuronske mreže

Glavni dijelovi neuronske mreže su slojevi. Slojevi su jednostavno skup neurona koji se nalaza u istom stupcu. Prvi sloj je ulazni, a zadnji sloj je izlazni. Svaki sloj između ta dva se naziva *skriveni sloj*. Najčešće mreže su one kod kojih izlaz jednog sloja postaje ulaz sljedećeg sloja. Takve mreže se nazivaju unaprijedne (engl. *feedforward*) mreže. Naravno, postoje i mreže koje nisu unaprijedne, i nazivaju se povratne neuronske mreže (*Recurrent neural networks*).



Slika 2.6: Primjer unaprijedne neuronske mreže

2.1.5. Univerzalni aproksimator

Teorem univerzalnog aproksimatora kaže da aciklička neuronska mreža s jednim skrivenim slojem, konačnog broja neurona, može aproksimirati bilo koju funkciju, uz neka ograničenja na prijenosnu funkciju.

2.2. Učenje neuronske mreže

Neuronska mreža se sastoji od većeg broja težina i varijabli koje označavaju pristranost. Učenje mreže je izmjena tih varijabli kako bi mreža predstavljala model koji najbolje opisuje te podatke. Najčešći način na koji mreža uči je koristeći funkiju gubitka i računajući njen gradijent.

2.2.1. Funkcija gubitka

Kako bi procjenili koliko naš model dobro procjenjuje naše podatke, koristimo funkciju koja predstavlja koliko model odstupa od željenih rezultata. Ako naš model loše procjenjuje, gubitak će biti veći. Kao i za prijenosne funkcije postoji više opcija za funkciju gubitka.

Stroj s potpornim vektorima (engl. *Support Vector Machine - SVM*) je prva od popularnijih funkcija gubitaka. Cilj SVM-a je dobiti zadanu razliku između izračunatih vrijednosti za točne razrede i ostalih razreda. Vektor izračunatih vrijednosti s za svaki razred ulaza x_i, i točnim razredom y_i uvrštavamo u funkciju gubitka:

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta). \tag{2.4}$$

Δ je hiperparametar koji se najčešće postavlja na vrijednost 1.0. Vrijednost L se računa kao srednja vrijednost gubitaka za svaki ulaz.

Softmax je još jedan popularan klasifikator. Za svoju funkciju gubitka koristi gubitak unakrsne entropije oblika:

$$L_i = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \tag{2.5}$$

, što je ekvivalento

$$L_i = -f_{y_i} + log \sum_j e^{f_j} \tag{2.6}$$

, gdje f_j predstavlja j-ti element vektora rezultate za svaki razred. Funkcija

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.7}$$

se zove softmax funkcija koja bilo kakav vektor realnih izračunatih vrijednosti z pretvara u vektor kojem su elementi na intervalu $[0, 1]$, a njihov zbroj daje 1. Funkcija gubitka se može prikazati i preko vjerojatnosti:

$$P(y_i|x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \tag{2.8}$$

Na temelju tog zapisa možemo vidjeti da koristeći softmax klasifikator nastojimo maksimizirati vjerojatnost točnog razreda, tj. minimizirati negativnu log vrijednost. Kao i kod stroja potpornih vektora, gubitak L se računa kao srednja vrijednosti izračunatih gubitaka za svaki ulaz.

2.2.2. Gradijentni spust

Nakon što smo definirali funkciju gubitka, cilj neuronske mreže je minimizirati izračunati gubitak. Kako bi varijable mijenjali u smjeru minimuma funkcije gubitka, a da u isto vrijeme pomak bude najveći, računamo gradijent funkcije gubitka. Ako je $L(v)$ funkcija koju pokušavamo minimizirati, a ∇L gradijent te funkcije, promjenu funkcije možemo zapisati kao:

$$\Delta L \approx \nabla L \cdot \Delta v \tag{2.9}$$

Uz to definiramo i:

$$\Delta v = -\eta \nabla L \tag{2.10}$$

gdje η predstavlja mali, pozitivan parametar (korak učenja) dobivamo
$\Delta L \approx -\eta \nabla L \cdot \nabla L = -\eta \|\nabla L\|^2$. Kako je $\|\nabla L\|^2 \geq 0$, sigurno će $\Delta L \leq 0$ što znači da će se L uvijek smanjiti.

U neuronskoj mreži postoje dvije varijable koje želimo mijenjati tako da se funkcija gubitka minimizira, težine w i pristranosti b. Varijable se mijenjaju izrazima:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial L}{\partial w_k} \tag{2.11}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial L}{\partial b_l} \tag{2.12}$$

Primjenjujući navedene izraze, polako ćemo se micati prema minimumu funkcije gubitka.

Međutim, kako ne bi računali gradijent za sve podatke odjednom, postupak možemo ubrzati tako da računamo gradijent samo za uzorak ulaznih podataka ∇C_x. Taj način gradijentnog spusta se naziva stohastički gradijentni spust. Dio podataka koje koristimo za treniranje nazivamo grupe (engl. *batch*). Prolazak kroz sve podatke za treniranje zovemo epoha. Ako se grupa sastoji od m podataka $x_1, x_2, ..., x_m$, formula za gradijentni spust prelazi u:

$$w_k \to= w'_k - \frac{\eta}{m} \sum_j \frac{\partial L_{x_j}}{\partial w_k} \tag{2.13}$$

$$b_l \to= b'_l - \frac{\eta}{m} \sum_j \frac{\partial L_{x_j}}{\partial b_l} \tag{2.14}$$

2.2.3. Optimizacija i regularizacija

Iako se gradijentnim spustom mičemo u u smjeru minimuma, postupak se može ubrzati raznim optimizacijskim metodama. Uz to, model treniramo na temelju podataka za treniranje, ali bi htjeli da može pravilno zaključivati za bilo kakve podatke. Kod učenja može doći do prenaučenosti (engl. *overfitting*) gdje model loše generalizira jer se previše prilagodio podacima za treniranje. Zbog toga je potrebno uvesti i razne regularizatore koji pokušavaju spriječiti pojavu prenaučenosti.

Optimizacija

Računanje gradijenata je bitan dio učenja. Kako bi model dosegao brže minimum postoje razni oblici optimizacije micanja po funkciji prema minimumu.

Momentum je jedan od prvih oblika optimizacije:

$$\begin{aligned} v &= mu \cdot v - \eta \cdot dx \\ x &= x + v \end{aligned} \tag{2.15}$$

Nova varijabla v predstavlja brzinu micanja prema lokalnom minimumu, a na početku se inicijalizira na 0. Što je veći gradijent, brzina će se povećati, a kako bi se pomicanje zaustavilo na dnu dodaje se varijabla mu koja predstavlja momentum koja se najčešće inicijalizira na 0.9.

Adam je jedan od novijih optimizatora. Definiran je kao:

$$\begin{aligned} m &= \beta_1 \cdot m + (1 - \beta_1) \cdot dx \\ v &= \beta_2 \cdot v + (1 - \beta_2) \cdot dx^2 \\ x &= x - \frac{\eta \cdot m}{\sqrt{v} + \epsilon} \end{aligned} \tag{2.16}$$

Trenutno je preporučeni optimizitor. Kombinira prijašnje ideje među kojima su momentum i RMSProp. Lagano je za implementirati, nije računski zahtjevna, ne zauzima previše memorije i postiže dobre rezultate na većem broju podataka i parametara.

Regularizacija

Kako bi izbjegli prenaučenost, moramo dodati neki oblik regularizacija.

L1 i L2 Jedan od načina je dodavanje funkciji gubitka funkciju regularizacija. Tako funkcija gubitka poprima oblik $L(X, Y) + \lambda N(w)$. Najčešće funkcije regularizacije su *L1* i *L2* norme:

$$L1 = \sum_{j=1} |w| \tag{2.17}$$

$$L2 = \sum_{j=1} w_j^2 \tag{2.18}$$

λ je u ovom slučaju hiperparametar. L1 ima svojstvo postavljanja težina manje bitnih značajki na 0, a neke značajke mogu potpuno nestati.

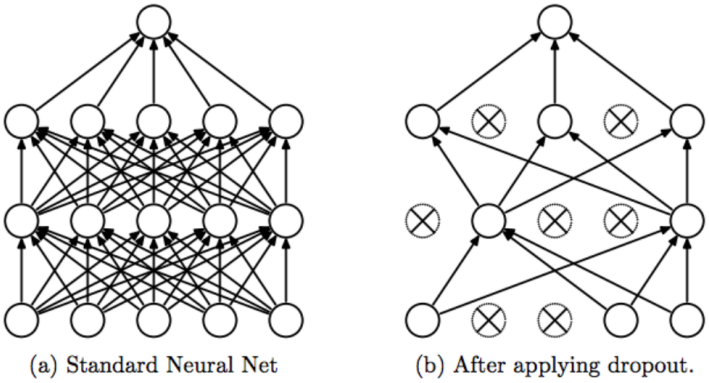
Droupout je još jedan oblik regularizacije. S dropoutom svaki neuron s određenom vjerojatnošću postaje isključen za vrijeme treniranja prilikom prolaska podataka kroz mrežu ili propagacije pogreške unazad. S takvim oblikom regularizacije, nastoji se pojačat nezavisnost svakog neurona. Za vrijeme testiranja modela, svaki se izlaz mora smanjiti kako bi se nadoknadilo izbacivanje pojedinih neurona za vrijeme treniranja.

Normalizacija grupe (engl. *batch norm*) se nedavno pojavila kao metoda optimizacije, međutim dodaje i oblik regularizacije unutar mreže. Normalizacija grupe se temelji na normalizaciji izlaza prije primjene nelinearnosti. Ako definiramo z^k kao:

$$z^k = W_k^T a^{k-1} + b \tag{2.19}$$

, gdje a^k predstavlja izlaz neurona, z^k se prvo normalizira:

$$z^k = \frac{z^k - E[z^k]}{\sqrt{V[z^k]}} \tag{2.20}$$



Slika 2.7: Dropout regularizacija

gdje $E[z^k]$ predstavlja prvi moment, a $V[z^k]$ drugi moment. Nelinearnost se onda primjenjuje na tako normaliziranom izlazu. Kako svaki sloj ovisi o prethodnom sloju, promjena prethodnog sloja mijenja distribuciju ulaza u sljedeći sloj. Zbog tog, pogotovo u dubljim modelima, dulje treba kako bi funkcija gubitka konvergirala. Dodavanjem normalizacije grupe, definirali smo da je ulaz svakog sloja normalne distribucije, tako promjene na prethodnom sloju neće umanjiti vrijednost promjena na sloju koji slijedi. Time smo dobili ubrzanje u samom treniranju. Normalizacija za vrijeme treniranja koristi srednju vrijednost i standardnu devijaciju grupe, a kako su grupe svaku epohu različite, model nikad neće trenirati na istim podacima zbog čega nastaje regularizacijski efekt.

2.2.4. Propagacija pogreške unazad

Učenje neuronske mreže uvelike ovisi o računanju gradijenata. Tako je za najveći napredak neuronskih mreže zaslužno upravo otkriće algoritma koji ubrzava taj proces. Algoritam propagacije pogreške unazad pojavio se prvi put 1970. Ubrzanje je bilo toliko veliko da je omoućilo korištenje neuronskih mreža za probleme koje su prije bilo nemogući za rješiti njima, a danas je glavni dio svake neuronske mreže.

Kako bi propagacija pogreške funkcionirala, moramo pretpostaviti dvije stvari za funkciju gubitka:

- Funkcija gubitka se može zapisati kao srednja vrijednost gubitaka L_x, tj. gubitaka za svaki pojedini ulaz. Algoritam propagacije unatrag računa $\partial L_x/\partial w$ i $\partial L_x/\partial b$, dok se $\partial L/\partial w$ i $\partial L/\partial b$ računaju kao srednja vrijednost prethodnih izraza.
- Funkcija gubitka se može zapisati kao funkcija izlaza neuronske mreže: $L = L(a^k)$. Kako ne možemo mijenjati ulaze x i željene izlaze y, želimo definirati

funkciju gubitka s izlazima neuronske mreže a^k koje možemo prilagođavati mijenjajući parametre w i b, tj. funkciju gubitka smo definirali s nečim što neuronska mreža može naučiti.

Propagacija pogreške unazad koristi 4 osnovne funkcije. Kako bi došli do njih prvo definiramo pogrešku j-tog neurona u k-tom sloju:

$$\delta_j^k \equiv \frac{\partial L}{\partial z_j^k} \tag{2.21}$$

a z_j^k predstavlja izlaz neurona prije nelinearnosti. Prva osnovna funkcija je računanje pogrešaka δ:

$$\delta_j^k = \frac{\partial L}{\partial a_j^k}\sigma'(z_j^k) \tag{2.22}$$

Tako definiranom formulom računamo koliko gubitak ovisi o pojedinom izlazu neurona, dok izraz $\sigma'(z_j^k)$ govori koliko brzo se prijenosna funkcija mijenja.

Druga formula nam govori kako možemo pogrešku δ izračunati koristeći pogrešku sljedećeg sloja:

$$\delta^k = ((w^{k+1})^T\delta^{k+1}) \odot \sigma'(z^k) \tag{2.23}$$

gdje $\odot$ predstavlja umnožak matrica gdje se elementi na jednakim mjestima pomnože.

Trenutne dvije formule nam omogućuju računanje pogreške δ za bilo koji sloj mreže. Međutim cilj je izračunati gradijent s obzirom na w i b. Gradijent s obzirom na pristranost iznosi upravo izračunatoj pogrešci:

$$\frac{\partial L}{\partial b_j^k} = \delta_j^k \tag{2.24}$$

Gradijent s obzirom na težine je također vrlo jednostavan i koristi već izračunate vrijednosti, pogrešku δ i izlaz neurona nakon primjenjene nelinearnosti a:

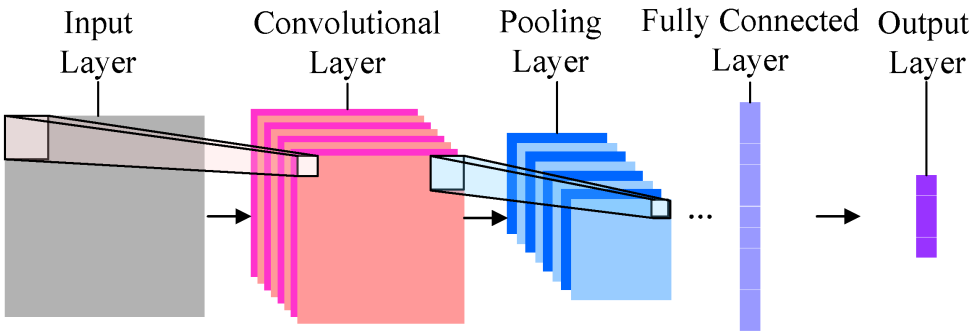
$$\frac{\partial L}{\partial w_{jl}^k} = a_l^{k-1}\delta_j^k \tag{2.25}$$

Time smo definirali algoritam propagacije pogreške unazad. Koristi vrlo jednostavne matematičke operacije, a iz prve formule je vidljivo zašto je bitno dobro definirati prijenosnu funkciju. Algoritam ovisi o derivaciji prijenosne funkcije pa loša svojstva, kao što su ravna mjesta gdje derivacija funkcije postaje približno nula, uvelike utječu na samo učenje mreže.

2.3. Konvolucijska mreža

Jedno od područja na koje se neuronske mreže u velikoj mjeri primjenjuju je područje računalnog vida. Osnovne neuronske nemaju nikakve pretpostavke o svojim po-

dacima, međutim pojavile su se konvolucjske mreže koje su posebno prilagođene za učenje nad slikama.



Slika 2.8: Prikaz konvolucijske mreže

Slike u sebi sadrže puno informacija pa tako nastaju problemi i kod onih manjih dimenzija. Prethodno definirane neuronske mreže imaju potpuno povezane neurone, što znači da bi za svaku sliku dimenzija HxWxD imali ulaz veličine $H \cdot W \cdot D$ i svaki od neurona u ulaznom sloju će biti povezan sa svakim neuronom sljedećeg sloja. Time smo već na početku uveli ogroman broj varijabli. Konvolucijske mreže svoje neurone slažu u 3 dimenzije - broj dimenzija koje definiraju slike.

Konvolucijske mreže se sastoje od 3 glavna tipa sloja:

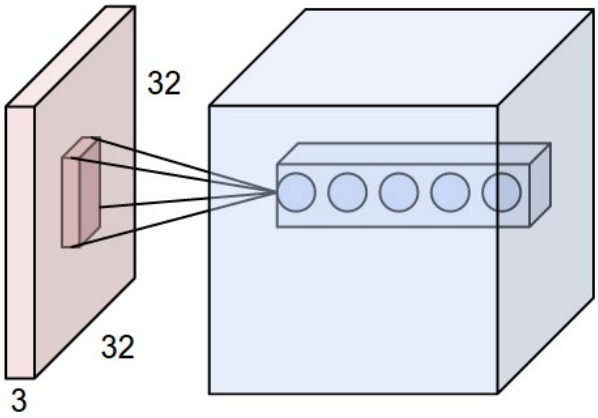
- konvolucijski sloj
- sloj sažimanja
- potpuno povezani sloj

gdje je potpuno povezani sloj onaj koji smo koristili u osnovnoj neuronskoj mreži.

Konvolucijski sloj je najbitniji dio konvolucijske mreže. Jedan konvolucijski sloj se sastoji od nekoliko filtera koje imaju mogućnost učenja. Manjih su širina i dužina, ali potpuno pokrivaju dubinu ulaza. Kod prolaska kroz mrežu, prolazimo filterom širinom i dužinom ulaza sve dok ne prođemo kroz cijeli ulaz. Prilikom prolaska, jednostavno računamo skalarni umnožak između vektora na istim pozicijama nakon čega umnoške samo zbrojima. Uloga filtera je uočavanje i učenje određenih uzoraka na slikama. Više filtera slažemo također u dubinu i tako nam je izlaz definiran s 3 dimenzije.

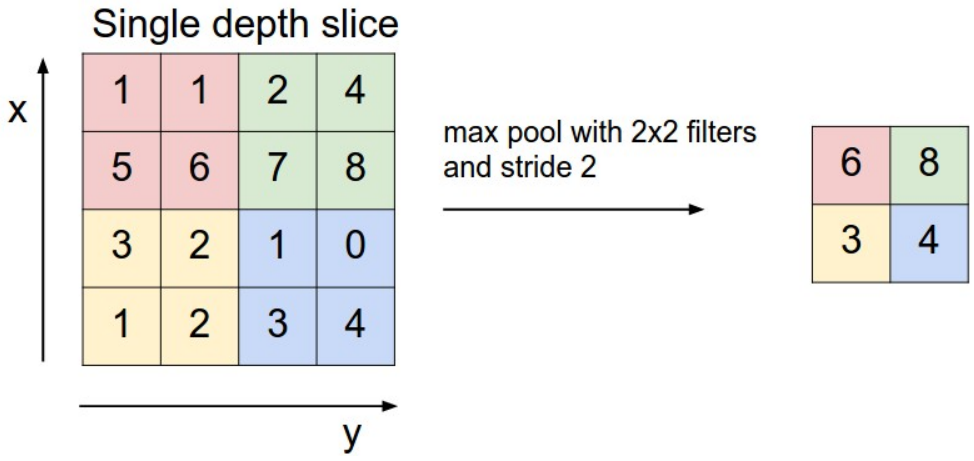
Kao što je već navedeno, kod slika je velik broj ulaznih informacija jedna od glavnih problema. Stoga je problematično tražiti povezanost između svakog neurona pa sa filterom pokušavamo, ovisno o njegovoj dimenziji, gledati odnose bliskih neurona.

Filter definiraju 3 hiperparametra: dubina (engl. *depth*), korak (engl. *stride*) i popunjavanje nulama (engl. *zero-padding*).



Slika 2.9: Konvolucijski sloj

Sloj sažimanja nije ključan dio svake konvolucijske mreže, ali je često korišten. Kao što mu ime kaže, slojem sažimanja nastojimo smanjiti dimenzije podataka kako bi time smanjili broj parametara i operacija. Također se primjenjuje na svakoj dubini neza-visno o ostalim primjenjujući operacije kao što su najveći broj ili srednju vrijednost. Sloj sažimanja također koristi filtere koji su istog oblika kao i filteri konvolucijskog sloja, a isto je i njihovo ponašanje.



Slika 2.10: Sloj sažimanja

3. Semantička segmentacija koristeći rezidualne neuronske mreže

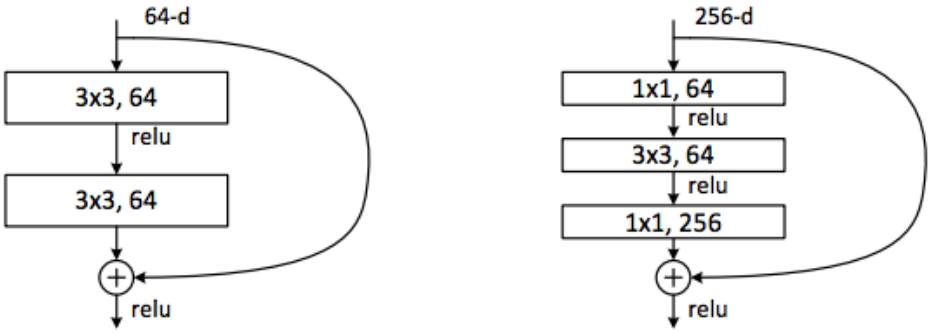
U zadnje vrijeme pojavio broj arhitektura neuronskih mreža. Trenutno najbolje rezultate pokazuje ResNet (engl. *residual neural network*) koji je sa svojim rezidualnim vezama omogućio stvaranje neuronskih mreža s jako velikim brojem slojeva. Cilj je bila primjena varijacije takve mreže na problem semantičke segmentacije.

3.1. Rezidualne neuronske mreže

Kod razvoja neuronskih mreža, bolji rezultati su se postizali dodavanjem više skrivenih slojeva. Međutim, nakon određenog broja rezultati su se pogoršavali. Jedan od faktora je bila pojava nestajućih gradijenata. Kako propagacija pogreške unazad koristi lančano pravilo, gradijenti bi nakon nekog vremena težili u 0 što spriječava daljnje učenje mreže. Iako se to smatrao glavni razlog negativnog učinka dodavanja više slojeva, eksperimentalno je dokazano da postoje još neutvrđenih uzroka takve pojave.

Kao pokušaj rješavanja takvog problema, Microsoftov istraživački tim (He et al., 2015) je razvio posebnu vrstu blokova nazvanih rezidualni blokovi. Rezidualni blokovi koriste takozvane "prečice". Ulaz rezidualnog bloka se grana gdje se jedna grana sastoji od nekoliko konvolucijskih slojeva dok je druga grana prečica do kraja rezidualnog bloka gdje se dvije grane spajaju prije primjene nelinearnosti. Izlazi dviju grana se jednostavno zbroje i na takvu sumu se primjenjuje nelinearnost, čime smo dobili izlaz rezidualnog bloka. Grana koja predstavlja prečicu u ovom slučaju koristi funkciju identiteta, tj. jednostavno preslikava ulaz, ali su moguće i razne druge funkcije. Uz to, kako se dvije grane zbrajaju, dimenzije im se moraju podudarati.

Dodatno, u svojoj implementaciji su prije primjene svake nelinearnosti koristili normalizaciju grupe, a nije prisutan *dropout* kao ni bilo koji drugi oblik regularizacije. Nakon određenog broja rezidualnih blokova, primjenjuje se globalni sloj sažimanja koji računa srednju vrijednost iza čega slijedi potpuno povezani sloj.



Slika 3.1: Lijevo: klasični reziudalni blok; **Desno**: rezidualni blok s uskim grlom

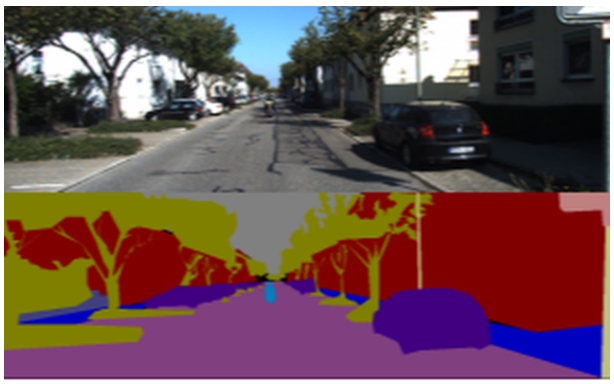
Uvođenjem rezidualnih blokova dodavanje dodatnih slojeva samo pospješuje rezultate neuronske mreže. Tako je broj slojeva mreža skočio s 19 na 150, a postoje i implementacije koje se sastoje od 1000 slojeva.

S povećanjem broja slojeva, raste i broj parametara u mreži. Kako bi taj broj ostao na normalnim razinama, u mrežama koje imaju 50 ili više slojeva koriste se blokovi s uskim grlom (engl. *bottleneck*). Također, blokovi s uskim grlom sprječavaju prenaučenost. Klasični rezidualni blok se sastoji od 2 $3x3$ konvolucije, dok blok uskog grla koristi samo jednu takvu konvoluciju, dok se ispred i nakon nje dodaju $1x1$ konvolucije s kojima se prvo smanji broj filtera, a nakon toga vrati broj filtera na početni kako bi se dimenzije podudarale prilikom zbrajanje u rezidualnom bloku. Uz blokove s uskim grlom smanjujemo broj parametara za učenje, time je zauzeće memorije manje i sprječava se prenaučenost.

Naravno, ResNet je postigao odlične rezultate na raznim natjecanjima čime je pokazao svoju učinkovitost.

3.2. Semantička segmentacija

Jedan od najučestalijih problema računalnog vida je semantička segmentacija. Problem koji semantička segmentacija pokušava riješiti je što svaki dio slike prikazuje. Koristeći duboko učenje taj problem rješavamo klasificiranjem svakog piksela. ResNet je pokazao dobre rezultate u tom području što ovaj rad pokušava dokazati.



Slika 3.2: Primjer semantičke segmentacije

4. Implementacija i rezultati

4.1. Korištene tehnologije i detalji implementacije

Za izradu ovog rada, korišten je programski jezik Python[1] pa tako i njegove biblioteke i radna okruženja. Veći dio programa je napisan koristeći Tensorflow[2] radno okruženje. Uz to je korištena Numpy[3] biblioteka za učitavanje i transformaciju podataka, kao i skimage[4] biblioteka za učitavanje i spremanje slika. Također su korištene biblioteke Pandas[5] i Seaborn[6] za vizualizaciju rezultata.

Korišten je Tensorflowov API više razine koji se sastoji od dva bitna dijela: Estimatora i Dataset-a.

Estimator se koristi za treniranje, evaluaciju i predikciju na temelju napisanog modela i ulaznih podataka. Kako bi se mogli koristiti Estimatori potrebno je definirati dvije funkcije: funkciju koja opisuje model i funkciju koja definira dohvaćanje podataka.

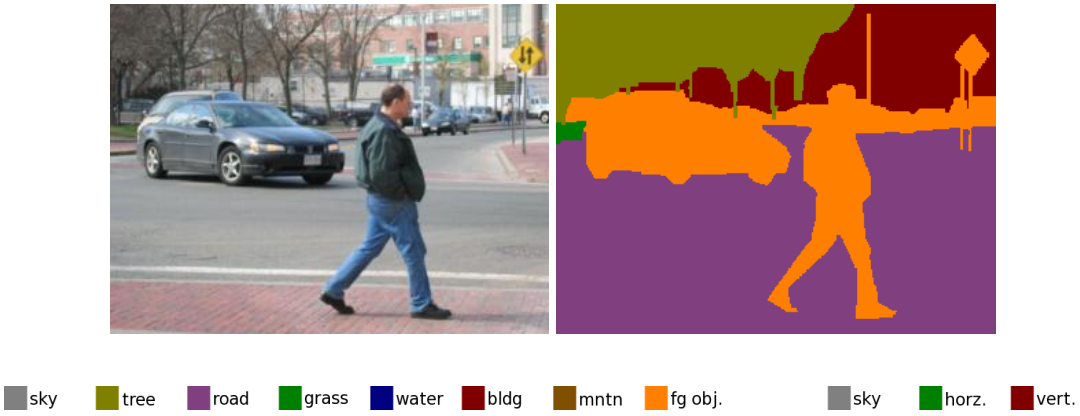
Funkcija koja dohvaća podatka vraća Dataset. Dataset je poseban tip podataka uz pomoć kojeg se definiraju sve stavke vezane za podatke kao što su ponavljanje i grupiranje.

Gubitak je izračunat koriteći softmax s unakrsnom entropijom. Kao optimizitor je korišten Adam s pretpostavljenim postavkama. Za normalizaciju grupe su također ostavljene pretpostavljene postavke.

[1] https://www.python.org/
[2] https://www.tensorflow.org/
[3] http://www.numpy.org/
[4] http://scikit-image.org/docs/dev/api/skimage.html
[5] https://pandas.pydata.org/
[6] https://seaborn.pydata.org/

4.2. Podaci

Za testiranje ResNet-a na semantičkoj segmentaciji koristio sam *stanford background dataset*[7]. Skup se sastoji od 715 probranih slika iz više javno dostupnih izvora kao što su LabelMe i PASCAL. Za svaku slike je definirano više stvari, ali što je najbitnije za semantičku segmentaciju, svaki piksel je klasificiran u jednu od 9 klasa - nebo, stablo, cesta, trava, voda, građevina, planina, objekt u prednjem planu i nedefinirane. Također, slike su varijabilne veličine gdje su maksimalna širina i visina 320 piksela.



Slika 4.1: Primjer ulaza i klasifikacije podataka iz *stanford background dataset-a*

4.3. Priprema podataka

Za učenje modela koristio sam nadzirano učenje (engl. *supervised learning*). Kod nadziranog učenja se podaci sastoji od ulaza i željenih izlaza. Uspješnost modela procjenjujemo s time koliko su dobiveni izlazi bliski željenim izlazima.

Prvo sam podijelio podatke u dva skupa. Prvi veći skup koristim za treniranje modela, dok drugi, testni skup koristim za ocjenu istreniranog modela. Nadalje, prvi skup za treniranje sam podijelio na skup podataka za treniranje i na skup za validaciju kako bi uočio pojavu prenaučenosti. 64% podataka je slučajnim odabirom pripalo skupu za učenje, 16% skupu za validaciju i 20% skupu za testiranje također slučajnim odabirom.

Kako radim s jako malim skupom za treniranje, generirao sam još slika primjenom nekoliko tehnika transformacije na postojeće slike koje pripadaju skupu za treniranje:

– slike su izrezane na slučajno odabranim mjestima

[7] http://dags.stanford.edu/projects/scenedataset.html

– s 50% šanse su vertikalno i horizontalno preokrenute

– svjetlina je promjenjena za Δ iz $[-0.2, 0.2]$

Postupak sam za svaku sliku ponovio 9 puta nakon čega sam dobio 4113 novih slika. Uz to, kako bi mogao koristiti grupno učenje (engl. *batch*), morao sam popuniti sve slike do HxW gdje su H i W maksimalna visina, odnosno širina.

4.4. Arhitektura mreže

Za model sam izabrao ResNet50, mreža koja koristi rezidualne blokove i sastoji se od 50 slojeva.

Veći dio implementacije se drži implementacije iz rada Kaiming He (He et al., 2015). Međutim, kako su slike iz korištenog skupa podataka varijabilne veličine, dodao sam manje preinake u graf. Kraj mreže koji se sastoji od globalnog sloja sažimanja i potpuno povezanog sloja zamjenio sam s još jednim konvolucijskim slojem gdje su filter i korak veličine 1, a broj filtera odgovara broju klasa.

Kako bi na kraju mreže dobio rezultate za svaki piksel, izlaz mora biti iste dimenzije kao i slika. Zbog toga sam nakon zadnjeg konvolucijskog sloja dodao i bilinearno naduzorkavanje koji podatke bilo kojih dimenzija transformira u zadane dimenzije.

Mreža ne koristi nikakav oblik regularizacije nego se u potpunosti oslanja na regularizacijski efekt normalizacije grupe i bloka s uskim grlom, zbog čega je grupno učenje izrazito bitno kod korištenja ove mreže.

4.5. Rezultati

Rezultate sam uspoređivao s rezultatima prijašnjih radova koji su koristili isti skup podataka, a među njima su prisutni rezultati kolega Ivana Grubišića (Grubišić, 2016) i Ivana Borka (Borko, 2015). Korištena je točnost kao procjena uspješnosti mreže. Točnost se jednostavno računa kao omjer točno klasificiranih podataka i ukupnog broja podataka:

$$\text{točnost} = \frac{\text{točno klasificirani pikseli}}{\text{ukupan broj piksela}} \tag{4.1}$$

Početna stopa učenja je postavljena na 10^{-3}, a nakon toga je svakih 50 epoha umanjena 10 puta.

Treniranje mreže je prvo provedeno samo na podacima koji nisu dobiveni transformacijom. Treniranje se sastojalo od 100 epoha, čime je dobivena točnost od 73.1%. Nakon tog je treniranje ponovljeno, ali u skup za treniranje su dodane generirane slike.

Nakon 50 epoha, postignuta je točnost od 74.5%. Na kraju sam spojio skup za treniranje i validacijski skup i time dobio najveću točnost od 75.5%.

Sustav	Točnost piksela
Farabet et al. 2013.	0.788
Farabet et al. + superpikseli 2013.	0.804
Mohan 2014.	0.842
Borko 2015.	0.742
Konv. mreža s 3 razine	0.743
Konv. mreža s 3 razine + dropout	0.757
Konv. mreža s 3 razine + dropout + SLIC	0.756
ResNet50	0.731
ResNet50 + generirane slike	0.745
ResNet50 + spojeni skupova	0.755

Nakon određenog broja epoha, pojavila se prenaučenost modela. Točnost validacijskog skupa je počeo opadat dok je gubitak i točnost na skupu za treniranje opadala. ResNet se u potpunosti oslanja na normalizaciju grupe za svoju regularizaciju koja u našem slučaju nije bila dovoljno učinkovita. Uz to, mogući razlog prenaučenosti je i malen broj podataka.

Spajanjem validacijskog skupa i skupa za treniranje proširio sam skup za treniranje s novim podacima. Značenje novih podataka pokazuje porast točnosti. Iako je točnost porasla, razlika nije prevelika jer ukupan broj podataka i dalje nije dovoljno velik.

Također, kako bi mogli koristiti grupe kod učenja, podatke je trebalo popuniti do zajedničke visine i širine. Kod treniranje je dio vremena otpalo na učenje dopuna koje nisu bile prisutne u podacima za validaciju i test.

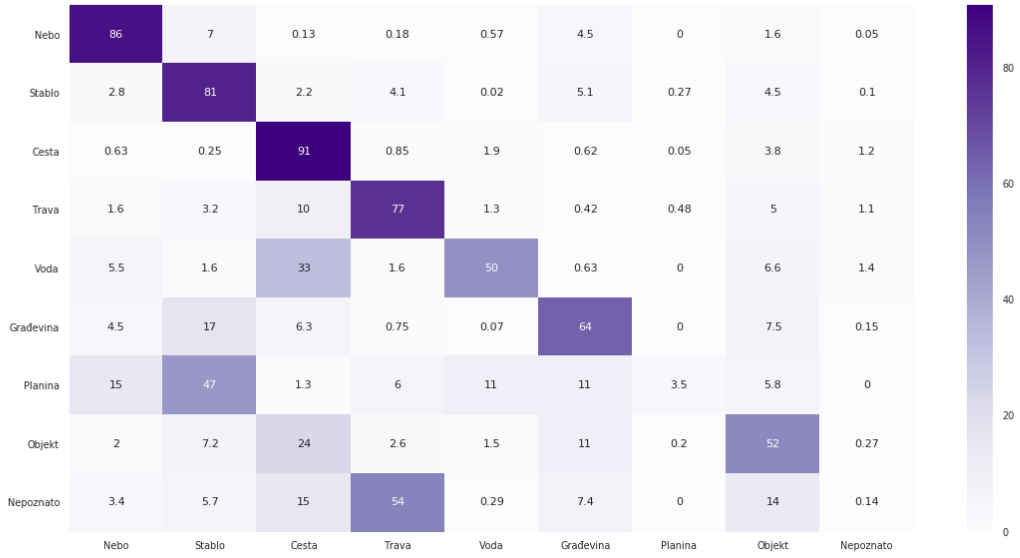
Matrica zabune

Točnost nam govori koliki postotak piksela je dobiveni model pravilno klasificirao. Kako bi dobili bolju sliku rezultate, ispisao sam i grafički prikazao (slika 4.2) matricu zabune. Matrica zabune je matrica kojoj redovi govore koji je razred točan za određeni piksel dok stupci određuju razred u koji je naš model smjestio taj isti piksel. Tako možemo uočiti koje razrede je model dobro naučio, koje lošije i između kojih razreda je dolazilo najviše do zabune. U grafičkom prikazu, svaki član matrice je relativna frekvencija klasificiranja određenom razredu u slučaju nekog razreda.

Model je dobro raspoznaje većinu razreda s dvije iznimke, planine i nepoznate piksele. Lošiji rezultat planina je bio očekivan jer se na slikama uglavnom nalazi u po-

zadinama i, kao što će biti vidljivije u grafičkom prikazu rezultata, izgleda vrlo slično šumama u pozadini. Postoji ogromna razlika u broju piksela koji pripadaju planinama i stablima. Kako je veći broj razreda pripadao stablima, model ima određenu pristranost tom razredu pa će u slučaju podjednakih rezultata stabla i planina odabrat stablo. Razred piksela koji su nepoznati je bio pridružen i pikselima koji su činili dopunu korištenu u treniranju. Zbog tog model vjerojatno jako rijetko raspozna pravilno navedeni razred. Također je zanimljivo zamjetiti greške kao što su klasificiranje piksela koje određuju vodu, pikselima koje određuju ceste.

	Nebo	Stablo	Cesta	Trava	Voda	Građevina	Planina	Objekt	Nepoznato
Nebo	1280423	104587	1899	2667	8457	66611	0	24107	759
Stablo	40968	1203131	33458	60534	246	75905	4043	67545	1529
Cesta	16268	6367	2334042	21757	48217	16069	1315	97776	29718
Trava	11637	23170	72609	552906	9541	3057	3445	35807	7627
Voda	20000	5867	120148	5925	179750	2288	0	23924	5210
Građevina	116603	429066	163148	19337	1717	1642388	0	192622	3960
Planina	14782	46549	1296	5869	10438	10417	3396	5728	0
Objekt	29645	107091	352908	39146	22272	157883	3018	772670	3961
Nepoznato	928	1570	4140	14938	80	2029	0	3777	39



Slika 4.2: Grafički prikaz matrice zabune

4.5.1. Grafički prikaz rezultata

Iscrtano je par rezultata kako bi vidjeli bolje koje razrede model pravilno raspoznaje na slikama, a koje lošije. Na trećoj slici se vidi problem prepoznavanja sličnih razreda

kao što su nebo, cesta i voda. Ostale slike prikazuju odlične rezultate. Stabla i travu raspoznaje jako dobro, klase koje su vjerojatno često bile prisutne u slikama za učenje. Od objekata u prednjem planu, najbolje prepoznaje automobile i krave, objekti koji su u velikoj mjeri bili prisutni u skupu za treniranje i validaciju.

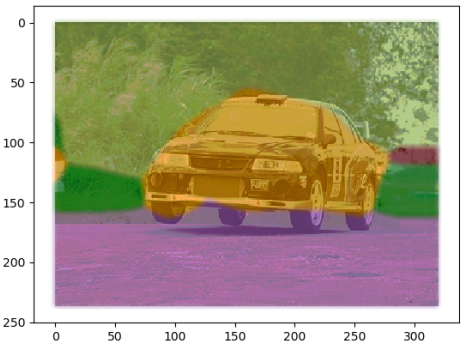
4.5.2. Eksperimenti

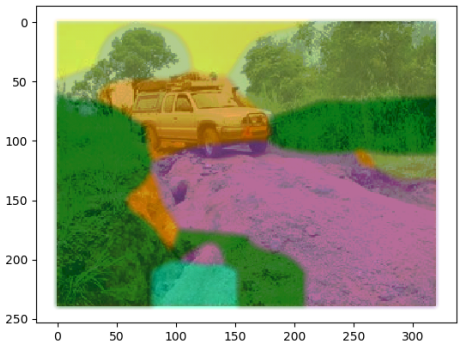
Dobivenu točnost je model dosegao relativno brzo. Model je treniran na Google-ovom Colabu koji je dopustio korištenje njihovih servera s NVIDIA TESLA K80 grafičkim karticama. Trajanje jedne epohe koja se sastojala od 457 slika je trajalo otprilike 20 sekundi. Jedna epoha na serveru s procesorom trajala je oko 3 minuta. 12GB radne memorije navedene grafičke kartice je bilo dovoljno za ResNet od 50 slojeva. Tome pomaže i *bottleneck* rezidualni blokovi koji pokušavaju održati broj parametara na normalnoj razini. Kako bi provjerili učinkovitost takvih blokova, zamjenio sam ih s običnim rezidualnim blokovima. Međutim, memorija je jako brzo bila popunjena pa treniranje nije bilo moguće.

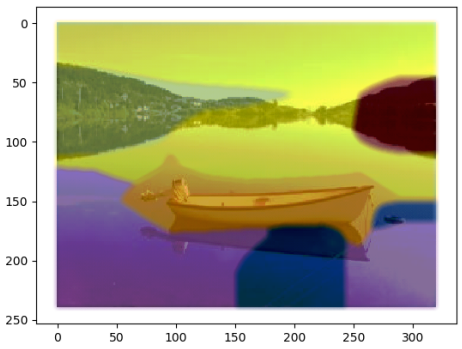
Možemo se još uvjeriti u učinkovitost "prečica" rezidualnih blokova. Rezidualne blokove sam pretvorio u obične konvolucijske slojeve micanjem prečica. Pokrenuto je trenirnje na istim podacima i postavkama kao i kod učenja s rezidualnim blokovima. Odmah na početku je bilo vidiljivo izrazito sporo učenje. Gubitak se smanjivao, ali je to bilo nezamjetno. Time smo pokazali da izrazito duboke neuronske mreže zahtjevaju rezidualne blokove kako bi postigli zadovoljavajuće rezultate.

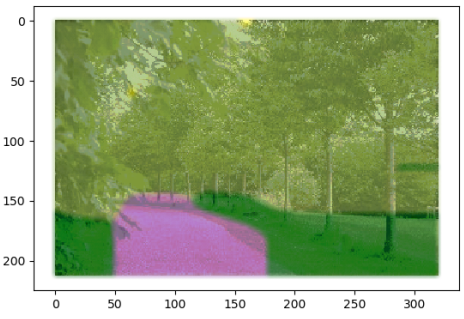
Na kraju sam provjerio učinak dodavanja *dropout-a* u rezidualne blokove između svakog konvolucijskog sloja. Dobio smo puno lošije rezultate što nam govori da je za ResNet trenutno najbolja opcija oslanjat se na normalizaciju grupe i blokove s uskim grlom. Rezidualni blokovi su relativno novi u svijetu dubokog učenja pa se tako još razvijaju rješenja koja ih mogu još efikasnije iskoristiti. Nedavno su se pojavile široke rezidualne mreže (Zagoruyko i Komodakis, 2016) koje koriste *dropout* za regularizaciju u velikoj mjeri, s tim da se umjesto blokova s uskim grlom koriste obični rezidualni blokovi, i tako postiže bolje rezultate što znači da rezidualni blokovi ne eliminiraju upotrebu *dropout-a*, ali je potrebno istražiti kako efikasno uklopiti druge oblike regularizacije u mreže koje koriste rezidualne blokove.

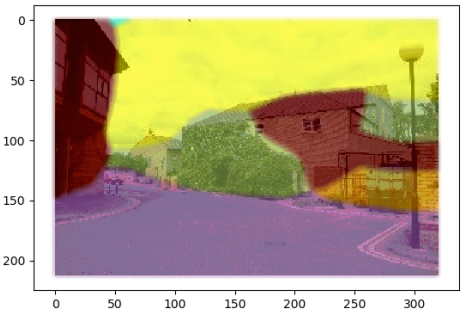
Nebo Stablo Cesta Trava Voda Građevina Planina Objekt Nepoznato

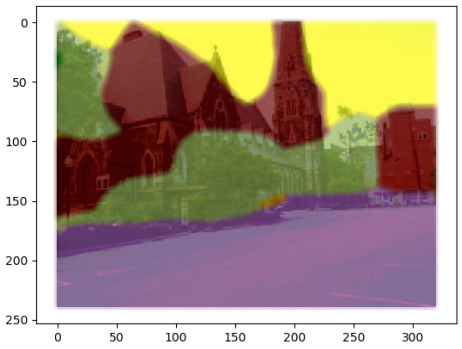


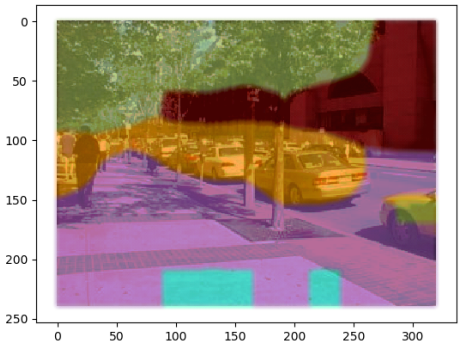


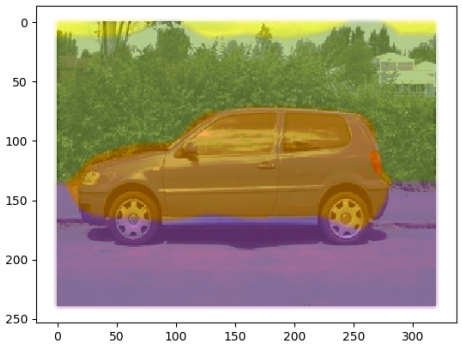


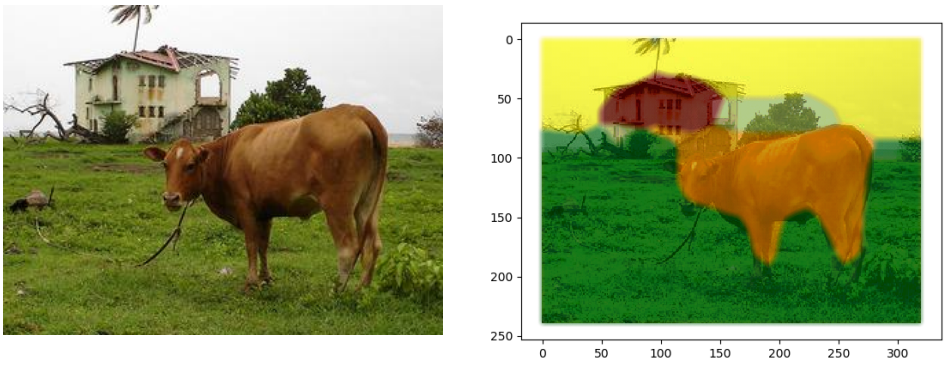



0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
250
0
50
100
150
200
250
300


0
50
100
150
200
250
0
50
100
150
200
250
300


0
50
100
150
200
250
0
50
100
150
200
250
300



Slika 4.5: Grafički prikaz rezultata

5. Zaključak

Rezidualne neuronske mreže su bile još jedan veliki iskorak u razvoju neuronskih mreža, ali i strojnog učenja. Jednostavna ideja kao što je stvaranje prečica između nekih slojeva omogućilo je stvaranje mreža puno većih dubina, a da se ne gubi na točnosti.

Međutim, kao i ostale mreže velikih dubina, rezidualne mreže također prate određeni nedostatci kao što je potreba za većom količinom memorije i velikim brojem podataka. Neke tehnike ne funkcioniraju pravilno s arhitekturom ResNet-a pa je potrebno eksperimentalno utvrditi kako bi se takve arhitekture mogle još poboljšati.

Kako računalni vid postiže odlične rezultate s metodama dubokog učenja, tako su i rezidualne mreže bile ogroman iskorak za to područje.

Kroz rad sam se upoznao s radnim okruženjem Tensorflow i implementirao sam rješenje za jedan od problema računalnog vida. Postigao sam dobre rezultate na problemu semantičke segmentacije, a vjerujem da bi uz veći broj podataka i manje preinake rezultati bili još bolji.

Na temelju dobivenih rezultata, možemo zaključiti da rezidualne veze omogućuju dublje neuronske mreže. Međutim, rezultati i dalje ovise u potpunosti o podacima, pogotovo o broju podataka. Uz pomoć rezidualnih veza možemo izgraditi različite mreže proizvoljne dubine, a njihov maksimalan potencijal je još potrebno utvrditi.

LITERATURA

Ivan Borko. Semanticka segmentacija prirodnih scena dubokim neuronskim mrežama. *Magistarski rad*, 2015.

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

Ivan Grubišić. Semantička segmentacija slika dubokim konvolucijskim mrežama. *Završni rad*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Jonathan Long, Evan Shelhamer, i Trevor Darrell. Fully convolutional networks for semantic segmentation. 2015.

Michael A. Nielsen. *Neural Networks and Deep Learning*. 2015.

S. Zagoruyko i N. Komodakis. Wide Residual Networks. *ArXiv e-prints*, Svibanj 2016.

Semantička segmentacija slika dubokim konvolucijskim mrežama

Sažetak

U ovom radu su opisane neuronske mreže s posebnim naglaskom na konvolucijske mreže. Opisani su razni dijelovi i tehnike korištene u izgradnji i učenju neuronskih mreža. Rad također opisuje rezidualne neuronske mreže kao i njihovu konkretnu implementaciju, ResNet50. Programski je implementiran i evaluiran model za semantičku segmentaciju dobiven arhitekturom ResNet50. Rezultati su prikazani i komentirani na kraju rada.

Ključne riječi: neuronske mreže, konvolucijske neuronske mreže, rezidualne neuronske mreže, ResNet, semantička segmentacija, klasifikacija

Semantic Segmentation With Deep Convolutional Models

Abstract

This paper describes neural networks with focus on deep convolutional neural networks. Different parts and techniques used in neural networks are described. Also, paper describes residual neural network as well as its concrete implementation, ResNet50. Paper contains details about the implementaion of the system for semantic segmentation using ResNet50. Lastly, the results are presented.

Keywords: neural networks, convolutional neural networks, residual neural networks, ResNet, semantic segmentation, classification