

Zahvaljujem se mentoru, prof. Marku Subašiću na potpori i savjetima na putu kroz nepoznato. Hvala mojoj obitelji te Lei, na svoj potpori koju su mi pružili i nastavljaju pružati.

SADRŽAJ

1. Uvod	1
2. Računalni vid	2
2.1. Podjela računalnog vida	2
2.1.1. Klasifikacija	2
2.1.2. Lokalizacija	3
2.1.3. Detekcija	3
2.1.4. Semantička segmentacija	3
2.1.5. Segmentacija instanci	3
3. Duboko učenje	5
3.1. Neuronske mreže	5
3.1.1. Umjetni neuron	6
3.1.2. Arhitektura neuronske mreže	8
3.1.3. Učenje neuronskih mreža	10
3.1.4. Regularizacija i optimizacija	12
3.2. Duboko učenje	14
3.2.1. Konvolucijske neuronske mreže	15
3.3. Primjena u računalnom vidu i semantičkoj segmentaciji	19
3.3.1. ResNet	20
3.3.2. U-net	21
3.3.3. Deeplab	22
4. Generativni modeli	24
4.1. GAN	25
4.2. cGAN	28
4.3. DCGAN	29
4.4. cycleGAN	29

5. Korištenje cycleGAN arhitekture za semantičku segmentaciju ortopanograma	33
5.1. Opis problema	33
5.2. Opis predloženog rješenja	35
5.2.1. Arhitektura rješenja	35
5.2.2. Programska implementacija	38
5.3. Eksperimentalni rezultati	39
6. Zaključak	47
Literatura	48

1. Uvod

U današnjem svijetu podataka iz dana u dan ima sve više. Međutim, iako je do podataka danas vrlo lako doći, izvući korisno znanje iz njih se može pokazati kao vrlo težak zadatak. Računalni vid je jedno od područja umjetne inteligencije koje na osnovu 2D slika i videozapisa - podataka, nastoji dobiti što više informacija, odnosno korisnog znanja. Primjene ovog područja su brojne, od autonomne vožnje i cestovne sigurnosti do primjena u zdravstvu i medicini. Jedna od metoda koja se zadnjih godina pokazala najuspješnijom su duboki neuronski modeli, matematički modeli učenja zasnovani na principu rada ljudskog mozga.

Međutim, da bi duboki modeli bili uspješni, potrebne su im velike količine označenih podataka, do kakvih je često vrlo teško doći. Kako bi se riješio taj problem, razvijena su brojna rješenja, među koje spada i polunadzirano učenje - pristup učenju dubokih mreža gdje, uz dio označenih podataka, model koristi i određen broj neoznačenih podataka kako bi poboljšao učenje. Jedan od takvih modela je i cycleGAN, nadogradnja popularne GAN arhitekture, posebice korisne u primjenama u računalnom vidu, kad je broj označenih podataka ograničen.

U ovom radu opisat ću osnove dubokih neuronskih mreža, s posebnim naglaskom na duboke konvolucijske modele. Predstaviti ću i koncept generativnih mreža, te ću se posebno fokusirati na jednu konkretnu generativnu mrežu - cycleGAN. Predloženu arhitekturu primijenit ću za rješavanje problema semantičke segmentacije ortopanograma, što je problem koji ima brojne praktične primjene. Rezultate predložene arhitekture ću usporediti s rezultatima postojećih diskriminativnih mreža, te donijeti zaključak koji pristup je prikladniji za navedeni problem.

2. Računalni vid

Računalni vid (engl. *computer vision*) je grana računarske znanosti, točnije područja umjetne inteligencije, koja pokušava omogućiti računalima da interpretiraju vizualne podatke te iz njih izvuku nove informacije. To je zasigurno jedno od trenutno najuzbudljivijih i najbrže rastućih područja umjetne inteligencije, čije primjene sežu od automatskog označavanja ljudi na slikama koje objavljujemo na Instagramu i Facebooku, sve do autonomne vožnje i primjena u medicini i znanosti. Ovo područje danas doživljava svoju renesansu, u velikom dijelu zbog pojave algoritama baziranih na dubokom učenju i napretka u računalnim resursima. No ne treba zaboraviti da je računalni vid UI-potpun problem, te da je put koji smo morali prijeći da dođemo do današnjih uspjeha bio sve osim lagan.

U ovom poglavlju dati ću osnovu podjelu područja računalnog vida te se kratko osvrnuti na njihovu praktičnu primjenu.

2.1. Podjela računalnog vida

Područje računalnog vida obuhvaća mnoga i raznolika područja. Analiza pokreta u video snimkama, detekcija objekata, rekonstrukcija 3D scene iz 2D slike su samo neki od primjera raznovrsnosti računalnog vida. Nas zanima jedno konkretno područje - detekcija objekata u 2D slikama. Glavni zadatak tog područja je, kako mu samo ime i kaže, izvući razne informacije o položaju objekata unutar slike. Ovisno o vrsti informacije, ovo područje dalje možemo podijeliti na klasifikaciju, lokalizaciju, detekciju, semantičku segmentaciju te segmentaciju instanci.

2.1.1. Klasifikacija

Najjednostavnija varijanta detekcije objekata, klasifikacija (engl. *classification*), nastoji zadanu sliku svrstati u neku od predefiniranih klasa. Na primjer, ukoliko

algoritmu za klasifikaciju kao ulaz predamo sliku na kojoj se nalazi mačić koji sjedi na livadu, mogli bismo očekivati da će algoritam slici pridijeliti oznaku *mačka*, iako su na slici prisutni i drugi elementi poput trave, oblaka ili sunca. Povećanjem broja klasa težina problema se povećava.

2.1.2. Lokalizacija

Sljedeći, nešto napredniji korak je lokalizacija (engl. *localization*), odnosno utvrđivanje točne lokacije **jednog** objekta na slici. Na već spomenutom primjeru s mačićem na livadi, cilj bi nam bio označiti točnu lokaciju mačića u slici, najčešće crtanjem pravokutnika oko njegove lokacije.

2.1.3. Detekcija

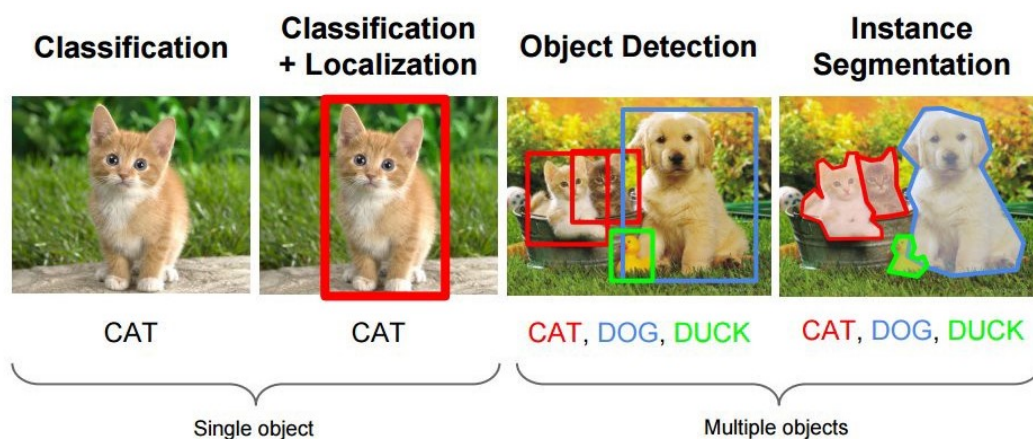
Detekcija (engl. *detection*) je u svojoj osnovi vrlo slična lokalizaciji, samo što za razliku od lokalizacije, koja utvrđuje lokaciju jednog objekta na slici, detekcija utvrđuje točnu lokaciju svih instanci objekata za slici (za koje postoje zadane klase). Česta primjena ove vrste detekcije objekata je u autonomnoj vožnji, gdje je za upravljački softver vrlo važno da zna točnu lokaciju ostalih vozila, pješaka itd., kako bi mogao donijeti ispravnu odluku o upravljanju vozilom.

2.1.4. Semantička segmentacija

Semantička segmentacija (engl. *semantic segmentation*) ide korak dalje, te za svaki piksel u slici nastoji odrediti kojoj klasi pripada. Ovaj problem je znatno složeniji nego prethodna tri, jer zahtjeva značajnije razumijevanje semantike slike u odnosu na prethodne. Ova vrsta detekcije također ima primjene u sustavu za autonomnu vožnju, ali i u drugim područjima, od kojih valja istaknuti primjenu u području medicinskih slika. Ovo je pristup koji nas najviše zanima i koji ćemo pobliže istražiti u nastavku ovog rada.

2.1.5. Segmentacija instanci

Segmentacija instanci (engl. *instance segmentation*) je složeniji oblik semantičke segmentacije, gdje se svaki piksel u slici nastoji pridijeliti točno jednoj **instanci** određene klase. Dakle, ako na slici imamo tri mačića, pikseli kojima su oni prikazani na slici pripadat će klasi mačić, ali model će također znati razlikovati pripada li određeni piksel mačiću 1, 2 ili 3.



Slika 2.1: Usporedba načina detekcije [1]

Iako je rješavanje svih navedenih problema gotovo trivijalno za ljude, oni računalima predstavljaju velik izazov. Stoga ćemo u sljedećim poglavljima detaljnije pogledati jedan pristup koji se do sad pokazao najboljim - duboko učenje.

3. Duboko učenje

Pojava i razvoj računala u drugoj polovici 20. stoljeća omogućila je automatizaciju mnogih poslova za koje su prije bili potrebni ljudi. Obrada i unos velike količine podataka, nadzor raznih proizvodnih procesa do povezivanja cijelog svijeta kroz jedinstvenu globalnu mrežu - Internet, računala su uspješno preuzela mnoge poslove za koje su prije bili nužni ljudi. No određena vrsta problema dugo je vremena izmicala pokušajima da se automatizira računalima - tzv. UI - potpuni problemi. To su problemi čija je složenost jednaka rješavanju glavnog problema umjetne inteligencije - izgradnja stroja inteligentnog poput čovjeka [2]. U to područje spadaju problemi poput obrade prirodnog jezika, raspoznavanja uzoraka, problemi kretanja i navigacije, te među ostalim, i područje računalnog vida. Konkretni pokušaji rješavanja ovih problema ovisili su o konkretnom problemu, te su se zasnivali na algoritamskom rješavanju problema (u području umjetne inteligencije često nazivan i simbolički pristup). Primjerice, u području računalnog vida često je bilo korištenje ručno podešenih filtera i klasifikatora za rješavanje raznih problema. No, do pravog napretka dolazi sredinom 2000-ih, pojavom dubokih neuronskih mreža, koje su uvele pravu revoluciju u područje računalnog vida.

U ovom poglavlju predstaviti ću osnovne koncepte rada dubokih neuronskih mreža, krenuvši od opisa umjetnog neurona i jednostavne neuronske mreže, sve do složenijih dubokih modela. Također ću se osvrnuti na njihovu primjenu u području računalnog vida i problemu semantičke segmentacije, te dati primjer par najčešće korištenih arhitektura.

3.1. Neuronske mreže

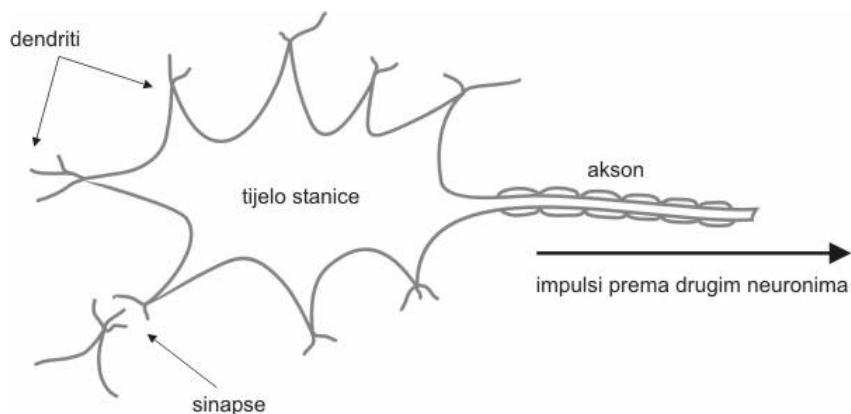
Neuronsku mrežu možemo definirati kao skup međusobno povezanih procesnih jedinica (neurona) čija se funkcionalnost temelji na modelu biološkog mozga, od-

nosno neurona. Dakle, to je sustav koji za obradu podataka i donošenje zaključka nastoji imitirati ljudski mozak. Glavne karakteristike ovakvog pristupa, koji se često naziva i konekcionistički, su samostalno učenje na temelju predloženih podataka te mogućnost distribuirane paralelne obrade podataka. Navedene karakteristike su u suprotnosti s algoritamskim pristupom, gdje se koraci više manje izvode slijedno, a novo znanje izvodi se pomoću predefiniраниh pravila. Opis arhitekture neuronske mreže započinjemo s njenim osnovnim gradivnim elementom - neuronom.

3.1.1. Umjetni neuron

Kako bismo bolje razumjeli model umjetnog neurona, pogledajmo prvo njegov biološki ekvivalent, kojim je i inspiriran.

Biološki neuroni su osnovne gradivne jedinice ljudskog mozga - prosječan ljudski mozak sadrži preko 10^{11} neurona, podijeljenih u preko 100 različitih vrsta. Glavni dijelovi biološkog neurona su dendriti, aksoni te tijelo (soma). **Dendriti** su ulazni dijelovi neurona, preko kojih neuron prima podražaje od susjednih neurona. Obrada primljenih podražaja događa se u **tijelu** neurona. U tijelu se zbrajaju sume svih podražaja primljenih preko dendrita, te se donosi odluka hoće li se neuron aktivirati ili ne. U slučaju pozitivne odluke o aktivaciji, šalje se izlazni podražaj kroz **akson** neurona.[2][3]



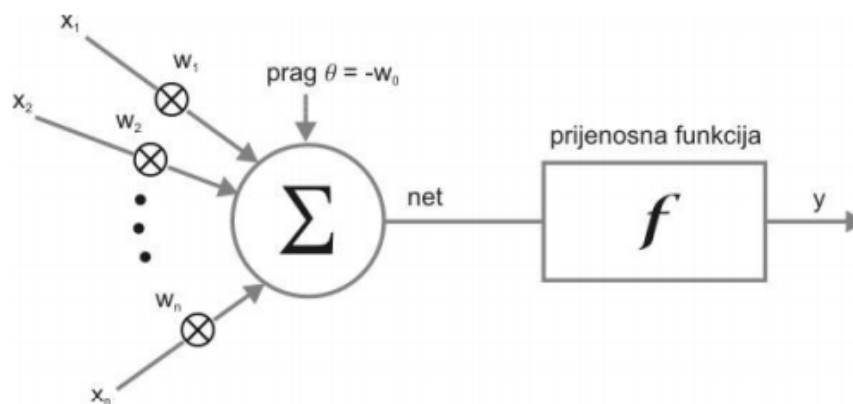
Slika 3.1: Model biološkog neurona [4]

Pogledamo li osnovni model umjetnog neurona, prikazan na slici 3.2, možemo primijetiti da je vrlo sličan modelu biološkog neurona. Neuron prima N nume-

ričkih ulaza, označenih sa \mathbf{x}_i . Svakom ulazu je također pridružena i pripadajuća osjetljivost, \mathbf{w}_i , s kojom se ulaz množi prije ulaska u neuron. U neuronu se pomnoženi ulazi sumiraju, te se ukupnoj sumi dodaje pomak (engl. *bias*), \mathbf{w}_0 . Ovime smo definirali akumuliranu vrijednost **net**, te ju možemo zapisati kao

$$\mathbf{net} = \left(\sum_{i=1}^n \mathbf{x}_i \cdot \mathbf{w}_i \right) + \mathbf{w}_0 \quad (3.1)$$

Dobivena *net* vrijednost propušta se kroz **izlaznu funkciju** f (engl. *activation function*), te dobivamo izlaznu vrijednost, o . [2][4]



Slika 3.2: Model umjetnog neurona [4]

Kako bi olakšali prikaz parametara neuronske mreže, možemo prijeći na vektorsku notaciju, gdje vektorom smatramo matricu sa točno jednim stupcem i N redaka. Onda ulaze \mathbf{x}_i neurona možemo zapisati kao jedan vektor $\vec{x} = (\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_n)$. Težine možemo prikazati vektorom $\vec{w} = (\mathbf{w}_1, \mathbf{w}_2 \dots \mathbf{w}_n)$, dok pomak \mathbf{w}_0 možemo ostaviti u skalarnom obliku. Konačno, izlaz neurona o možemo zapisati kao:

$$o = f(\vec{w}^T \cdot \vec{x} + b) \quad (3.2)$$

Upravo opisan model neurona se naziva TLU-perceptron (engl. *Threshold logic unit*), te je prvi put predstavljen od strane McCullougha i Pittsa 1943. godine. Iako je razmjerno jednostavan, dovođenjem odgovarajućih ulaza te podešavanjem težina, pomaka i izlazne funkcije, njega već možemo koristiti za jednostavnije postupke klasifikacije i regresije. No, oni ipak imaju svoje ograničenje. To ograničenje se očituje činjenici da je **granica odluke** (skup točaka oko kojih neuron mijenja svoju odluku) TLU-perceptrona linearna, što znatno ograničava moguće

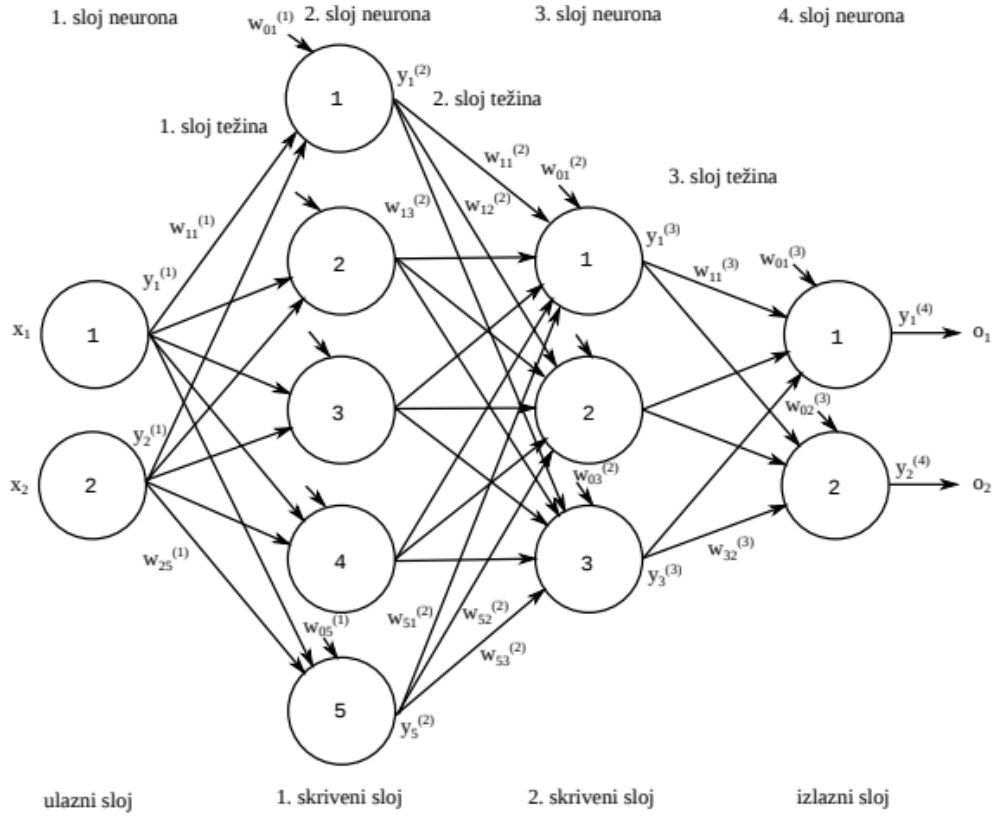
primjene. Kako bismo riješili ovaj problem, prelazimo na sustave više povezanih neurona - neuronske mreže.

3.1.2. Arhitektura neuronske mreže

Neuronske mreže nisu ništa drugo osim sustavi međusobno povezanih umjetnih neurona. Svaka neuronska mreža sastoji se od više slojeva. **Sloj** je skup neurona unutar neuronske mreže koji se uvijek zajedno aktiviraju. Razlikujemo 3 vrste slojeva: ulazni sloj, skrivene slojeve te izlazni sloj. **Ulazni sloj** služi kao ulazna točka za podatke, te uvijek postoji samo jedan. **Izlazni sloj** je uvijek također samo jedan, te služi za izlaz iz neuronske mreže. Priroda tog izlaza ovisi o tome kako koristimo neuronsku mrežu - primjerice, koristimo li ju za klasifikaciju ulaza u N klasa, izlazni sloj može činiti N neurona, od kojih svaki kao izlaz daje vjerojatnost da ulaz pripada i -toj klasi. **Skriveni slojevi** su glavni dio neuronske mreže, te upravo podešavanjem njihovih težina i pragova možemo neuronsku mrežu prilagoditi, odnosno naučiti nekom konkretnom problemu. Na slici 3.3 također možemo primijetiti da svaki neuron u nekom sloju kao ulaz prima izlaze neurona iz prijašnjeg sloja. Svaka takva veza ima određenu težinu w_{qp} , gdje je q indeks neurona u prijašnjem sloju, a p indeks neurona u trenutnom sloju. Npr. težinu veze koja spaja neuron indeksa 1 u prijašnjem sloju te neuron indeksa 3 u trenutnom sloju mogli bismo označiti sa w_{13} .

Izlaz određenog neurona ovisi o vrijednosti izlaza neurona iz prošlog sloja, težine između trenutnog i prošlog neurona te vrijednosti praga u neuronu. Ovakve mreže, gdje izlaz neurona trenutnog sloja ovisi isključivo o izlazima neurona prethodnog sloja nazivamo **slojevitima**. Uz njih, postoje i **neslojevite** neuronske mreže, gdje izlaz neurona u nekom sloju može ovisiti o nekom neuronu više slojeva prije, ili čak o neuronu u nekom od sljedećih slojeva. Međutim, ovakve neuronske mreže nas trenutno ne zanimaju, te se u nastavku rada bavimo samo sa unaprijednim slojevitim neuronskim mrežama.

Sada nas još zanima kako doći do izlaza određenog sloja neuronske mreže. Kako bi olakšali zapis parametara neuronske mreže, kao i kod umjetnog neurona uvodimo vektorski zapis. Matricu težina između i -tog i $(i-1)$ sloja neuronske mreže možemo zapisati kao \mathbf{W}_i . Vrijednost u i -tom retku te j -tom stupcu matrice označava vrijednost težine između i -tog neurona trenutnog sloja te j -tog neurona



Slika 3.3: Arhitektura neuronske mreže 2 x 5 x 3 x 2 [5]

prethodnog sloja. Ukoliko još sa \mathbf{b}_i označimo pragove i -tog sloja neurona, izlaz i -tog sloja neuronske mreže, h_i , možemo zapisati kao:

$$\vec{h}_i = f(\mathbf{W}_i \cdot \vec{h}_{i-1} + \mathbf{b}_i) \quad (3.3)$$

gdje je f izlazna funkcija [2]. Drugim riječima, izlaz svakog sloja neuronske mreže jednak je skalarnom produktu matrice težina sloja i i izlaza prethodnog sloja, na koji još dodamo vektor pragova trenutnog sloja. Nad dobivenim vektorom primijenimo izlaznu funkciju, te smo dobili izlazni vektor sloja, u kojem i -ti stupac odgovara izlazu i -tog neurona u sloju. Ukoliko se radi o prvom skrivenom sloju, vektor h_{i-1} je ustvari samo ulazni vektor. Vidimo da, kako bi u potpunosti definirali našu mrežu, potreban nam je vektor težina, vektor pragova i prijenosna funkcija za svaki od slojeva. Modifikacijom nekog ili svih parametara možemo promijeniti izlaz naše mreže. Upravo modifikacija navedenih parametara je jedan od ključnih dijelova u procesu učenja odnosno treniranja mreže, detaljnije obrađenog u sljedećem potpoglavlju.

Zadnja stvar koju je bitno napomenuti je odabir odgovarajuće prijenosne funkcije.

Ako želimo da naša mreža bude ekspresivnija od običnog neurona, odnosno ako želimo modelirati i nelinearne odnose, nužno je koristiti nelinearnu prijenosnu funkciju. To je rezultat činjenice da je linearna kombinacija linearnih kombinacija također linearna kombinacija. Odnosno, ako primijenimo linearnu prijenosnu funkciju na izlaz sloja neuronske mreže, koji je i sam linearna kombinacija težina i ulaza, kao rezultat možemo dobiti samo novu linearnu kombinaciju. Najčešće korištene prijenosne funkcije uključuju sigmoidu, tangens hiperbolni, softmax i reLU, od kojih je posljednja posebice korisna prilikom treniranja dubokih modela.[2]

3.1.3. Učenje neuronskih mreža

Učenje neuronskih mreža svodi se na mijenjanje jakosti veza između neurona - dakle na podešavanje vrijednosti pragova i težina neurona unutar mreže. Te parametre prilagođavamo kroz iterativan postupak predočavanja ulaznih primjera, najčešće u kombinaciji s očekivanim izlazom - treniranje. Stoga možemo zaključiti da je znanje o izlazu kao funkciji izlaza unutar neuronske mreže pohranjeno implicitno, kroz pragove i težine između neurona. To rezultira time da neuronske mreže najčešće funkcioniraju po principu crne kutije - za dani ulaz, vraćaju odgovarajući izlaz, međutim način na koji je neuronska mreža došla do tog izlaza je teško opisati. Takav način pohrane znanja o mapiranju ulaza na izlaze u suprotnosti je algoritamskim pristupom, gdje točno možemo opisati svaki korak pomoću kojeg smo došli do izlaza.[4]

Prije nego detaljnije opišemo postupak učenja, napomenimo da najčešće razlikujemo 3 vrste učenja. Prva, najčešće korištena metoda u svijetu dubokog učenja, je **nadzirano** (engl. *supervised*) učenje. Glavna karakteristika ove metode je da modelu, u ovom slučaju neuronskoj mreži, dajemo parove (ulaz, izlaz). Sustav zatim pokušava konstruirati funkciju koja će što točnije moći preslikati dani ulaz na očekivani izlaz. Druga metoda je **nenadzirano** (engl. *unsupervised*) učenje. U ovoj metodi modelu dajemo samo određen broj ulaza, bez pripadajućih izlaza, i modelu prepuštamo da pronade pravilnost u podacima. Ova metoda se može primijeniti u kombinaciji sa nadziranim učenjem. Baš tu kombinaciju ćemo pobliže istražiti u nastavku ovog rada. Treća i ujedino posljednja vrsta učenja je **podržano** (engl. *reinforcement*) učenje. Glavni cilj ovog načina učenja je doći do niza koraka, odnosno strategije, koja rješava neki problem. U nastavku ćemo po-

gledati primjer nadziranog učenja neuronskih mreža uz pomoć jednog od najčešće korištenih algoritama - **algoritma propagacije pogreške unazad** (engl. *error backpropagation*).

Kako bi objasnili algoritam, potrebno je uvesti još par pojmova. Prvi je pojam **funkcije pogreške** (engl. *loss function*), često nazivane i funkcija gubitka. Već smo spomenuli da se, u svojoj osnovi, treniranje neuronskih mreža svodi na predočavanje ulaznih primjera te usporedbom dobivenog izlaza s očekivanim izlazom. Na temelju razlike između očekivanog i dobivenog izlaza modificiraju se parametri mreže. Međutim, ostaje pitanje kako točno kvantificirati razliku između ta dva izlaza? Za to nam služe funkcije gubitka. To su funkcije koje na osnovu očekivanog i dobivenog izlaza računaju jedinstvenu numeričku vrijednost pogreške. Očito, model je bolji što je vrijednost funkcije pogreške manja, stoga se učenje modela u praktičnoj primjeni zasniva na minimizaciji funkcije pogreške. U teoriji, model u kojem je svaki dobiveni izlaz jednak svakom očekivanom izlazu bi imao vrijednost funkcije pogreške jednaku 0.

Jedna od češće korištenih funkcija gubitka je (polovična) suma srednjih kvadratnih odstupanja (engl. *Mean Squared Error, MSE*), koju definiramo kao:

$$MSE = \frac{1}{2N} \sum_{s=1}^N \sum_{i=1}^{N_0} (t_{s,i} - o_{s,i})^2 \quad (3.4)$$

pri čemu s označava redni broj para (očekivani izlaz, dobiveni izlaz), N broj takvih parova, N_0 dimenziju izlaza, t očekivani izlaz i o dobiveni izlaz. Kvadrat u izrazu služi kako bi osigurali da će dobivena vrijednost uvijek biti pozitivna, neovisno o tome je li očekivani izlaz manji ili veći od dobivenog.

Prisjetimo se na trenutak matematičke analize i pojma gradijenta funkcije - vektora nastalog parcijalnom derivacijom funkcije po svim njenim varijablama. On ima korisno svojstvo - pokazuje nam smjer najbržeg rasta funkcije. Jednom kada smo izračunali gradijent funkcije, možemo ga iskoristiti za minimizaciju funkcije korištenjem **tehnike gradijentnog spusta** (engl. *gradient descent*). Tehnika se sastoji od malih modifikacija svake varijable funkcije u smjeru suprotnom od onog u kojem pokazuje gradijent. Ova tehnika nam garantira eventualni dolazak u neki od lokalnih minimuma funkcije. [6] Ova ideja se sad lako može primijeniti na funkciju gubitka neuronske mreže - funkcija gubitka je funkcija čiju vrijednost želimo minimizirati, a parametri koje prilagođavamo su upravo parametri neuronske mreže - težine i pragovi. To je upravo osnovna ideja algoritma propagacije

pogreške unazad, čiju točnu definiciju dajem u nastavku. Algoritam je preuzet iz [5], te pretpostavlja da je korištena sigmoidalna prijenosna funkcija:

Algoritam 1 Algoritam propagacije pogreške unazad

1. Sve težine neuronske mreže postavi na slučajne vrijednosti
 2. **while** uvjet zaustavljanja nije zadovoljen **do**:
 - repeat**
 - for** svaki uzorak s iz skupa za učenje **do**
 1. Postavi podatak \mathbf{x} na ulaz mreže
 2. Izračunaj izlaze neurona svih slojeva, od prvog prema posljednjem (izlaz posljednjeg sloja označimo sa $(o_{S,1}, \dots, o_{S,N_0})$)
 3. Odredi pogrešku svakog od neurona izlaznog sloja
 S δ_i^k označimo pogrešku i -tog sloja te ju računamo prema izrazu:

$$\delta_i^K = o_{s,i} \cdot (1 - o_{s,i}) \cdot (t_{s,i} - o_{s,i}),$$
 gdje je
 $(t_{s,i} - o_{s,i})$ - odstupanje između dobivenog i očekivanog izlaza, a
 $o_{s,i} \cdot (1 - o_{s,i})$ - derivacija prijenosne funkcije neurona (sigmoide)
 4. Vraćaj se sloj po sloj prema početku mreže i za svaki neuron i u sloju k računaj pogrešku prema izrazu:

$$\delta_i^k = y_i^k \cdot (1 - y_i^k) \cdot \sum_{d \in \text{Downstream}} w_{i,d} \cdot \delta_d^{k+1}$$
 5. Napravi korekciju svih težina. Težinu $w_{i,j}^k$ korigiraj prema izrazu

$$w_{i,j}^k \leftarrow w_{i,j}^k + \eta \cdot y_i^k \cdot \delta_d^{k+1}$$
 gdje η predstavlja stopu učenja modela
 - end for**
-

3.1.4. Regularizacija i optimizacija

Sada kada znamo osnove arhitekture i treniranja umjetnih neuronskih mreža, osvrnimo se na još dva pojma važna za izgradnju kvalitetnog modela - regularizacija i optimizacija.

Regularizacija je tehnika korištena prilikom treniranja modela koja kao cilj ima smanjiti pogrešku generalizacije modela, odnosno smanjiti prenaučenosť (engl. *overfitting*), ali pritom ne utjecati na pogrešku prilikom treniranja.[6]. Regularizacija postaje sve važnija kako idemo prema dubokim, složenijim modelima, jer povećanjem složenosti modela njegova tendencija prenaučenosťi raste. Najčešće

korištene tehnike su augmentacija podataka te L1 i L2 regularizacija.

Augmentacija podataka (engl. *data augmentation*) je jednostavan, ali vrlo moćan oblik regularizacije. Njegova osnovna ideja je kroz razne vrste manipulacija na umjetan način povećati broj uzoraka korištenih za treniranje. Povećanje broja uzoraka kao direktnu posljedicu ima bolju generalizaciju modela. Konkretna način augmentacije podataka ovisi o podacima koje koristimo, no primjerice ako koristimo 2D slike kao ulaz, augmentacija može uključivati razne manipulacije poput invertiranja slika, slučajnog obrezivanja dijelova slika ili dodavanja šuma.

Osnovna ideja **L1 i L2 regularizacija** je modifikacija funkcije gubitka tako da joj se doda određena dodatna vrijednost, koja ima cilj penalizirati prenaučeni model. Bez ulaženja u detalje, bez te dodatne vrijednosti model bi težio savršenoj prilagodbi podacima za treniranje, te bi loše generalizirao. Dodavanjem tog penala model "odmičemo" od pozicije u kojoj je savršeno prilagođen podacima za treniranje prema poziciji u kojoj će bolje generalizirati. Glavna razlika između L1 i L2 regularizacije je u načinu izračuna vrijednosti spomenutog penala. Konkretno, izraz za L1 je:

$$\sum_{j=1}^N |w_j| \quad (3.5)$$

Vidimo da se penal izračunava kao suma absolutnih vrijednosti težina mreže. S druge strane L2 regularizaciju definiramo kao:

$$\sum_{j=1}^N (w_j)^2 \quad (3.6)$$

Penal se izračunava kao suma kvadrata težina mreže. Konačni izraz za funkciju pogreške E prilikom korištenja L1 ili L2 regularizacije je onda:

$$E_{reg} = E_{orig} + \lambda(L) \quad (3.7)$$

gdje je λ parametar pomoću kojeg možemo modificirati jakost utjecaja regularizacije, a L korištena L1 ili L2 regularizacija.[7]

Optimizacija u kontekstu učenja neuronskih mreža najčešće podrazumijeva optimizaciju gradijentnog spusta, ključnog dijela učenja.

Jedan pristup optimizaciji gradijentnog spusta je kroz varijaciju količine podataka korištene u izračunu gradijenta funkcije pogreške. Osnovna varijanta gradijentnog spusta računa gradijent te provodi optimizaciju parametara na osnovu

cijelog skupa podataka za treniranje. Ovo kao posljedicu može imati unošenje redundancije, jer će se gradijenti za slične ulazne primjere računati nekoliko puta prije ažuriranja parametara. **Stohastički gradijentni spust** provodi izračun gradijenta nakon svakog ulaznog primjera. To kao posljedicu ima ubrzanje konvergencije prema lokalnom minimumu, no također rezultira većom fluktuacijom vrijednosti funkcije pogreške. Kombinacija ova dva pristupa je **gradijentni spust nad podskupu podataka** (engl. *mini-batch gradient descent*). Ovaj pristup provodi izračun gradijenta i ažuriranje parametara modela nakon svakog podskupa svih podataka za učenje. Kako veličinu tog podskupa možemo lako mijenjati, ovo nam omogućava laku kontrolu gradijentnog spusta ovisno o konkretnom problemu koji rješavamo. [8]

Druge metode optimizaciji su znatno složenije, te uključuju metode poput Momentuma, AdaGrada, AdaDelte, i drugih. Jedan od najčešće korištenih je **Adam** (engl. *Adaptive Moment Estimation*), koji ćemo ovdje ukratko opisati. Kao što smo vidjeli u Algoritmu 1, algoritam propagacije pogreške unazad, zasnovan na tehnici gradijentnog spusta, prilikom korekcije težina koristi parametar η - stopa učenja. On određuje u kojoj mjeri će se parametri modela promijeniti - što je η veći, to je promjena veća te model brže konvergira prema minimumu. Klasična tehnika gradijentnog spusta, pa čak i stohastičkog gradijentnog spusta koristi samo jedan η za sve težine u modelu, te je on konstantan tijekom cijelog učenja modela. Adam, te algoritmi na kojima se on zasniva (AdaGrad i RMSProp) računa zaseban η za svaki parametar mreže. Za računanje vrijednosti η , Adam koristi prosjek momenta prvog reda gradijenta (prosjek), ali i prosjek moment drugog reda (varijanca). [9] U praksi, Adam se pokazao najboljim, te se preporučuje njegova primjena. [8]

3.2. Duboko učenje

Duboko učenje, usprkos svom pomalo misterioznom i intrigantnom nazivu, zapravo nije ništa drugo već primjena dubokih neuronskih mreža na razne vrste problema. Pod pojmom **duboke neuronske mreže** najčešće podrazumijevamo bilo koju unaprijednu neuronsku s većim brojem skrivenih slojeva. Iako ne postoji točna definicija koliko slojeva neuronska mreža mora imati kako bismo ju smatrali dubokom, često se uzima uvjet od 3 ili više (skrivenih) slojeva. [10]

Iako ideja dodavanja većeg broja slojeva u neuronske mreže nije ni u kojem smislu nova (prve "duboke" neuronske mreže pojavljuju se već 1990-ih godina),

glavna prepreka koja je dugo kočila razvoj ovog područja je bio nedostatak adekvatnog hardvera. Razvojem adekvatnog hardvera u drugoj polovici 2000-ih te u 2010-im godinama omogućio je adekvatne resurse potrebne za treniranje i rad dubokih neuronskih mreža. Odjednom je bilo moguće trenirati dublje modele nego prije, te za treniranje koristiti skupove podataka nekoliko redova veličine iznad dotadašnjih, što je rezultiralo puno boljim rezultatima. Sve ovo je kao posljedicu imalo pravu renesansu u području dubokih neuronskih mreža, koje je ubrzo postalo najpopularnije i najbrže rastuće područje strojnog učenja. [6]

Duboko učenje danas ostvaruje odlične rezultate u brojnim područjima primjene. Primjerice, desetak godina prije navedenih napredaka u dubokom učenju, područje raspoznavanja govora je stagniralo. Primjena metoda dubokog učenja rezultirala je u znatnom smanjenju pogrešaka modela, u nekim slučajevima i preko 50%. Duboke mreže također postižu sjajne rezultate u području računalnog vida, primjerice u poljima semantičke segmentacije ili klasifikacije.

U nastavku ćemo se fokusirati na primjenu dubokog učenja na računalni vid, specifičnije na semantičku segmentaciju. Razmotrit ćemo jednu posebnu vrstu dubokih neuronskih mreža posebno prikladnih za ovaj problem - duboke konvolucijske mreže.

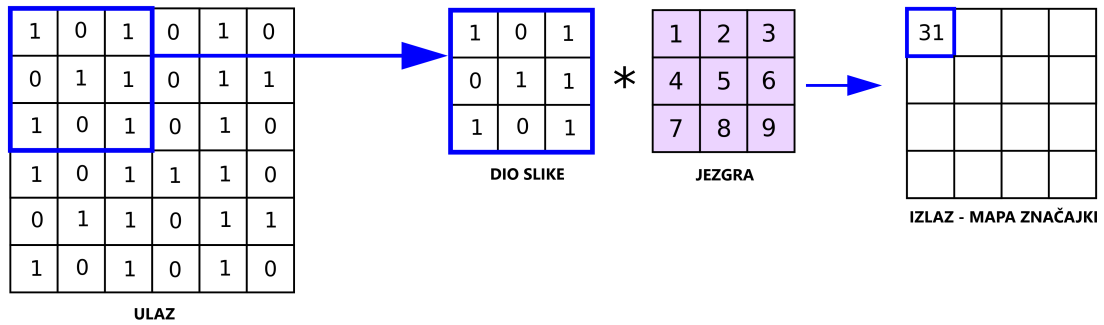
3.2.1. Konvolucijske neuronske mreže

Naša motivacija za uvođenje konvolucijskih neuronskih mreži je rješavanje problema iz domene računalnog vida. Glavni izvor podataka u tom području je naravno 2D slika. Ako želimo primijeniti duboko učenje na problem iz domene računalnog vida, moramo pronaći način kako 2D slike predati kao ulaz u model. Jedan naivan pristup bi mogao biti pretvoriti cijelu sliku u niz piksela, te svaki piksel dovesti kao jedan ulaz u mrežu. Zasiurno, mreža s takvim ulazima na raspolaganju ima sve podatke koji joj ikad mogu trebati! No uzmemo li u obzir da prosječna slika dimenzija 1980x1080 ima preko 2 milijuna piksela, vidimo gdje nastaje problem. Ovakav pristup bi podrazumijevao da samo ulazni sloj naše mreže ima 2 milijuna neurona - da ne spominjemo ostale slojeve. Treniranje i rad s ovakvom mrežom u praksi bi bili nemogući. Također, ovakav pristup praktički ignorira povezanost između susjednih piksela - npr. na slici koja prikazuje nebo, puno je veća vjerojatnost da u blizini jednog piksela plave boje pronađemo još piksela plave boje, nego primjerice piksel crvene boje. Očito je da moramo promijeniti naš pristup ovom problemu.

Konvolucijske neuronske mreže (engl. *convolutional neural networks, CNNs*) su specijalizirana vrsta umjetnih neuronskih mreža koje karakterizira upotreba konvolucije (specijalne vrste funkcije) umjesto matričnog množenja u barem jednom od svojih slojeva. Ova vrsta mreža je posebno dobra u obradi podataka sa mrežastom topologijom - uz što među ostalim spadaju i 2D slike. [6] Konvolucijske neuronske mreže se sastoje od 3 glavna sloja - konvolucijski, sloj sažimanja i potpuno povezani sloj. Rad konvolucijskih mreža objasniti ćemo na primjeru rada svakog od navedenih slojeva.

Konvolucijski sloj (engl. *convolutional layer*) je sloj čija je glavna zadaća ekstrakcija značajki iz originalne slike (ili pak iz izlaza prethodnog konvolucijskog sloja). Te značajke su primjerice rubovi, određene boje ili linije unutar slike. Ključni pojam u ovom sloju je **jezgra** (engl. *kernel*). U svojoj osnovi, jezgra je filter koji prolazi kroz originalnu sliku, dio po dio, te na svakom od tih dijelova provodi operaciju konvolucije. **Konvolucija** je operacija nad dvjema funkcijama, f i g , čiji je rezultat treća funkcija koja opisuje kako funkcija f utječe na oblik funkcije g . [11] Konkretno u našem primjeru, funkcija f bi bila jezgra (odnosno filter), a funkcija g originalna slika. Konvolucija se provodi jednostavnim matričnim množenjem i zbrajanjem dijela ulazne matrice (slike) koji se trenutno promatra i jezgre. Veličina ulazne matrice jednaka je veličini jezgre, dok je veličina pomaka po ulaznoj matrici određen parametrom koji nazivamo **korak** (engl. *stride*) - što je korak veći, izlaz konvolucijskog sloja će biti manjih dimenzija, i obratno. Rezultat konvolucije, odnosno primjene filtera na neki dio slike nam može dati informaciju je li element za koji je taj filter zadužen prisutan na tom dijelu slike ili ne. Primjerice ako se radi o filteru zaduženom za ekstrakciju rubova, mogao bi pridijeliti veće vrijednosti dijelovima slike na kojima se nalaze rubovi, a manje onima na kojima ne, tražeći mjesta gdje dolazi do nagle promjene svjetline ili intenziteta boja. Rezultat provođenja ove operacije, odnosno izlaz ovog sloja je nova matrica, manjih dimenzija od ulazne, koju zovemo **mapa značajki**. Važno je napomenuti da ako se ulazna slika sastoji od više kanala (primjerice RGB format), jezgra se mora sastojati od istog broja kanala, odnosno u primjeru za RGB format od 3 zasebna filtera, po jedan za svaki od kanala. U tom slučaju izlazna mapa značajki također ima 3 kanala. Taj parametar se naziva **dubina jezgre**. Učenjem konvolucijske mreže, vrijednosti jezgre se modificiraju kako bi preslikavanje ulaza na izlaznu mapu značajki bilo točnije. Postupak konvolucije

je prikazan na slici 3.4, gdje se također može primijetiti da veličina izlazne mape značajki ovisi o dimenziji ulaza te o dimenziji filtera.



Slika 3.4: Operacija konvolucije u CNN-u [12]

Nakon svakog konvolucijskog sloja, nad dobivenom mapom značajki primjenjuje se neka aktivacijska funkcija, koja je nužno nelinearna. Ako se sjetimo poglavlja 3.1.3, znamo da je nelinearnost aktivacijske funkcije bitna kako bi mreža bila u stanju modelirati složenije pojave. Najčešće korištena aktivacijska funkcija u konvolucijskim mrežama je **ReLU** (engl. *Rectified Linear Units*). U svojoj osnovi, ReLU je vrlo jednostavna funkcija:

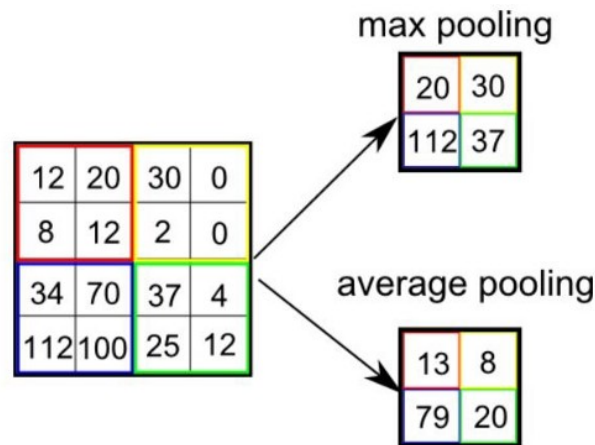
$$ReLU(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Glavne prednosti ReLU nad ostalim nelinearnim aktivacijskim funkcijama, poput sigmoide ili tangensa hiperbolnog su

1. Vrlo lagan izračun u odnosu na alternative - samo treba odrediti je li x veći ili manji od 0
2. Derivacija ReLU je uvijek 0 ili 1. Ovo je izuzetno korisno u rješavanju problema tzv. nestajućeg gradijenta (engl. *vanishing gradient*), koji je karakterističan za neke funkcije poput sigmoide. Problem se javlja zbog činjenice da derivacije sigmoidalnih funkcija teže 0 kada $x \rightarrow \infty$. ReLU, čija je derivacija uvijek 0 ili 1 nema takav problem, čime se omogućuje algoritmu propagacije pogreške da nastavi čak i za velike ulazne vrijednosti. [13]

Sloj sažimanja (engl. *pooling layer*) ima kao cilj smanjiti dimenziju mape značajki, te time smanjiti broj parametara i resursne zahtjeve modela. Također ima za ulogu učiniti model manje osjetljivim na manje promjene u ulazima, time poboljšavajući generalizaciju modela. Veličinu dijela mape značajki nad kojim se provodi sažimanje određuje **dimenzija sažimanja**, dok veličinu pomaka, kao i kod konvolucijskog sloja, određuje **korak**. Naravno, veličina koraka u ovom sloju nije ni na koji način povezana s veličinom koraka u konvolucijskom sloju. Sam postupak sažimanja je vrlo sličan postupku u konvolucijskom sloju - idemo po dijelovima mape značajki, te za svaki dio računamo sažetak koristeći funkciju sažimanja. Sažetak zatim mapiramo na izlaz ovog sloja.

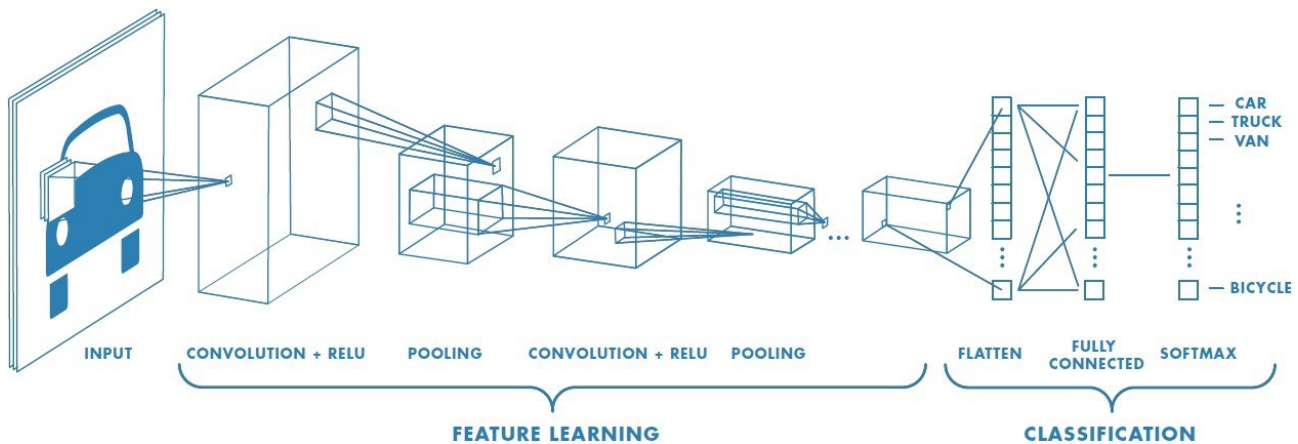
Razlikujemo dvije vrste sažimanja - **sažimanje po maksimalnoj vrijednosti** (engl. *max pooling*) te **sažimanje po prosječnoj vrijednosti** (engl. *average pooling*). Sažimanje po maksimalnoj vrijednosti za svaki dio ulaza (mape značajki) vraća njegovu maksimalnu vrijednost kao izlaz. Sažimanje po prosječnoj vrijednosti za svaki dio ulaza vraća njegovu prosječnu vrijednost. Između ova dva pristupa, preporučuje se korištenje sažimanja po maksimalnoj vrijednosti, jer uz smanjenje dimenzionalnosti, pridonosi i uklanjanju šuma iz ulaza. [14]



Slika 3.5: Sloj sažimanja [14]

U nekim primjenama, primjerice klasifikacija slika, dodaje se i treći sloj, tzv. **potpuno povezani sloj** (engl. *fully connected layer*). Ovaj sloj se nalazi pri kraju mreže, te na njega dolaze izlazi prethodna dva sloja, najčešće pretvoreni u jednodimenzionalni vektor. Ovaj dio konvolucijske mreže se zatim ponaša kao obična neuronska mreža.

Sve slojeve konvolucijske neuronske mreže možemo vidjeti na slici 3.6



Slika 3.6: Arhitektura konvolucijske neuronske mreže [14]

3.3. Primjena u računalnom vidu i semantičkoj segmentaciji

U ovom poglavlju ćemo pogledati kako se duboke neuronske mreže mogu iskoristiti za semantičku segmentaciju. Naglasak će biti na arhitekturama modificiranih konvolucijskih neuronskih mreža.

Generalni pristup semantičkoj segmentaciji korištenjem dubokih konvolucijskih modela možemo razdvojiti u dvije bitne komponente - **enkoder**, iza kojeg slijedi **dekoder**. Zadaća enkodera je obaviti klasifikaciju ulazne slike, na način opisan u poglavlju 3.2, no umjesto da rezultat klasifikacije koristeći potpuno povezani sloj dovedemo na izlaz mreže kao odluku o klasifikaciji slike, dovodimo ga na ulaz dekodera. Zatim je zadatak dekodera da naučenu klasifikaciju niže rezolucije projektira na izlaznu matricu, odnosno sliku, kako bi dobili gušću klasifikaciju, te u konačnici segmentacijsku mapu. Naravno, ovo je vrlo poopćen opis, te konkretni implementacijski detalji ovise o korištenoj arhitekturi.

U nastavku ćemo pogledati par istaknutih dubokih konvolucijskih modela koji, među ostalim, imaju primjenu u računalnom vidu i semantičkoj segmentaciji

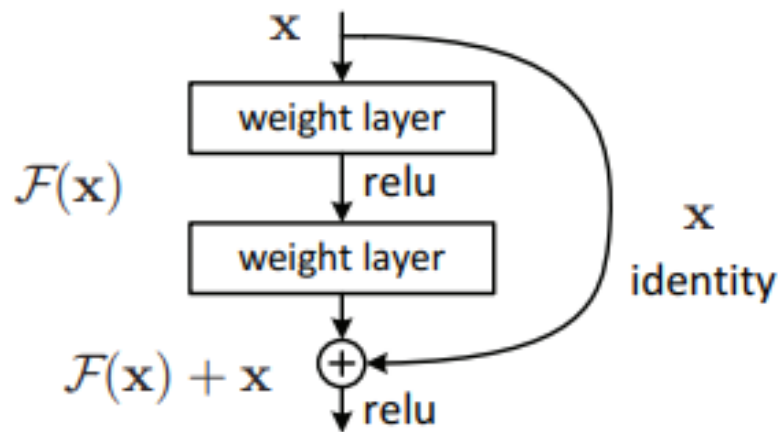
3.3.1. ResNet

ResNet je, uz AlexNet, jedan od najpoznatijih dubokih modela u svijetu računalnog vida. Ova mreža je poznata po svojoj velikoj dubini (152 sloja), te uvođenju koncepta rezidualne jedinice.

ResNet kreće od ideje da, iako povećanje dubine mreže rezultira boljim performansama, također otežava treniranje. Ovo je najviše zbog pojave problema nestajućeg gradijenta (već spomenut u poglavlju 3.2.1), te slične pojave zvane eksplodirajući gradijent (engl. *exploding gradient*). Oba ova problema postaju izraženija povećanjem dubine mreže.

Kako bi riješili ovaj problem, originalni autori uvode pojam **rezidualne jedinice** (engl. *residual block*), prikazanog na slici 3.7 Na slici vidimo da se ulazni podaci, prilikom ulaska u neki od konvolucijskih slojeva, također nepromijenjeni šalju nekoliko slojeva dalje. Ovo je korisno zato što omogućava lakšu propagaciju gradijenta prilikom propagacije pogreške unazad. Konkretni primjer - ako jedan od konvolucijskih slojeva radi više štete nego koristi - recimo vodi do povećavanja funkcije pogreške umjesto smanjenja, koristeći ResNet arhitekturu praktički ga možemo ignorirati, jer imamo drugi put do ostatka mreže, stvoren grananjem rezidualne jedinice. Također, pošto je taj "alternativni" put ustvari put prema nepromijenjenom ulazu, ako odaberemo taj put efektivno smanjujemo dubinu mreže. Na ovaj način, dodatni slojevi mreže se koriste samo ukoliko pomaže u smanjenju funkcije gubitka. Ukoliko ne pomažu, mreža modificira težine na takav način da se prilikom dolaska do te rezidualne jedinice uvijek odabere "zaobilazni" put, te na taj način efektivno smanji dubina mreže. U originalnom radu, [15] ResNet arhitektura je korištena za problem klasifikacije slika, no manjim modifikacijama arhitekture te uvođenjem dekodera, primjenjiva je i za probleme semantičke segmentacije.

ResNet nije glavna tema ovog rada, stoga u dodatne implementacijske detalje neću ulaziti, no zainteresiranom čitatelju za daljnje istraživanje preporučujem originalni članak o ResNet arhitekturi [15], te članak [16], koji na razumljiv i čitak način objašnjavaju ključne komponente arhitekture.



Slika 3.7: Rezidualna jedinica, ključna komponenta ResNet arhitekture [15]

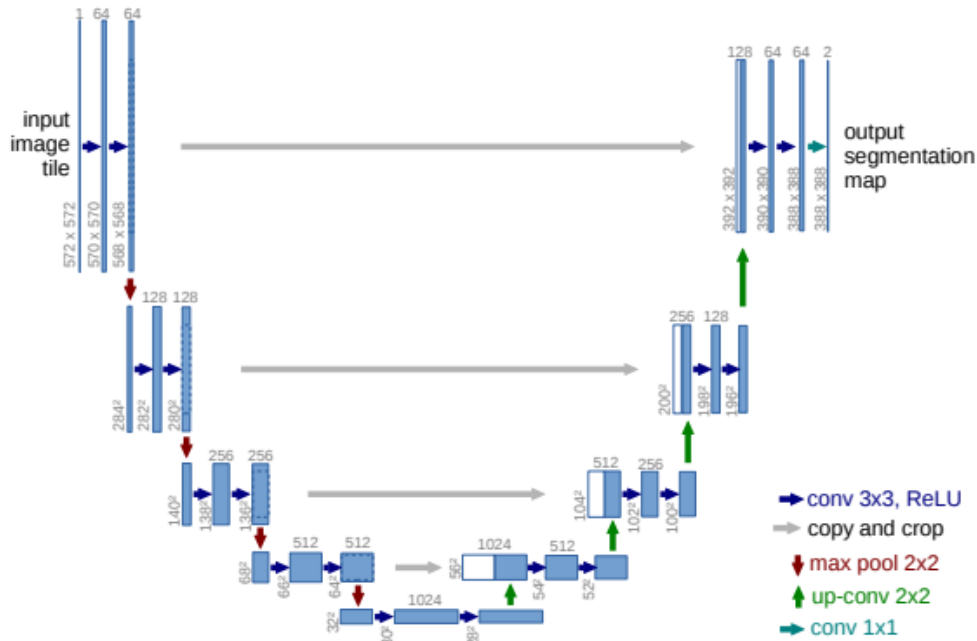
3.3.2. U-net

Osnovna motivacija nastanka U-net arhitekture je bila omogućiti dobre rezultate semantičke segmentacije uz relativno malenu količinu podataka. Razvijena je s ciljem segmentacije medicinskih slika, te je u tom području pokazala vrlo dobre rezultate. Formalno, ova arhitektura pripada u tzv. **potpuno konvolucijske mreže** (engl. *fully convoluted neural networks*, *FCNN*), konvolucijske mreže koje se sastoje samo od konvolucijskih slojeva i slojeva sažimanja.

Najbolji način za razumijevanje arhitekture ove mreže je kroz skicu arhitekture, prikazanu na slici 3.8

Na slici, svaki plavi kvadrat predstavlja višekanalnu mapu atributa. Broj na vrhu kvadrata predstavlja broj kanala u mapi, dok je sa strane dana i dimenzija mape. Strelicama različitih boja označene su operacije unutar mreže (legenda na slici).

Vidimo da se mreža sastoji od dva suprotna dijela - enkoderski dio, u kojem ulazna slika prolaskom kroz konvolucijske slojeve i slojeve sažimanja biva sve više i više poduzorkovana (engl. *downsampling*). Ovime dobivamo mapu značajki malih dimenzija pomoću koje model može raspoznati veća grupiranja piksela iste klase na ulaznoj slici. Međutim, kako je dobivena slika vrlo malih dimenzija, potrebno je provesti proces naduzorkovanja (engl. *upsampling*), u kojem se, koristeći proces "obrnute konvolucije", slici povećava rezolucija. Kako ne bi došlo do prekomjernog gubitka informacija prilikom povećanja rezolucije, koristan je velik broj kanala koji svaka mapa atributa ima. U zadnjem sloju dekoderskog dijela završavamo s mapom atributa jednake rezolucije kao i ulazna slika. U tom sloju se dodatno



Slika 3.8: U-net arhitektura. [17]

provodi 1x1 konvolucija, koja kao cilj ima mapirati dobivenu mapu atributa u odgovarajući broj klasa.

Prilikom treniranja, autori predlažu korištenje procesa augmentacije podataka, kako bi se poboljšalo svojstvo generalizacije mreže i smanjila količina potrebnih podataka.

3.3.3. Deeplab

Deeplab je zajednički naziv za obitelj dubokih neuronskih mreža razvijanih od strane Googlea. Trenutno postoje 4 verzije Deeplab arhitekture, od kojih je najnoviji DeeplabV3+ iz 2018.

U svojoj osnovi, Deeplab je modifikacija ResNet arhitekture, gdje se za proces enkodiranja, odnosno poduzorkovanja koristi manji broj slojeva sažimanja u kombinaciji sa **dilatiranjem konvolucijom** (engl. *atrous convolution*). Ova vrsta konvolucija omogućava obrade većeg dijela slike manjom jezgrom, odnosno polje slike nad kojim se primjenjuje konvolucija ima veće dimenzije od same jezgre. Nedostajući podaci se nadopunjuju nulama. Parametar koji određuje stupanj dilatacije konvolucije naziva se **stopa dilatacije**. Primjerice, jezgra dimenzija 3x3 sa stopom dilatacije 2 pokriva područje veličine 5x5 originalne slike. Ovakav pristup omogućava pokrivanje većeg dijela slike koristeći konvoluciju s manjim

brojem parametara. U Deeplab arhitekturi ovo svojstvo iskorištavamo tako da originalnu sliku poduzorkujemo manji broj puta nego što je to slučaj u ostalim arhitekturama. Umjesto toga, "povećavamo" efektivnu veličinu filtera koristeći dilatirane konvolucije umjesto običnih, te zatim te filtere primjenjujemo nad poduzorkovanim slikama. Primijetimo, dvije su glavne prednosti ovog pristupa: [18]

1. Smanjenjem potrebnog broja operacija poduzorkovanja, ubrzavamo rad modela
2. Kako konvolucije sad provodimo na slikama većih dimenzija (zbog manjeg broja poduzorkovanja), ovo nam omogućava da primjenom filtera izvučemo više značajki iz njih nego što je slučaj sa slikama manjih dimenzija, te samim time dobijemo točniju mapu značajki

DeeplabV2 i DeeplabV3 uvode još nekoliko modifikacija, poput piramidalnog dilatiranog sažimanja i oštrije detekcije rubova na slikama.

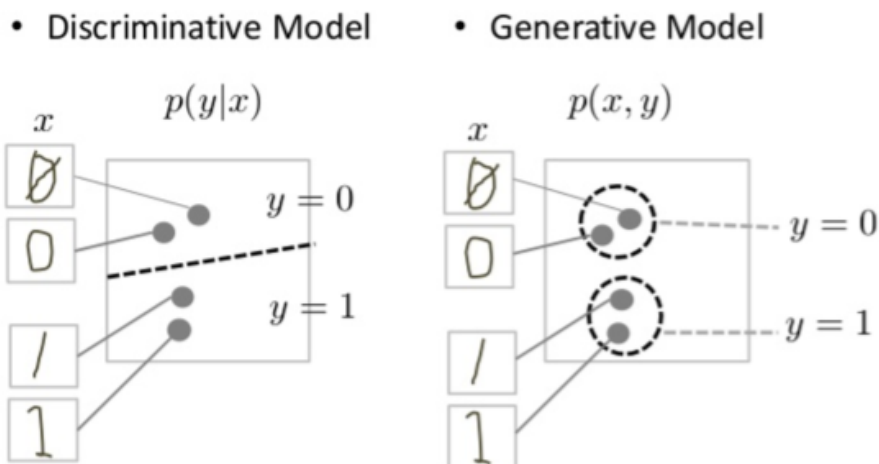
4. Generativni modeli

U ovom poglavlju pogledat ćemo jedan drugačiji pristup učenju. Tehnike i arhitekture s kojima smo se do sada bavili mogli bismo okarakterizirati kao **diskriminativne** - cilj nam je bio povezati uzorke skupa za učenje sa odgovarajućim oznakama (klasa slike, oznake piksela itd.) S druge strane, **generativni** pristup nastoji što točnije naučiti distribuciju podataka za učenje, kako bi na osnovu njih mogao generirati nove podatke iz te distribucije. Primijetimo da rezultati ova dva pristupa mogu biti identični, no način na koji dolazimo do rezultata je fundamentalno različit. Za određene primjene, generativni modeli su se u praksi pokazali kao puno bolji izbor od diskriminativnih.

Malo formalnije, ako imamo zadan skup ulaznih podataka X te set očekivanih izlaza Y , diskriminativni modeli pokušavaju naučiti uvjetnu vjerojatnost $P(Y|X)$. Dakle, diskriminativne modele ne zanima koja je vjerojatnost pojave takvog para, nego samo što će biti s izlazom Y ako se na ulazu pojavi X . S druge strane, generativni modeli uče vjerojatnost $P(X, Y)$, odnosno mogu naučiti koja je vjerojatnost pojave konkretnog ulaza X i ulaza Y u isto vrijeme. Dakle oni uče i koja je vjerojatnost pojave određene kombinacije ulaza i izlaza

Konačno, pogledajmo razliku u pristupima diskriminativnog i generativnog modela na jednom konkretnom primjeru, prikazan na slici 3.9. Primjer je preuzet iz [19]

Razlika je objašnjena u razlici pristupu problemu klasifikacije slika rukom napisanih znamenki, vrlo čestog primjera u području računalnog vida. Diskriminativni model nastoji odrediti razliku između dvije prikazane klase, 0 i 1, povlačenjem linije u prostoru svih podataka. Ta linija dijeli prostor na dva dijela - prostor u kojem su podaci koje je potrebno klasificirati kao 0, te prostor u koji pripadaju podaci koje je potrebno klasificirati kao 1. Ako model dobro "pogodi" poziciju te linije, može uspješno klasificirati dane ulaze u jednu od te dvije klase, bez da ikad mora naučiti koja je točna pozicija konkretnih ulaza u tom prostoru. S



Slika 4.1: Diskriminativni vs generativni pristup [19]

druge strane, generativni model nastoji naučiti točne pozicije konkretnih ulaza u prostoru podataka, dakle moraju naučiti na koji način su podaci **distribuirani** u prostoru podataka. Očito je da je to znatno složeniji pristup u odnosu na diskriminativni model, no to mu također omogućuje generiranje novih podataka iz dane distribucije.

U nastavku ćemo detaljnije pogledati primjere danas vrlo popularne obitelji arhitektura generativnih modela, zasnovanih na GAN arhitekturi.

4.1. GAN

Generativna suparnička mreža (engl. *generative adversarial network*, *GAN*) je naziv za arhitekturu dubokih generativnih modela prvi put predloženu od strane Goodfellowa i dr. u njihovom poznatom radu [20].

Autori predlažu arhitekturu koja se sastoji od 2 suparničke mreže - **generatora**, *G* i **diskriminatora** *D*. *G* je generativni model čija je zadaća što točnije naučiti distribuciju ulaznih podataka te generirati nove ("lažne") primjerke iz te distribucije. *D* je diskriminativni model čija je zadaća za dani ulaz odrediti vjerojatnost dolazi li iz skupa ulaznih (originalnih, pravih) podataka ili se radi o "lažnom" primjerkul ulaza koji je *G* generirao. *G* se trenira tako da nastoji maksimizirati vjerojatnost da *D* napravi pogrešku, dok *D* treniramo tako da njegove procjene budu što točnije. Ovakav pristup odgovara igri **minimaks** sa 2 igrača, *G* i *D*, u kojoj svaki igrač nastoji maksimizirati vjerojatnost svoje pobjede i mi-

nimimizirati vjerojatnost pobjede protivnika (provjeri je li ovo dobra interpretacija minimaxa!). Kako želimo da generator generira što točnije uzorke iz ciljne distribucije, konačni cilj ove igre je situacija u kojoj je G savršeno naučio ulaznu distribuciju i može generirati podatke praktički jednake onima iz ulazne distribucije, dok D za svaki ulaz daje vjerojatnost 0.5 jer ne može razaznati radi li se o originalnom ili "lažnom" podatku.

Kako bismo formalno definirali funkciju cilja ove arhitekture, uvedimo sljedeću nomenklaturu. Neka nam x predstavlja određeni ulazni podatak (primjerice 2D sliku). Sa $D(x)$ ćemo označiti diskriminatorsku mrežu koja za dani ulaz x vraća skalar - vjerojatnost da x dolazi iz podataka za treniranje, a ne iz generatora. Ukoliko x dolazi iz podataka za treniranje, očekujemo da će izlaz $D(x)$ biti velik, dok za slučaj kad je x izlaz generatora očekujemo da će $D(x)$ biti malen. $D(x)$ se dakle ponaša kao običan binarni klasifikator. Nadalje, sa z označimo slučajni vektor iz normalne distribucije. $G(z)$ će onda biti generatorska funkcija koja mapira z na prostor ulaznih podataka. Kao što smo već rekli, cilj $G(z)$ je što točnije naučiti distribuciju ulaznih podataka kako bi to mapiranje bilo što točnije. Iz navedenog slijedi da sa $D(G(z))$ možemo označiti vjerojatnost s kojom je $D(x)$ siguran da izlaz generatora $G(z)$ dolazi iz originalnog skupa podataka. Konačno, funkciju cilja ove arhitekture možemo definirati kao:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{x \sim p_z(z)} [\log(1 - D(G(z)))] \quad (4.1)$$

gdje $\mathbb{E}_{x \sim p_{data}(x)}$ predstavlja očekivanu vrijednost nad svim pravim ulazima, a $\mathbb{E}_{x \sim p_z(z)}$ očekivanu vrijednost nad svim lažnim (generiranim) ulazima. D nastoji maksimizirati $\log D(x)$, vjerojatnost da će uspješno klasificirati ulazne podatke. G nastoji minimizirati vjerojatnost ($\log(1 - D(G(z)))$), odnosno da će D ispravno klasificirati njegove izlaze kao lažne.

Autori također navode da u ranim fazama treniranja, dok generator još nije dobro naučio ciljnu distribuciju ulaza, D može raspoznavati prave od lažnih ulaza s vrlo visokom točnošću, što može dovesti do zasićenja $\log(1 - D(G(z)))$, odnosno do toga da GAN "zapne" prilikom treniranja. Kao alternativu možemo trenirati G da maksimizira $\log D(G(z))$ umjesto da minimizira $\log(1 - D(G(z)))$. Ovo u osnovi predstavlja istu ideju, samo što ovaj rezultira puno strmijim gradijentima u ranoj fazi treniranja.

Treniranje GAN-a je zahtjevnije od treniranja ostalih dubokih modela iz dva razloga:

- (a) Moramo uspješno trenirati 2 mreže: generator i diskriminator
- (b) Konvergenciju GAN-a je teško definirati, a time i otkriti

Stoga GAN-ove obično treniramo na tako da prvo fiksiramo parametre generatora i treniramo diskriminator jednu ili više epoha. Ovo je potrebno jer u suštini diskriminator treniramo da prepozna mane u generatoru; kad bi istovremeno mijenjali diskriminator i generator, to bi bio gotovo nemoguć problem. Slično u sljedećem koraku: fiksiramo parametre diskriminatora te treniramo generator. Također, bitno je treniranje zaustaviti na vrijeme, obično kada diskriminator više ne može razlikovati prave ulaze od lažnih. Ako nastavimo trenirati nakon te točke, generator se trenira na pogrešnim povratnim informacijama od diskriminatora, te može doći do pogoršanja njegovih performansi.

Puni algoritam za treniranje dajem u nastavku.

Algoritam 2 Algoritam treniranja generativnih suparničkih mreža

for broj iteracija **do**

for k koraka **do**

- Uzmi m slučajnih uzoraka šuma $(z^{(1)} \dots z^{(m)})$
- Uzmi m ulaznih primjera $(x^{(1)} \dots x^{(m)})$ iz skupa ulaznih podataka
- Ažuriraj diskriminator penjući se njegovim stohastičkim gradijentom:

$$\nabla_{\Theta_d} \cdot \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$

end for

- Uzmi m slučajnih uzoraka šuma $(z^{(1)} \dots z^{(m)})$
- Ažuriraj generator spuštajući se njegovim stohastičkim gradijentom:

$$\nabla_{\Theta_g} \cdot \frac{1}{m} \sum_{i=1}^m \log(1 - d(G(z^{(i)})))$$

end for=0

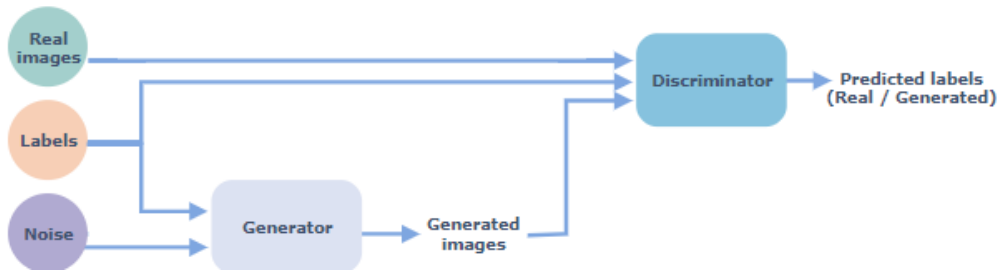
4.2. cGAN

Uvjetne generativne suparničke mreže (engl. *conditional generative adversarial networks, cGAN*), prvi put predstavljene u [21], su nadogradnja na postojeću GAN arhitekturu nastale s ciljem usmjeravanja izlaza GAN-a. U običnoj GAN arhitekturi, ne možemo kontrolirati izlaz mreže, jer se izlaz bazira na slučajnom vektoru koji GAN mapira u odgovarajući izlaz. cGAN rješava ovaj problem tako da prilikom treniranja provodi uvjetovanje diskriminatora i generatora pomoću dodatne informacije, y . Ta dodatna informacija može biti bilo što, poput oznaka klasa, segmentacijskih mapa itd. Na taj način generator nauči da prilikom generiranja izlaza mora uzeti u obzir informaciju y . Jednom kada smo dovršili treniranje i želimo pokrenuti naš model u produkciji, za dobivanje željenog izlaza, primjerice segmentacijske mape dane slike, dovoljno je da kao ulaz u model damo sliku čiju segmentacijsku mapu želimo dobiti. Ta segmentacijska mapa predstavlja dodatnu informaciju y uz pomoć koje cGAN generira izlaz - traženu segmentacijsku mapu dane slike.

Modificirana funkcija cilja cGAN-a je:

$$\min_G \max_d V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x|y)] + \mathbb{E}_{x \sim p_z(z)} [\log(1 - D(G(z|y)))] \quad (4.2)$$

Jedna poznata primjena cGAN modela je pix2pix arhitektura, koja služi za pretvorbu jedne vrste slika u drugu (npr. možemo naučiti mrežu da mapira slike konja na slike zebri).



Slika 4.2: Arhitektura cGAN mreže [22]

4.3. DCGAN

Još ću se vrlo kratko osvrnuti na DCGAN arhitekturu [23], modifikaciju GAN arhitekture posebno prilagođene za rad sa slikama. **Duboke konvolucije generativne suparničke mreže** (engl. *deep convolutional adversarial generative networks*, *DCGAN*) nadograđuju klasični GAN model uvodeći sljedeće izmjene:

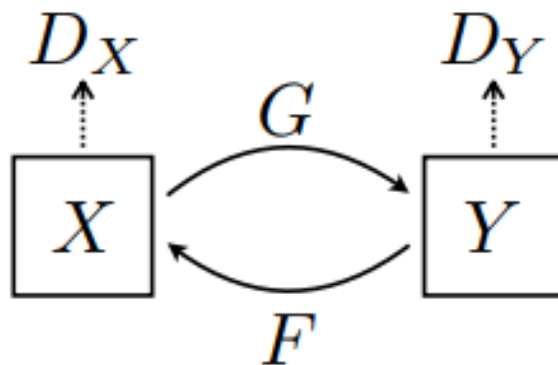
- • Slojeve sažimanja u diskriminatoru mijenja s konvolucijom s koracima (engl. *strided convolution*), dok slojeve sažimanja u generatoru mijenja s konvolucijom s djelomičnim koracima (engl. *fractional-strided convolution*). Ovo omogućava mreži da sama nauči provoditi poduzorkovanje
- • Koristi *batch* normalizaciju u generatoru i diskriminatoru
- • Uklanja potpuno povezane slojeve u dubljim arhitekturama
- • Koristi ReLU aktivacijsku funkciju u svim slojevima, osim u izlaznoj gdje koristi tangens hiperbolni
- • Koristi *LeakyReLU* kao aktivacijsku funkciju u svim slojevima diskriminatora

Ove promjene poboljšavaju učenje, te u konačnici daju bolje rezultate u odnosu na klasični GAN. Također, autori su nakon treniranja mreže uspješno izdvojili diskriminatorski dio mreže te ga (samostalno) koristili kao klasifikator.

4.4. cycleGAN

Glavna motivacija za uvođenje arhitekture **cikličnih generativnih suparničkih mreža** (engl. *cycle generative adversarial networks*, *cycleGAN*) je problem prevođenja slika iz domene A u domenu B. Primjerice, želimo naučiti mrežu da za danu sliku livade u proljeće generira sliku iste livade ali recimo zimi. Ili na osnovu poznatog umjetničkog djela, primjerice Van Goghove *Zvezdane noći*, dobiti sliku koja izgleda kao fotografija krajolika na osnovu kojeg je djelo naslikano. Tradicionalni pristupi ovakvom problemu zahtijevaju velik broj povezanih parova slika iz domene A i domene B, odnosno za ulaznu sliku iz domene A model zahtijeva **istu** tu sliku, samo iz domene B. Konkretno na našem primjeru prevođenja slika livada u proljeće u slike, postojeći modeli zahtijevaju da im kao ulaz damo puno slika livada u proljeće, ali i u zimu. Očito je kako je u određenim situacijama vrlo nezgodno, pa čak i nemoguće doći do takvih parova (kako doći do fotografije krajolika koji je Van Gogh gledao kad je slikao *Zvezdanu noć*?)

Arhitektura cycleGAN, prvi put opisana u [24], rješava navedeni problem. U klasičnom (c)GAN pristupu, generator bi naučio mapiranje $G : X \rightarrow Y$, gdje je X početna, a Y ciljna domena. U kombinaciji s diskriminatorom koji bi učio razliku između y (originalnih ulaza) i \hat{y} (generiranih ulaza), G bi u idealnom slučaju savršeno naučio distribuciju od Y , te bi generirane slike bilo nemoguće raspoznati od originalnih slika. Međutim, ovo uvijek ne garantira da će originalni x i generirani y biti povezani na smislen način, jer postoji beskonačno mnogo mapiranja $G : X \rightarrow Y$ koji će rezultirati savršenom distribucijom y . Također, kod ovakvog pristupa pojavljuje se *mode collapse* problem, gdje se svi ulazni x mapiraju na isti y , te učenje stagnira. Postoje određena rješenja ovih problema (evidentno iz postojanja pix2pix cGAN arhitekture spomenute u 4.2), no autori za rješavanje ovih problema predlažu da bi treniranje modela trebalo biti *ciklički konzistentno*, odnosno da model paralelno uz učenje mapiranja $G : X \rightarrow Y$, uči i inverzno mapiranje $F : Y \rightarrow X$, odnosno G i F moraju biti inverzne funkcije, te ujedno i bijekcije. Tu praksi znači da ćemo u mreži imati dva generatora i dva diskriminatora. Prvi generator, G , uči mapiranje $G : X \rightarrow Y$. Prvi diskriminator, D_y , uči razlikovati originalne slike iz skupa X i slike iz X koje generira G . Drugi generator, F , uči obrnuto mapiranje, $F : Y \rightarrow X$, a drugi diskriminator, D_x , kako razlikovati originalne slike iz Y , te slike koje generira F . Slika 4.3 prikazuje navedenu arhitekturu.



Slika 4.3: Generatori i diskriminatori cycleGAN arhitekture [24]

Kako bi uspješno trenirali ovu mrežu, uvodimo nekoliko novih funkcija gubitka. Za generator G i pripadajući mu diskriminator D_y izražavamo cilj kao:

$$\mathcal{L}_{GAN}(G, D_y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))] \quad (4.3)$$

Kao i kod klasičnog GAN-a, G nastoji minimizirati ovu funkciju, dok ju D nastoji maksimizirati. Na ekvivalentan način definiramo i cilj za F i D_x :

$$\mathcal{L}_{GAN}(F, D_x, X, Y) = \mathbb{E}_{x \sim p_{data}(x)}[\log D_X(x)] + \mathbb{E}_{y \sim p_{data}(y)}[\log(1 - D_X(F(y)))] \quad (4.4)$$

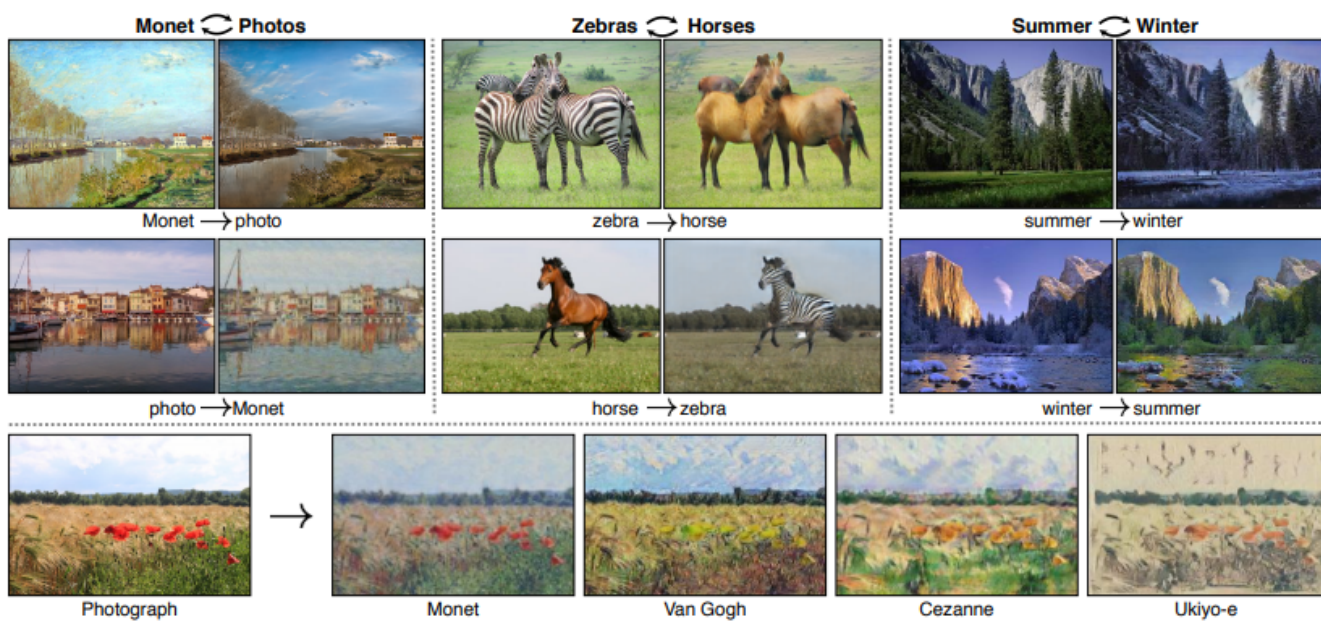
Konačno, uvodimo pojam **ciklične funkcije gubitka** (engl. *cycle loss*), kako bi osigurali da mapiranja G i F budu **ciklički konzistentna**. Odnosno da za svaku ulaznu sliku x , prolaskom kroz cijeli krug (ciklus) naše mreže, kao izlaz generatora F ponovno trebamo dobiti sliku x . Odnosno, želimo postići $x \rightarrow G(x) \rightarrow F(G(x)) = x$. Ovu relaciju zovemo **unaprijedna ciklična konzistentnost**. Također želimo postići i obrnuto, odnosno za svaku sliku iz domene Y, treba vrijediti $y \rightarrow F(y) \rightarrow G(F(y)) = y$. Model potičemo da nauči ovakvo ponašanje uvođenjem cikličke funkcije gubitka:

$$\mathcal{L}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1] \quad (4.5)$$

Ukupnu funkciju gubitka dobivamo zbrajanjem navedenih funkcija:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_x, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) \quad (4.6)$$

Na slici 4.4 vidimo neke od brojnih mogućih primjena cycleGAN arhitekture.



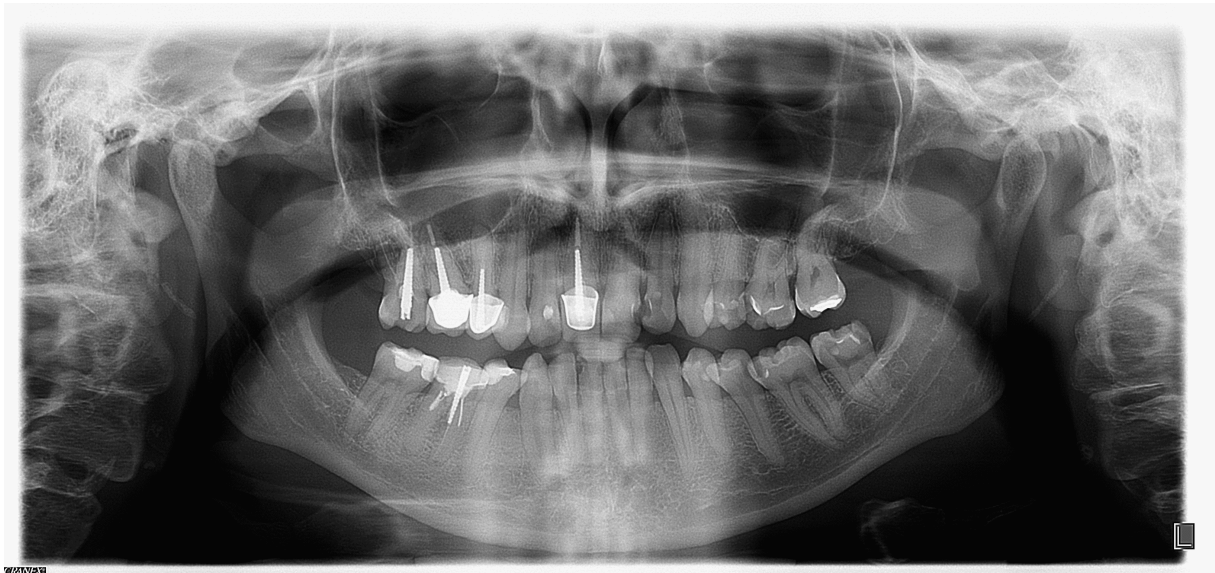
Slika 4.4: Primjene cycleGAN arhitekture [24]

5. Korištenje cycleGAN arhitekture za semantičku segmentaciju ortopanograma

5.1. Opis problema

Jedan od najčešće korištenih, pa i najkorisnijih dijagnostičkih alata u području stomatologije su zasigurno rendgenske panoramske snimke čeljusti. Relativno su jednostavne za provođenje, a stomatologu daju velik broj informacija o stanju pacijenta. Međutim, u određenim situacijama, poput prisustva šuma u ortopanogramu, ili većeg broja umjetnih zubi/implanata čak i najboljem stomatologu može biti zahtjevno odrediti točnu poziciju svakog od zuba u čeljusti. Stoga bi programsko rješenje koje bi omogućilo automatsko označavanje svakog zuba u ortopanogramu zasigurno bilo korisno, te bi se, uz primjenu kao pomoćni alat stomatolozima, moglo iskoristiti kao početna točka daljnjim nadogradnjama, poput automatske detekcije bolesti zuba ili slično. Stoga u ovom poglavlju predlažem rješenje za semantičku segmentaciju stomatoloških panoramskih rendgenskih snimaka, temeljnog na cycleGAN arhitekturi.

Kao podatke za treniranje predloženog modela na raspolaganju sam imao dvije vrste ortopanograma: 376 ortopanograma te njihovih pripadajućih segmentacijskih mapa (označenih ručno), te 3950 neoznačenih ortopanograma, odnosno ortopanograma bez pripadajuće segmentacijske mape. Upravo ovakva distribucija podataka, gdje na raspolaganju imamo relativno malen broj označenih slika ali relativno velik broj neoznačenih, je bila glavna motivacija za uvođenje cycleGAN arhitekture. Ta arhitektura bi u teoriji, uz sve označene slike, trebala moći iskoristiti i neoznačene slike prilikom treniranja kako bi poboljšala generalizaciju.



(a) Ortopanogram



(b) Ortopanogram + segmentacijska mapa

Slika 5.1: Ortopanogram i njegova segmentacijska mapa

5.2. Opis predloženog rješenja

Korišteno je rješenje prvi put predloženo u [25]. U tom radu autori predlažu iskorištavanje mogućnosti cycleGAN-a za učenje bidirekcijskog mapiranja korištenjem neuparenih slika iz domene A i domene B za polunadziranu semantičku segmentaciju. Navode da korištenjem svojstva cikličke konzistentnosti cycleGAN-a (opisanog u 4.4) model može naučiti bidirekcijsko mapiranje između **neoznačenih** slika i segmentacijskih mapa. Ovime dodajemo određeni nenadzirani regularizacijski efekt koji poboljšava treniranje, te također imamo način za iskoristiti i neoznačene slike u modelu, do kojih je često puno lakše doći nego do označenih. Baš zato što, uz klasične parove (slika, segmentacijska mapa) koristimo i dodatne neoznačene slike, koje nemaju svoje segmentacijske mape model vrši oblik **polunadziranog** (engl. *semisupervised*) učenja. Autori su proveli testiranje predloženog modela na tri skupa podataka (PASCAL VOC 2012, Cityscapes, ACDC), te zaključili da model postiže 2-4% bolje rezultate u odnosu na odgovarajući potpuno nadzirani model.

U nastavku ću detaljnije opisati arhitekturu korištenog rješenja, kao i modifikacije potrebe za rad s ortopanogramima.

5.2.1. Arhitektura rješenja

Predložena arhitektura rješenja nešto je drugačija od običnog cycleGAN-a. Kao i kod običnog cycleGAN modela, predloženi model ima 2 generatora i 2 diskriminatora. Zadaća prvog generatora, kojeg ćemo označiti sa \mathbf{G}_{IS} (od engl. *image to segmentation*), je mapirati slike (ortopanograme) na njihove odgovarajuće segmentacijske mape. Prvi diskriminator, \mathbf{D}_S , nastoji razlikovati te generirane segmentacijske mape od originalnih. Drugi generator, \mathbf{G}_{SI} (engl. *segmentation to image*), uči kako mapirati segmentacijsku mapu s odgovarajućom slikom tj. ortopanogramom. Ovaj generator se koristi samo za poboljšanje treniranja. Konačno, drugi diskriminator \mathbf{D}_I kao ulaz prima sliku ortopanograma te nastoji odrediti radi li se o stvarnoj ili generiranoj slici. Kako bi se osigurala ciklička konzistentnost, generatori se treniraju tako da kad segmentacijsku mapu koju je generirao G_{IS} predamo u G_{SI} kao izlaz dobijemo originalnu sliku ortopanograma. I obrnuto, kad generatoru G_{IS} predamo sliku koju je generirao G_{SI} , kao izlaz dobiti originalnu segmentacijsku mapu koju je G_{SI} primio kao ulaz.

No, nije sve tako jednostavno. Uz dodatak opisanim generatorima i diskriminatorima, model ima još dva para generatora, nazovimo ih \hat{G}_{IS} , \hat{G}_{SI} , te još

jedan dodatni diskriminator, \hat{D}_I . Zanimljivo je da se ovi dodatni generatori i diskriminatori ne spominju u originalnom radu, već su ih autori naknadno dodali u **službeni** kod modela, priložen radu, uz još nekoliko izmjena. Jedino objašnjenje koje je dano je da se radi o "daljnjim poboljšanjima". Ove 3 mreže se u kodu isključivo koriste na mjestima gdje bi se u običnom cycleGAN-u računala tzv. ciklična pogreška slika. Koriste se na način da diskriminator \hat{D}_I odlučuje dolazi li rekonstruirana slika iz ciklusa \hat{G}_{IS} , \hat{G}_{SI} , ili iz ciklusa G_{IS} , G_{SI} . Iako nisam sasvim siguran zašto su autori odlučili uvesti ove izmjene, u praksi se pokazalo da značajno pridonose uspješnosti modela (mIoU 0.5 za originalni model vs mIoU 0.65 za modificirani model, 33 klase). Iz tog razloga odlučio sam koristiti modificiranu arhitekturu umjesto one predložene u samom radu.

Kako bi formalnije definirali predloženu arhitekturu, kao i korištene funkcije gubitka, uvedimo sljedeću nomenklaturu: sa $\mathcal{X}_{\mathcal{L}}$ označimo skup označenih slika (dakle ortopanograma koji u skupu segmentacijskih maski imaju svoj par), sa $\mathcal{Y}_{\mathcal{L}}$ skup svih originalnih segmentacijskih maski (engl. *ground truth*), te sa $\mathcal{X}_{\mathcal{U}}$ skup svih neoznačenih slika ortopanograma (dakle ortopanograma koji nemaju svoj par u skupu segmentacijskih maski).

Uvodimo prvu funkciju gubitka, L_{gen}^S . Radi se o nadziranoj funkciji pogreške za segmentaciju, čija je zadaća naučiti G_{IS} da, na osnovu predane slike ortopanograma generira odgovarajuću segmentacijsku mapu. Funkciju definiramo kao:

$$L_{gen}^S(G_{IS}) = \mathbb{E}_{x,y \sim \mathcal{X}_{\mathcal{L}}, \mathcal{Y}_{\mathcal{L}}}[\mathcal{H}(y, G_{IS}(x))] \quad (5.1)$$

gdje \mathcal{H} predstavlja **unakrsnu entropiju** (engl. *cross-entropy*) po pikselima:

$$\mathcal{H}(y, \hat{y}) = \sum_{j=1}^N \sum_{k=1}^K y_{j,k} \log \hat{y}_{j,k} \quad (5.2)$$

U danoj jednadžbi za \mathcal{H} , $y_{j,k}$ predstavlja vjerojatnosti da piksel $j \in (1, \dots, N)$ originalne segmentacijske mape ima oznaku $k \in (1, \dots, K)$, gdje je K broj oznaka. Kako se radi originalnim oznakama, za $y_{j,k}$ ta vjerojatnost je uvijek 1 ili 0 - piksel ili ima tu oznaku ili nema. $\hat{y}_{j,k}$ predstavlja slične vjerojatnosti, samo za generiranu segmentacijsku mapu.

Za G_{SI} koristimo L1 normu kao nadziranu funkciju gubitka, kako bi procijenili razliku između originalne označene slike, te slike koju G_{SI} generira iz njene pripadajuće segmentacijske mape:

$$L_{gen}^I(G_{SI}) = \mathbb{E}_{x,y \sim \mathcal{X}_{\mathcal{L}}, \mathcal{Y}_{\mathcal{L}}} [\|G_{SI}(y) - x\|_1] \quad (5.3)$$

Prisjetimo se, L1 norma se računa kao apsolutna razlika između dobivenih i očekivanih vrijednosti.

Kako bi iskoristili neoznačene ortopanograme, uvodimo još dvije vrste funkcija gubitka: suparničke funkcije gubitka i funkcije gubitka za cikličku konzistentnost.

Ako je $D_S(y)$ predviđena vjerojatnost da segmentacijska mapa y odgovara originalnoj segmentacijskoj mapi, suparničku funkciju gubitka za D_S možemo definirati kao:

$$L_{disc}^S(G_{IS}, D_S) = \mathbb{E}_{y \sim \mathcal{Y}_{\mathcal{L}}} [MSE((D_S(y)))] + \mathbb{E}_{x' \sim \mathcal{X}_{\mathcal{U}}} [MSE((D_S(G_{IS}(x'))))] \quad (5.4)$$

Vidimo da se ustvari radi u funkciji srednje kvadratne greške (engl. *mean squared error*, *MSE*). Funkcija se računa na osnovu uspjeha diskriminatora u razlikovanju pravih i lažnih segmentacijskih mapa.

Funkciju gubitka za drugi diskriminator, D_I , definiramo na ekvivalentan način:

$$L_{disc}^I(G_{SI}, D_S) = \mathbb{E}_{y \sim \mathcal{Y}_{\mathcal{L}}} [MSE(D_S(y))] + \mathbb{E}_{x' \sim \mathcal{X}_{\mathcal{U}}} [MSE((D_S(G_{IS}(x'))))] \quad (5.5)$$

Prvu cikličku funkciju gubitka definiramo za segmentacijske mape. Kao što smo rekli, za svaki y (segmentacijsku mapu) očekujemo sljedeće: $G_{IS}(G_{SI}(y)) = y$. Ovakvo ponašanje možemo poticati koristeći sljedeću funkciju gubitka:

$$L_{cycle}^s(G_{IS}, G_{SI}) = \mathbb{E}_{y \sim \mathcal{Y}_{\mathcal{L}}} [\mathcal{H}(y, G_{IS}(G_{SI}(y)))] \quad (5.6)$$

Koristimo unakrsnu entropiju jer su labele u segmentacijskim mapama kategorijske varijable.

Pogledajmo još uloge \hat{G}_{IS} , \hat{G}_{SI} i \hat{D}_I . Prilikom treniranja generatora, \hat{D}_I na ulazu prima rekonstruiranu sliku iz ciklusa $G_{SI}(G_{IS}(\mathcal{X}_{\mathcal{U}}))$, te određuje dolazi li rekonstruirana slika iz ciklusa $G_{SI}(G_{IS}(\mathcal{X}_{\mathcal{U}}))$, ili iz ciklusa $\hat{G}_{SI}(\hat{G}_{IS}(\mathcal{X}_{\mathcal{U}}))$. Na osnovu te odluke, koristeći MSE, računa se funkcija gubitka za \hat{D}_I . Tu funkciju gubitka možemo označiti sa $L_{D_I}^1$.

Tijekom druge faze treniranja, kada treniramo diskriminatore a generatori su fiksirani, računamo funkciju gubitka $L_{D_I}^2$, kao:

$$L_{D_I}^2 = MSE(\hat{D}_i(\hat{G}_{SI}(\hat{G}_{IS}(x))), 0) + MSE(\hat{D}_i(G_{SI}(G_{IS}(x))), 1) \quad (5.7)$$

Primijetimo, ova funkcija je slična $L_{D_I}^1$, samo što u nju ulazi i odluka o rekonstruiranoj slici iz ciklusa $\hat{G}_{SI}(\hat{G}_{IS}(\mathcal{X}_U))$.

Konačno, ukupnu funkciju gubitka možemo napisati kao:

$$L_{total} = \lambda_1 L_{gen}^S(G_{IS}) + \lambda_2 L_{gen}^I(G_{SI}) + \lambda_3 L_{cycle}^s(G_{IS}, G_{SI}) - \lambda_4 L_{disc}^S(G_{IS}, D_S) - \lambda_5 L_{disc}^I(G_{SI}, D_S) + \lambda_6 L_{D_I}^1 - \lambda_7 L_{D_I}^2$$

U praksi, treniramo generatore dok su parametri diskriminatora fiksirani, i obrnuto.

Kao što je već rečeno, generatori G_{IS} i G_{SI} zasnivani su na DeeplabV2 arhitekturi, dok su \hat{G}_{IS} i \hat{G}_{SI} zasnivani na ResNet arhitekturi. Korišteni su diskriminatori koji rade na razini piksela, odnosno diskriminator za svaki piksel donosi odluku je li lažan ili ne. Diskriminatori se sastoje od 3 konvolucijska sloja, nakon kojih dolazi *Leaky ReLU* aktivacijska funkcija sa $\alpha = 0.2$. Tijekom treniranja također je korišten Adam optimizator s parametrima $\beta_1 = 0.5$ i $\beta_2 = 0.999$. Stopa učenja postavljena je na 0.0002, s linearnim padom nakon svakih 100 epoha. U svim eksperimentima veličina grupe (engl. *batch*) je bila 2. Ukoliko u rezultatima modela nije drugačije navedeno, svi parametri λ su jednaki 1.0.

Uz navedeni polunadzirani cycleGAN model, paralelno je treniran i nadzirani model (samo jedan generator, odnosno samostalna DeeplabV2 arhitektura), koji se bavi isključivo segmentacijom ortopanograma (dakle, nema veze s GAN-ovima). On je korišten kao usporedna točka za performanse cycleGAN modela.


5.2.2. Programska implementacija

Za programsku implementaciju korišten je programski jezik Python uz knjižnicu **Pytorch**, a za praćenje treniranja modela koristan je bio alat **Tensorboard**. Modifikacije koda većinom su uključivale prilagođavanje nove vrste podataka -

ortopanograma i njihovih segmentacijskih mapa formatu u kojem ih je model očekivao. U tu svrhu napisan je novi dataloader (naziv za klasu u Pytorchu zaduženu za učitavanje podataka u model) za rad s ortopanogramima. Dodana je funkcionalnost za segmentaciju ortopanograma u 2 klase (pozadina + zubi), 3 klase (gornja zubi, donji zubi, pozadina) te 33 klase (32 zuba + pozadina). Također su bile potrebne manje modifikacije u glavnom dijelu koda, većinom kako bi se podržao rad s novim podacima, te modifikacije dijelova koda zaduženih za testiranje i validaciju. Model je treniran na serveru opremljenom s NVIDIA GeForce RTX 2080 grafičkom karticom, sa 11GB video memorije.

5.3. Eksperimentalni rezultati

Za evaluaciju dobivenih rezultata korištena je mjera **srednjeg presjeka nad unijom** (engl. *mean intersection over union, mIoU*). Ovo je jedna od najčešće korištenih metoda za evaluaciju segmentacijskih modela. Metoda se zasniva na računanju postotka područja nad kojim se generirana oznaka i stvarna oznaka podudaraju, podijeljeno s unijom njihovih ukupnih područja. U slučaju više od jedne klase, IoU se računa za svaku klasu zasebno, te se na kraju računa prosjek dobivenih veličina. Moguće vrijednosti IoU su $[0,1]$, pri čemu bi vrijednost 1 imao model kod kojeg je svaki piksel generirane segmentacijske mape jednak svakom pikselu originalne segmentacijske mape. mIoU je samo prosječan IoU nad svim uzorcima za validaciju. Formalno, IoU definiramo kao

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Slika 5.2: Vizualizacija mIoU-a [26]

Dostupni označeni podaci za treniranje bili su podijeljeni u 3 kategorije: trening (250), validacija (61), testiranje (62). Ako se radilo o nenadziranom (cycleGAN) modelu, tada se u model još i učitavao slučajni podskup od 250 neoznačenih ortopanograma. Trenirani su modeli za segmentaciju 2, 3 i 33 klase. Svaki model je treniran prvo na 400 epoha, s iznimkom modela s 3 klase, koji je treniran i na 200 epoha. Nakon treniranja tog modela na 200 epoha, zaključeno je da se bolji rezultati dobivaju s 400 epoha, pa su ostali modeli trenirani samo na 400 epoha. Prije ulaska u model, sve slike su postavljene na rezoluciju 200x400, zbog memorijskih ograničenja. Rezultati treniranja modela prikazani su u nastavku.

Tablica 5.1: Rezultati modela

Model	Broj epoha	mIoU
Deeplab, 2 klase	400	0.8729981767948044
cycleGAN, 2 klase	400	0.8735012386115315
Deeplab, 3 klase	200	0.8427474738296579
cycleGAN, 3 klase	200	0.8357329101641446
Deeplab, 3 klase	400	0.847020406648659
cycleGAN, 3 klase	400	0.8479922780975828
Deeplab, 33 klase	400	0.7023904701665826
cycleGAN, 33 klase ($\lambda_1 = 1.0$)	400	0.650361416818396
cycleGAN, 33 klase ($\lambda_1 = 1.25$)	400	0.6810801346650556

Iz priložene tablice zaključujemo nekoliko stvari. Prva je da se povećanjem broja klasa mIoU smanjuje, no to nije ništa neočekivano. Vidimo da nadzirani (Deeplab) i polunadzirani (cycleGAN) model pri manjem broju klasa postižu gotovo identične rezultate - prilikom treniranja kroz 400 epoha cycleGAN je marginalno bolji, iako razlika je dovoljno mala da ju možemo zanemariti. Također vidimo da, prilikom treniranja sa 3 klase kroz 200 epoha nadzirani model je bio nešto bolji od polunadziranog. Međutim, kad smo broj epoha povećali na 400, polunadzirani model je uspio dostići, pa čak i preći nadzirani model. Općenito sam primijetio da cycleGAN model sporije konvergira prema minimumu u odnosu na Deeplab model, pogotovo pri početku treniranja, te mogu zaključiti da je cycleGAN bolje trenirati kroz veći broj epoha u odnosu na nadzirane modele.

Rezultati za 33 klase su nešto drugačiji. Vidimo da nadzirani model ima mIoU vrijednost za čak 0.05 bolju od polunadziranog modela, odnosno skoro 8%. Ovakav rezultat nije u skladu s očekivanjima, jer su autori originalnog rada [25], dobili bolje rezultate cycleGAN-a u odnosu na nadzirani model u svim slučajevima. Također, radili smo sa relativno malom količinom podataka (250 slika), te je bilo očekivano da će cycleGAN, uvođenjem dodatnih neoznačenih slika poboljšati treniranje.

Smatram da razlog ovakvoj razlici između nadziranog i polunadziranog modela leži u činjenici da polunadzirani model jednostavno nije (dovoljno) dobro naučio generirati nove slike ortopanograma, što možemo vidjeti iz slika 5.9 te 5.10. Vidimo da je model dobro naučio generirati segmentacijske mape, no da nije bio toliko uspješan prilikom generiranja slika ortopanograma. Slike su mutne i pune šuma, što većinom nije bio slučaj u originalnom radu. Iako te slike jesu u većini slučajeva bile dovoljne da bi model iz njih rekonstruirao segmentacijsku mapu ortopanograma, ta mapa se u većini slučajeva ipak djelomično razlikovala od originalne, što je zasigurno negativno utjecalo na treniranje modela. Smatram da je to dijelom zbog relativno malenog skupa podataka, a dijelom i zbog činjenice da su ulazne slike bile niske rezolucije, što odmaže treniranju modela. Također sam primijetio da je funkcija gubitka za diskriminator D_I bila vrlo niska (<0.01), što ukazuje na to da je D_I vrlo lako zaključivao da se radi o lažnim slikama, što ukazuje na njihovu lošu kvalitetu. Što se tiče funkcija gubitka diskriminatora, pokazivale su slično ponašanje kao i D_I . Konačno, kako ukupna funkcija gubitka u cycleGAN modelu ovisi i o funkcijama gubitka zaduženim za generiranje slika, pretpostavljam da su upravo te funkcije gubitka zaustavljale ukupan model da ide prema nižem minimumu. Smatram da je to glavni razlog zašto je nadzirani model postigao bolje rezultate od nenadziranog.

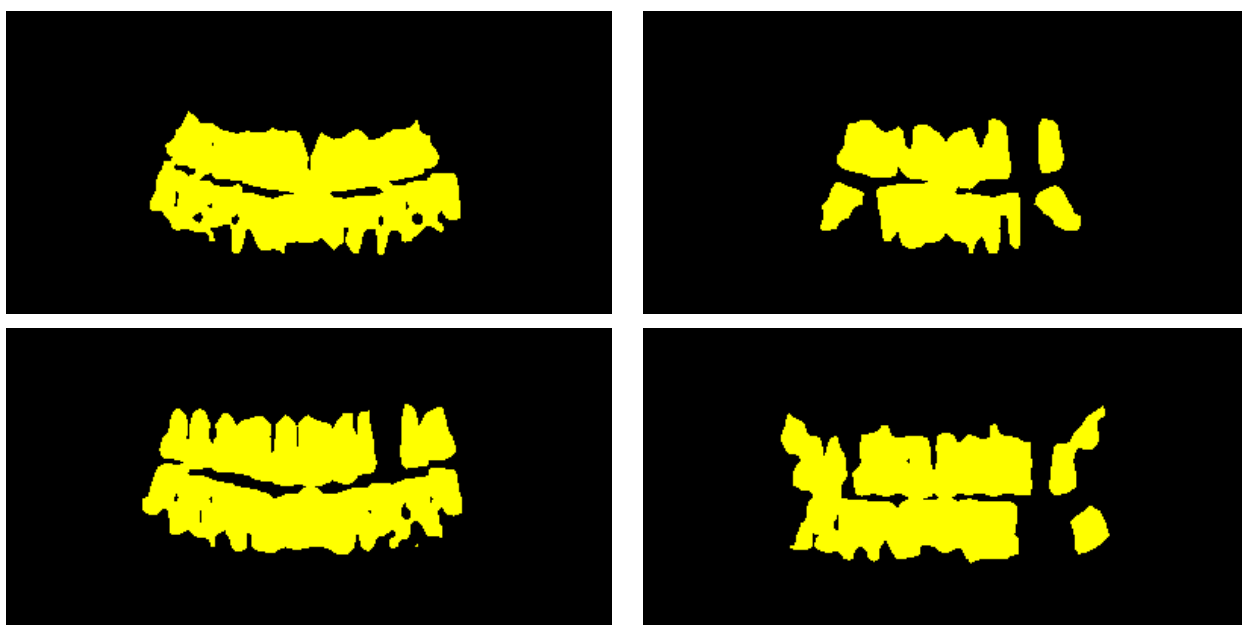
Djelomično kako bi testirao tu teoriju, a djelomično kako bi poboljšao rezultate modela, trenirao sam još jedan model - cycleGAN sa 33 klase, ali sa parametrom λ_1 povećanim sa 1.0 na 1.25. To je parametar koji određuje koliko funkcija gubitka generatora zaduženog za generiranje segmentacijskih mapa utječe na ukupnu funkciju gubitka. Pretpostavio sam da njegovo povećanje može potaknuti model da se u ukupnoj funkciji gubitka više fokusira na poboljšanje generiranja segmentacijskih mapa, te na taj način smanji utjecaj (loše) generiranih ortopanograma. Dobiveni rezultati potvrđuju tu pretpostavku - mIoU tog modela je za 0.03 uspješniji od mIoU-a istog modela sa $\lambda_1 = 1.0$, no i dalje je lošiji od nadziranog modela.

Kao ideju za neka buduća istraživanja, smatram da bi se bilo korisno fokusirati na poboljšanje generiranja neoznačenih slika u modelu, jer se to trenutno čini kao najslabiji dio modela. Korisno bi bilo i mijenjati ostale parametre modela (npr. λ_2 , λ_3 itd.) da vidimo kako utječu na model. Također, kao i uvijek prilikom treniranja dubokih modela, rezultati modela bi sigurno bili bolji kada bi na raspolaganju imali više podataka. Bilo bi zanimljivo i pogledati kako rezolucija ulaznih slika utječe na performanse modela - očito je da bi rezultati modela bili bolji, no pitanje je u kojoj mjeri. Trenutno se pokazalo da je za problem segmentacije ortopanograma bolje koristiti neki klasičan konvolucijski model, poput Deeplaba, nego cycleGAN. Osim što se takav model pokazao bolji za segmentaciju većeg broja klasa, pokazuje gotovo identične performanse kao i cycleGAN kod nižeg broja klasa, a pošto puno brže konvergira, može ga se trenirati kroz manji broj epoha nego cycleGAN. Konačno, prilikom treniranja cycleGAN troši skoro 3x veću količinu memorije nego Deeplab (3GB vs 10.5GB), što ga čini prikladnijim i jednostavnijim za treniranje.

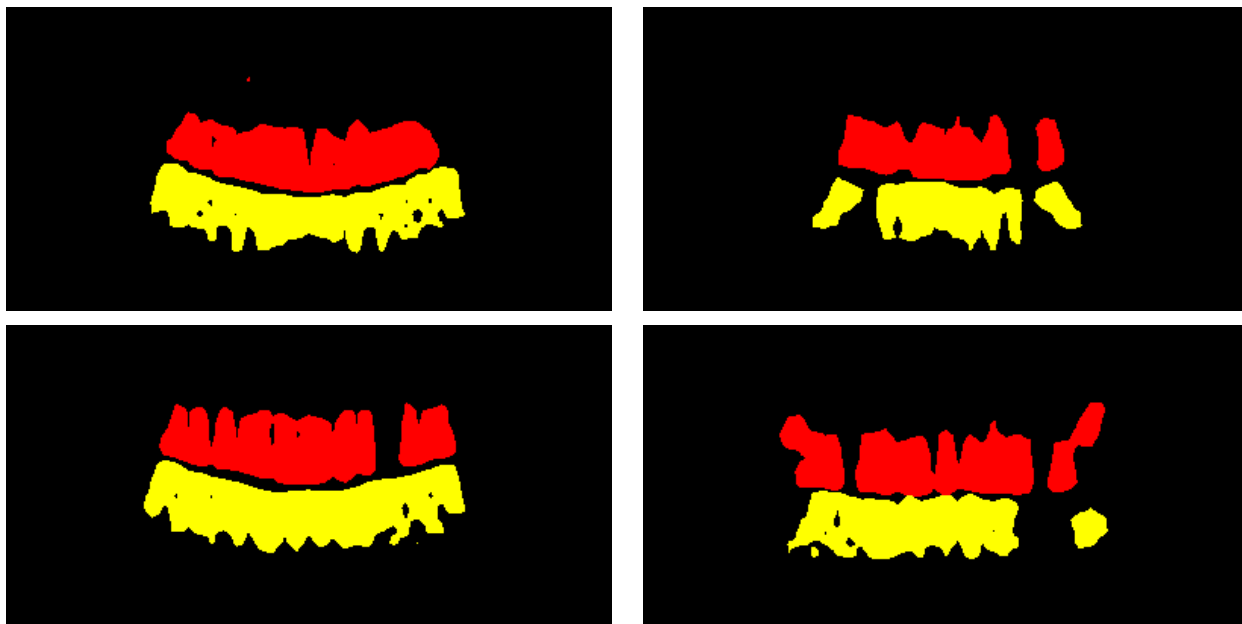
U nastavku prilažem slike koje su generirali odgovarajući modeli. U slučaju 2 klase, žutom bojom su označeni zubi, a crnom pozadina. Ako imamo 3 klase, žuto su donji zubi, a crveno gornji. Slično je u slučaju 33 klase, samo što je svaki donji zub označen svojom nijansom žute, a svaki gornji svojom nijansom crvene.



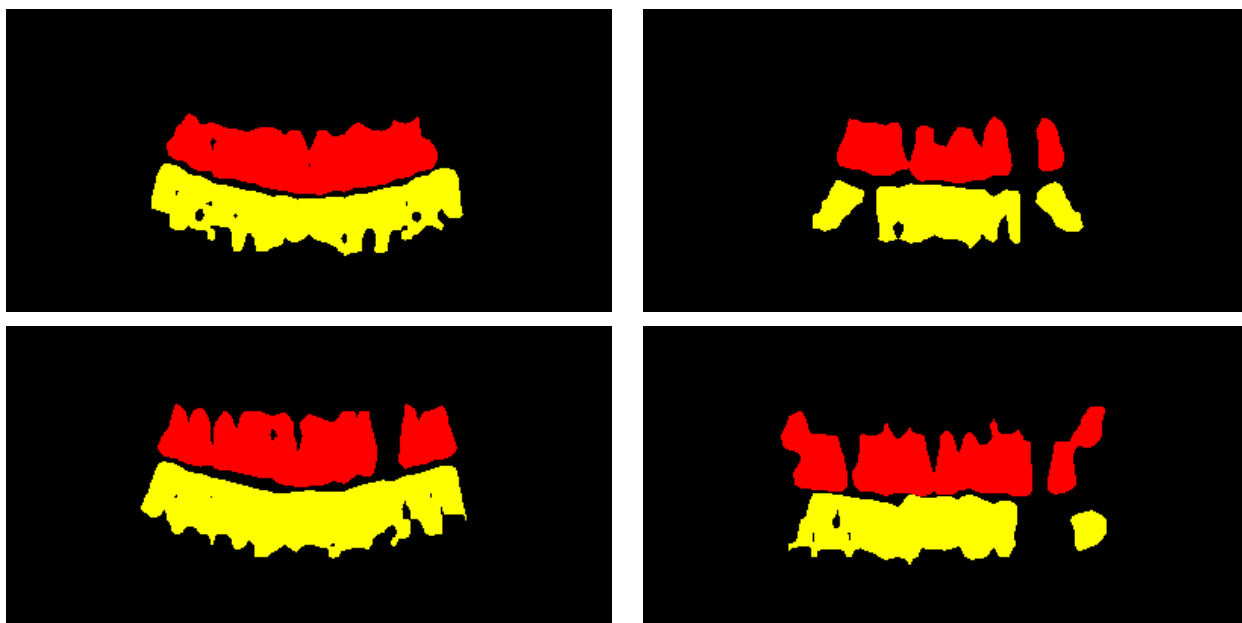
Slika 5.3: Neki od rezultata nadziranog modela za 2 klase



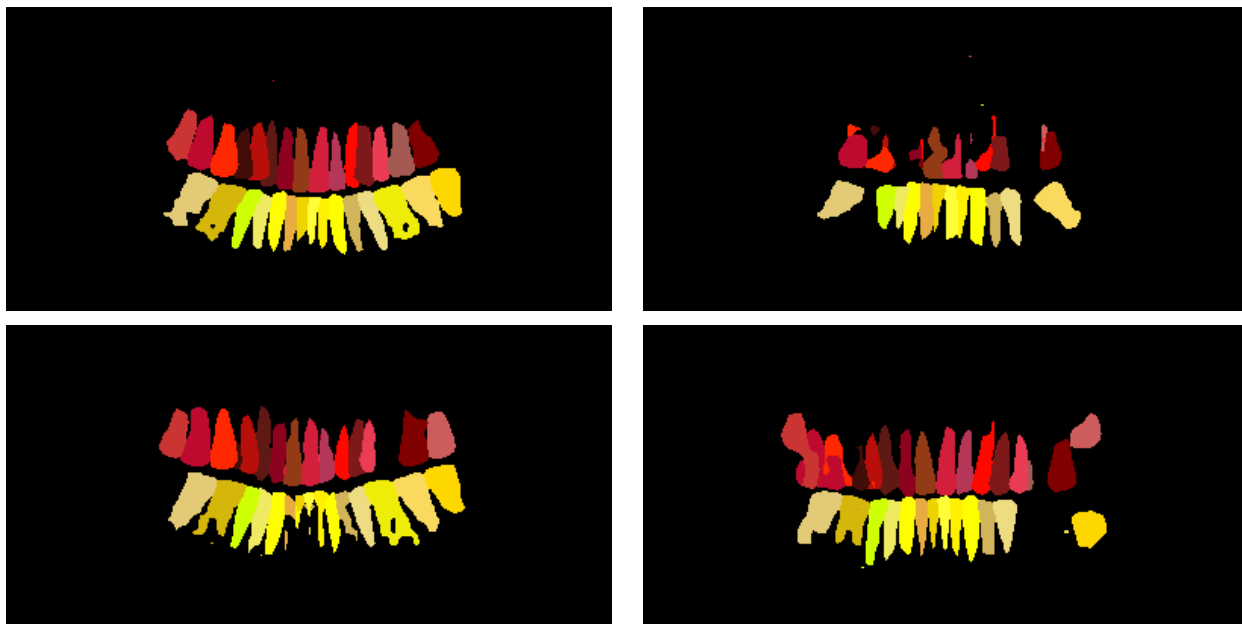
Slika 5.4: Neki od rezultata polunadziranog modela za 2 klase



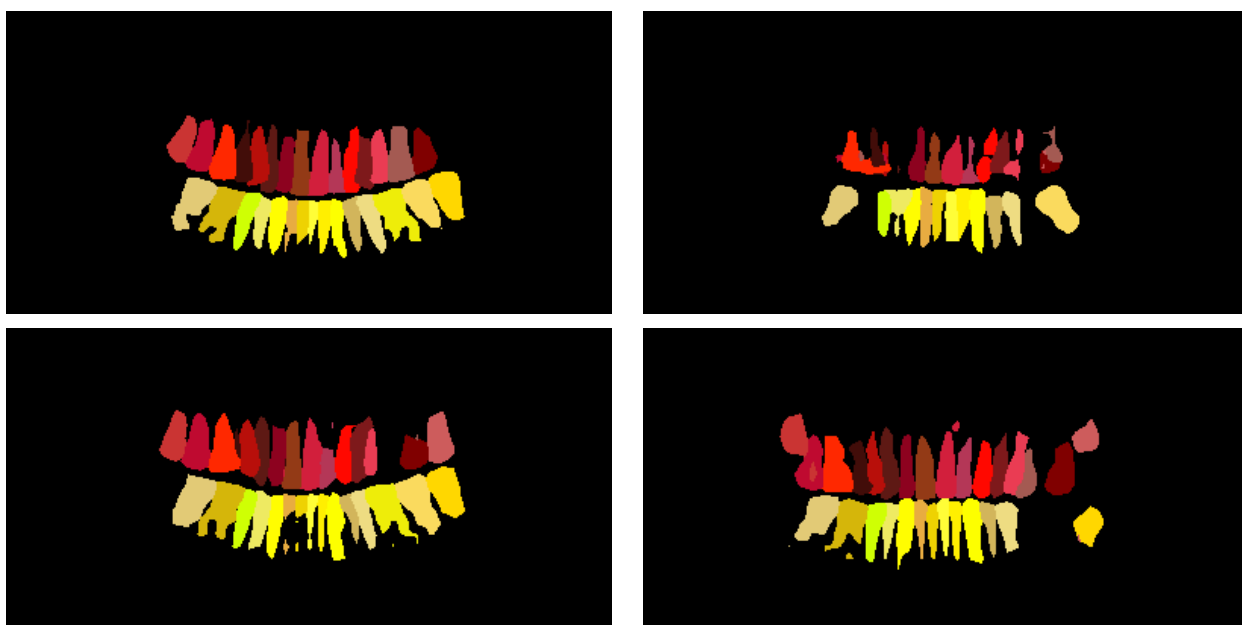
Slika 5.5: Neki od rezultata nadziranog modela za 3 klase



Slika 5.6: Neki od rezultata polunadziranog modela za 3 klase



Slika 5.7: Neki od rezultata nadziranog modela za 33 klase



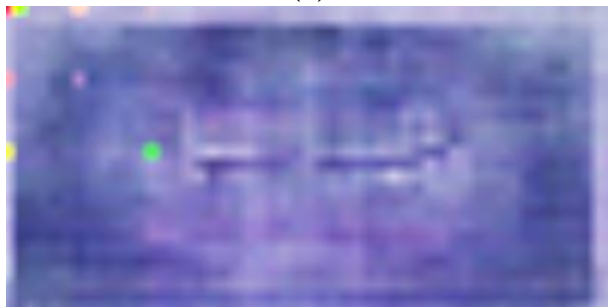
Slika 5.8: Neki od rezultata polunadziranog modela za 33 klase ($\lambda_1 = 1.0$)



(a)



(b)

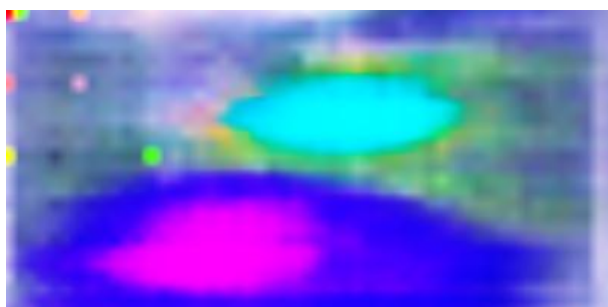
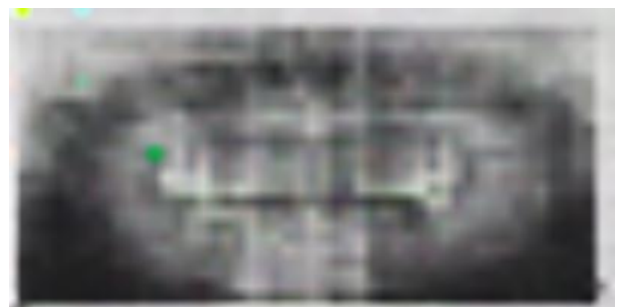


(c)



(d)

Slika 5.9: Prikaz svih generiranih slika tijekom jednog prolaska kroz cycleGAN mrežu, primjer. 1 Slika a) prikazuje prvi prolazak originalne slike kroz G_{IS} . Slika b) prikazuje sliku dobivenu prolaskom originalne segmentacijske mape kroz G_{IS} , dakle $G_{SI}(x)$. c) prikazuje sliku dobivenu prolaskom generirane segmentacijske mape kroz G_{SI} , dakle $G_{SI}(G_{IS}(x))$. d) prikazuje segmentacijsku mapu dobivenu prolaskom b) kroz G_{IS} .



Slika 5.10: Prikaz svih generiranih slika tijekom jednog prolaska kroz cycleGAN mrežu, primjer 2

6. Zaključak

Cilj ovog rada bio je pronaći odgovarajuće rješenje za problem semantičke segmentacije ortopanograma, u slučaju kad na raspolaganju imamo relativno malo označenih, a relativno puno neoznačenih slika. U tu svrhu ukratko su predstavljeni modeli dubokog učenja, s naglaskom na duboke konvolucijske modele, te je dano par primjera takvih arhitektura na konkretnom problemu semantičke segmentacije. Zatim je predstavljen koncept generativnih modela, te su predstavljene njihove prednosti nad klasičnim diskriminativnim modelima. Odabrana je jedna konkretna generatorska arhitektura, cycleGAN, iz razloga što, uz označene, može iskoristiti i neoznačene slike prilikom treniranja. Arhitektura je trenirana da raspozna 2, 3 i 33 klase u slici ortopanograma. Za treniranje je na raspolaganju bilo 250 označenih slika za treniranje, 62 slike za validaciju, 61 za testiranje i slučajan podskup od 250 slika od ukupno 3950 neoznačenih slika. Rezultati modela su pokazali da, iako je cycleGAN bio uspješan u problemu segmentacije, nije pokazao značajno bolje rezultate u odnosu na najnaprednije diskriminativne modele (Deeplab), te je u slučaju 33 klase čak pokazao lošije rezultate. Konačno, dano je par prijedloga za moguća daljna istraživanja.

LITERATURA

- [1] Mike Tamir. Comparison of object detection approaches, 2019. URL https://miro.medium.com/max/2400/1*z89KwWbF59XXrsXXQCECPA.jpeg. [Online; zadnji pristup 28.5.2021].
- [2] izv. prof. dr.sc Marko Čupić prof. dr. sc. Bojana Dalbelo Bašić, izv. prof. dr. sc. Jan Šnajder. Predavanja iz predmeta *Uvod u umjetnu inteligenciju*, FER, 2019. URL <https://www.fer.unizg.hr/predmet/uuui>. [Online; zadnji pristup 03.06.2021].
- [3] Dr. Alan Woodruff. članak *What is a neuron*, *The University of Queensland, Brain Institute*. URL <https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron>. [Online; zadnji pristup 03.06.2021].
- [4] izv. prof. dr.sc Marko Čupić prof. dr. sc. Bojana Dalbelo Bašić, izv. prof. dr. sc. Jan Šnajder. *Umjetne neuronske mreže*. FER, 2008. https://www.fer.unizg.hr/_download/repository/UmjetneNeuronskeMreze.pdf.
- [5] izv. prof. dr. sc Marko Čupić. *Umjetne neuronske mreže - skripta za predmet "Umjetna inteligencija"*. FER, 2016. <http://java.zemris.fer.hr/nastava/ui/ann/ann-20180604.pdf>.
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [7] S.R. Shinde. *Improving Artificial Neural Network with Regularization and Optimization*. Towards A.I, 2020. <https://towardsai.net/p/machine-learning/improving-artificial-neural-network-with-regularization-and-optimization>.
- [8] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. Insight Centre for Data Analytics, NUI Galway, 2017. <https://arxiv.org/pdf/1609.04747.pdf>.

- [9] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. 2021. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>.
- [10] Eda Kavlakoglu. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?* IBM, 2020. <https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks>.
- [11] Pratik Choudhari. *Understanding “convolution” operations in CNN*. Medium, 2020. <https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4>.
- [12] Ann. H. Reynolds. *Convolutional Neural Networks (CNNs)*. 2019. <https://anhreynolds.com/blogs/cnn.html>.
- [13] How relu and dropout layers work in convolutional neural networks, 2020. <https://www.baeldung.com/cs/ml-relu-dropout-layers>.
- [14] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way, 2018. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b116>.
- [15] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. *Deep Residual Learning for Image Recognition*. Microsoft Research, 2015. <https://arxiv.org/pdf/1512.03385.pdf>.
- [16] Anand Saha. *Decoding the ResNet architecture*. 2017. <http://teleported.in/posts/decoding-resnet-architecture/>.
- [17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Computer Science Department and BIOS Centre for Biological Signalling Studies, University of Freiburg, Germany, 2015. <https://arxiv.org/pdf/1505.04597.pdf>.
- [18] Beeren Sahu. The evolution of deeplab for semantic segmentation, 2019. <https://towardsdatascience.com/the-evolution-of-deeplab-for-semantic-segmentation-95082b025571>.
- [19] Background: What is a generative model? <https://developers.google.com/machine-learning/gan/generative>.

- [20] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. *Generative Adversarial Nets*. Departement d’informatique et de recherche operationnelle,Universite de Montreal, 2014. <https://papers.nips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>.
- [21] Simon Osindero Mehdi Mirza. *Conditional Generative Adversarial Nets*. Departement d’informatique et de recherche operationnelle,Universite de Montreal, 2014. <https://arxiv.org/pdf/1411.1784.pdf>.
- [22] Nouman Abbasi. What is a conditional gan (cgan)? <https://www.educative.io/edpresso/what-is-a-conditional-gan-cgan>.
- [23] Simon Osindero Mehdi Mirza. *Alec Radford,Luke Metz,Soumith Chintala*. 2016. <https://arxiv.org/pdf/1511.06434.pdf>.
- [24] Phillip Isola Alexei A. Efros Jun-Yan Zhu, Taesung Park. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. Berkeley AI Research (BAIR) laboratory, UC Berkeley, 2017. <https://arxiv.org/pdf/1511.06434.pdf>.
- [25] Jose Dolz Christian Desrosiers Arnab Kumar Mondal, Aniket Agarwal. *Revisiting CycleGAN for semi-supervised segmentation*. 2019. <https://arxiv.org/pdf/1908.11569.pdf>.
- [26] Metrics to evaluate your semantic segmentation model. [<https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2>].

Sažetak

Cilj ovog rada bio je pronaći i testirati odgovarajuću arhitekturu za problem semantičke segmentacije kada je prisutan manji broj označenih i veći broj neoznačenih primjera. Predstavljena je cycleGAN arhitektura, te je testirana na konkretnom problemu semantičke segmentacije ortopanograma. Arhitektura je izabrana iz razloga što može iskoristiti i neoznačene slike kao pomoć pri treniranju. Dobiveni rezultati su pokazali da cycleGAN arhitektura uspješno rješava navedeni problem, no da ipak zaostaje u usporedbi s najnaprednijim diskriminativnim modelima.

Ključne riječi: duboko učenje, konvolucijske neuronske mreže, semantička segmentacija, GAN, cycleGAN, ortopanogrami, medicinske snimke

Segmentation of Stomatologic Panoramic X-ray Images

Abstract

The goal of this work was to find and test an adequate deep learning architecture applicable to the problem of semantic segmentation, in the case when we at our disposal have a limited number of labeled images, but a relatively high amount of unlabeled images. cycleGAN, a type of generative model was proposed as solution to the starting problem. It was chosen due to the fact that it can use unlabeled images as well as labeled to improve training performance. It was tested on the dataset of stomatologic X-ray images (orthopantomograms), and it was shown that while it successfully solves the given problem, it still falls behind most advanced discriminative models.

Keywords: deep learning, convolutional neural networks, semantic segmentation, GAN, cycleGAN, orthopantomograms, medical imaging