Generativne neuronske mreže za sintezu robotskih putanja

Janković, Tin

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Tin Janković

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Izv. prof. dr. sc. Petar Ćurković, dipl. ing.

Student:

Tin Janković

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru izv. prof. dr. sc. Petru Ćurkoviću na savjetima, pristupačnosti i pomoći tijekom pisanja ovog rada.

Također se želim zahvaliti svojoj obitelji, djevojci i prijateljima na podršci i razumijevanju tijekom studiranja.

Tin Janković

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske radove studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-04/22-6/1	
Ur. broj: 15-1703-22-	

DIPLOMSKI ZADATAK

Student: **TIN JANKOVIĆ** Mat. br.: 1191237962

Naslov rada na hrvatskom jeziku: **Generativne neuronske mreže za sintezu robotskih putanja**

Naslov rada na engleskom jeziku: **Generative neural networks for synthesis of robot paths**

Opis zadatka:

Generativne neuronske mreže koriste se za tvorbu rješenja principom natjecanja dviju neuronskih mreža – generativne i klasifikacijske. Princip rada ovakvih mreža razlikuje se od klasičnih, kod kojih se mreža koristi samo za prepoznavanje odnosno klasifikaciju skupa ulaznih podataka.

Primjenom generativnih mreža moguće je stvoriti vrlo realistične modele ljudi, životinja, ali i predmeta koji ne postoje, već su u potpunosti proizvod procesa rada mreže. Primjenom ovakvih mreža moguće je, uz zadana ograničenja u okolini, stvoriti i rješenja u vidu putanja koje robot može slijediti bez opasnosti kolizije s preprekom.

U radu je potrebno napraviti sljedeće:

- upoznati se s teorijom i radom generativnih neuronskih mreža
- identificirati mogućnosti primjene ovih mreža u području umjetne inteligencije s naglaskom na primjenu za rješavanje problema sinteze robotskih putanja
- u programskom jeziku Python implementirati generativnu mrežu za pronalazak putanje u 2D prostoru koji sadrži prepreke
- analizirati potrebu za računalnim resursima (procesorskim vremenom) u ovisnosti o finoći prostorne diskretizacije

U radu je potrebno navesti literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
5. svibnja 2022.	7. srpnja 2022.	18. srpnja do 22. srpnja 2022.

Zadatak zadao:
izv. prof. dr. sc. Petar Ćurković

Predsjednica Povjerenstva:
prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
a_n	-	Elemnti matrice
D( )	-	Funkcija diskriminatora
G( )	-	Funkcija generatora
n_A	-	Dimenzija matrice A
n_B	-	Dimenzija matrice B
n_c	-	Broj korištenih kernela u konvolucijskom procesu
n_K	-	Dimenzija matrice filtera
p_{data}	-	Skup stvarnih podataka
p_z	-	Skup ulaznih podataka šuma
s	-	Korak u procesu konvolucija
V	-	Funkcija gubitka
x	-	Podatak iz seta za učenje diskriminatora
x1	-	Početna točka na koordinati x
x2	-	Krajnja točka na koordinati x
y1	-	Početna točka na koordinati y
y2	-	Krajnja točka na koordinati x

POPIS KRATICA

Kratica	Opis
API	Aplikacijsko programsko sučelje
AWS	Amazon Web Services
CGAN	Uvjetne generativne kontradiktorne mreže engl. Conditional generative adversarial networks
CPU	Central Processing Unit
GAN	Generativne kontradiktorne mreže engl. Generative adversarial networks
GPU	Graphics Processor Unit
TPU	Tensor processing unit

SAŽETAK

U ovom radu proučavat će se mogućnost računalnog generiranja sadržaja pomoću Generativnih kontradiktornih neuronskih mreža (engl. GAN) s ciljem poboljšanja navigiranja robota. Mreža je vrlo složene strukture jer sadrži dvije manje neuronske mreže koje se međusobno sukobljavaju. Prva manja mreža je generator koji želi kreirati takav sadržaj koji druga mreža diskriminator neće moći razlikovati od stvarnog seta podataka.

U prvom dijelu rada objasnit će se osnove rada GAN mreža i objasniti njihova struktura. Zatim će se opisati korišteni alati za razvoj mreže i upoznati sa ostalim potencijalnim primjenama GAN mreža koje mogu olakšati svakodnevicu čovjeka. Predstavit će se jedno od mogućih rješenja GAN mreže za generiranje trajektorija i navesti dobiveni rezultati te će se na kraju primijeniti generirane trajektorije za vođenje simulacijskog modela robota.

Ključne riječi: Generativne kontradiktorne mreže, GAN, CGAN, strojno učenje, neuronske mreže.

SUMMARY

In this paper, the possibility of computer-generated content generation using Generative adversarial neural networks (GAN) is investigated with the aim of improving robot navigation. The network has a very complex structure as it contains two smaller neural networks that collide with each other. The first smaller network is a generator that wants to produce such a content that the second network discriminator is not able to distinguish from the actual data set.

The first part of the paper will explain the basics of how GAN networks work and explains their structure. Then, the tools used for network development are described and other possible applications of GAN networks that can facilitate everyday life will be introduced. One of the possible solutions of the GAN network for trajectory generation is presented and the obtained results will be stated, and finally the generated trajectories are used to control the robot simulation model.

Key words: Generative adversarial networks, GAN, CGAN, machine learning, neural networks.

1. UVOD

Napredak računalne moći omogućilo je korištenje brojnih algoritama iz područja umjetne inteligencije. Svakim danom raste broj tih primjena koje bitno olakšavaju život značajnom broju ljudi. Iako su računala uvijek intenzivno olakšavala ljudske zadatke, nikad nisu imala mogućnost kreiranja vlastitih podataka kao što to može čovjek. Veliki uspjesi su postignuti u posljednjem desetljeću na tom području, što je otvorilo vrata k brojnim novim potencijalnim mogućnostima. Generativne kontradiktorne neuronske mreže (engl. GAN) puno su doprinijele toj promjeni i često su predmet istraživanja mnogih znanstvenika. U području umjetne inteligencije koriste se razni pristupi za pronalaženje putanja robota: evolucijski algoritmi, koevolucijski algoritmi, ART neureonske mreže, a u ovom radu pokazat će se pristup generiranja robotskih trajektorija pomoću GAN mreže. [1], [2], [3], [4], [5]

1.1. Povijest GAN mreža

Algoritmi strojnog učenja već desetljećima pokazuju odlične rezultate u prepoznavanja uzoraka i koristeći tu sposobnost obavljaju zadatke klasifikacije i regresije. U izvršavanju takvih zadataka daleko su nadmašili ljudsku sposobnost. Algoritmi su uspješno pobjeđivali najbolje šahiste svijeta, dijagnosticirali bolesti, estimirali vrijednosti nekretnina u budućnosti, ali nisu pokazivali dobre rezultate u kreiranju novog sadržaja. Tvorac GAN mreža Ian Goodfellow 2014. svojim izumom značajno unaprjeđuje mogućnost računala za samostalno kreiranje sadržaja. GAN mreže nikako nisu prvi računalni program koji se koristio za samostalno računalno stvaranje sadržaja, ali rezultati GAN mreža pokazali su značajno poboljšanje rezultata u odnosu na rezultate prijašnjih najboljih programa. GAN mreže kao i sve neuronske mreže svrstavaju se u algoritme strojnog učenja. To su duboke neuronske mreže koje sadrže dvije manje neuronske mreže koje se međusobno sukobljavaju. Prva neuronska mreža naziva se generator, a druga diskriminator. Model generatora kao što naziv sugerira, generira odnosno stvara slike. Model diskriminatora prima generirane slike od strane generatora, a također i slike iz stvarnog skupa podataka. Mreža diskriminatora mora odlučiti koja je slika stvarna, odnosno pripada stvarnom skupu podataka, a koja je slika generirana generatorom. Popularna usporedba sa stvarnim životom je da se generator uspoređuje s krivotvoriteljem, a diskriminator s detektivnom. Cilj generatora (krivotvoritelja) je napraviti takvu sliku da diskriminator (detektiv) ne može razlučiti stvarnu sliku od generirane. Mreža generatora pokušava

unaprijediti generirane slike sve dok diskriminator ne može razlučiti koja je slika iz stvarnog skupa, a koja generirana od strane mreže.

Od samih početaka GAN mreža popularna kategorija testiranja mogućnosti i stvaranja slika pomoću GAN mreža bila je generiranje slika čovjeka. Iako su smatrane velikim uspjehom, slike generirane 2014. godine i dalje ljudskom oku nisu izgledale kao stvarne reprezentacije čovjeka. Zbog intenzivnih istraživanja na ovom području, popraćenih napretkom tehnologije omogućeno je kreiranje slika koje se ne mogu razlikovati od stvarnih fotografija ljudi. Odličan primjer prikazan je na slici 1. Na slici su prikazani ljudi koji u stvarnosti ne postoje nego ih je generirala mreža. Slika 2. pokazuje napredak generiranih slika od 2014. do 2017. godine. [6]

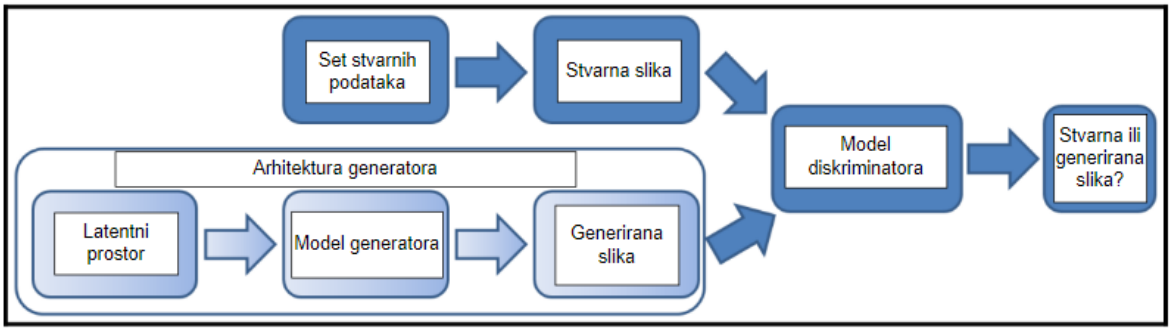
Slika 1. Ljudi kreirani pomoću GAN mreže [6]

Slika 2. Napredak u generiranju slika u vremenskom periodu od 2014. do 2017. godine [6]

1.2. Arhitektura GAN mreže

Struktura GAN mreže poprilično je složena s obzirom na to da se sastoji od dviju neuronskih mreža. U prethodnom poglavlju spomenuto je da se međusobno natječu, odnosno sukobljavaju, odakle i dolazi naziv kontradiktorne mreže. Model GAN mreže sastoji se od modela generatora i diskriminatora, a za njihovo učenje potreban je set podataka za treniranje mreže. Osim toga, modelu generatora je potreban izvor šuma iz kojeg bi mogao kreirati sadržaj. Generator na svojem ulazu uzima jedan vektor fiksne duljine i generira uzorak (najčešće sliku) u određenoj domeni. Vektor koji služi kao izvor šuma sastoji se od elemenata odabranih nasumičnim odabirom iz Gaussove normalne distribucije. Nakon treninga mreže, točke u vektoru će odgovarati točkama u domeni uzoraka, te će se na taj način stvoriti komprimirani prikaz distribucije podataka. Spomenuti višedimenzionalan vektor s elementima se naziva latentni prostor. Dakle, treningom mreže generator daje značenje točkama iz latentnog prostora, tako da se nove točke iz latentnog prostora mogu dati kao ulaz generatoru i na taj način generirati novi primjerci unutar domene.

Diskriminator na svom ulazu uzima uzorak koji može biti generiran ili stvaran iz domene, a zatim predviđa je li uzorak napravljen od strane generatora ili nije. Model diskriminatora je normalan klasifikacijski model, a zajedno s modelom generatora čini strukturu GAN mreže prikazanu na slici 3.



Slika 3. Arhitektura GAN modela [7]

1.3. Funkcije gubitka

Trening mreže podrazumijeva podešavanje vrijednosti težina mreže tako da se optimizira funkcija gubitka za određeni problem. Zbog tog je razloga funkcija gubitka ključna da mreža konvergira k rješenju i daje dobre rezultate. U članku Iana Goodfellowa navedena je sljedeća funkcija gubitka.

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

, gdje je x podatak iz seta za učenje, p_{data} je skup stvarnih slika, $G(\,)$ diferencijalna funkcija modela generatora, $p_z(z)$ skup ulaznih podataka šuma.

D(x) predstavlja vjerojatnost, koju daje model diskriminatora, da je podatak x došao iz stvarnog seta podataka. S obzirom na to da je cilj diskriminatora ispravno klasificirati stvarne slike u odnosnu generirane, cilj je naučiti diskriminator pridjeljivati veće vjerojatnosti stvarnim slikama od generiranih. To se matematički može opisati maksimiziranjem logaritma vjerojatnosti predviđanja stvarnih slika i logaritma suprotne vjerojatnosti predviđanja generiranih slika. Također, simultano se želi naučiti model generatora minimizirati vjerojatnost da je model diskriminatora u pravu. Matematički se to može opisati maksimiziranjem izraza $\log(1 - D(G(z)))$.

Može se primijetiti da model diskriminatora za funkciju gubitka zapravo koristi standardnu unakrsnu entropiju koja se često koristi u klasifikacijskim problemima. Istraživanja su pokazala da ovakva GAN funkcija gubitka može uzrokovati zasićenje generatora. To znači da diskriminator puno bolje uči od generatora, a time trening mreže postaje beskoristan. Varijacija nezasićene funkcije gubitka, s ciljem da se izbjegne problem zasićenja GAN-a, je modifikacija generatora. Promjena je ta što se zahtjeva od generatora da maksimizira logaritamsku vjerojatnost diskriminatora $\max \log(D(G(z)))$ za prepoznavanje generiranih slika umjesto da minimizira $\log(1 - D(G(z)))$ funkciju. [8]

2. PODRUČJE PRIMJENE GAN MREŽE

Kreiranje GAN mreža od samih početaka bilo je popraćeno pohvalama od strane stručnjaka u području umjetne inteligencije. Većina stručnjaka se slaže da je njihov izum otvorio vrata k mnogim budućim istraživanjima i omogućio brojne nove primjene. Tako je Yann LeCun bivši direktor AI sektora u Facebooku i jedan od najvećih stručnjaka u području umjetne inteligencije rekao kako je izum GAN mreža i njihove varijacije najbolja ideja u području umjetne inteligencije u posljednjih 20 godina. [9]

Područje istraživanja GAN mreža danas je vrlo široko. Trenutna najveća primjena mreže je u generiranju slika zbog njihove obrade. GAN mreže se koriste za zadatke kao što su prebacivanje slika u drugu domenu (engl. image to image translation), prebacivanje slikovitog tekstualnog opisa u sliku (engl. text to image translation), editiranje slika, miješanje slika i druge. Iako se pretežito koriste za generiranje slika, njihovo korištenje nije ograničeno samo na slike, već se može generirati tekst, zvuk i ostali sadržaji koji se mogu pružiti mreži za njeno učenje. U inženjerskoj praksi GAN mreže testiraju se na više primjena. Jedna od njih je generiranje trajektorija robota. Iako je za ljude planiranje putanje kretanja vrlo trivijalan zadatak, za robote je to složen problem.

Planiranje putanje robota svodi se na dva problema. Prvi se odnosi na postojanje trajektorije koja izbjegava sve prepreke, a drugi se svodi na računanje takvog puta. Algoritam planiranja trajektorije mora biti efikasan kako bi robot bio siguran za okolinu. Pri kretanju robot mora cijelo vrijeme biti svjestan svoje pozicije u odnosu na njemu postavljeni cilj. Također, optimalna trajektorija trebala bi omogućiti da robot svoj zadatak obavi u najkraće moguće vrijeme pritom koristeći najmanje energije za obavljanje istog. Postoji mnogo metoda kojima se računa optimalna trajektorija robota, a većina njih koristi iterativni postupak za poboljšavanje inicijalne trajektorije optimizirajući funkciju gubitka. Mnogi algoritmi za planiranje trajektorija ne koriste educirani pokušaj inicijalne trajektorije, već je inicijalna trajektorija pravac između početne pozicije i cilja. Iako su trajektorije generirane takvim algoritmima vrlo precizne, sporo konvergiraju k rješenju pa se zbog tog razloga manje koriste. Jedno od rješenja je korištenje robota s bazom pripremljenih problema i njihovih rješenja koja bi se davale algoritmu kao inicijalne trajektorije. U tom je slučaju potrebno robotu omogućiti pristup bazi podataka za vrijeme njegovog rada. Dodatni problem je ograničenost trajektorija koje se mogu staviti u bazu, jer ako je baza prevelika bit će potrebno dulje vrijeme za pronalazak najbolje trajektorije. Primjena GAN mreže u rješavanju ovog problema je naučiti mrežu s trajektorijama koje već

izbjegavaju prepreke kako bi generirane trajektorije i same izbjegavale prepreke i vodile do određenog cilja. Mrežu se može učiti s velikim brojem trajektorija offline odnosno prije nego bi se robot trebao koristiti. S obzirom na to da bi takve trajektorije već izbjegavale prepreke one bi mogle poslužiti kao bolje inicijalne trajektorije optimizirajućem algoritmu. Na taj bi se način vrijeme računanja algoritma za pronalazak optimalne trajektorije skratilo, a veličina podataka potrebnih za računanje znatno smanjila. [10]

U radu [10] autori su kreirali 60 000 slučajeva prepreka koristeći program „The Open Motion Planning Library“. To je softverski alat u kojem se može simulirati planiranje robota raznim algoritmima i tako kreirati trajektorije koje izbjegavaju prepreke. Simulaciju robota proveli su u virtualnom 2D koordinatnom sustavu, a pred njega su postavljene četiri prepreke reprezentirane kružnicama fiksnog radijusa. S obzirom na to da je robot u svakom slučaju kretao iz iste početne točke u istu završnu točku, slučajevi su se razlikovali samo po rasporedu prepreka. Kreirane trajektorije koje izbjegavaju prepreke poslužile su za treniranje GAN mreže, što je rezultiralo generiranjem trajektorija GAN mreže koje također izbjegavaju prepreke. Ono čime autori nisu bili zadovoljni je to što je GAN mreža kreirala svoj raspored prepreka i svoju trajektoriju. Time je mreža kreirala svoj problem i rješenje te nije postojala mogućnost da se kreira trajektorija za specificirani problem. Kako bi riješili taj problem, odlučili su kreirati CGAN mrežu. Uvjetna generativna kontradiktorna mreža (CGAN) je podvrsta GAN mreža koja generira slike s obzirom na zadani uvjet. Takvoj mreži može se zadati raspored prepreka tj. klasu za koju se generira trajektorija. Pokazalo se da je CGAN mreža kreirala puno bolje inicijalne trajektorije od linearne inicijalizacije za optimizacijski algoritam. [10]

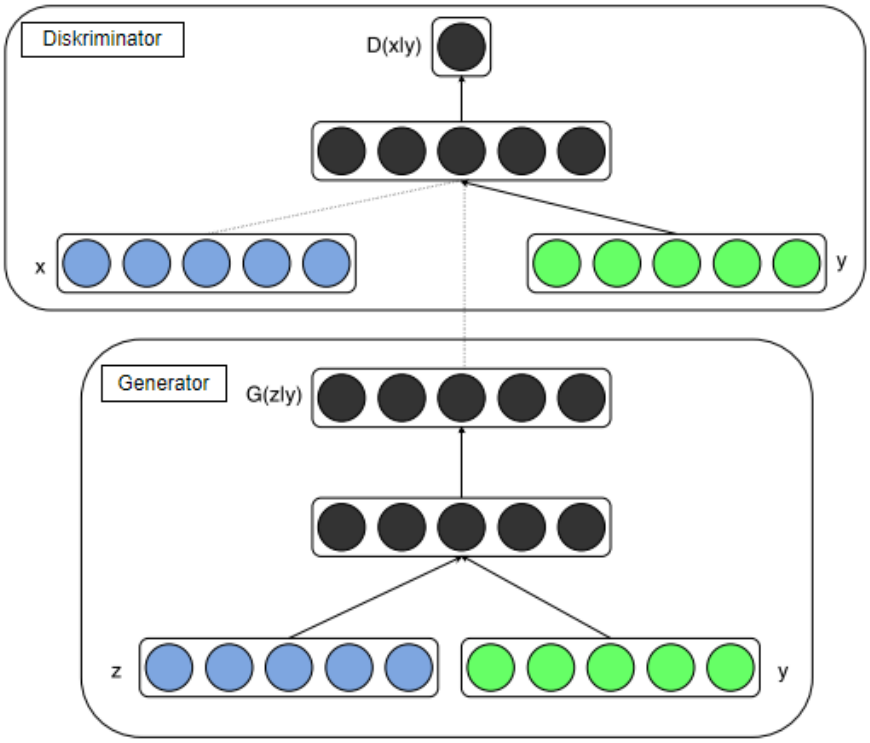
Danas živimo u svijetu gdje većina populacije koristi pametne uređaje. Uz dopuštenje njihovih korisnika moguće je sakupiti podatke o njihovim kretnjama preko njihovih mobilnih uređaja. Drugim riječima moguće je sakupiti mnogo trajektorija koje se mogu koristiti za učenje neuronskih mreža. Primjena GAN mreža nije nužno vezana na pronalazak trajektorija robota. Svojom primjenom mogu olakšati i svakodnevni život ljudi. Autori rada [11] predložili su učenje GAN mreže pomoću podataka sakupljenih iz gomile ljudi (engl. crowd-sourcing) s ciljem olakšavanja mobilnosti korisnika. Generirane trajektorije pružile bi se kao multimodalne informacije osobama s invaliditetom, naročito s oslabljenim vidom i sluhom i time olakšale njihovo kretanje. Također, moguće je u izvanrednim situacijama i evakuacijama generirati najbolje i najsigurnije rute za pojedine grupe korisnika.

Još jedna vrlo zanimljiva moguća primjena GAN mreža je umjetno kreiranje podataka. U današnjici pojedine podatke moguće je jednostavno i jeftino prikupiti, međutim za prikupljanje nekih podataka potrebno je značajno vrijeme i financije. Kvalitetni podaci su vrlo traženi, a naročito za primjene strojnog učenja. Za primjer se može uzeti mobilni robot koji koristi neuronsku mrežu za slijeđenje trajektorije. Ako njegova neuronska mreža za kvalitetno učenje zahtjeva veliki broj trajektorija, u više tisuća primjeraka, možda je bolji i jeftiniji pristup koristiti GAN mrežu. Mreža bi se mogla učiti na malobrojnijem setu podataka, a zatim služiti za generiranje mnogobrojnih trajektorija. Tako bi se trajektorije koje su zadovoljile generiranu kvalitetu mogle pružiti kao set podataka za učenje drugoj mreži.

2.1. Razlika u arhitekturi između GAN i CGAN mreža

Nedostatak GAN mreža je taj što ne mogu ciljano generirati slike nekog tipa. Iako postoji korelacija između točaka u latentnom prostoru i generiranih slika vrlo ju teško raspoznati. Zbog tog razloga GAN mreža mora se modificirati i omogućiti joj dodatne informacije o podacima. Da bi se GAN mreža promijenila u CGAN, generator i diskriminator trebaju na ulazima dobiti uvjet klase koja se generira odnosno provjerava. CGAN mreža prvi puta je opisana u radu „Conditional Generative Adversarial Nets“ autora M. Mirza i S. Osindero [12]. Oni su opisali učenje CGAN mreže spajajući dodatne informacije y (oznake klase) u model generatora G( ) i model diskriminator D( ). To su učinili tako da su ulazne podatke u generator z spojili s oznakama klasa y. Njihov spoj tvorio je skrivenu reprezentaciju koja je pružena modelu generatora. Ulazni podaci u diskriminator x koji se sastoje od stvarnih slika i od generiranih su pruženi modelu diskriminatoru također u spoju s podacima y. Opisana struktura CGAN mreže prikazana je na slici 4. radi lakšeg shvaćanja modela.

Kreiranje CGAN mreže omogućuje korisniku kreiranje slika određene klase. Primjerice mreža može učiti fotografije ljudi prema klasama boje kose. Mreži se pruži klasa boje kose koja se želi generirati i ona će generirati slike ljudi samo te određene boje kose.



Slika 4. Arhitektura CGAN mreže [12]

3. ALATI ZA STVARANJE NEURONSKIH MREŽA

U ovom poglavlju bit će opisani programski alati koji su se koristili za stvaranje CGAN mreže i razlike između procesora koji se koriste u strojnom učenju za provedbu računskih operacija.

3.1. TensorFlow

TensorFlow je besplatna platforma i biblioteka stvorena za strojno učenje koja koristi programski jezik Python. Stvorio ju je Google, a glavni cilj platforme je omogućiti korisnicima jednostavno korištenje matematičkih izraza nad tenzorima. Prednosti TensorFlow platforme su:

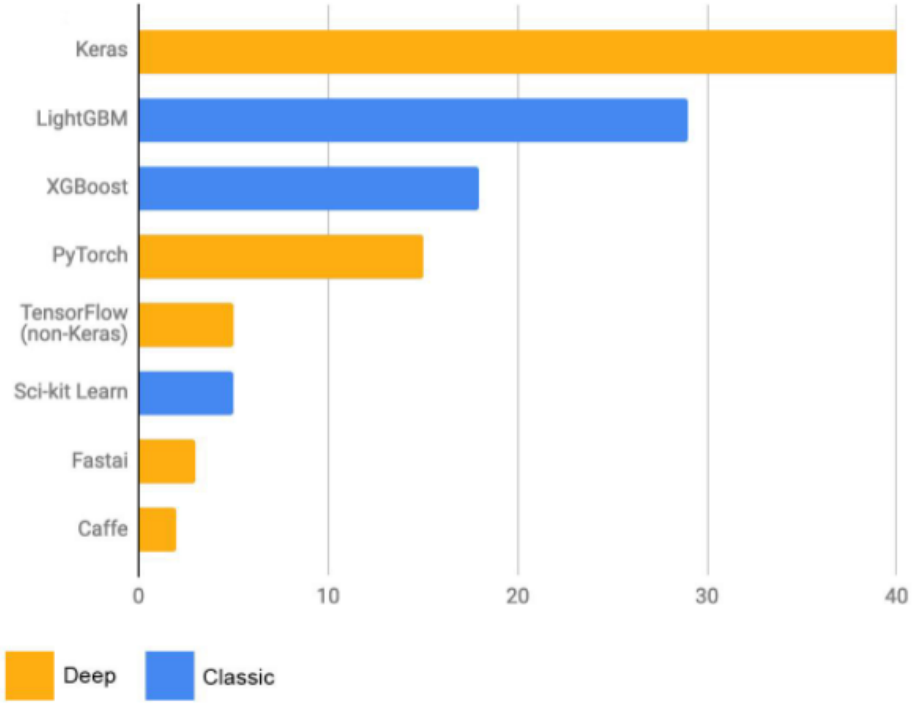
- Automatsko računanje gradijenta svakog diferencijalnog izraza, što je vrlo korisno za strojno učenje
- Radi na CPU, a također GPU i TPU paralelnim procesorima
- Programi napisani pomoću TensorFlowa kompatibilni su sa C++ i JavaScript programskim jezicima što olakšava upotrebu u realnim primjenama

S obzirom da je platforma besplatna ona ima sjajnu zajednicu korisnika. Zbog toga je jednostavno naći rješenja na probleme s kojima su se već susreli korisnici. Uspješnost platforme dokazuje podatak da ju koriste najveće svjetske tvrtke (Google, Coca Cola, Twitter, Intel i druge.) a koristi se i u istraživačke svrhe. [13]

3.2. Keras

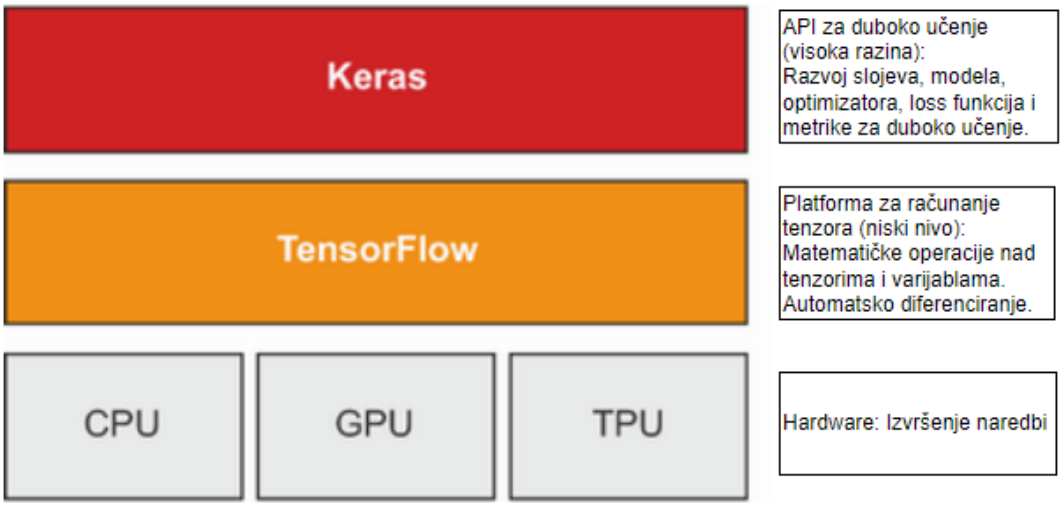
Složenost dubokih neuronskih mreža može biti poprilično velika. Godinama se pokušavalo olakšati njihovo kreiranje na način da se kreiraju jednostavni API (Aplikacijsko programsko sučelje) za visoku razinu izgradnje neuronskih mreža. Vodeći API otvorenog koda napisan u Pythonu, a napravljen na temelju Theano i kasnije TensorFlow platforme je Keras. Google je 2017. godine odlučio integrirati Keras s TensorFlowom, tako da je korisnicima dostupan preko tf.keras modula. Unatoč tome osim TensoFlowa u svojoj pozadini Keras i dalje može koristiti CNTK, Theano ili MXNet. Najveća prednost Keras-a je jednostavno kreiranje i učenje modela te je zbog toga vrlo široko primijenjen. Preko TensorFlowa Keras može raditi na procesorima GPU, TPU ili na skromnijem CPU. Također, zadatke može raspodijeliti na mnogobrojna računala. Unatoč svojoj jednostavnosti korištenja, pomoću Kerasa se mogu razvijati vrlo komplicirani modeli te zbog toga nije iznenađujuće što ga koriste platforme poput Netflixa, Yelpa i Ubera, a u istraživačke svrhe koriste ga CERN i NASA. Istraživanje provedeno na natjecanju Kaggle (internetskoj stranici koja predstavlja zajednicu znanstvenika i praktičara strojnog učenja) pokazalo je da su najboljih 5 timova za svoje projekte najviše koristili Keras

biblioteku za duboko učenje. Na sljedećem grafu je prikaz rezultata istraživanja tj. zastupljenosti programa za strojno učenje na natjecanju 2019. godine.



Slika 5. Rezultati istraživanja na natjecanju 2019. godine [14]

Keras se smatra API visoke razine jer minimalizira broj postupaka koje korisnik treba napraviti kako bi se napravio model neuronske mreže, dakle njegova upotreba orijentirana je korisniku. Pod pojmom da se Keras bazira na TensorFlow platformi misli se prvenstveno na to što Keras ne računa operacije nad tenzorima nego za to koristi TensorFlow kao svoj pozadinski motor.



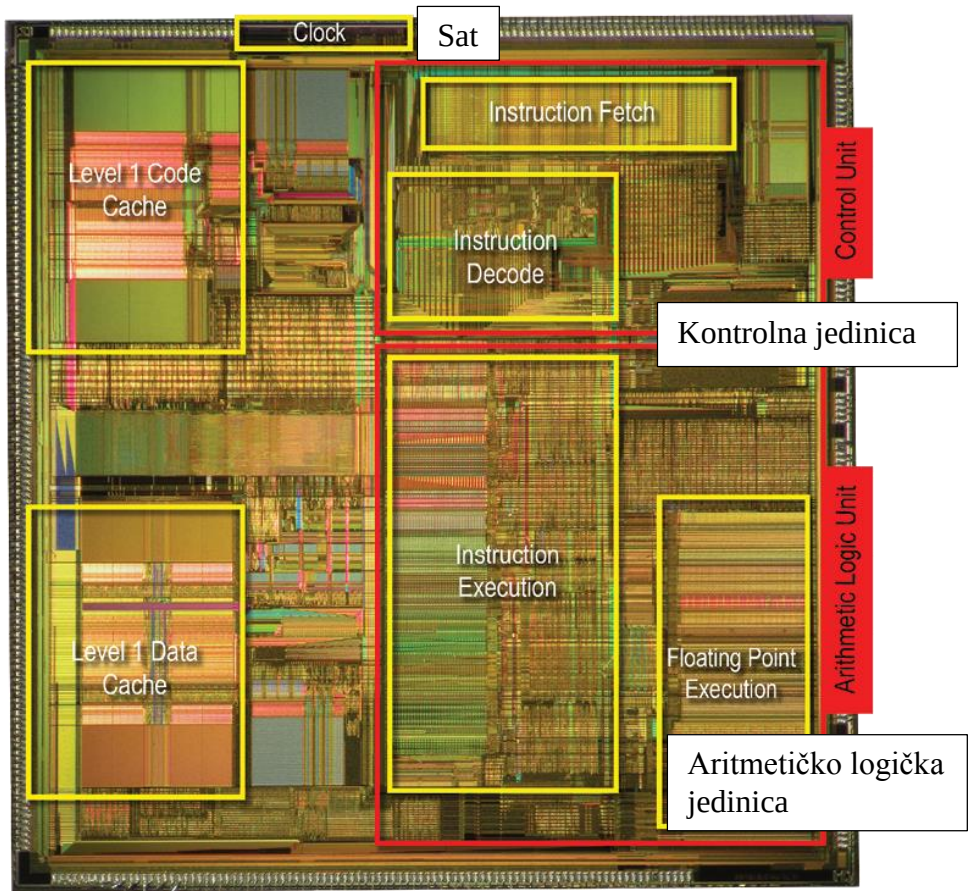
Slika 6. Raspodjela zadataka Kerasa, Tensorflowa i hardwera [13]

3.3. CPU,GPU i TPU procesori

Prethodno je objašnjeno da se za računanje tenzora, rad u Kerasu i TensorFlowu potrebno imati hardverske komponentne na kojima će se to računanje provoditi. U nastavku poglavlja opisati će se tri procesora koji se koriste za provedbu računskih operacija.

3.3.1. CPU

Svako računalo ima CPU ili procesor koji se vrlo često naziva mozgom računala. CPU je čip kojem je puni naziv Centralized Processing Unit te se pomoću njega se odvijaju sve funkcije računala. Današnji procesori imaju više jezgri, minimalno dvije, ali sve češće se koriste procesori sa 4, 6 , 8 ili 16 jezgara. Svaka jezgra je nezavisna cjelina za procesuiranje podataka. Na taj način procesor može izvršavati više zadataka preko različitih jezgri. Moderni programi osmišljeni su na način da koriste više jezgri kako bi povećali brzinu obrade programa.



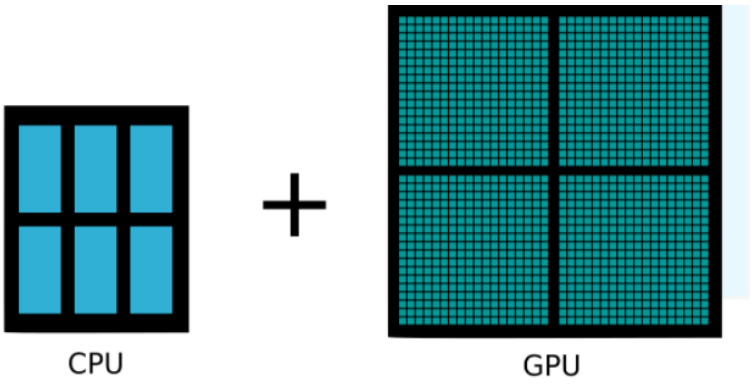
Slika 7. Prikaz starog Pentium procesora [15]

Na slici 7. pri vrhu se može vidjeti dio procesora zvan sat (clock) koji određuje radni takt tj. brzinu kojom se izvršavaju instrukcije. To je još jedna bitna stavka koja govori koliko je procesor moćan, a mjeri se u GHz (gigahertzima) ili rjeđe u MHz (megahertzima). Svakim

radnim taktom Kontrolna jedinica (Control unit) dohvaća i dekodira instrukciju. Zatim Aritmetičko logička jedinica (Arithmetic logic unit) izvršava tu instrukciju i sprema podatke. Taj se ciklus ponavlja milijardama puta u sekundi, tako npr. procesor od 2 GHz može izvesti dvije milijarde instrukcije u sekundi. [15]

3.3.2. GPU

GPU je procesor čiji je puni naziv na engleskom jeziku Graphics Processor Unit. To je čip koji je poznat po tome da se koristi kad programi zahtijevaju dobru grafiku, kao npr. u računalnim igrama. To je bila njegova prvenstvena zadaća, ali danas se sve više koriste General purpose GPU-i koji su dobar hardware kada je potrebno ubrzati računanje računski zahtjevnih procesa, brzo manipuliranje memorijom ili rješavanje zadataka koji su jako paralelizirani. Najveće razlika između CPU-a i GPU-a su sljedeća. Za razliku od GPU procesora, CPU procesor je namijenjen za rješavanje različitih zadataka, a pokušava se optimizirati da se ti zadaci što prije rješavaju, tj. da dohvaćanje, dekodiranje, izvršavanje i spremanje bude što brže i da promjena između tih zadataka bude brza. S druge strane GPU se ne oslanja na brzinu izvedbe nego na to da se što više informacija obradi odjednom, odnosno oslanja se na visoko paraleliziranje procesa. To je moguće izvesti s GPU procesorom jer ima znatno više jezgri nego što to ima CPU procesor. Jezgre CPU-a su jače i sposobnije od jezgara GPU procesora, ali njihova mnogobrojnost to nadoknađuje. S obzirom da se slika visoke rezolucije 1080p sastoji od nešto više od 2 milijuna piksela, a slika 4K rezolucije od preko 8 milijuna piksela koji se u vrlo kratkom vremenu istovremeno izmjenjuju jasno je zašto je mogućnost paralelnog računanja GPU procesora bitna u računalnoj grafici. Operacije koje GPU mora obaviti pri renderiranju slika su jednostavne operacije nad tenzorima koje se koriste i u strojnom učenju. Zbog tog je razloga vrlo korisno upotrijebiti sposobnost GPU-a da se računanje pri treniranju mreže paralelizira i na taj način značajno ubrza proces. Jeftiniji GPU procesori imaju oko 700 jezgri, a skuplji oko 4000 jezgri. Sljedeća slika pokazuje usporedbu CPU i GPU procesora i koliko više u prosjeku ima jezgri GPU procesor. [16]



Slika 8. Uspordba brojeva jezgri CPU-a i GPU-a [16]

3.3.3. TPU

Iako GPU procesori pokazuju dobre rezultate u strojnom učenju, oni nisu bili dizajnirani prvenstveno za tu svrhu. TPU je procesor koji je napravljen baš za strojno učenje odnosno za operacije nad velikim tenzorima, koji se javljaju u strojnom učenju. Njegov puni naziv je Tensor processing unit i brži je čak do 20 puta od najskupljih GPU procesora. TPU se sastoji od od dvije cjeline. Prva je MXU (Matrix Multiply Unit) koja je zadužena za matrično množenje, dok je druga VPU (Vector processing unit) zaslužna za provođenje ostalih zadataka kao što je računanje aktivacijskih funkcija. Za odlične performanse TPU procesora zaslužna je njegova sistolička arhitektura (sistoličko polje). Naziv sistolički obično se veže uz sistolički krvni tlak koji nastaje izbacivanjem krvi u arterije, ali ovdje se taj naziv koristi jer se ulazni podaci ubacuju u procesor u valovima slično kao što srce pumpa krv u žile. Ulazni podaci ulaze u sistoličko polje koje se sastoji od puno međusobno povezanih procesnih elemenata zaduženih za množenje. Najjednostavniji primjer koji objašnjava rad TPU procesora i njegovu prednost u strojnom učenju nad GPU procesorima je množenje matrica velikih dimenzija. Ako se žele pomnožiti dvije matrice, redovi jedne matrice moraju se skalarno množiti s stupcima druge matrice. U slučaju da je rezultirajuća matrica dimenzija 128x128 za brz rad GPU procesora trebalo bi se aktivirati što više slobodnih jezgri da paralelno računaju svaku vrijednost matrice. U tom slučaju trebalo bi se aktivirati 16384 jezgri (128x128=16384), ali to nije moguće jer najbolji GPU procesori imaju otprilike oko 4000 jezgri. S druge strane TPU procesori mogu odjednom matrično množiti matrice takvih dimenzija jer su procesne jedinice toliko male da TPU može koristiti 16 tisuća takvih jedinica da napravi kalkulaciju. Rad sistoličke mreže objasniti će se na primjeru množenja matrice A i B dimenzija 4x4. [17]

Neka je matrica A:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}, \tag{2}$$

matrica B:

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}, \tag{3}$$

a rezultat množenja je matrica C:

$$C = A \cdot B = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}. \tag{4}$$

Onda se mogu ispisati elementi matrice C kao:

$$c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + a_{14} \cdot b_{41} \qquad c_{31} = a_{31} \cdot b_{11} + a_{32} \cdot b_{21} + a_{33} \cdot b_{31} + a_{34} \cdot b_{41}$$

$$c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} + a_{14} \cdot b_{42} \qquad c_{32} = a_{31} \cdot b_{12} + a_{32} \cdot b_{22} + a_{33} \cdot b_{32} + a_{34} \cdot b_{42}$$

$$c_{13} = a_{11} \cdot b_{13} + a_{12} \cdot b_{23} + a_{13} \cdot b_{33} + a_{14} \cdot b_{43} \qquad c_{33} = a_{31} \cdot b_{13} + a_{32} \cdot b_{23} + a_{33} \cdot b_{33} + a_{34} \cdot b_{43}$$

$$c_{14} = a_{11} \cdot b_{14} + a_{12} \cdot b_{24} + a_{13} \cdot b_{34} + a_{14} \cdot b_{44} \qquad c_{34} = a_{31} \cdot b_{14} + a_{32} \cdot b_{24} + a_{33} \cdot b_{34} + a_{34} \cdot b_{44}$$

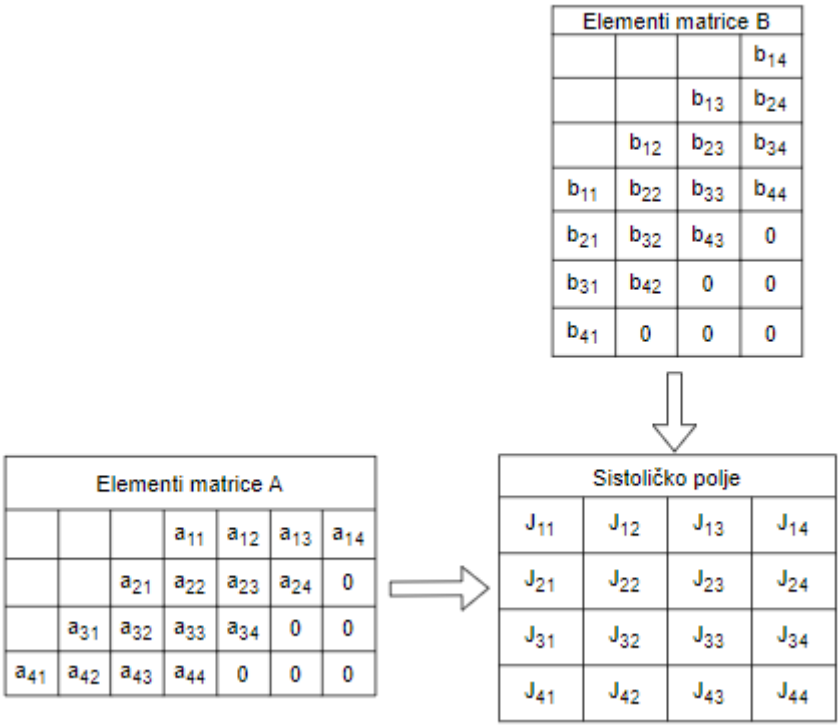
$$c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} + a_{24} \cdot b_{41} \qquad c_{41} = a_{41} \cdot b_{11} + a_{42} \cdot b_{21} + a_{43} \cdot b_{31} + a_{44} \cdot b_{41}$$

$$c_{22} = a_{21} \cdot b_{12} + a_{22} \cdot b_{22} + a_{23} \cdot b_{32} + a_{24} \cdot b_{42} \qquad c_{42} = a_{41} \cdot b_{12} + a_{42} \cdot b_{22} + a_{43} \cdot b_{32} + a_{44} \cdot b_{42}$$

$$c_{23} = a_{21} \cdot b_{13} + a_{22} \cdot b_{23} + a_{23} \cdot b_{33} + a_{24} \cdot b_{43} \qquad c_{43} = a_{41} \cdot b_{13} + a_{42} \cdot b_{23} + a_{43} \cdot b_{33} + a_{44} \cdot b_{43}$$

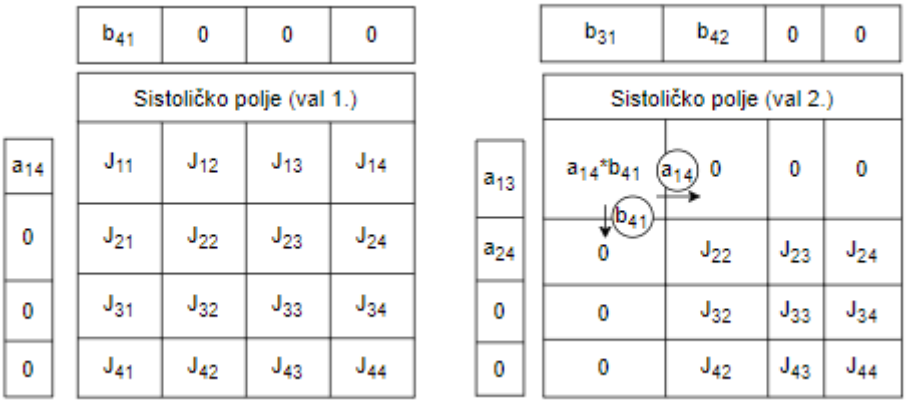
$$c_{24} = a_{21} \cdot b_{14} + a_{22} \cdot b_{24} + a_{23} \cdot b_{34} + a_{24} \cdot b_{44} \qquad c_{44} = a_{41} \cdot b_{14} + a_{42} \cdot b_{24} + a_{43} \cdot b_{34} + a_{44} \cdot b_{44}$$

S obzirom da rezultirajuća matrica ima 16 elemenata, za izračun će biti potrebno jednako toliko procesnih jedinica (akumulatora) koje će činiti sistoličko polje. Na slici 5. procesne jedinice su prikazane slovima J_{xy}, gdje x i y označuju poziciju jedinice u mreži. Slika prikazuje redoslijed ulaska podataka u sistoličko polje. Elementi matrice A ući će u procesne jedinice redoslijedom po stupcima tablice matrice A, dok će za tablicu B matrice elementi ulaziti u sistoličko polje po redcima.

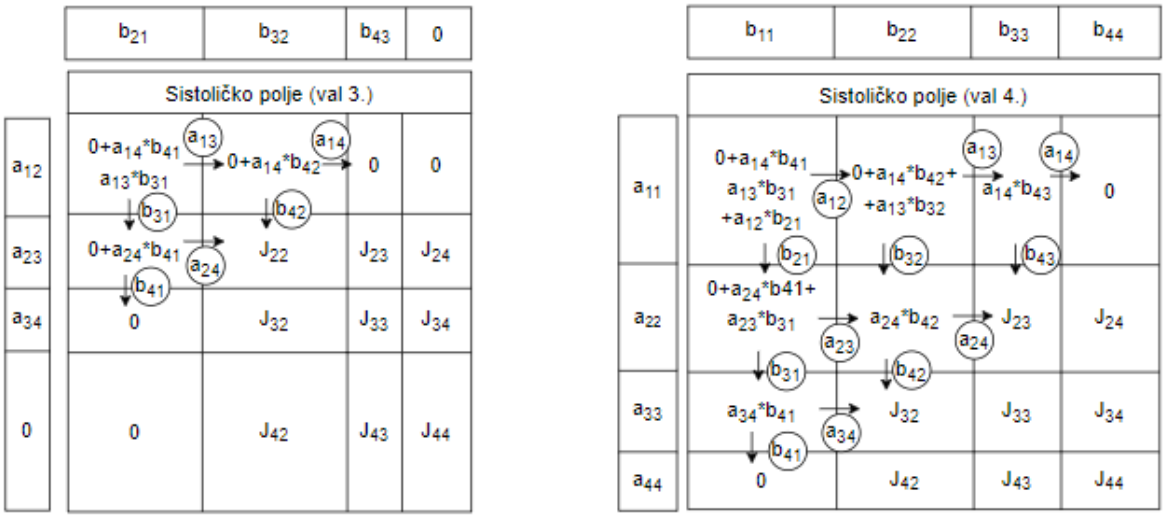


Slika 9. Raspored ulaska elemenata matrica u sistoličko polje

S takvim rasporedom ulaznih elemenata procesne jedinice će množiti ulazne podatke i zbrajati ih s prethodnim umnoškom, zatim će ulazne podatke proslijediti drugim jedinicama u vertikalnom i horizontalnom smjeru. Na slikama od 6. do 14. ulazni podaci koji dolaze u valovima su prikazani u pravokutnicima iznad i s lijeve strane sistoličkog polja. Podaci koji se međusobno prosljeđuju između procesnih jedinica nalaze se u krugu popraćenim strelicom zbog boljeg razumijevanja smjera prosljeđivanja.

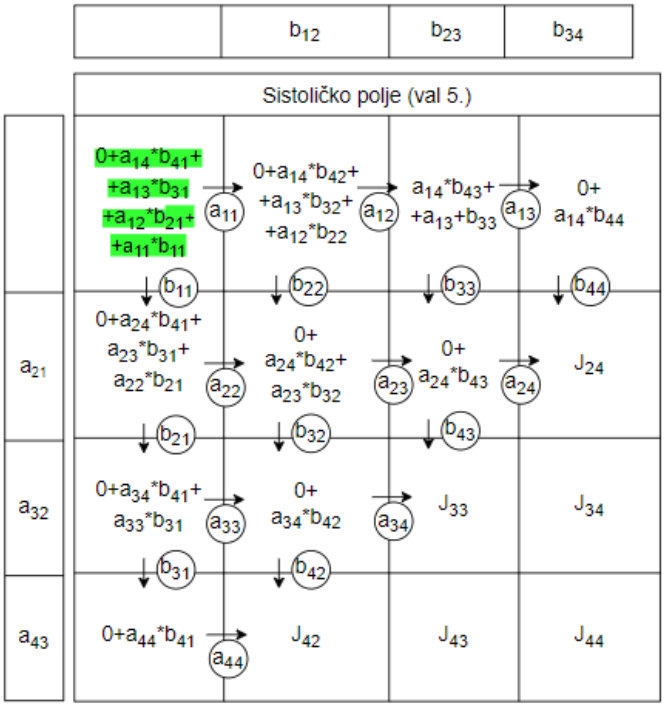


Slika 10. Prvi i drugi val ulaznih podataka

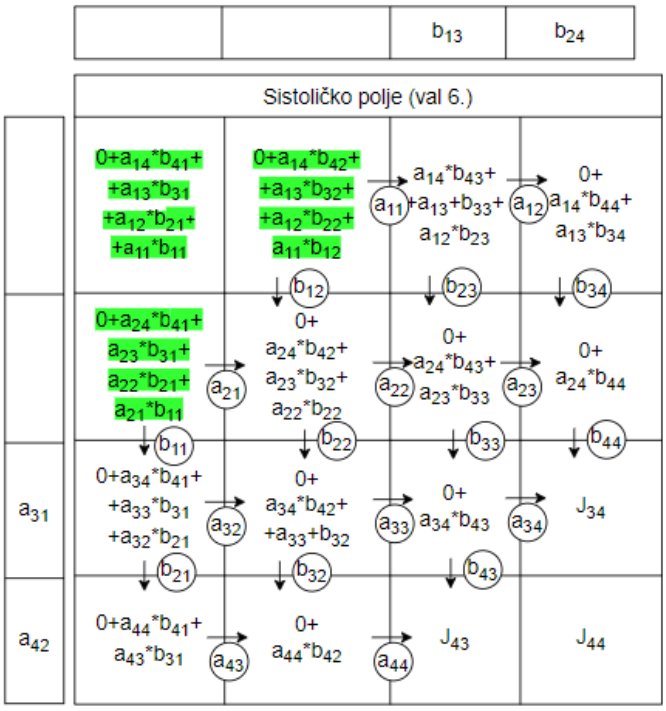


Slika 11. Treći i četvrti val ulaznih podataka

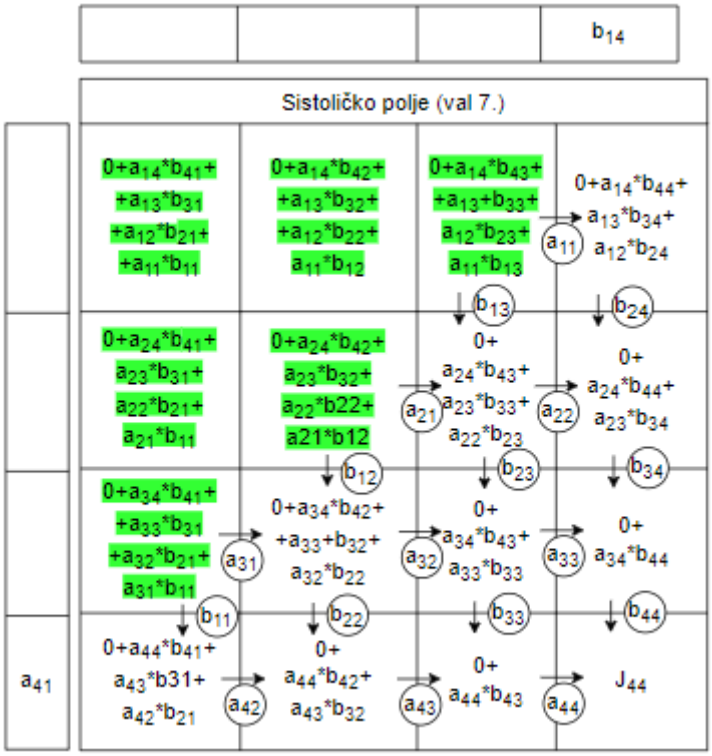
Na slici 8. zelenom bojom označena je procesna jedinica koja je obradila sve ulazne podatke i izračunala konačan rezultat. Tako će biti označene u svim ostalim slikama sve procesne jedinice koje su završile proces računanja.



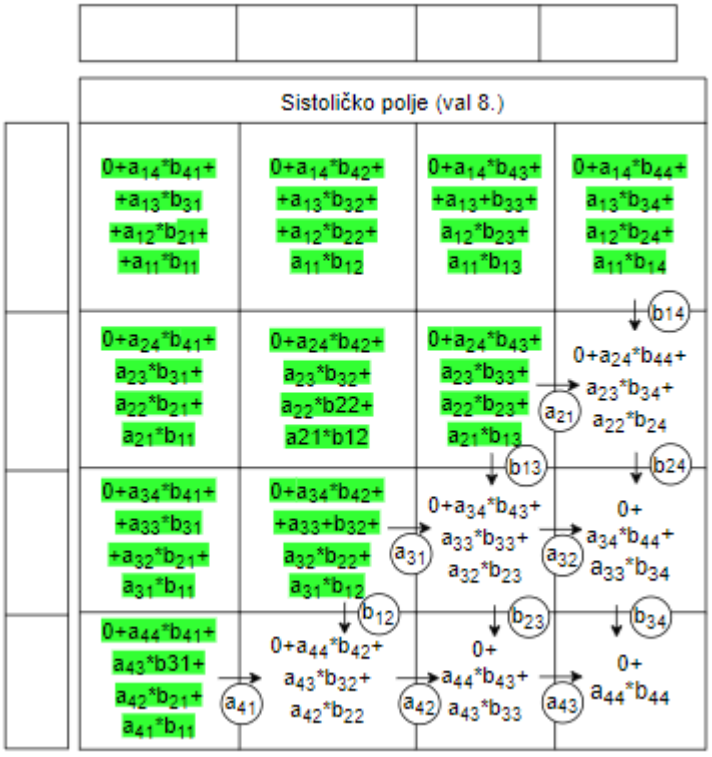
Slika 12. Peti val ulaznih podataka



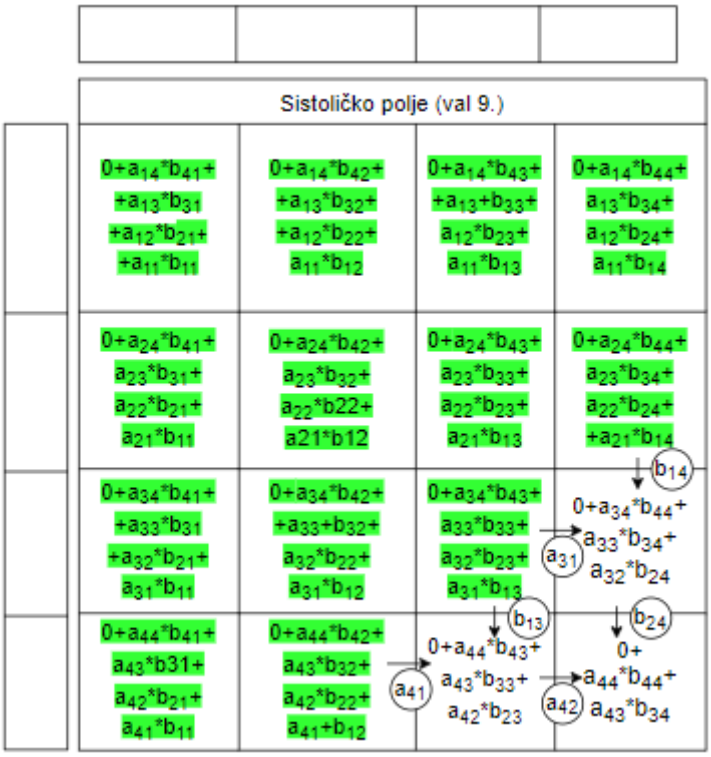
Slika 13. Šesti val ulaznih podataka



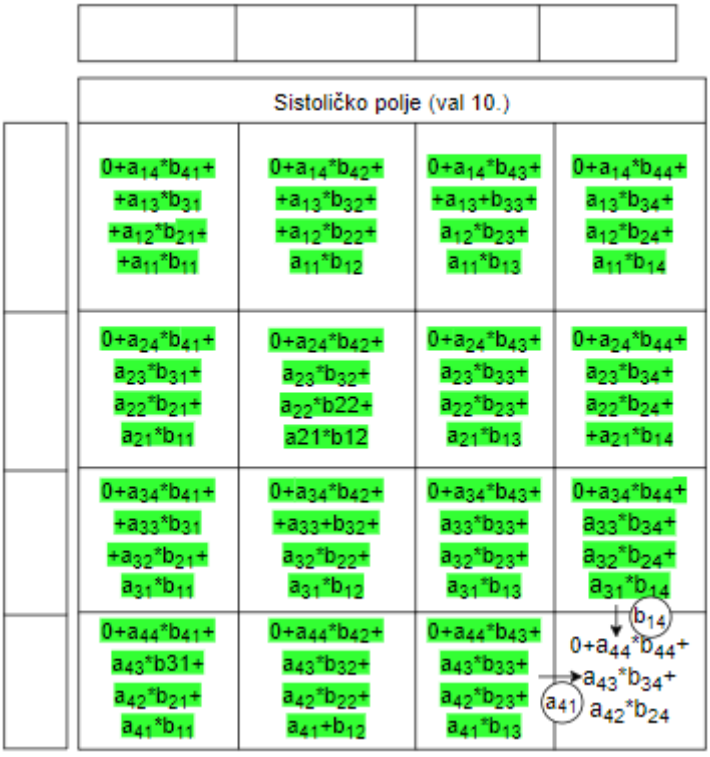
Slika 14. Sedmi val ulaznih podataka



Slika 15. Osmi val ulaznih podataka



Slika 16. Deveti val ulaznih podataka



Slika 17. Deseti val ulaznih podataka

Slika 18. Jedanaesti val ulaznih podataka

Na slici 14. može se vidjeti završen proces računanja umnoška matrica A i B. Svaka procesna jedinica sadrži jednu vrijednost rezultirajuće matrice C po odgovarajućoj poziciji.

U primjeru se moglo vidjeti kako podaci struje iz jedne procesne jedinice u drugu. Za razliku od GPU-a to znači da oni ne moraju biti spremljeni u memoriju niti se potom pretraživati iz memorije ili registara. Krajnji rezultat je značajno veća brzina TPU procesora pri množenju matrica nego GPU procesora, što ga čini najboljom opcijom za implementaciju strojnog učenja.

3.3.4. Dostupnost GPU i TPU procesora

S obzirom da se korištenje CPU procesora nikako ne preporučuje za učenje kompleksnijih mreža, jer učenje može trajati danima, istraživači i ostali korisnici najčešće se odlučuju za korištenje GPU i TPU procesora. Postoji 3 opcije za korištenje bržih procesora. Prvi način je da se fizička komponenta procesora kupi i instalira na računalu. To je najbolja opcija za one koji često razvijaju neuronske mreže s obzirom da su procesori poprilično skupi. Druga mogućnost je koristiti instance preko Google Clouda ili Amazon Web Services (AWS) za učenje mreže. Instance su virtualni serveri čije se konfiguracije mogu odabrati prema željama korisnika. To znači da se može odabrati željeni hardware kao što su procesori, memorija i ostale komponente. Cijena posudbe instanci određuje se prema odabranim konfiguracijama i

obračunava se prema satu korištenja. Treća mogućnost je korištenje Google Colaboratorya (Google Colab) koji je besplatna Jupyter notebook usluga od Googlea, napravljena za strojno učenje i vrti se samo u oblaku. Google Colaboratory omogućava korištenje GPU i TPU procesora za treniranje mreže, ali besplatna verzija dozvoljava korištenje s vremenskim limitom. Postoji i pretplatnička verzija koja garantira prednost pri pristupanju boljim procesorima u odnosu na korisnike koji koriste besplatnu inačicu.

U izradi ovog rada koristila se besplatna verzija Google Colaba sa pristupom GPU i TPU procesorima. Znalo se dogoditi da se pri učenju mreže dosegne limit korištenja ili da procesor trenutno nije dostupan, ali ta pauza nije trajala dugo te se proces učenja mogao nastaviti na nekom drugom procesoru kroz desetak minuta. Besplatna verzija Google Colaba daje mogućnost odabira koja se vrsta procesora želi koristiti (GPU ili TPU), ali ne i model. Model procesora dodjeljuje Google Colab prema njihovoj dostupnosti i tipu korisničkog računa (besplatan ili pretplatnički).

4. PROCES IZRADE NEURONSKE MREŽE

4.1. Postupak stvaranja neuronske mreže u Kerasu

Svaka neuronska mreža da bi se izradila i koristila u Kerasu mora proći kroz postupak koji se sastoji od 5 faza. Postupak se mora slijediti sljedećim redom [18]:

- Definiranje modela mreže
- Kompiliranje modela mreže
- Učenje (treniranje) mreže
- Evaluiranje mreže
- Stvaranje pretpostavki mreže

4.1.1. Definiranje modela mreže

U ovoj fazi korisnik kreira model mreže odnosno bira koji će slojevi graditi mrežu i kako će se međusobno povezivati. Keras dopušta različite načine kako se može kreirati model, a svi načini su opisani u poglavlju Pristupi u izradi modela neuronskih mreža u Kerasu.

4.1.2. Kompiliranje modela mreže

Druga faza pri stvaranju neuronske mreže je kompiliranje mreže. U toj se fazi kreirani slojevi u prethodnoj fazi transformiraju u niz tenzora koji su pogodni za vršenje matematičkih operacija pomoću CPU, GPU ili TPU-a. Kompiliranje mreže uvijek je nužno učiniti nakon stvaranja modela. Također, u ovom koraku je potrebno odrediti optimizacijski algoritam koji će se koristiti za trening mreže, ali i funkciju gubitka prema kojoj će se mreža evaluirati.

Mreža se u Kerasu može kompilirati pomoću metode model.compile( ), koja prima argumente funkciju gubitka koja se želi koristiti, optimizirajući algoritam i metriku. Metrika je obično postavljena na 'accuracy' (preciznost) koja računa koliko često predikcije mreže odgovaraju etiketama. U sljedećem redu je primjer koda koji služi za kompiliranje mreže. [18]

```
model.compile(loss='binary_crossentropy', optimizer = opt, metrics =
['accuracy'])
```

4.1.3. Učenje (treniranje) mreže

Učenje mreže podrazumijeva podešavanje težina modela kako bi bio rezultat bio bolji tj. da vrijednosti težina mreže minimiziraju funkciju gubitka. U slučaju GAN mreža, dobiveni rezultati nakon treninga bi trebali biti slični stvarnim uzorcima. Za učenje mreže potrebno je strukturirati ulazne trening podatke s odgovarajućim izlaznim podacima. Mreža se trenira s

backpropagation algoritmom, a optimizira pomoću optimizirajućeg algoritma i funkcije gubitka. U Kerasu mrežu se može trenirati pomoću metode fit( ). Njoj se mora definirati broj epoha treninga, odnosno koliko će puta cijeli set podataka za učenje proći mrežom kako bi se težine ažurirale. U jednoj epohi može proći jedan uzorak za učenje kroz mrežu, ali može i više njih. Tu brojku određuje batch size koji govori koliko uzoraka za učenje mreže će proći mrežom prije nego li se težine modela ažuriraju. Također, postoji i drugi način treniranja mreže, a taj je pomoću metode train_on_batch koja uzima samo jedan batch podataka koji će proći mrežom i tako promijeniti parametre mreže. [19]

4.1.4. Evaluiranje mreže

Evaluiranje mreže je proces nakon učenja. Mreža se može evaluirati pomoću podataka korištenih za trening, ali dobiveni rezultati evaluiranja nisu toliko korisni s obzirom da je mreža već vidjela te podatke. Najbolje je evaluirati mrežu na zasebnim podacima gdje se realno vidi koliko dobro mreža izvodi svoj zadatak. Metodom evaluate( ) u Keras biblioteci vrlo jednostavno se može evaluirati model. Ona iteracijski računa vrijednosti gubitka i metrike modela na jednom setu uzoraka (tj. batch-u). Slijedi primjer koda u kojem se koristi metoda evaluate koja u svojim argumentima prima ulazne podatke X i ciljane podatke y, a računa gubitak i preciznost. [19]

```
loss, accuracy = model.evaluate(X, y)
```

4.1.5. Stvaranje pretpostavki mreže

Mreža koja je prošla proces treninga, a rezultati evaluacije zadovoljavaju korisnika, može se koristiti za stvaranje svojih pretpostavki. Rezultati mreže bit će u obliku izlaznog sloja mreže. Metoda koja omogućava stvaranje pretpostavki mreže je predict ( ). Primjer koda je sljedeći, gdje su X ulazni podaci mreže [19]:

```
predictions = model.predict(X)
```

4.2. Pristupi u izradi modela neuronskih mreža u Kerasu

Građevna cjelina svakog modela neuronskih mreža su slojevi. Slojevi se razlikuju po svojoj strukturi i načinu primjene. Svi slojevi korišteni u ovom radu detaljno će biti objašnjeni u poglavlju Slojevi, dok će se ovo poglavlje fokusirati na različite pristupe u izradi modela u Kerasu. Modeli u Kerasu mogu se napraviti na 3 različita načina, a svaki pristup ima svoje prednosti i nedostatke. Načini izrade modela su [19]:

- Sekvencijalni
- Funkcionalni
- Podklasno modeliranje

4.2.1. Sekvencijalni API

Prvi i najjednostavniji pristup za jednostavne modele je sekvencijalni način. To je način u kojem se slojevi dodaju sloj po sloj ili kao lista. Zbog jednostavnosti može se koristiti samo za razvoj modela koji imaju linearnu topologiju. Ovaj način izrade modela nije primjenjiv kada se treba napraviti model s više ulaza ili više izlaza. Isto tako, svaki sloj u modelu ne smije imati više ulaza ili izlaza. U slučaju izrade mreže koja mora dijeliti slojeve ili ima grananja u svojoj strukturi mora se koristiti Funkcionalni API jer takve modele mreže nije moguće napraviti u Sekvencijalnom API-u. [20]

```
# Primjer izrade modela pomoću Sekvencijalnog API-a [DODAVANJE KAO LISTA]
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
```

U primjeru se vidi kako se model dodaje kao lista slojeva konstruktoru klase Sequential. Slojevi su u ovom slučaju potpuno povezani (Dense slojevi) koji će biti detaljno objašnjeni u nastavku rada.

Model se može stvarati sekvencijalno i pomoću metode .add( ). U tom slučaju slojevi se dodaju iterativno slagajući se na način da se sloj nastavlja na drugi sloj redom kojim su dodani u kodu [20].

Sekvencijalni način izrade modela može se koristiti za izradu jednostavnijih GAN mreža, ali ne za CGAN mreže koje imaju više ulaza u slojevima.

```
# Primjer izrade modela pomoću Sekvencijalnog API-a [POMOĆU METODE .add()]
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

4.2.2. Funkcionalni API

To je alternativni način izrade modela neuronskih mreža koji omogućuje stvaranje kompleksnijih modela jer nudi veću fleksibilnost pri izradi. Funkcionalni API koristio se u izradi ovog rada zbog složene konstrukcije CGAN konvolucijske mreže. Korištenje funkcionalnog API-a omogućava izradu modela nelinearnih topologija, slojevi se mogu dijeliti, a također mogu imati i više ulaza odnosno izlaza.

Općenito neuronske se mreže često prikazuju pomoću grafova u obliku stabla zbog lakšeg razumijevanja i preglednosti. Takvi grafovi mogu poslužiti kao dobar predložak za izradu modela u funkcionalnom API-u. Za ulaz podataka u mrežu mora se napraviti zaseban ulazni sloj (Input sloj) koji definira strukturu ulaznih podataka i uzima za argument N-terac (engl. Tuple).

N-terac je tip podataka u Pythonu koji omogućava pohranu skupova različitih vrijednosti. Može se usporediti s listom koja je također tip podataka u Pythonu s tom razlikom što je N-terac nepromjenjiv tip podatka. N-terac kao argument u ulaznom sloju definira dimenzionalnost ulaznih podatka. Ulazni podaci mogu imati više dimenzija, ali važno je napomenuti da se za jednodimenzionalni ulazni podatak (npr. red uzoraka) mora definirati na način da se napiše dimenzija i zatim zarez iza dimenzije. Na taj se način određuju jednodimenzionalni N-terci u Pythonu. [13]

```
# primjer stvaranja Input sloja
inputs = keras.Input(shape=(784,))
```

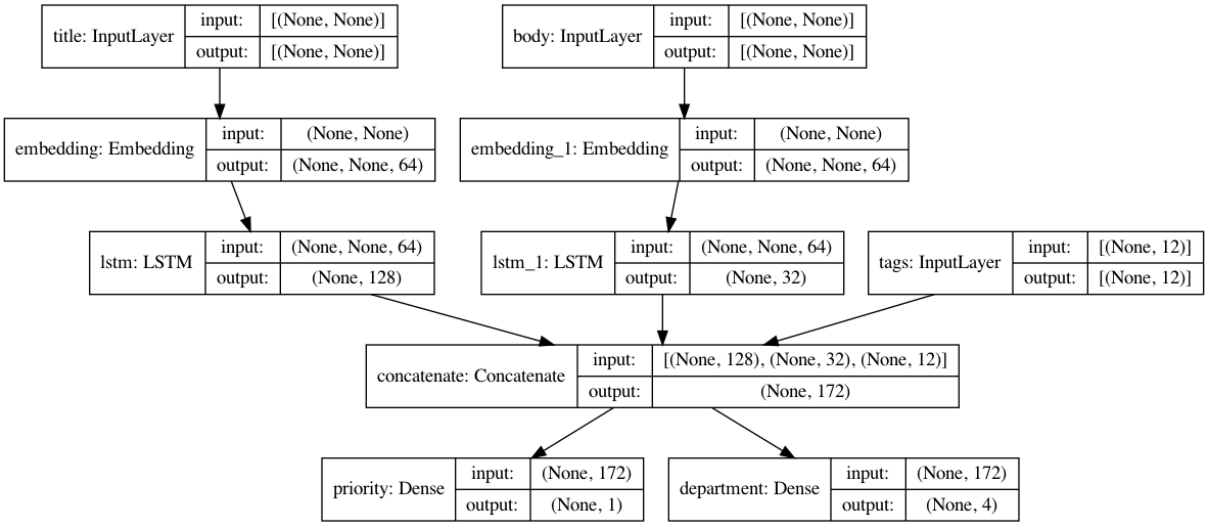
Stvaranje sljedećeg sloja vrlo je jednostavno jer potrebno je samo pozvati sloj koji se želi koristiti i upisati sloj koji će biti njegov ulaz (obično je to prethodni sloj) tako da se koristi notacija zagrada.

```
# primjer dodavanja sloja na input sloj
inputs = keras.Input(shape=(784,))
x = layers.Dense(64, activation="relu")(inputs)
```

Kao što se vidi u primjeru iznad novi Dense sloj dobiva ulazne podatke od izlaza input sloja. U zagradi sloja Dense osim njegove dimenzije određena je i aktivacijska funkcija ReLu. Aktivacijske funkcije objašnjene su u zasebnom poglavlju.

Ovakav način slijeđenja slojeva je i dalje linearne topologije jer samo jedan sloj prethodi drugom. Ukoliko se žele koristiti višebrojni ulazi i izlazi, to se može napraviti pomoću metode

.concatenate( ) koja prihvaća više slojeva kao argumente i spaja ih u jedan sloj. Tako se može dobiti efekt grananja mreže. [18] Sljedeća slika preuzeta je kao primjer iz službene Kerasove dokumentacije i prikazuje stablo koje objašnjava grananje koristeći metodu .concatenate( ).



Slika 19. Prikaz grananja modela [21]

Za slučaj koji prikazuje slika 3. kod koji se odnosi na treći i četvrti red na slici za spajanje 3 sloja title_features, body_features i tags_input u jedan x sloj bio bi sljedeći:

```
x = layers.concatenate([title_features, body_features, tags_input])
```

, gdje su title_features i body_features LSTM slojevi , tags_input je ulazni sloj, a x je sloj koji je sačinjen od prethodno navedenih slojeva što se može vidjeti jer je dimenzija sloja zbroj dimenzija njegovih ulaznih slojeva.

Jednom kad se odrede svi slojevi mreže to ne znači da je model mreže napravljen. Model se mora izraditi pozivajući klasu Model( ) iz Keras biblioteke. U argumentima klase bitno je definirati ulazni sloj modela i izlazni sloj modela. Da se želi na temelju prošlog primjera napraviti model mreže gdje je postojao ulazni sloj zvan Inputs i zatim Dense sloj x koji je ujedno i izlazni sloj modela treba upisati sljedeći kod:

```
model = keras.Model(inputs=inputs, outputs=x, name="ime modela")
```

4.2.3. Podklasno modeliranje

Posljednji način izrade modela u Kerasu je podklasno modeliranje. Takvo stvaranje modela je na najnižoj razini jer svaki dio modela mora se samostalno definirati unutar podklase. Tim načinom korisnik dobiva apsolutnu kontrolu nad modelom. Glavni nedostatak ovog pristupa je taj što je ovaj način programiranja modela podložan stvaranju grešaka koje je teže uočiti.

```
# Primjer podklasnog modeliranja
class Nas_model(keras.Model):

def __init__(self, num_departments):
super().__init__()
self.concat_layer = layers.Concatenate()
self.mixing_layer = layers.Dense(64, activation="relu")
self.priority_scorer = layers.Dense(1, activation="sigmoid")
self.department_classifier = layers.Dense(
num_departments, activation="softmax")

def call(self, inputs):
title = inputs["title"]
text_body = inputs["text_body"]
tags = inputs["tags"]
features = self.concat_layer([title, text_body, tags])
features = self.mixing_layer(features)
priority = self.priority_scorer(features)
department = self.department_classifier(features)
return priority, department
```

U primjeru iznad napravila se klasa Nas_model koja je podklasa klase Model iz Keras biblioteke (navedena u zagradama). U Python jeziku pri kreiranju klase u zagradama se može navesti roditeljska klasa od koje će nova klasa naslijediti svojstva. Tako je klasa Nas_model naslijedila svojstva od klase Model . U __init__() metodi moraju se navesti svi slojevi koji će se koristiti u modelu, dok će se u metodi call() definirati njihov raspored i međusobna povezanost.

Nakon izrade podklase mora se napraviti njegova instanca koja će biti model mreže. Instanca je objekt koji pripada određenoj klasi. U sljedećem kodu napraviti će se instanca gotov_model koji pripada klasi Nas_model. [13]

```
gotov_model = Nas_model(num_departments=4)
```

Jednom kad se napravi model njega se može trenirati baš kao što bi se to učinilo i preko Sekvencijalnog i Funkcionalnog API-a.

4.3. Klasa Sloj (Layer class)

Neuronske mreže neovisno o svojoj strukturi i složenosti sastoje se od slojeva mreže. Slojevi su fundamentalne građevni objekti neuronskih mreža koji sadrže neurone i omogućuju prolaz informacija prema drugim slojevima. Slojevi imaju ulaze koji prihvaćaju podatke od prethodnog sloja (osim ako se ne radi o prvom sloju) i izlaze koji nakon određenih računskih operacija prenose informacije drugim slojevima.

U Keras biblioteci koja se koristila u izradi ovog rada svi slojevi su objekti koji pripadaju klasi Layer. Tako svaki sloj koji se koristi za izradu neuronske mreže preuzima značajke i metode iz

klase Layer. Prethodno je navedeno da se slojevi sastoje od neurona, a oni su povezani s okolinom preko težinskih vrijednosti (težina). Težinske vrijednosti mogu biti pozitivne, negativne ili nula. Ukoliko je vrijednost težine nula, to znači da neuron nema uspostavljenu vezu s vanjskom okolinom preko tog ulaza. Brojčana vrijednost težina označava intenzitet veze gdje veće vrijednosti težina predstavljaju veze jačeg značaja. [13]

5. PROCESI U KONVOLUCIJSKIM MREŽAMA

Primarna CGAN mreža koja je napravljena u sklopu ovog rada nije imala konvolucijske slojeve. Rezultati koje je kreirala mreža bili su vrlo zašumljeni, a samo učenje mreže je trajalo dugo. Nova CGAN mreža napravljena je s konvolucijskim slojevima koji su odličan odabir za mreže koje rade sa slikama. Oni omogućuju bolju obrada slika i lakše računanje dok se odvija proces učenja mreže. Mreže koje imaju konvolucijske slojeve se nazivaju konvolucijske neuronske mreže (skraćenica CNN na eng.) U nastavku će biti objašnjeni procesi konvolucije, sažimanja, popunjavanje do rubova i konvolucijski slojevi.

5.1. Konvolucija

Konvolucija je proces koji se odvija nad tenzorima. U proces konvolucije ulaze dva tenzora, prvi je slika, a drugi je filter. Izlaz iz procesa je samo jedan rezultirajući tenzor. Sljedeća formula opisuje proces konvolucije:

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix} = \sum_{i=1}^{9} a_i k_i \tag{5}$$

, gdje su a_n, $n \in [1,9]$ elementi prve matrice, a k_m $m \in [1,9]$ elementi filtera.

Rezultat konvolucije je suma umnožaka elemenata na istim pozicijama. Za proces konvolucije nužno je koristiti konvolucijske filtere. Oni se često nazivaju kerneli, a naziv dolazi iz klasičnog pristupa procesiranja slika. U konvolucijskom procesu pomoću njih se mogu detektirati značajke unutar slike. Tako su neki od mnogih postojećih filtera kerneli za detekciju horizontalnih rubova, filteri za detekciju vertikalnih rubova, filteri za zamagljivanje slika i drugi. Oni su matrice najčešćih dimenzija 3x3 ili 5x5 i obično su manji od tenzora s kojima sudjeluju u procesu konvolucije. Na primjer slike mogu imati dimenzije 1024x1024x3, gdje je zadnja dimenzija dubina RGB slike. Rezolucija slika može biti i veća u naprednim primjenama. Zbog lakšeg shvaćanja djelovanja filtera kada dimenzije matrica slike i filtera nisu jednake, proces će se opisati na sljedećem primjeru. [22]

Neka je matrica A tenzor slike dimenzija 4x4:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{bmatrix}, \quad (6)$$

,gdje su elementi matrice a_n, $n \in [1,16]$ indeksirani tako da njihovi indeksi ne označavaju broj retka i stupca u kojem se nalaze nego redoslijed elemenata po redcima.

Neka je matrica filtera K:

$$K = \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix}, \quad (7)$$

gdje su k_n $n \in [1,9]$ su elementi filtera.

S obzirom da je filter K dimenzija 3x3 može obuhvatiti samo 9 elemenata matrice A to znači da će morati mijenjati svoju poziciju u velikoj matrici. Prvo će krenuti s gornjim lijevim dijelom matrice A označen plavim okvirom, zatim će se pomaknuti u gornji desni kut označen zelenom bojom, a nakon toga će se pomaknuti na pozicije žutog pa crvenog okvira.

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{bmatrix}$$

Za svaku poziciju u velikoj matrici filter će zajedno s obuhvaćenim elementima matrice A napraviti konvolucijski proces.

Za poziciju plavog okvira:

$$B_1 = A_1 * K = a_1 \cdot k_1 + a_2 \cdot k_2 + a_3 \cdot k_3 + a_5 \cdot k_4 + a_6 \cdot k_5 + a_7 \cdot k_6 + a_9 \cdot k_7 + a_{10} \cdot k_8 + a_{11} \cdot k_9 \quad (8)$$

Za poziciju zelenog okvira:

$$B_2 = A_2 * K = a_2 \cdot k_1 + a_3 \cdot k_2 + a_4 \cdot k_3 + a_6 \cdot k_4 + a_7 \cdot k_5 + a_8 \cdot k_6 + a_{10} \cdot k_7 + a_{11} \cdot k_8 + a_{12} \cdot k_9 \quad (9)$$

Za poziciju žutog okvira:

$$B_3 = A_3 * K = a_5 \cdot k_1 + a_6 \cdot k_2 + a_7 \cdot k_3 + a_9 \cdot k_4 + a_{10} \cdot k_5 + a_{11} \cdot k_6 + a_{13} \cdot k_7 + a_{14} \cdot k_8 + a_{15} \cdot k_9 \quad (10)$$

Za poziciju crvenog okvira:

$$B_4 = A_4 * K = a_6 \cdot k_1 + a_7 \cdot k_2 + a_8 \cdot k_3 + a_{10} \cdot k_4 + a_{11} \cdot k_5 + a_{12} \cdot k_6 + a_{14} \cdot k_7 + a_{15} \cdot k_8 + a_{16} \cdot k_9 \quad (11)$$

Rezultat konvolucije je matrica B:

$$B = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}, \quad (12)$$

U ovom primjeru filter se micao za jedan korak u desno i jedan korak prema dolje, no to nije nužno uvijek tako. Filter se može micati i za više od jednog koraka, a to naravno utječe na konačnu dimenziju rezultirajuće matrice. Konačna dimenzija matrice B može se prikazati slijedećom formulom. [22]

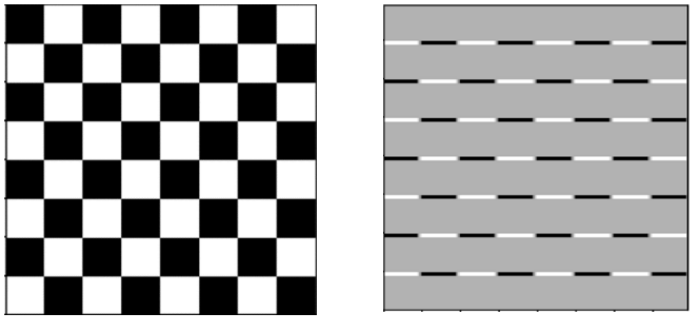
$$n_B = \left\lfloor \frac{n_A - n_K}{s} + 1 \right\rfloor, \quad (13)$$

, gdje su n_B dimenzija konačne matrice B, n_A dimenzija matrice A, n_K dimenzija matrice filtera K i s korak filtera.

Kao što je prethodno navedeno filtera ima mnogo vrsta, a neki od najpoznatijih su:

Filter za detekciju horizontalnih rubova $K = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

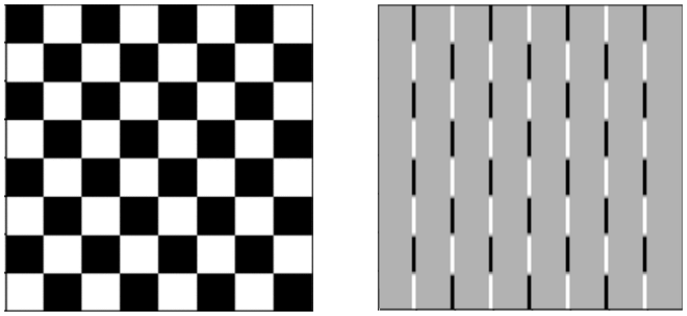
Slika 16. prikazuje primjer primjene filtera za detekciju horizontalnih rubova. Filter je izdvojio sve horizontalne rubove gdje je na originalnoj slici došlo do promjene crne i bijele boje.



Slika 20. Lijevo-prikaz slike prije konvolucije, desno-slika nakon konvolucije [22]

Filter za detekciju vertikalnih rubova $K = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

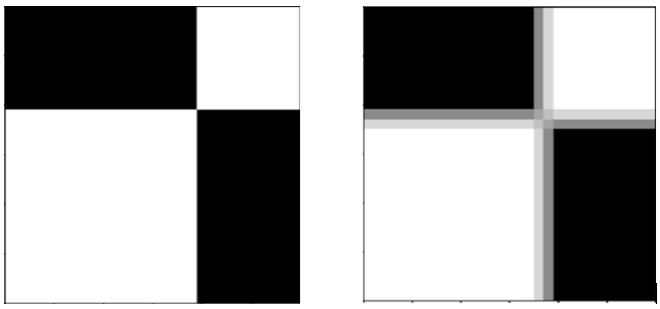
Na slici 17. vidi se primjer korištenja filtera za detekciju vertikalnih rubova. Filter je također kao u gornjem primjeru pronašao rubove kao dijelove slike gdje dolazi do promjene boje.



Slika 21. Lijevo-prikaz slike prije konvolucije, desno-slika nakon konvolucije [22]

Filter za zamagljivanje rubova na slici $$K = -\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slika 18. predstavlja primjer zamagljivanja rubova pomoću filtera.



Slika 22. Lijevo-prikaz slike prije konvolucije, desno-slika nakon konvolucije [22]

5.2. Sažimanje (engl. Pooling)

Drugi proces koji se koristi pri kreiranju konvolucijskih mreža je sažimanje. Ponekad je dobro sažeti značajke nakon procesa konvolucije. Na taj se način ostale operacije vrše nad sažetim značajkama. To daje robusnost modelu mreže kod variranja pozicija značajki na slici. Također, sažimanjem se smanjuje broj parametra koji se trebaju računati. Postoje više vrsta sažimanja, neke od najčešćih su [22]:

- Maksimalno sažimanje (engl. Maximum Pool)
- Minimalno sažimanje (engl. Minimum Pool)
- Uprosječeno sažimanje (engl. Average Pool)

Maksimalno sažimanje obuhvaćane elemente uspoređuje i odabire za rezultat element najveće vrijednosti. Suprotno od toga je minimalno sažimanje gdje se za rješenje odabire element

najmanje vrijednosti od svih obuhvaćenih elemenata. Uprosječeno sažimanje daje za rješenje prosjek svih vrijednosti selektiranih elemenata.

Postupak sažimanja objasnit će se na primjeru prethodne matrice A:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{bmatrix}, \tag{14}$$

Za provedbu procesa sažimanja treba se odrediti dimenzija djelovanja filtera sažimanja n_k, slično kako je u procesu konvolucije bila određena dimenzija konvolucijskog filtera. Neka je vrsta sažimanja za ovaj primjer maksimalna i neka je $n_p = 2$, što znači da će se sažimanje događati na 4 elementa matrice A. Korak s ima jednako značenje kao u procesu konvolucije i neka je za primjer s = 2.

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{bmatrix}$$

U prikazu iznad, svaki okvir različite boje u matrici A predstavlja proces sažimanja na obuhvaćenim elementima unutar okvira.

Zeleni okvir: Žuti okvir: Plavi okvir: Crveni okvir:

$$B_1 = \max_{i=1,2,5,6} a_i \qquad B_2 = \max_{i=3,4,7,8} a_i \qquad B_3 = \max_{i=9,10,13,14} a_i \qquad B_4 = \max_{i=11,12,15,16} a_i$$

Rezultirajuća matrica B je:

$$B = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}, \tag{15}$$

Općenita formula za računanje dimenzija matrice B je ista kao u procesu konvolucije.

5.3. Popunjavanje po rubovima (engl. Padding)

Ponekad nije dobro to što su rezultati konvolucije i sažimanja manjih dimenzija od početnih slika. Tada je potrebno koristi popunjavanje matrice slike po rubovima. Postupak se vrši dodavanjem piksela po rubovima slike (gornjim, donjim, lijevim i desnim) prije procesa konvolucije i sažimanja, tako da rezultirajuće matrice budu jednakih dimenzija kao i originalne

matrice slika. Jedna od više strategija popunjavanja dodaje piksele vrijednosti nula po rubovima (engl. zero padding), dok na primjer druga dodaje vrijednosti susjednih elemenata. [22]

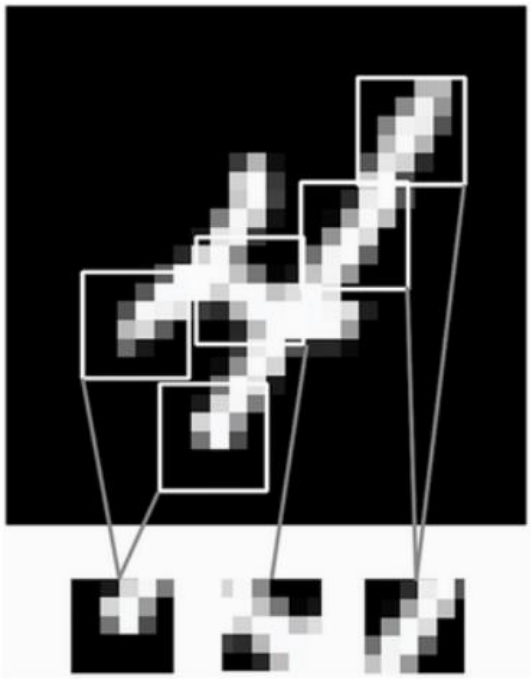
5.4. Konvolucijski slojevi

Konvolucijski slojevi na svojim ulazima primaju tenzore koji su za slike trodimenzionalni. Prve dvije dimenzije tenzora označavaju broj piksela po horizontalnoj i vertikalnoj duljini slike. Treća dimenzija je broj kanala slike, koji ima vrijednost 3 za RGB slike u boji (svaka boja ima jedan kanal), a vrijednost 1 za slike prikazane sivom ljestvicom. Nad ulaznim tenzorom vrši se proces konvolucije s većim brojem kernela (tipično 10 ili 16), dodaje se bias i koristi aktivacijska funkcija da bi se kreirao izlaz iz sloja. Izlaz je ekvivalent matrici B iz prethodnih primjera, ali s obzirom da se u procesu koristi više kernela, izlaz B je sada tenzor s 3 dimenzije. Više kernela koji vrše konvoluciju djeluju pojedinačno na ulazni tenzor tako da svaki kernel s tenzorom stvara rezultat koji se sprema u treću dimenziju izlaznog tenzora B. Stoga slijedi da je veličina treće dimenzije izlazne matrice B jednaka broju korištenih kernela u sloju. Izlazna matrica B često se naziva mapa uzoraka (engl. Feature map). Vrijednosti unutar kernela su ujedno i težine konvolucijskih slojeva. Inicijalno prije učenja, odabrane su slučajno, a zatim se učenjem značajki mijenjaju njihove vrijednosti. Neka je dimenzija kernela vrijednost n_K, a n_c broj korištenih kernela, onda slijedi da je broj težina, odnosno parametara mreže:

$$\text{Broj parametara mreže} = n_K{}^2 \cdot n_c \tag{16}$$

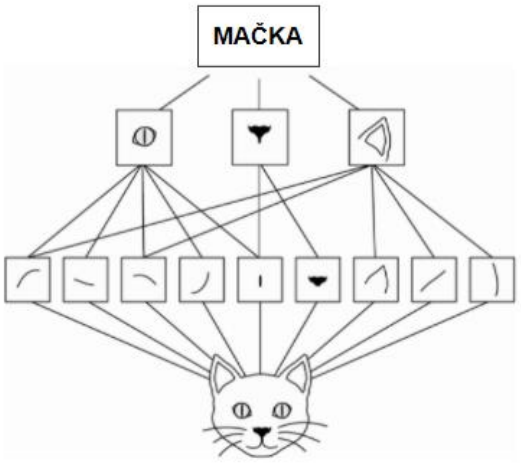
Iz jednadžbe se može vidjeti da broj parametara mreže ne ovisi o veličini ulazne slike.

Učenje značajki koje se ponavljaju u slikama može se učiti pomoću neuronskih mreža bez konvolucijskih slojeva, ali se zbog nedostatka konvolucijskih slojeva uče se globalne značajke. To znači da mreža mora promatrati značajke uzimajući u obzir sve piksele na slici. Suprotno tome, konvolucijske mreže omogućuju učenje lokalnih uzoraka. Lokalni uzorci su prikazani u kvadratima na primjeru pisanog broja četiri na slici 19. [13]



Slika 23. Lokalni uzorci slike [13]

Dvije su prednosti konvolucijskih mreža zbog pronalaženja lokalnih uzoraka. Jedna je ta što je takav način učenja neovisan o translataciji uzoraka. Na primjer uzorak mreža može učiti u donjem desnom dijelu slike, ali jednom naučeni uzorak, mreža će prepoznati gdje god se on nalazio na slici. Mreža bez konvolucijskih slojeva, na primjer mreža s potpuno povezanim slojevima morala bi isti uzorak naučiti ponovno jer se ne nalazi na istom mjestu na slici. Stoga je konvolucijskim mrežama potrebno manji broj slika u datasetu da nauči dobro prepoznati pojmove na slici. Druga prednost konvolucijskih mreža je učenje prostorne hijerarhije uzoraka. S obzirom da se mreža može sastojati od više konvolucijskih slojeva, prvi sloj može naučiti male lokalne uzorke kao što su rubovi. Drugi konvolucijski sloj će učiti uzorke koji se sastoje od uzoraka iz prvog sloja. Na taj se način stvara hijerarhija koja omogućava mreži da uči iznimno komplicirane koncepte. Primjer takve hijerarhije na primjeru mačke prikazan je na slici 24. Mreža najprije uči rubove koji čine složenije oblike poput ušiju, nosa i očiju. [13]



Slika 24. Prikaz učenja prostorne hijerarhije [13]

5.5. Pretreniranost modela

Neuronske mreže trebaju davati dobre rezultate na trening datasetovima kao i na stvarnim. Tako će jednom naučeni model diskriminatora morati davati dobra predviđanja i sa generiranim trajektorijama od strane generatora. Za davanje dobrih rezultata, mreža mora imati dovoljno kapaciteta da bi pravilno naučila trening podatke. Pod tim se podrazumijeva da je struktura dorasla zadatku, tj. da ima dovoljno težina kako bi se mogla aproksimirati funkcija problema. Kapacitet mreže može se unaprijediti dodavanjem slojeva u mrežu i dodatnih neurona u slojeve. Također, vrlo je bitno da pri učenju mreže ne dolazi do pretreniranosti mreže sa podacima za učenje. Pretrenirane mreže daju dobre rezultate samo na podacima za učenje, ali na stvarnim podacima performans mreže je loš. Problem pretreniranosti često se javlja u velikim mrežama s malim setom podataka za učenje jer dolazi do efekta učenja statističkog šuma u setu učenja što rezultira lošim rezultatom kod drugih podataka. Problemu pretreniranosti se može pristupiti na način da se smanji kapacitet mreže. To se može učiniti na dva načina. Prvi način je promjenom strukture mijenjanjem broja težina mreže dok je drugi način mijenjanje vrijednosti težina mreže. Postoji više regularizacijskih metoda za sprečavanje pretreniranosti modela mreže, a ovom radu se koristi funkcija izostavljanja (eng. Dropout).

5.6. Regularizacija mreže pomoću dropout funkcije

Za rješavanje problema pretreniranosti mreže, pokazalo se da je dobar pristup učenje više različitih mreža (s različitim parametrima) na istom setu podataka (engl. ensemble model averaging). Zatim kao konačno rješenje uzeti prosjek svih pojedinačnih predviđanja svake mreže. U praksi takav pristup je poprilično zahtjevan ako su modeli veliki jer korisnik mora

učiti više modela što je dugotrajan proces. Regularizacijska metoda Dropout oponaša djelovanje više neuronskih mreža sa različitim parametrima na samo jednom modelu mreže. To radi tako da se tijekom treniranja mreže izostavlja nekolicina slučajno odabranih neurona rezultirajući time promjenu strukture sloja. Keras biblioteka ima Dropout sloj i koristi se pri učenju modela diskriminatora. Sloj zahtjeva od korisnika da navede parametar koji predstavlja vjerojatnost da će ulaz u Dropout sloj biti izostavljen. [23]

6. PRIJEDLOG RJEŠENJA CGAN MREŽE ZA GENERIRANJE TRAJEKTORIJA

Nakon što rezultati mreže bez konvolucijskih slojeva nisu bili zadovoljavajući odlučilo se dodati konvolucijske slojeve za bolju kvalitetu slika. Dodavanjem konvolucijskih slojeva, nije se samo poboljšala kvaliteta generiranih slika nego se i ubrzao proces učenja mreže. U ovom poglavlju opisat će se modeli generatora, diskriminatora i cjelokupna struktura CGAN modela implementirana u Kerasu. Zatim će se opisati setovi za učenje prve mreže bez konvolucijskih slojeva i druge mreže s konvolucijskim slojevima.

6.1. Model diskriminatora

Funkcijom definiraj_diskriminator stvara se model diskriminatora. Bitno je definirati sve argumente funkcije, a to su dimenzije slike (zajedno s brojem kanala slike) i definirani broj klasa slika. Model diskriminatora sastoji se od raznih slojeva koji će biti objašnjeni u nastavku.

```
def definicija_diskriminatora(in_shape=(12,12,1), n_klasa=2):
```

6.1.1. Ulaz u mrežu (Input sloj)

Kao što je prethodno objašnjeno mreža ima ulaz za oznaku klase i ulaz za sliku. Keras biblioteka omogućuje funkciju Input kojom se gradi ulazni sloj. S obzirom da mreža mora biti uvjetna tako da se generiraju slike koje su odabrane klasom, mreža mora primiti oznaku klase koju će generirati. Oznaka klase slike mora biti pružena generatoru i također diskriminatoru. To se omogućava Input slojem nazvanim ul_label koji će u argumentu primiti cjelobrojnu vrijednost oznaka klase. [18]

```
ul_label = Input(shape=(1,))
```

6.1.2. Ugrađeni sloj (engl. Embedding layer)

Tehnika ugrađivanja (engl. embedding) je tehnika mapiranja varijabli različitog značaja u vektore kontinuiranih brojeva gdje su vektori sličnog značenja, smješteni bliže jedan drugom u prostoru. Tehnika ugrađivanja općenito u neuronskim mrežama koristi se za tri svrhe [19]:

1) Pronalaženje najbližih susjeda u prostoru ugrađivanja. To može biti korisno kada se trebaju pružiti preporuke sadržaja korisniku prema njegovim prijašnjim odabirima.
2) Za vizualizaciju koncepata i odnosa između kategorija (klasa).

3) Kao ulaz u mrežu za nadgledano učenje.

Ugrađeni sloj u ovom radu koristit će se za treći navedeni slučaj. Postoji više načina kako da se oznaka klase pruži mreži, ali prema preporuci iz [18] za stabilniju mrežu koristit će se Embedding sloj nakon kojeg slijedi potpuno povezani sloj (Dense sloj) s linearnom aktivacijskom funkcijom.

U Kerasu postoji ugrađena funkcija Embedding koja stvara ugrađeni sloj i prima pozitivne cjelobrojne vrijednosti (engl. integers), a pretvara ih u vektore bogatim informacijama koji su gusto raspoređeni. U ovom primjeru GAN mreže to bi značilo da ako mreža ima dvije klase, njihove oznake (engl. labels) će biti mapirane u reprezentaciju dvaju različitih vektora dimenzije 50.

```
ugrad_sloj = Embedding(n_classes, 50)(in_label)
```

6.1.3. Potpuno povezani sloj (Dense layer) i sloj preoblikovanja (Reshape layer)

Najčešći sloj koji se koristi u neuronskim mrežama je potpuno povezani sloj. Potpuno povezani sloj je sloj u kojem svaki neuron prima podatak od prethodnog sloja, tj. svaki neuron je povezan sa svim neuronima iz prethodnog sloja. Ovaj sloj najčešće se koristi za mijenjanje dimenzija vektora, ali isto tako može se koristiti i za operacije rotacije, skaliranja i translatiranja. Za sve operacije potpuno povezani sloj primjenjuje sljedeću formulu [25]:

Izlaz = aktivacijska funkcija(dot(ulaz, kernel) + bias)

, gdje je dot() operacija matričnog množenja, ulaz su ulazni podaci u sloj, kernel je tenzor ispunjen težinama sloja, a bias je vektor.

U primjeru funkcije diskriminatora koristi se s dovoljno neurona i linearnim aktivacijama da se vektor može proširiti, a zatim pomoću Reshape sloja preoblikovati na dimenziju slike. Reshape je sloj koji ulaz preoblikuje u željenu strukturu, u ovom slučaju preoblikuje na dimenzije slike. [26]

```
broj_cvorova = in_shape[0] * in_shape[1]
ugrad_sloj = Dense(broj_cvorova)(ugrad_sloj)
ugrad_sloj = Reshape((in_shape[0], in_shape[1], 1))(ugrad_sloj)
```

6.1.4. Spajanje slojeva (Concatenate)

Jednom kad je izlaz iz Reshape sloja proširen na dimenziju slike on se može spojiti sa slikom. To se može učiniti slojem Concatenate koji uzima na ulazu više tenzora i spaja ih u jedan tenzor. Novonastali tenzor će se sljedećem sloju činiti kao slika koja ima 2 kanala.

```
spojeni_sloj = Concatenate()([ul_slika, ugrad_sloj])
```

6.1.5. Conv2D sloj

Proces spajanja slike s njenom klasom je gotov. Sada takvu sliku treba provesti kroz konvolucijske slojeve kako bi se prepoznale značajke trajektorije na slici. Sloj koji će se koristiti je Conv2D i jedan je od konvolucijskih slojeva u Keras biblioteci. Pri definiranju sloja, mora se navesti broj filtera (kernela) koji će se koristiti u procesu konvolucije te njihove dimenzije. Također, moraju se navesti koraci filtera u horizontalnom i vertikalnom smjeru i popunjavanje rubova (padding). Prema preporuci iz [18] broj filtera je 128, korak u horizontalnom i vertikalnom smjeru je 2, a popunjavanje rubova je „same“ što znači da će se rubovi popuniti pikselima vrijednosti nula.

```
re = Conv2D(128, (3, 3), strides=(2, 2), padding='same')(spojeni_sloj)
re = LeakyReLU(alpha=0.2)(re)
re = Conv2D(128, (3, 3), strides=(2, 2), padding='same')(re)
re = LeakyReLU(alpha=0.2)(re)
```

Nakon prvog konvolucijskog sloja slijedi Leaky ReLu aktivacijska funkcija, a zatim se postupak ponavlja s još jednim konvolucijskim slojem i Leaky ReLu aktivacijskom funkcijom. Time se postiže prostorna hijerarhija spomenuta u potpoglavlju Konvolucijski slojevi.

Model diskriminatora daje predikciju o ispravnosti ulazne slike tako da na izlazu da broj vrijednosti između 0 i 1 koji označava vjerojatnost da je ta slika stvarna. Zato na izlazu diskriminatora stavlja Dense sloj s jednim neuronom i Sigmoid aktivacijskom funkcijom koja ima raspon vrijednosti od 0 do 1. Prije potpunog povezanog sloja koristio se Dropout layer s parametrom 0.4 kako bi se spriječila pretreniranost modela. Nakon toga se definirao model diskriminatora sa svojim ulazima i izlazima. Kao ulazi su postavljeni slika i labele slika, a izlaz je posljednji potpuno povezani sloj u mreži. Za optimizacijski algoritam odabran je algoritam Adam, a za funkciju gubitka je postavljena binarna unakrsna entropija koja je karakteristična za zadatke klasifikacije.

```
re = Flatten()(re)
# dropout
re = Dropout(0.4)(re)
```

```
# output
out_layer = Dense(1, activation='sigmoid')(re)
# definiraj model
model = Model([ul_slika, ul_label], out_layer)
# compile model
opt = Adam(learning_rate=0.0002, beta_1=0.5)
model.compile(loss='binary_crossentropy', optimizer = opt, metrics =
['accuracy'])
return model
```

6.2. Generator

S obzirom da se zahtjeva od generatora kreiranje trajektorije po uvjetu, njemu se također mora na ulazu pružiti oznaka klase koja se želi kreirati. To će se učiniti jednako kao i kod modela diskriminatora pozivajući funkciju Input( ), kreirajući Embedding i potpuno povezani (Dense) sloj.

```
ul_label = Input(shape=(1,))
# ulaz oznake
ul_label = Input(shape=(1,))
# embedding sloj
ugrad_sloj = Embedding(n_classes, 50)(ul_label)
n_nodes = 3*3
ugrad_sloj = Dense(n_nodes)(ugrad_sloj)
ugrad_sloj = Reshape((3, 3, 1))(ugrad_sloj)
```

U poglavlju arhitektura GAN mreže objašnjeno je da je za kreiranje slika modelu generatora potreban šum podataka i da je latentni prostor vektor u kojem se nalaze nasumično odabrane vrijednosti iz Gaussove normalne distribucije. Veličina latentnog prostora može se odabrati proizvoljno, a u ovom radu odabrana je dimenzija 100. Zadatak generatora je transformirati vektor iz latentnog prostora u sliku određene dimenzije. To se može učiniti tako da se definira potpuno povezani sloj koji ima dovoljno neurona da se njima može reprezentirati slika slabije rezolucije koja je u ovom radu dimenzije 3x3 (četvrtina dimenzije od konačne slike 12 x 12). Međutim želi se napraviti puno paralelnih verzija tih manjih slika. Kao što je u procesu prepoznavanja slika kod diskriminatora postupak koristiti mnogobrojne filtere koji će izdvojiti značajke, ovdje se želi postići potpuno obrnuti učinak. Tako se želi generirati puno reprezentacija manjih slika s različitim naučenim značajkama koje će se spojiti u konačnu izlaznu sliku. Zbog tog razloga se dodaje još jedan potpuno povezani sloj s aktivacijskom funkcijom koja ima dovoljno neurona za reprezentaciju puno manjih slika (128 u ovom radu). Te se slike manjih dimenzija moraju spojiti s prethodno pripremljenom oznakom slike koristeći Concatenate sloj.

```
gen = LeakyReLU(alpha=0.2)(gen)
gen = Reshape((3,3,128))(gen)
# spajanje slike s oznakom slike
merge = Concatenate()([gen, ugrad_sloj])
```

Kreiranim slikama manje rezolucije (3x3) želi se povećati rezolucija. To se može učiniti procesom obrnutim od procesa konvolucije zvanim dekonvolucija. U Keras biblioteci dekonvolucija se može postići funkcijom Conv2Dtranspose koja će dvostruko povećati rezoluciju slike. Rezolucija slike se može povećati i četverostruko povezujući dva dekonvolucijska sloja. Prema [18] pokazala se najbolja praksa koristiti aktivacijske funkcije LeakyReLU s parametrom nagiba 0,2. Nakon povećanja slika na veću rezoluciju, moraju se slike sa svojim značajkama povezati u jednu sliku iste dimenzije. Zbog tog se razloga za izlazni sloj modela koristi Conv2D sloj. On neće smanjiti rezoluciju slike jer je funkcija Conv2D definirana tako da ne mijenja dimenzije slike, ako se koristi korak iznosa 1 i popunjavanje po rubovima "same".

```
# uvećanje na 6x6
gen = Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same')(merge)
gen = LeakyReLU(alpha=0.2)(gen)
#uvećanje na 12x12
gen = Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same')(gen)
gen = LeakyReLU(alpha=0.2)(gen)
# output
out_sloj = Conv2D(1, (7, 7), activation='tanh', padding ='same')(gen)
# define model
model = Model([in_lat, ul_label], out_sloj)
return model
```

6.3. Gan model

Za razliku od modela diskriminatora koji se trenira zasebno, model generatora uči se preko modela diskriminatora. Kada model diskriminatora dobro prepoznaje lažne slike, težine generatora se značajnije mijenjaju. Time se postiže kontradiktorni odnos između dvaju diskriminatora i generatora koji se može u Kerasu postići na način da se kreira GAN model. On mora sadržavati prethodno definirane modele diskriminatora i generatora posložene tako da je izlaz modela generatora ulaz u model diskriminatora. Prije treniranja GAN modela bitno je onemogućiti učenje parametara diskriminatora. Time će treniranje GAN modela značiti treniranje generatora u odnosu na diskriminator. S obzirom da će model diskriminatora u GAN modelu biti nepromjenjiv, tada da će model generatora mijenjati svoje parametre prema tome koliko dobro diskriminator prepoznaje slike. Vrlo je bitno naglasiti da kad se na ovaj način uči

GAN model, diskriminatoru se mora prikazati su mu sve ulazne slike stvarne. Stvarne i generirane slike razlikuju se prema klasi koja im se pridijeli. Stvarne slike imaju klasu 1, dok klasu 0 imaju generirane slike. Bitno je razumjeti da te klase nisu klase koje reprezentiraju trajektorije, nego samo razlučuju stvarne slike od generiranih. Diskriminatoru se kod treniranja GAN modela uz sve ulazne slike pridjeljuje klasa 1. Model diskriminatora će zatim prepoznati da je ta slika lažna pa će se izračunati velika greška, što će rezultirati značajnim mijenjanjem parametara modela generatora kako bi se smanjila greška.

```
def definicija_gan(g_model, d_model):
    # parametrima diskriminatora onemogućiti učenje
    d_model.trainable = False
    # uzimanje šuma i oznaka klase za ulazak u generator
    gen_noise, gen_label = g_model.input
    # izlaz generatora
    gen_output = g_model.output
    gan_output = d_model([gen_output, gen_label])
    model = Model([gen_noise, gen_label], gan_output)
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer = opt)
    return model
```

Naučeni model generatora može samostalno generirati slike prema oznaci klase slike (klase trajektorije) koja se želi kreirati.

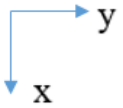
6.4. Set podataka za učenje

6.4.1. Podaci za učenje CGAN mreže bez konvolucijskih slojeva

Podaci za učenje prvobitne mreže preuzeli su se iz rada [27] na platformi Kaggle u obliku CSV datoteka. Trajektorije su nastale snimanjem trajektorija pomoću senzora u knjižnici Waldo na sveučilištu Western Michigan kako bi se koristile za aplikacije navigiranja. One su podijeljene u 6 klasa, a svaka od njih razlikuje se po koordinatama točaka početka (x1, y1) i kraja (x2, y2). Svaka snimljena trajektorija izbjegava prepreke u knjižnici, i to čine na različite načine. Neovisno o načinu, tj. putu kojim trajektorija prolazi do cilja, ako trajektorije počinju i završavaju u istoj koordinatnoj točki one pripadaju istoj klasi. Tablica 1. opisuje početne i završne koordinate svake trajektorije. Ukupno je 312 uzoraka u preuzetim podacima, a svaka klasa ima 52 uzorka. Koordinatni (x, y) sustav čini rešetka dimenzija 19 x 13. Trajektorija je u CSV formatu opisana tako što jedinice čine trajektoriju, dok su vrijednosti 0 prepreke. Izgled jedne trajektorije prikazane u znamenkama i iste trajektorije pretvorene u sliku prikazane su ispod tablice 1.

Tablica 1. Koordinate seta za učenje

Redni broj	Klasa	x1 (početak)	y1 (početak)	x2 (cilj)	y2 (cilj)
1.	0	0	0	14	7
2.	1	0	0	16	12
3.	2	2	5	16	12
4.	3	2	5	16	8
5.	4	2	5	18	0
6.	5	4	0	10	10

Napomena: orijentacija koordinatnog sustava je 

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0	0	0	0
10	0	0	0	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	0	0	0	0
12	0	0	0	0	0	0	0	0	1	0	0	0	0
13	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	1	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0

Slika 25. Trajektorija prikazana u znamenkama (lijevo), trajektorija pretvorena u sliku (desno)

6.4.2. Podaci za učenje CGAN mreže s konvolucijskim slojevima

Problem dodavanja konvolucijskih slojeva uzrokovan je zbog neparne rezolucije seta za učenje. Bilo je potrebno promijeniti rezoluciju slika koje su u setu podataka za učenje s obzirom da je

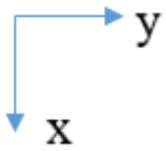
set za učenje 19x13 mreža nije mogla raditi ispravno s takvom rezolucijom. Dolazilo bi do problema kada bi generator kreirao vlastite slike jer u slučaju da se krenulo s generiranjem šuma slika rezolucije 4x5, procesom dekonvolucije ona bi se povećala na 8x10, tako da bi izlazeći iz drugog sloja ona bila 16x20. Ta slika rezolucije 16x20 trebala bi se pružiti modelu diskriminatora na klasifikaciju, no to nije moguće s obzirom da je set stvarnih slika drugačije rezolucije originalnih 19x13. Iz tog se razloga odlučilo napraviti novi set za učenje s podacima za slike rezolucije 12x12.

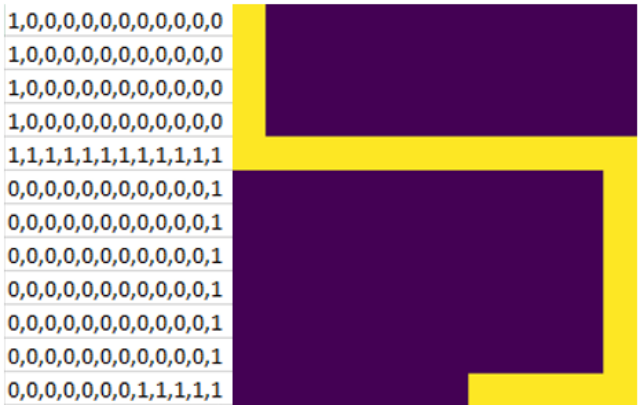
U novom setu za učenje napravljene su 4 klase koje se također razlikuju prema koordinatama početne i krajnje točke. U tablici 2. nalaze se njihove koordinate.

Tablica 2. Koordinate novog seta za učenje

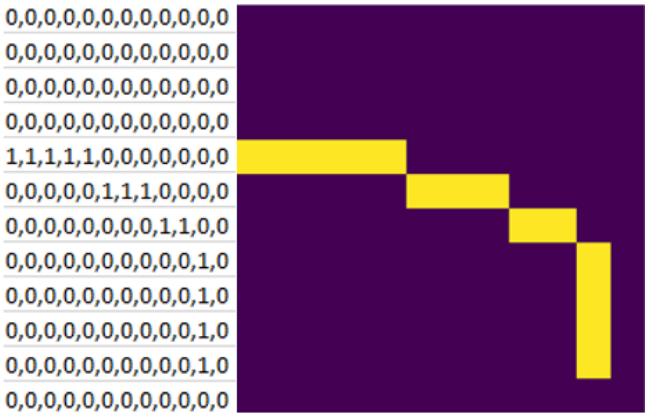
Redni broj	Klasa	x1 (početak)	y1 (početak)	x2 (cilj)	y2 (cilj)
1.	0	0	0	11	7
2.	1	4	0	10	10
3.	2	1	11	9	2
4.	3	0	11	11	0

Napomena: orijentacija koordinatnog sustava je ,a ishodište u gornjem lijevom kutu slike.





Slika 26. Trajektorija klase 0 prikazana u csv formatu (lijevo) i prikazana kao slika (desno)



Slika 27. Trajektorija klase 1 prikazana u csv formatu (lijevo) i prikazana kao slika (desno)

Slika 28. Trajektorija klase 2 prikazana u csv formatu (lijevo) i prikazana kao slika (desno)

Slika 29. Trajektorija klase 3 prikazana u csv formatu (lijevo) i prikazana kao slika (desno)

Slika 30. Trajektorije klase 3 s vizualiziranim preprekama

7. REZULTATI CGAN MREŽE

7.1. Rezultati CGAN mreže bez konvolucijskih slojeva

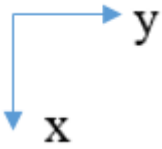
Zadatak naučenog generatora bio je izgenerirati trajektorije koje pripadaju određenoj klasi. Generator je uspješno generirao slike, ali se njihova kvaliteta razlikovala. Na svim slikama vidi se pokušaj generiranja trajektorije koja je počela u određenoj definiranoj točki i završila u drugoj definiranoj točki, međutim trajektorije su na nekim mjestima prekinute. Također, na dijelovima slika postoje zašumljeni pikseli koji kvare preglednost cijele slike. Primjeri izgeneriranih trajektorija, nalaze se u tablici 3.

Tablica 3. Primjeri trajektorija mreže bez konvolucijskih slojeva

KLASE	Trajektorija	(x1,y1)	(x2,y2)
0		(0,0)	(14,7)
1		(0,0)	(16,12)
2		(2,5)	(16,12)
3		(2,5)	(16,8)

4		(2,5)	(18,0)
5		(4,0)	(10,10)

Napomena: orijentacija koordinatnog sustava je, a ishodište u gornjem lijevom kutu slike.

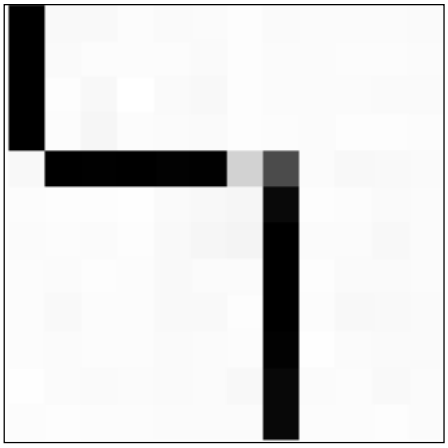
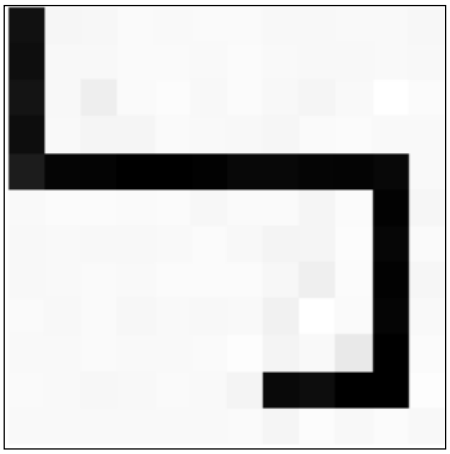


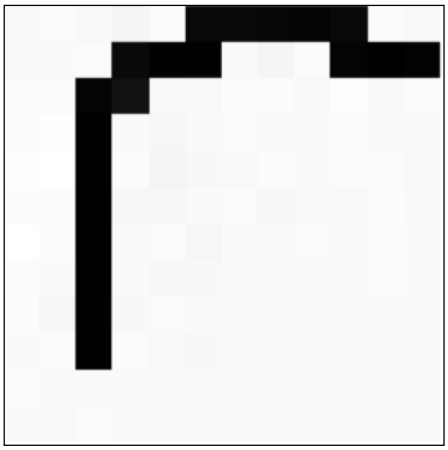
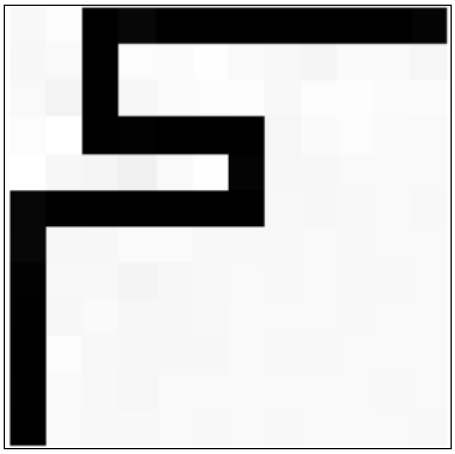
Trajektorije s ovakvim prekidima i šumom nisu zadovoljavajuće kvalitete da bi se mogle upotrijebiti za navigiranje robota. Zbog tog se razloga odlučilo uvesti konvolucijske slojeve u strukturu mreže jer su poznati po tome da olakšavaju učenje mreže kad su podaci za učenje slike.

7.2. Rezultati CGAN mreže s konvolucijskim slojevima

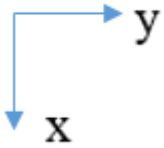
Nakon izmjene u CGAN mreži dodavanjem konvolucijskih slojeva u modele generatora i diskriminatora, mreža se ponovno učila, ali na novom setu podataka. Primjeri izgeneriranih trajektorija prikazani su po klasama u tablici 4.

Tablica 4. Primjeri dobrih trajektorija

KLASE	Trajektorija	(x1,y1)	(x2,y2)
0		(0,0)	(11,7)
1		(4,0)	(10,10)

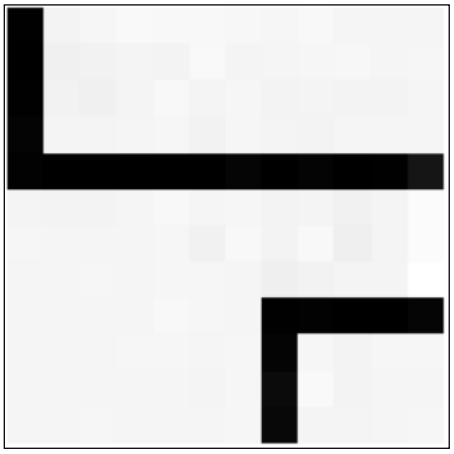
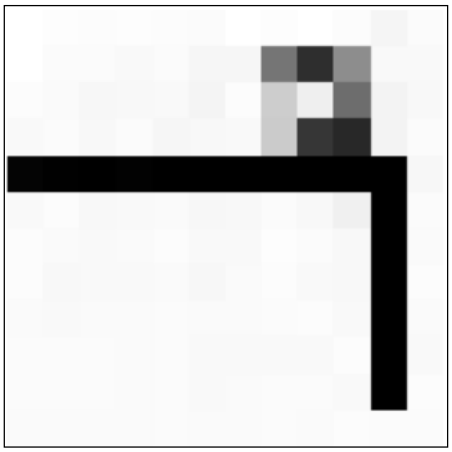
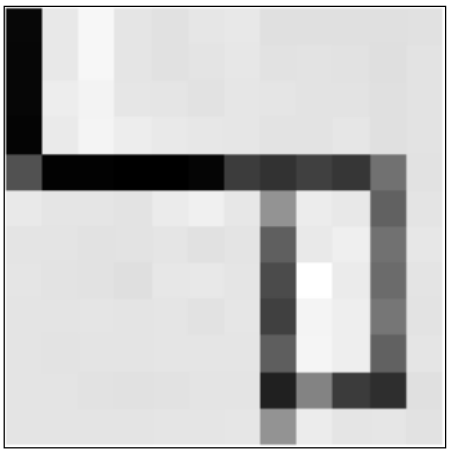
2		(1,11)	(9,2)
3		(0,11)	(11,0)

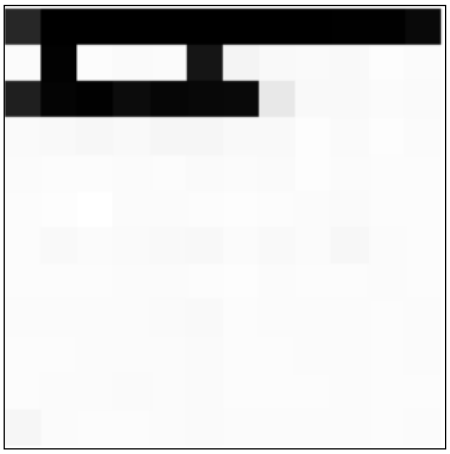
Napomena: orijentacija koordinatnog sustava je, a ishodište u gornjem lijevom kutu slike.



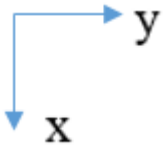
Dodavanje konvolucijskih slojeva u CGAN pokazalo se uspješnim jer je mreža generirala trajektorije bez šuma i prema odgovarajućim kriterijima koji ovise o pripadnosti klase. Pojedini pikseli nisu potpuno crne boje, no to se nije pokazalo problematično za navigiranje robota zbog toga što se i ti pikseli mogu proglasiti valjanima. Ipak, nisu svi rezultati bili zadovoljavajuće kvalitete jer su neke generirane trajektorije stvorile čudne artefakte na slici koji se nisu mogli povezati s trajektorijom. Pojedine trajektorije su imale efekt kao da izlaze iz slike što također nije ispravno generirana trajektorija. Primjeri loše generiranih trajektorija priloženi su u tablici 5. po klasama.

Tablica 5. Primjeri loših trajektorija

KLASE	Trajektorija	(x1,y1)	(x2,y2)
0		(0,0)	(11,7)
1		(4,0)	(10,10)
2		(1,11)	(9,2)

3		(0,11)	(11,0)

Napomena: orijentacija koordinatnog sustava je ,a ishodište u gornjem lijevom kutu slike.

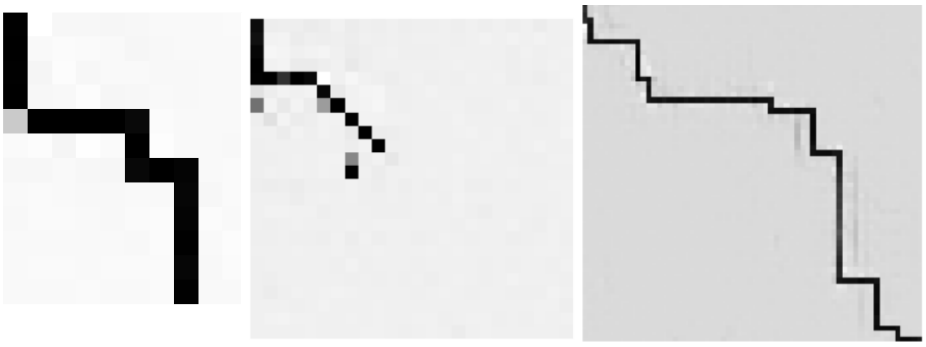


7.3. Vrijeme potrebno za učenje CGAN mreže

Koliko će trajati učenje mreže ovisi o više stvari kao što su vrsta i snaga procesora, zatim odabrani broj epoha i veličina batch size pri učenju mreže. Također utjecaj na vrijeme učenja ima i rezolucija i veličina seta za učenje mreže.

Prema preporuci iz [28] za učenje mreže koristio se batch size iznosa 2, što znači da su dvije slike prošle kroz mrežu prije nego li su se unutarnje parametri mreže ažurirali. Prednosti korištenja manjeg batch sizea su korištenje manje memorije pri računanju i brže učenje mreže. Koristi se manje memorije zbog toga što se mreža uči na manjem broju uzoraka, a učenje je brže zato što se češće ažuriraju parametri mreže. Nedostatak korištenja manjeg batch sizea je taj što procjena gradijenta manje precizna. Broj epoha pri učenju mreže bio je 1000.

Generirane su slike i u većim rezolucijama na jednakom broju primjeraka setova za učenje. Za učenje se koristio Googleov grafički procesor NVIDIA Tesla T4 s 2560 jezgri i 16 GB memorije čija je trenutna cijena 2259$ američkih dolara. Učenje mreže na slikama rezolucije 12x12 trajalo je u prosjeku sat vremena i 56 minuta, rezolucije 24x24 2 h i 14 min, a 64x64 2 h i 53 min.



Slika 31. Usporedba slika rezolucije 12x12 (lijevo), 24x24 (sredina), 64x64 (desno)

7.4. Navigiranje robota pomoću trajektorija u simulacijskom programu RoboDK

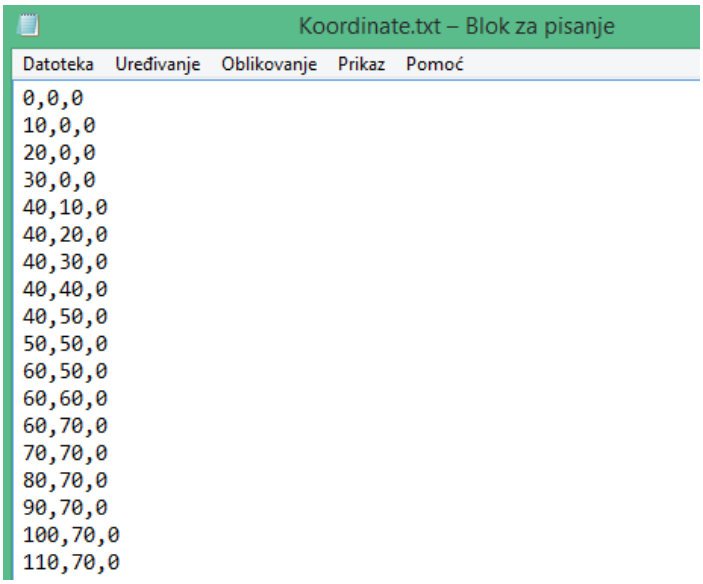
Prije nego li se generirane trajektorije primjene za vođenje robotskog modela, potrebno je svim slikama pridijeliti već spomenuti koordinatni sustav kako bi se mogle odrediti pozicije piksela trajektorije u odnosu na cijelu sliku. To se može učiniti na način da se svaki piksel promatra kao točka u koordinatnom sustavu koja je udaljena od susjednog piksela za iznos njegove dimenzije. S obzirom da je jedinica duljine koordinatnog sustava dimenzija jednog piksela, ukupni put trajektorije je umnožak broja piksela koji čine trajektoriju i dimenzija jednog piksela. Pridodavanje veće ili manje dimenzije pikselima ovisi o aplikaciji robota koji prati trajektoriju.

Normalizirana matrica slike sastoji se od vrijednosti intenziteta sive boje piksela u rasponu između 0 i 1. Vrijednosti piksela koji su 0 činit će potpuno bijele piksele na slici, nasuprot tome vrijednosti 1 će biti potpuno crni pikseli. Na prikazanim primjerima trajektorija moglo se uočiti da neki pikseli koji su dio trajektorije nisu potpuno crne boje nego su sivi, tj. imaju vrijednost između 0 i 1. Štoviše, neki pikseli koji se ljudskom oku čine potpuno crnim nemaju vrijednost 1 nego vrlo blizu jedinici. Zbog tog razloga mora se definirati jasna granica vrijednosti piksela koje će biti dovoljno velike da bi se piksel prihvatio kao dio trajektorije. Jedan od načina pridruživanja koordinata pikselima je koristeći ugniježđene for petlje. Pomoću petlji se prolazi kroz matricu slike i pridjeljuju se koordinate svim vrijednostima većim od 0.8 vrijednosti intenziteta sive boje.

```
X = model.predict([latent_points, labels])
# normaliziranje matrice slike [-1,1] to [0,1]
X = (X + 1) / 2.0
print(X[0,:12,:12,0])
putanja=X[0,:12,:12,0]
print(X.shape)
```

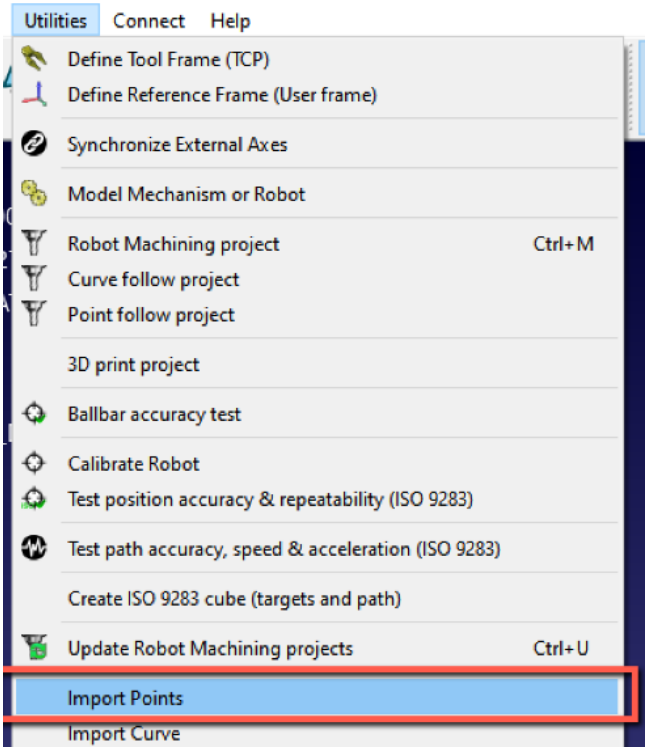
```
# ispis trajektorije
save_plot(X, 1)

x=12
y=12
A=[]
A=np.array(A)
B=[]
B=np.array(B)
for i in range(x):
    for j in range(y):
        if putanja[i,j]>0.8:
            A=np.append(A,i)
            B=np.append(B,j)
A=A*10
B=B*10
```

Za implementaciju na robotu u programu RoboDK potrebno je tim koordinatama pridijeliti i treću, koordinatu visine Z. Ona će biti nula u svim točkama, tako da gotova lista izgleda kao na slici 32.



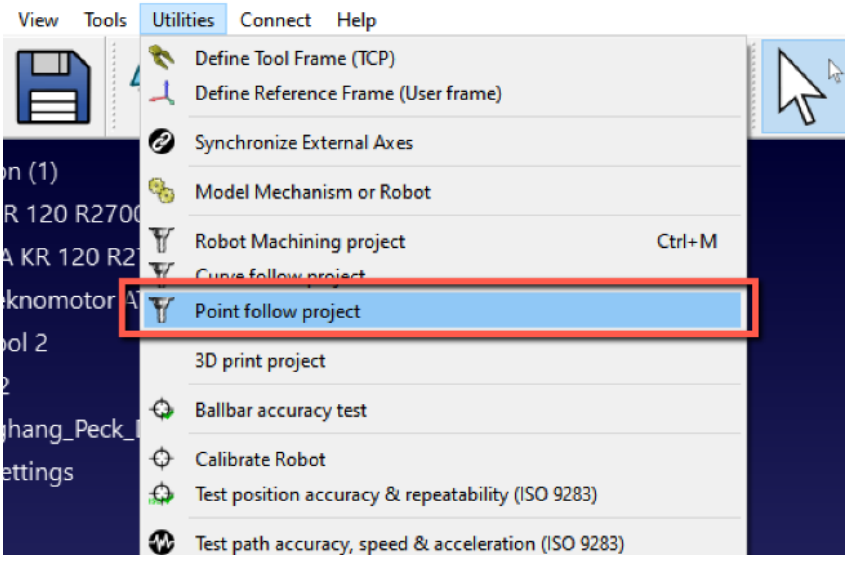
Slika 32. Lista koordinata trajektorije

Točke je u program vrlo jednostavno unijeti pomoću kartice Utilities i opcije Import points.

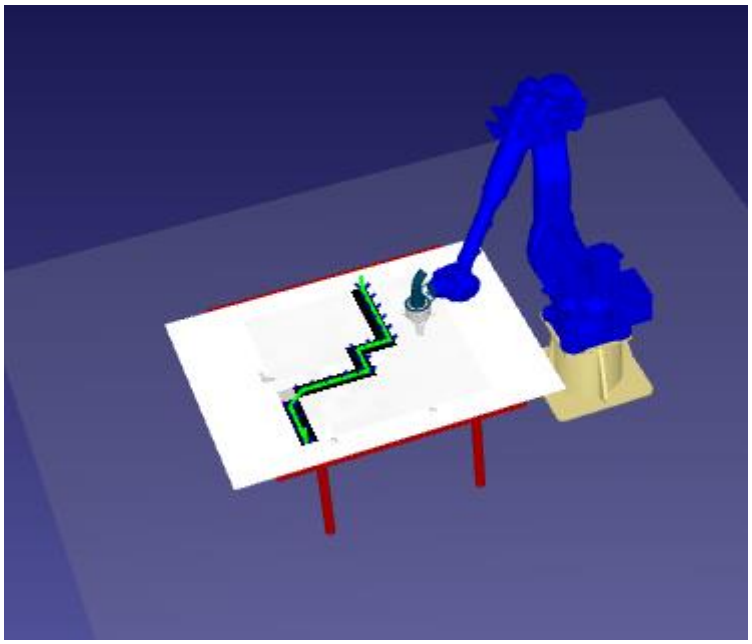


Slika 33. Unos točaka u program RoboDK

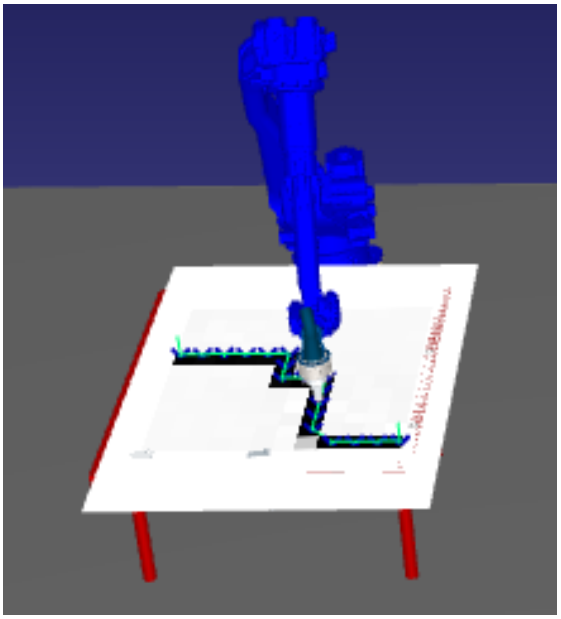
Zatim se odabere opcija Point follow project gdje se mogu odabrati prethodno priložene točke. Time program računa inverznu kinematiku potrebnu za praćenje točaka.



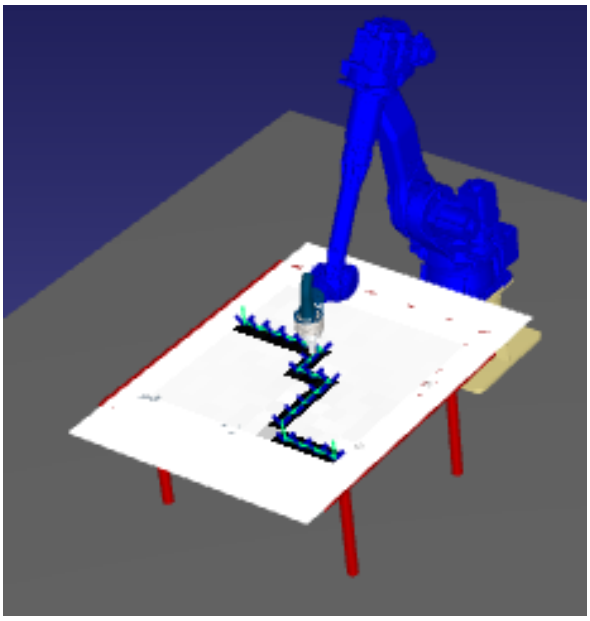
Slika 34. Stvaranje programa za slijeđenje točaka



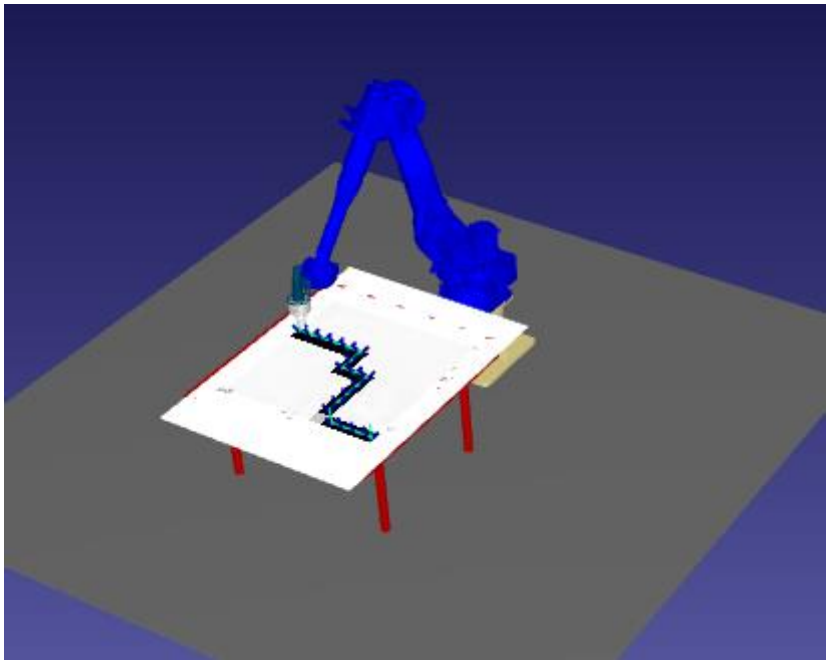
Slika 35. Robot u početnoj poziciji



Slika 36. Robot prati trajektoriju



Slika 37. Robot nastavlja slijediti trajektoriju



Slika 38. Robot je u završnoj točki trajektorije

8. ZAKLJUČAK

Generativne kontradiktorne mreže unatoč svojoj složenosti mogu imati svoju primjenu u rješavanju problema sinteze robotskih putanja. S obzirom na to da današnji trendovi pokazuju kako će u budućnosti autonomna vozila biti jedno od mogućih prijevoznih sredstava. GAN mreže mogle bi se koristiti kao generatori podataka za učenje algoritama strojnog učenja ili za generiranje trajektorija koje će biti dobre inicijalne pretpostavke algoritmima za planiranje optimalnih trajektorija. Konvolucijski slojevi u mreži pokazali su se kao odličan izbor jer su generirane trajektorije bile znatno bolje kvalitete. Za slučaj generiranja trajektorija robota najviše ima smisla koristiti uvjetnu GAN mrežu odnosno CGAN mrežu kako bi se moglo utjecati na oblik trajektorije. Za utilizaciju generiranih trajektorija na robotima dovoljna je i skromnija rezolucija slika (veličine 12x12) jer se one mogu skalirati proizvoljno. Međutim za neke potrebe gdje je potrebna bolja diskretizacija, slike bi trebale biti bolje rezolucije, ali je zato učenje nešto dugotrajnije. Za učenje mreže preporučuje se koristiti GPU ili TPU procesore jer značajno ubrzavaju računske operacije. Kreirana CGAN mreža kreirala je trajektorije zadovoljavajuće kvalitete, međutim neke generirane nisu bile dovoljno dobre da bi se mogle koristiti za daljnje primjene. Vjerujem kako je najveći razlog pojedinih loših trajektorija relativno mali broj podataka za učenje i da bi set podataka u desecima tisuća primjeraka davao znatno bolja rješenja.

LITERATURA

[1] Ćurković P, Čehulić L. Diversity maintenance for efficient robot path planning. Applied Sciences-Basel. 2020; 10(5):1721 doi. 10.3390/app10051721.

[2] Ćurković P, Jerbić B, Stipančić T. Co-Evolutionary Algorithm for Motion Planning of Two Industrial Robots with Overlapping Workspaces. International Journal of Advanced Robotic Systems. 2013; 10(1):55. doi. 10.5772/54991

[3] Ćurković P, Jerbić B, Stipančić T. Coordination of Robots With Overlapping Workspaces Based on Motion Co-Evolution. International journal of simulation modelling. 2013; 12(1):27–38. doi:10.2507/IJSIMM12(1)3.222

[4] Ćurković P, Jerbić B, Stipančić T. Swarm-based approach to path planning using Honey-bees mating algorithm and ART neural network. Solid state phenomena. 2009; 147–149:74–80.

[5] Ćurković P, Jerbić B, Stipančić T. Hybridization of adaptive genetic algorithm and ART 1 neural architecture for efficient path planning of a mobile robot. Transactions of FAMENA. 2008; 32(2):11–21.

[6] Langr J, Bok V. GANs in action: Deep learning with Generative Adversarial Networks. Manning Publications; 2019.

[7] Kalin J. Generative Adversial Networks cookbook. Pack Publishing; 2018.

[8] Goodfellow I, Pouget-Abadie J. Generative Adversial Networks. Université de Montréal. 2014 Jun; doi. 10.48550/ARXIV.1406.2661.
Dostupno online: https://arxiv.org/pdf/1406.2661.pdf
[Datum pristupa: 23.04.22]

[9] Castelvecchi D. Astronomers explore uses for AI-generated images. Nature. 2017. Feb:16-17.
Dostupno online: https://www.nature.com/articles/542016a
[Datum pristupa: 27.04.22]

[10] Nishida T, Barbi T. Trajectory prediction using Conditional Generative Adversarial Network. Atlantis Press. 2018 Jan; doi. 10.2991/anit-17.2018.33

[11] Mohammadi M, Al-Fuqaha A, Jun-Seok O. Path planning in support of smart mobility applications using Generative Adversarial Networks. In: IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). Halifax 2018; Halifax, Canada 30.07- 03.08 2018. Western Michigan University; 2018.

[12] Mirza M, Osindero S. Conditional Generative Adversarial Nets. Université de Montréal. 2014 Nov; doi. 10.48550/ARXIV.1411.1784.

[13] Chollet F. Deep learning with Python. 2nd ed. Manning Publications; 2021.

[14] Keras, https://keras.io/why_keras/
[Datum pristupa: 10.05.22]

[15] Wilson K. Exploring computer systems: The illustrated guide to understanding computer systems. Elluminet Press; 2020.

[16] Levinas M. GPU vs CPU: What are the key differences? Cherry Servers. 2020.
https://blog.cherryservers.com/gpu-vs-cpu-what-are-the-key-differences
[Datum pristupa: 19.05.22]

[17] Ibrahim M. What is a Tensor Processing Unit (TPU) and how does it work? Towards Data Science. 2021.

[18] Brownlee J. Generative Adversarial Networks with Python: Deep Learning Generative Models for Image Synthesis and Image Translation; 2019

[19] Keras, https://keras.io/api/models/model_training_apis/
[Datum pristupa: 21.05.22]

[20] Keras, https://keras.io/guides/sequential_model/
[Datum pristupa: 21.05.22]

[21] Keras, https://keras.io/guides/functional_api/
[Datum pristupa: 22.05.22]

[22] Michelucci U. Advanced applied Deep learning: Convolutional Neural Networks and object detection. Apress; 2019.

[23] Brownlee J. : Better Deep Learning, Train Faster, Reduce Overfitting and Make Better Predictions; 2019.

[24] Koehrsen W. : Neural Network Embeddings Explained, Toward data science,
https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

[Datum pristupa: 28.05.22.]

[25] Sharma P. Keras Dense Layer Explained for Beginners. MLK ; 2020 Oct.
Available from: https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/
[Datum pristupa: 30.05.22]

[26] Keras, https://keras.io/api/layers/reshaping_layers/reshape/

[27] Mohammadi M. Path planning using GAN and trajectory images. 2019. Kaggle
Available from: https://www.kaggle.com/datasets/mehdimka/path-planning
[Datum pristupa: 30.05.22]

[28] C. Luschi, D. Masters. Revisiting small batch training for Deep Neural Networks. Graphcore; 2019.
Available from:
https://www.graphcore.ai/posts/revisiting-small-batch-training-for-deep-neural-networks
[Datum pristupa: 04.06.22]

PRILOZI

I. CD-R disk

II. Programski kod u Python programskom jeziku

II. Programski kod u Python programskom jeziku

```
from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from numpy import array
from numpy import asarray
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Concatenate
import os
import pandas as pd
import numpy as np

# model dikriminatora
def definicija_diskriminatora(in_shape=(12,12,1), n_klasa=2):
    # ul_label je ulaz koji uzima vrijednost klase slike (njezinu oznaku)
koja se želi generirati
    ul_label = Input(shape=(1,))
    # stvaranje embedding sloja
    ugrad_sloj = Embedding(n_klasa, 50)(ul_label)
    # skalirati do dimenzija slika
    broj_cvorova = in_shape[0] * in_shape[1]
    ugrad_sloj = Dense(broj_cvorova)(ugrad_sloj)
    # dodaje se dodatni kanal
    ugrad_sloj = Reshape((in_shape[0], in_shape[1], 1))(ugrad_sloj)
    # ulaz za sliku
    ul_slika = Input(shape=in_shape)
    # spajanje dva sloja
    spojeni_sloj = Concatenate()([ul_slika, ugrad_sloj])
    # downsample
    re = Conv2D(128, (3, 3), strides=(2, 2), padding='same')(spojeni_sloj)
    re = LeakyReLU(alpha=0.2)(re)
    # downsample
    re = Conv2D(128, (3, 3), strides=(2, 2), padding='same')(re)
    re = LeakyReLU(alpha=0.2)(re)
    re = Flatten()(re)
    # dropout
    re = Dropout(0.4)(re)
    # output
    out_layer = Dense(1, activation='sigmoid')(re)
    # definiranje modela
    model = Model([ul_slika, ul_label], out_layer)
    # kompaliranje modela diskriminatora
    opt = Adam(learning_rate=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer = opt, metrics =
['accuracy'])
    return model
```

```
# define the standalone generator model
def definicija_generatora(latent_dim=100, n_classes=2):
    # ulaz oznake
    ul_label = Input(shape=(1,))
    # embedding sloj
    ugrad_sloj = Embedding(n_classes, 50)(ul_label)
    n_nodes = 3*3
    ugrad_sloj = Dense(n_nodes)(ugrad_sloj)
    # reshape da se doda dodatni kanal slike
    ugrad_sloj = Reshape((3, 3, 1))(ugrad_sloj)
    # image generator input
    in_lat = Input(shape=(latent_dim,))
    n_nodes = 3*3*128
    gen = Dense(n_nodes)(in_lat)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Reshape((3,3,128))(gen)
    # spajanje slike s oznakom slike
    merge = Concatenate()([gen, ugrad_sloj])
    # uvećanje na 6x6
    gen = Conv2DTranspose(128, (4, 4), strides=(2, 2),
padding='same')(merge)
    gen = LeakyReLU(alpha=0.2)(gen)
    #uvećanje na 12x12
    gen = Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same')(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    # output
    out_sloj = Conv2D(1, (7, 7), activation='tanh', padding ='same')(gen)
    # definaranje modela generatora
    model = Model([in_lat, ul_label], out_sloj)
    return model
# definiranje GAN (CGAN) modela
def definicija_gan(g_model, d_model):
    # parametrima diskriminatora onemogućiti učenje
    d_model.trainable = False
    # uzimanje šuma i oznaka klase za ulazak u generator
    gen_noise, gen_label = g_model.input
    # izlaz generatora
    gen_output = g_model.output
    # spajanje izlaza iz generatora i ulaza diskriminatora
    gan_output = d_model([gen_output, gen_label])
    # definiranje gan modela da uzima šum i oznaku klase i daje
klasifikaciju na svom izlazu
    model = Model([gen_noise, gen_label], gan_output)
    # kompajliranje modela
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer = opt)
    return model
data_path = r'C:\Users\Tin\PycharmProjects\moj_projekt\dataset'



def ucitavanje_podataka(path=data_path):
    files = os.listdir(path)
    data = []
    labels = []
    for fn in files:
        ffn = os.path.join(path, fn)
        df = pd.read_csv(ffn, index_col=None, header=None)
        data.append(df.values)
```

```
        label = int(fn[0:2])
        labels.append(label)
    data = np.array(data)
    return [data, labels]







def ucitavanje_stvarnih_podataka():
    # učitavanje dataseta
    (trainX, trainy) = ucitavanje_podataka()
    X = expand_dims(trainX, axis=-1)
    # convert from ints to floats
    X = X.astype('float32')
    # skaliranje [0,255] u [-1,1]

    return [X, trainy]
[X,trainy]= ucitavanje_stvarnih_podataka()


dataset=[X,trainy]
def generiranje_stvarnih_uzoraka(dataset, n_samples):
    # odvajanje slika i oznaka
    images, labels = dataset
    # biranje random instance
    ix = randint(0, images.shape[0], n_samples)
    # selektiranje slika i labela
    slike=[]
    etikete=[]
    for k in range (0,n_samples):
         slike.append(images[ix[k]])
         etikete.append(labels[ix[k]])
    # generiranje oznaka klase
    slike=asarray(slike)
    etikete=asarray(etikete)
    y = ones((n_samples, 1))
    return [slike, etikete], y
# generiranje latentnog prostora
def generiranje_latentnog_vektora(latent_dim, n_samples, n_classes=2):
    # geeriranje točaka u latentnom prostoru
    x_input = randn(latent_dim * n_samples)
    # restrukturiranje u batch ulaznih podataka za mrežu
    z_input = x_input.reshape(n_samples, latent_dim)
    # generiranje oznaka
    labels = randint(0, n_classes, n_samples)
    return [z_input, labels]

def generiranje_laznih_uzoraka(generator, latent_dim, n_samples):
    # generiranje točaka u latentnom prostoru
    z_input, labels_input = generiranje_latentnog_vektora(latent_dim,
n_samples)
    images = generator.predict([z_input, labels_input])
    # stvaranja oznaka
    y = zeros((n_samples, 1))
    return [images, labels_input], y
# učenje generatora i diskriminatora
def train(g_model, d_model, gan_model,dataset,latent_dim, n_epochs=1000,
```

```
n_batch=2):
    bat_per_epo = int(dataset[0].shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    # broj epoha
    for i in range(n_epochs):
        # broj batcha po epohi
        for j in range(bat_per_epo):
            # odabir slučajno selektiranih stvarnih uzoraka
            [X_real, labels_real], y_real =
generiranje_stvarnih_uzoraka(dataset, half_batch)
            # ažuriranje parametara diskriminatora
            d_loss1, _ = d_model.train_on_batch([X_real, labels_real],
y_real)
            # generiranje lažnih uzoraka
            [X_fake, labels], y_fake = generiranje_laznih_uzoraka(g_model,
latent_dim, half_batch)
            # ažuriranje parametara diskriminatora
            d_loss2, _ = d_model.train_on_batch([X_fake, labels], y_fake)
            # pripremanje latentnog prostora za ulaz u generator
            [z_input, labels_input] =
generiranje_latentnog_vektora(latent_dim, n_batch)
            # kreiranje obrnutih labela za lažne uzorke
            y_gan = ones((n_batch, 1))
            # ažuriranje generatora na temelju pogreške diskriminatora
            g_loss = gan_model.train_on_batch([z_input, labels_input],
y_gan)

            print( '> %d, %d/%d, d1 =%.3f, d2=% .3f g = %.3f' %
                (i + 1, j + 1, bat per epo, d loss1, d loss2, g loss))
    # spremanje modela generatora
    g_model.save('cgan_generator2.h5')



latent_dim = 100
# stvaranje modela diskriminatora
d_model = definicija_diskriminatora()
# stvaranje modela generatora
g_model = definicija_generatora(latent_dim)
# stvaranje GAN modela
gan_model = definicija_gan(g_model, d_model)
# ucitavanje podataka
dataset= ucitavanje_podataka(data_path)
# train model
train(g_model, d_model, gan_model, dataset, latent_dim)
predictions = model.predict(X)
loss, accuracy = model.evaluate(X, y)
```

```
from numpy import asarray
from numpy.random import randn
from numpy.random import randint
from tensorflow.keras.models import load_model
from matplotlib import pyplot
import numpy as np
from matplotlib import pyplot as plt
from pylab import *

import pandas as pd
```

```
from tensorflow.keras.layers import Input, Dense, Reshape, Flatten, Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam

from tensorflow.keras import utils
from sklearn.preprocessing import LabelEncoder

import os
import time
from pylab import *

# generiranje točaka za input u generator
def generate_latent_points(latent_dim, n_samples, n_classes=4):
    # generiranje točaka u latentnom prostoru
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    z_input = x_input.reshape(n_samples, latent_dim)
    # generiranje oznaka slika
    labels = randint(0, n_classes, n_samples)
    return [z_input, labels]
# ispis slika
def save_plot(examples, n):

    for i in range(n * n):

        pyplot.subplot(n, n, 1 + i)
        pyplot.axis('off')
        pyplot.imshow(examples[i, :, :, 0], cmap='gray_r')
        pyplot.show()
    # učitaj model
model = load_model('cgan_generator.h5')
# generiraj slike

latent_points, labels = generate_latent_points(100, 100)
labels=3*ones(100)
print(labels)
# generate images
X = model.predict([latent_points, labels])
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
save_plot(X, 1)
```