Generativna suparnička mreža za izradu umjetničkih slika

Mlakić, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Marko Mlakić

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor: Student:

Prof. dr. sc. Tomislav Stipančić, dipl. ing. Marko Mlakić

Zagreb, 2023.

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

Zahvaljujem se svom mentoru, izv. prof. dr. sc. Tomislavu Stipančiću, koji me je sa svojim predavanjima i savjetima tijekom studija motivirao i pomogao za izradu ovog rada.

Posebno se zahvaljujem obitelji, prijateljima i djevojci Valentini na ohrabrenju, podršci i strpljenju tijekom cijelog studija.

Ovaj rad posvećujem svojoj baki Katici i didi Anti.

Marko Mlakić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 23 - 6 / 1**	
Ur.broj: **15 - 23 -**	

DIPLOMSKI ZADATAK

Student: **Marko Mlakić** JMBAG: 0082056722

Naslov rada na hrvatskom jeziku: **Generativna suparnička mreža za izradu umjetničkih slika**

Naslov rada na engleskom jeziku: **A generative adversarial network for artistic images creation**

Opis zadatka:

Metode umjetne inteligencije omogućavaju automatsko kreiranje multimedijalnih sadržaja koji mogu biti različitih oblika i formi, uključujući tekst, slike, zvuk i video. U tu je svrhu na zadanom skupu podataka moguće trenirati posebne arhitekture neuronskih mreža koje su prikladne za obavljane tih zadaća.
U radu je potrebno koristeći metodologiju generativnih suparničkih mreža (GAN) razviti računalni model koji omogućuje izradu umjetničkih slika temeljem danih ulaznih informacija. Razvijeni model potrebno je testirati koristeći prikladnu evaluacijsku metriku. Model treba uzimati u obzir stilove više različitih umjetnika. Prilikom generiranja izlaznog rješenja model mora poštivati ulazne informacije kao što su dominantni stil te ulazna slika. Programska aplikacija treba biti integrirana u sklopu web stranice za koju je potrebno razviti te implementirati odgovarajuće komunikacijsko sučelje za učitavanje slika te za interakciju s korisnikom.
U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
28. rujna 2023.	30. studenoga 2023.	4. – 8 . prosinca 2023.
Zadatak zadao:		Predsjednik Povjerenstva:
Izv. prof. dr. sc. Tomislav Stipančić		Prof. dr. sc. Ivica Garašić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS KORIŠTENIH KRATICA

A

- AdaIN - *Adaptive Instance Normalization*
- ADAM - *Adaptive Moment Estimation*

B

- BCE - *Binary Cross Entropy*
- BigGAN - *Big Generative Adversarial Network*
- bin - *Binary*
- BM - *Boltzmann Machine*

C

- CPU - *Central Processing Unit*
- CSS - *Cascading Style Sheets*
- CycleGAN - *Cycle-consistent Generative Adversarial Network*

D

- DBN - *Deep Belief Network*
- DCGAN - *Deep Convolutional Generative Adversarial Network*

F

- FID - *Fréchet Inception Distance*

G

- GAN - *Generative Adversarial Network*
- GPU - *Graphics Processing Unit*

H

- HDF5 - *Hierarchical Data Format, Version 5*
- HTML - *HyperText Markup Language*

I

- IS - *Inception Score*

J

- JSON - *JavaScript Object Notation*

K

- KL - *Kullback-Leiblerova (divergencija)*

M

- MMD - *Maximum Mean Discrepancy*

P

-ProGAN - *Progressively Growing Generative Adversarial Network*

R

- RBM - *Restriced Boltzmann Machine*
- ReLU - *Rectified Linear Unit*
- RGB - *Red, Green, Blue*
- RMSProp - *Root Mean Squared Propagation*

S

- SGD - *Stohastic Gradient Descent*

T

- TFDS - *TensorFlow Dataset*
- TPU - *Tensor Processing Unit*

V

-VAE - *Variational Autoencoder*

W

- WGAN - *Wasserstein Generative Adversarial Network*

POPIS OZNAKA

Oznaka	Jedinica	Opis
$D(x)$	-	Vjerojatnost diskriminatora da su x podatci pravilni
D_{KL}	-	Kullback-Leiblerova divergencija
E_G	-	Očekivana vrijednost za skup generiranih slika
$FID(R, G)$	-	Fréchetova udaljenost između stvarnih i generiranih podataka
$G(z)$	-	Generirani podatci
$IS(G)$	-	Ocjena po „Inception“ modelu za generator
$P(i)$	-	Vjerojatnost događaja i za P distribuciju
P_g	-	Distribucija generiranih podataka
P_r	-	Distribucija stvarnih podataka
$p(y\|x)$	-	Uvjetna distribucija s obzirom na sliku x
$p(y)$	-	Marginalna distribucija s obzirom na sve generirane slike
$p_z(z)$	-	Distribucija šuma
$Q(i)$	-	Vjerojatnost događaja i za Q distribuciju
W	-	Wassersteinova udaljenost
$\mathbf{z}$	-	Slučajni vektor
μ_g	-	Srednja vrijednost generiranih podataka
μ_r	-	Srednja vrijednost stvarnih podataka
$\sum_g$	-	Kovarijanca generiranih podataka
$\sum_r$	-	Kovarijanca stvarnih podataka

SAŽETAK

Rad se bavi razvojem računalnog modela zasnovanog na metodologiji generativnih suparničkih mreža (poznato i kao GAN) za stvaranje umjetničkih slika. Model je treniran na određenom skupu podataka te je uspješan u generiranju slika stilova različitih umjetnika. U radu su detaljno opisani tehnički aspekti modela, uključujući arhitektura generatora i diskriminatora. Također je opisana strategija učenja GAN-a te izazovi (kao i rješenja) koji su se događali tijekom procesa. Dodatno je model testiran s pomoću maksimalne prosječne razlike, koja je služila kao evaluacijska metrika radi provjere učinkovitosti i preciznosti generativnog modela. Dio projekta uključuje i razvoj „web“ stranice koja služi kao platforma za interakciju s modelom. „Web“ stranica sadrži korisničko sučelje za učitavanje slika i pružanje povratnih informacija o generiranim slikama.

Ključne riječi: generativna suparnička mreža, generator, diskriminator, strojno učenje

SUMMARY

The paper focuses on developing a computational model based on the methodology of generative adversarial networks (also known as GAN) for creating artistic paintings. The model was trained on a specific dataset and is successful in generating images in the style of various artists. The paper details the technical aspects of the model, including the architecture of the generator and discriminator. It also describes the GAN learning strategy for the assigned task, as well as the challenges and solutions encountered during the process. Additionally, the model was tested using the maximum mean discrepancy, which served as an evaluation metric to check the effectiveness and precision of the generative model. Part of the project also includes the development of a website that serves as a platform for interacting with the model. The website contains a user interface for uploading images and providing feedback on the generated images.

Key words: generative adversarial network, generator, discriminator, machine learning

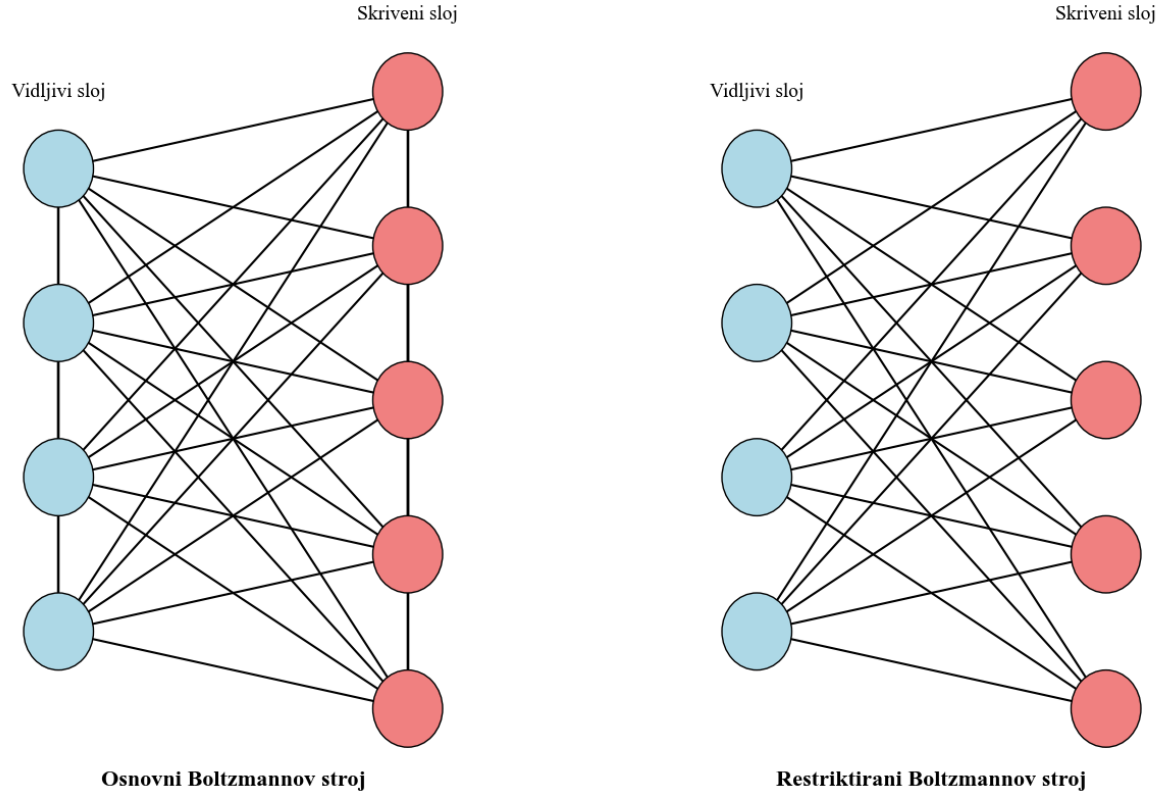
1. UVOD

U ovom radu razmatra se primjena generativnih suparničkih mreža (engl. ***G**enerative **A**dversarial **N**etworks*, GAN) za izradu umjetničkih slika. GAN spada pod duboko učenje i svoju primjenu najčešće nalazi u kreiranju multimedijalnog sadržaja. Kroz uvodni dio se gleda kroz povijest generativne umjetne inteligencije: rani početci do današnje svakodnevice. Daljnje se gleda u teorijsku dubinu GAN-a, s objašnjenjem osnovnih komponenti i načina funkcioniranja, da bi se nastavilo s dodatnim primjerima primjene kao i uobičajenim podvrstama GAN-a koji su u upotrebi. U drugom poglavlju se navodi tehnički okvir i alati koji su bili korišteni za izradu zadatka s detaljnijim obrazloženjem odabranog skupa podataka, struktura GAN modela kao i strategije učenja i optimizacije za uspješno treniranje modela. Zatim u trećem poglavlju se objašnjava način implementacije GAN modela: od izgradnje do ugradnje na napravljenoj „web“ stranici. Poseban fokus u ovom dijelu je prebačen na sve izazove koji su se pojavljivali tijekom procesa treniranja kao i pristupi u rješavanju. Na kraju se generirane slike prikazuju vizualno, evaluiraju te se rezultati analiziraju radi provjere uspješnosti rada GAN modela.

1.1. Povijest generativne umjetne inteligencije

Inicijalni GAN model je proizašao iz rada Iana Goodfellowa koji je 2014. predstavio strukturu dubokog učenja sačinjena od dvije neuronske mreže u suparničkom odnosu – generator i diskriminator [1]. Međutim, generativni modeli vuku dublje korijenje u povijesti umjetne inteligencije i strojnog učenja. Rani predstavnik generativnog modela je Boltzmannov stroj (engl. ***B**oltzmann **M**achine*, BM): tip generativnog modela koji uči vjerojatnosne distribucije podataka. Koristi se za otkrivanje obrazaca u podatcima kroz kombinaciju vidljivih i skrivenih slojeva koji su povezani kako unutar tako i između slojeva [2-4]. No, upravo zbog te unutarslojne povezanosti, Boltzmannovi strojevi imaju veći rizik od prenaučenosti (engl. *overfitting*) i samim time smanjiti sposobnost generiranja novih podataka [2]. Zbog ovoga predloženo je pojednostavljenje na osnovni model Boltzmannovog stroja – restriktirani Boltzmannov stroj (engl. ***R**estricted **B**oltzmann **M**achine*, RBM) nema veze unutar istog vidljivog ili skrivenog sloja čime se smanjuje složenost i omogućuje jednostavnije treniranje. Prvi prikaz ovog rada [Slika 1] predstavlja strukture osnovnog i restriktivnog Boltzmannovog

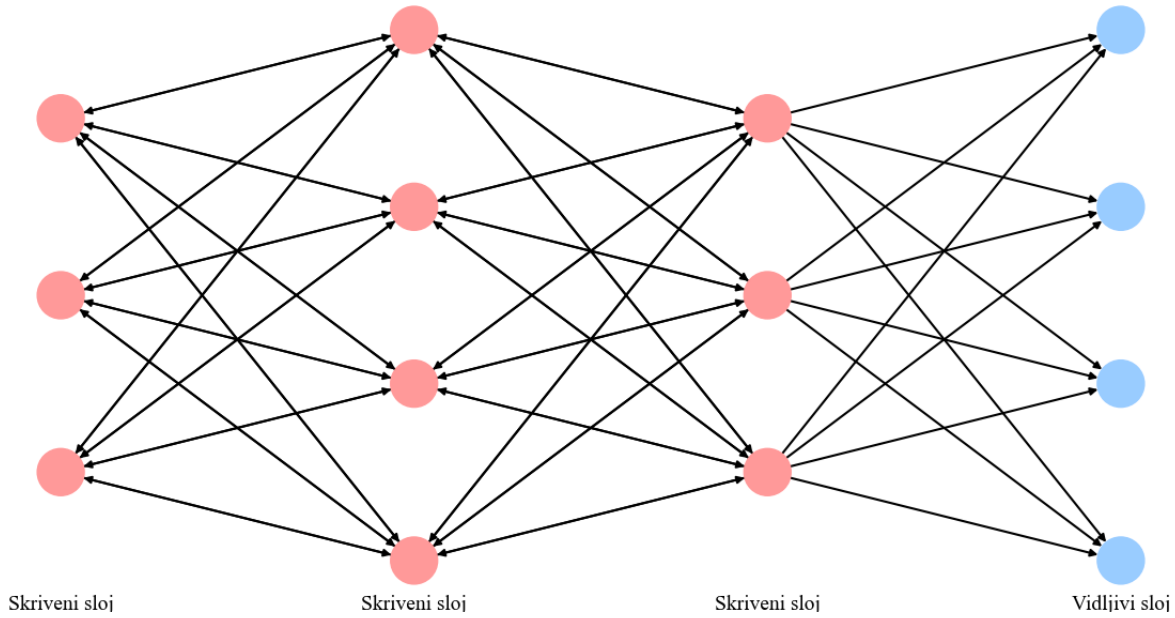
stroja, gdje je jasno uočljiva razlika u shemi spajanja među čvorove vidljivih (plave) i skrivenih (crvene) slojeva.



Slika 1. Osnovni i restriktirani Boltzmannov stroj

Premda je sa slike vidljiv relativni pad složenosti kod RBM strukture naspram BM-a, vrijedno je napomenuti kako i ova struktura podliježe svojim izazovima. Primjerice u slučaju šuma kod težina modela primjetni su značajni padovi u performansama modela [3]. Ipak, spajanjem više RBM-ova se dobije tzv. duboka mreža vjerovanja (engl. ***D**eep **B**elief **N**etwork,* DBN) koja omogućuje hijerarhijsko učenje, odnosno, omogućuje poboljšanje performansa modela. Ovaj sofisticiran pristup generativnom modeliranju je predstavljen 2006. i jedan je od prvih primjera uspješnog dubokog učenja [4]. Budući da ove mreže su sačinjene od više RBM mreža ne postoje međuslojne veze već su svi čvorovi sloja spojeni sa svakim čvorom susjednog sloja. Duboke mreže vjerovanja se učinkovito treniraju kombinacijom usmjerenih i neusmjerenih veza. Usmjerene veze stvaraju jasnu hijerarhiju unutar mreže, stvarajući uzročne odnose između varijabli. Neusmjerene veze, koje omogućuju slobodan protok informacija, pojednostavljuju treniranje. Ovo omogućuje provođenje generativnog učenja preko višeslojne strukture, uz otkrivanje složenih, dubokih uzoraka u podatcima (koji bi često mogli biti neprepoznatljivi kod

drugih modela). Također, DBN-ovi koriste tehniku poznatu kao Gibbsovo uzorkovanje (engl. *Gibbs sampling*) na svojim najvišim slojevima, što omogućuje efikasnije generiranje uzoraka i bolje razumijevanje distribucije podataka [4]. Naredna slika [Slika 2] pokazuje strukturu duboke mreže vjerovanja.



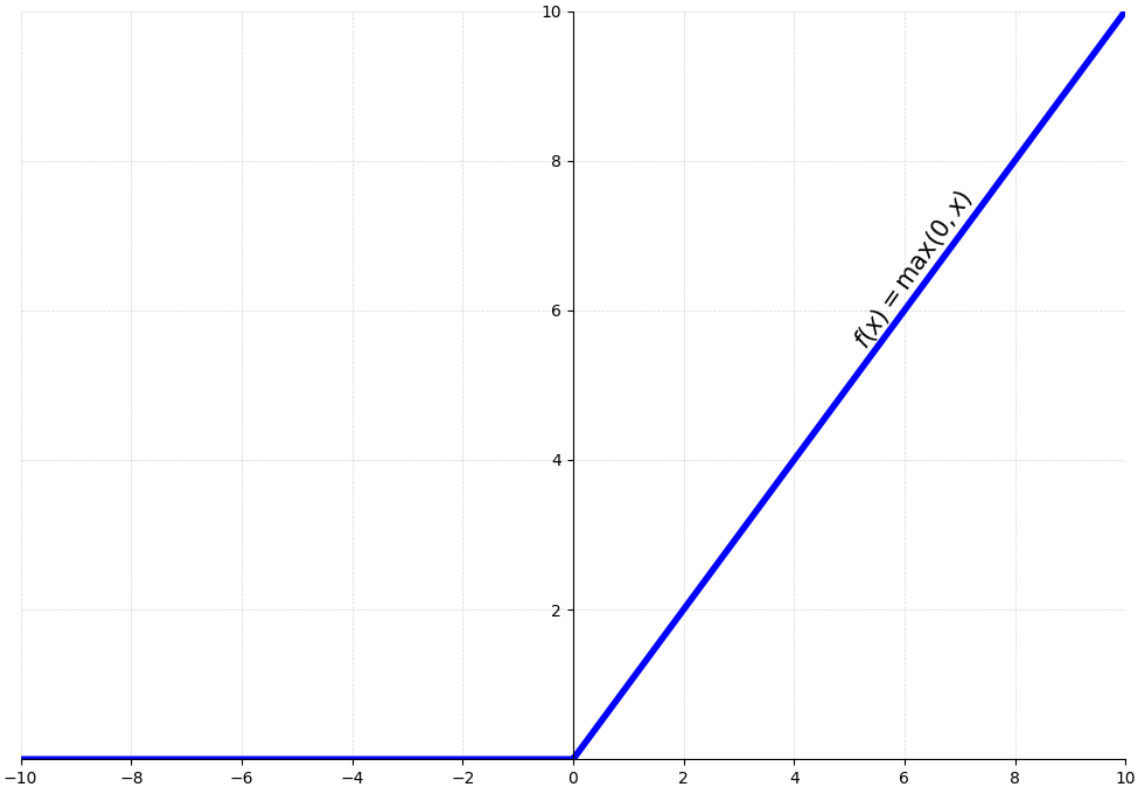
Slika 2. Arhitektura duboke mreže vjerovanja

Iako su duboke mreže vjerovanja bile za to vrijeme jedinstvene u svom pristupu, danas se rijetko koriste uslijed napretka tehnika optimizacije radi kojih je moguće izravno trenirati duboke neuronske mreže. To je postignuto s uvođenjem ReLU aktivacijske funkcije (engl. ***R**ectified **L**inear **U**nit*) kao standard za sve suvremene arhitekture dubokih neuronskih mreža [4]. Za razliku od sigmoidalne funkcije ili tangensa hiperbolnog, ReLU se aktivira samo kada je potrebno. Drugim riječima, neuroni se aktiviraju samo kada primaju pozitivne ulaze. Matematički se to može opisati sljedećom formulom:

$$f(x) = \begin{cases} 0, & \text{ako je } x < 0 \\ x, & \text{ako je } x \geq 0 \end{cases} \tag{1.1.1}$$

Ova karakteristika omogućava bržu konvergenciju tijekom treniranja i rješavanje problema tzv. nestajućeg gradijenta (engl. *vanishing gradient*): gradijenti postanu vrlo mali u početnim

slojevima mreže čime se težine ažuriraju vrlo sporo, odnosno dolazi do sporog učenja. Koristeći ReLU aktivacijsku funkciju, omogućuje se da se ne prenosi negativni gradijent dok pozitivni ostane nepromijenjen. Na narednoj stranici grafički je prikazana ReLU aktivacijska funkcija [Slika 3].

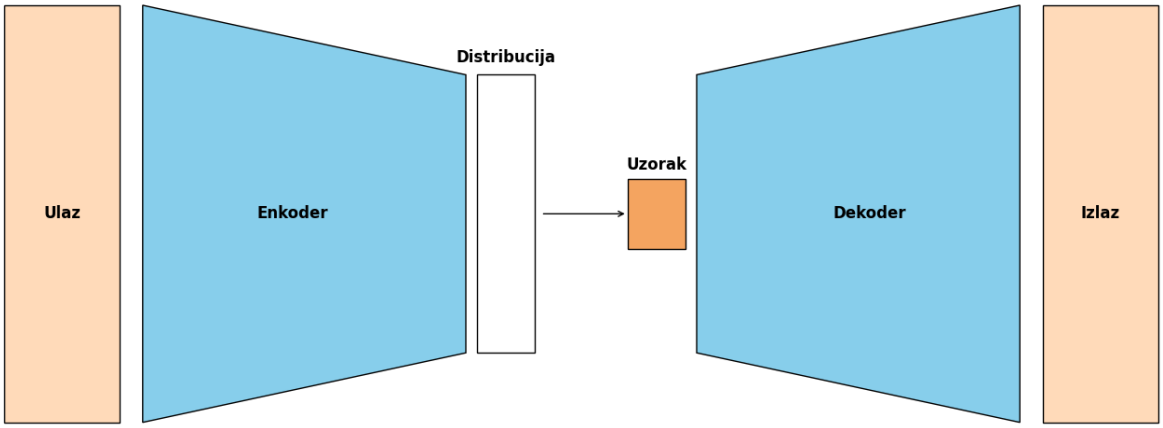


Slika 3. ReLU aktivacijska funkcija

Posebna vrsta generativnog modela koja koristi ReLU aktivacijsku funkciju (ovisno o specifičnim potrebama i arhitekturi modela) jest varijacijski autoenkoder (engl. ***V**ariational **A**utoencoder*, VAE). Lakoća generiranja sličnih ili potpuno novih uzoraka, na temelju postojećih podataka, je omogućena jedinstvenom strukturom ove mreže: enkoder i dekoder. Enkoder uzima originalni ulazni podatak i pretvara ga u kompaktnu reprezentaciju, poznatu kao latentni prostor, dok dekoder koristi tu reprezentaciju za rekonstrukciju originalnog ulaza. Ključna značajka VAE-a je kontinuiranost latentnog prostora što omogućuje lako uzorkovanje i interpolaciju. Kako bi se ova kontinuiranost mogla osigurati i pravilno rekonstruirati, koristi se tzv. donja granica vjerojatnosti (engl. ***E**vidence **L**ower **B**ound*, ELBO), funkcija gubitka čijom maksimizacijom se „prisiljava“ latentni prostor da prati normalnu distribuciju [5]. Time se doprinosi poboljšanju generativnog modela tako da bolje rekonstruira ulazne podatke i

istovremeno osigurava stabilan latentni prostor. Zbog toga, ovaj model se često koristi u generiranju novih podataka, najčešće slika i zvuka, kada je ograničen skup podataka za učenje. Ipak, jedan od nedostataka varijacijskih autoenkodera jest taj da generirane slike često mogu biti „mutne“ [4, 5].

Razlog tome jest upravo u načinu na koji VAE optimizira funkciju gubitka s minimizacijom između originalnih ulaznih podataka i podataka koji se rekonstruiraju dok se istovremeno zadržava niska dimenzionalnost latentnog prostora radi sažimanja informacija. Ovaj pristup dovodi do gubitka informacija za jasnu rekonstrukciju podataka, odnosno „mutnih“ slika. Kako bi se poboljšala kvaliteta rekonstrukcije važno je odabrati odgovarajuću arhitekturu modela ili dovoljno fleksibilan generativni model tako da se minimizira rekonstrukcijski gubitak, ali također da prostor može sačuvati sve važne informacije [5]. Slika ispod prikazuje pojednostavljenu strukturu varijacijskog autoenkodera s ugrađenom normalnom distribucijom za generiranje novih podataka [Slika 4].



Slika 4. Pojednostavljena struktura VAE-a s ugrađenom normalnom distribucijom

Time se dolazi do generativnih suparničkih mreža koje su i danas u širokoj upotrebi za generiranje različitih vizualnih sadržaja. Najčešće to uključuje nove slike za skupove podataka, fotografije ljudskih lica ili fotorealistične slike. U nastavku, istražuje se teorijska osnova, primjene i podvrste GAN-ova kako bi se bolje shvatila njihova kompleksnost.

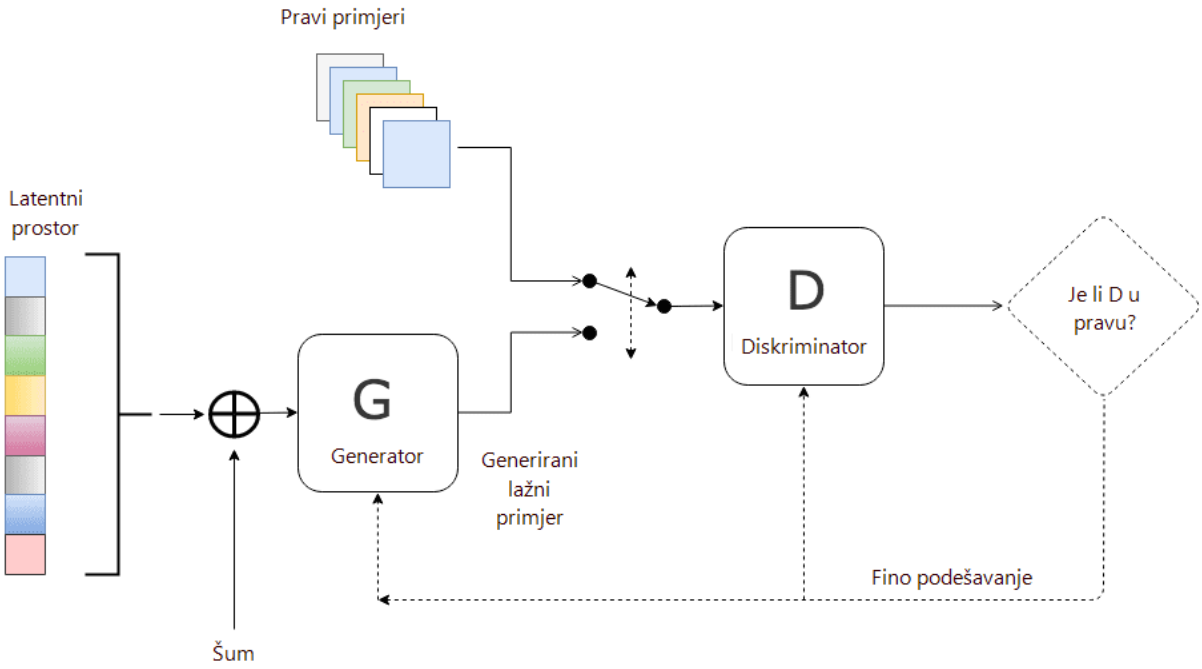
1.2. Teorijska osnova, primjena i podvrste GAN-a

Kao što je u početku rada navedeno, osnovu svakog GAN-a čine generator i diskriminator. Generator je treniran za stvaranje novih primjera iz određenog skupa podataka (maksimizira se vjerojatnost da diskriminator pogriješi [1]), a zadatak diskriminatora je razlučivanje je li generirani podatak autentičan ili krivotvorina (izraženo skalarnom vrijednošću od 0 do 1). Točnost diskriminatora, u razlikovanju između stvarnih i generiranih podataka, koristi se za izračunavanje gubitka tijekom treninga i propagiranje pogreške unatrag (engl. *backpropagation*) kroz generator i diskriminator. Time kroz vrijeme, modeli se poboljšavaju te generirani podatci postaju kvalitetniji i uvjerljiviji. Ovakva dinamika između generatora i diskriminatora, gdje se jedan model uči generirati što vjerodostojnije krivotvorine dok drugi uči razlikovati takve primjere od stvarnih, ključna je za učenje GAN-a. Ova cijela minimaks procedura se može opisati sljedećom funkcijom cilja:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (1.2.1)$$

gdje *G* predstavlja generator, *D* diskriminator, *x* stvarne podatke, *G*(*z*) generirane podatke i *z* slučajni vektor uzet iz šuma $p_z(z)$ [1]. Prva komponenta, $log\, D\,(x)$, predstavlja očekivanu vrijednost logaritma izlaza diskriminatora za stvarne podatke. Budući da je potrebno što točnije prepoznavanje stvarnih podataka od strane diskriminatora, cilj je minimizirati vrijednost prve komponente izraza (1.2.1). Druga komponenta izraza, $\log\big(1 - D\big(G(z)\big)\big)$, predstavlja očekivanu vrijednost logaritma suprotnosti izlaza diskriminatora za generirane podatke. Cilj je ovu vrijednost minimizirati – generator treba stvarati podatke koje diskriminator teško razlikuje od stvarnih. U praksi ova dvostrana „igra“ između generatora i diskriminatora se rješava iterativnim, numeričkim postupkom [1, 4]. Ključno, generator ima za cilj maksimizirati logaritam vjerojatnosti pogreške diskriminatora, umjesto minimiziranja sposobnosti diskriminatora da ispravno klasificira ulazni podatak. Ovime derivacija funkcije cilja generatora ostaje visoka u odnosu na izlaze diskriminatora što rezultira stabilnijim treniranjem GAN-a i omogućuje generiranje podataka koji se teže razlikuju od stvarnih podataka. Pojednostavljeno, generator se uči dok diskriminator miruje. Zatim se diskriminator uči s pomoću sintetičkih podataka iz generatora, razlučuje je li dobiveni ulaz autentičan ili krivotvoren te ti rezultati se koriste za daljnje učenje generatora kako bi postao bolji u sljedećoj

iteraciji. S druge strane, kada diskriminator uči generator miruje. U ovoj fazi je prisutna samo propagacija unaprijed gdje diskriminator uči i s pravim i s krivotvorenim podatcima, pokušavajući ispravno prepoznati dobiveni podatak. Cijeli rad generativnih suparničkih mreža može se vizualizirati s pomoću narednog prikaza [Slika 5], gdje su jasno vidljivi položaji generatora i diskriminatora te njihova međusobna relacija.



Slika 5. Generalna struktura generativne suparničke mreže (GAN), [6]

U kontekstu učenja GAN-a, važno je naglasiti ulogu Kullback-Leiblerove (KL) divergencije, koja služi za određivanje razlike između dvije vjerojatnosne distribucije. Ovo se može opisati sljedećom jednadžbom:

$$D_{KL}(P||Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right) \tag{1.2.2}$$

gdje $P(i)$ i $Q(i)$ predstavljaju vjerojatnost događaja i za distribuciju P odnosno Q.

KL divergencija je uvijek nenegativna i jednaka je nuli ako i samo ako su P i Q iste distribucije. Međutim, KL divergencija nije simetrična - $D_{KL}(P||Q)$ nije nužno jednako $D_{KL}(Q||P)$ [4]. U vezi GAN-a, jedna distribucija može predstavljati stvarnu distribuciju podataka, dok druga bi

predstavljala distribuciju generiranih podataka. Tada bi se ova divergencija mogla koristiti kao dio funkcije gubitka kako bi se generator naučio proizvesti podatke koji bolje oponašaju stvarnu distribuciju, čime raste mogućnost „varanja“ diskriminatora i generiranja boljih slika. Ipak, u procesu treniranja GAN-a često je susresti se s izazovima koji mogu ometati učenje generatora/diskriminatora i kvalitetu generiranih podataka. Prije svega, treniranje GAN modela je notorno po svojoj nestabilnosti i čestog mijenjanja parametara (nerijetko i cjelovite strukture modela kako bi se udovoljili zahtjevi prilikom generiranja novih podataka). Ova nestabilnost može proizaći iz nekoliko čimbenika, neki od najistaknutijih su:

- Kolaps modela (engl. *mode collapse*) – Događa se kada generator počne proizvoditi ograničen spektar izlaznih podataka, često generirajući vrlo slične ili potpuno identične podatke bez obzira na ulazne šumove. Ovo bi značilo da generator nije u stanju naučiti potpuni spektar mogućih podataka, ograničavajući se na limitiranu raznolikost generiranih podataka. Mogući uzroci su široki: od nedovoljno početnih podataka u skupu do nedovoljno razvijene strukture cijele mreže.
- Nestajući gradijent (engl. *vanishing gradient*) – Za razliku od prethodnog problema, gdje se uzrok može prepoznati u generatoru, problem nestajućeg gradijenta se pojavljuje u diskriminatoru kada postane previše dobar u razlikovanju pravih i generiranih podataka. To dovodi do situacije gdje generator ne prima dovoljno dobru povratnu informaciju radi potencijalnog poboljšanja (gradijenti postaju vrlo mali ili čak nestanu). Ovaj problem je istaknut tijekom propagiranja pogreške unatrag, gdje gradijent slabi dok putuje unatrag do početnih slojeva – posebno kada se koriste aktivacijske funkcije kao što su sigmoidna ili tangens hiperbolni (uobičajena praksa u GAN-ovima zbog načina predobrade, s normalizacijom u rasponima [0, 1] ili [-1, 1]). Kao rješenje ovog problema mogu se koristiti aktivacijske funkcije poput ranije već spomenute ReLU ili LeakyReLU funkcije (vidjeti sljedeće poglavlje).
- Problemi s konvergencijom – S obzirom na strukturu generativnih suparničkih mreža, gdje je cijela ideja da su dvije neuronske mreže u odnosima konstantnog međusobnog nadmetanja radi poboljšanja u performansama, dostizanje konvergencije može biti izazovno. Ako jedan od modela postane znatno bolji od drugog može doći do situacije gdje se učenje zaustavlja i GAN više ne napreduje. Najčešći način rješavanja ovog problema jest prilagođavanja stope učenja (engl. *learning rate*) generatora i/ili diskriminatora.

Unatoč ovim poteškoćama, kao što bi se moglo pretpostaviti, generativne suparničke mreže su se pokazale vrlo popularnim izborom u sferi strojnog učenja zbog mogućnosti lakog rekreiranja vizualnog sadržaja s rastućom razinom preciznosti. No, osim što su korisne u kreiranju realističnog vizualnog sadržaja, generativne suparničke mreže se također mogu primjenjivati i u drugim sektorima poput:

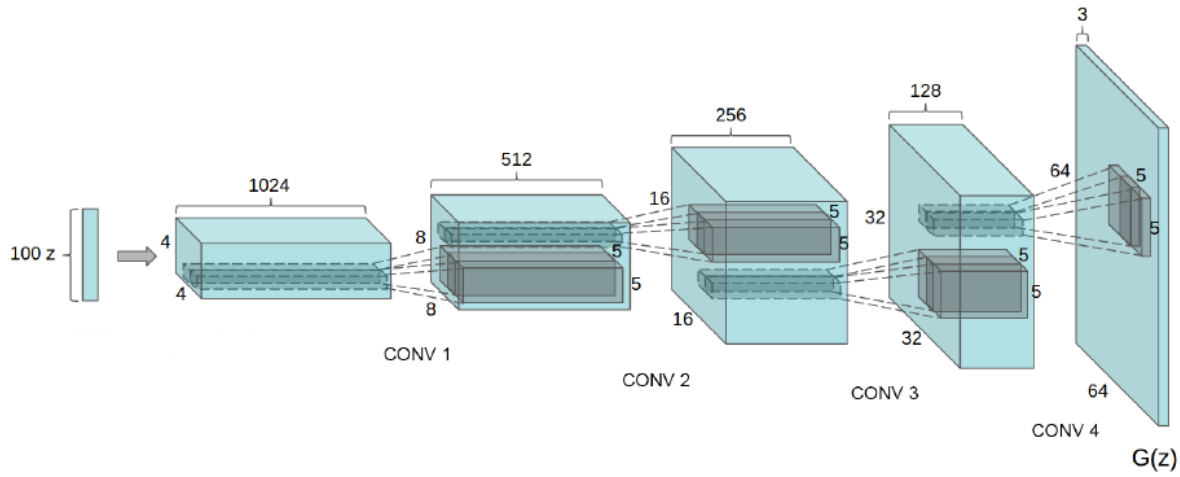
- **Medicine** – Koristeći napredne strukture, pokazalo se korisnim u oftalmologiji primijeniti GAN za obradu i poboljšavanje slika, uključujući segmentaciju, povećanje kvalitete, uklanjanje šuma te generiranje realističnih oftalmoloških slika za dijagnostičke svrhe [7]. Dodatno, generativne suparničke mreže se dokazano mogu koristiti u otkrivanju i razvoju novih lijekova. GAN se koristi za pronalazak kemijskih spojeva sa željenom funkcionalnošću te u stvaranju realističnih proteinskih sekvenci koje pomažu u ciljanju potrebnih staničnih receptora za otkrivanje lijekova [8].
- **Autonomne vožnje** – Sustavi autonomne vožnje moraju biti izrazito robusni što zahtijeva treniranje modela s raznim mogućim scenarijima koji se mogu dogoditi u stvarnom svijetu. Priprema takvog skupa podataka u praksi je neizvediva, zbog čega se često koriste simulirani, sintetički podatci – ovdje GAN nudi obećavajući pristup u generiranju upravo takvih skupova podataka [9]. Dodatno, GAN se već od prije pokazao vrlo korisnim u obradi slika te se radi toga može koristiti za izmjenu postojećih slika, dodavanjem ili mijenjanjem scenarija (dan u noć, crno-bijelu sliku pretvoriti u sliku u boji, dodavanje pješaka u slikama i sl.).
- **Glazbe** – Osim generiranja slika, GAN-ovi se mogu koristiti i za generiranje glazbe. 2019. je predstavljen tzv. GANSynth koji koristi ProGAN strukturu (više u podpodnaslovu 1.2.3) za inkrementalno povećavanje zvuka iz jednog vektora. U kombinaciji s „NSynth“ skupom podataka različitih tonova, može se stvoriti zvuk kojem je moguće mijenjati ton i boju [10].
- **Financije** – Budući da GAN ima mogućnost stvaranja sintetičkih skupova podataka, svakako da bi ovo bilo uporabljivo u uvjetima gdje stvarni podatci su nedostupni. Radi toga se u financijama GAN može koristiti za predviđanje kretanja tržišta kao i optimiranje financijskih portfelja [11].

Naravno, postoji još primjera primjene GAN-a osim navedenih, ali svi dijele istu centralnu strukturu međusobnog nadmetanja generatora i diskriminatora s krajnjim ciljem generiranja sve boljih podataka. Ipak, postoje različite vrste GAN-a, sa značajnim razlikama u strukturama generatora/diskriminatora, funkcijama gubitka, optimizatorima, stopama učenja ili načinima predobrade skupa podataka prije treniranja GAN-a. Na sljedećim stranicama su opisane najznačajnije GAN strukture koje su i danas u širokoj primjeni te se na kraju završava usporednom tablicom radi lakše predodžbe arhitekture, primjene, karakteristike i ograničenja navedenih podvrsti GAN-a.

1.2.1. DCGAN

Naprednija verzija generativnih suparničkih mreža, duboke konvolucijske generativne suparničke mreže (engl. ***D**eep **C**onvolutional **G**enerative **A**dversarial **N**etworks*, DCGAN) predstavljaju značajni pomak u generativnom modeliranju. Kao što naziv sugerira, DCGAN kombinira duboke konvolucijske neuronske mreže (engl. ***D**eep **C**onvolution **N**eural **N**etworks*, DCNN) s konceptom suparničkog učenja što omogućuje precizno generiranje izuzetno složenih vizualnih sadržaja. Konvolucijski slojevi koriste filtere koji se pomiču po ulaznoj slici kako bi se izdvojile lokalne značajke. Najčešće bi to bili rubovi ili boje, ali duboke konvolucijske neuronske mreže su složenije i imaju veći broj slojeva od običnih konvolucijskih neuronskih mreža te, kao takve, imaju mogućnost naučiti apstraktnije i složenije značajke iz podatka. Ovime se poboljšava sposobnost rješavanja kompleksnih vizualnih problema. Uz konvolucijske slojeve su obično prisutni i slojevi sažimanja (engl. *pooling layers*) koji imaju ulogu smanjivanja dimenzija podataka. Svi ovi slojevi su i povezani čime DCNN ima mogućnost donošenja odluka na temelju naučenih značajki.

Međutim, kako bi se omogućio veći raspon skupova podataka, veća rezolucija izlaznih slika i sveukupno veća kvaliteta generativnog modela, potrebno je ukloniti potpuno povezane slojeve te zamijeniti ih globalnim prosječnim slojem sažimanja [12]. Naredna slika prikazuje strukturu generatora u dubokoj konvolucijskoj generativnoj suparničkoj mreži [Slika 6].



Slika 6. Struktura generatora u DCGAN modelu, [12]

1.2.2. Wasserstein GAN

Wasserstein generativne suparničke mreža (engl. ***W**asserstein **G**enerative **A**dversarial **N**etwork*, WGAN) je varijanta tradicionalnog GAN-a koja se razlikuje po načinu mjerenja razlike između generirane i stvarne distribucije podataka. Način mjerenja u WGAN-ovima je tzv. Wassersteinova udaljenost, poznato i kao *Earth Mover's distance* (EM *distance*) – opisano sljedećom jednadžbom:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}[k(x,y)], \qquad (1.2.2.1)$$

gdje je γ član skupa $\prod(P_r, P_g)$ (P_r – distribucija stvarnih podataka, P_g – distribucija generiranih podataka), a $k(x, y)$ je funkcija udaljenosti između uzoraka. Prednost ovog pristupa je u tome što čini učenje GAN-a stabilnijim [13], čime se izbjegavaju dosadašnji izazovi kod učenja običnih GAN-ova (kao što je ranije spomenuti kolaps modela).

1.2.3. ProGAN

Progresivno rastuće generativne suparničke mreže (engl. ***P**rogressively growing **G**enerative **A**dversarial **N**etwork*, ProGAN) predstavljaju značajni pomak u generativnoj tehnologiji. Ovaj pristup, predstavljen od strane NVIDIA-e 2017. [14], implementira inovativnu metodu progresivnog povećavanja rezolucije generiranih slika tijekom procesa

učenja. Postupnim učenjem se omogućuje da mreža pređe s osnovne strukture slike te da postepeno prelazi na detalje sve finije rezolucije. Time se postigne znatno poboljšanje u kvaliteti generiranih podataka, istovremeno omogućavajući usavršavanje detalja bez simultanog učenja svih razina složenosti – što izravno doprinosi stabilnosti modela. Dodatno, ovim postupkom ProGAN-u je potrebno značajno manje vrijeme za učenje čime se smanjuje i vjerojatnost za kolaps modela. Ipak, nedostatak ProGAN-a jest što zahtijeva veću računalnu snagu i memoriju zbog svoje složenosti i zahtjeva za generiranjem slika visoke rezolucije. Unatoč tome, progresivno rastući GAN-ovi su našli široku primjenu u različitim područjima zbog svojeg preciznijeg modeliranja distribucije podataka, uključujući generiranje realističnih ljudskih lica (primjer ispod [Slika 7]) i umjetničkih slika.

Slika 7. Primjeri generiranih ljudskih lica pomoću ProGAN-a, [14]

1.2.4. StyleGAN

Godinu dana nakon ProGAN-a, NVIDIA je predložila novi generativni model u StyleGAN-u koji, uz generiranje slika visoke rezolucije, ima značajnu kontrolu nad stilom i sadržajem generiranih slika. Ova inovacija se postiže mapiranjem latentnog prostora u međuprostore koji kontroliraju različite aspekte izgleda slike; poput teksture, boje i forme [15]. Takav pristup omogućava stvaranje detaljnih i izrazito realističnih slika te i sposobnost manipulacije stilom prikaza slike bez mijenjanja osnovne strukture objekta. Arhitektura StyleGAN-a nadograđuje temelje postavljene ProGAN-om, nastavljajući s inkrementalnim povećanjem rezolucije generiranih slika tijekom procesa učenja. Ključna inovacija je ranije spomenuta mreža koja transformira ulazni latentni vektor u međuprostor, ali i tehnika adaptivne normalizacije instance (engl. ***A**daptive **I**nstance **N**ormalization*, AdaIN) koja prilagođava stil svakog sloja u generatoru – čime se omogućuje djelovanje modela i na najsitnije detalje generirane slike. Radi ovih specifičnosti, StyleGAN se ističe sa svojom sposobnošću generiranje izuzetno realističnih slika (posebno ljudskih lica, [16]) s visokom razinom detalja zbog mogućnosti preciznog upravljanja stilom slike. Međutim, kao i ProGAN, StyleGAN također zahtijeva značajne računalne resurse što može biti ograničavajući čimbenik za širu upotrebu. Također, postoje etičke dileme vezano uz korištenje ovakve tehnologije za generiranje lažnih podataka. U posljednjih nekoliko godina su se značajno razvile mogućnosti ovakvih modela s primjetnim porastom u kvaliteti, toliko da je potpomoglo nastajanju fenomena tzv. deepfakea – lažni foto, video ili audio materijal. Primjerice, sljedeća slika prikazuje osobe koje ne postoje – potpuno generirane s pomoću StyleGAN2 [Slika 8]. Iako nije savršeno, jasan je napredak ovih generiranih slika naspram onih koji su generirani ProGAN strukturom [Slika 7].

Slika 8. Generirana lica pomoću StyleGAN2 strukture, [16]

1.2.5. BigGAN

Jedan od nedostataka uobičajenih generativnih suparničkih mreža kod generiranja visokokvalitetnih slika jest što često generirane slike su manjih dimenzija. Zbog jednostavnosti u strukturi i manjka resursa, generiranje većih dimenzija često dovede do slika koje su niže rezolucije bez detalja. Kako bi se suprotstavilo ovome, 2018. *DeepMind* razvija velike generativne suparničke mreže (engl. ***B**ig **G**enerative **A**dversarial **N**etworks*, BigGAN) – napredna verzija GAN-a koja je posebno dizajnirana za generiranje visokokvalitetnih slika velike rezolucije. BigGAN koristi zajedničku ugradbenu klasu koja se linearno projektira na svaki sljedbeni sloj [17], za razliku od ranijih GAN modela gdje bi često svaki sloj imao svoju zasebnu ugradbenu klasu. Ovime se poboljšavaju performanse i brzina učenja što omogućuje učinkovitije upravljanje informacijama vezanih za klasu slike koja se generira. Još jedna inovacija BigGAN-a jest uvođenje izravne veze između vektora šuma i više slojeva generatora (engl. *skip-z*), a ne samo u početku što je uobičajena praksa. Ideja ovoga jest da generator će moći koristiti latentni prostor za izravan utjecaj na različite razine hijerarhije u generiranim slikama, što doprinosi većoj raznolikosti i realističnosti. Ova promjena je omogućila poboljšanje performansi modela od oko 4% i ubrzala učenje za dodatnih 18% [17]. Ipak, kao u i ranije opisanim modelima koji generiraju visokorealistične slike s velikom rezolucijom i količinom detalja, BigGAN također ima veliku potrebu za računalnim resursima te ima dodatni rizik prenaučenosti zbog svog velikom kapaciteta.

1.2.6. CycleGAN

Ciklički konzistentna generativna suparnička mreža (engl. ***C**ycle-consistent **G**enerative **A**dversarial **N**etwork*, CycleGAN) predstavlja rješenje za još jedan problem kod konvencionalnih GAN-ova, a to je problem nesparenog prevođenja slike iz jedne domene u drugu. Obično, da bi prevođenje slike bilo uspješno potrebno je imati uparene podatke (npr. ista scena – slika/fotografija u različitim stilovima ili pod različitim kutovima). Za CycleGAN je pak dovoljno da ima skupove slika koji su općenito iz iste domene (npr. skup slika jabuka i odvojeni skup slika naranči), ali bez potrebe da za svaku sliku iz jednog skupa postoji izravno uparena slika iz drugog skupa što je značajno jer time se pruža mogućnost veće manipulacije izlaznog podatka iz modela te dobivanje raznovrsnijih slika. Ovo se postiže kroz gubitak cikličke konzistentnosti (engl. *cycle consistency loss*), koje osigurava da slika koja je prevedena iz jedne domene u drugu, kada se ponovno prevede natrag u izvornu domenu, zadržava svoje

ključne karakteristike. Nadalje, CycleGAN implementira dvije zasebne mreže generiranja, $G: X \rightarrow Y$ i $F: Y \rightarrow X$, koje rade na pretvaranju slika između dviju nesparenih domena [18]. Ovi principi pomažu u smanjenju mogućnosti za gubitak relevantnih informacija tijekom procesa prijevoda, što je čest problem u tradicionalnim GAN modelima. U arhitekturi CycleGAN-a, generator se obično sastoji od konvolucijskih i dekonvolucijskih slojeva za rekonstrukciju slike dok diskriminator koristi poseban tip dizajna s tzv. PatchGAN strukturom [18], koja kažnjava strukturu samo na razini lokalnih dijelova slike, fokusirajući se na detalje umjesto na sliku kao cjelinu. Ovi elementi omogućuju detaljnu i preciznu transformaciju slika između dviju domena te čine CycleGAN izrazito fleksibilnim modelom imajući u vidu da se može koristiti za generiranje realističnih slika kada upareni podatci nisu lako dostupni. Izazovi prilikom učenja ovakvog modela bi svakako bili vrijeme i resursi budući da je ponekad potrebna značajna količina slika za učenje. Dodatno, budući da je struktura CycleGAN-a relativno kompleksna naspram već navedenih podvrsti GAN-a, rezultati ovog modela mogu biti teški za interpretaciju – često ne znajući zašto neke generirane slike su takve kakve jesu. No, unatoč tim nekim poteškoćama, inovacije uvedene u ciklički konzistentnim generativnim suparničkim mrežama značajno su doprinijele generiranju visokokvalitetnih rezultata sa širokim spektrom primjena. Sljedeća slika prikazuje uspješan primjer primjene CycleGAN modela za generiranje slika jabuka i naranči [Slika 9].

Slika 9. Primjer preobrazbe objekta iz jabuke u naranču i naranče u jabuku, [18]

1.3. Tablica usporedbe podvrsti GAN-a

Sljedeća tablica pruža sveobuhvatan pregled ranije opisanih podvrsta generativnih suparničkih mreža, uključujući arhitekture, primjene, ključne karakteristika kao i izazovi/ograničenja s kojima je moguće se susresti prilikom učenja/treniranja. Primjerice, „Vanilla“ GAN predstavlja osnovni GAN model koji je u najširoj upotrebi; premda je jednostavan, podložan je izrazitim nestabilnostima prilikom učenja.

Tablica 1. Pregled podvrsta generativnih suparničkih mreža

Podvrsta GAN-a	**Arhitektura modela**	**Primjene**	**Ključne karakteristike**	**Izazovi/ Ograničenja**
„Vanilla“ GAN	Jednostavna arhitektura s potpuno povezanim slojevima.	Generiranje slika, osnovni GAN eksperimenti.	Jednostavnost.	Nestabilnost učenja, teškoće u generiranju visokokvalitetnih slika.
DCGAN	Konvolucijske mreže.	Umjetnost, moda.	Konvolucijski slojevi za bolju kvalitetu slika.	Zahtijeva prilagodbu hiperparametara.
WGAN	Slično običnom GAN-u, s prilagodbom u funkciji gubitka.	Generiranje realističnijih slika.	Bolja stabilnost učenja zahvaljujući Wassersteinovoj udaljenosti.	Složenost implementacije i podešavanja.
ProGAN	Progresivno povećanje rezolucije kroz slojeve.	Visokorezolucijske slike.	Progresivno dodavanje slojeva za detaljnije slike.	Zahtijeva više resursa i vremena za učenje.
StyleGAN	Složena arhitektura s dodatnim stilskim slojevima.	Umjetničke slike, modifikacija lica.	Kontrola stila i detalja putem stilskih slojeva.	Složenost modela i kontrola generiranja.

BigGAN	Velike mreže s visokim kapacitetom i proširenim skupom podataka.	Umjetnost, znanstvena vizualizacija.	Izuzetno visoka rezolucija i kvaliteta slika.	Zahtijeva iznimno mnogo računalnih resursa.
CycleGAN	Dvije mreže za cikličke transformacije.	Omogućuje prenošenje stila bez sparenih podataka.	Učinkovito za višestruke zadatke transformacije.	Složenost raste s brojem zadataka transformacije, mogući gubitak detalja.

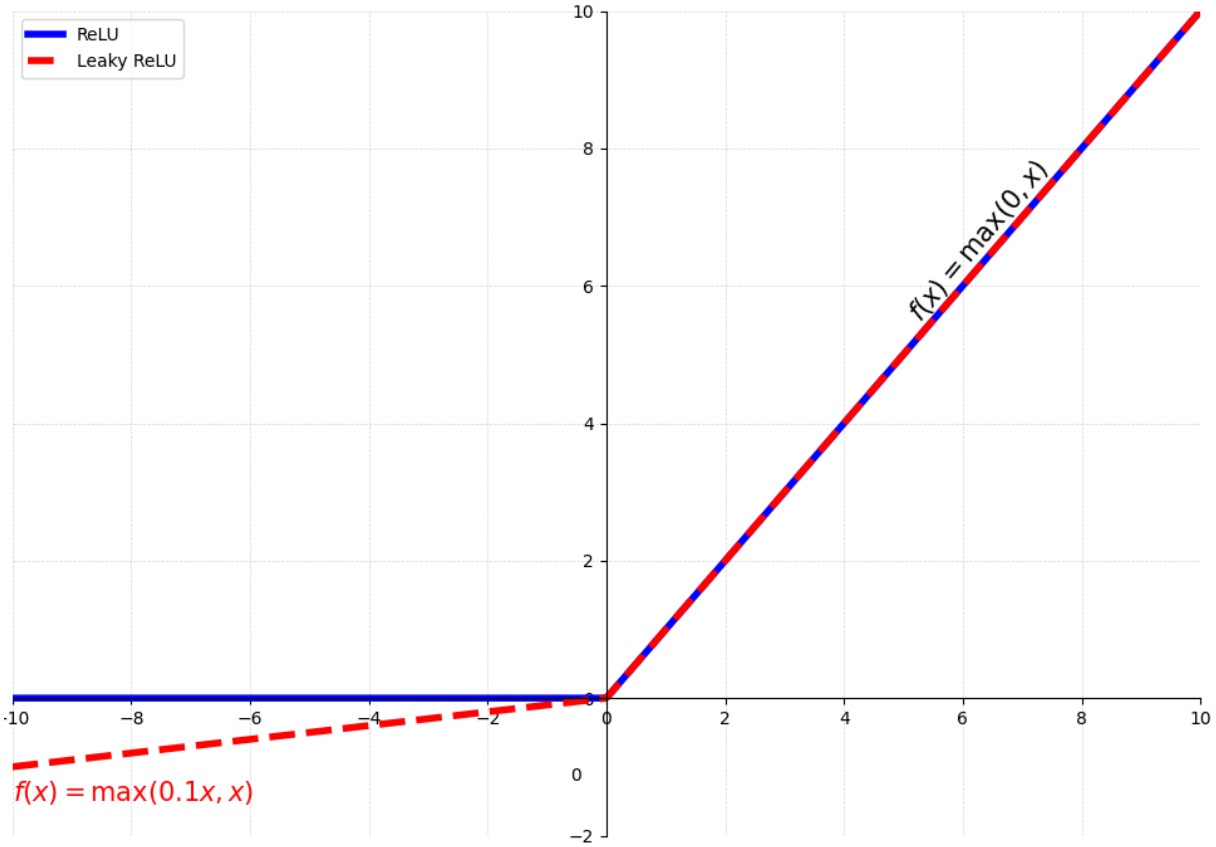
Iako ova tablica može služiti kao putokaz za biranje određene podvrste generativne suparničke mreže za određeni zadatak, često se kombiniraju različiti aspekti GAN podvrsti radi najboljih rezultata. Na primjer, ranije spomenuta funkcija gubitka WGAN-a, Wassersteinova udaljenost, se može koristiti kao funkcija gubitka u većini GAN struktura zbog smanjivanja nestabilnosti tijekom učenja i bolje konvergencije učenja [13].

2. METODOLOGIJA

2.1. Tehnički okvir i alati za razvoj GAN modela

Kao i kod svakog projekta u domeni strojnog učenja, temeljni korak u procesu razvoja generativne suparničke mreže jest biranje odgovarajućeg tehničkog okvira i alata za razvoj, jer pravilnim odabirom se može znatno smanjiti potreban rad dok se istovremeno osigurava visoka kvaliteta i učinkovitost razvijenog modela. Prije svega, važno je odlučiti u kojem programskom jeziku će generativna suparnička mreža biti razvijena jer se u strojnom učenju nadasve cilja na brzinu, djelotvornost i laku implementaciju. Zbog tih svih razloga, *Python* se pokazao kao industrijski standard u strojnom učenju jer pruža obilje biblioteka specijaliziranih za strojno učenje koje poboljšavaju performanse, relativno je jednostavan jezik i ima veliku zajednicu s mnoštvo resursa te se lako integrira s drugim sustavima i alatima. Upravo kada je riječ o bibliotekama, *TensorFlow*, koji je razvijen od strane *Googlea* 2015. godine, ističe se kao izbor za razvoj GAN-a zbog svojih naprednih mogućnosti i fleksibilnosti. Prije svega, *TensorFlow* podržava automatsko diferenciranje i paralelno izvršavanje, što omogućuje treniranje modela na različitim hardverskim platformama, uključujući CPU-e (središnja procesorska jedinica, engl. ***C**entral **P**rocessing **U**nit*), GPU-e (grafički procesor, engl. ***G**raphics **P**rocessing **U**nit*) i TPU-e (jedinica za obradu tenzora, engl. ***T**ensor **P**rocessing **U**nit*) [19]. Ovo je važno jer ako je cilj razviti GAN model, takav da generira kvalitetne umjetničke slike, nužno je učinkovito koristiti računalne resurse na raspolaganju. Ipak, korištenje *TensorFlow* biblioteke pruža dodatan benefit u vidu integracije, a to je na *Googleovoj Colab* platformi, koja pruža pristup naprednim računalnim resursima poput već spomenutog GPU-a i TPU-a – nešto što bi iznimno ubrzalo učenje jednog složenog GAN modela. Nadalje, *TensorFlow* se može kombinirati s različitim API-jima (aplikacijskom programsko sučelje, engl. ***A**pplication **P**rogramming **I**nterface*) za izgradnju i učenje modela pogodnih za razvoj generativnih suparničkih mreža. Jedan od tih je *Keras*, API visoke razine koji se koristi za duboko učenje te je izravno dostupan preko *TensorFlow* biblioteke kao „tf.keras“ [19]. Ova biblioteka ima niz vrlo korisnih funkcija i optimizatora koje se koriste prilikom razvoja ikakvih modela strojnog učenja. Primjerice, funkcije poput „Conv2D“, „Dense“, „Flatten“ i „Reshape“ daju mogućnost manipulacije dimenzijama tenzora i ulaza. Također je u slojeve moguće dodati zasebne aktivacijske funkcije radi drugačijeg izlaza. Jedna od tih je i LeakyReLU, koja je posebna vrsta aktivacijske funkcije gdje umjesto potpunog poništavanja negativnog dijela ulaza događa se mali propust negativnog dijela ulaza što može pomoći u ublažavanju problema nestajućeg gradijenta i pomaže za

prikupljanje više podataka. Ažuriranjem ranijeg prikaza ReLU aktivacijske funkcije iz uvodnog dijela dobije se usporedni prikaz gdje je jasno vidljiva razlika između „obične“ ReLU i LeakyReLU aktivacijske funkcije [Slika 10].



Slika 10. Razlika između ReLU i LeakyReLU aktivacijske funkcije

Osim ovoga, potrebno je koristiti još i druge *Python* biblioteke koje bi pružile podršku tijekom razvoja. Jedna od tih je *NumPy*, popularna biblioteka za znanstveno računanje s podrškom za velike, višedimenzionalne nizove i matrice. Dalje tu je *Matplotlib*, biblioteka za vizualizaciju podataka koja je neizostavna u svakom okviru rada strojnog učenja. Dodatno bi vrijedilo spomenuti *h5py* i *OS*: prvi se koristi za interakciju s HDF5 (engl. ***H**ierarchical **D**ata **F**ormat version 5*) binarnim formatom podataka (mogućnost pohrane velike količine podataka), a druga biblioteka je pogodna za interakciju s operativnim sustavom kako bi se lakše čitalo ili pisalo u sustav datoteka. Potencijalna alternativa je korištenje *PyTorch* biblioteke, koja je razvijena 2017. od strane *Facebooka* te se ističe po svojoj jednostavnosti i lakoći korištenja. Obje biblioteke nude snažne mogućnosti za automatsko diferenciranje što je ključno za treniranje GAN-a zbog kompleksnosti modela gdje se traži učinkovito ažuriranje i prilagodba težina.

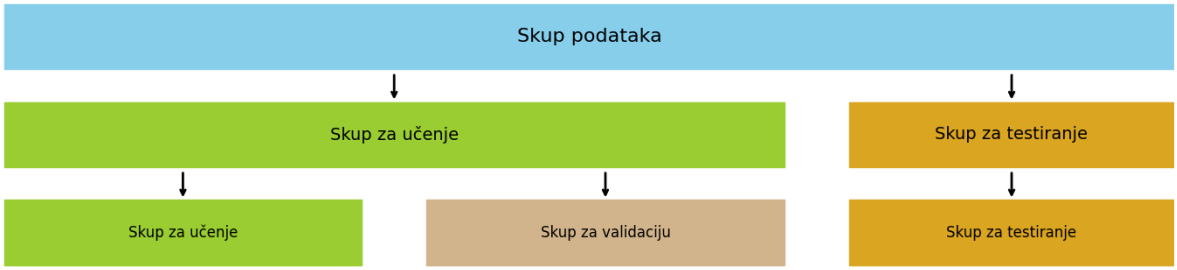
Ipak, izbor *Tensorflowa* nad *PyTorch* bibliotekom za razvoj GAN-a u ovom radu je motiviran iz već spomenute integracije s *Google Colab* platformom gdje ima pristup besplatnim GPU-ima i TPU-ima. Ovo omogućuje brže treniranje i iteraciju modela. Uz to, integracija *Tensorflowa* i *Kerasa* za funkcionalnost GAN-a omogućuje stvaranje jasnih i modularnih definicija za generator i diskriminator.

2.2. Detalji o odabranom skupu podataka

Podatci su temeljna komponenta u strojnom učenju. Preciznost i učinkovitost modela ovise o kvaliteti i relevantnosti skupa podataka koji se koristi. Skup podataka s bogatim i raznolikim informacijama omogućava modelu učinkovito učenje, odnosno prepoznavanje kompleksnih obrazaca, što je ključno za ostvarivanje točnih i pouzdanih rezultata. Ako podatci nisu dovoljno kvalitetni, postoji mogućnost prenaučenosti – model radi vrlo dobro na dostupnim podatcima, ali ne kada se dodaju novi. No, osim kvalitete važan je i kvantitet (količina) podataka jer, u slučaju da ih je nedovoljno, model neće biti adekvatno naučen za svoju namjenu. Dobar skup podataka bi također trebao biti lak za manipulaciju budući da često je potrebna predobrada podataka prilikom ikakve vrste modeliranja u strojnom učenju. Kako bi se povećala skalabilnost modela, podatci bi trebali biti raznoliki tj. trebali bi pokriti sve aspekte problema koji se rješava. Time se osigurava da će model naučiti i prepoznati širok spektar obrazaca čime se povećava mogućnost generiranja novih, nepoznatih podataka. Također, u svakom skupu podataka važan je balans kategorija kako bi se izbjegla pristranost modela prema dominantnoj kategoriji. Ovo bi se posebno odnosilo na generativne suparničke mreže sa svojom arhitekturom dvaju, suprotstavljenih neuronskih mreža.

Osim za učenje modela, pojedini skup podataka se može koristiti u druge svrhe, poput testiranja i validacije. Podatci za testiranje omogućuju procjenu uspješnosti modela, omogućavajući brzu procjenu trenutnih rezultata i mogućnost prilagodbe modela za poboljšanje performansi. Dodatna razina provjere je moguća s pomoću podataka za validaciju, koji su dio podataka za učenje te se koriste za povremenu evaluaciju modela prije završnog testiranja. Validacijski podatci omogućuju praćenje i optimizaciju parametara modela tijekom procesa učenja, bez utjecaja na testne podatke. Ovo pomaže u sprječavanju prenaučenosti modela jer model nije izložen testnim podatcima do završne faze. Ako model pokazuje visoku točnost na podatcima za učenje, ali nisku na validacijskim podatcima model je prenaučen, odnosno prekomjerno je prilagođen specifičnim obilježjima skupa za učenje, ali ne i za općenite obrasce.

Radi ovoga je važno dodatno podijeliti skup podataka te se ove podjele skupa podataka obično rade u omjeru 80:20 ili 70:30 [20]. Hijerarhija podjele skupa podatka na zasebne skupove za učenje, testiranje i validaciju je vidljiva sljedećom slikom [Slika 11].



Slika 11. Hijerarhija podjela skupa podataka (skup za učenje, skup za testiranje i skup za validaciju)

Međutim, nije nužna ova podjela gdje je zadatak generirati nove podatke (slike u ovom slučaju). Dovoljno je imati skup za učenje, gdje generator će imati kompleksniju strukturu naspram diskriminatora koji bi imao ulogu klasifikatora u ovom slučaju – prepoznavanje je li generirana slika lažna ili autentična. Imajući ovo u vidu, za uspješnu izradu zadatka potreban je raznolik skup slika s visokom rezolucijom kako bi se osigurala kvaliteta generirane slike. Preko platforme „DeepLake“, koja omogućuje učinkovito rukovanje velikim skupovima podataka, dostupni su skupovi idealni za učenje GAN-a. Istaknuti skup podataka umjetničkih slika je „wiki-art“ s preko 80 000 slika za preko dvadeset različitih umjetničkih stilova [21]. Slike su visoke rezolucije, što pruža modelu GAN-a detaljne i kvalitetne informacije za učenje. Da bi se pristupilo ovom skupu podataka potrebno je instalirati odgovarajuću *Python* biblioteku, što se može učiniti pomoću „pip“ naredbe na sljedeći način:

```
!pip install hub
```

Nakon instalacije, skup podataka se može učitati i izravno pristupiti za pregled, manipulaciju i korištenje u procesu učenje GAN-a. Raznolikost u stilovima i tehnikama omogućava modelu da nauči i rekonstruira širok spektar umjetničkih obrazaca. No, treba imati u vidu i računalne resurse kao i vrijeme procesiranja; dovoljno je uzeti jedan umjetnički stil s velikom količinom slika. S najviše podataka u skupu izdvaja se impresionizam s 13 060 slika te će to poslužiti kao centralni i jedini izvor za učenje generatora, ali potrebno je prvotno izdvojiti te podatke od ostalih što uključuje nekoliko koraka.

Prvo, izdvaja se lista dostupnih umjetničkih stilova, a zatim se identificira indeks koji odgovara impresionizmu:

```
artstyle_for_training =  'impressionism'
labels_list = ds.labels.info['class_names']

# Pronaći indeks za impresionizam
artstyle_index = labels_list.index(artstyle_for_training)
```

Zatim pomoću naredbe „filter“ se skup podataka filtrira tako da se izdvajaju samo slike impresionizma. Ovo omogućuje stvaranje fokusiranog skupa podataka koji će se koristiti za treniranje GAN-a.

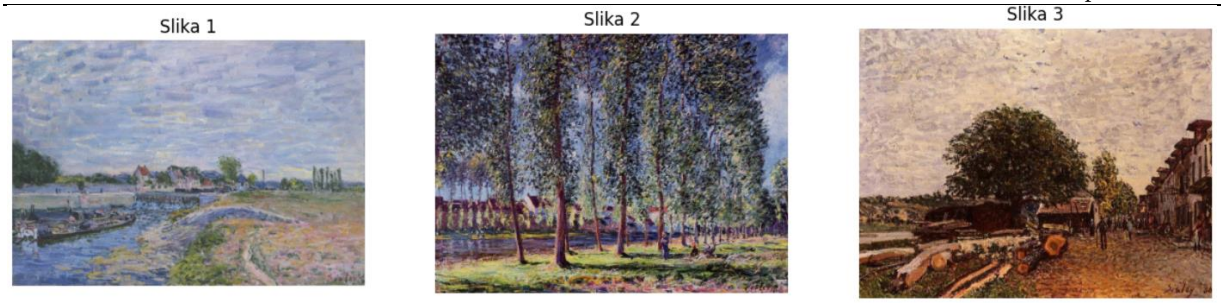
```
artstyle_ds = ds.filter(lambda x: x.labels.numpy() == artstyle_index)
```

Filtrirani skup podataka se zatim pretvara u *TensorFlow* skup podataka (kratica TFDS) što omogućava lakšu integraciju i manipulaciju podatcima unutar *TensorFlow* okruženja. Ovo se može napraviti jednom linijom kôda:

```
artstyle_ds_tf = artstyle_ds.tensorflow()
```

Svakako se može i provjeriti je li filtriranje skupa podataka ispravno izvedeno. Za to se definira funkcija „show_images“ koja prikazuje uzorak slika iz skupa podataka, omogućavajući vizualnu provjeru da su slike uistinu impresionističke. Definicija funkcije je vidljiva ispod te njenim pokretanjem se generiraju tri slike u stilu impresionizma [Slika 12].

```
def show_images(dataset, num_images):
    # Postavi veličinu slike temeljem izbora broja slika za prikazati
    rows = 1
    cols = num_images
    plt.figure(figsize=(5 * cols, 5 * rows))

    for i, batch in enumerate(dataset.shuffle(buffer_size=500).take(num_images)):
        image = batch['images'].numpy()  # Convert: tensor -> numpy array
        plt.subplot(rows, cols, i+1)
        plt.imshow(image)
        plt.title(f'Slika {i+1}')
        plt.axis('off')

    plt.show()

show_images(artstyle_ds_tf, 3) # Prikazuje 3 slike u jednom redu
```



Slika 12. Tri nasumično odabrane impresionističke slike iz filtriranog skupa podataka

Ovim je potvrđeno da je filtriranje skupa podataka uspješno te je time zaključen skup podataka koji će služiti za generiranje umjetničkih slika.

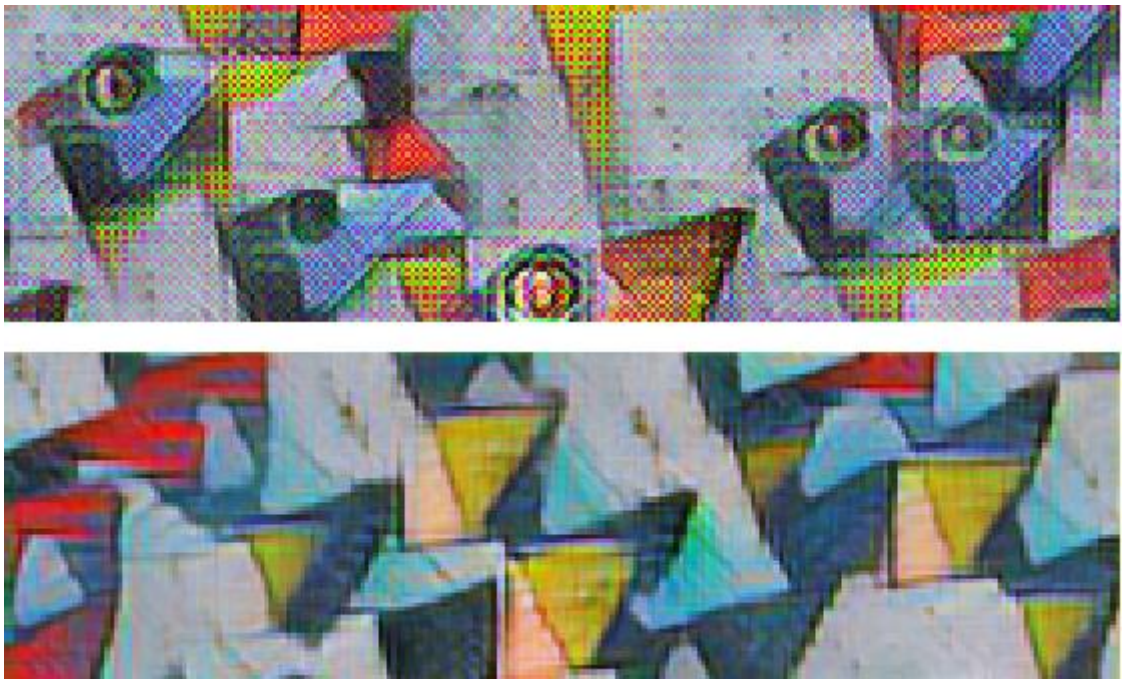
2.3. Struktura i dizajn primijenjenog GAN modela

Kako bi generativna suparnička mreža stvarala visokokvalitetne umjetničke slike postoji par preduvjeta koji bi se trebali ispuniti u vidu strukture. Prije svega, kao što je već ranije istaknuto, GAN je računalno intenzivan model – pogotovo ako je primjena za generiranje slika visoke rezolucije. Kompleksnija arhitektura obično zahtijeva veće računalne resurse; veći broj slojeva predstavlja dublju neuronsku mrežu što traži više vremena i resursa za učenje modela jer tada ima više parametara koji se moraju optimizirati. Isto tako utječe i širina mreže, odnosno broj čvorova unutar svakog sloja – veća širina obično zahtijeva više memorije i procesorske snage. Budući da je zadatak izraditi umjetničke slike to nosi svoje zahtjeve na računalne resurse, posebno zbog boje (sva tri RGB kanala, engl. ***R**ed **G**reen **B**lue*) i rezolucije (iako je teško očekivati sliku 4K rezolucije, generirana slika bi trebala biti razumne kvalitete). Ono što značajno utječe na količinu potrebnih resursa jest broj epoha za učenje. Epohe su termin u strojnom i dubokom učenju kojima se opisuje proces prolaska cjelokupnog skupa podataka kroz neuronsku mrežu za učenje. Veći broj epoha obično dovodi do bolje preciznosti modela, ali također produžuje vrijeme potrebno za učenje modela i zahtijeva više memorije za pohranu međupodataka koji se zadržavaju za svaku epohu. Dodatno, povećava se rizik od prenaučenosti te radi toga je važno koristiti tehniku poput ranog zaustavljanja (engl. *early stopping*) gdje se učenje zaustavlja kada se performanse na validacijskom skupu podataka prestanu poboljšavati [22]. Jasno je iz ovoga da broj epoha je jedan od hiperparametara koji se podešava tijekom učenja modela kako bi se postigla ravnoteža između prenaučenosti i nedostatka učenja (engl. *underfitting*) što je također moguće. Arhitektura generatora i diskriminatora mora biti prilagođena specifičnostima umjetničkih djela. Generator treba stvarati nove instance koje

oponašaju distribuciju stvarnih umjetničkih djela, dok diskriminator radi na prepoznavanju je li slika stvarna ili generirana. U ovoj GAN arhitekturi, generator bi imao konvolucijske transponirane slojeve (engl. *transposed convolutional layers*), odnosno dekonvolucijske slojeve, koji bi omogućavali generatoru stvaranje detaljnih slika, a u diskriminatoru bi konvolucijski slojevi (engl. *convolutional layers*) pomogli u prepoznavanju uzoraka i detalja po slici. Ovi slojevi se lako mogu dodati s pomoću *Keras* naredbi „Conv2D“ i „Conv2DTranspose“ [23]. „Conv2D“ je sloj koji provodi konvoluciju, temeljni sloj diskriminatora koji bi služio za prepoznavanje i učenje različitih značajki iz slika. Konvolucijski sloj radi tako što primjenjuje filtre na ulazne slike i stvara tenzor značajki koji sažima prisutne vizualne značajke poput rubova, tekstura i oblika [22]. Neki od ključnih parametara ovog sloja uključuju:

- „filters“ – broj izlaznih filtara u konvoluciji,
- „kernel_size“ – veličina 2D konvolucijskog prozora,
- „strides“ – koraci konvolucijskog prozora (veći koraci mogu smanjiti dimenzionalnost izlaznog tenzora),
- „padding“ – *valid* (bez dodavanja) ili *same* (dodavanje nula kako bi izlaz bio iste dimenzije kao ulaz) i
- „activation“ – funkcija aktivacije koja se primjenjuje na izlaz.

Za GAN, „Conv2D“ sloj u diskriminatoru pomaže u učenju razlikovanja između stvarnih i generiranih slika, identificirajući ključne značajke koje razlikuju jedne od drugih. „Conv2DTranspose“ omogućava stvaranje većih dimenzija izlaza iz manjih dimenzija ulaza, što je korisno u arhitekturi generatora u GAN mreži, gdje se želi povećati rezolucija slike. Parametri u ovom sloju su jednaki kao kod „Conv2D“ sloja. Ipak postoje potencijalni problemi s korištenjem „Conv2Dtranspose“ sloja, a to je tzv. *checkerboard* učinak – neželjeni uzorci u generiranim slikama koji podsjećaju na šahovsku ploču. Ovaj fenomen najvjerojatnije nastaje zbog načina na koji rade dekonvolucijski slojevi, pogotovo kada se koriste s neodgovarajućim veličina koraka (engl. *strides*) i veličinama kernela. Potencijalno rješenje je korištenje kombinacije slojeva „UpSampling2D“ i„Conv2D“: prvo se koristi „UpSampling2D“ za povećavanje dimenzije slike nakon čega slijedi „Conv2D“ sloj [24]. Ovaj pristup može smanjiti pojavu neželjenih artefakata i poboljšati vizualnu kvalitetu generiranih slika. Primjer slike gdje je vidljiv rezultat s primjenom „Conv2DTranspose“ i kombinacije „UpSampling2D“ – „Conv2D“ je prikazanom sljedećom slikom [Slika 13].



Slika 13. Razlika u kvaliteti slike: „Conv2DTranspose“ (gore) i „UpSampling2D“ – „Conv2D“ (dolje), [24]

Nastavljajući s daljnjim razmatranjem arhitekture generativnih suparničkih mreža, važno je naglasiti ulogu i funkciju ostalih slojeva te kako se oni uklapaju u strukturu generatora i diskriminatora za problem generiranja slika:

- **Sequential** – Temeljni gradbeni sloj za generator i diskriminator, omogućuje linearno stvaranje modela sloj po sloj. Ovo je posebno korisno za brzo slaganje arhitekture modela što je korisno za GAN budući da se obično generator i diskriminator grade kao dva odvojena „Sequential“ modela.
- **Dense** – Ključni sloj u svakoj neuronskoj mreži, omogućuje „hvatanje“ kompleksnih značajki i oblika podataka. U kontekstu GAN-a, „Dense“ slojevi se koriste na početku generatora kako bi se proširio ulazni šum (latentni vektor) u veću dimenzije, koja se zatim dalje obrađuje kroz mrežu. U diskriminatoru se „Dense“ sloj koristi na kraju za smanjenje izvedenih značajki na jedan izlaz koji pokazuje vjerojatnost je li slika stvarna ili lažna.
- **Reshape** – Ovaj sloj se koristi za promjenu oblika podataka bez mijenjanja njihovog sadržaja te se obično koristi nakon prvog „Dense“ sloja kako bi se pretvorio u potrebnu dimenziju prije ulaza u prvi konvolucijski sloj.

- **BatchNormalization –** Sloj koji se koristi za normalizaciju izlaza prethodnog sloja kako bi izlaz imao srednju vrijednost blizu nule (0) i standaradnu devijaciju blizu vrijednosti jedan (1) čime se često poboljšava stabilnost i performanse mreža tijekom učenja.
- **Dropout –** Česta tehnika u sprječavanju prenaučenosti je isključivanje određenih neurona u mreži tijekom procesa učenja. Ovime se smanjuje ovisnost o određenim putevima unutar mreže i potiče mrežu za robusnija rješenja. Ovo se može postići korištenjem „Dropout“ sloja te se obično koristi u diskriminatoru kako bi se povećala sposobnost generalizacije i smanjila pristranost prema specifičnim obilježjima generiranih slika.
- **Flatten –** „Flatten“ je sloj koji transformira višedimenzionalni izlaz prethodnih slojeva u jednodimenzinalni vektor čime se omogućuje da diskriminator donese odluku o tome je li ulazna slika stvarna ili generirana.

Postoji još slojeva koji se mogu koristiti, ali ovi navedeni se uvijek koriste u izgradnji robusnog i djelotvornog GAN modela. Pravilno balansiranje ovih slojeva i njihova integracija u arhitekturu generatora i diskriminatora su ključni za uspjeh GAN-a u generiranju realističnih slika. U trećem poglavlju, posebna pažnja bit će posvećena detaljnijoj analizi arhitekture generatora i diskriminatora.

2.4. Strategija učenja i optimizacije

Strategija učenja je temelj za svaki strojni model, uključujući GAN-ove. Nakon odabira odgovarajućeg skupa podataka i predobrade (npr. normalizacija) važno je odrediti kako će izgledati učenje (treniranje) GAN-a. Najveći izazov u ovome dolazi od činjenice da su generator i diskriminator istovremeno učeni, odnosno da poboljšanja u jednom modelu dolaze nauštrb drugog modela. No, iako ne postoji generalni konsenzus u vezi „pravilnog" načina učenja generativnih suparničkih mreža, kroz godine se ispostavilo nekoliko praksi koje su davale zadovoljavajuće rezultate. Neke od ovih su već ranije spomenute: poput upotrebe LeakyReLU aktivacijske funkcije. LeakyReLU dopušta prolaz nekih negativnih vrijednosti, što može poboljšati učenje i preporučljivo je s vrijednošću od 0,2 [12, 25]. Nadalje, normalizacija serije (engl. *batch normalization*) uveliko stabilizira proces učenja, posebno kada se koriste GAN strukture s dubokim konvolucijskim mrežama [12]. Ideja iza ovoga je smanjivanje unutarnjeg pomaka kovarijanti (engl. *internal covariate shift*) – promjena u distribuciji podataka tijekom učenja nastala promjenom parametara u prethodnim slojevima [26]. Normalizacija serije u GAN-u se može primjenjivati nakon aktivacijske funkcije u konvolucijskim i transponiranim konvolucijskim slojevima za diskriminator i generator. Ovime se održava stabilnost distribucije ulaznih podataka kroz različite slojeve mreže, što olakšava učenje i povećava učinkovitost učenja GAN modela. Daljnja preporuka je i ručna inicijalizacija parametara modela s malim nasumičnim vrijednostima, a specifično kod ponovno DCGAN arhitekture (koje se pokazalo kao vrlo dobro rješenje za generiranje slika [11]) je korištena Gaussova, normalna distribucija sa standardnom devijacijom od 0,02 [25]. Korištenjem ove distribucije težine su raspoređene na takav način koji omogućuje mreži učenje različitih značajki iz podataka bez značajnog rizika od prebrzog zasićenja podataka. Ovakvim pristupom inicijalizaciji, svaki sloj u modelu ima težine koje su dobro postavljene za učenje, čime se uspostavlja ravnoteža između generatora i diskriminatora koja inače može biti vrlo osjetljiva. Ono što je također važno jest da diskriminator prima konzistentan ulaz iz generatora – odnosno, da je raspon piksela isti. Ovo se postiže time da aktivacijska funkcija na zadnjem sloju generatora bude tangens hiperbolni (th, tanh) čime se osigurava da diskriminator prima slike s vrijednošću piksela u rasponu [-1, 1]. Naravno, ovo je jedino korisno ako bi se i stvarne slike skalirale da imaju isti raspon vrijednosti piksela. Ovo je lako za izvesti s pomoću ranije spomenute *NumPy* biblioteke u *Python* jeziku. Pored ovoga, određivanje funkcije gubitka je jednako važan aspekt u procesa učenja GAN-a. Najčešća funkcija gubitka koja se koristi za generativne suparničke mreže je

funkcija binarnog unakrsnog entropijskog gubitka (engl. ***B**inary **C**ross-**E**ntropy loss,* BCE) u kojoj se mjeri razlika između predviđanja i stvarne oznake – idealno za diskriminator koji radi klasifikaciju podataka. Međutim, nerijetko ova funkcija gubitka dovodi do problema zasićenja, gdje diskriminator postane previše siguran u svoja predviđanja što dovodi do nestabilnog učenja. Zbog toga, postoje alternative kao već ranije spomenuta Wassersteinova udaljenost (minimizacija udaljenosti između stvarne i lažne distribucije podataka). Još jedan važan aspekt u učenju GAN-a je balansiranje procesa učenja generatora i diskriminatora. Prije svega, diskriminator se obično brže uči od generatora jer najčešće ima jednostavniju strukturu te je radi toga važno uskladiti i optimizirati hiperparametre strojnog modela. Strategija učenja se više odnosi na metodu i pristup koji se koristi za treniranje modela, dok optimizacija se odnosi na finu prilagodbu hiperparametara unutar tog okvira. Jedan od tih je već ranije spomenuti broj epoha. No, osim broja epoha, postoji niz hiperparametara koje treba uzeti u obzir prilikom optimizacije GAN-a. Za početak je važno odlučiti koliku seriju (engl. *batch size*) koristiti. U kontekstu strojnog učenja, serija označava broj uzoraka podataka koji se obrađuju u jednom prolasku kroz mrežu tijekom učenja. Primjerice, ako je ukupan broj uzoraka u skupu podataka 5000, a veličina serije je 500, broj iteracija potrebnih za završetak jedne epohe izračunava se kao ukupan broj uzoraka podijeljen s veličinom serije. Ovo se može izraziti sljedećom formulom:

$$Broj\ iteracija\ po\ epohi = \frac{Ukupan\ broj\ uzoraka}{Veličina\ serije} \tag{2.4.1}$$

Velika serija može omogućiti stabilnije i brže učenje modela jer omogućava mreži da vidi više primjera prije nego što ažurira težine s tim da to zahtijeva više memorije.

Nakon odabira veličina serije, sljedeći ključan hiperparametar je brzina učenja (ili stopa učenja). Ovo utječe na brzinu ažuriranja težina mreže tijekom procesa učenja. Cilj pažljivim biranjem brzine učenja jest optimizirati model da konvergira prema optimalnim parametrima bez ikakve nestabilnosti. Konvergencija bi značila da model „stabilizira“ i prestaje značajno mijenjati svoje parametre – ovo bi ukazivalo na to da je pronađena dovoljno dobra aproksimacija za rješavanje zadanog problema. Previsoka brzina učenja može dovesti do nestabilnosti i spriječiti konvergenciju, dok preniska brzina uzrokuje spor napredak ili zaglavljivanje modela u lokalnim minimumima. U kontekstu GAN-a, točno podešavanje brzine učenja bi omogućilo da gubitak modela prestane značajno opadati između iteracija učenja, a

točnost modela postaje stabilna ili prestane značajno rasti. Ovo bi značilo da dodatno učenje neće značajno poboljšati performanse generatora i diskriminatora. Ipak, u praksi postizanje konvergencije je izazovno za kompleksne modele poput GAN-a i često se treba mijenjati i testirati zbog svoje inherentne nestabilnosti. Zbog ovoga je često potrebno provoditi kontinuiranu prilagodbu i testiranje kako bi se osiguralo uspješno učenje.

Svakako uz navedeno je važan i izbor odgovarajućeg optimizatora – algoritam koji će se koristiti za ažuriranje težina mreže na osnovnu izračunatog gradijenta gubitka. Različiti optimizatori mogu imati značajan utjecaj na brzinu i stabilnost procesa učenja. Među popularnim optimizatorima u učenju GAN-a su „Adam“ (engl. ***A**daptive **M**oment Estimation optimizer*), „RMSProp“ (engl. ***R**oot **M**ean **S**quared **P**ropagation*) i „SGD“ (engl. ***S**tohastic **G**radient **D**escent*) [23].

- **Adam** – Jedan od najpopularnijih optimizatora zbog svoje učinkovitosti u brzom konvergiranju težina modela. „Adam“ kombinira prednosti „RMSProp“ i „SGD“ optimizatora; s „beta_1“ parametrom (slično momentumu u „SGD“ optimizatoru) i „beta_2“ parametrom (slično momentumu u „RMSProp“ optimizatoru). Ovaj optimizator se ističe svojom računalnom učinkovitošću, malim zahtjevima za memorijom, invarijantnošću na dijagonalno skaliranje gradijenta i posebno je pogodan za probleme s velikim količinama podataka ili parametara [23]. Preporučena vrijednost stope učenja za „Adam“ optimizator je 0,0002, a „beta_1“ parametra 0,5 [25].
- **RMSProp** – Osnovna ideja iza ovog optimizatora je prilagodbe stope učenja za svaku težinu u mreži neovisno. Ovo se radi u dva koraka: prvi je održavanje eksponencijalne prosječne vrijednosti kvadrata gradijenta za svaku težinu, a drugi je dijeljenje gradijenta svake težine s korijenom prosječne vrijednosti [23]. Ovime se postiže efektivna prilagodba stope učenja za svaku težinu mreže, omogućavajući brže i stabilnije učenje. Ovaj optimizator je preporučljivo koristiti u situacijama s nestabilnim ili oscilirajućim gradijentom. Dodatno, „RMSProp“ pomaže u mogućem problemu GAN-a sa zaglavljivanjem u lokalnim minimumima zbog preniske brzine učenja.
- **SGD** – Klasični optimizacijski algoritam koji koristi metodu gradijentnog spusta za ažuriranje težina modela. Zbog svoje jednostavnosti, efikasnosti i manje agresivnog ažuriranja težina naspram „Adam“ optimizatora, „SGD“ se može koristiti kada postoji potreba za stabilnošću kod kompleksnijih GAN struktura.

Premda navedeni optimizatori su se pokazali kao vrlo dobri u rješavanju većina GAN problema, to ne treba biti pravilo. Nema jedinstvenog, „najboljeg“ optimizatora za sve vrste

GAN modela već izbor optimizatora ovisi o specifičnim karakteristikama problema, strukturi modela kao i skupu podataka koji se koristi. Eksperimentiranje s različitim optimizatorima (kao i sa sukladnim hiperparametrima) pruža bolji uvid u koji optimizator bi najbolje funkcionirao za određenu generativnu suparničku mrežu. U konačnici, strategija učenja i optimizacija za GAN-ove često zahtijeva metodu pokušaja i pogreške koji uključuje razumijevanje dinamike između generatora i diskriminatora, kao i pažljivo biranje različitih komponenti učenja, odgovarajućih hiperparametara i optimizacijskih metoda kako bi se postigli željeni rezultati.

3. IMPLEMENTACIJA

3.1. Izgradnja GAN modela

Sa svim potencijalnim pristupima u izgradnji generativne suparničke mreže za generiranje umjetničkih slika, s obzirom na kompleksnost zadatka, potrebno je pomno strukturirati i prilagoditi cjelovitu strukturu mreže. Prije svega, prvi korak je uspostaviti vezu s *Colab* platformom, uključujući instalaciju neophodnih biblioteka poput *TensorFlow* i *Keras*. Ovo se može uzeti kao „nulti“ korak ili preduvjet (engl. *prerequisite*) u ovom procesu – uvoz svih potrebnih biblioteka. Dalje bi slijedila inicijalizacija svih varijabli, što bi uključivalo određivanje puta do skupa podataka, hiperparametara poput broja epoha ili brzine učenja. Nakon inicijalizacije, važan korak u procesu izgradnje GAN-a je učitavanje i predobrada skupa podataka ako je potrebna. Skup podataka impresionističkih slika sadrži umjetnička djela različitih autora te se mora standardizirati i prilagoditi za GAN model. To bi uključivalo normalizaciju slike, skaliranje na uniformnu veličinu i transformaciju u format prikladan za obradu GAN modelom. Konkretno, potrebno je smanjiti veličine slika na 64x64 piksela. Ovo je neophodan korak kako bi se standardizirale dimenzije ulaznih podataka za mrežu, a uniformnost veličine slika je preduvjet za učinkovito treniranje u strojnom učenju. Dodatno ovime se olakšava procesiranje i smanjuje se potrebna računalna snaga. Zatim su vrijednosti piksela normalizirane u rasponu od -1 do 1 radi kompatibilnosti s tangens hiperbolnom aktivacijskom funkcijom te činjenice kako bi ovo poboljšalo proces učenja – normalizacija podataka u rasponu od -1 do 1 pomaže u centriranju podataka oko nule, što dovodi do bolje konvergencije tijekom treniranja. Ovaj cijeli postupak se može definirati u jednoj funkciji (nazvana „preprocess_image“) te primijeniti na cijeli skup podataka na sljedeći način:

```
def preprocess_image(image):
    # Promjena veličine slike na 64x64
    image = tf.image.resize(image, [64, 64])
    # Normalizacija pixel vrijednosti na raspon [-1, 1]
    image = (tf.cast(image, tf.float32) / 127.5)
    return image

# Primjena predobrade na svaku sliku u datasetu
artstyle_ds_tf = artstyle_ds_tf.map \
(lambda x: {'images': preprocess_image(x['images']), 'labels': x['labels']})
```

Na sljedećoj stranici su prikazane slike iz skupa podataka nakon predobrade [Slika 14].

Slika 14. Slike iz skupa podataka nakon predobrade

Nakon primjene ovih transformacija skup podataka se dodatno obrađuje sljedećim redoslijedom: kaširanje (pohranjivanje podataka za brži pristup, engl. *caching*), miješanje (engl. *shuffling*) kako bi se smanjila pristranost tijekom učenja, grupiranje u serije (engl. *batching*) kako bi učenje bilo učinkovitije i unaprijedno dohvaćanje (engl. *prefetching*) poboljšava performanse time što podatke priprema unaprijed dok se trenutna iteracija obrađuje. Ovo se smatra standardnom procedurom u strojnom i dubokom učenju te su ti koraci definirani sljedećim linijama kôda:

```
# Ostali potrebni koraci: cache, shuffle, batch i prefetch
artstyle_ds_tf = artstyle_ds_tf.cache()
artstyle_ds_tf = artstyle_ds_tf.shuffle(buffer_size=13060)
artstyle_ds_tf = artstyle_ds_tf.batch(32)
artstyle_ds_tf = artstyle_ds_tf.prefetch(tf.data.AUTOTUNE)
```

Korištenjem „tf.data.AUTOTUNE“ *TensorFlow* dinamički podešava vrijednost konfiguracijske opcije kako bi se optimizirale performanse – u ovom slučaju to je za unaprijedno dohvaćanje.

Zatim, slijedi proces definiranja arhitekture generatora i diskriminatora – koristeći kombinaciju konvolucijskih, dekonvolucijskih i drugih slojeva kako bi se postigla željena razina detalja i realističnosti generiranih slika. Struktura diskriminatora i posebno generatora treba uzeti u obzir neophodnu ravnotežu broja parametara po modelu: premalo i neće se moći generirati slike koje nalikuju na impresionizam, a ako ih je previše, uz rizik prenaučenosti i nestabilnog treniranja, može doći i do nepotrebnog trošenja resursa. Optimalna struktura generatora obično uključuje slojeve koji postupno povećavaju razlučivost slike, dok se aktivacijske funkcije poput ReLU i tanh koriste za modeliranje nelinearnosti i ograničavanje vrijednosti piksela. Diskriminator pak ne mora biti toliko kompleksan, obično se oslanja na konvolucijske slojeve za izvlačenje značajki iz slika te na kraju klasificira istu kao pravu ili

generiranu. Uspostava precizne arhitekture zahtijeva eksperimentalni pristup te u većini slučajeva je odlučujući čimbenik o mogućnosti uspjeha – iterativno prilagođavanje slojeva, filtara i veličina kernela je ključno za postizanje optimalnog performansa generativne suparničke mreže. U implementaciji odgovarajućeg GAN modela za generiranje slika, strukture generatora i diskriminatora sadrže slojeve navedene u drugom poglavlju s par izmjena radi prilagodbe specifičnosti zadatka. U prilogu ovog rada je dodan cijeli kôd kojim je razvijen generator i diskriminator za treniranje GAN modela, uključujući vizualni prikaz arhitekture generatora i diskriminatora.

Sljedeća tablica [Tablica 2] predstavlja usporedbu parametara arhitektura GAN modela gdje u redovima su navedeni parametri po modelu. Ovi parametri uključuju sve težine i pristranosti (engl. *bias*) u svim slojevima modela. Promjenjivi parametri (engl. *trainable parameters*) su takvi parametri čije se vrijednosti mogu promijeniti; bilo kroz propagiranje pogreške unatrag ili putem optimizacijskog algoritma (poput „Adam“ ili „SDG“). Nepromjenjivi parametri su parametri koje se ne mijenjaju tijekom procesa treniranja. Obično to uključuje parametre koji su ostali fiksirani prije treniranja, npr. težine iz prethodno treniranog modela ili težina u slojevima za normalizaciju („BatchNormalization“ sloj). Ovi parametri mogu biti vrlo korisni u slučaju da je potrebno model prilagođavati novim podatcima što je uobičajeno tijekom procesa treniranja GAN-a.

Tablica 2. Usporedba parametara generatora i diskriminatora izgrađenog GAN modela

	Generator	**Diskriminator**
Ukupan broj parametara	3 075 971	1 613 889
Promjenjivi parametri	3 075 075	1 611 969
Nepromjenjivi parametri	896	1 920

Uočljivo je kako generator ima gotovo dvostruko više parametara od diskriminatora što je i logično budući da generator ima „teži“ zadatak od diskriminatora s učenjem složenih uzoraka boja i kompozicija u slikama. Iako je diskriminator jednostavniji naspram generatora, mora biti dovoljno „sposoban“ za razlikovanje generirane slike od stvarnih – što zahtijeva preciznu ekstrakciju značajki korištenjem niza konvolucijskih slojeva. Važno je još naglasiti kako u razvoju ovakvih modela se često eksperimentira s različitim arhitekturama te prvotna ne mora biti i najbolja. Ipak, iz konteksta treniranja, bolje je pridržavati se jednom zadanog oblika

strukture, budući da promjena strukture tijekom treniranja može stvoriti niz problema. Prvi od problema je tzv. „katastrofalno zaboravljanje“ (engl. *catastrophic forgetting*) , gdje neuronska mreža zaboravlja prethodno naučene značajke kada se suviše promijeni struktura koja se koristi za treniranje [4]. Ovo negativno utječe na sposobnost mreže da generalizira i nauči nove značajke dovodeći do kolaps modela gdje generator počne proizvoditi vrlo ograničen spektar uzoraka. Još jedan problem jeste manjak reproducibilnosti. U kontekstu generatora, promjena u broju slojeva može značajno promijeniti distribuciju generiranih slik, čime se gubi na konzistentnosti u kvaliteti slika. Stoga je preporučljivo temeljito testiranje i optimiziranje strukture prije početka intenzivnijeg treniranja, kako bi se osiguralo da su arhitektura i hiperparametri adekvatno postavljeni te da je minimalan zahtjev za promjenom tijekom treniranja.

U narednom koraku se uspostavlja način učenja mreže, optimizacija težina, parametara i prigodna funkcija gubitka koja bi omogućila „optimalno“ generiranje slika kroz fazu učenja. Osim što je najvažniji, ovo je ujedno i programski najteži korak s velikom količinom parametara za uzeti u obzir. Prema tome važno je detaljno isplanirati strukturu ovog dijela kôda te se u ovom dijelu programa rade sljedeći koraci:

1. Postaviti optimizatore i funkcije gubitka za generator i diskriminator:
    - Koristi se „Adam“ optimizator za oba modela s različitim stopama učenja.
    - Za funkciju gubitka koristi se funkcija binarnog unakrsnog entropijskog gubitka, odnosno „BinaryCrossentropy“.
2. Definiranje klase „PaintingGAN“:
    - Definiranjem klase za upravljanje generatorom i diskriminatorom (nazvano „PaintingGAN“) omogućuje se koordinirano treniranje oba modela.
3. Inicijalizacija i kompajliranje:
    - Generator i diskriminator se inicijaliziraju te zatim kompajliraju s odabranim optimizatorom i funkcijom gubitka.
4. Definiranje procesa treniranja:
    - Proces treniranja definiran unutar metode, koja obuhvaća korake potrebne za treniranje GAN-a na jednoj seriji podataka.
5. Generiranje lažnih slika:
    - Korištenjem latentnog vektora s nasumičnim vrijednostima iz normalne distribucije („tf.random.normal“) stvaraju se lažne slike.

6. Treniranje diskriminatora:
    - Diskriminator se trenira na kombinaciji pravih i lažnih slika. Potencijalni dodatak malog šuma radi povećanja robusnosti modela.
    - Izračunava se gubitak diskriminatora te se ažuriraju parametri koristeći izračunati gradijent.
7. Treniranje generatora:
    - Treniranje generatora na temelju kako dobro generirane slike „varaju“ diskriminator.
    - Izračunava se gubitak generatora te se ažuriraju parametri.
8. Povratne vrijednosti:
    - Na kraju svakog koraka treniranja vraćaju se izračunati gubitci za generator i diskriminator.

Ovaj postupak omogućuje iterativno poboljšavanje sposobnosti generatora da stvara uvjerljive slike dok istovremeno trenira diskriminator da postane efikasniji u razlikovanju između stvarnih i generiranih slika. Na kraju, cilj je postići ravnotežu gdje generator stvara realistične slike koje diskriminator ne može lako klasificirati kao lažne.

U prilogu se nalazi cjelovito rješenje za ovaj dio rada, a ispod je izdvojen dio kôda za treniranje diskriminatora radi dubljeg objašnjenja logike koja se odnosi na gubitke diskriminatora (definirano kao „d_loss“) i generatora (definirano kao „g_loss“).

Za generator to izgleda na sljedeći način:

```
# Treniranje generatora
with tf.GradientTape() as g_tape:
    random_latent_vectors = tf.random.normal(shape=(batch_size, 128))
    misleading_labels = tf.ones((batch_size, 1))

    generated_images = self.generator(random_latent_vectors, training=True)
    predictions = self.discriminator(generated_images, training=False)
    g_loss = self.g_loss(misleading_labels, predictions)
```

U ovom dijelu, generator stvara slike kojima će pokušati „zavarati“ diskriminator da ih pogrešno klasificira kao stvarne. Varijabla „g_loss“ predstavlja gubitak generatora na temelju sposobnosti da „prevari“ diskriminator.

Za diskriminator, gubitak je definiran na sljedeći način:

```
# Treniranje diskriminatora
with tf.GradientTape() as d_tape:
    # Generiranje lažnih slika
    fake_images = self.generator(random_latent_vectors, training=True)

    # Prave i lažne slike
    real_images = batch['images']
    combined_images = tf.concat([real_images, fake_images], axis=0)

    # Etikete i šum
    labels = tf.concat([tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))], axis=0)
    labels += 0.05 * tf.random.uniform(tf.shape(labels))

    # Predikcije i gubitak
    predictions = self.discriminator(combined_images, training=True)
    d_loss = self.d_loss(labels, predictions)
```

Ovdje se pak diskriminator trenira tako da razlikuje stvarne i generirane slike, a „d_loss“ predstavlja gubitak diskriminatora u odnosu na njegovu sposobnost ispravnog klasificiranja slika kao stvarnih ili generiranih. Tehnika poznata kao *label smoothing* se koristi kako bi se poboljšala generalizacija i spriječila prenaučenost [4, 25]. Time se podrazumijeva korištenje „mekih“ oznaka umjesto „tvrdih“ – primjerice, umjesto da se stvarnim slikama pridaje oznaka od jedan, može biti u rasponu $\pm 0{,}3$. Ovime se potiče diskriminatora da bude manje siguran u svoje klasifikacije slika. U praksi se ovo radi dodavanjem slučajnog šuma na oznake.

Nakon početne inicijalizacije, prikupljanje podataka iz skupa, predobrade, definiranje GAN strukture te definiranje svih preostalih komponenti, ostaje još pokrenuti proces treniranja. Ovo se radi preko „.fit()“ metode – standard u *TensorFlowu* i *Kerasu* za treniranje modela [23]. S time se može stati, ali kako je već ranije navedeno, treniranje GAN-a je izrazito zahtjevno i dodatno zahtijeva prilagođavanje hiperparametara. Ponekad u tijeku treniranja potrebno je imati brzu povratnu informaciju stanja učenja generativne suparničke mreže. Radi toga je razvijen poseban dio kôda koji bi igrao ključnu ulogu u praćenju napretka GAN modela tijekom treniranja – zasebna *callback* funkcija (poseban tip funkcije koja se koristi da izvršava određene radnje tijekom procesa) koja bi služila za vizualizaciju i ocjenu uspješnosti generatora i diskriminatora. Ova funkcija, nazvana „ModelMonitor“, bi služila kao „prozor“ u trenutni rad generatora time što bi nakon svake epohe spremila definiran broj slika, time dajući najvrjedniji uvid u to kako model napreduje s vremenom. Na temelju generiranih slika nakon svake epohe, može se razlučiti je li potrebno zaustaviti treniranje, promijeniti parametre i nastaviti ili je trenutni status zadovoljavajući. Uz to se svakako nakon svake epohe prikažu vrijednosti

gubitaka diskriminatora i generatora, pružajući još jedan važan kontrolni parametar kojim se može pratiti status treniranja modela. Ovime se, imajući u vidu ranije definiranih način računanja gubitaka, može izvesti općenita procedura u slučaju da model prikazuje nezadovoljavajuća svojstva – što je i definirano u tablici ispod [Tablica 3].

Tablica 3. Interpretacija gubitaka i mogući postupci

Tip gubitka	Vrijednost gubitka	Interpretacija	Mogući postupak
„d_loss“	< 0,5	Diskriminator dobro radi, ispravno klasificira stvarne i lažne slike.	Smanjenje stope učenja diskriminatora ili blago povećanje stope učenja generatora.
„d_loss“	> 0,5	Diskriminator loše radi, teško razlikuje stvarne i lažne slike.	Povećanje stop učenja diskriminatora ili smanjenje kod generatora.
„g_loss“	< 0,5	Generator uspješno vara diskriminator.	Ako nastavi padati, mogući pokazatelj prenaučenosti. Smanjiti brzinu učenja generatora.
„g_loss“	> 0,5	Generator nije efikasan u varanju diskriminatora.	Povećanje brzine generatora ili blago smanjivanje stope učenja diskriminatora.

Ipak, važno je istaknuti da su ovo samo općenite upute te nisu pravilo. Najvažnija vrijednost su slike koje se generiraju – u slučaju da je jedan gubitak u domeni koja bi se mogla smatrati „opasnom“, ali slike koje se generiraju drže kvalitetu ili čak napreduju, onda nema potrebe za ikakvim promjenama. Postoji još niz faktora koji određuju koliko će generativna suparnička mreža biti uspješna – ovo su samo neki općeniti koji su poslužili tijekom procesa treniranja GAN-a za generiranje umjetničkih slika u ovom radu.

3.2. Izazovi i rješenja tijekom implementacije

Tijekom izgradnje GAN modela moguće je naići na razne poteškoće koji utječu na učinkovitost i kvalitetu generiranih slika. Većinom ovi izazovi su vezani uz balansiranje suparničkog odnosa generatora i diskriminatora: ako jedan od njih postane previše dominantan, to dovodi do problema s učenjem modela. No, kao što je već ranije opisano, na performans mreže utječe izbor hiperparametara – često je potrebno mijenjati i prilagođavati vrijednosti kako bi se dobili zadovoljavajući odzivi, tj. slike. Dodatni problemi bi mogli proizaći iz skupa podataka ako se ispostavi da isti ne sadrži dovoljno kvalitetne ili raznolike podatke. Nadalje, izazov predstavlja i održavanje kvalitete slika tijekom skaliranja, što je bitno za realističan prikaz umjetničkih djela. Tijekom treniranja GAN modela nastalo je nekoliko problema. Prije svega i s dostupnim procesorskim jedinicama preko *Google Colab* platforme, treniranje GAN modela je bilo previše zahtjevno i dovelo je do preopterećenja GPU-a, što je uzrokovalo pad programa (engl. *crash*). Radi toga je napravljeno nekoliko koraka kako bi se ovaj problem minimizirao:

- Smanjenje veličine serije (*batch size*) sa 64 na 32.
- Smanjenje rezolucije generiranih slika sa 128x128 piksela na 64x64 piksela.

Kako bi se omogućilo konstantno unapređivanje trening ciklusa te da se pomaci u generiranje realističnijih slika ne bi izgubili, definira se nova *callback* funkcija („GANCheckpoint“) koja je definirana ispod:

```
from tensorflow.keras.callbacks import ModelCheckpoint

class GANCheckpoint(Callback):
    def __init__(self, generator, discriminator, filepath, save_freq=10):
        super GANCheckpoint  self).__init__()
        self.generator = generator
        self.discriminator = discriminator
        self.filepath = filepath
        self.save_freq = save_freq

    def on_epoch_end(self, epoch, logs=None):
        if self.save_freq == 'epoch' or (epoch + 1) % self.save_freq == 0:
            gen_path = os.path.join(self.filepath,\
f"generator_epoch_{epoch+1}.h5")
            disc_path = os.path.join(self.filepath,\
f"discriminator_epoch_{epoch + 1}.h5")
            self.generator.save(gen_path)
            self.discriminator.save(disc_path)
```

U *Keras* biblioteci postoji ugrađena *callback* funkcija nazvana „ModelCheckpoint“ koja služi za automatsko spremanje *Keras* modela tijekom treninga. Ova funkcija je posebno korisna jer

omogućuje spremanje modela u određenim intervalima ili u slučaju da je određeni kriterij ispunjen. Ipak, potreban je specifičan dizajn za GAN modele koji imaju odvojene komponente koje se trebaju nezavisno spremati. Pozivanjem te funkcije potrebno je još inicijalizirati modele generatora i diskriminatora, putanju za spremanje i frekvenciju spremanja (tijekom treniranja ovo se najčešće mijenja); kao što je to napravljeno ispod:

```
checkpoint_cb = GANCheckpoint(
    generator=generator,
    discriminator=discriminator,
    filepath=''  # Putanja prema mapi za spremanje
    save_freq=50 # Spremi modele svakih X epoha
)
```

Dodatno, *Google Colab* platforma nudi mogućnost sinkronizacije s *Google Drive* korisničkim računom te je prema tome preporučljivo omogućiti pristup istom zato što postoji mogućnost gubitka podataka koji su spremljeni lokalno na *Google Colab* platformi. S ovim model treniranja GAN-a je robusniji na nepredviđene padove. Redovitim spremanjem modela može se nastaviti trening s posljednjim spremljenim modelom ili eksperimentirati s modelima iz različitih faza treninga.

Sljedeći problem jest mogućnost prenaučenosti modela. Naime, treniranje kompleksnih modela kao što su generativne suparničke mreže zahtijeva uz resurse i vremena. Primjerice, trajanje treniranja skupa podataka impresionističkih slika na 2000 epoha je otprilike sedam sati. U tom vremenu potrebno je kontinuirano pratiti trening modela za eventualne pokazatelje kolapsa modela ili prenaučenosti. Kolaps modela je lako primijetiti - jedan od prvih pokazatelja je generiranje istih slika nakon svake epohe, ali prenaučenost nema uvijek jasne pokazatelje. Zbog toga, kako bi se automatiziralo praćenje modela koristi se „EarlyStopping“ funkcija koja prekida trening kada model više ne postiže poboljšanja:

```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping_cb = EarlyStopping(
    monitor='d_loss',  # Monitoring gubitka diskriminatora
    patience=40, # Broj epoha bez poboljšanja nakon kojeg se trening
automatski prekida
    restore_best_weights=True,  # Vraćanje modela na najbolje težine
    start_from_epoch=150 # Počni pratiti nakon 150-te epohe
)
```

U inicijalizaciji ove funkcije parametar „monitor“ određuje metriku koju će funkcija pratiti (u ovom slučaju to je gubitak diskriminatora) [23]. „Patience“ određuje koliko epoha se može nastaviti bez poboljšanja praćenog pokazatelja prije nego što se automatski prekine [23].

Sljedeći parametar, „restore_best_weights“, omogućuje da težina modela se vrati na one vrijednosti koje su imale najbolji praćeni pokazatelj [23]. Ovo znači da će se zadržati najbolje stanje modela čak i ako se trening prekine zbog nedostatka napretka. Zadnji parametar u definiranoj funkciji, „start_from_epoch“ određuje koliko epoha funkcija „čeka“ prije nego što je upaljena za nadgledanje [23]. Ovo je korisno zbog toga što često GAN modeli imaju nepredvidiva ponašanja u početku novog trening ciklusa: vrijednosti gubitaka mogu divergirati ili se ponašati nestabilno dok model ne uđe u stabilniju fazu učenja. S tim su definirane sve *callback* funkcije koje su se pokazale izuzetno korisnim tijekom faze treniranja GAN modela. Potrebno je iste još inicijalizirati u ranije spomenutoj „.fit()“ metodi:

```
hist = painting_gan.fit(
    artstyle_ds_tf,
    epochs=500,
    callbacks=[ModelMonitor(num_img=4, latent_dim=128), checkpoint_cb,
early_stopping_cb]
)
```

te je ovim sada omogućena lakša analiza i prilagodba treninga bez potrebe za stalnim ručnim nadzorom.

Sljedeći mogući problem je već mnogo puta spomenuti kolaps modela. Tijekom treniranja GAN-a generator bi počeo proizvoditi ograničen raspon izlaza, bez obzira na varijabilnost ulaznog šuma. Drugim riječima, proizvodile bi se slične ili gotovo identične slike za različite ulazne podatke. Na sljedećoj stranici se nalazi primjer kolapsa modela, nastao tijekom rane faze treniranja (generirane četiri nasumične slike) [Slika 15].

Nekoliko je potencijalnih uzroka zbog ovog:

- preniska varijabilnost ulaznog šuma,
- neadekvatno učenje generatora,
- dominacija diskriminatora nad generatorom,
- nedostatak raznolikosti podataka,
- prekompleksna arhitektura.

U ovom slučaju uzrok je bilo zadnje navedeno; inicijalna struktura generatora i diskriminatora je bila prekompleksna. Ako je generator previše složen u odnosu na diskriminator to dovodi do situacije gdje je diskriminator neučinkovit, omogućujući generatoru da s bilo kojom slikom (bez obzira na kvalitetu) prođe klasifikaciju diskriminatora. S druge strane, ako je diskriminator previše složen može postati previše dobar u razlikovanju pravih i generiranih slika. Time se

dovodi generator u situaciju gdje konvergira prema samo jednoj vrsti slike ili vrlo ograničenoj skupini slika.

Slika 15. Primjer kolapsa modela tijekom procesa treniranja

Pojednostavljenjem strukture je ovaj problem riješen (opisana struktura GAN modela u drugom poglavlju). U pojednostavljenoj strukturi je dodan još i ranije spomenuti „BatchNormalization“ sloj koji uvodi dodatnu stabilnost u procesu učenja sa skaliranjem i centriranjem ulaza u svakom sloju mreže. Ovime se pomaže s promjenom distribucije ulaznih podataka što omogućuje efikasnije i brže učenje jer mreža ne treba neprestano prilagođavati svoje težine promjenjivim distribucijama ulaza.

Međutim, pojednostavljenjem strukture generatora i diskriminatora predstavlja se novi problem gdje, u smislu broja slojeva odnosno broja neurona po slojevima, model nema dovoljno

kapaciteta za napredak. Dostiže se maksimum u sposobnosti naučenih obrazaca iz podataka. Imajući u vidu kako učenje modela ne može nastaviti u slučaju promjene strukture, potrebno je prilagoditi ono što se može: stope učenja, optimizacijski algoritam ili funkcija gubitka. Konkretno, nakon određenog broja epoha bi se stopa učenja prepolovila što je omogućilo GAN-u da prepoznaje detaljnije karakteristike slika. Ovo je valjano rješenje, ali ključni pomak u daljnjem razvoju slika je bila promjena optimizatora iz „Adam“ u „SGD“ koji, iako sporiji, je dosljedniji u konvergenciji čime se potiče bolja generalizacija i izbjegavaju se prevelike varijabilnosti u generiranim slikama.

S ovim modifikacijama može se nastaviti trenirati model uz i dalje uključene *callback* funkcije za nadgledanje i redovitim praćenjem generiranih slika za indikacije napretka ili nazadovanja.

3.3. Integracija modela u „web“ sučelje

Kao završni korak u ovom diplomskom radu, potrebno je integrirati razvijeni GAN model u „web“ stranicu, gdje je važno razviti sučelje koje će omogućiti učitavanje slika i interakciju s korisnikom. Obično za ovakve integracije se koriste tehnologije poput *JavaScripta,* HTML-a (engl. ***H**yper**T**ext **M**arkup **L**anguage*) i CSS-a (engl. ***C**ascading **S**tyle **S**heets*) s pomoću razvojne cjeline poput *Flask* ili *Django* koje služe za lakše stvaranje složenih „web“ aplikacija. Ovim pristupom se omogućava korisnicima da na jednostavan način učitavaju slike na „web“ stranici koje su generirane GAN modelom. *Backend* sustav bi koristio API za komunikaciju s GAN modelom, omogućavajući prijenos slika korisniku. Radi ovoga je važno osigurati da sučelje bude optimizirano za brzo učitavanje i obradu podataka. Međutim, postoji olakšavajući faktor u „TensorFlow.js“ konverziji. Ova konverzija omogućuje učitavanje i izvođenje modela izravno u „web“ pregledniku, bez potrebe za *backend* poslužiteljem čime se znatno olakšava ovaj korak.

Za početak, nakon što je GAN model razvijen i uspješno treniran na jednu zadovoljavajuću razinu, potrebno je model sačuvati u formatu „.h5“, koji je standard za *Keras* modele. Konkretno, potrebno je sačuvati samo model generatora budući da on kreira nove slike. Pomoću „.save()“ metode u *Keras* biblioteci [23], ovo se radi jednom linijom kôda:

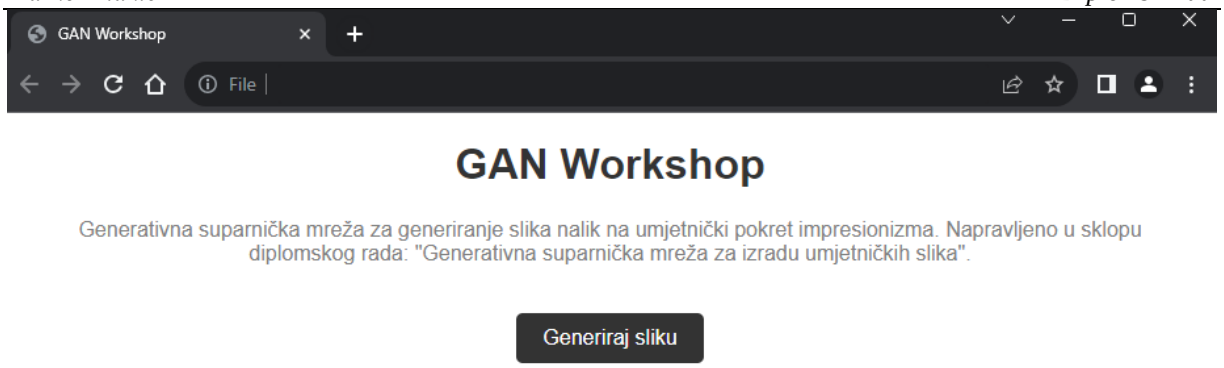
```
generator.save('impressionism_generator.h5')
```

Sljedeći korak jest prebacivanje sačuvanog *Keras* modela u format prikladan za „web“, što je moguće pomoću ranije spomenute „TensorFlow.js“ konverzije. Kako bi zamjena bila moguća, potrebno je definirati putanju do „.h5“ datoteke i putanju gdje bi se pretvoreni model trebao spremiti, npr.:

```
!tensorflowjs_converter --input_format=keras \
'/content/impressionism_generator.h5' '/content/drive/MyDrive/tfjs_model'
```

Model je pretvoren u „.json“ (engl. ***JavaS**cript **O**bject **N**otation*) datoteku i pripadne „.bin“ (engl. *binary*) datoteke u kojima su sadržane težine modela.

Za stvaranje korisničkog sučelja, potrebno je razviti HTML dokument koji bi služio kao osnova „web“ stranice. Zatim s integracijom CSS-a se definira vizualni izgled i raspored elemenata na stranici; uključujući kontejner za slike i gumb za generiranje slika kako bi se omogućila interakcija korisnika i sučelja. Inicijalni izgled stranice (nazvana „GAN Workshop“), bez interakcije s GAN modelom je vidljiv na sljedećem prikazu [Slika 16].



Slika 16. Inicijalni izgled sučelja na *Chrome* pregledniku

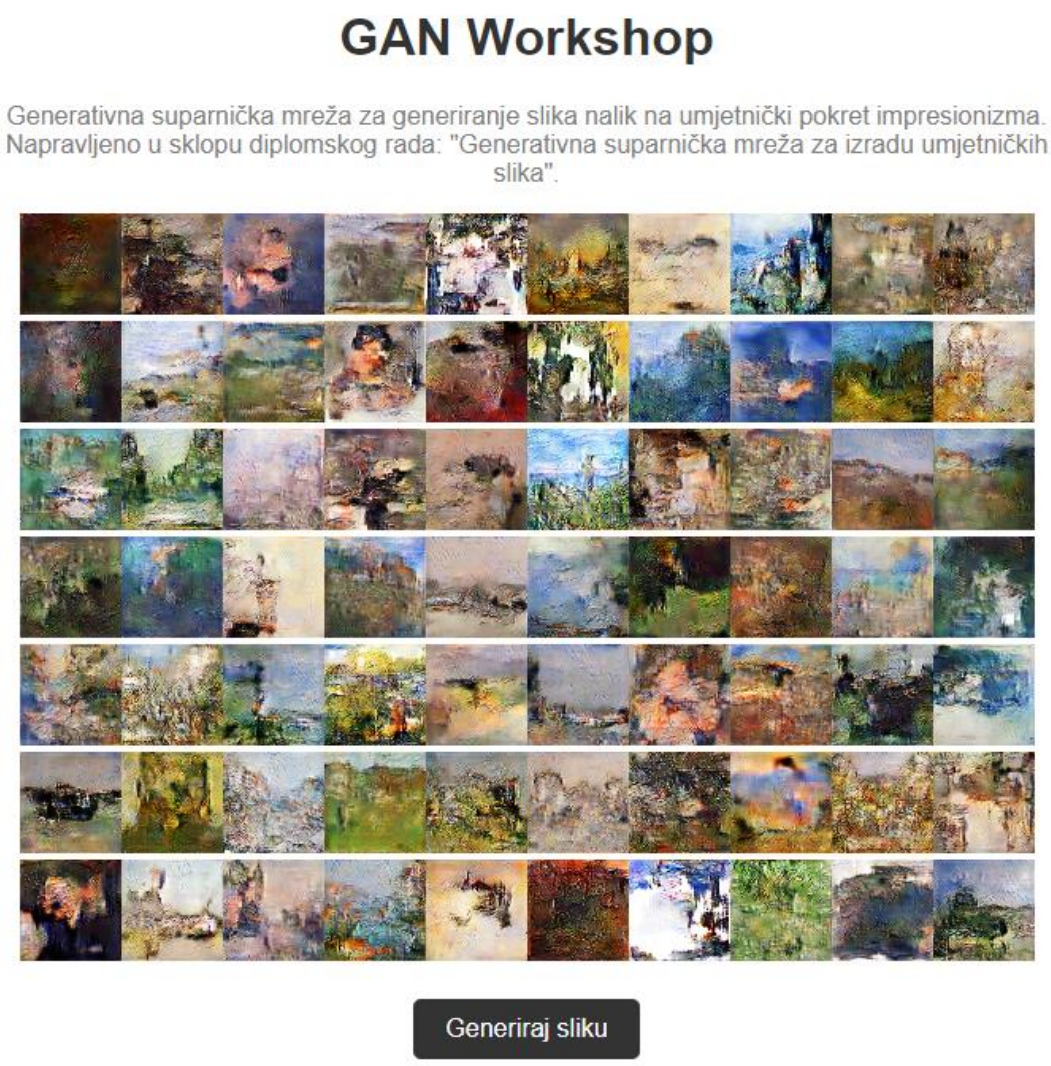
Kako bi se omogućila interakcija GAN modela i sučelja potrebno je koristiti ranije pretvorene datoteke (nazvana „model.json“) i *JavaScript*. Učitava se model u preglednik, a nakon što korisnik klikne na gumb „Generiraj sliku“, *JavaScript* pokreće model koji generira novu sliku. Generiranje slika bi se trebalo odvijati u realnom vremenu te najjednostavniji pristup bi bio napraviti *JavaScript* kôd koji naliči na *Python* kôd ispod:

```
# Učitaj spremljeni model
generator_path = '/content/impressionism_generator.h5'

generator = load_model(generator_path)

# Pomoćna f-ja -- Testiranje generatora

def generate_images(generator, num_images):
    random_latent_vectors = tf.random.normal(shape=(num_images,
generator.input_shape[1]))
    generated_images = generator(random_latent_vectors)

    # Rescale -> [-1, 1] u [0, 1]
    generated_images = (generated_images + 1) / 2.0

    return generated_images

def show_generated_images(images, num_images=64):
    rows = cols = math.ceil(math.sqrt(num_images))
    plt.figure(figsize=(5 * cols, 5 * rows))

    for i in range(num_images):
        plt.subplot(rows, cols, i+1)
        plt.imshow(images[i])
        plt.axis('off')

    plt.show()
```

Ovo je pomoćna funkcija koja je bila korištena za pregled generiranih slika tijekom treniranja GAN modela. Sličnu strukturu bi imao i „web“ implementirani *JavaScript* kôd: slanje

nasumičnog šuma, skaliranje normalizirane slike natrag u format koji se može prikazati u preglednicima i prikazati sliku. Cjelovito rješenje ovog dijela rada se nalazi u prilogu.

Nakon uspješnog povezivanja GAN modela i *frontenda* napravljene „web“ stranice, može se testirati. Na sljedećoj slici je vidljiva „web“ stranica s generiranim nizom slika te je ovim gotova integracija modela na „web“ stranicu - omogućeno je da korisnik bez tehničkog predznanja interaktivno isproba GAN model i stvori vlastitu umjetničku sliku.



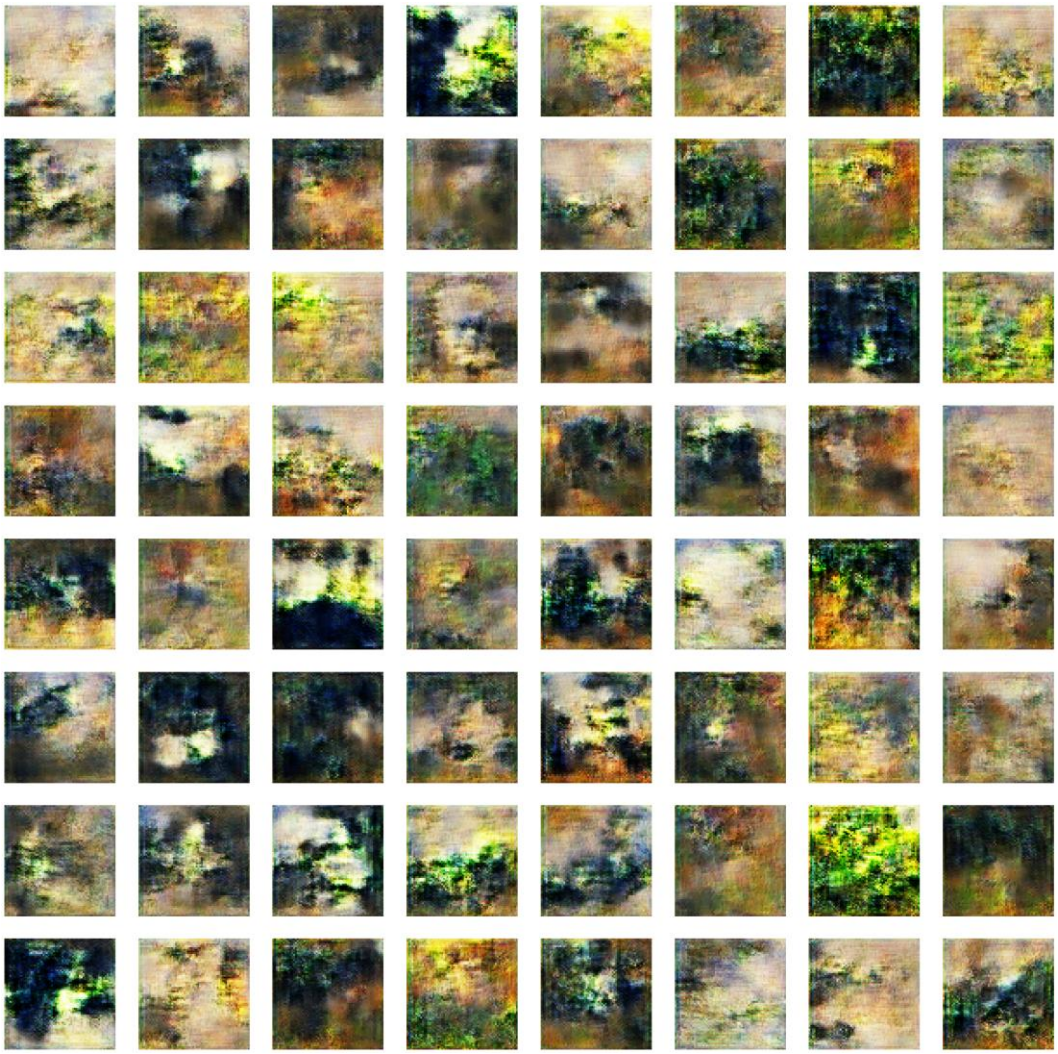
Slika 17. Izgled stranice nakon dodavanja interakcije GAN modela

4. REZULTATI

U ovom poglavlju se prikazuju rezultati nastali tijekom i na kraju učenja generativne suparničke mreže. Vizualni prikazi su selektivno birani radi predočavanja promjena tijekom epoha. Zatim se generirane slike evaluiraju i analiziraju radi određivanja kvalitete generiranih slika.

4.1. Vizualni prikaz generiranih slika

U početnim epohama generirane slike prikazuju osnovne oblike s dominacijom zelenih boja, vidljivo ispod [Slika 18]. Neki oblik šuma je prisutan u svim generiranim slikama te svaka pojedinačna slika sadrži mješavinu boja i tekstura bez jasnih ili prepoznatljivih oblika.



Slika 18. Generirane slike nakon 100 epoha

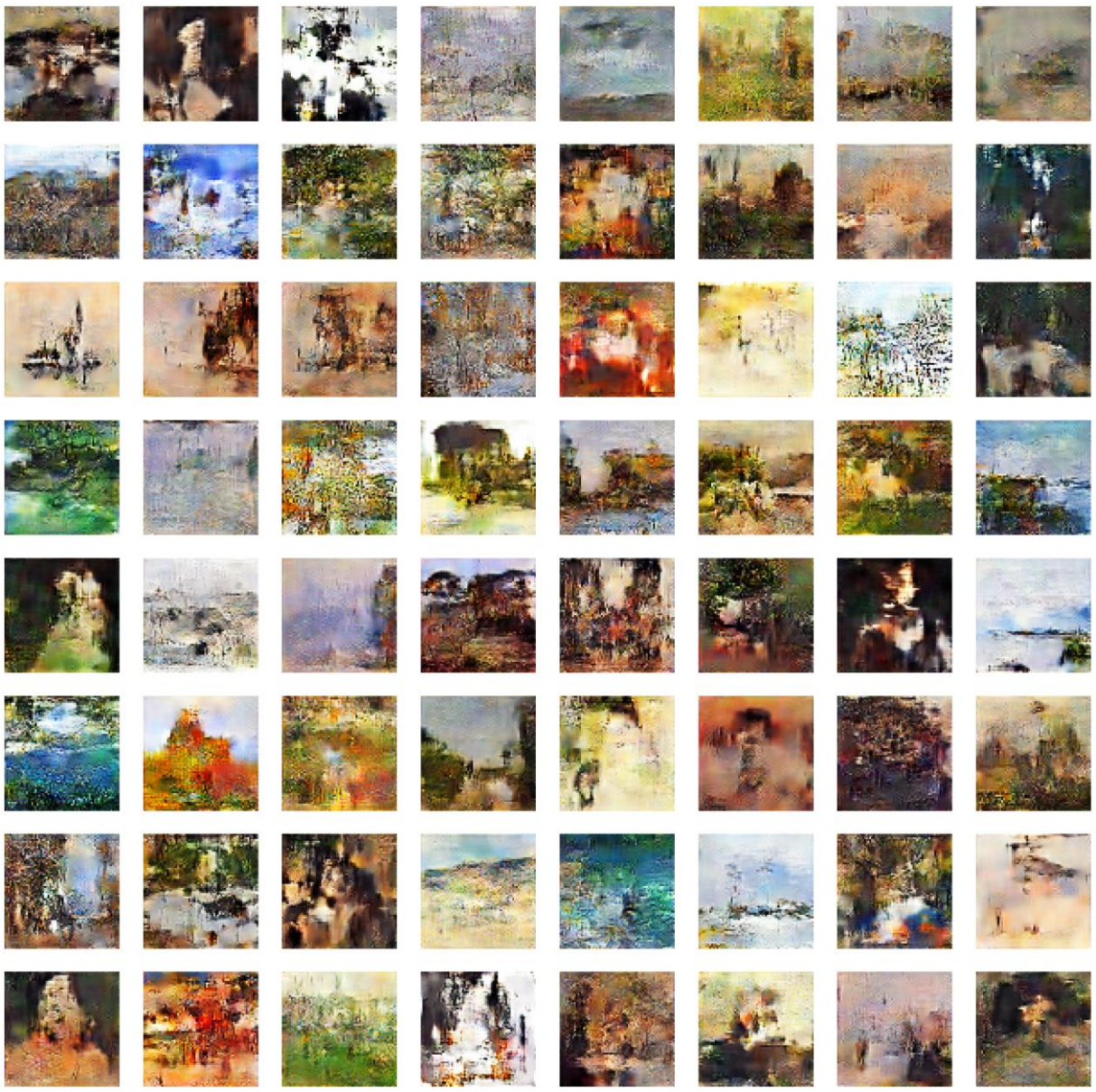
Sljedeća slika [Slika 19] prikazuje značajan napredak u odnosu na prethodni set. Slike imaju jasniju strukturu i različite su u boji i teksturi. Ovo ukazuje da je generator naučio bolje diferencirati između različtiih elemenata u scenama koje pokušava generirati. Šum je i dalje prisutan, ali neke od slika pokazuju i složenije kompozicije s vidljivim linijama koje sugeriraju na horizont ili krajolik. Premda u ovom setu generirane slike i dalje više spadaju u domenu apstraktnog, a ne impresionističkog slikarstva, daljnjim treniranjem modela se može postići veća preciznosti i realizam.

Slika 19. Generirane slike nakon 500 epoha

Naredni set slika [Slika 20] prikazuje nastavljeni napredak u smislu definicije i razlikovanja elemenata unutar svake slike: raznolikost boja je zastupljenija i teksture su složenije i dijelom detaljnije. Svaka slika ima jedinstven karakter s određenim elementima koji počinju dovoljno se razlikovati da bi postali djelomično prepoznatljivi kao uzorci iz prirode. Dodatno se da primijetiti i kako osvjetljenje i sjenčanje je poboljšano što pridonosi osjećaju dubine i realističnosti. Šum je dijelom i dalje prisutan, ali u znatno manjoj količini relativno na početak procesa učenja mreže. Ovaj napredak je ohrabrujući te sugerira da bi model s daljnjim treniranjem se mogao znanto poboljšati.

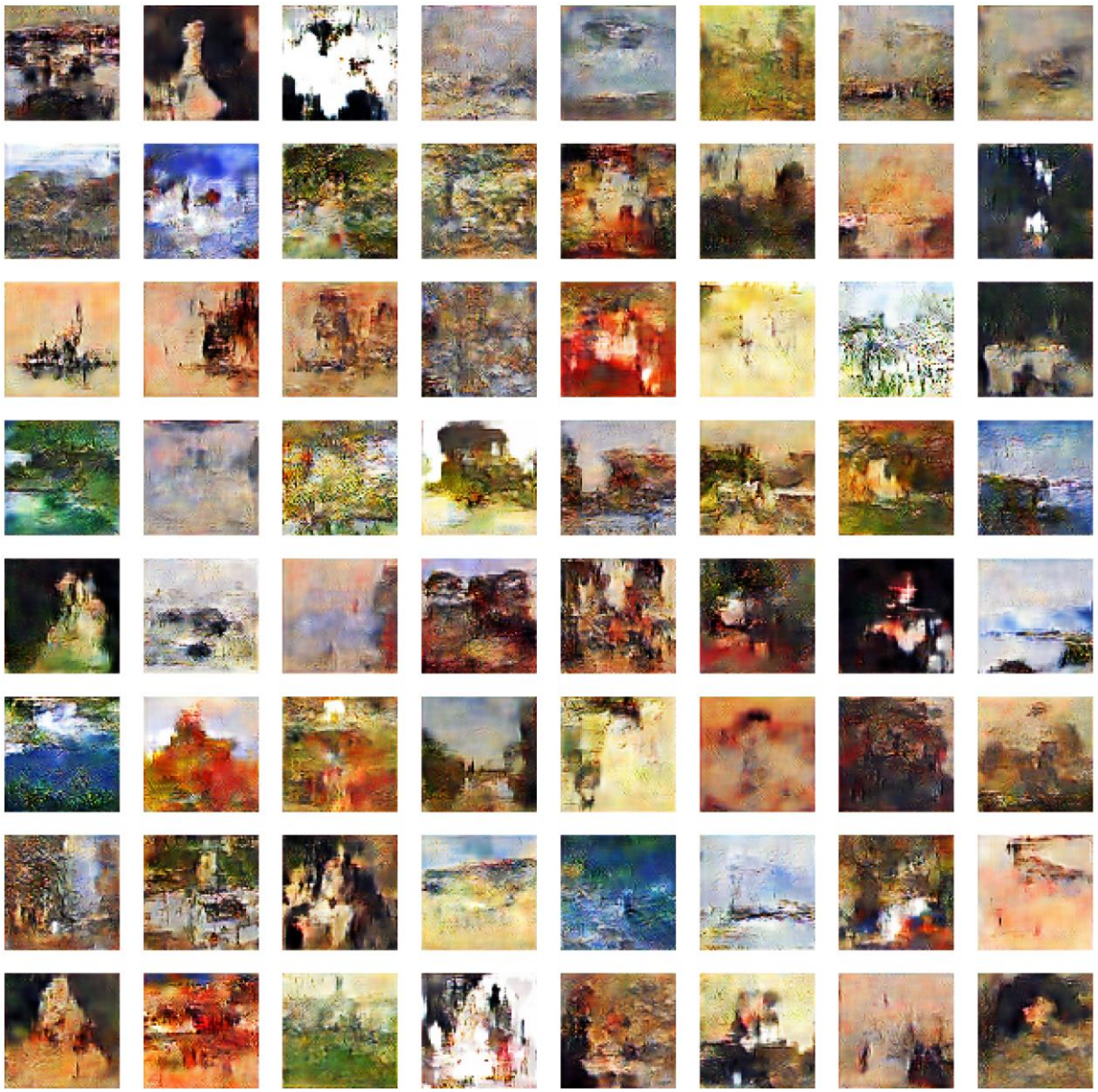
Slika 20. Generirane slike nakon 2000 epoha

Sljedeći set generiranih slika [Slika 21] je nastao nakon 3300 epoha treniranja te je vidljivo kako generativna suparnička mreža nastavlja napredovati - slike imaju još veću razinu detalja i jasnoće. Vidljiva je varijacija u stilu i tehnikama, podsjećajući na impresionizam. Boje su postale bogatije i više odražavaju prirodne tonove, a sjenčanje je sada još izraženije čime se doprinosi kontrastu i prezentaciji slika. Šum je i dalje prisutan kod slika koje imaju očitiji kontrast s tamnijim tonovima, ali GAN napreduje i generira uvjerljivije i estetski privlačnije slike.



Slika 21. Generirane slike nakon 3300 epoha

Radi usporedbe, sljedeći set slika [Slika 22] prikazuje iste generirane slike nakon prolaska većeg broja epoha. GAN nastavlja napredovati, generirajući slike s oštrijim i definiranijim linijama gdje su prisutni svijetliji tonovi, dok kod slika s tamnijom pozadinom je uz slabiju definiciju prisutan i značajniji šum. Ovo navodi na potrebu smanjenja stope učenja generatora i diskriminatora kako bi se mogli „uhvatiti“ detalji gdje je GAN dosad imao poteškoća.



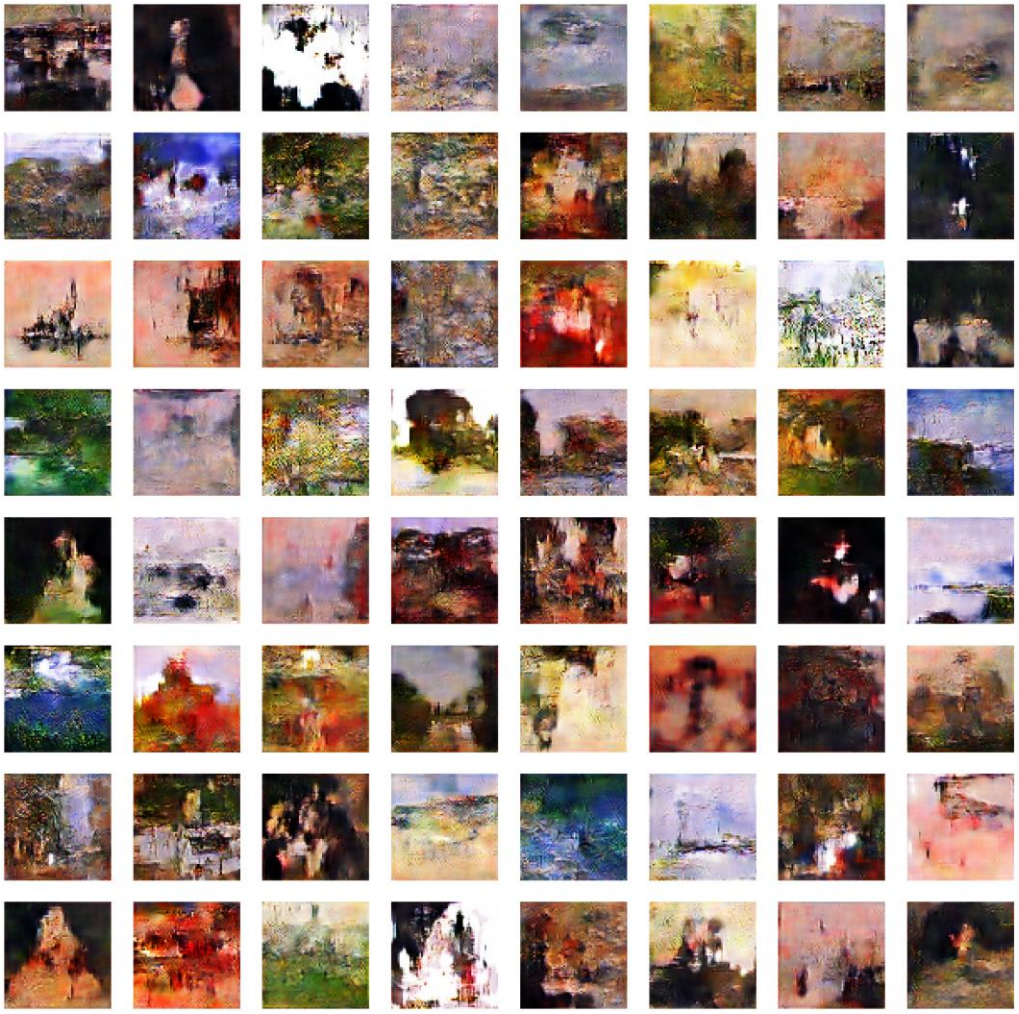
Slika 22. Generirane slike nakon 4000 epoha

Nakon 4350 epoha slike u setu ispod [Slika 23] prikazuju daljnji napredak u stvaranju složenih vizualnih scena. Svaka slica imaju svoje osobitosi i ne nalikuju što je ohranrujuće iz aspekta ranije opisanog kolapsa modela. Elementi prirode kao što su drveća, voda i nebo su sada prepoznatljivi u nekim slikama. Ovime se nastavljaju prikazi koji nalikuju na impresionističke slike.

Slika 23. Generirane slike nakon 4350 epoha

Ovaj prikaz je ujedno i „vrhunac“ tijekom učenja budući da sve kasnije iteracije bi počele pokazivati elemente nestabilnosti te generirane slike ne bi napredovale već stagnirale ili čak

postajale lošije kvalitete. Na narednoj slici [Slika 24] je vidljiv „razvoj“ generiranih slika nakon nastavka treniranja.



Slika 24. Rezultat nastavka treniranja GAN modela

Primjetno je prevladavanje crvenih tonova te da kompozicijski slike nazaduju. Promjenom stope učenja, optimizatora ili funkcije gubitka nije dovelo do poboljšanja rezultata; slike bi nastavile nazadovati dok se ne bi potpuno pretvorile u šum. Ovo upućuje na mogućnost zasićenja kapaciteta modela te u slučaju da je to tako bi se ovo rješavalo promjenom inicijalne strukture generatora i diskriminatora - nešto što se ne može raditi tijekom treniranja. S obzirom na to da generalna vrsta GAN-a koja se koristila za ovaj zadatak ne spada u kompleksnije te da skup podataka je nesavršen (velike dimenzije koje je bilo potrebno normalizirati i skalirati na veličinu 64x64 piksela), odlučeno je da se treniranje zaustavi kako bi se izbjeglo daljnje degradiranje kvalitete generiranih slika i bespotrebo trošenje resursa.

4.2. Evaluacija i analiza kvalitete generiranih slika

Osim vizualnog pregleda, važno je imati i kvantitativnu procjenu koja bi mogla uputiti jesu li izrađene slike od strane GAN-a kvalitetne, raznolike i realistične. Iako u posljednjim godinama je došlo do značajnih napredaka u razvoju generativnih suparničkih mreža, evaluacija GAN modela (posebno onih sa zadatkom generiranja novih podataka) i dalje predstavlja veliki izazov. Prije svega, iako postoji nekoliko metoda mjerenja performansi GAN-a, ne postoji opće prihvaćen standard koji bi objektivno ocijenio snage i ograničenja modela te omogućio pravednu procjenu performansi modela. Ipak, kao i u drugim područjima strojnog učenja, ključno je uspostaviti jednu ili nekoliko metoda mjerenja koje bi usmjeravale napredak u ovom području. U trenutku pisanja postoje dvije metode koje su najzastupljenije: Ocjena po „Inception" modelu (poznatije kao ***I**nception **S**core* na engleskom, IS) i Fréchetova „Inception" udaljenost (engl. ***F**réchet **I**nception **D**istance,* FID) [27-29]. Prva od navedenih, ocjena po „Inception" modelu, temelji se na korištenju prethodno istrenirane mreže (tzv. *Inception Net,* trenirane na „ImageNet" skupu podataka [27-29]) kako bi se ocijenila raznolikosti i mogućnost klasifikacije generiranih podataka. Može se opisati sljedećom jednadžbom:

$$IS(G) = \exp\left(E_G\left[D_{KL}\left(p(y|x)||p(y)\right)\right]\right), \tag{4.2.1}$$

gdje je E_G očekivana vrijednost za skup generiranih slika, D_{KL} je već spomenuta KL-divergencija, $p(y|x)$ je uvjetna distribucija s obzirom na sliku x i $p(y)$ je marginalna distribucija preko svih generiranih slika. Ukratko, iz jednadžbe se procjenjuje ocjena temeljem koliko su GAN generirane slike klasificirane (jasno određene klase, niska entropija $p(y|x)$) i koliko su raznolike (slike pokrivaju različite klase, visoka entropija $p(y)$) [27]. Visoka ocjena po „Inception" modelu sugerira da su generirane slike raznolike i kvalitetne.

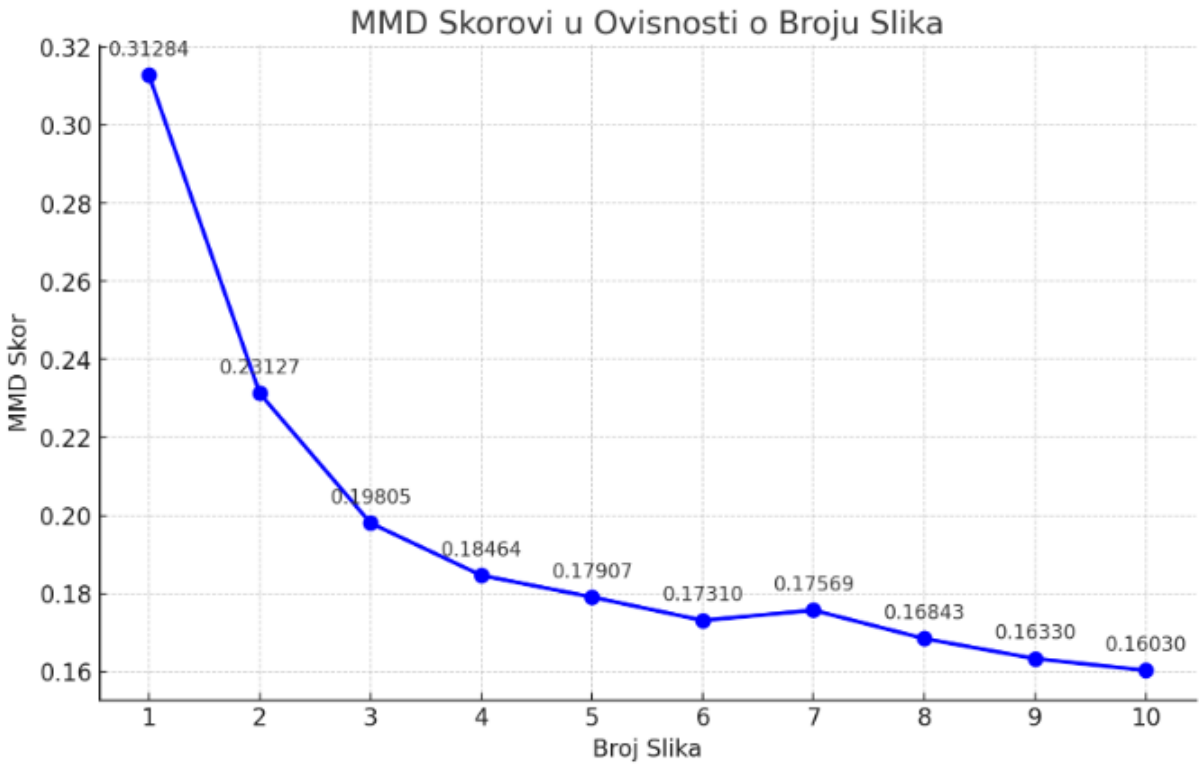
Fréchetova „Inception" udaljenost izračunava, kao što ime sugerira, udaljenost između distribucija stvarnih i generiranih slika u sloju konvolucijske neuronske mreže. Ova metoda se također temelji na istoj „Inception" mreži kao i IS te definirana je sljedećim izrazom:

$$FID(R,G) = \left\|\mu_r - \mu_g\right\|_2^2 + Tr\left(\sum_r + \sum_g - 2\left(\sum_r \sum_g\right)^{1/2}\right), \tag{4.2.2}$$

gdje su μ_r i $\sum_r$ srednja vrijednost i kovarijanca stvarnih podataka, a μ_g i $\sum_g$ srednja vrijednost i kovarijanca generiranih podataka. Metoda radi tako što se za skup podataka stvarnih i generiranih slika procjenjuje srednja vrijednost i kovarijanca uz pretpostavku da su značajke distribuirane kao kontinuirana, višedimenzionalna Gaussova distribucija [27]. Zatim se ova udaljenost između dvije Gaussove distribucije (poznato još i kao Wassersteinova-2 udaljenost) koristi za kvantificiranje kvalitete generiranih uzoraka. Niski FID ukazuje na to da su distribucije stvarnih i generiranih slika slične što upućuje da su generirane slike bolje kvalitete dok visoki FID upućuje da su slike lošije kvalitete [27, 28].

Međutim, iako su ocjena po „Inception“ modelu i Fréchetova „Inception“ udaljenost kvalitetne metode za kvantitativnu evaluaciju generiranih slika s pomoću GAN-a, one imaju svoja ograničenja - posebno kada se uzme u obzir procjena kvalitete slika koje su generirane na skupu podataka umjetničkih slika (u ovom slučaju impresionizma). Prije svega, kao što je već spomenuto, obje metode se oslanjaju na prethodno trenirani „Inception“ model koji je treniran na „ImageNet“ skupu podataka - skup koji primarno sadrži realistične slike te stoga ove metode mogu biti pristrane kada se koriste za evaluaciju umjetničkih slika. Ovo je posebno izraženo u kontekstu evaluacije slike u stilu impresionizma, gdje stil i estetike igraju ključnu ulogu, a nisu nužno dobro zastupljeni ili prepoznati kroz „ImageNet“ skup podataka. Drugim riječima, rezultati koji bi se dobili evaluiranjem generiranih umjetničkih slika pomoću ovih metoda bi mogli biti irelevantni budući da se IS i FID prvenstveno koncentriraju na raznolikost i realističnost slika u smislu reprodukcije stvarnih objekata i scena. Dakle, za evaluaciju GAN-a koji generira umjetničke slike, potrebne su metode koje mogu adekvatno ocijeniti umjetničku originalnost i estetski izražaj, a ne samo tehničku preciznosti i raznolikost. Ove metode bi trebale uzeti u obzir specifičnosti umjetničkog izraza, kao što su: stil, kompozicija, boja i tekstura. Također, važno je da evaluacijski algoritam je takav da je treniran na skupu podataka koji je relevantan za umjetnički stil koji se istražuje. Time se osigurava veća preciznost i relevantnost rezultata. S obzirom na ovaj izazov, u ovom radu je istražena alternativa s izračunavanjem maksimalne prosječne razlike (engl. ***M**aximum **M**ean **D**iscrepancy*, MMD). Ova metoda izračunava razliku između distribucije generiranih i stvarnih slika time što uzorci se uspoređuju korištenjem Gaussove funkcije kako bi se izračunao stupanj sličnosti između dvije distribucije [27]. Niži MMD skor bi ukazao na veću sličnost između generiranih i stvarnih slika - što bi u kontekstu rada impliciralo da generirane slike bolje oponašaju umjetnički stil originalnog skupa podataka (u ovom slučaju impresionizam).

Sljedeća slika [Slika 25] prikazuje MMD skor u ovisnosti o broju slika koje su generirane na modelu treniranom nakon 4350 epoha. Računanje se radilo tako što bi se za svaku sliku iteriralo određeni broj puta (u ovom slučaju stotinu) te bi se uzela prosječna vrijednost kao konačni MMD skor (prikladni kôd se nalazi u prilogu kao dodatak).



Slika 25. Maksimalna prosječna razlika u ovisnosti o broju generiranih slika nakon 4350 epoha

Rezultati pokazuju kako MMD skor ima tendenciju smanjivanja povećanjem broja slika što upućuje na to da generirane slike se u velikoj mjeri podudaraju s karakteristikama impresionističkih umjetničkih djela što je i vidljivo u ranijem podnaslovu sa slikom 24.

Ovime se zaključuje kako generirani model, premda netreniran do krajnje konvergencije, uspješno je proizvodio primjere koji su nalikovali na originalni skup podataka impresionističkih slika što je ujedno bio i cilj.

5. ZAKLJUČAK

U ovom radu je predstavljena generativna suparnička mreža; relativno nova metodologija u dubokom i strojnom učenju, koja je postala posebno popularna posljednjih godina u generiranju novih podataka na temelju stvarnih. U uvodnom dijelu je opisana povijest generativne umjetne inteligencije: od jednostavnog Boltzmannovog stroja do varijacijskih autoenkodera. U nastavku je izdvojen poseban podnaslov za detaljniji pogled u teorijsku osnovu, primjenu i podvrste GAN-a koje su i danas u upotrebi. Primjetno je kako GAN-ovi, zahvaljujući svojoj robusnosti i mogućnosti prilagodbe inicijalne strukture, se ne trebaju samo koristiti u sferi generiranja multimedijalnog sadržaja već se može primjenjivati i u druge svrhe. Kroz metodologiju je opisano kako za izradu ovog zadatka je korišten *Python* u *Google Colab* okruženju radi pogodnosti korištenja GPU-a i TPU-a. Odabrani skup podataka je „wiki-art“ s fokusom na umjetnički stil impresionizma s preko 13 000 slika. Struktura i dizajn primijenjenog GAN modela temeljila se na dubokoj analizi i eksperimentaciji, prilagođavajući hiperparametre tijekom treniranja modela poput stope učenja, optimizacijskih algoritama i funkcije gubitka. Tijekom implementacije predstavilo se niz izazova, uključujući problem nestabilnosti, prenaučenosti i kolaps modela. Prilagodba veličina serije, inicijalne strukture generatora i diskriminatora kao i prilagodba hiperparametara je dovela do rješavanja ovih problema. Dodane *callback* funkcije su omogućile lakše nadgledanje modela tijekom procesa treniranja modela za generiranje novih slika sličnih skupu podataka. Iako je model pokazao veliki potencijal u generiranju visokorealističnih umjetničkih djela nakon određenog broja epoha slike se nisu nastavile razvijati već je model stagnirao. Ovo je objašnjeno time da je vjerojatno došlo do zasićenja kapaciteta te bi daljnje napredovanje generatora zahtijevalo dodatno eksperimentiranje s hiperparametrima kao i prekomjerne računalne resurse radi čega je proces treniranja zaustavljen. Također je prikazan način integracije GAN modela u izrađeno „web“ sučelje s korištenjem *JavaScript* programskog jezika uz osnove HTML-a i CSS-a. Na kraju su rezultati vizualno prezentirani i analizirani s fokusom na raznolikost generiranih slika kako bi se potvrdilo da ne dolazi do kolapsa modela. Korištenjem maksimalne prosječne razlike su generirane slike kvantitativnu analizirane te je potvrđeno kako se skor smanjuje s povećanjem broja slika, što ukazuje na to da se razlike između generiranih i stvarnih slika smanjuju kako se broj slika poveća. Zaključno, rad predstavlja doprinos u razumijevanju i primjeni GAN tehnologije u generiranju umjetničkih slika. Kako tehnologija napreduje, mogu se očekivati daljnje inovacije i primjene GAN-a u različitim poljima, čime se otvaraju nova vrata u svijetu generativne umjetne inteligencije.

LITERATURA

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: *Generative Adversarial Nets*, Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, 2014.

[2] *Understanding Boltzmann Machines Applications and Markov Chain*, https://www.mygreatlearning.com/blog/understanding-boltzmann-machines/, 17.10.2023.

[3] Dumoulin, V., Goodfellow, I. J., Courville, A., Bengio, Y.: *On the Challenges of Physical Implementations of RBMs*, Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, 2014.

[4] Goodfellow, I. J., Bengio, Y., Courville, A.: *Deep Learning*, MIT Press, 2016.

[5] Kingma, D. P., Welling, M.: *An Introduction to Variational Autoencoders*, Foundations and Trends® in Machine Learning: Vol. 12, No. 4, pp 307-392, 2019., doi: https://doi.org/10.1561/2200000056

[6] *Generative Adversarial Networks - Hot Topic in Machine Learning*, https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html, pristupljeno: 21.10.2023.

[7] You, A., Kim J. K., Ryu, I. H., Yoo T. K.: *Application of generative adversarial networks (GAN) for ophthalmology image domains: a survey*, Eye and Vision, 2022., doi: https://doi.org/10.1186/s40662-022-00277-3

[8] Tripathi, S. *et al*: *Recent advances and application of generative adversarial networks in drug discovery, development, and targeting*, Artificial Intelligence in the Life Sciences, Volume 2, 2022., doi: https://doi.org/10.1016/j.ailsci.2022.100045

[9] Uricar, M. *et al*: *Yes, we GAN: Applying Adversarial Techniques for Autonomous Driving*, 2020., doi: https://doi.org/10.48550/arXiv.1902.03442

[10] Engel, J. *et al*: *GANSynth: Adversarial Neural Audio Synthesis*, ICLR konferencija, 2019., doi: https://doi.org/10.48550/arXiv.1902.08710

[11] Eckerli, F., Osterrieder, J.: *Generative Adversairal Networks in finance: an overview*, 2021., doi: https://doi.org/10.48550/arXiv.2106.06364

[12] Radford, A., Metz, L., Chintala, S.: *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, ICLR konferencija, 2016., doi: https://doi.org/10.48550/arXiv.1511.0643

[13] Arjovsky, M., Chintala, S., Bottou, L.: *Wasserstein GAN*, Courant Institute of Mathematical Sciences, Facebook AI Research, 2017., doi: https://doi.org/10.48550/arXiv.1701.07875

[14] Karras, T., Aila, T., Laine, S., Lehtien, J.: *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, ICLR konferencija, 2018., doi: https://doi.org/10.48550/arXiv.1710.10196

[15] Karras, T., Laine, S., Aila, T.: *A Style-Based Generator Architecture for Generative Adversarial Networks*, CVPR konferencija, 2019., doi: https://doi.org/10.48550/arXiv.1812.04948

[16] https://thispersondoesnotexist.com/, pristupljeno: 3.11.2023.

[17] Brock, A., Donahue, J., Simonyan, K.: *Large Scale GAN Training for High Fidelity Natural Image Synthesis*, ICLR konferencija, 2019., doi: https://doi.org/10.48550/arXiv.1809.11096

[18] Zhu, J-Y., Park, T., Isola, P., Efros, A. A.: *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, Berkeley AI Research (BAIR) laboratory, UC Berkeley, 2020., doi: https://doi.org/10.48550/arXiv.1703.10593

[19] *Distributed training with TensorFlow*, https://www.tensorflow.org/guide/distributed_training, pristupljeno: 7.11.2023.

[20] *Machine Learning and Training Data: What You Need to Know*, Label Your Data, https://labelyourdata.com/articles/machine-learning-and-training-data#splitting_your_data_set_training_data_vs_testing_data_in_machine_learning, pristupljeno: 8.11.2023.

[21] „wiki-art“ skup podataka, Deep Lake, https://app.activeloop.ai/activeloop/wiki-art/firstdbf9474d461a19e9333c2fd19b46115348f, pristupljeno 10.11.2023.

[22] *Early Stopping*, Deeplearning4j, https://deeplearning4j.konduit.ai/v/en-1.0.0-beta7/getting-started/tutorials/early-stopping, pristupljeno 11.11.2023.

[23] *Keras 3 API documentation*, Keras, https://keras.io/api/, pristupljeno: 11.11.2023.

[24] Odena, A. *et al*.: *Deconvolution and Checkerboard Artifacts*, Distill, https://distill.pub/2016/deconv-checkerboard/, pristupljeno: 13.11.2023.

[25] Brownlee, J.: *How to Implement GAN Hacks in Keras to Train Stable Models*, Machine Learning Mastery, https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/, pristupljeno: 14.11.2023.

[26] Ioffe, S., Szegedy, C.: *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, doi: https://doi.org/10.48550/arXiv.1502.03167

[27] Borji, A.: *Pros and Cons of GAN Evaluation Measures*, doi: https://doi.org/10.48550/arXiv.1802.03446

[28] Shmelkov, K. *et al*: *How good is my GAN?*, ECCV konferencija, 2018., doi: https://doi.org/10.48550/arXiv.1807.09499

[29] Guan, S., Loew, M.: *Evaluation of Generative Adversarial Network Performance Based on Direct Analysis of Generated Images*, IEEE Applied Imagery Pattern Recognition Workshop (AIPR), 2019., Washington, doi: 10.1109/AIPR47015.2019.9174595

PRILOZI

I. Blok shema generatora

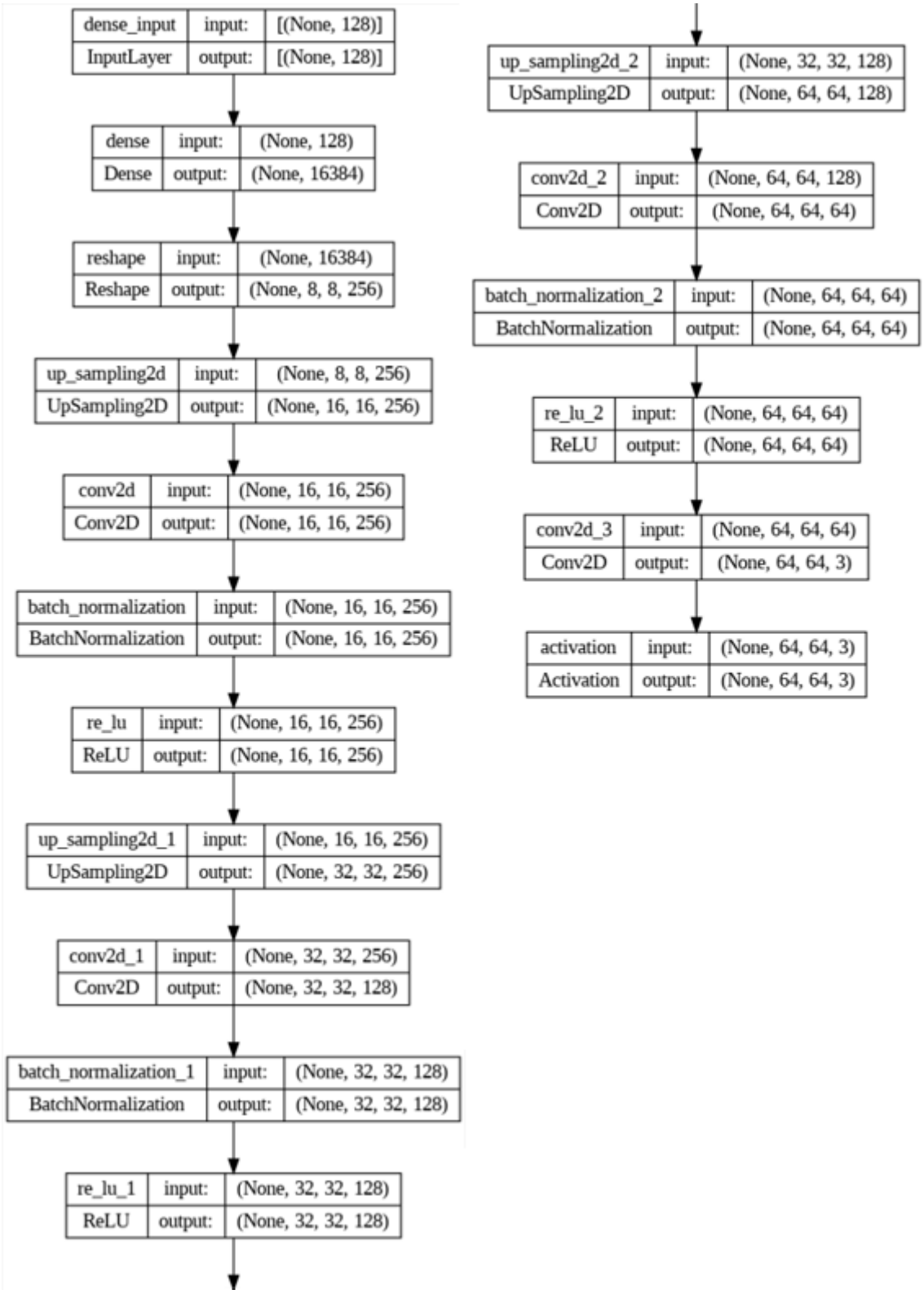
II. Blok shema diskriminatora

III. Dio Python kôda

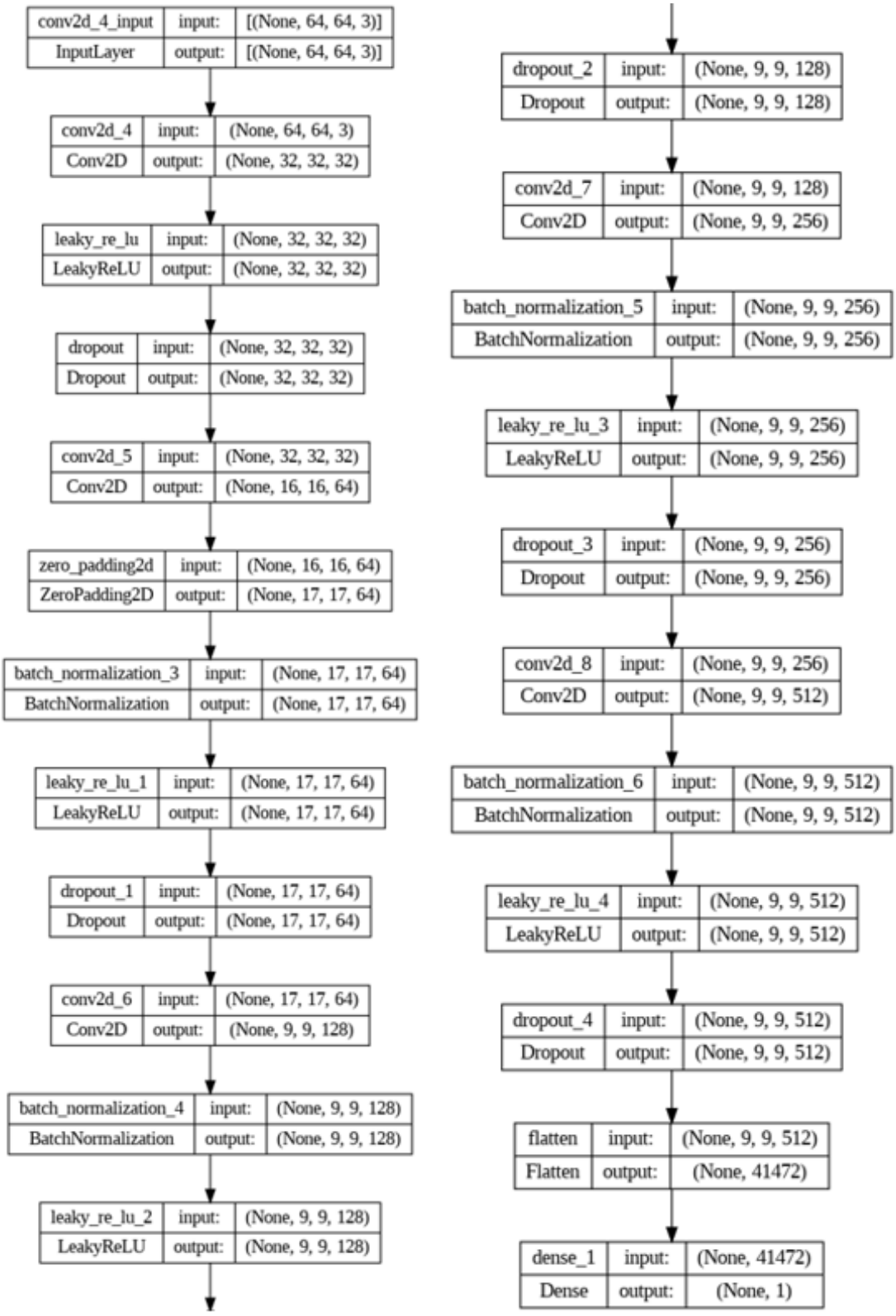
IV. Kôd za implementaciju GAN modela na „web“ stranicu

V. Dio kôda za računanje prosječnog MMD skora

I. Blok shema generatora



II. Blok shema diskriminatora



III. Dio Python kôda

```
!pip install hub
!pip install pytz

# 0 - Imports
import tensorflow as tf
import os
import hub
import time
import math
import random
import numpy as np
import matplotlib.pyplot as plt
import datetime
import pytz
from PIL import Image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, ReLU,
UpSampling2D, ZeroPadding2D, Dropout, BatchNormalization, LeakyReLU,
Reshape, Conv2DTranspose, Conv2D, Flatten
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import array_to_img
from tensorflow.keras.callbacks import Callback, ModelCheckpoint,
EarlyStopping



# 1.1 - Load dataset

from google.colab import userdata
API_token = userdata.get('ds_API_key')
## Dataset -- API token je globalna varijabla
dataset_path = 'hub://activeloop/wiki-art'
ds = hub.load(dataset_path, token=API_token) # WikiArt skup podataka
```

```
# 1.2 - Inicijalizacija

seed_size = 128
batch_size = 32
artstyle_for_training = 'impressionism'
# Mijenjanjem naziva varijable omogućuje se treniranje na drugom skupu podataka,
     odnosno drugi stil (realizam, renesansa, ekspresionizam itd.) čineći ovaj kôd
     modularnim za uzeti skup podataka

# 2 - Get data za treniranje

labels_list = ds.labels.info['class_names']
# Pronaći indeks za impresionizam
artstyle_index = labels_list.index(artstyle_for_training)

label_counts = {label: 0 for label in labels_list}

for label in ds.labels.numpy():
     label_index = label[0]
     class_name = labels_list[label_index]
     label_counts[class_name] += 1
print(label_counts)  # Broj slika po stilu

# Sortiranje label_counts, descending
sorted_label_counts = dict(sorted(label_counts.items(), key=lambda item: item[1],
     reverse=True))
print(sorted_label_counts)

print(f"Indeks umjetničkog stila ({artstyle_for_training}): {artstyle_index}")
print(f"Broj slika: {label_counts[artstyle_for_training]}")

# Filtriranje dataseta koristeći Hub API
artstyle_ds = ds.filter(lambda x: x.labels.numpy() == artstyle_index)

# Pretvori u TFDS
artstyle_ds_tf = artstyle_ds.tensorflow()
```

```
# 2.1 - Predobrada

def preprocess_image(image):
     # Promjena veličine slike na 64x64
     image = tf.image.resize(image, [64, 64])
     # Normalizacija pixel vrijednosti na raspon [-1, 1]
      image = (tf.cast(image, tf.float32) / 127.5) - 1
     return image

# Primjena predobrade na svaku sliku u datasetu
artstyle_ds_tf = artstyle_ds_tf.map(lambda x: {'images':
     preprocess_image(x['images']), 'labels': x['labels']})

# Ostali potrebni koraci: cache, shuffle, batch i prefetch
artstyle_ds_tf = artstyle_ds_tf.cache()
artstyle_ds_tf = artstyle_ds_tf.shuffle(buffer_size=13060)
artstyle_ds_tf = artstyle_ds_tf.batch(32)
artstyle_ds_tf = artstyle_ds_tf.prefetch(tf.data.AUTOTUNE)
```

```
# 3 - Gradnja mreža
## Generator

def build_generator(seed_size, channels):
    model = Sequential()

    # Prvi sloj - Dense i Reshape
    model.add(Dense(8*8*256, activation="relu",input_dim=seed_size))
    model.add(Reshape((8, 8, 256)))

    # Blok 1 - Upsample i Conv2D
    model.add(UpSampling2D())
    model.add(Conv2D(256, kernel_size=3, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(ReLU())
    # Blok 2 - Upsample i Conv2D
    model.add(UpSampling2D())
    model.add(Conv2D(128, kernel_size=3, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(ReLU())

    # Blok 3 - Upsample i Conv2D
    model.add(UpSampling2D())
    model.add(Conv2D(64, kernel_size=3, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(ReLU())

    # Blok 4 - Conv2D (Finalni sloj)
    model.add(Conv2D(channels, kernel_size=3, padding="same"))
    model.add(Activation("tanh"))

    return model

generator = build_generator(seed_size, 3)
generator.summary()
```

```
## Diskriminator

def build_discriminator():
    model = Sequential()

     model.add(Conv2D(32,kernel_size=3,strides=2,           input_shape=(64,64,3),
padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
    model.add(ZeroPadding2D(padding=((0,1),(0,1))))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(256, kernel_size=3, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(512, kernel_size=3, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    return model

discriminator = build_discriminator()
discriminator.summary()
```

```
# 4 - Training loop

g_opt = Adam(learning_rate=0.0002, beta_1=0.5)
d_opt = Adam(learning_rate=0.00001, beta_1=0.5)
g_loss = BinaryCrossentropy()
d_loss = BinaryCrossentropy()

class PaintingGAN(Model):
  def __init__(self, generator, discriminator, *args, **kwargs):
    super().__init__(*args, **kwargs)
    # Atributi za G i D
    self.generator = generator
    self.discriminator = discriminator

  def compile(self, g_opt, d_opt, g_loss, d_loss, *args, **kwargs):
    super().compile(*args, **kwargs)
    self.g_opt = g_opt
    self.d_opt = d_opt
    self.g_loss = g_loss
    self.d_loss = d_loss

  def train_step(self, batch):
    # Dimenzije za generiranje slučajnog šuma
    batch_size = tf.shape(batch['images'])[0]
    random_latent_vectors = tf.random.normal(shape=(batch_size, 128))

    # Treniranje diskriminatora
    with tf.GradientTape() as d_tape:
      # Generiranje lažnih slika
      fake_images = self.generator(random_latent_vectors, training=True)

      # Prave i lažne slike
      real_images = batch['images']
      combined_images = tf.concat([real_images, fake_images], axis=0)

      # Etikete i šum
      labels = tf.concat([tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))],
axis=0)
      labels += 0.05 * tf.random.uniform(tf.shape(labels))
```

```
      # Predikcije i gubitak
      predictions = self.discriminator(combined_images, training=True)
      d_loss = self.d_loss(labels, predictions)

    # Gradijenti i optimizacija za diskriminator
    grads = d_tape.gradient(d_loss, self.discriminator.trainable_variables)
    self.d_opt.apply_gradients(zip(grads, self.discriminator.trainable_variables))

    # Treniranje generatora
    with tf.GradientTape() as g_tape:
      random_latent_vectors = tf.random.normal(shape=(batch_size, 128))
      misleading_labels = tf.ones((batch_size, 1))

      generated_images = self.generator(random_latent_vectors, training=True)
      predictions = self.discriminator(generated_images, training=False)
      g_loss = self.g_loss(misleading_labels, predictions)

    # Gradijenti i optimizacija za generator
    g_grads = g_tape.gradient(g_loss, self.generator.trainable_variables)
    self.g_opt.apply_gradients(zip(g_grads, self.generator.trainable_variables))

    return {"d_loss": d_loss, "g_loss": g_loss}


# Stvaranje instance GAN-a
painting_gan = PaintingGAN(generator, discriminator)

# Kompajliranje modela
painting_gan.compile(g_opt, d_opt, g_loss, d_loss)
```

```
# 5 - Callbackovi

## Napravi Callback

class ModelMonitor(Callback):
    def __init__(self, num_img=3, latent_dim=128):
        self.num_img = num_img
        self.latent_dim = latent_dim

    def on_epoch_end(self, epoch, logs=None):
      random_latent_vectors = tf.random.normal((self.num_img, self.latent_dim))
      generated_images = self.model.generator(random_latent_vectors)
      generated_images = (generated_images + 1) / 2.0  # Za tanh
      generated_images = tf.clip_by_value(generated_images, 0, 1)
      generated_images = (generated_images * 255).numpy().astype(np.uint8)

      print(f"Epoch {epoch}:")
      for i in range(self.num_img):
        img = array_to_img(generated_images[i])
        save_path     =     os.path.join('%path%',     f'gen_img_{epoch}_{i}.png')
#hardkodirana vrijednost %path% prilikom svakog testiranja
        img.save(save_path)
        print(f"Image saved: {save_path}")

      if epoch % 100 == 0:
        current_time = datetime.datetime.now(pytz.timezone('Europe/Zagreb'))
        print(f"Current  Time  at  Epoch  {epoch}:  {current_time.strftime('%Y-%m-%d
%H:%M:%S')}")

print("-------")

# Dodani „GANCheckpoint“ i „EarlyStopping“
```

```
# 6 - Treniraj

# Pokretanje treninga
hist = painting_gan.fit(
    artstyle_ds_tf,
    epochs=500, # Preporuka 20000
    callbacks=[ModelMonitor(num_img=4,    latent_dim=128)]    #,    checkpoint_cb,
early_stopping_cb] # Dodati po mogućnosti
)

# 7 - Spasi i plot

generator.save('impressionism_generator.h5')
discriminator.save('impressionism_discriminator.h5')

plt.suptitle('Gubitci diskriminatora i generatora')
plt.plot(hist.history['d_loss'], label='d_loss')
plt.plot(hist.history['g_loss'], label='g_loss')

last_d_loss = hist.history['d_loss'][-1]
last_g_loss = hist.history['g_loss'][-1]

len_history = len(hist.history['d_loss'])

plt.text(len_history, last_d_loss, f"{last_d_loss:.2f}", ha='right')
plt.text(len_history, last_g_loss, f"{last_g_loss:.2f}", ha='right')

plt.legend()
plt.show()
```

IV. Kôd za implementaciju GAN modela na „web“ stranicu

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>GAN Workshop</title>
    <link rel="stylesheet" href="style.css">
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
</head>
<body>
    <div id="content">
        <h1>GAN Workshop</h1>
        <p id="description">
            Generativna suparnička mreža za generiranje slika nalik na umjetnički
pokret impresionizma.
            Napravljeno u sklopu diplomskog rada: "Generativna suparnička mreža za
izradu umjetničkih
slika".
        </p>
        <div id="image-container">
            <!-- Generirana slika-->
        </div>
        <button id="generate-button">Generiraj sliku</button>
    </div>

    <script>
        let model;
        async function loadModel() {
            model = await tf.loadLayersModel('model/model.json');
            console.log('Model loaded');
        }
        window.onload = loadModel;
        async function generateAndDisplayImage() {
            if (!model) {
                console.error('Model not loaded');
                return;
            }
```

```
            const latentDim = 128;
            const zs = tf.randomNormal([1, latentDim]);
            const generatedImage = model.predict(zs); // Generiraj sliku

            // Rescale: [-1, 1] -> [0, 1]
            const scaledImage = generatedImage.add(1).div(2);

            const canvas = document.createElement('canvas');
            const [height, width] = generatedImage.shape.slice(1, 3);
            canvas.width = width;
            canvas.height = height;
            document.getElementById('image-container').appendChild(canvas);

            await tf.browser.toPixels(scaledImage.squeeze(), canvas);
        }

        document.getElementById('generate-button').addEventListener('click',
generateAndDisplayImage);
    </script>
</body>
</html>
```

V. Dio kôda za računanje prosječnog MMD skora

```
# Pomoćne funkcije
def gaussian_kernel(x, y, sigma=1.0):
    beta = 1. / (2. * sigma ** 2)
    norm = tf.reduce_sum((x[:, None] - y) ** 2, axis=-1)
    return tf.exp(-beta * norm)

def compute_mmd(x, y, sigma=1.0):
    x_kernel = gaussian_kernel(x, x, sigma)
    y_kernel = gaussian_kernel(y, y, sigma)
    xy_kernel = gaussian_kernel(x, y, sigma)

    return tf.reduce_mean(x_kernel) + tf.reduce_mean(y_kernel) - 2 *
tf.reduce_mean(xy_kernel)

# Broj iteracija
num_iterations = 100
mmd_scores = []

for _ in range(num_iterations):
    # Generiranje jedne slike
    generated_images = generate_images(generator, 10)

    # Dobivanje jedne stvarne slike
    for batch in artstyle_ds_tf.take(10):
        real_image = (batch['images'][0] + 1) / 2  # Normalizacija i konverzija u
float32

    real_image = tf.expand_dims(real_image, 0)  # Dodavanje batch dimenzije

    # Izračun MMD
    mmd_score = compute_mmd(generated_images, real_image)
    mmd_scores.append(mmd_score)

# Izračun prosječnog MMD rezultata
average_mmd_score = tf.reduce_mean(mmd_scores)
print(f"Average MMD Score: {average_mmd_score}")
```