

---

# GENERISANJE SLIKA KORIŠĆENJEM GAN

---

Jovan Ležaja

brindeksa

Matematički fakultet, Beograd  
navoj96@gmail.com

Aleksandar Vraćarević

434/2016

Matematički fakultet, Beograd  
vracarevicaleksandar@gmail.com

June 15, 2019

## Sadržaj

<b>1 Uvod</b>	<b>2</b>
<b>2 Generativni modeli</b>	<b>2</b>
2.1 Podela generativnih modela . . . . .	2
2.2 GAN . . . . .	3
<b>3 Implementacija</b>	<b>3</b>
3.1 Vanila GAN . . . . .	4
3.2 Problemi prilikom treninga . . . . .	6
3.2.1 Kolaps modusa . . . . .	7
3.2.2 Jednostrano glaćanje oznaka . . . . .	7
3.2.3 Eksperiment . . . . .	8
3.2.4 Odmotani GAN . . . . .	9
3.2.5 Eksperiment . . . . .	10
<b>4 Drugačije arhitekture koje koriste GAN</b>	<b>11</b>
4.1 DCGAN . . . . .	11
4.1.1 Konvolucione neuronske mreže . . . . .	11
4.1.2 Arhitektura DCGAN-a . . . . .	11
4.1.3 Eksperiment . . . . .	11

## 1 Uvod

Razviće računarske inteligencije i njena sve veća popularnost, koja uslovljava "rađanje" novih ideja, omogućava rešavanje sve većeg spektra problema sa kojima se suočava računarstvo. Ono na šta je posebno uticalo razviće računarske inteligencije, konkretnije razviće mašinskog učenja, jeste obrada slika i "računarski vid" (eng. *Computer vision*). Posebno zanimljiv problem sa kojim se suočava mašinsko učenje jeste mogućnost generisanja novih slika nalik na određenu familiju slika. Jedno od mogućih rešenja predstavalju generativni modeli.

## 2 Generativni modeli

Problem koji generativni modeli pokušavaju da reše odnosi se na učenje nekog skupa osobina na osnovu ulaznih podataka i korišćenje tih osobina za modelovanje novih podataka nalik na onima koji su zatećeni u originalnom skupu. Ovaj problem može se predstaviti uz pomoć verovatnoće. Ukoliko trening podaci pripadaju nekoj raspodeli  $p_{stvarno}(x)$  a generisani podaci pripadaju raspodeli  $p_{generisano}(x)$ , cilj je da veštačka raspodela  $p_{generisano}(x)$  bude što sličnija raspodeli  $p_{stvarno}(x)$ . Dakle, generativni model se "trudi" da na osnovu podataka za trening generiše nove uzorke koji pripadaju istoj raspodeli. Primetimo da u nekim slučajevima nije neophodno eksplicitno pronaći raspodelu modela. Naime, nekada je dovoljno da model ima sposobnost da izvlači uzorke koji pripadaju raspodeli  $p_{generisano}(x)$  bez toga da se išta eksplicitno definiše. Naše rešenje je zasnovano na ovom pristupu.

### 2.1 Podela generativnih modela

Da bi se olakšali poređenje različitih modela, potrebno ih je generalizovati na one koji koriste **metodu maksimalne verodostojnosti**<sup>1</sup> i ignorisati one koji imaju drugačiji pristup. Osnovna ideja ove metode je da se procenjuje parametar  $\theta$  za koji **funkcija verodostojnosti** dostiže maksimalnu vrednost, odnosno traži se vrednost parametra za koju je najverovatnije da će se ostvariti realizovani uzorak. Funkcija verodostojnosti definiše se kao  $\prod_{i=1}^m p_{model}(x_i; \theta)$ . Dakle, ocenjujemo parametar  $\hat{\theta}$ :

$$\begin{aligned}\hat{\theta} &= \arg \max_{\theta} \prod_{i=1}^m p_{model}(x_i; \theta) = \\ &= \arg \max_{\theta} \log \prod_{i=1}^m p_{model}(x_i; \theta) = \\ &= \arg \max_{\theta} \sum_{i=1}^m \log p_{model}(x_i; \theta)\end{aligned}$$

Uvođenje logaritama olakšava račun jer se umesto proizvoda izračunava suma koja je otpornija na prisustvo veoma malih vrednosti. Drugi način formulacije problema je preko minimizovanja **Kulbak-Lajblerove divergencije** (eng. *Kullback-Leibler divergence*) između raspodele podataka iz trening skupa i raspodele koju generiše model:

$$\hat{\theta} = \arg \min_{\theta} D_{KL}(p_{podaci}(x) || p_{model}(x; \theta))$$

gde je

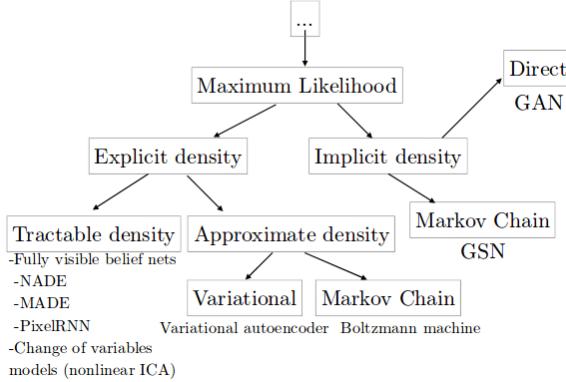
$$D_{KL}(p_{podaci}(x) || p_{model}(x; \theta)) = \sum_{i=q}^N p_{podaci}(x_i) \log \frac{p_{podaci}(x_i)}{p_{model}(x_i; \theta)} = E[\log p_{podaci}(x) - \log p_{model}(x; \theta)] \quad (1)$$

U praksi se ne zna raspodela  $p_{podaci}$  direktno, već su dostupni uzorci iz te raspodele. Na osnovu tih uzoraka, pravi se empirijska raspodela  $\hat{p}_{podaci}$  koja aproksimira  $p_{podaci}$ . Minimizacija Kulbak-Lajblerove divergencije između  $\hat{p}_{podaci}$  i  $p_{model}$  ekvivalentna je maksimizovanju logaritma verodostojnosti trening skupa.

Na slici 1 je prikazana podela generativnih modela zasnovanih na metodi maksimalne verodostojnosti koju predlaže Ijan Gudfelou (eng. *Ian Goodfellow*) u [1], ali će u nastavku rada fokus biti stavljen na GAN (eng. *Generative Adversarial Networks*).

---

<sup>1</sup>Mnogi modeli ne koriste ovu metodu, ali se mogu preraditi tako da je koriste, što je slučaj i sa GAN[1].



Slika 1: Podela generativnih modela.

## 2.2 GAN

Ideja iza GAN modela je zapravo igra između dva igrača, od kojih je jedan **generator**, a drugi **diskriminator**. Generator pokušava da sintetiše podatke koji prate raspodelu trening podataka, odnosno trudi se da na što bolji način reprodukuje originalne podatke. Diskriminator je binarni klasifikator koji pokušava da razluči da li su podaci lažni ili pravi. Generatoru je cilj da generiše dovoljno dobre podatke koje diskriminator neće uspeti da klasifikuje sa velikom sigurnošću. U primeru koji je naveden u [2] se generator predstavlja kao tim falsifikatora koji se trude da naprave lažan novac, dok je diskriminator policija i pokušava da razotkrije falsifikovan novac, a dopusti pravi. Ovo nadmetanje forsira oba igrača da se unapređuju sve dok generisane novčanice ne mogu da se razlikuju od pravih.

I generator i diskriminator su višeslojni perceptroni predstavljeni funkcijama  $G(z, \theta_g)$  i  $D(x, \theta_x)$  redom. Izlaz funkcije diskriminatora je verovatnoća da je  $x$  pravi podatak, a ne lažni, generisani. Ukoliko matematički formulišemo ovu igru, videćemo sličnost sa jednačinom 1:

$$\arg \min_G \arg \max_D V(D, G) = E_{x \sim p_{\text{podaci}}(x)} [\log D(x)] + E_{z \sim p_{\text{generator}}(z)} [\log 1 - D(G(z))]$$

Napomenimo da generator nema pristup podacima iz  $p_{\text{podaci}}$  već uči samo na osnovu interakcije sa diskriminatorom. Diskriminator ima pristup i pravim i generisanim podacima i svoju grešku procenjuje u skladu sa time.

U praksi se javlja problem sa ovom jednačinom prilikom treniranja. Naime, u početku je  $G$  loš, a  $D$  može s velikom sigurnošću da odbaci podatke koje generiše  $G$  jer su očigledno različiti od pravih. U tom scenariju,  $\log 1 - D(G(z))$  je blisko nuli pa  $G$  nema dobre izgledne za poboljšanje. Kako bi se ovaj problem rešio,  $G$  se ne trenira da minimizuje  $\log 1 - D(G(z))$ , već da maksimizuje  $\log D(G(z))$ . Ovaj pristup se ponaša isto kao i originalan, ali su gradjeni za  $G$  u početku mnogo pogodniji za trening.

Algoritam treninga modela dat je u nastavku 1. Umesto da se diskriminator istrenira do optimalnog rešenja u unutrašnjoj petlji, trenira se u  $k$  koraka nakon čega se u jednom koraku trenira generator. Ovim pristupom se diskriminator održava blizu optimuma sve dok generator uči umereno.

## 3 Implementacija

Koristeći se programskim jezikom Python, bibliotekom pytorch i idejama Gudfeloua i drugih, pokušaćemo da, na više različitim načina, konstruišemo generativni model. Prvi od njih se najvećim delom zasniva na [2] i predstavlja najosnovniju, "vanila" verziju modela. Nakon toga, prateći neke od saveta iz [3], pokušavamo da poboljšamo model. Na samom kraju biće prikazani i neki napredniji koncepti u rešavanju problema generisanja podataka korišćenjem GAN-a. U okviru svakog pristupa, biće prikazane korištene transformacije podataka koje mogu imati znatan uticaj na kvalitet modela.

---

**Algorithm 1** Treniranje GAN

---

```

for broj epoha do
  for k koraka do
    Izvuci uzorak  $\{z^{(1)}, \dots, z^{(m)}\}$  od m elemenata iz  $p_{generator}(z)$ 
    Izvuci uzorak  $\{x^{(1)}, \dots, x^{(m)}\}$  od m elemenata iz  $p_{podaci}(x)$ 
    Ažuriraj diskriminator uvećavanjem njegovog stohastičkog gradijenta:
```

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

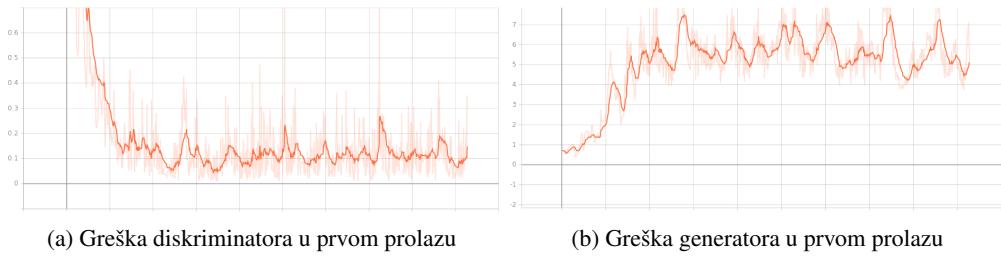
Izvuci uzorak  $\{z^{(1)}, \dots, z^{(m)}\}$  od m elemenata iz  $p_{generator}(z)$   
Ažuriraj generator smanjivanjem njegovog stohastičkog gradijenta:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m [\log 1 - D(z^{(i)})]$$


---

### 3.1 Vanila GAN

Skup podataka koji pokušavamo da reprodukujemo<sup>2</sup> sastoji se od skoro 16.000 slika mačijih lica u formatu 64x64. Iako su originalne slike u boji, mi smo primenili transformaciju Grayscale kako bismo smanjili vreme treninga modela<sup>3</sup>. Nakon toga, podaci su normalizovani tako da imaju srednju vrednost 0.5 i disperziju 0.5 opet u svrhu lakše i bolje obrade INSERTREF. Generator je potpuno povezana mreža koja prima vektor od 100 nasumičnih brojeva i propušta ih kroz 4 skrivena sloja. Izlaz iz generatora je 4096 piksela koji predstavljaju generisalu sliku. Diskriminator je takođe potpuno povezana mreža koja prihvata 4096 piksela, odnosno generisanu ili pravu sliku i nakon prolaska kroz 4 skrivena sloja kao odgovor daje svoju procenu da li je ulazna slika prava ili generisana. U skrivenim slojevima obe mreže koristi LeakyReLU funkciju aktivacije. U završnom sloju generatora koristi se hiperbolični tangens, a diskriminator u svom poslednjem sloju ima sigmoidnu funkciju aktivacije.<sup>4</sup> Procenat čvorova mreže koji se odbacuju prilikom propagacije unapred, odnosno dropout (eng. *dropout*) je 0.3. Koeficijent učenja je 0.00002, a algoritam za optimizaciju je Adam[4]. Trening se zasniva na algoritmu 1 za vrednost  $k = 1$ <sup>5</sup>. Zbog vremenskih i hardverskih ograničenja, treniranje je dva puta izvršeno u 30 epoha. Za vizualizaciju funkcija gubitka koristi se biblioteka tensorflowBoardX.



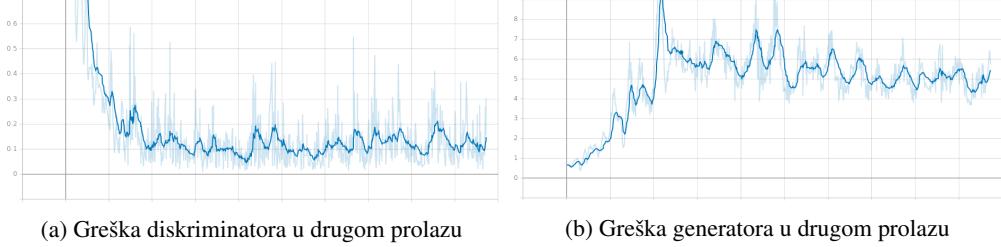
Sa grafikama grešaka generatora i diskriminatora vidimo da je u početku treninga diskriminator dosta loš i ne uspeva da raspozna stvarne od lažnih slika, dok generator ima veoma nisku vrednost greške. Kako vreme prolazi, vidimo da se diskriminator drastično poboljšava i greška mu se snižava, dok greška generatora raste. Kako bi generisane slike bile optimalne, potrebno je da dođe do konvergencije između grešaka generatora i diskriminatora, što se u našem slučaju nije dogodilo. Naime, vidimo da se diskriminator veoma brzo poboljša što dovodi do povećanja greške generatora, ali se na kraju ove dve veličine ne izjednače kao što bi se u optimalnom slučaju dogodilo. Ostaje pitanje da li bi treniranje u više epoha dovelo do boljih rezultata ili je loš odabir parametara algoritma učenja doveo do ovakvih rezultata. Napomenimo da je nad prikazanim graficima primenjena funkcija glaćanja (eng. *smoothing*) sa parametrom 0.9 kako bi se lakše uočio

<sup>2</sup>Za implementaciju vanila GAN-a vodili smo se već postojećim kodom sa sledećeg linka: <https://github.com/diegoalejom/gans>

<sup>3</sup>Naime, umesto da koristimo tri kanala i 64x64x3 piksela, koristimo samo 64x64x1 piksela i time smanjujemo potrebljeno vreme racunanja.

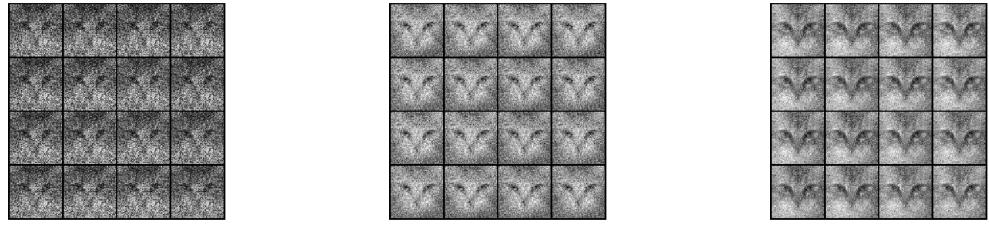
<sup>4</sup>U originalnom radu, generator koristi mešavinu ReLU i sigmoidnih aktivacionih funkcija, a diskriminator koristi maxout funkciju aktivacije.

<sup>5</sup>Na osnovu predloga u radu.



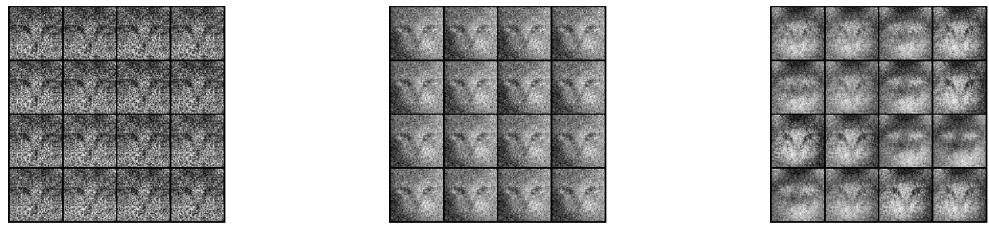
trend kretanja grešaka jer je puno šuma prisutno na originalnom grafiku (linija slabijeg intenziteta predstavlja grafik pre glaćanja).

Uprkos tome što greške nisu konvergirale, rezultujuće slike blago podsećaju na lica mačaka. Kao što smo već spomenuli, smatramo da bi veći broj epoha za treniranje doveo do boljih rezultata. Ispod su predstavljene mačke generisane od strane naše implementacije GAN-a.



(a) Generisane slike u 1. epohi      (b) Generisane slike u 15. epohi      (c) Generisane slike u 28. epohi

Slika 4: Prvi prolaz



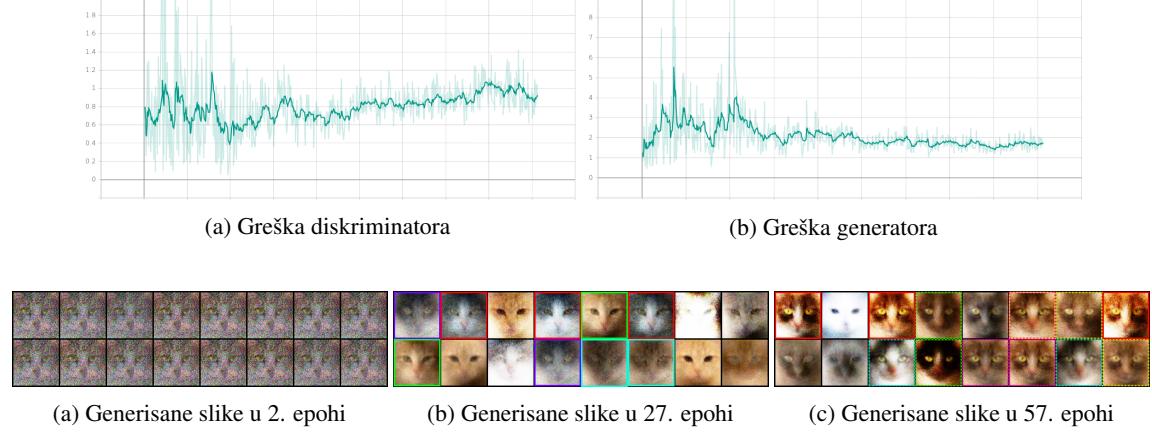
(a) Generisane slike u 1. epohi      (b) Generisane slike u 15. epohi      (c) Generisane slike u 30. epohi

Slika 6: Drugi prolaz

Na slikama ?? i ?? se jasno vidi da postoje generisane slike u okviru jedne epohe koje su međusobno veoma slične, što nam može sugerisati da generator u datoј epohi zna koji elementi slike najviše mogu zavarati diskriminator i na osnovu toga proizvodi slike koje imaju dat skup karakteristika, i to sa vrlo malo međusobnih razlika. Ovaj problem naziva se kolaps modusa (eng. *mode collapse*) i predstavlja jedan od najčešćih i najopasnijih problema prilikom treniranja GAN [5, 1, 3]. Iako je čest, uzroci i načini oporavka od ovog problema još uvek su predmet istraživanja. Pre nego što malo detaljnije opišemo ovaj fenomen i neke od tehnika za njegovo ublažavanje, navešćemo rezultate eksperimenta ponovljenog sa drugačijim parametrima.

Nezadovoljni rezultatima, ponovo smo pokrenuli trening, ali ovog puta sa slikama u boji i sa ažuriranim koeficijentom učenja, koji smo povećali za red veličine na 0.0002. Trening je izvršen u 57 epoha. Na slici ?? vidimo da se ponašanje grešaka dosta popravilo u odnosu na prethodni pokušaj, odnosno da se razlika između diskriminatora i generatora smanjila i da su obe mreže generalno stabilnije. U rezultujućim slikama i dalje se uočavaju sličnosti, ali u mnogo

manjoj meri. Dakle, promena koeficijenta učenja imala je pozitivan uticaj na kvalitet slike<sup>6</sup> iako kolaps modusa nije potpuno uklonjen. U nastavku ćemo trenirati model sa slikama u boji i novim koeficijentom učenja.



Slika 9: U početku su sve mačke međusobno veoma slične, ali već oko 15. epohe uočavamo diverzifikaciju generisanih slika. Uprkos većoj različitosti, i dalje se, u okviru jedne epohe, generiše značajan broj međusobno sličnih slika (uokvirenih istim bojama).

### 3.2 Problemi prilikom treninga

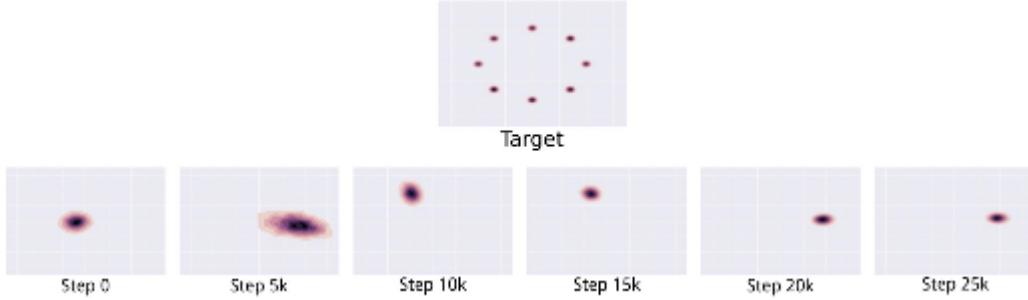
Iako su rezultati rada GAN modela impresivni, njihov trening je veoma nestabilan i teško je pogoditi optimalne parametre koji će dovesti do konvergencije oba modela i zadovoljavajućih rezultata. Kao što smo već videli u prethodnom odeljku, lako se može desiti da su generisane slike veoma slične ili da postoji značajna razlika između "moći" generatora i diskriminatora, odnosno njihov disbalans. Takođe smo videli da se gradjenci generatora lako mogu ugasiti u početku treninga ukoliko se odabere pogrešna funkcija za optimizaciju. Iako su GAN relativno nova ideja u svetu računarskog vida, broj objavljenih radova je u stalnom porastu, što dovodi do rađanja novih predloga i rešenja za neke od postojećih problema. Mi ćemo se u nastavku pozabaviti nekim od pomenutih poboljšanja.

---

<sup>6</sup>Smatramo da boja nije imala preveliki uticaj na rezultate, ali zbog manjka računske moći i vremena nismo stigli da potvrdimo tu pretpostavku.

### 3.2.1 Kolaps modusa

Kolaps modusa, poznat i kao Helvetica scenario (eng. *Helvetica scenario*) je problem koji se javlja kada generator nauči da relativno veliki broj ulaznih vektora slika u jednu tačku [1, 3]. Ovo se ogleda u velikoj sličnosti između slika u okviru jedne epohe. Potpuni kolaps modusa (sve generisane slike su identične) je redak u praksi, dok je delimičan kolaps češći (u svim generisanim slikama se javlja ista boja, tekstura ili postoji veći broj slika koje predstavljaju razne perspektive posmatranja istog objekta). Slika 11 (preuzeta iz [1]) ilustruje navedeni problem:



Slika 11: Na gornjoj slici vidimo raspodelu podataka koju model treba da nauči. U redu ispod vidimo različite distribucije naučene tokom treninga. Očigledno je da model nije naučio celokupnu raspodelu već se koncentriše na malu okolinu u različitim koracima, odnosno kolabira na jedinstvene moduse.

Jedan od mogućih uzroka za kolaps modusa proizilazi iz iterativnog algoritma 1 i činjenice da minimax problem nije identičan maximin problemu. Kao što je opisano u algoritmu, diskriminator se trenira za fiksiran generator u k koraka, nakon čega se generator trenira za fiksiran diskriminator. Prilikom računanja  $\max_D \min_G V(D, G)$ , gde je  $V$  funkcija vrednosti GAN modela, minimizovanje generatora se nalazi u unutrašnjoj petlji algoritma. Generatoru je dakle cilj da generiše podatak za koji će diskriminator poverovati da je stvaran. Iako je u stvarnosti raspodela podataka multimodalna, generatoru je dovoljno da se koncentriše na mali podskup tih modusa kako bi zavarao diskriminatora, što znači da generator pokriva mali broj različitih podataka. Nakon što diskriminator nauči da odbaci generisan podatak kao lažan, generator prebacuje fokus na drugu tačku za koju će diskriminator poverovati da je prava. Ovo smenjivanje može potrajati zauvek i ne dovodi do željene konvergencije.

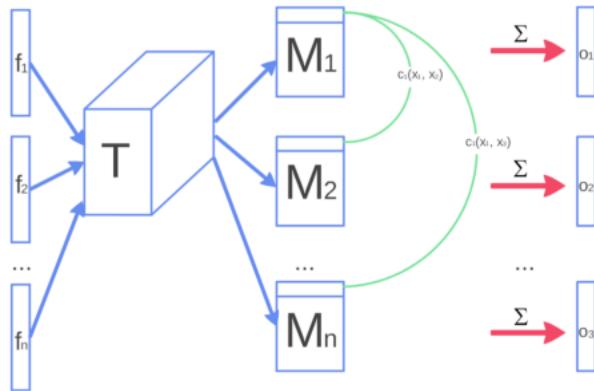
Pokušaj ublažavanja ovog problema jeste diskriminacija minibeća (eng. *Minibatch discrimination*)[3]. Kao što smo videli, problem se javlja kada su podaci generisani od strane generatora međusobno veoma slični. Ovo se može ublažiti tako što se diskriminatoru pružaju informacije o čitavoj grupi instanci umesto o samo jednom objektu. Ukoliko su podaci u jednoj grupi podataka slični, diskriminator lako može zaključiti da se radi o lažnim podacima i naterati generator da proizvodi međusobno različitije podatke. Neka je  $f(x_i) \in \mathbf{R}^A$  vektor koji predstavlja izlaz iz nekog međusloja diskriminatora za ulaz  $x_i$ . Dalje,  $f(x_i)$  se množi sa tenzorom  $T \in \mathbf{R}^{A \times B \times C}$ . Rezultat te operacije su matrice  $M_i \in \mathbf{R}^{B \times C}$ . Nakon toga se računa formula:  $c_b(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|M_{i,b} - M_{j,b}\|_{L_1}) \in \mathbf{R}$ . Izlaz  $o(x_i)$  za ovaj minibeć sloj za ulaz  $x_i$  je definisan kao suma  $c_b$ -ova do svih ostalih uzoraka:

$$\begin{aligned} o(x_i)_b &= \sum_{j=1}^n c_b(\mathbf{x}_i, \mathbf{x}_j) \in \mathbf{R} \\ o(x_i) &= [o(x_i)_1, o(x_i)_2, \dots, o(x_i)_B] \in \mathbf{R}^B \\ o(\mathbf{X}) &\in \mathbf{R}^{n \times B} \end{aligned}$$

Na kraju se izlaz  $o(\mathbf{x}_i)$  minibeć sloja spajaju sa  $\mathbf{f}(\mathbf{x}_i)$  koji su bili ulaz za taj sloj i rezultat tog spajanja se šalje narednim slojevima kao što je prikazano na slici ispod:

### 3.2.2 Jednostrano glaćanje oznaka

Još jedan problem koji se javlja kod GAN je to što diskriminator (kao i ostale duboke neuronske mreže) ima tendenciju da klasificuje sa velikom verovatnoćom, što može dovesti do toga da se gradijent generatora rano ugasi i onemogući mu učenje. U 2.2 je već pomenuto kako se izmenom funkcije gubitka može izbeći gašenje gradijenata generatora u početku treninga, a u ovom odeljku će biti opisan dodatan pristup rešavanju tog problema.



Slika 12: Diskriminacija minibeča

Jednostrano glaćanje oznaka (eng. *one-sided label smoothing*) je tehnika pomoću koje se diskriminator penalizuje ukoliko je previše siguran u svoju odluku, odnosno koristi mali broj karakteristika da klasificuje objekat. Tehnike koje se već koriste kod dubokih neuronskih mreža su dropout i regularizacija. Dakle, pošto diskriminator zavisi od malog broja karakteristika, generator se uči da reproducuje samo te karakteristike. Ideja je da se ublaži procena diskriminatora dodavanjem neke konstante na oznaku stvarne klase.

Ukoliko zamenimo oznake pozitivne klasifikacije sa  $\alpha$  i oznake negativne klasifikacije sa  $\beta$ , funkcija optimalnog diskriminatora postaje:

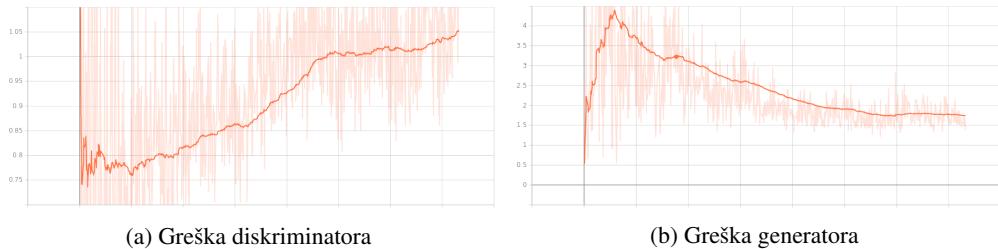
$$D^*(x) = \frac{\alpha p_{\text{podaci}}(x) + \beta p_{\text{model}}(x)}{p_{\text{podaci}}(x) + p_{\text{model}}(x)}$$

Problem se javlja u oblastima gde je  $p_{\text{podaci}}(x)$  malo, a  $p_{\text{model}}(x)$  veće, funkcija diskriminatora će imati veliku vrednost za pogrešne podatke. Diskriminator će, dakle, ohrabrivati pogrešno ponašanje generatora i neće ga motivisati da se približi željenoj raspodeli. Stoga je bitno glaćati samo pozitivne oznake, a negativne ostaviti na 0.

### 3.2.3 Eksperiment

U ovom odeljku ćemo prikazati rezultate naše implementacije navedenih poboljšanja. Arhitektura modela je slična onome opisanom u 3.1. Naime, generator je identična potpuno povezana mreža. Razlika je prisutna u diskriminatoru, koji je potpuno povezana neuronska mreža nalik na onu iz 3.1, sa dodatkom međusloja za diskriminaciju minibeča<sup>7</sup>. Izlaz iz minibeč sloja se prosleđuje izlaznom sloju koji slika 256 neurona u jedinstveni izlaz. Algoritam, kao i ostali parametri su identični kako bi se lakše uporedili rezultati<sup>8</sup>.

Vidimo da se greška generatora kreće u manjem intervalu nego u vanila verziji i da se u neku ruku stabilnije približava tački konvergencije. Greška diskriminatora je takođe manje skokovita i ravnomernije se kreće ka tački konvergencije. Uprkos tome, u globalu se greške ne razlikuju previše.



<sup>7</sup>Korišćena je biblioteka `torchgan`.

<sup>8</sup>Kao što je već napomenuto, koristi se koeficijent učenja 0.0002, a ne prvobitni od 0.00002. Takođe, jednostrano glaćanje oznaka je prisutno u ovom kao i u ostalim modelima zbog konzistentnosti, tako da ne možemo direktno posmatrati efekat ove tehnike na kvalitet učenja, ali verujemo da poboljšanje postoji kao što je navedeno u [3].

Uprkos boljem ponašanju grešaka i primeni saveta iz [3], smatramo da proizvedene slike nisu pokazale značajnije poboljšanje u odnosu na vanila verziju. Štaviše, vanila pristup je proizveo uzorke približno iste međusobne različitosti. Rezultati rada modela dati su na slici ispod.



Slika 14: (a) U ranijim fazama treninga, uočava se mali broj različitih slika reproducovanih u više uzoraka. (b) Čak i mačke u kasnoj fazi treninga deluju veoma slično međusobno iako postoji veći diverzitet. (c) Tek na samom kraju treninga počinje da se uočava veći broj različitih slika, ali među mačkama i dalje postoje "parovi" (uokvireni istim bojama).

Napominjemo da je evaluacija kvaliteta GAN predmet istraživanja i da je teško kvantitativno odrediti isti[1, 3], te se često koristi ljudska procena realističnosti generisanih slika. U nastavku ćemo objasnitи i implementirati još jedan od predloženih načina za suočavanje sa kolapsom modusa.

### 3.2.4 Odmotani GAN

Odmotani (eng. *unrolled*) GAN je predložen u [6] kao tehnika stabilizacije treninga generatora. Kao što smo već videli u prethodnim odeljcima, generator se često svede na mali podskup modusa. Ideja za uklanjanje ovog problema jeste da se za ažuriranje parametara generatora koristi "odmotani" diskriminator. Odmotavanje ima za svrhu da generatoru pruži uvid u dalju budućnost kako bi se takmičio protiv boljeg, pametnijeg diskriminatora, što bi kao efekat trebalo da proizvede i pametnijeg generatora. Ovo bi trebalo da spreči skakanje generatora između modusa i poveća različitost generisanih podataka.

Kod klasičnog treninga GAN cilj je pronaći optimalne parametre  $\theta_G^*$  za funkciju generatora  $G(z; \theta_G)$ :

$$\theta_G^* = \arg \min_{\theta_G} \max_{\theta_D} (\theta_G, \theta_D) = \arg \min_{\theta_G} f(\theta_G, \theta_D^*(\theta_G)) \theta_D^* = \arg \max_{\theta_D} f(\theta_G, \theta_D) \quad (2)$$

gde se za  $f$  koristi  $E_{x \sim p_{\text{podaci}}} [\log D(x; \theta_D)] + E_{x \sim p_{\text{model}}} [\log 1 - D(G(z; \theta_G); \theta_D)]$ . U [2] je dokazano da za ovako formulisan problem optimalan generator možemo predstaviti kao:

$$D^*(x) = \frac{p_{\text{podaci}}(x)}{p_{\text{podaci}}(x) + p_{\text{model}}(x)}$$

Ukoliko bi  $f(\theta_G, \theta_D)$  bilo konveksno u  $\theta_G$  i konkavno u  $\theta_D$ , onda bi se gradijentnim spustom došlo do tačke konvergencije[6]. U realnosti se ovo veoma retko dešava, i stoga GAN pati od kolapsa modusa i drugih problema.

Da bi se ovo izbeglo,  $\theta_D^*$  se može predstaviti kao  $\theta_D^*(\theta_G) = \lim_{k \rightarrow \infty} \theta_D^k$  gde se  $\theta_D^k$  induktivno definiše kao:

$$\theta_D^0 = \theta_D \quad (3)$$

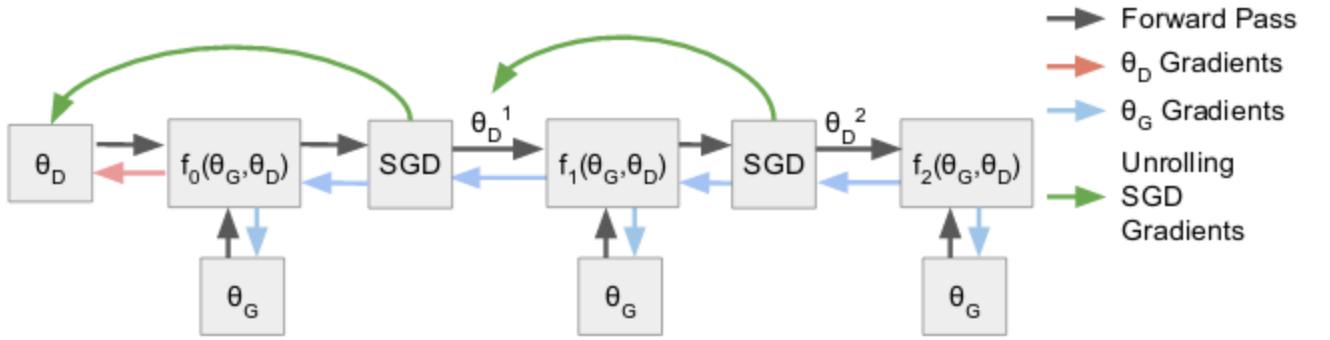
$$\theta_D^{k+1} = \theta_D^k + \eta^k \frac{df(\theta_G, \theta_D^k)}{d\theta_D^k} \quad (4)$$

Uz primedbu da se umesto gradijentnog uspona mogu koristiti i komplikovanije tehnike optimizacije. Dakle, kada se odmotavanje vrši u  $K$  koraka, dobija se sledeća funkcija cilja:

$$f_K(\theta_G, \theta_D) = f(\theta_G, \theta_D^k(\theta_G, \theta_D))$$

Ako bi  $K$  bilo jednako 0, funkcija cilja bila bi identična funkciji cilja klasičnog GAN modela, a kada  $K \rightarrow \infty$ , funkcija cilja se poklapa sa funkcijom cilja generatora,  $f(\theta_G, \theta_D^*(G))$ . Ažuriranje parametara se odvija na sledeći način:

$$\theta_G \leftarrow \theta_G - \eta \frac{df_K(\theta_G, \theta_D)}{d\theta_G} \quad (5)$$



Slika 16: Ažuriranje generatora podrazumeva propagaciju unazad kroz odmotanu optimizaciju (označeno plavim strelicama). Svaki korak  $k$  koristi gradijente  $f_k$  u odnosu na  $\theta_D^k$  kao u 4 (označeno zelenim strelicama). Takođe, diskriminotor ne zavisi od odmotavanja, što se vidi iz 6 (označeno crvenom strelicom).

$$\theta_G \leftarrow \theta_G + \eta \frac{df(\theta_G, \theta_D)}{d\theta_D} \quad (6)$$

Primetićemo da je u 5 za ažuriranje parametara generatora potrebno propagiranje unazad kroz 4. Kao ilustraciju ovog koncepta, navedena je sledeća slika:

Još jedna bitna stvar koju autori [6] ističu je da se ovaj pristup razlikuje od onog opisanog u [2] u algoritmu 1, jer se u tom pristupu koristi gradijentni spust (odnosno uspon) u odnosu na **fiksirane** parametre drugog modela, umesto u odnosu na novoformiranu funkciju cilja  $f(\theta_G, \theta_D^K(\theta_G, \theta_D))$ .

### 3.2.5 Eksperiment

U ovom odeljku biće opisana implementacija našeg modela. Reč je o sličnoj arhitekturi kao i u prethodnim primerima, sa bitnom razlikom u algoritmu treniranja generatora. Naime, generator dobija svoju odmotanu verziju diskriminatora na osnovu čijeg izlaza ažurira svoje težine. Nažalost, model nije uspeo da trenira duže od jedne epohe zbog problema sa radnom memorijom. Uzrok problema je verovatno veliki broj grafova izračunavanja koji se formiraju prilikom računanja gradijenata uz pomoć pytorch modula autograd. Kompletan kod za pokušaj ovog eksperimenta je dostupan na <https://github.com/rubidijum/gan>.

## 4 Drugačije arhitekture koje koriste GAN

Nakon pojave GAN-a, bilo je raznih pokušaja njegovog poboljšavanja, koje su se uglavnom bazirali na kombinovanju već poznatih metoda u cilju poboljšavanja problema koji se pojavljuju kod korišćenja GAN-a. Među njima su između ostalog: infoGAN, discoGAN, DCGAN itd. Mi ćemo se fokusirati na ideji iza DCGAN-a.

### 4.1 DCGAN

DCGAN (eng. *Deep Convolutional Generative Adversarial Networks*) je klasa konvolucionih neuronskih mreža sa određenim arhitektonskim ograničenjima namenjena za nenadgledano učenje. Osnovni koncept iza DCGAN-a je da se uz pomoć GAN-a nauče međupredstave slika iz kojih mogu da se izvuku određene karakteristike slika, koje kasnije mogu biti iskorišćene za različite oblike nadgledanog učenja, kao na primer za klasifikovanje slika. Pre nego što se upustimo u arhitekturu samog DCGAN-a, potrebno je objasniti šta su to konvolucione neuronske mreže. Ideja iza DCGAN je predložena u [7].

#### 4.1.1 Konvolucione neuronske mreže

Konvolucione neuronske mreže (eng. *Convolutional Neural Networks*), su vrsta neuronskih mreža koje su se pokazale vrlo efektivne u prepoznavanju i klasifikaciji slika. Jedna od prvih konvolucionih neuronskih mreža je LeNet5 mreža, koju je napravio Jan Lekun (*Yann LeCun*) 1990. godine. U poslednjih par godina je predloženo nekoliko novih arhitektura kao nadogradnja LeNet5 arhitekture, ali glavni koncepti LeNet5 arhitekture se nisu promenili. U toj arhitekturi postoje 4 važne operacije:

- Operacija konvolucije — svrha ove operacije je da se izvuku karakteristike ulazne slike;
- Uvođenje nelinearnosti — pošto je konvolucija linearna operacija, želimo da uvedemo nelinearnost u naš model, jer je i većina slika iz stvarnog sveta koje želimo da naša mreža nauči zapravo nelinearna;
- Operacija ujedinjavanja — ova operacija služi za smanjivanje dimenzije slike, pri tome pamтивши glavne karakteristike;
- Klasifikacija (korišćenjem potpuno povezanog sloja) — svrha ove operacije je da, koristeći karakteristike iz prethodnog sloja, klasificiše ulaznu sliku u klase naučene u trening periodu učenja.

#### 4.1.2 Arhitektura DCGAN-a

U [7] je predložena familija arhitektura za koju se pokazalo da daje dobre rezultate. Glavni fokus je na primeni određenih izmena u arhitekturi KNN-a.

Prva izmena je da se umesto operacije ujedinjavanja koristi operacija konvolucije sa većom dužinom koraka. Na taj način mreža može sama da nauči svoje prostorno proširenje i sažimanje (eng. *spatial upsampling*, *spatial downsampling*). Konvolucije sa većom dužinom koraka se koriste i u generatoru i u diskriminatoru.

Druga izmena za cilj ima eliminisanje potpuno povezanog sloja i povezivanje izlaza iz generatora sa ulazom u diskriminator.

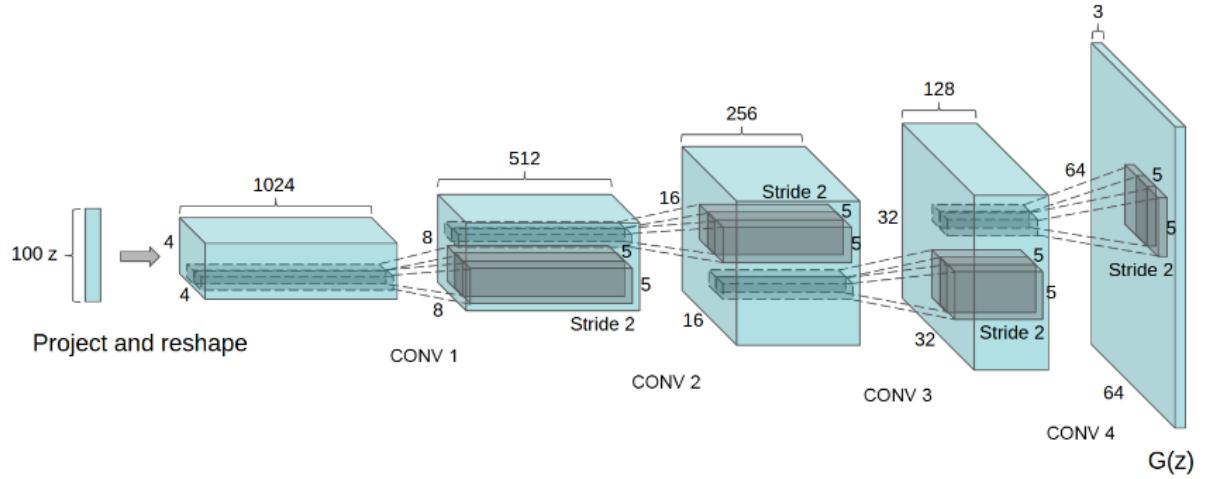
Treća izmena je uvođenje normalizacije gomile (eng. *Batch Normalization*), tj. svaku jedinicu ulaza normalizuje tako da ima srednju vrednost 0 i varijansu 1, što za posledicu ima ustaljivanje učenja. U [7] se navodi da ova izmena tera duboke generatore da uče, ujedno sprečavajući kolaps modusa (više reči o tome je bilo u 3.2.1). Takođe navode da je uvođenje normalizacije gomile u svaki sloj dovelo do oscilacija modela, tako da se normalizacija gomile izbacila sa izlaza iz generatora i ulaza u diskriminator.

Još jedna izmena koja se navodi je korišćenje ReLU aktivacije u generatoru, sem u poslednjem (izlaznom) sloju, u kojem se koristi hiperbolički tangens. Naime, u radu se komentariše da je korišćenje ove aktivacije bolje uticalo na to da model generatora brže nauči kako da koristi zasićenje i boje, dok je za model diskriminatora korišćena LeakyReLU aktivacija koja dobro utiče na model, posebno kada je reč o slikama sa većom rezolucijom.

#### 4.1.3 Eksperiment

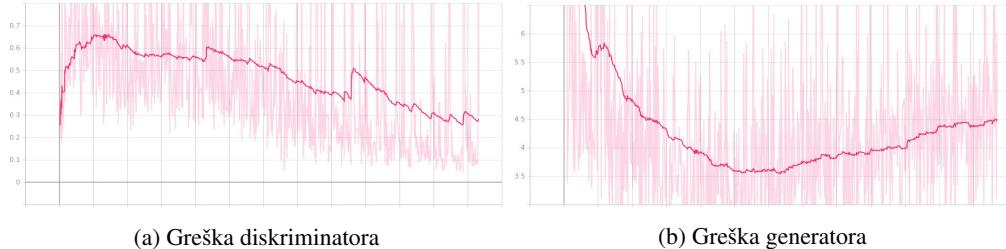
Ovde ćemo prikazati rezultate implementacije DCGAN-a, koju smo radili u programskom jeziku *Python*, vodeći se arhitekturom opisanom u [7] i priručnikom<sup>9</sup> za pravljenje DCGAN modela sa zvaničnog sajta PyTorch. Model generatora je konvolucionna neuronska mreža, koja prima vektor od 100 nasumičnih brojeva i propušta ih kroz konvolucioni sloj, praćen slojem koji vrši normalizaciju gomile, a potom ReLU slojem, i tako 4 puta. Nakon svakog sloja se postepeno povećava dimenzija uzorka. U poslednjem sloju se vrši operacija konvolucije praćena operacijom koja primenjuje

<sup>9</sup>Više o tom priručniku može da se pogleda na ovom linku: [https://pytorch.org/tutorials/beginner/dcgan\\_faces\\_tutorial.html](https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html)



Slika 17: Arhitektura DCGAN generatora

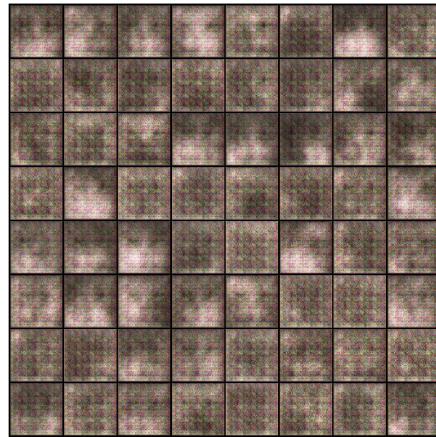
hiperbolički tangens i kao takva se prosleđuje diskriminatoru. Diskriminator je takođe konvolucionna neuronska mreža koja prima 4096 piksela po svakom kanalu (ima ih 3 – crveni, zeleni i plavi kanal) i arhitektura je slična kao i kod generatora, s tim što se umesto ReLu aktivacije koristi LeakyReLU (kao što je rečeno u [7]) i dimenzija uzorka se postepeno smanjuje, sve dok ne dođe do poslednjeg sloja u kojem se nakon operacije konvolucije primenjuje sigmoidna funkcija, čiji je rezultat izlaz iz diskriminatora. Važno je napomenuti je da su inicijalne težine za diskriminator i generator slučajno izabrane iz normalne raspodele sa srednjom vrednošću 0 i standardnom devijacijom 0.02, kao što je navedeno u [7]. Koeficijent učenja je 0.0002, za funkciju gubitka se koristi BCE (eng. *Binary Cross-Entropy*), a za primenu optimizacije je korišćen Adam optimizator. Treniranje je izvršeno u 100 epoha, a za vizuelizaciju funkcije gubitka je korišćena biblioteka tensorboardX.



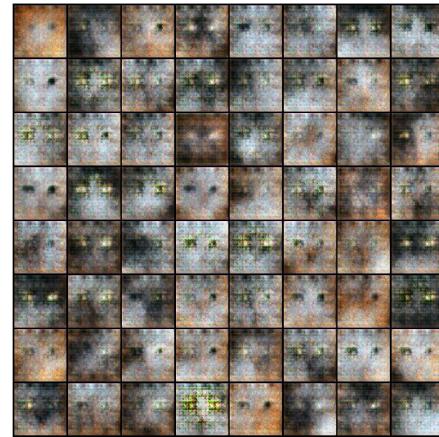
(a) Greška diskriminatora

(b) Greška generatora

Posmatrajući grafike grešaka diskriminatora i generatora možemo da primetimo da greška diskriminatora nema prevelike fluktuacije, što je i poželjno, dok je greška generatora u početku dosta velika (jer je diskriminator dovoljno dobar da šum dobro klasificuje), ali se sa vremenom drastično smanjuje sve do kraja, gde možemo primetiti blagi trend, ali nije značajna fluktuacija. Međutim, razlika između greške generatora i greške diskriminatora je veća nego što je poželjna i za 100 epoha ove dve greške nisu konvergirale. Samo posmatrajući grafike grešaka, moglo bi se zaključiti da model nije baš najuspešniji. Ali slike ipak pokazuju da je ovaj model uspeo u generisanju sasvim uverljivih mačaka. Na početku treninga vidimo da su mačke većinski slične, i takođe mogu da se primete granice filtera (uniformno raspoređene distorzije na slikama). Već u 10. epohi se primećuje veći diverzitet mačaka i znatno poboljšanje u sličnosti sa stvarnim mačkama. Od 40. pa do poslednje epohе nije se mnogo promenio izgled mačaka, već su se menjale fineze u izgledu mačaka.



(a) Generisane slike u nultoj epohi



(b) Generisane slike u 2. epohi



(a) Generisane slike u 10. epohi



(b) Generisane slike u 40. epohi



(a) Generisane slike u 75. epohi

(b) Generisane slike u 99. epohi

Slika 21: Mačke generisane DCGAN-om

## Literatura

- [1] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1701.00160, Dec 2016.
- [2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [3] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.
- [4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [5] Yongjun Hong, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon. How generative adversarial networks and their variants work. *ACM Computing Surveys*, 52(1):1–43, Feb 2019.
- [6] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2016.
- [7] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.