

Generiranje slika nalik Pokemonima koristeći DCGAN

Marijeta Pleskina

Prirodoslovno-matematički
fakultet u Zagrebu

Matteo Pašić

Prirodoslovno-matematički
fakultet u Zagrebu

Ena Škopelja

Prirodoslovno-matematički
fakultet u Zagrebu

Sažetak—GAN i DCGAN mreže su se kao arhitektura modela strojnog učenja pojavile tek prije nekoliko godina, i od tada daju iznimno obećavajuće rezultate u raznim zadacima iz područja sinteze umjetnih uzoraka. Jedan od njih je generiranje fotorealističnih slika. U ovom radu bavimo se generiranjem slika nalik Pokemonima u nadi dobivanja novih zanimljivih oblika. Napravili smo eksploratornu analizu skupa podataka te ga pripremili i augmentirali za proces treniranja mreža. Opisujemo arhitekturu generatora i diskriminadora te proces njihovog treniranja. Prikazujemo rezultate koji su nadmašili naša očekivanja i možemo reći da smo uspješno generirali nove Pokemone.

1. Uvod

Generativna suparnička mreža ili GAN (*Generative Adversarial Network*) sastoji se od dvije neuronske mreže: generatora i diskriminadora. Generator iz slučajnih podataka, najčešće slučajnih uzoraka iz normalne distribucije, generira neki uzorak. Diskriminator dobiveni uzorak klasificira kao pravi ili lažni, daje odluku o tome dolazi li podatak iz distribucije iz koje je uzorkovan skup podataka za treniranje. Generator uči kreirati uzorak tako da prevari diskriminatora, a paralelno diskriminator uči razlikovati originalne i lažne uzorke.

Duboka konvolucijska generativna suparnička mreža ili DCGAN (*Deep Convolutional Generative Adversarial Network*) je vrsta GAN-a koja sadrži konvolucijske i dekonvolucijske slojeve. Može se koristiti za generiranje slika koje nalikuju slikama iz skupa podataka.

1.1. Opis problema

Problem koji nas zanima je generiranje slika koristeći DCGAN. Koristimo skup podataka koji se sastoji od 721 slike različitih Pokemonona, preuzet s Kaggle-a ([2], Slika 1).

Cilj projekta je istrenirati generator i diskriminator slika Pokemonona. Nakon toga moguće je generirati realistične slike nalik onima iz originalnog skupa podataka. Dakle, cilj nam je kreirati nove Pokemone.

Kroz projekt namijenili smo istrenirati nekoliko DCGAN-ova s različitim arhitekturama koristeći različite hiperparametre i u konačnici odabratи onaj koji daje (većinski metodom pogledaj pa prosudi) najbolje rezultate.



Slika 1. Nekoliko slika iz Pokemon *dataset*.

Tijekom treniranja zanimljivo je pratiti kompeticiju između modela i kako kao rezultat toga generirane slike postaju sve realističnije.

2. Pregled dosadašnjih istraživanja

GAN je dizajnirao Ian Goodfellow s grupom znanstvenika 2014. godine (vidi [3]). DCGAN je prvi put prezentiran u radu iz 2016. (vidi [4]), kao alat za reprezentaciju slika. Ideja generiranja novih Pokemonona već je obrađena i dobiveni su zanimljivi oblici (vidi npr. [5] i [6]).

3. Skup podataka

Spomenuti Pokemon *dataset* će nam služiti kao *training set*. Potrebno je uskladiti dimenzije slika te augmentacijom uvećati broj slika u skupu podataka (na primjer zrcaljenjem u različitim smjerovima, zumiranjem, rotacijom i slično). Klasteriranjem skupa podataka te reduciranjem na 2 ili 3 dimenzije možemo dobiti dojam o vrstama Pokemona i postoje li neke grupacije među njima.

3.1. Klasteriranje

Podatke smo klasterirali koristeci k-means++ metodu s brojem klaster $k = 10$. Odabir $k = 10$ nametnuo se zbog



Slika 2. Centroidi klastera.



Slika 3. Nekoliko slika Pokemona koji pripadaju klasteru 2.

toga što je u takvoj konfiguraciji većina klastera podjednako popunjena (sadrži otprilike podjednako primera).

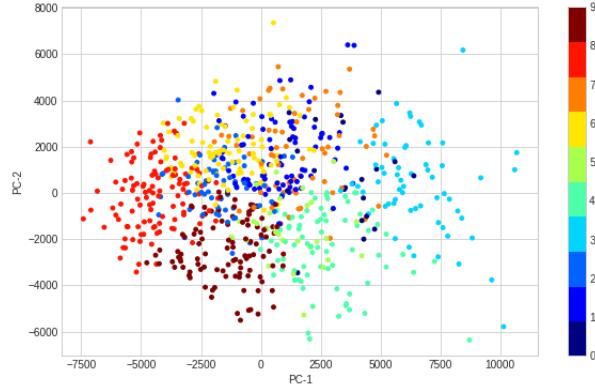
Kako bismo analizirali strukture naših klastera te procjenili koliko su dobro strukturirani, promatrali smo njihove centroide (Slika 2.). Kako centroidi klastera nemaju jasne karakteristike, daljnje zaključke donosili smo na temelju boja unutar centroma. Centroide smo s lijeva na desno i od gore prema dolje numerirali od 0 do 9.

Primjerice u klasteru 2, možemo vidjeti kako u centru klastera dominiraju crvena, žuta i narančasta boja. U principu se vide svijetlige boje u centru klastera, pa očekujemo da će većina Pokemona u klasteru 2 biti svijetlige boje. To se i vizualno može potvrditi promatranjem Slike 3 (u klasteru dominiraju crvena, žuta i narančasta boja).

Gledajući centar klastera 4, vidljivo je kako se dvije boje posebno ističu, crvena i siva boja (tj. "boja kamenja"). Možemo vidjeti kako je dosta Pokemona u tom klasteru neka varijacija ovih boja, ali unutar klastera ima pokemoni i drugih boja poput plave i crne, koje se nisu baš dale primjetiti unutar samog centra klastera. Što se tiče zaključivanja po bojama, vidimo da smo u centru klastera 4 mogli izdvojiti puno manje boja nego što se to zapravo vidi na podacima klastera 4.

3.2. Principal component analysis

Koristeći rezultate klasteriranja, tj. dobivene klasterne, napravili smo i *Principal component analysis* za $n = 2$ dimenzije na našem datasetu kao još jedan način interpretacije



Slika 4. Prikaz PCA za $n = 2$ dimenzije



Slika 5. Prikaz UMAP-a za $n = 15$ najbližih susjeda

naših podataka. Rezultat je prikazan na Slici 4. Tu možemo vidjeti da su neki klasteri stvarno dobro strukturirani, dok su neki raspršeni i imaju slabu strukturu, npr. klaster 0 je dosta raspršen kroz prostor, dok su klasteri 4,8,9 odlično strukturirani, a to su ujedno i klasteri iz čijih centara je bilo teže primjetiti kakvi su pokemoni u takvom klasteru.

3.3. 3D vizualizacija podataka

Radi eksploratorne analize podataka također smo pomoću TensorBoard-a napravili PCA i UMAP. Iz ovih metoda se jako teško može primjetiti nešto o strukturi podataka kao što vidimo na Slici 5.

Vektore s kojima smo radili ovu analizu dobili smo feature extractionom koristeći model EfficientNet B0 pre-treniran na Imagenet datasetu [11].

3.4. Priprema skupa podataka za treniranje

Dataset koji koristimo za treniranje GAN-ova je već spomenuti Pokemon dataset. Kako bismo smanjili dimensionalnost problema a time i vrijeme i resurse potrebne za



Slika 6. Prikaz Pokemon-a za 5 PIL-ovih augmentacija

treniranje modela, veličine slika koje koristimo ograničili smo na 64×64 piksela (u 3 RGB kanala). Ipak, pošto je u datasetu samo sedamstotinjak slika, kako bismo eliminirali mogućnost da diskriminator nauči dataset *na pamet* i tako prebrzo konvergira, dataset smo na dva načina augmentirali.

Prvi tip augmentacija koristi Pythonov Pillow (PIL) library i fizički dodaje slike u dataset, dok je drugi direktno ugrađen u diskriminator u formi Tensorflowovog Sequential modela sastavljenog od eksperimentalnih *preprocessing* slojeva. Primjetili smo kako kombinacija ovih dviju augmentacijskih metoda rezultira stabilnijim treniranjem i boljim rezultatima. Same augmentacije pomoću samo PIL-a dalju bolje rezultate ali pošto je i taj dataset konačan, divergencija losseva generatora i diskriminatora dogodi se nešto prije (a time je treniranje onda nestabilnije) nego kod dodatka augmentacija unutar diskriminatora. Samostalne augmentacije pomoću *preprocessing* slojeva daju stabilnije treniranje i ali slabije rezultate što je i za očekivati jer uvode manju varijaciju u originalni dataset (vidi dolje).

Augmentacije koje smo koristili s PIL-om su bile *hue*, *saturation* i *contrast*. Tako smo od početnih 721 slika došli na veličinu dataseta od 4326 slika. Koristili smo *green hue*, *red hue*, mjenjali saturaciju za faktore 2.5 i 0.5 te promjenili kontrast za faktor 2 i to za svakog Pokemon-a iz početnog *dataset-a*. Na Slici 6 možemo vidjeti sve PIL-ove vrste augmentacija koje smo koristili

Od ugrađenih augmentacija koristili smo random slojeve, tj. slojeve koji svakim pozivom rade *coin-flip* hoće li izvršiti zadani augmentaciju, a zatim, ukoliko je odgovor potvrdan, još jedan za uzorkovanje faktora s kojim će ju izvršiti. Takvi slojevi svakim pozivom rezultiraju različitom slikom.

Među njima odabrali smo random rotaciju, random horizontal flip i random zoom. Za random rotaciju smo koristili faktor od 0.1, kojim se u drugom *coin-flipu* kut bira nasumično u intervalu $[-0.2\pi, 0.2\pi]$. Random horizontal flip nema dodatne nasumičnosti van odabira izvršava li se na odabranom primjeru ili ne. Za random zoom smo koristili faktor od 0.1, što znači da će se za svaku sliku odabranu za ovu augmentaciju dodatno nasumično odabrati *zoom-in* ili *zoom-out* u rasponu od 0 do 10%.

Primjer ovakvih augmentacija može se vidjeti na Slici 7.



Slika 7. Prikaz Pokemon-a za augmentacije u *preprocessing* sloju

4. Model

U ovom poglavlju ćemo detaljnije opisati kako funkcioniraju GAN-ovi. Već smo spomenuli kako je posao generatora generirati slike (uzorke) koje nalikuju slikama iz skupa za treniranje, a posao diskriminatora prepoznati je li slika lažna (generirana) ili originalna (iz skupa podataka za treniranje.)

Kako trenirati GAN? Generator dobije slučajni uzorak na input-u iz njega generira sliku. Slučajan uzorak općenito u arhitekturama generatora odabran je kako bi se spriječilo da generator odredi jednu sliku koja savršeno nalikuje originalnoj distribuciji i na dalje producira samo nju, tzv. *mode collapse*. Diskriminator s druge strane klasificira slike iz skupa podataka za treniranje i generirane slike. Funkcija gubitka diskriminatora suma je binarne *cross entropy* funkcije između outputa za lažne slike i nul-vektora jednakih dimenzija te outputa za generirane slike i vektora jedinica jednakih dimenzija. Funkcija gubitka generatora, slično, računa se kao binarna *cross entropy* funkcije između outputa za lažne slike i vektora jedinica jednakih dimenzija. Drugim riječima, generator se nagrađuje za svaku generiranu sliku za koju je diskriminator pomislio da je Pokemon, a kažnjava za svaku koju je točno detektirao kao lažnu. Update težina modela vrši se gradijentnim spustom s optimizatorom Adam s parametrima: $learning_rate = 2e - 4$ te $\beta_1 = 0.5$ i standardnih $\beta_2 = 0.999$ i $\epsilon = 1e - 07$. Ravnoteža se postiže kad generator uspješno generira slike koje izgledaju kao da su iz originalnog dataset-a, a diskriminator ih sa 50% sigurnošću klasificira kao prave ili lažne.

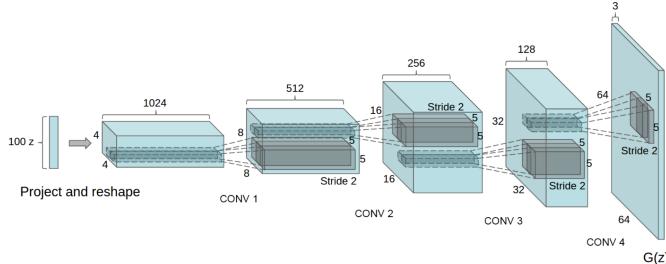
Uvedimo neke oznake kako bismo formalnije opisali ovaj model. Neka je x podatak (slika) i $D(x)$ output diskriminatora (vjerojatnost da je slika iz originalnog dataseta). Neka je z vektor iz latentnog prostora uzorkovan iz distribucije $p(z)$. Tada je $G(z)$ output generatora koji preslikava latentni vektor z u prostor slika. Neka je p_{data} distribucija iz koje dolaze originalne slike iz skupa podataka za treniranje. Primijetimo da je $D(G(z))$ vjerovatnost da je generirana slika $G(z)$ prava.

Diskriminator pokušava maksimizirati vjerovatnost da točno klasificira prave i lažne slike, a generator pokušava minimizirati vjerovatnost će te diskriminator prepoznati generirane slike kao lažne. Ovo matematički zapisujemo na način:

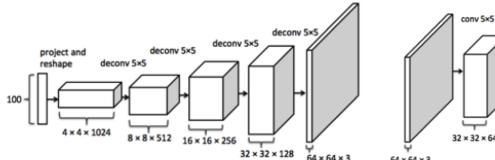
$$\min_G \max_D V(D, G)$$

gdje je

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



Slika 8. Primjer generatora.



Slika 9. Primjer arhitekture generatora (lijevo) i diskriminatora (desno).

4.1. Arhitektura

4.1.1. Generator. Na Slikama 8 i 9 prikazan je primjer DCGAN generatora korišten u [4]. Ulaz u model je 100 dimenzionalni vektor iz uniformne distribucije koji nakon jednog potpuno povezanog sloja prolazi niz blokova koji se sastoje od dekonvolucije (*upsampling*) s kernelom 5×5 i strideom 2×2 , *Batch Normalization* slojeva i *ReLU* nelinearnosti (dimenzije se mogu vidjeti na gore spomenutoj slici).

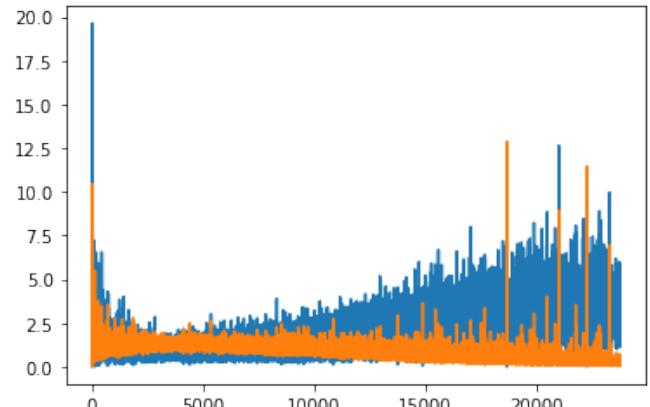
Dekonvolucijski blokovi generatora ulaznim tenzorima smanjuju dubinsku i proširuju prostorne dimenzije. Na izlazu generiraju se tenzori dimenzije $64 \times 64 \times 3$ kakav je i ulaz u diskriminator.

4.1.2. Diskriminator. Na Slici 9 prikazan je primjer DC-GAN diskriminatora korišten u [4]. Ulaz u model je tenzor dimenzije $64 \times 64 \times 3$ koji prolazi niz blokova konvolucije (*upsampling*) s kernelom 5×5 i strideom 2×2 (dimenzije se mogu vidjeti na gore spomenutoj slici). Na izlazu generira tenzor dimenzije 1 koji označava odluku nalazi li se primjer u generiranom ili originalnom skupu podataka. Konvolucijski blokovi diskriminatora ulaznim tenzorima povećavaju dubinsku i smanjuju prostorne dimenzije.

Našu arhitekturu može se vidjeti na 16 i 17. U odnosu na referentni primjer arhitektura nije značajno promijenjena. Dodatak diskriminatoru je dodatan blok konvolucija sa kernelom 3×3 i strideom 2×2 na ulazu u model te dodaci na oba modela koji su značajno popravili rezultate: u prvom bloku nakon potpuno povezanog sloja generatora i u zadnjem bloku prije izlaznog sloja diskriminatora uklonjeni su slojevi *Batch Normalization* [4].

5. Rezultati

Za treniranje naših neuronskih mreža koristili smo ranije opisan Pokemon dataset, uz batch size 128. Kroz batcheve



Slika 10. Prikaz funkcije gubitka za generator (plava) i diskriminator (narandžasta) u ovisnosti o broju training stepova.

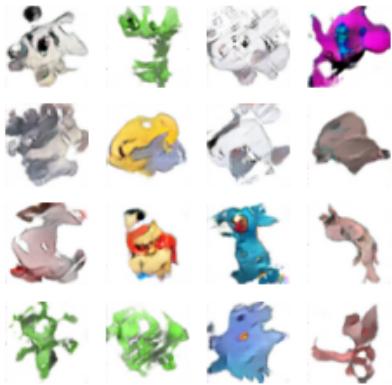
smo pratili funkcije gubitka generatora i diskriminatora (*generator loss* i *discriminator loss*) te smo za konstantnih 16 slučajnih vektora iz normalne razdiobe pratili izlaz iz generatora. Prikaz grafa treniranja nakon 700 epoha vidljic jena Slici 10. Treniranje smo prekinuli na 700. epohi zbog toga što su funkcije gubitka generatora i diskriminatora počele sve više divergirati što je ukayivalo da učenje više nije produktivno te zbog primjetnog malog pogoršanja u kvaliteti generiranih slika. U našem GitHub repozitoriju [10] nalazi se Pokemon.gif datoteka u kojoj je animacija slika generiranih iz prethodno spomenutih 16 random noise vektora kroz 700 epoha treniranja, dok ćemo mi ovdje izdvojiti stanje 600. i 700. epohu koje se nalaze na slikama 11 i 12. Uspoređujući sliku 11 te rezultate [6] na slici 14 vidimo razliku u rezultatima jer na slici 13 ne vidimo samo oblike koji liče na pokemone, nego vidimo da generator generira i neke značajke lica vidi 11

Prilikom testiranja modela napravili smo još i provjeru imo li naš model dovoljnu reprezentacijsku moć da nauči Pokemone. U toj provjeri smo uzeli sliku Bulbasaura 15 te smo za dataset uzeli 2048 identičnih slika Bulbasaura bez ikakvih augmentacija te smo isključili preprocessing layer za augmentacije. Pustili smo generator da uči na takvom datasetu te smo nakon 40 epoha dobili 15 kao rezultat. Također u [10] se nalazi Bulbasaur.gif u kojem su slike nastale generiranjem iz istog šuma nakon svake epohe.

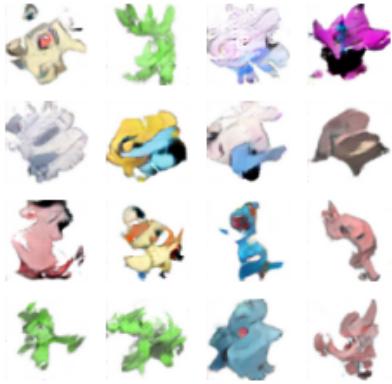
Na modelu treniranom 600 epoha napravili smo linearno istraživanje latentnog prostora. Odabrali smo jedan slučajan vektor iz latentnog prostora i u njegovom smjeru se kretali kroz prostor i bilježili slike koje za njih dobivamo iz generatora. Tako smo dobili datoteku latent.gif koja se također nalazi u [10].

6. Drugi pristupi temi

Naša tema je problem *image synthesis*. Kako smo mi koristili DCGAN za *image synthesis*, jedan od drugih pristupa je korisiti druge GAN-ove. Jedan pristup je korištenje



Slika 11. Prikaz 16 generiranih Pokemona nakon 600 epoha.



Slika 12. Prikaz 16 generiranih Pokemona nakon 700 epoha.



Slika 13. Prikaz 9 odabranih nakon 600 epoha.

conditional GAN-ova za image synthesis (vidi [7]). Također mogu se koristiti StyleGAN-ovi, PGGAN-ovi itd.

Što se tiče pristupa koji ne koriste GAN-ove, jedan



Slika 14. Rezultati [6]



Slika 15. Slika Bulbasaura (lijevo) i slika koju je generirao generator (desno)

od boljih pristupa su difuzijski modeli. Difuzijski modeli generiraju slike iz distribucije pomoću obrnutog gradual noising processa. U [8] je ta distribucija dijagonalna Gausssova distribucija. Preciznije, krećemo od primjera slike sa šumom x_T i postepeno se dobivaju slike sa manjim šumovima x_{T-1}, x_{T-2}, \dots sve dok ne dobijemo zadnju sliku x_0 . Zanimljivo je da pristup [8] ima najmanji FID score na primjeru *image synthesisa* za *dataset LSUN Bedroom 256 × 256* [9]. Također, jedan od trenutno najrazvijanijih pristupa, a i onaj koji najviše u tom smjeru obećava su Transformerske *self-attention* arhitekture. [11]

7. Zaključak

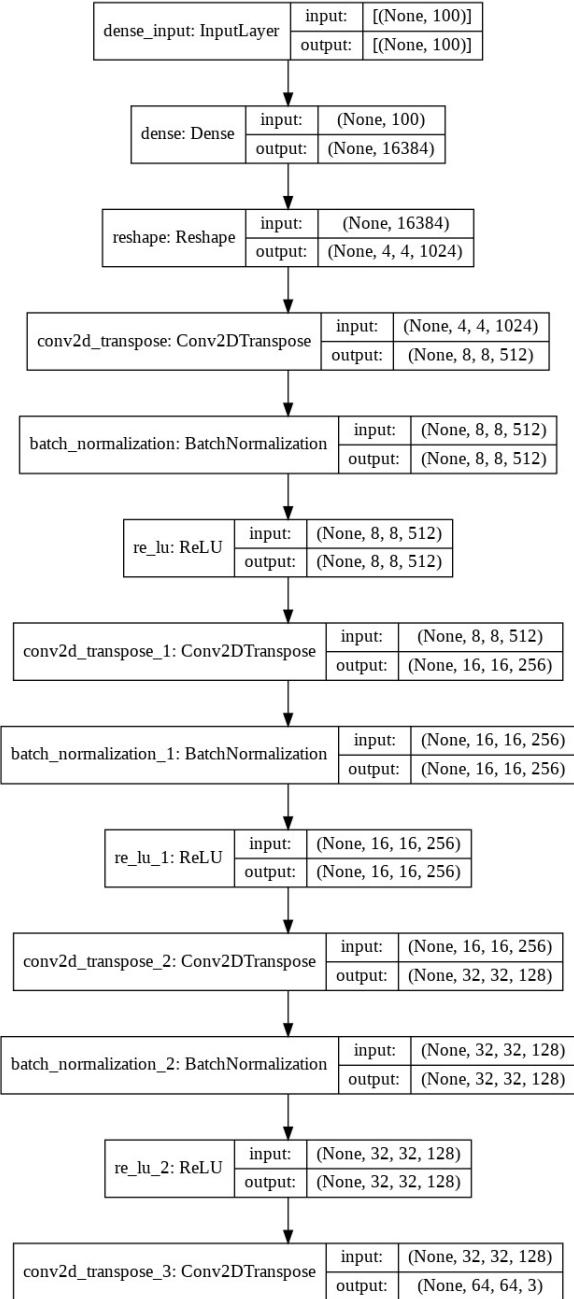
U ovom radu smo predstavili kako generirati Pokemone pomoću DCGAN-ova. Taj se pristup pokazao boljim od očekivanog u ovom zadatku i ostavlja dosta prostora za zanimljive nadogradnje. Što se tiče mogućih budućih istraživanja u vezi ove teme, kako su nam slike bile veličine 64×64 , jedan smjer bi bio ići povećavati veličinu slike i gledati kako će to utjecati na generiranje. Također uz to ide i prilagođavanje same arhitekture. Neke od mogućih poboljšanja su također i otežavanje treniranja generatora i diskriminadora da pokušamo proizvesti broj epoha, npr. s dodavanjem šuma na input diskriminatora te dodavanjem šuma na output svakog layera diskriminatora. [13]. Također ostaje i isprobati potpuno drugačiji pristup u vidu Transformerske arhitekture.

Literatura

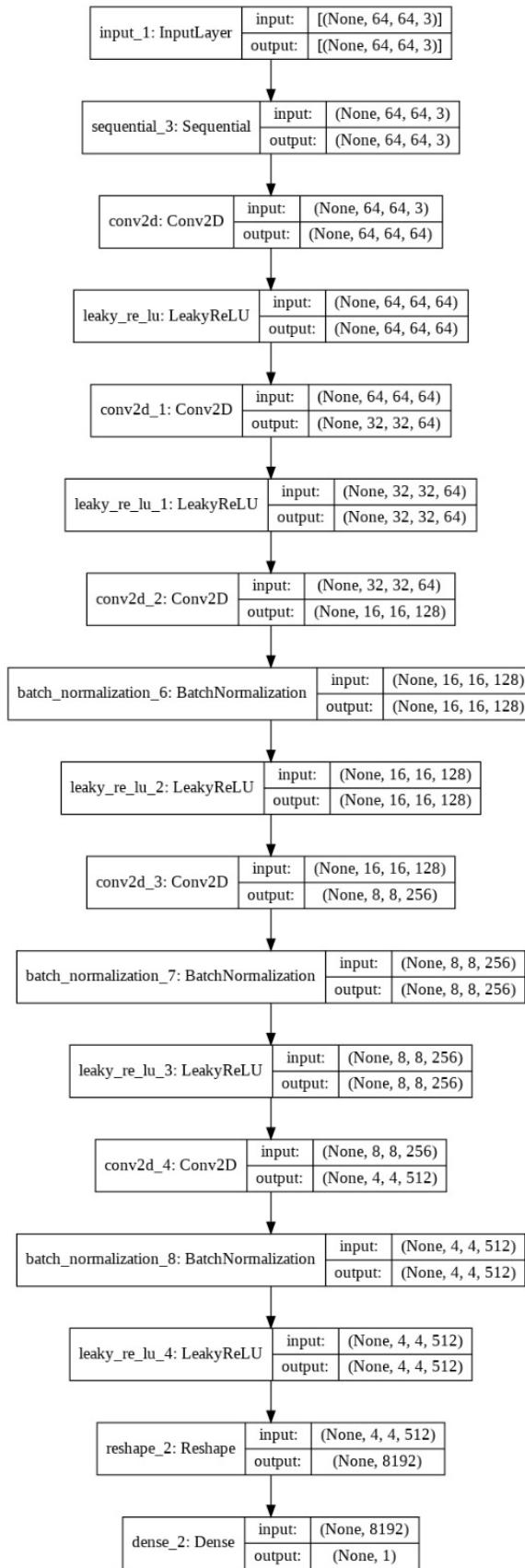
- [1] <https://www.tensorflow.org/tutorials/generative/dcgan>
- [2] <https://www.kaggle.com/kvpratama/pokemon-images-dataset>
- [3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio (2014.). *Generative Adversarial Nets*.

- [4] A. Radford, L. Metz, S. Chintala (2016.). [Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks](#).
- [5] <https://github.com/kvpratama/gan/tree/master/pokemon>
- [6] <https://towardsdatascience.com/how-to-create-unique-pokemon-using-g-gans-ea1cb6b6a5c2>
- [7] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, Bryan Catanzaro, NVIDIA Corporation, UC Berkeley (2018.). [High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs](#)
- [8] Prafulla Dhariwal, Alex Nichol (2021.). [Diffusion Models Beat GANs on Image Synthesis](#)
- [9] <https://paperswithcode.com/sota/image-generation-on-lsun-bedroom-256-x-256>
- [10] <https://github.com/marjetap/Tim-Ksi>
- [11] Mingxing Tan, Quoc V. Le (2020.). [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](#)
- [12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin (2017.). [Attention Is All You Need](#)
- [13] Simon Jenni, Paolo Favaro [On Stabilizing Generative Adversarial Training with Noise](#)

8. Dodatak



Slika 16. Vizualizacija našeg modela generatora.



Slika 17. Vizualizacija našeg modela diskriminatora.