SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1557

Izlučivanje slikovnih reprezentacija generativnim suparničkim modelima

Filip Zelić

Zagreb, veljača 2018.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljem mentoru prof. dr. sc. Siniši Šegviću na stručnoj pomoći i prijedlozima tijekom rada na studiju. Također zahvaljujem se svojoj obitelji, prijateljima i kolegama na potpori kroz obrazovanje tijekom studija.

SADRŽAJ

1. Uvod

U zadnje vrijeme nadzirano učenje s dubokim konvolucijskim mrežama počelo se primjenjivati u sve većem broju aplikacija računalnog vida. Dok s druge strane nenadzirano učenje nije imalo takve uspjehe. U ovom radu se bavimo nenadziranim učenjem jer otkrivanjem novih generativnih modela počeli su se postizati bolji rezultati u njihovim primjenama. Nenadzirano učenje je vrlo zanimljivo područje jer nije potrebno ručno označavati velike skupove podataka te tako moguće je iskoristiti ogromne količine podataka za treniranje dubokih modela.

Točnije bavit ćemo se generativnim modelima koji omogućuju generiranje slika iz distribucije stvarnih podataka te izvlačenje slikovnih reprezentacija. Navedeni generativni modeli zasnivaju se na suparničkim neuronskim mrežama. Generativne suparničke mreže sastoje se od dviju neuronskih mreža, generatora i diskriminatora. Generator pokušava generirati primjere koji dolaze iz iste distribucije kao i podaci za treniranje. Diskriminator dobiva na ulazu generirane primjere i primjere iz podatkovnog skupa te ih procjenjuje kao stvarne ili generirane. Ideja je da dvije mreže igraju igru gdje generator pokušava nadmudriti diskriminator i stvoriti primjere koje ne može razlikovati od primjera iz podatkovnog skupa. Kroz postupke optimizacije, obje mreže poboljšavaju svoj cilj. Jedan od fokusa ovog rada je postupno vidjeti razvoj generativnih suparničkih mreža do trenutno stabilnih modela. Nakon toga utvrditi koji modeli generiraju vizualno kvalitetnije primjere, a koji postižu najbolje rezultate u učenju slikovnih reprezentacija.

Rad kroz nekoliko dijelova opisuje arhitekturu korištenih mreža, implementaciju te dobivene rezultate. Prvi dio rada opisuje osnovne dijelove dubokih konvolucijskih mreža i algoritme koje koristimo kako bi izgradili duboke generativne modele. Drugi dio opisuje same generativne suparničke mreže, različite vrste takvih mreža te njihove prednosti i nedostatke. Nakon čega slijedi opis implementacijskih detalja modela i korištenih podatkovnih skupova. Peto poglavlje opisuje provedene eksperimente i dobivene rezultate prilikom korištenja opisanih modela. Konačno, zaključak daje buduće smjerove rada na problemu i moguća poboljšanja.

2. Duboke konvolucijske neuronske mreže

U ovom poglavlju će biti objašnjeni temelji konvolucijskih mreža, aktivacijskih funkcija te optimizacijskih postupaka za učenje takvih dubokih modela. Navedeni gradivni elementi modela detaljnije su opisani u literaturi.

2.1. Duboko učenje

Duboko učenje kao grana strojnog učenja zabilježila je brojne uspjehe posebno u primjenama u području računalnog vida. Razvojem područja počeli su se koristiti novi modeli neuronskih mreža te robusniji algoritmi pronalaženja optimalnih parametara za pojedine zadatke. Razvoju je posebno pridonio napredak računalne snage, iskorištavanje grafičkih kartica za treniranje neuronskih mreža te veliko povećanje dostupnih podatkovnih skupova. Neuronske mreže općenito su modeli koji se sastoje od skupa međusobno povezanih osnovnih jedinica - neurona. Izlaz jednog neurona funkcija je ulaza i parametara odnosno težina neurona.

$$y = f(\vec{x} \cdot \vec{w} + b) = f(\sum_{i=1}^{n} x_i \cdot w_i + b) \tag{2.1}$$

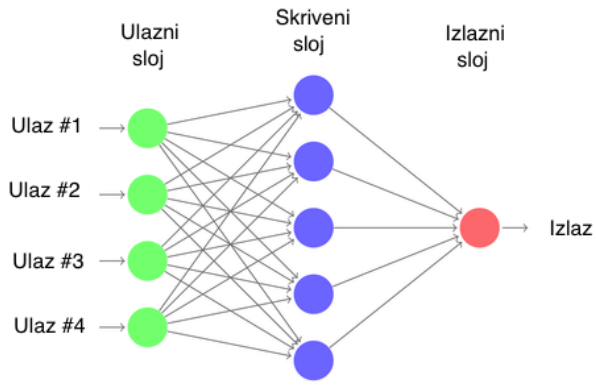
Većina neuronskih mreža organizirane su po grupama neurona koje nazivamo slojevima. Slojevi su lančano povezani tako da izlazi jednog sloja su ulazne vrijednosti za sljedeći sloj. Struktura izgleda sljedeće, prvi sloj je definiran:

$$h^{(1)} = f^{(1)}(W^{(1)\top} x + b^{(1)}) \tag{2.2}$$

sljedeći sloj:

$$h^{(2)} = f^{(2)}(W^{(2)\top} h^{(1)} + b^{(2)}) \tag{2.3}$$

i tako do zadnjeg sloja definirane mreže. Oznaka $h^{(i)}$ prikazuje skrivene jedinice i-tog sloja mreže, $W^{(i)}$ je matrica težina sloja te $b^{(i)}$ parametar koji nazivamo prag (engl. bias). Funkcija f je definirana aktivacijska funkcija koja uvodi nelinearnost između slojeva.



Slika 2.1: Jednostavna neuronska mreža

Duboke neuronske mreže su naziv za mreže koje sadrže dva ili više skrivena sloja. One omogućuju modeliranje vrlo kompleksnih nelinearnih zavisnosti kakve pronalazimo u podacima. Duboke mreže koje sadrže konvolucijske slojeve nazivaju se konvolucijske neuronske mreže i najviše se koriste u primjenama računalnog vida.

2.2. Konvolucijski sloj

Konvolucijske neuronske mreže su temeljene na radu vidnog korteksa (engl. visual cortex). Istraživanja su pokazala da vidni korteks ima male regije stanica koje su osjetljive na određena područja vidnog polja. Ovu ideju proširili su Hubel i Wiesel 1962. eksperimentom gdje su pokazali da neke pojedine neuronske stanice u mozgu reagiraju samo u nazočnosti rubova određene orijentacije. Na primjer, neki neuroni otpuštaju signale kada su izloženi okomitim rubovima, a neki kada se prikazuju vodoravni ili dijagonalni rubovi. Hubel i Wiesel otkrili su da su svi ti neuroni organizirani u arhitekturi po stupnjevima složenosti te zajedno stvaraju vizualnu percepciju.

Konvolucijske neuronske mreže osim strukturne informacije imaju za cilj iskoristiti i prostornu informaciju iz piksela slike. Konvolucijski sloj temelji se na linearnoj transformaciji koja se naziva konvolucija. Podaci na računalu su diskretni stoga koristimo

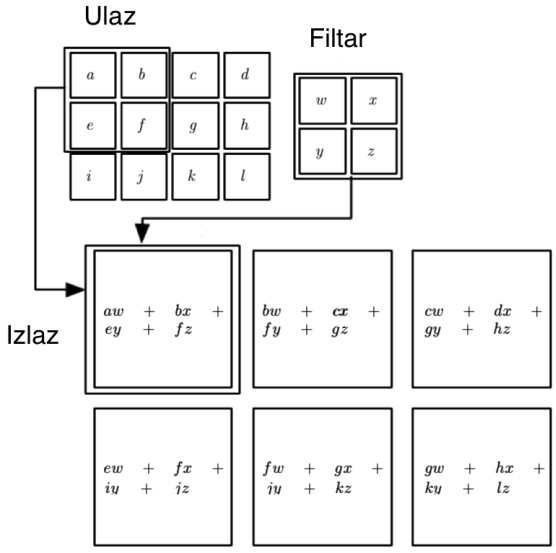
diskretnu konvoluciju definiranu:

$$I(i) * K(i) = \sum_m I(m)K(i-m) \tag{2.4}$$

gdje je I ulaz konvolucije, a K filtar (engl. kernel). Konvolucije možemo koristiti za dvodimenzionalne podatke, kao što su slike, tada je i filtar dvodimenzionalni.

$$I * K(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.5}$$

Izlazi dvodimenzionalne konvolucije nazivaju se mape značajki. Konvolucija se vrši na sljedeći način, unutar dvodimenzionalnog ulaza definira se prozor veličine filtra koji se naziva receptivno polje. Vrijednosti ulaznog dijela unutar prozora se množe s filtrom i sumiraju te daju izlaze za pojedine dijelove izlazne mape značajki. Filtar se primjenjuje tako da se pomiče kroz cijeli ulaz, pri čemu se receptivna polja preklapaju kao što je prikazano na slici (2.2). U modernim konvolucijskim dubokim modelima najčešće se koriste kvadratne jezgre dimenzije k gdje je k neparan broj (uglavnom se koriste veličine 3,5 ili 7). Jedan konvolucijski sloj sadrži više takvih filtra što ima za posljedicu da nastaje i više mapa značajki.



Slika 2.2: Izgled dvodimenzionalne konvolucije [9]

Parametri koji određuju dimenzije izlazne mape su S korak konvolucije (engl. stride), P popunjavanje do rubova (engl. padding) te veličina filtra k i dimenzije ulaza $m \cdot n$, gdje je m širina slike, a n visina slike. Korak konvolucije određuje pomak

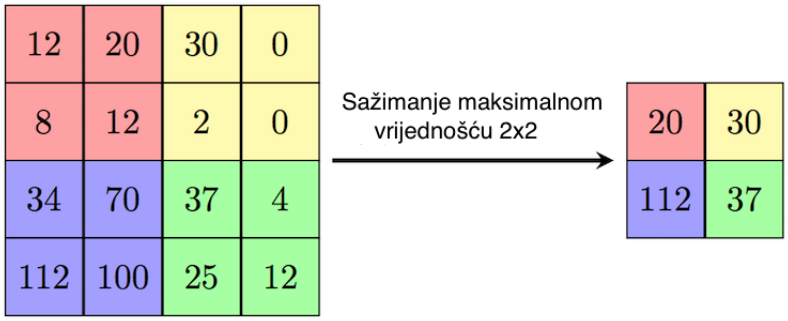
prozora po širini, odnosno visini ulazne mape značajki, a popunjavanje do rubova je vrlo korisno u slučaju kada je potrebno zadržati rezoluciju nakon konvolucijskog sloja. Izračun dimenzija izlazne mape će biti definiran sljedećim formulama:

$$x = \frac{m + 2P - k}{S} + 1 \qquad y = \frac{n + 2P - k}{S} + 1 \tag{2.6}$$

gdje je x širina izlazne mape, a y visina izlazne mape [9].

Bitna prednost konvolucijskih mreža je puno manji broj težina u svakom sloju u odnosu na potpuno povezani sloj. To se postiže zato što filtar je puno manji od ulazne slike. Prilikom obrade slike, ulazna slika može imati tisuće ili milijune piksela, ali korištenjem konvolucije s filtrom spremaju se značajke kao što su rubovi koji zauzimaju samo desetke ili stotine piksela. To znači da moramo pohraniti manje parametara, što smanjuje memorijske zahtjeve modela i poboljšava statističku učinkovitost. Ako je ulazna slika u boji tada dodatno definiramo r kao broj kanala (RGB slika). Tada je ulazna slika $m \cdot n \cdot r$, a dimenzionalnost konvolucijskog sloja ovisi o veličini filtra k, broju kanala prethodnog sloja r te broju značajki trenutnog sloja f. Broj težina trenutnog sloja se računa prema formuli: $k \cdot k \cdot r \cdot f$. Vidimo da dimenzionalnost sloja ne ovisi o ulazu, u ovom slučaju slike.

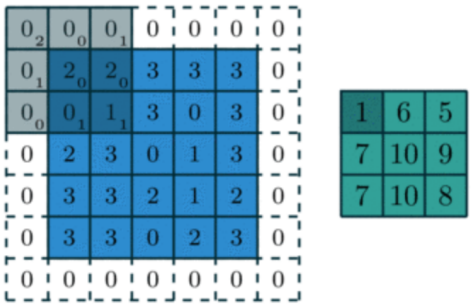
Sloj sažimanja je druga vrsta konvolucijskog sloja gdje se obavlja vrsta poduzorkovanja ulazne slike ili ulazne mape značajki. Ulaz ćemo podijeliti na dijelove jednakih dimenzija $k \times k$, gdje je k konstanta sažimanja čime se ulazne mape značajki smanjuju za taj faktor.



Slika 2.3: Sažimanje maksimalnom vrijednošću

Kod sažimanja maksimalnom vrijednošću (engl. max pooling), u svakom djelu uzimamo maksimalnu vrijednost unutar prozora (slika 2.3) dok kod sažimanja srednjom vrijednošću (engl. average pooling) uzimamo srednju vrijednost. Postupak sažima-

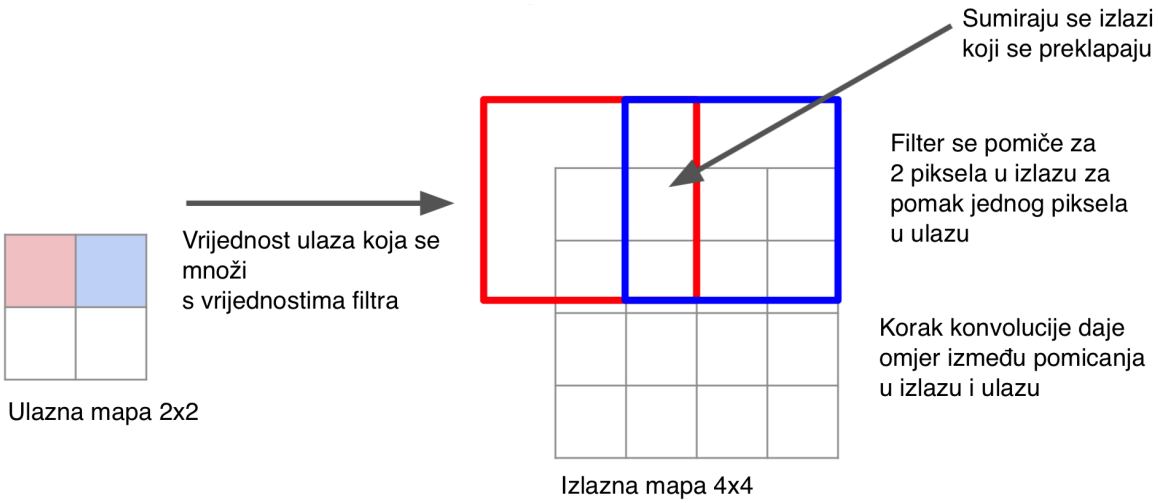
nja ostvaruje invarijantnost na lokalne translacije te može biti koristan kada nam je bitnije da neka značajka postoji nego da znamo na kojoj je točno lokaciji na ulazu [9]. Umjesto sloja sažimanja sličan efekt se može postići koristeći konvolucije s većim korakom (engl. strided convolution), na primjer $S = 2$ (slika 2.4). Autori rada [14] uspoređuju konvolucijsku mrežu koja koristi sažimanje maksimalnom vrijednošću s veličinom sažimanja $k = 2$ i istu konvolucijsku mrežu koja umjesto sažimanja koristi konvoluciju s korakom $S = 2$. Dobiveni rezultati prikazuju da nema velike razlike u performansi modela za zadatak klasifikacije.



Slika 2.4: Prikaz konvolucije s korakom (engl. strided convolution) $S = 2$, veličinom filta 3x3 i popunjavanjem do rubova $P = 1$ nad ulaznom mapom (plava boja), generira izlaznu mapu (zelena boja) [26]

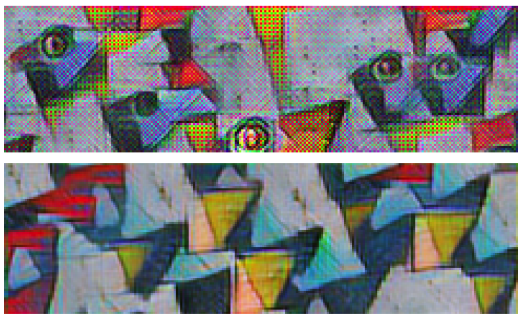
Unatražna konvolucija omogućava da se nauči prostorno povećanje (engl. deconvolution, transposed convolution, upsampling) [13, 1] što radi vrlo dobro za generativne suparničke mreže. Nastala je iz potrebe da se obavlja transformacija koja ide u suprotnom smjeru od obične konvolucije. Na primjer koristi se kod povećanja rezolucije, u našem radu takve konvolucije se nalaze u generatoru jer iz vektora šuma pokušavamo generirati sliku. Unatražna konvolucija se obavlja na sljedeći način, možemo pratiti prema slici (2.5). Vrijednost ulazne mape u crvenoj boji množi se s vrijednostima filtra, nakon toga te vrijednosti se zapišu na mjesta unutar crvenog kvadrata kako je prikazano na slici. Korak unatražne konvolucije je $S = 2$, zato pomak jednog piksela u ulazu odgovara pomaku u izlazu za 2 piksela. Nadalje uzimamo vrijednost ulazne mape u plavoj boji te množimo s istim filtrom te zapišemo kako je prikazano u plavom kvadratu. Na mjestima gdje se izlazi preklapaju jednostavno se sumiraju vrijednosti. Proces obavljamo za sve ulazne vrijednosti te generiramo izlaz dimenzija 4x4. Možemo primjetiti da postupak unatražne konvolucije odgovara unatražnom prolazu obične konvolucije.

Nedavno je izašao članak [19] koji je prikazao da unatražna konvolucija kao pos-



Slika 2.5: Prikaz unatražne konvolucije s korakom $S = 2$, veličinom filta 3x3 te popunjavanjem do rubova $P = 1$ nad ulaznom mapom 2x2 [7]

ljedicu ima pojavljivanje šahovskih artefakata (engl. checkboard artifacts) na slikama tijekom treniranja i nakon obavljenog treniranja. Efekt možemo već vidjeti nakon inicijalizacije težina tijekom prvog prolaza i na nekim primjerima nakon obavljenog treniranje mreže (slika 2.6). Tijekom treniranja generativnih suparničkih modela primijetio sam sličan efekt. Preporuke autora [19] da se umjesto unatražne konvolucije koristi povećanje slike klasičnim metodama interpolacije kao što su interpolacija najbližeg susjeda (engl. nearest-neighbor interpolation) ili bilinearna interpolacija (engl. bilinear interpolation) nakon čega slijedi konvolucija (korak $S = 1$). Najbolje rezultate su postigli koristeći interpolaciju najbližeg susjeda. Navedena izmjena rezultirala je izbacivanjem šahovskih artefakata sa slika.



Slika 2.6: Efekt pojave šahovskih artefakata. Gornja slika koristi unatražnu konvoluciju s korakom (S=2). Pojava šahovskih artefakata, dok donja slika koristi interpolaciju najbližeg susjeda nakon čega slijedi konvolucija (S=1). Nema pojave šahovskih artefakata. [19]

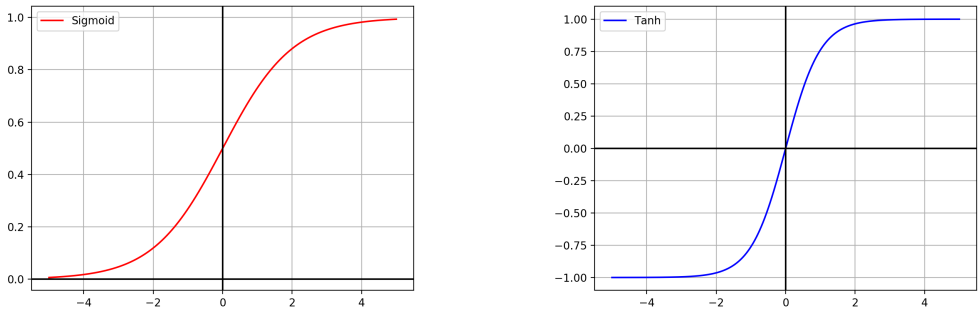
2.3. Aktivacijske funkcije

Aktivacijska funkcija služi kako bi se postigla nelinearnost između slojeva neuronske mreže. Bez nelinearnih aktivacijskih funkcija mreža ne bi mogla naučiti nelinearnost koja postoji među podacima. U dubokim konvolucijskim mrežama često se koriste sljedeće aktivacijske funkcije: sigmoidalna funkcija (2.7), tangens hiperbolni (2.7) te ReLU (2.9) i LeakyReLU (2.10). One se nalaze nakon izlaza konvolucijskog sloja mreže. Grafovi navedenih funkcija nalaze se na slikama (2.5) i (2.6).

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

$$f(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.8}$$

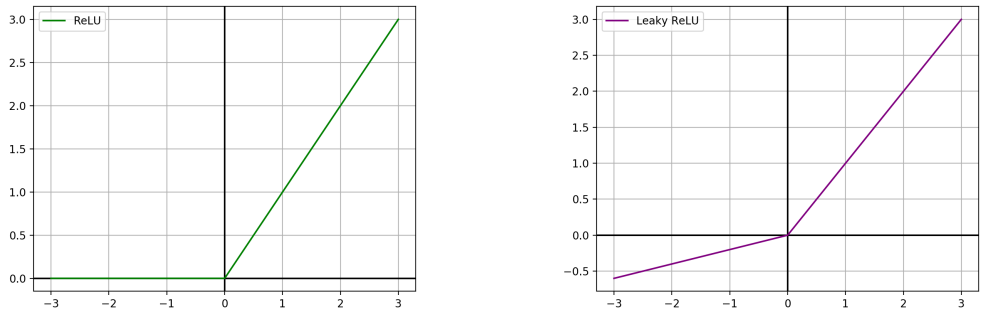
Sigmoidalna funkcija je česta aktivacijska funkcija koja monotono raste te njen izlaz se asimptotski približava nekoj konačnoj vrijednosti (vrijednosti 1, graf (a) 2.5), dok ulaz te aktivacije raste prema $\pm\infty$. Sigmoidalna funkcija ima nekoliko nedostataka: kada uđe u zasićenje, pojavljuje se problem nestajućeg gradijenta (eng. vanishing gradient), izlazi nisu centrirani oko 0 te računanje eksponenta je skupa operacija. Tangens hiperbolni obično se ponaša bolje od sigmoide jer sliči identitetu u dijelu oko $x = 0$, što omogućuje jednostavan transfer gradijenata unatrag. Također izlazi su centrirani oko nule, ali još uvijek dolazi do problema nestajućeg gradijenta u zasićenju (kada su uzlazne vrijednosti jako visoke u pozitivnom ili negativnom smjeru).



(a) Sigmoidalna funkcija

(b) Tangens hiperbolni

Slika 2.7: Grafovi aktivacijskih funkcija



(a) ReLU **(b)** Leaky ReLU

Slika 2.8: Grafovi aktivacijskih funkcija

$$f(x) = \max(0, x) \tag{2.9}$$

$$f(x) = \max(x, ax) \tag{2.10}$$

ReLU aktivacijska funkcija (engl. Rectified Linear Unit) se vrlo često koristi u dubokim mrežama jer modeli puno brže konvergiraju. Vidimo da u pozitivnom dijelu $x > 0$ nema zasićenja što za posljedicu ima smanjen problem nestajućeg gradijenta. Izračun ReLU je vrlo efikasan, ali u prolazu prema naprijed, ako je $x < 0$ neuron postaje neaktivan te gasi gradijent prilikom prolaska unatrag. Da bismo riješili navedeni problem nestajanja gradijenta možemo koristiti LeakyReLU aktivacijsku funkciju. Ona nasljeđuje sve prednosti ReLU aktivacije i ne gasi u potpunosti gradijent. U radu [4] prikazana je usporedba različitih ReLU aktivacijskih funkcija, LeakyReLU s $a = 0.2$ radi vrlo dobro u nekim arhitekturama. Eksperimentalno je utvrđeno da rezultati poboljšanja nisu konzistentni. Stoga preporuka je koristiti ReLU, a LeakyReLU te ostale aktivacije testirati za određeni problem posebno.

2.4. Optimizacijski postupak

Optimizacijski postupak se svodi na traženje takvih parametara modela za koje je odabrana funkcija gubitka minimalna. Metoda učenja širenjem unazad je osnovni optimizacijski postupak učenja dubokih mreža. Fokus ovog dijela će biti na novijim algoritmima koji pomažu ubrzanju konvergencije duboke mreže. Postupak optimizacije se obavlja do zadovoljavanja zadanih kriterija konvergencije ili do isteka nekog postavljenog uvjeta. Optimizacijski postupci se dijele na nekoliko vrsta:

- postupci koji koriste čitav skup primjera za učenje nazivaju se grupni (engl. batch)
- postupci koji koriste jedan po jedan primjer iz skupa primjera nazivaju se stohastički (engl. stohastic)
- postupci koji koriste podskup primjera u koraku, veličina podskupa (engl. batch size) je hiperparametar modela, nazivaju se postupci učenja nad mini-grupama (engl. mini-batch)

Pri optimizaciji modela dubokog učenja najviše se koristi učenje nad mini-grupama jer brže vodi do konvergencije. Kod grupnih algoritama potrebno je puno vremena i memorije za računanje gradijenata cijele grupe svih primjera. Algoritmi učenja nad mini-grupama računaju gradijent na dijelu podataka, što znači da aproksimiraju gradijent cijelog skupa podataka. Tijekom aproksimacije unose dodatni šum, no upravo šum je koristan pri ne-konveksnoj optimizaciji jer može izvući postupak iz lokalnog minimuma ili sedlastih područja u kojima je zapeo. Nadalje spomenuti algoritmi su opisani korištenjem učenja nad mini-grupama.

Gradijentni spust je iterativan način minimizacije funkcije gubitka $J(\theta)$ parametrizirane parametrima modela θ. To čini ažuriranjem parametara θ u suprotnom smjeru gradijenata $J(\theta)$. Formalno:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta J\left(x^{(i)}, y^{(i)}; \theta\right) \tag{2.11}$$

gdje η predstavlja stopu učenja. Stopa učenja određuje veličinu koraka koji koristimo da bismo postigli (lokalni) minimum. Definiranje stope učenja je vrlo bitno jer ima značajan utjecaj na učinak modela. Bira se eksperimentalno na temelju promatranja krivulja učenja koje prikazuju vrijednost funkcije gubitka ovisno o vremenu.

Opisat ćemo dva algoritma koji spadaju u grupu algoritama s adaptivnom stopom učenja.

2.4.1. RMSProp

RMSProp (engl. Root Mean Square Propagation) je algoritam temeljen na gradijentnom spustu koji može ubrzati spust koristeći adaptivnu stopu učenja. Empirijski utvrđeno RMSProp efikasno minimizira funkciju cilja te vrlo dobro radi za duboke modele [9]. Osnovna ideja je da stopa učenja u početku veća te omogućuje brže gradijentno spuštanje, a blizu konvergencije je manja, čime usporava učenje i pomiče se za manje korake. RMSProp kod računanja gradijenata u svakom koraku računa i eksponencijalni pomični prosjek kvadrata gradijenata prijašnjih koraka koji označavamo s r. Nakon čega trenutnu vrijednost gradijenta podijeli s korijenom srednjih kvadrata r. G. Hinton, autor algoritma preporučuje koristiti vrijednosti za parametar smanjenja $p = 0.9$, a stopu učenja $\eta = 0.001$. Algoritam u cijelosti:

Algorithm 1 RMSProp algoritam

Require: stopa_ucenja η, parametar smanjenja p, inicijalni parametar θ

Require: konstanta δ (10^{-6}), koristi se za izbjegavanja dijeljenja s nulom

Inicijaliziraj varijablu akumulacije $r = 0$

while kriterij za zaustavljanje nije zadovoljen **do**

Kreiraj m skup primjera iz podataka za treniranje $\{x^{(i)}, ..., x^{(m)}\}$ zajedno s oznakama $y^{(i)}$

$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$

$r \leftarrow pr + (1 - p) g \odot g$

$\triangle\theta = -\frac{\eta}{\sqrt{\delta + r}} \odot g$($\frac{1}{\sqrt{\delta+r}}$ operacija po elementima)

$\theta \leftarrow \theta + \triangle\theta$

end while

2.4.2. ADAM

ADAM (engl. Adaptive Moments) je algoritam koji izračunava adaptivnu stopu učenja za različite parametre iz procjena prvog i drugog momenta gradijenata. Kombinira metode momenta i adaptivnog pomaka što pospješuje učenje dubokih modela. Nastao je kombinacijom prednosti algoritama AdaGrad koji dobro funkcionira s rijetkim gradijentima [6] i prije navedenog RMSProp algoritma. Rijetki gradijenti nastaju zbog korištenja ReLU aktivacije, negativni ulazi u aktivaciju se postavljaju na 0 što znači da ulazi u sljedeći sloj imaju veliki broj ulaza vrijednosti 0 [24]. Algoritam ADAM izračunava eksponencijalni pomični prosjek gradijenta i kvadratnog gradijenta, a parametri β_1 i β_2 kontroliraju stope smanjenja tih pomičnih prosjeka. Neke od prednosti algoritma ADAM su da prirodno izvodi korekciju koraka te magnitude ažuriranih gradijenata su invarijantne na skaliranje gradijenta [6].

Algorithm 2 Algoritam ADAM

Require: stopa_ucenja η, β_1 i β_2, δ, inicijalni parametar θ

Inicijalizacija prvog i drugog momenta $s = 0, r = 0$ i vremenskog koraka $t = 0$

while kriterij za zaustavljanje nije zadovoljen **do**

Kreiraj m skup primjera iz podataka za treniranje $\{x^{(i)}, ..., x^{(m)}\}$ zajedno s oznakama $y^{(i)}$

$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$

$t \leftarrow t + 1$

$s \leftarrow \beta_1 s + (1 - \beta_1) g$

$r \leftarrow \beta_2 r + (1 - \beta_2) g \odot g$

$\hat{s} \leftarrow \frac{s}{1-\beta_1^t}$

$\hat{r} \leftarrow \frac{r}{1-\beta_2^t}$

$\triangle\theta = -\eta \frac{\hat{s}}{\sqrt{\hat{r}}+\delta} \odot g(\frac{1}{\sqrt{\delta}+r}$ operacija po elementima)

$\theta \leftarrow \theta + \triangle\theta$

end while

Algoritam Adam koristi nekoliko hiperparametara: β_1 i β_2 su eksponencijalne stope smanjenja za procjene momenta, iz raspona [0, 1), t je vremenski korak koji se povećava s korekcijom parametara te stopa učenja η. Konstanta δ služi za izbjegavanje dijeljenja s nulom i postavlja se na vrlo malu vrijednost $(10^{-6}, 10^{-8})$. Preporučene vrijednosti su $\eta = 0.001$, $\beta_1 = 0.9$ i $\beta_2 = 0.999$ [6]. Adam se općenito smatra prilično robustan za izbor hiperparametara, iako se ponekad moraju mijenjati predložene zadane vrijednosti.

2.5. Normalizacija po grupama

Normalizacija po grupama (engl. batch normalization) je vrlo zanimljiva inovacija u optimizaciji dubokih neuronskih mreža. Metoda nije optimizacijski algoritam nego adaptivna reparametrizacija podataka tijekom procesa treniranja duboke mreže. Praksa pokazuje da mreže lakše uče nad podatcima koji su normalizirani, a to znači preslikani u podatke čija je srednja vrijednost 0 i varijanca jedinična [9]. Gradijent govori kako ažurirati svaki parametar, pod pretpostavkom da se drugi slojevi ne mijenjaju. Slojeve ažuriramo istovremeno i to nam stvara problem jer izlaz jednog sloja je ulaz u sljedeći. Normalizacija po grupama pruža elegantan način ublažavanja navedenog problema. Posljedica je da grupe tijekom treniranja imaju sličnu distribuciju te se postiže bolje propagiranje gradijenata time i brže učenje mreže. Algoritam normalizacije po grupama:

Ulaz: Mini-grupa primjera $\{x^{(i)}, ..., x^{(m)}\}$ veličine m; Parametri koji se uče γ, β

Izlaz: Normalizirana mini-grupa $\{y^{(i)}, ..., y^{(m)}\}$ parametrima γ, β

$$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{(srednja vrijednost primjera)}$$

$$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2 \qquad \text{(varijanca mini-grupe)}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \eta}} \qquad \text{(normalizacija grupe)}$$

$$y_i \leftarrow \gamma\hat{x}_i + \beta \qquad \text{(skaliranje i pomak distribucije)}$$

Normalizacija svakog ulaza sloja može promijeniti neke bitne informacije o značajkama. Stoga zadnji korak normalizacije grupe je transformacija koja omogućava da se originalne vrijednosti mogu vratiti iz normaliziranih ulaznih podataka. Parametri te transformacije γ, β se uče kao i ostali parametri modela. Normalizacija po grupama osim pomoći u treniranju, vrši regularizaciju dubokog modela te omogućuje korištenje veće stope učenja [22]. Regularizacija sprječava prenaučenost modela te tako model bolje generalizira za nepoznate primjere. U konvolucijskim slojevima je bitno primijetiti da se srednja vrijednost i varijanca računaju posebno za svaku izlaznu mapu značajki [22].

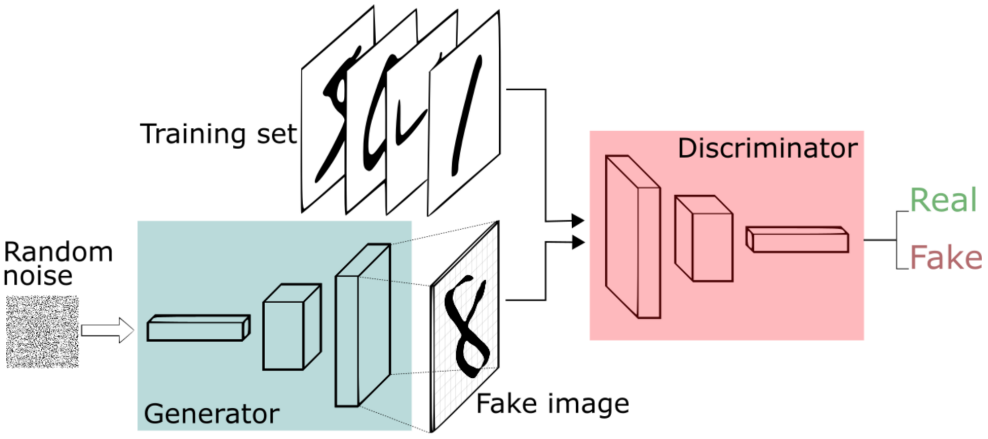
3. Generativni suparnički modeli

U ovom poglavlju ćemo opisati granu generativnih modela koja uzorkuje primjere direktno iz distribucije podataka predstavljene u modelu. Generativne suparničke mreže su dizajnirane kako bi se izbjegli nedostatci ostalih generativnih modela. Mogu paralelno generirati primjere, proces generiranja ne ovisi o primjeru prije, nije potrebno koristiti Markovljeve lance te definicija generatorske funkcije ima vrlo malo restrikcija. No, ove prednosti uvode novi nedostatak generativnih suparničkih mreža. Treniranje modela zahtijeva pronalaženje Nashove ravnoteže igre, a to je teži zadatak od optimizacije funkcije cilja. Osim problema treniranja, vrlo je teško evaluirati distribuciju generiranih primjera [8]. U narednim potpoglavljima opisat ćemo različite tipove modela.

3.1. Jednostavni generativni suparnički model

Generativni suparnički model možemo zamisliti kao igru između dva igrača. Svaki igrač je predstavljen s diferencijabilnom funkcijom koja je kontrolirana skupom parametara. U većini slučajeva te funkcije su implementirane kao duboke neuronske mreže. Jedan od igrača je model za generiranje podataka ili generator. Generator pokušava generirati primjere što bliže distribuciji podataka za treniranje. Drugi igrač je model za razlikovanje stvarnih od generiranih podataka ili diskriminator. Diskriminator se zapravo uči na klasični nadzirani način jer je odluka o primjeru izbor između dvije definirane klase. Cilj generatora je nadmudriti diskriminatora tako da generira primjere koji izgledaju kao primjeri iz distribucije podataka za treniranje, dok diskriminator želi što točnije klasificirati dane primjere.

Generator se može interpretirati kao jednostavna diferencijabilna funkcija G koja na ulazu prima vektor šuma z. Vektor šuma z je uzorkovan iz uniformne distribucije p_z. Šum se provodi kroz generator koji se definira parametrima θ_g te se na izlazu dobiva generirani primjer $\tilde{x}$. Pojednostavljeno zadatak generatora je približiti jednostavnu distribuciju p_z distribuciji stvarnih podataka p_{data}.



Slika 3.1: Izgled generativnog suparničkog modela. Slučajno generiran vektor šuma uniformne distribucije (engl. Random noise) provodi se kroz generator te generira lažnu sliku (engl. Fake image). Diskriminator na ulaz prima primjere iz podatkovnog skupa (engl. Training set) ili generiranu lažnu slike te donosi odluku o primjeru je li generiran ili stvaran. [25]

Diskriminator predstavlja funkciju D koja na ulazu prima primjer x te na izlazu određuje binarnu klasifikaciju pripada li stvarnoj klasi (1) ili generiranoj (0). Model je definirana skupom parametara θ_d. Kod definiranja generativnog suparničkog modela funkcija gubitka diskriminatora je uvijek ista, dok se funkcija gubitka generatora može mijenjati [8]. U najjednostavnijoj verziji modela diskriminator i generator igraju igru sa sumom nula (engl. zero-sum game) u kojoj je zbroj vrijednosti funkcija gubitka igrača uvijek nula. Diskriminator želi minimizirati binarnu unakrsnu entropiju:

$$J^{(D)}(\theta_g, \theta_d) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} logD(x) - \frac{1}{2}\mathbb{E}_z log(1 - D(G(z))) \tag{3.1}$$

mijenjanjem samo svojih parametara modela θ_d. Generator želi minimizirati:

$$J^{(G)}(\theta_g, \theta_d) = -J^{(D)} \tag{3.2}$$

mijenjanjem parametara modela θ_g. Iz razloga što funkcije gubitka svakog igrača ovise o parametrima drugog igrača, a igrači nisu u mogućnosti mijenjati parametre drugog igrača rješenje nije optimizacija nego igra. Učenje se stoga svodi na pronalaženje Nashove ravnoteže igre. Nashova ravnoteža igre je skup strategija, jedna za svakog igrača, takva da igrači nemaju poticaj da promjene svoju strategiju u odnosu na drugog igrača. Pronalaženje Nashove ravnoteže ove igre je teži zadatak od klasične optimizacije funkcije cilja. Nashova ravnoteža igre u ovom slučaju je par (θ_g, θ_d) takav da je za parametre modela θ_d funkcija $J^{(D)}$ u minimumu, a za parametre θ_g funkcija

$J^{(G)}$ u minimumu [8]. Upravo zato generativni suparnički modeli su vrlo nestabilni za treniranje. Proces treniranja obavlja se tako da se naizmjenično treniraju diskriminator i generator. Formalno to možemo zapisati:

$$V^*(G,D) = \min_{\theta_g}\max_{\theta_d}[\mathbb{E}_{x\sim p_{data(x)}} logD_{\theta_d}(x) + \mathbb{E}_{z\sim p_z} log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \quad (3.3)$$

Kroz praksu se pokazalo da u generatoru minimiziranje izglednosti diskriminatora da bude točan daje slabije gradijente. Umjesto toga želi se maksimizirati izglednost diskriminatora da bude u krivu [12]. Ideja je da maksimiziramo $logD(G(z))$ umjesto minimiziranja $log(1 - D(G(z))$, što odgovara istom cilju, nadmudrivanja diskriminatora. Algoritam treniranja generativne suparničke mreže na kraju izgleda ovako:

Algorithm 3 Treniranje generativne suparničke mreže stohastičkim gradijentnim spustom. Broj k koraka je hiperparametar modela, ali zbog vremenski dugog procesa, u praksi koristi se k=1.

1: **for** `broj_iteracija_treniranja` **do**
2: **for** `k` **do**
3: Kreiraj m skup primjera vektora šuma $\{z^{(i)}, ..., z^{(m)}\}$ iz distribucije p_z
4: Kreiraj m skup primjera $\{x^{(i)}, ..., x^{(m)}\}$ iz distribucije podataka p_{data}
5: Osvježi parametre diskriminatora pomicanjem njihovog stohastičkog gradijenta uzlazno:

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

6: **end for**
7: Kreiraj m skup vektora šuma $\{z^{(i)}, ..., z^{(m)}\}$ iz distribucije p_z
8: Osvježi parametre generatora pomicanjem njihovog stohastičkog gradijenta uzlazno:

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}[logD(G(z^{(i)}))]$$

9: **end for**

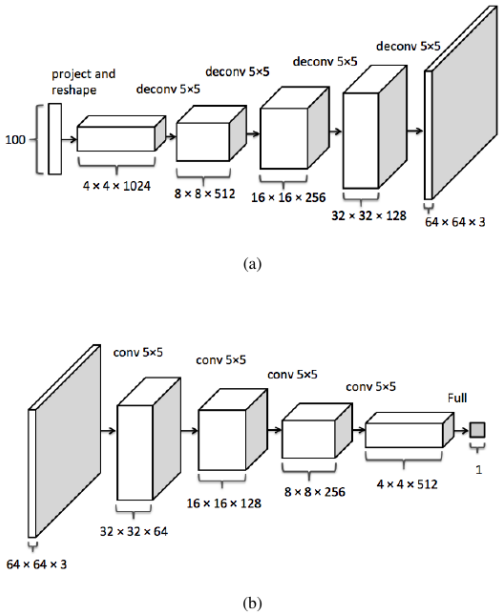
Možemo primijetiti da ciljna funkcija generatora ne može vidjeti podatke za treniranje direktno. Sve informacije o podacima za treniranje dobiva iz toga što je diskriminator naučio na primjerima [8]. Zbog navedenog, generativne suparničke mreže su otpornije na pretreniranje od različitih tipova autoenkodera. Pretreniranje modela je slučaj kada model ne generalizira dobro na novim neviđenim primjerima.

3.2. Duboki konvolucijski generativni suparnički model

U stvarnosti distribucije podataka, u našem slučaju slike, imaju kompliciranu i višemodalnu distribuciju. Raspodjela vjerojatnosti koja opisuje višemodalne podatke može imati više "vrhova" gdje su koncentrirane različite podgrupe uzoraka. Jednostavni generativni suparnički modeli zbog nestabilnog treniranja mogu završiti u fazi degradacije treniranja (engl. mode collapse). Navedena pojava označava da generator za proizvoljan vektor šuma generira isti uzorak, odnosno naučio je samo jedan mod podataka te nije u mogućnosti modelirati sve modove podataka.

Autori konvolucijskog generativnog suparničkog modela pronašli su konfiguraciju i arhitekturu modela u kojoj znatno povećavaju stabilnost treniranja [1]. U svom radu iznijeli su tri bitne modifikacije.

Prva modifikacija je korištenje normalizacije po grupama s kojoj se stabilizira proces treniranja [22]. Normaliziraju se ulazi tako da primjeri imaju srednju vrijednost 0 te jediničnu varijancu. Posljedica je bolja propagacija gradijenata za dublje arhitekture te se postiže ubrzanje učenja. Normalizaciju po grupama ne bi trebalo koristiti na izlaznom sloju generatora i na ulaznom sloju diskriminatora jer to uvodi nestabilnosti u model [1].



Slika 3.2: Slika a) prikazuje arhitekturu konvolucijskog generatora. Slika b) prikazuje arhitekturu konvolucijskog diskriminatora [21]

Nadalje trebalo bi izbaciti potpuno povezane slojeve. Zadnja modifikacija je zamjena slojeva za sažimanje (engl. pooling layer) kao što su sažimanje maksimalnom vrijednošću ili sažimanje srednjom vrijednosti s konvolucijom s korakom (engl. strided convolution). Unatražna konvolucija (engl. transponse convolution) služi kako bi se moglo naučiti prostorno povećanje. Uporabom unatražne konvolucije iz šuma želimo generirati sliku dok u diskriminatoru koristimo konvoluciju s korakom za prostorno sažimanje. Arhitetkura dubokog konvolucijskog generativnog suparničkog modela na slici (3.2) koristi se u radu [1]. Na temelju navedene arhitekture napravili smo implementaciju modela koji smo koristili u radu, arhitektura je detaljnije opisana u četvrom poglavlju.

Smjernice za stabilnu arhitekturu dubokog konvolucijskog modela

- koristiti grupnu normalizaciju u generatoru i diskriminatoru (osim na izlaznom sloju generatora i ulaznom sloju diskriminatora)
- izbaciti poptuno povezane skrivene slojeve za dublje arhitekture
- zamjeniti slojeve sažimanja (engl. pooling layer) s konvolucijama s korakom
- koristiti ReLU aktivaciju u generatoru za sve slojeve, osim zadnjeg koji koristi Tanh aktivaciju
- koristiti LeakyReLU aktivaciju u svim slojevima diskriminatora
- koristiti ADAM optimizator umjesto stohastičkog gradijentnog spusta s momentom

3.3. Wassersteinov generativni suparnički model

Problem s kojim smo se susreli tijekom treniranja jednostavnog i konvolucijskog generativnog suparničkog modela je nemogućnost interpretacije grafa funkcije gubitka. Kvaliteta slike koju generiramo kroz generatorsku mrežu nema korelacije s funkcijom gubitka. Proces treniranja jedino smo mogli nadgledati promatrajući kvalitetu generirane slike kroz vrijeme. Zbog navedenog problema predložen je novi model Wassersteinov generativni suparnički model koji za cilj ima dodatno poboljšanje stabilnosti treniranja te pružanje bolje interpretacije grafa učenja. Najveća promjena u modelu se odnosi na odabir drugačije mjere izračuna udaljenosti distribucije modela i distribucije podataka. Kako bi bolje objasnili predloženi model opisat ćemo nekoliko metrika koje se koriste u familiji generativnih modela.

Kullback-Leibler divergencija mjeri koliko jedna distribucija vjerojatnosti p odstupa od druge očekivane distribucije vjerojatnosti q. Formalno to možemo zapisati:

$$D_{KL}(p||q) = \int_x p(x) log \frac{p(x)}{q(x)} dx \tag{3.4}$$

D_{KL} postiže minimum nula kada je $p(x) == q(x)$ posvuda. Možemo primijetiti da je formula KL divergencije asimetrična i moguće beskonačna u slučaju kada za neke točke vrijedi $q(x) = 0$ i $p(x) > 0$. Generativni modeli bazirani na maksimizaciji log izglednosti koriste ovu mjeru udaljenosti između distribucije podataka i modela. Minimizacija D_{KL} između $\hat{p}_{data}$ i p_{model} je ekvivalentna maksimizaciji log izglednosti podataka za treniranje [8].

Jensen-Shannon divergencija je mjera sličnosti između dviju distribucija, omeđena u rasponu $[0, 1]$. Uvijek je definirana i simetrična. Funkcija gubitka generativnog suparničkog modela koristi Jensen-Shannon divergenciju kao mjeru između distribucije podataka i modela kada je diskriminator optimalan. [17].

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2}) \tag{3.5}$$

Autori predloženog novog modela uvode Wassersteinovu udaljenost. Naziva se još i udaljenost pomicanja zemlje (engl. Earth Mover's distance) jer se može interpretirati kao pomicanje nakupine zemlje koja ima neku distribuciju p za minimalnu cijenu da prati drugu distribuciju q. Wassersteinova udaljenost je cijena optimalnog plana transporta zemlje. Formalno:

$$W(p,q) = \inf_{\gamma \in \prod((p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||] \tag{3.6}$$

gdje $\prod(p,q)$ označava skup svih mogućih distribucija između distribucija p i q. Jedna zajednička distribucija $\gamma \in \prod(p,q)$ opisuje jedan plan transporta zemlje. Ako uzmemo da je x početna točka, a y odredišna točka transporta, ukupna količina premještene zemlje je $\gamma(x,y)$. Ukupna udaljenost je $||x-y||$, tada cijena iznosi $\gamma(x,y)\cdot||x-y||$. Očekivana cijena kroz sve parove (x,y) se izračunava:

$$\mathbb{E}_{x,y\sim\gamma}\,||x-y|| = \sum_{x,y}\gamma(x,y)\cdot||x-y|| \tag{3.7}$$

Na kraju od svih cijena uzimamo minimalnu kao Wassersteinovu udaljenost [18]. Formula (3.5) zahtijeva iscrpno pretraživanje svih mogućih zajedničkih distribucija u $\prod(p,q)$ kako bi izračunali $\inf_{\gamma\in\prod((p,q)}$, što je nemoguće. Autori su predložili transformaciju formule na oblik koji se bazira na Kantorovich-Rubstein dualnosti:

$$W(p,q) = \frac{1}{K}\sup_{||f||_L\leq K}\mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \tag{3.8}$$

gdje sup (supremum) označava suprotno od inf (infinum), drugim riječima želimo maksimalnu vrijednost. Funkcija f u novom obliku treba zadovoljiti $||f||_L \leq K$ čime postaje K-Lipschitz kontinuirana. Realna funkcija f naziva se K-Lipschitz kontinuirana ako postoji realna konstanta $K \in R, K \geq 0$ takva da za $\forall x_1, x_2 \in R$ vrijedi:

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \tag{3.9}$$

U Wassersteinovom modelu diskriminator se koristi za učenje težina w kako bi se pronašla funckija f_w koja se koristi za izračun Wassersteinove udaljenosti između distribucije modela p_{model} i podataka p_{data}. Funkcija f bi trebala biti iz familije K-Lipschitz kontinuiranih funkcija. Formalno:

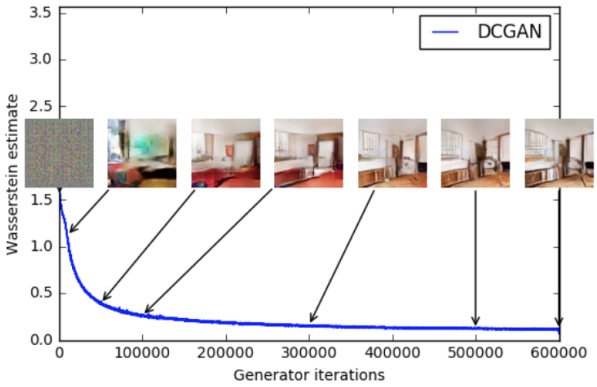
$$W(p_{data}, p_{model}) = \max_{w \in \mathcal{W}} \mathbb{E}_{x \in p_{data}}[f_w(x)] - \mathbb{E}_{x \in p_{model}}[f_w(G_{\theta_g}(z)] \tag{3.10}$$

U odnosu na klasični generativni suparnički model, diskriminator više nema za cilj klasifikaciju primjera kao generiranog ili stvarnog, nego kroz proces treniranja pokušava naučiti funkciju f_w iz jednadžbe (3.8) tako da bude K-Lipschitz kontinuirana funkcija kako bi se mogla izračunati Wassersteinova udaljenost distribucija. Kako se smanjuje funkcija gubitka kroz proces treniranja, Wassersteinova udaljenost se smanjuje te kao posljedicu generatorska mreža generira slike bliže distribuciji podataka [11]. Tijekom procesa treniranja problem je zadržavanja K-Lipschitz kontinuiranosti funkcije f_w. Autori rada da bi zadržali težine w u kompaktnom prostoru $\mathcal{W}$ te time održali K-Lipschitz kontinuiranost nakon svakog osvježavanja gradijenata ograničavaju težine w u rasponu vrijednosti ($\mathcal{W}$ = $[-0.01, 0.01]$). Ograničavanje težina (engl. weight clipping) nije dobar način održavanja K-Lipschitz kontinuiranosti funkcije [18]. Kasnije ćemo vidjeti elegantniji način. Algoritam Wassersteinovog generativnog modela glasi:

Algorithm 4 WGAN. Svi eksperimenti koriste zadane vrijednosti stopa_učenja=0.00005, c=0.01, m=64, n_{critic} = 5

Require: $stopa\_ucenja$, c-parametar ograničavanja težina, m-veličina grupe, n_{critic}-broj iteracija diskriminatorske mreže, θ_g-parametri generatorske mreže, w-parametri diskriminatorske mreže.

while θ_g nije konvergirala **do**
  for $0, ..., n_{critic}$ **do**
    Kreiraj m skup primjera vektora šuma $\{z^{(i)}, ..., z^{(m)}\}$ iz distribucije p_z
    Kreiraj m skup primjera $\{x^{(i)}, ..., x^{(m)}\}$ iz distribucije podataka p_{data}
    $g_w \leftarrow \nabla_w[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(G_{\theta_g}(z^{(i)}))]$
    $w \leftarrow w + stopa\_ucenja \cdot RMSProp(w, g_w)$
    $w \leftarrow clip(w, -c, c)$
  end for
  Kreiraj m skup vektora šuma $\{z^{(i)}, ..., z^{(m)}\}$ iz distribucije p_z
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(G_{\theta_g}(z^{(i)}))$
  $\theta_g \leftarrow \theta_g - stopa\_ucenja \cdot RMSProp(\theta_g, g_\theta)$
end while



Slika 3.3: Prikaz korelacije kvalitete generirane slike i pada funkcije gubitka [18].

Činjenica da je Wassersteinova udaljenost kontinuirana i diferencijabilna gotovo posvuda [11], omogućava treniranje diskriminatorske mreže do optimalnosti. Pojava degradacije treniranja je tada uvelike umanjena jer treniranjem diskriminatora više, dobivamo kvalitetnije gradijente za treniranje generatorske mreže. U slučaju Jensen-Shannon mjere odnosno u prijašnjim generativnim suparničkim modelima, kako diskriminator postaje bolji, imamo kvalitetnije gradijente, ali Jensen-Shannon mjera nije svugdje kontinuirana i diferencijabilna te zbog toga nekad ne daje iskoristiv gradijent, zato se pojavljuju nestajući gradijenti što može dovesti do degradacije treniranja [18]. Proces treniranja možemo nadgledati iscrtavanjem Wassersteinove udaljenosti ovisno o iteracijama generatorske mreže. Na slici (3.3) vidimo da nam takav graf daje uvid u korelaciju kvalitete slike koju generatorska mreže generira i Wassersteinovu udaljenosti distribucija.

Glavni benefiti Wassersteinovog generativnog suparničkog modela

- mogućnost treniranja diskriminatorske mreže do optimalnosti time sprečavajući degradaciju treniranja (engl. mode collapse)
- interpretacija grafa funkcije gubitka koja smanjenjem, zapravo smanjuje Wassersteinovu udaljenost između distribucije generiranih podataka i stvarnih

Navedena verzija Wassersteinovog modela je korak u dobrom smjeru, ali još uvijek ima problema tijekom treniranja. Odabirom prevelikog parametra ograničavanja težina konvergencija je puno sporija, a premalog dovodi do nestajućih gradijenata [11]. Također empirijski je utvrđeno da model je stabilniji ako umjesto ADAM optimizatora

koristi RMSProp optimizator [18]. Najnoviji rad [11] predlaže alternativnu metodu za održavanje K-Lipsschitz kontinuiranosti funkcije, penaliziranje gradijenata. Ideja je da direktno ograničavamo norme gradijenata izlaza diskriminatorske mreže u odnosu na ulaz. Nova funkcija cilja je sljedeća:

$$L = \underbrace{\mathbb{E}_{\hat{x}\sim p_{model}}[D(\hat{x})] - \mathbb{E}_{x\sim p_{data}}[D(x)]}_{Originalna\ funkcija\ gubitka} + \underbrace{\lambda\,\mathbb{E}_{\hat{x}\sim p_{\hat{x}}}[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2]}_{Penaliziranje\ gradijenata} \tag{3.11}$$

gdje vrijednost λ = 10 radi vrlo dobro u različitim arhitekturama [11]. Zbog navedene funkcije cilja, normalizaciju po grupama treba izbaciti iz diskriminatorske mreže jer penaliziramo gradijente u odnosu na svaki ulaz pojedinačno, a ne u odnosu na grupu. Model postiže najbolje rezultate koristeći ADAM optimizator umjesto RMSProp optimizatora koji je korišten u prijašnjoj verziji. Algoritam stabilnije verzije Wassersteinovog modela izgleda ovako:

Algorithm 5 WGAN s penaliziranjem gradijenta Svi eksperimenti koriste zadane vrijednosti stopa_učenja=0.0001, λ=10, m=64, n_{critic} = 5,β_1 = 0, β_2 = 0.9.

Require: λ-koeficijent penaliziranja gradijenta, m-veličina grupe, n_{critic}-broj iteracija diskriminatorske mreže, $stopa\_ucenja$,β_1, β_2-hiperparametri ADAM optimizatora, w-parametri diskriminatorske mreže, θ_g-parametri generatorske mreže.

while θ_g nije konvergirala **do**
  for $0, ..., n_{critic}$ **do**
    for $0, ..., m$ **do**
      Generiraj vektor šuma z iz distribucije p_z, slučajan broj $\eta \sim U[0, 1]$,
      preuzmi primjer x iz distribucije podataka p_{data}
      $\hat{x} \leftarrow G_{\theta_g}(z)$
      $\hat{x} \leftarrow \eta x + (1 - \eta)\hat{x}$
      $L^{(i)} \leftarrow D_w(\hat{x}) - D_w(x) + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2$
    end for
    $w \leftarrow ADAM(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, stopa\_ucenja, \beta_1, \beta_2)$
  end for
  Kreiraj m skup vektora šuma $\{z^{(i)}, ..., z^{(m)}\}$ iz distribucije p_z
  $\theta_g \leftarrow ADAM(\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} -D_w(G_{\theta_g}(z)), \theta_g, stopa\_ucenja, \beta_1, \beta_2)$
end while

4. Implementacija

U okviru rada implementirani su redom generativni suparnički modeli kako su se pojavljivali stabilniji modeli i njihove konfiguracije. Za cijelu programsku izvedbu korišten je programski jezik Python 3.6 te nekoliko razvojnih okvira Pytorch, Tensorflow i Numpy. Pytorch[20] je razvojni okvir otvorenog koda koji omogućava izradu modela strojnog i dubokog učenja. Tim iza Pytorch okvira napravio je alat za korištenje u istraživačkom radu koji omogućava računanje višedimenzionalnih polja (engl. tensor) s jakim ubrzanjem grafičkih kartica te izradu dubokih neuronskih mreža s automatskim sustavom za računanje gradijenata. Tensorflow[2] je također razvojni okvir otvorenog koda, sličan Pytorch-u s razlikom da se u prvom koraku generira računski graf, a nakon toga obavljaju izračuni računskog grafa dok u Pytorch okviru se računski grafovi stvaraju dinamički. U sklopu ovog rada Tensorflow je korišten samo za vizualizaciju rezultata na Tensorboard-u. Numpy okvir se koristi kao podrška za efikasan rad s višedimenzionalnim poljima.

Eksperimentalni rezultati provedeni su na serverskom računalu s operacijskim sustavom Linux Ubuntu koji ima dvije Nvida grafičke kartice GeForce GTX 1070 (8 GB) i GeForce GTX 970 (4GB). U nastavku poglavlja opisani su detalji programske izvedbe i korištene arhitekture modela.

4.1. Arhitektura mreže

4.1.1. Linearni generativni suparnički model

Arhitektura linearnog generativnog suparničkog modela sastoji se od potpuno povezanih slojeva. Ulaz generatora je vektor šuma dimenzionalnosti $z = 100$, uzorkovan iz normalne distribucije $\mathcal{N}(0, 1)$. Generator ima 3 potpuno povezana sloja od kojih prvi sadrži 256 neurona, drugi 512 i treći 1024. Izlazi potpuno povezanih slojeva koriste LeakyReLU ($a = 0.2$) aktivacijsku funkciju, dok izlazni sloj koristi tangens hiperbolni kao aktivaciju. Izlaz generatora je vektor dimenzije 1024 (odgovara slici 1x32x32).

Diskriminator također sadrži 3 potpuno povezana sloja koja sadrže redom 1024, 512, i 256 neurona. Izlazi potpuno povezanih slojeva koriste LeakyReLU (a = 0.2), a sada na izlaznom sloju je sigmoida kao aktivacija jer je potrebno dobiti izlaz koji odgovara klasifikaciji slike kao stvarne ili generirane. Obje mreže koriste ADAM algoritam za optimizaciju sa stopom učenja 0.0002 te s težinskim smanjenjem 0.00001. Tijekom treniranja koristili smo veličinu mini-grupa 64. Pretprocesiranje slika smo obavili skaliranjem na dimenzije 32x32, nakon čega smo raspon vrijednosti $[0, 255]$ prebacili u raspon $[0, 1]$ te na kraju smo obavili normalizaciju te sveli vrijednosti u raspon $[-1, 1]$.

4.1.2. Konvolucijski generativni suparnički model

Konvolucijski model korišten u radu je napravljen po uzoru na model iz rada [1], s razlikom da je arhitektura prikazana za ulaz slike dimenzija $Cx32x32$, gdje je C broj kanala slike. Izlaz svakog konvolucijskog sloja prolazi kroz ReLU aktivaciju za model generatora, a LeakyReLU (a = 0.2) aktivacija za model diskriminatora. Osnova arhitektura modela prikazana je tablicom (5.1).

Tablica 4.1: Arhitektura konvolucijskog generativnog suparničkog modela.

	Generator	**Diskriminator**
Ulaz	Vektor šuma z dimenzije 100 uzorkovan iz $\mathcal{N}(0, 1)$	Slika dimenzije Cx32x32
1.Blok	Konvolucija s korakom S = 2, 1024 filtra veličine 2x2	Konvolucija s korakom S = 2, 256 filtra veličine 2x2
Izlaz	1024x4x4	256x16x16
2.Blok	Konvolucija s korakom S = 2, 512 filtra veličine 2x2	Konvolucija s korakom S = 2, 512 filtra veličine 2x2
Izlaz	512x8x8	512x8x8
3.Blok	Konvolucija s korakom S = 2, 256 filtra veličine 2x2	Konvolucija s korakom S = 2, 1024 filtra veličine 2x2
Izlaz	256x16x16	1024x4x4
4.Blok	Konvolucija s korakom S = 2, C filtra veličine 2x2	Konvolucija s korakom S = 2, 1 filtar veličine 2x2
Izlaz modela	Generirana slika Cx32x32	Klasifikacija 0 ili 1

Navedenu osnovnu arhitekturu smo koristili za rad s tri različita modela: konvolucijski duboki generativni suparnički model (engl. DCGAN), Wassersteinov GAN koji koristi ograničavanje težina i Wassersteinov GAN koji koristi penaliziranje gradijenata. Nadalje ćemo navesti izmjene koje svaki model koristi za stabilno treniranje:

- **DCGAN**

  Model koristi normalizaciju po grupama u diskriminatoru i generatoru, osim na izlaznom sloju generatora i ulaznom sloju diskriminatora. Izlazni sloj generatora ima aktivaciju tangens hiperbolni, dok diskriminator ima sigmoidu kao aktivaciju na izlaznom sloju. Algoritam ADAM se koristi za optimizacijski postupak s izmijenjenim parametrima, stopa učenja $lr = 0.0002$, $\beta_1 = 0.5, \beta_2 = 0.999$ [1].

- **Wassersteinov GAN - ograničavanje težina**

  Model također koristi normalizaciju po grupama kao i DCGAN. Izlazni sloj generatora ima aktivaciju tangens hiperbolni, ali sada diskriminator na izlaznom sloju nema sigmoidu jer izlaz nije više vjerojatnost klase. Prisjetimo se da kod Wassersteinovog modela diskriminator ne uči klasifikaciju nego pokušava naučiti K-Lipschitz kontinuiranu funkciju kako bi se mogla izračunati Wassersteinova udaljenost distribucija. Za optimizacijski postupak koristi se RMSProp algoritam s parametrima, stopa učenja $lr = 0.00005$ jer algoritmi koji koriste moment utječu na stabilnost treniranja [18]. Parametar ograničavanja težina smo postavili na $c = 0.01$, što ograničava težine u rasponu vrijednosti $[-0.01, 0.01]$. Diskriminator se trenira više od generatora te smo broj iteracija diskriminatora postavili na $n_{critic} = 5$.

- **Wassersteinov GAN - penaliziranje gradijenata**

  Model koji koristi penaliziranje gradijenata koristi slične izmjene kao Wasserstein GAN koji ograničava težine diskriminatora s nekoliko razlika. Zbog metode penaliziranje gradijenata, više nije preporuka koristiti normalizaciju po grupama u diskriminatoru jer penaliziramo gradijente u odnosu na svaki primjer pojedinačno, a ne u odnosnu na grupu. No možemo koristiti normalizaciju za svaki primjer pojedinačno, u razvojnom okviru Pytorch metoda se naziva $nn.InstanceNorm2d()$. Parametar penaliziranja gradijenata se postavlja na $\lambda = 10$. Optimizacijski postupak se obavlja algoritmom ADAM sa sljedećim parametrima, stopa učenja $lr = 0.0001$, $\beta_1 = 0.5, \beta_2 = 0.999$ [11].

5. Rezultati

Implementirani modeli su evaluirani nad nekoliko podatkovnih skupova koje ćemo detaljnije navesti u sljedećim potpoglavljima. Evaluacija generativnih modela je težak i još uvijek otvoren problem. Različiti generativni suparnički modeli imaju različite ciljne funkcije što otežava direktnu usporedbu modela. Jedna intuitivna mjera modela može se dobiti tako da postoji skup ljudi koji ocjenjuje vizualnu kvalitetu primjera. No to je vrlo skupa metoda te nije potpuno objektivna pri ocjeni kvalitete primjera. Nadalje navest ćemo nekoliko metoda evaluacije koje smo koristili i dobivene rezultate.

5.1. Podatkovni skupovi

5.1.1. MNIST

Podatkovni skup MNIST [28] često je korišten u evaluaciji modela dubokog učenja zbog svoje jednostavnosti. Sastoji se od 70000 primjera rukom pisanih znamenki $0-9$, 60000 primjera se koristi za učenje te 10000 za testiranje. Slike su dimenzija 28x28 s vrijednostima u rasponu $[0,1]$. Slike smo skalirali na dimenziju 32x32 prije korištenja u implementiranim modelima. Slika (6.1) prikazuje neke od primjere iz skupa.

Slika 5.1: Slučajno odabrana 64 primjera iz podatkovnog skupa MNIST

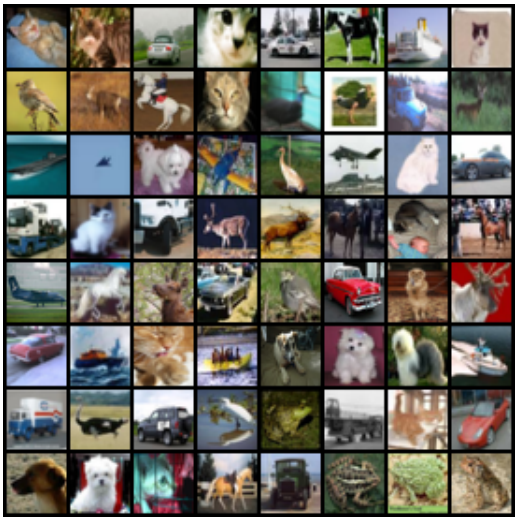
5.1.2. Fashion-MNIST

Podatkovni skup MNIST se u zadnjih nekoliko godina koristio u mnogim evaluacijama modela, ali zbog svoje jednostavnosti vrlo je lako isprobavati različite ideje i prototipove modela. Rezultati klasifikacija nad tim skupom dostizali su točnosti do 99.7%. Upravo zato kreiran je zahtjevniji skup Fashion-MNSIT [27] istog formata. Sastoji od 70000 primjera, sadrži 10 klasa odjevnih predmeta. Svaka klasa sadrži po 7000 primjera. Skup je podijeljen na 60000 primjera za učenje i 10000 za testiranje. Ideja autora navedenog podatkovnog skupa je pronaći zahtjevniji skup primjera koji će se moći vrlo jednostavno zamijeniti sa MNIST skupom. Dimenzije slika su 28x28 koje smo skalirali na 32x32. Slika (6.2) prikazuje primjere iz ovog podatkovnog skupa.

Slika 5.2: Slučajno odabrana 64 primjera iz podatkovnog skupa Fashion-MNIST

5.1.3. CIFAR10

Evaluaciju smo izvršili i nad podatkovnim skupom CIFAR-10 [3] koji sadrži slike u boji. Sastoji se od 60000 slika podijeljenih u 10 klasa. Neke od klasa su avioni, automobili, kamioni, psi, ptice te još nekoliko drugih. Slike su dimenzija 3x32x32, gdje prva dimenzija odgovara crvenom, plavom i zelenom kanalu slike (engl. RGB). Skup za učenje sadrži 50000 slika nad kojima smo trenirali modele. Slika (6.3) prikazuje primjere iz skupa.



Slika 5.3: Slučajno odabrana 64 primjera iz podatkovnog skupa CIFAR-10

5.2. Metrika

Razvoj dubokih generativnih modela potaknuo je i razvoj dobre evaluacijske mjere za usporedbu rezultata modela. U nedostatku značajnih evaluacijskih metoda, postaje izazov napredovanja novih modela te njihovih poboljšanja. Mjera koja se nedavno počela koristiti u većini istraživačkih radova koji rade s generativnim suparničkim modelima je Inception mjera (engl. Inception score) [10]. Inception mjera je neizravna metoda za evaluaciju koja koristi treniranu neuronsku mrežu na generiranim slikama i računa statistiku izlaza. Ideja je da mjera daje dobre rezultate za generirane slike koje su realistične i raznovrsne te koreliraju s vizualnom kvalitetom [23]. Prikazano je da Inception mjera vrlo dobro korelira s ljudskom procjenom vizualne kvalitete slike [10]. Duboki konvolucijski model Inception v3[5] napravljen je za zadatak klasifikacije na skupu ImageNet koji sadrži 1.2 milijuna slika u boji raspodijeljenih u 1000 klasa. Zadatak modela je klasificirati sliku x u klasu y s vjerojatnošću $p(y|x) \in [0,1]^{1000}$. Inception v3 model se koristi za klasificiranje generirane slike i izračun statistike pri izračunu Inception mjere na sljedeći način:

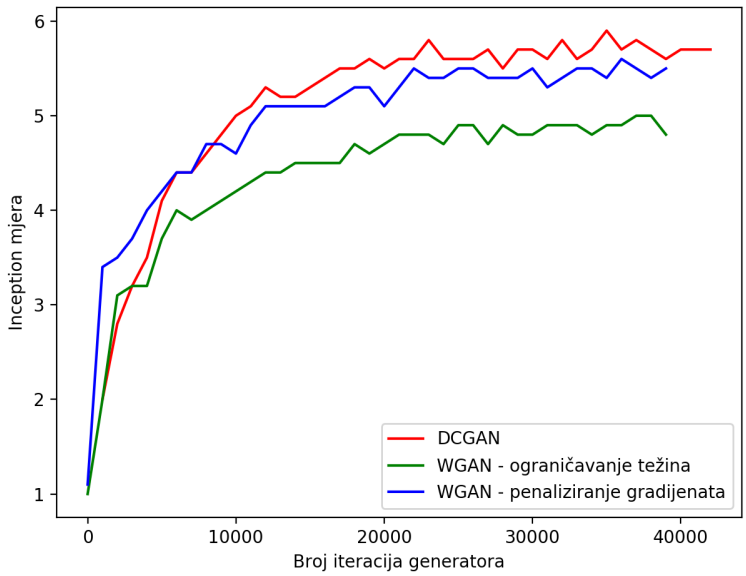
$$IS(G) = \exp\left(\mathbb{E}_{x\sim p_g} D_{KL}(p(y|x)||p(y))\right) \tag{5.1}$$

gdje $x \sim p_g$ označava da je slika uzorkovana iz distribucije generatora G. D_{KL} je Kullback-Leibler divergencija između uvjetne distribucije $p(y|x)$ i marginalne dis-

tribucije $p(y) = \int_x p(y|x)p_g(x)$. Autori su predložili navedenu mjeru zbog dva bitna svojstva:

- Inception v3 model trebao bi klasificirati objekt na generiranoj slici s velikom vjerojatnošću (slika nije mutna jer je objekt vidljiv) odnosno $p(y|x)$ ima nisku entropiju
- Generator bi trebao uzorkovati raznovrsne slike iz svih klasa skupa ImageNet

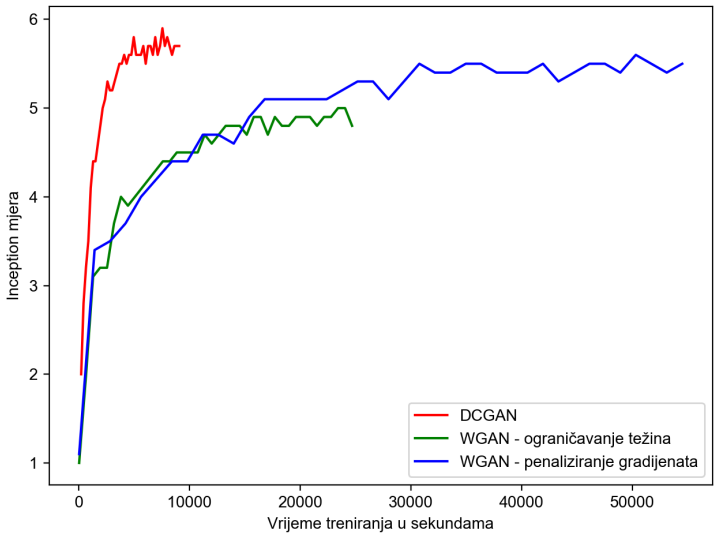
Ako generativni model generira slike s navedenim svojstvima tada očekujemo veću Kullback-Leibler divergenciju time i veću Inception mjeru. Inception mjeru smo koristili kako bi prikazali rezultate tri različita generativna modela (navedeni u 4. poglavlju) nad podatkovnim skupom CIFAR-10. Radi usporedbe Inception mjera na pravom podatkovnom skupu CIFAR-10 iznosi 9.4. Na slici (5.4) prikazujemo ovisnost Inception mjere o broju iteracija generatora.



Slika 5.4: Graf ovisnosti Inception mjere o broju iteracija generatora za 3 različita modela

Modele prikazane na grafu trenirali smo 40000 iteracija generatora te računali Inception mjeru nakon svake 1000-te iteracije na skupu od 8000 generiranih slika. Vidimo da modeli DCGAN i WGAN koji koristi penaliziranje gradijenata postižu sličnu mjeru, drugačija arhitektura WGAN modela (arhitektura ResNet) koja se koristi u izvornom radu [11] nakon 200000 iteracija generatora postiže Inception mjeru čak 7.8. Moguće da smo trenirali naš model WGAN-a duže dobili bi bolje rezultate od DCGAN

modela, no za to bi bilo potrebno nekoliko dana učenja mreže na korištenom serverskom računalu. Model WGAN koji koristi penaliziranje gradijenata generira puno kvalitetnije slike prema Inception mjeri u odnosu na WGAN model koji ograničava težine diskriminatora. Tablica (5.2) prikazuje generirane primjere za sva tri modela. Dobiveni rezultati su vrlo slični grafovima autora Wassersteinova modela [11] s razlikom u dužini treniranja generatora. Graf na slici (5.5) prikazuje ovisnost Inception mjere o vremenu treniranja u sekundama. Na njemu vidimo da je konvolucijskom modelu potrebno puno manje vremena za treniranje što omogućuje brže testiranje i evaluiranje različitih konfiguracija modela.



Slika 5.5: Graf ovisnosti Inception mjere o vremenu treniranja modela u sekundama 3 različita modela

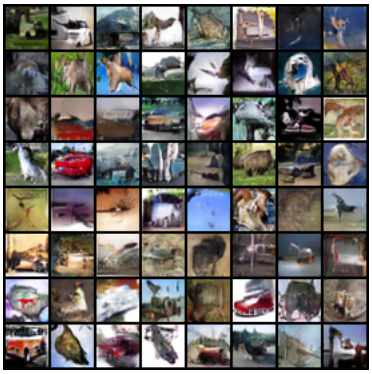
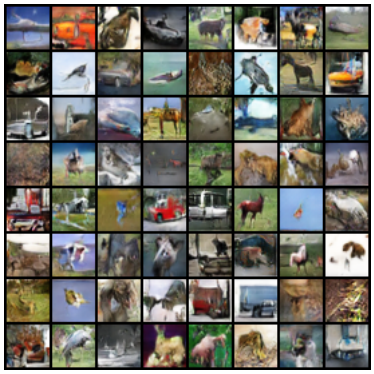
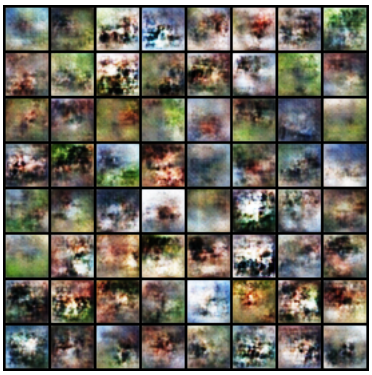
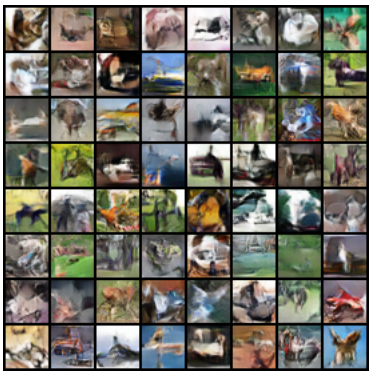
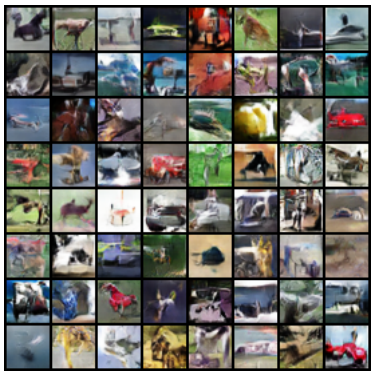
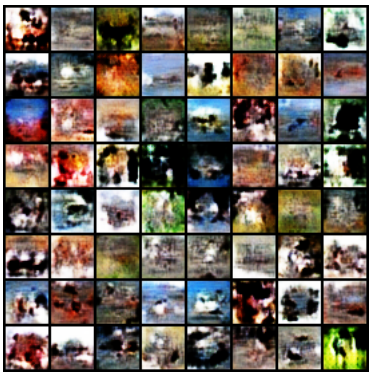
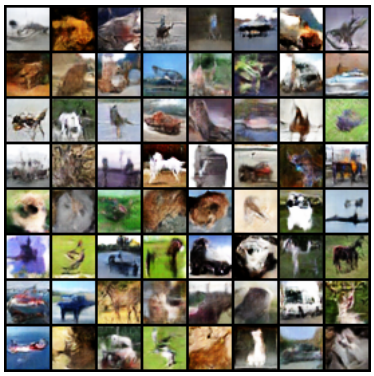
Naravno treba uzeti u obzir da Inception mjera nije savršena mjera za uspoređivanje generativnih model, može nam pomoći pri donošenju nekih zaključaka [10, 23]. Primjenjujući Inception mjeru nad podatkovnim skupovima koji nisu CIFAR-10 ili CIFAR-100 može dovesti do pogrešaka u rezultatu. Mjera je predložena za korištenje nad CIFAR-10 skupu radi svoje jednostavnosti i mogućnosti brzog iteriranja novih modela.

Evaluacija nad podatkovnim skupovima MNIST i Fashion-MNIST nije moguća koristeći se Inception mjerom jer slike nisu u boji i klasifikacija nema veze s klasifikacijom ImageNet skupa. Stoga njih ćemo prikazati na temelju kvalitete slika dobivenih generativnim modelima. Tablica (5.1) prikazuje generirane slike skupa Fashion-MNIST, a slike generirane za MNIST skup nalazi se u dodatku A.

Tablica 5.1: Generirani primjeri tijekom treniranja modela nad podatkovnim skupom Fashion-MNIST. Broj iteracija se odnosi na broj iteracija generatora kada su generirane slike. Prvi redak prikazuje generirane primjere slika za konvolucijski model (engl. DCGAN), sljedeći redak prikazuje Wassersteinov model koji koristi ograničavanje težina diskriminatora (engl. WGAN-Clipping, WGAN-CP). Zadnji redak prikazuje generirane primjere pri treniranju Wassersteinovog modela koji koristi penaliziranje gradijenata diskriminatora (engl. WGAN-Gradient Penalty, WGAN-GP)

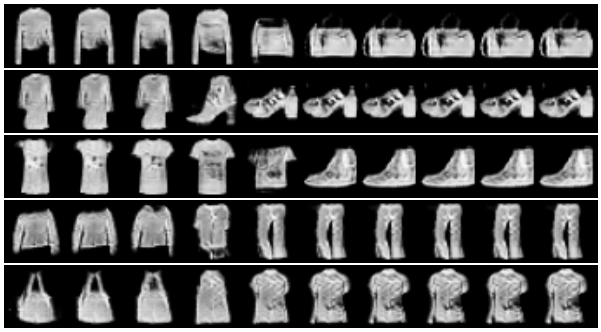
	1000. iteracija	**10000. iteracija**	**20000. iteracija**
DCGAN			
WGAN-CP			
WGAN-GP			

Tablica 5.2: Generirani primjeri tijekom treniranja modela nad podatkovnim skupom CIFAR-10. Broj iteracija se odnosi na broj iteracija generatora kada su generirane slike. Prvi redak prikazuje generirane primjere slika za konvolucijski model (engl. DCGAN), sljedeći redak prikazuje Wassersteinov model koji koristi ograničavanje težina diskriminatora (engl. WGAN-Clipping, WGAN-CP). Zadnji redak prikazuje generirane primjere pri treniranju Wassersteinovog modela koji koristi penaliziranje gradijenata diskriminatora (engl. WGAN-Gradient Penalty, WGAN-GP)

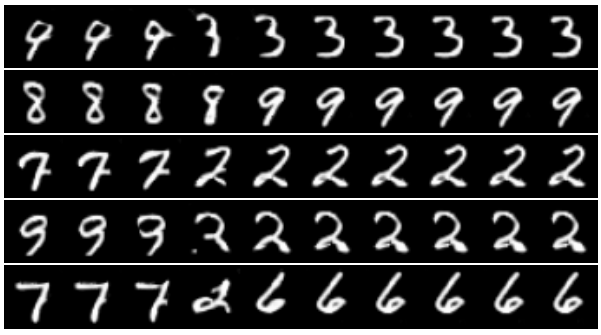
	1000. iteracija	**20000. iteracija**	**40000. iteracija**
DCGAN			
WGAN-CP			
WGAN-GP			

5.3. Šetnja u latentnom prostoru

Jedan od eksperimenata koji možemo iskoristiti za razumijevanje naučenog latentnog prostora generativnih modela je prikazivanje šetnje u latentom prostoru. Modeli implementirani u radu koriste vektor z dimenzije 100 uzorkovan iz $\mathcal{N}(0, 1)$ iz kojeg kroz generator generiramo primjere. Kvaliteta naučenog mapiranja koje obavlja generator ovisi o parametrima θ_g generatora G. Ako uzmemo dva slučajna vektora z_1 i z_2 te između njih prolazimo po točkama na istoj hiperavnini i generiramo primjere. Glatki prijelazi između generiranih primjera koji rezultiraju semantičkim promjenama na slikama ukazuju da je model naučio relevantne i zanimljive reprezentacije. Prikazat ćemo latente šetnje Wassersteinovog modela koji koristi penaliziranje gradijenta za podatkovne skupove Fashion-MNIST i MNIST. Na slikama vidimo da se događaju glatki prijelazi između klasa odjeće te također kod prikaza brojeva. Generator je vrlo dobro naučio distribuciju podataka za treniranje u slučaju brojeva MNIST i u slučaju klasa odjevnih predmeta Fashion-MNIST.



Slika 5.6: Prikaz latente šetnje, jedan red predstavlja jedan primjer šetnje između dva slučajna vektora nad podatkovnim skupom Fashion-MNIST



Slika 5.7: Prikaz latente šetnje, jedan red predstavlja jedan primjer šetnje između dva slučajna vektora nad podatkovnim skupom MNIST

5.4. Izvlačenje slikovne reprezentacije

Jedna od često korištenih metoda evaluacije kvalitete slikovnih reprezentacija algoritama nenadziranog učenja je njegova primjena na klasifikacijskom zadatku [1]. Model koji je treniran potpuno nenadzirano iskoristimo kako bi izvukli slikovne reprezentacije podatkovnog skupa za klasifikacijski zadatak. Evaluiramo rezultate tako da dobivene slikovne reprezentacije stavimo kao ulaz u jednostavni linearni klasifikator. Na ovaj način dobit ćemo poštenu usporedbu slikovnih reprezentacija različitih nenadziranih algoritama jer linearni klasifikator nema dovoljni kapacitet da dodatno unaprijedi rezultate klasifikacije [15]. Evaluaciju rezultata obavit ćemo nad podatkovnim skupom CIFAR-10, gdje ćemo 50000 primjera koristiti za učenje linearnog klasifikatora, a 10000 primjera za testiranje točnosti klasifikacije.

Diskriminator generativnog suparničkog modela nakon treniranja možemo iskoristiti za izvlačenje slikovnih reprezentacija tako da izbacimo zadnju aktivaciju na izlazu diskriminatora. Kako bi usporedili slikovne reprezentacije koristit ćemo diskriminator konvolucijskog generativnog suparničkog modela (engl. DCGAN) treniranog 1000 iteracija generatora, isti model treniran 40000 iteracija, Wassersteinov model s penaliziranjem gradijenata treniran također 40000 iteracija generatora. Prikazat ćemo rezultate klasifikacije nad pikselima slike, svaku sliku ćemo pretvoriti u jednodimenzionalni vektor koji ćemo koristiti kao ulaz u linearni klasifikator. Usporedbu ćemo napraviti i s modelom koji je treniran nadzirano na podatkovnom skupu ImageNet. Tehnikom prijenosa učenja izvući ćemo slikovne reprezentacija koristeći trenirani duboki konvolucijski model ResNet152 [16]. Rezultate točnosti klasifikacija prikazat ćemo na dva linearna klasifikatora, logističkoj regresiji i linearnom SVM-u.

Tablica 5.3: Tablica prikazuje točnost klasifikacija CIFAR-10 podatkovnog skupa koristeći linearni SVM kao klasifikator

Metoda izvlačenja značajki	**Dimenzionalnost vektora značajki**	**Točnost klasifikacije**
Pikseli slike	3072	23.50%
DCGAN - 1000	16348	44.47%
DCGAN - 40000	16348	59.95%
WGAN-GP 40000	16348	**61.15%**
ResNet152	2048	**73.52%**

Rezultati pokazuju da generativni suparnički modeli mogu naučiti dobru slikovnu reprezentaciju do neke mjere, ali modeli trenirani nadzirano postižu puno bolje rezultate. Tablica (5.3) i (5.4) pokazuju da je točnost klasifikacije slična za dva jednostavna linearna klasifikatora, linearni SVM i logističku regresiju. Korištene su implementacije iz biblioteke strojnog učenja sklearn. Bolji rezultati mogli bi se postići da u Wasserstinovom modelu koristimo ResNet [16] arhitekturu kao u radu [11] te da se model trenira 200000 iteracija generatora.

Tablica 5.4: Tablica prikazuje točnost klasifikacija CIFAR-10 podatkovnog skupa koristeći logističku regresiju kao klasifikator

Metoda izvlačenja značajki	**Dimenzionalnost vektora značajki**	**Točnost klasifikacije**
Pikseli slike	3072	26.87%
DCGAN - 1000	16348	45.67%
DCGAN - 40000	16348	59.10%
WGAN-GP 40000	16348	**61.39%**
ResNet152	2048	**75.12%**

6. Zaključak

Područje dubokih generativnih modela počelo se sve brže razvijati, nakon predloženog generativnog suparničkog modela koji ostvaruje odlične rezultate u nenadziranom učenju. U ovom radu su opisane duboke konvolucijske mreže, konvolucijski slojevi te bitne razlike vezane uz njih kao i relevantni optimizacijski postupci. Fokus rada je bio na prikazu različitih generativnih suparničkih modela te njihovih konfiguracija. Opisani su duboki konvolucijski suparnički model i dvije inačice Wassersteinovog modela te njihove razlike. Navedeni modeli iskorišteni su za izvlačenje slikovnih reprezentacija slika koje smo evaluirali nad jednostavnim linearnim klasifikatorom.

U sklopu rada implementirano je više inačica generativnih suparničkih modela od jednostavnijih do modela koji postižu najbolje rezultate (engl. state-of-the-art). Takvi modeli trenutno se mogu vrlo stabilno trenirati u različitim arhitekturama. Mreže su implementirane korištenjem razvojnog okvira Pytorch s podrškom za CUDA paralelno procesiranje. Implementirani modeli su evaluirani nad tri različita podatkovna skupa MNIST, Fashion-MNIST i CIFAR-10. Rezultate smo prikazali koristeći nekoliko metoda evaluacije. Kvaliteta generiranih primjera slika i rezultati Inception mjere vrlo su slični onima iz literature [11]. Rezultati vezani uz slikovne reprezentacije pokazuju da modeli nenadziranog učenja ne postižu tako dobre rezultate kao modeli učeni na nadzirani način.

U budućem radu bilo bi zanimljivo isprobati nove predložene suparničke modele te drugačije arhitekture. Dobiveni rezultati mogli bi se poboljšati dužim treniranjem određenih modela. Generativni suparnički modeli vrlo brzo napreduju, istraživački radovi stalno poboljšavaju kvalitetu generiranih primjera kao i metode evaluacije modela koje su vrlo značajne za napredak područja.

LITERATURA

[1] S. Chintala A. Radford, L. Metz. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016. URL https://arxiv.org/pdf/1511.06434.pdf.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, i Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[3] AlexKrizhevsky. Learning multiple layers of features from tiny images. 2009. URL https://www.cs.toronto.edu/~kriz/cifar.html.

[4] T. Chen M. Li B. Xu, N. Wang. Empirical evaluation of rectified activations in convolutional network@article. 2015. URL https://arxiv.org/abs/1505.00853.

[5] Sergey Ioffe Jonathon Shlens Zbigniew Wojna Christian Szegedy, Vincent Vanhoucke. Rethinking the inception architecture for computer vision. 2015. URL https://arxiv.org/pdf/1512.00567.

[6] J. Lei Ba D. P. Kingma. Adam: A method for stoahastic optimization. 2014. URL https://arxiv.org/pdf/1412.6980.

[7] Serena Yeung Fei-Fei Li, Justin Johnson. Cs231n: Convolutional neural networks for visual recognition. 2018. URL http://cs231n.stanford.edu/.

[8] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. 2016. URL https://arxiv.org/abs/1701.00160.

[9] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[10] W. Zaremba V. Cheung A. Radford X. Chen I. Goodfellow, T. Salimans. Improved techniques for training gans. 2016. URL https://arxiv.org/abs/1606.03498.

[11] M. Arjovsky V. Dumoulin A. Courville I. Gulrajani, F. Ahmed. Improved training of wasserstein gans. 2017. URL https://arxiv.org/abs/1704.00028v25.

[12] M. Mirza B. Xu D. Warde-Farley-S. Ozair A. Courville Y. Bengio I. J. Goodfellow, J. Pouget-Abadie. Generative adversarial networks. 2014. URL https://arxiv.org/abs/1406.2661.

[13] T. Darrell J. Long, E. Shelhamer. Fully convolutional networks for semantic segmentation. 2016. URL https://arxiv.org/pdf/1605.06211.pdf.

[14] T.ox M. Riedmiller J. Tobias Springenberg, A. Dosovitskiy. Striving for simplicity: The all convolutional net. 2015. URL https://arxiv.org/pdf/1412.6806.pdf.

[15] Marko Jelavić. *Diplomski rad, Fakultet elektrotehnike i računarstva, Zagreb*.

[16] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. 2015. URL https://arxiv.org/abs/1512.03385.

[17] L. Bottou M. Arjovsky. Towards principled methods for traning generative adversarial networks. 2017. URL https://arxiv.org/abs/1701.04862.

[18] L. Bottou M. Arjovsky, S. Chintala. Wasserstein gan. 2017. URL https://arxiv.org/abs/1701.07875.

[19] Augustus Odena, Vincent Dumoulin, i Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. URL http://distill.pub/2016/deconv-checkerboard.

[20] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, i Adam Lerer. Automatic differentiation in pytorch. 2017.

[21] Teck Yian Lim Alexander G. Schwing Mark Hasegawa-Johnson Minh N. Do Raymond A. Yeh, Chen Chen. Semantic image inpainting with deep generative models. 2016. URL https://arxiv.org/abs/1607.07539.

[22] C. Szegedy S. Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015. URL https://arxiv.org/pdf/1502.03167v3.pdf.

[23] Rishi Sharma Shane Barratt. A note on the inception score. 2018. URL https://arxiv.org/abs/1801.01973.

[24] Xiaowen Chu Shaohuai Shi. Speeding up convolutional neural networks byexploiting the sparsity of rectifier units. 2017. URL https://arxiv.org/pdf/1704.07724.pdf.

[25] Thalles Silva. An intuitive introduction to generative adversarial networks (gans). 2017. URL https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans

[26] Francesco Visin Vincent Dumoulin. A guide to convolution arithmetic for deep learning. 2016. URL https://arxiv.org/abs/1603.07285.

[27] Han Xiao, Kashif Rasul, i Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017. URL http://arxiv.org/abs/1708.07747.

[28] Christopher J.C. Burges Yann LeCun, Corinna Cortes. Gradient-based learning applied to document recognition. 1998. URL http://yann.lecun.com/exdb/mnist/.

Dodatak A
Dodatne slike rezultata

Tablica A.1: Generirani primjeri tijekom treniranja modela nad podatkovnim skupom MNIST.

	1000. iteracija	20000. iteracija	40000. iteracija
DCGAN			
WGAN-CP			
WGAN-GP			

Izlučivanje slikovnih reprezentacija generativnim suparničkim modelima

Sažetak

Nenadzirano učenje je vrlo zanimljivo područje jer omogućuje iskoristiti velike količine dostupnih neoznačenih podataka. U radu se bavimo generativnim suparničkim modelima koji omogućuju generiranje slika iz distribucije podatke te izvlačenje slikovnih reprezentacija. Opisane su duboke neuronske mreže s fokusom na konvolucijske mreže. Detaljno je prikazano nekoliko tipova generativnih suparničkih modela i njihove konfiguracije. Implementirani su duboki generativni suparnički model i dvije inačice Wassersteinovog modela u Pytorch razvojnom okviru. Modeli se evaluirani na tri različita podatkovna skupa MNIST, Fashion-MNIST i CIFAR-10. Na kraju su prikazani dobiveni rezultati, s opisima i slikama.

Ključne riječi: Duboki generativni suparnički modeli, učenje reprezentacije, nenadzirano učenje

Extracting image representations with generative adversarial models

Abstract

Unsupervised learning is a very interesting area because it allows one to take advantage of large volumes of unlabeled data. This paper deals with generative adversarial networks that enable generation of images from given data distribution and feature extraction. Deep neural networks are described with emphasis on convolutional networks. Several types of generative adversarial networks are shown with their configuration. A deep convolutional adversarial model and two versions of Wasserstein model are implemented using the Pytorch library. Models were evaluated on three different sets of data MNIST, Fashion-MNIST and CIFAR-10. Finally, we present results with description and images.

Keywords: Deep generative adversarial models, representation learning, unsupervised learning