



**NEUMANN JÁNOS
INFORMATIKAI KAR**



SZAKDOLGOZAT

OE-NIK

Hallgató neve:

Tuba Márk

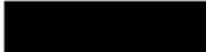
2022

Hallgató törzskönyvi száma:

T/006503/FI12904/N

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Tuba Márk**
Törzskönyvi száma: T/006503/FI12904/N
Neptun kódja: 

A dolgozat címe:

Arckép generálás neurális hálózattal
Human face generation using neural networks

Intézményi konzulens: Sipos Miklós László, Szabó-Gali Ákos
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Szoftvertechnológia és grafikus felhasználói interfész
tervezése
Szoftvertervezés és -fejlesztés specializáció

A feladat

A feladat egy olyan neurális hálózat készítése, melyet felhasználva képesek vagyunk magas minőségű képek generálására. Készítsen el egy megfelelő felépítésű neurális hálózatot Tensorflow segítségével. Vizsgálja meg, hogy az egyes technológiák mellett még milyen egyéb módszerekkel lehetne megvalósítani a feladatot. Az architektúra felépítésekor vizsgálja meg a diszkriminátor és a generátor közötti kapcsolatot, illetve milyen biasre és aktivációs függvényre van szükségünk egy ilyen hálózathoz. A hálózathoz használja fel az MNIST adathalmazt. Mielőtt továbbfejlesztésre kerülne a hálózat, ezen az adathalmazon kell kiértékelni, hogy az elkészített architektúra rendesen funkcionál. Ezután az elkészített architektúra konvolúciós rétegeinek bővítését végezze el, hogy képes legyen nagyobb felbontású képeken tanulni. A hálózathoz nagyon fontos cél, hogy ha egy azonos felbontású képekkel rendelkező adathalmazon, még ha különböző képekből is állna, újra használható legyen a kód és ennek demonstrálása egy másik adathalmazon. A feltanított neurális hálózatnak diszkriminátora elhagyásával pedig ugyan úgy lehetséges legyen a képek generálása. Végezzen továbbá elemzést a tesztelési lehetőségekről és készítse is el az egyes teszt típusokat és teszteseteket.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- magas szintű szerkezeti mintát az alkalmazásról,
- az egyes funkciók megvalósításához felhasználható lehetséges technológiákat,
- az elkészített rendszer tervét, blokkdiagramját,
- a megvalósítás menetét, felmerülő problémákat és az azokra adott megoldást,
- a tesztelési tervet, a tesztelés módszertanát és a tesztelés eredményeit,
- az elkészült szoftver felhasználási lehetőségeit,



Vámossy Zoltán

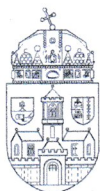
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

Lipos Zoltán
.....
intézményi konzulens



HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. május 13.

Tilla M. B.

hallgató aláírása



KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Tuba Márk Nappali

Telefon: Levelezési cím (pl: lakcím):

.....

Szakdolgozat / Diplomamunka¹ címe magyarul:

Arcképek generálása neurális hálózattal

Szakdolgozat / Diplomamunka² címe angolul:

Human face generation using neural networks

Intézményi konzulens:

Külső konzulens:

Szabó-Gali Ákos.....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.15	Tárggyal kapcsolatos információk egyeztetése	
2.	2021.09.22	Szakdolgozat név és téma átbeszélése	
3.	2021.11.05	Szakirodalom áttekintése	
4.	2021.12.07	Rendszerterv áttekintése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4³ tantárgy követelményét teljesítette, beszámolóra / védésre⁴ bocsátható.

Budapest, 2021.12.08

.....
intézményi konzulens

¹ Megfelelő aláhúzendó!

² Megfelelő aláhúzendó!

³ Megfelelő aláhúzendó!

⁴ Megfelelő aláhúzendó!



KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Tuba Márk [REDACTED] Nappali

Telefon: Levelezési cím (pl: lakcím):

[REDACTED]

Szakdolgozat / Diplomamunka¹ címe magyarul:

Arcképek generálása neurális hálózattal

Szakdolgozat / Diplomamunka² címe angolul:

Human face generation using neural networks

Intézményi konzulens:

Külső konzulens:

Szabó-Gali Ákos.....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.27	Tárggyal kapcsolatos információk egyeztetése	[Signature]
2.	2022.03.18	Fejlesztés áttekintése	[Signature]
3.	2022.04.25	Eredmények áttekintése	[Signature]
4.	2022.05.07	Tesztelés áttekintése	[Signature]

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4³ tantárgy követelményét teljesítette, beszámolóra / védésre⁴ bocsátható.

Budapest, 2022.05.08

[Signature]
.....
intézményi konzulens

¹ Megfelelő aláhúzendó!

² Megfelelő aláhúzendó!

³ Megfelelő aláhúzendó!

⁴ Megfelelő aláhúzendó!

Absztrakt

A mai világunkban szinte akármilyen hirdetmény vagy akár weboldal tekintése közben találkozni foguk emberi arcképekkel. Ezeket licencezni kell, esetleg modelleket keresni, fényképészt és lehetséges, hogy felszerelést is kell bérelni. Nagyon sok időbe és pénzbe kerülhet ez a folyamat. Pont erre a célra lehet felhasználni a generált arcképeket.

Célom egy olyan neurális hálózat készítése, amely képes nagy minőségű, egyedi arcképek generálására. Fontos szempont számomra a gyors generálás, de lassú és alapos tanítás. A rendszer pedig felhasználható legyen más projekteknél is, ezért a súlyainak a mérete minél kisebb legyen.

A szakdolgozatomban specifikálom az arcképek felépítését és az általam fejlesztett rendszernek az elvárásait. Tárgyalom az ilyen fajta kép generálásnak a komplex kihívásait is. Ezután áttekintem a kép generálás feladatra képes neurális hálózatokat, ezeknek rétegzéseit és felhasznált technológiáit. Az ilyen rendszerek áttekintése után pedig mélyebb architektúrai szempontokról lesz szó. A neurális hálózatom rétegzését, architektúráját és felhasznált technológiáiról is beszámolok.

A neurális hálózatomat otthoni GPU-val tanítottam, tensorflow környezetben amely az egyik legelterjedtebb mély tanulós framework. Az adathalmaz, rétegzés és tanítás implementációs komplexitására is kitérek a fejlesztés során. A hálózat különböző architektúrai problémáinak diagnosztizálása, manuális értékelési módszer segítségével történt. Kijavításuk után, pedig hiperparaméter optimalizálást hajtottam végre, amely közben számszerűsítettem is az eredményeket, IS és FID mennyiségi értékelési módszerekkel. Az értékelési módszerek az MNIST és FFHQ adathalmazokon voltak applikálva.

Abstract

In today's world, you will notice human faces in almost any advertisement or even website. These will need to be licensed, you may need to find people to model for you, a photographer and possibly even hiring equipment. It can potentially cost a lot of time and money. This is exactly what the generated face images can be used for.

My goal is to create a neural network that is capable of generating high quality images that contain unique faces. It was a very important design philosophy for me that the network is able to generate images at a fast rate, at the cost of slow but thorough training. The neural network also needs to be portable, so the weights of the model need to be as small as possible. This is so I can reuse the model in another project.

In my thesis I specify the structures of faces and the requirements for my network. I will discuss the complex challenges that come with this kind of image generation. Then I will review the neural networks that are capable of image generation, their layering and the technologies used. After the examination of these models, I will investigate their deeper architecture. The layering, architecture, and technologies I've used for my model will also be discussed.

My model was trained on my own GPU, developed in a tensorflow environment, which is one of the most popular deep learning frameworks. The implementation details about the dataset, layering and training will also be reviewed. The various architectural problems were diagnosed using manual evaluation techniques. After correcting these problems, I performed hyperparameter optimization and quantified the results using IS and FID quantitative evaluation methods. The manual and quantitative evaluation methods were used on the MNIST and FFHQ datasets.

Tartalomjegyzék

1. Bevezetés	1
2. Irodalomkutatás	3
2.1. Arcképek felépítése	3
2.2. Arcképek generálása	5
2.3. Generatív modell	6
2.3.1. Felügyeletlen tanítás	6
2.3.2. Mély generatív modell	7
2.4. Hasonló rendszerek és technológiái	8
2.4.1. Variational autoencoderek	8
2.4.2. Autoregressive generative model	10
2.4.3. GAN	13
2.4.4. Továbbfejlesztett GAN modellek	15
2.5. Aktivációs függvények	20
2.5.1. Sigmoid és hasonló függvények	20
2.5.2. ReLU	21
2.5.3. ReLU alternatívák	22
2.6. Paraméter inicializálás	22
2.7. Veszteségfüggvények	23
2.7.1. Veszteségfüggvény optimalizálása	24
2.8. Szabályozás	25
2.8.1. Alul-tanulás és túlilleszkedés	25
2.8.2. Dropout	26
2.8.3. Kötegelt normalizálás	27
2.9. Hiperparaméterek optimalizálása	27
2.9.1. Sztochasztikus gradiens ereszkedés módszer	28
3. Követelmény specifikáció	30
4. Tervezés	31
4.1. Mély tanuláshoz használt szoftver és hardver	31
4.1.1. Hardver mély tanulásra	31
4.1.2. Szoftver választása	32
4.2. Adathalmaz specifikáció	33

4.2.1. Adathalmaz előkészítése	34
4.3. Fejlesztés tervezett menete	35
4.4. Rendszer funkcionalitása	36
5. Fejlesztés folyamata	38
5.1. Architektúra	38
5.1.1. Prototípus	38
5.1.2. Prototípus továbbfejlesztési lehetőségei	40
5.1.3. Architektúra specifikáció	42
5.2. Modell implementálása	42
5.2.1. Adathalmaz előkészítése	43
5.2.2. Hálózati blokkok és rétegzés implementálása	44
5.2.3. Tanítás implementálása	46
5.3. Modell tanítása MNIST adathalmazon	47
5.3.1. Tanítási adatok	47
5.3.2. Eredmények kiértékelése	48
5.4. Modell tanítása FFHQ adathalmazon	52
5.4.1. Tanítási adatok	52
5.4.2. Eredmények kiértékelése	54
5.5. Feltanított modell használata webapplikációban	55
6. Továbbfejlesztési lehetőségek	57
Hivatkozások	58
Ábrajegyzék	63

1. Bevezetés

Mai világunknak egyik legérdekesebb jelensége a mesterséges intelligencia (Artificial Intelligence), egy nagyon jó indok miatt. Rendkívül sok példát lehet arra mutatni, hogy mennyi terepen képesek a gépek legyőzni az embereket. A legjobb sakk és póker játékos egy mesterséges intelligencia, még olyan dolgokban is jobban teljesítenek, amikre sohasem gondolhattunk volna, mint például az aljas Captcha megfejtésében. Ez egy olyan teszt, ami direkt azért lett készítve, hogy ne engedjenek be botokat adott weboldalakra. Az elmúlt néhány évtizedben nagyon bő tudományi ág lett a gépi tanulásból, illetve a neurális hálózatokból. Egy dolog kiemelkedik amely emberek számára egyszerű, de ezeknek a modelleknek a számára jelentősen nehéz. Ez lenne a gépi látásnak a tudománya.

A gépi látás az egyik legaktívabb kutatási területe a mesterséges intelligenciának, ahol általában képekből vagy videókból információt nyerünk ki és átdolgozzuk valamilyen formába. A gépi látásnak különböző alkalmazásai közül az egyik az emberek vizuális képességeinek az előállítása, mint az emberi arcoknak a felismerése [1] amely a klasszifikáció és felismerés feladatba tartozik. Ezen kívül gépi látásnak az alkalmazásai képesek, képekre feliratokat generálni [2] amelyek képesek leírni, hogy mit tartalmaz egy kép, képeknek a beszínezése [3] és sok más.

Elég sok információval rendelkezik egy kép, lehet nagy felbontású, lehet szürkeárnyaltos vagy akár többcsatornás(RGB) is. A nagyobb felbontás pedig exponenciálisan több számítási időt igényel. Emellett még beszélhetünk arról is, hogy mit szeretnénk generálni, mi legyen rajta a képen, legyen-e valamilyen variáció a képek között. Legyen-e egyáltalán háttérben vagy csak az előtérben legyen objektum. Itt jönnek be a mesterséges neurális hálózatok.

A mesterséges neurális hálózatok (ANN), az emberi agyhoz hasonlóan, neuronokból állnak össze, amelyek össze vannak kötve egymással. Ezekre a neuronokra, nézhetünk úgy is mint csomópontokra egy gráfban, kapnak valamilyen adat értéket, feldolgozzák és továbbküldik egy másik csomópontra. Ilyen módon hasonlóak egy funkcióra is, mivel ezek az adatok számok szoktak lenni, amelyet feldolgoznak és továbbküldik. A neuronoknak létezik w súlyuk is, amely tanítási folyamat során szabályozva van a megfelelő érték felé. A mesterséges neurális hálózatoknak egy továbbfejlesztett verziójáról lesz szó a szakdolgozatomban, amelyeket mély neurális hálózatoknak(DNN) nevezünk. Ezek olyan mesterséges neurális hálózatok amelyek több réteggel rendelkeznek a bemeneti és a kimeneti

réteg között.

A mély neurális hálóknak, a mély konvolúciós neurális hálózat(CNN) változata, hatalmas fejlődés volt a gépi látás tudományágban, főleg a kép és videó feldolgozás szempontjából. A CNN hálózatok a felépítésükből adódóan több dimenziós tömbökkel dolgoznak, ami azért különösen előnyös a kép feldolgozás szempontjából, mert a képeket is több dimenziós tömbökként vannak reprezentálva. Eredetileg csak diszkriminatív célra használták mint klasszifikáció, de ez megváltozott amikor egy feltanított CNN modellt felhasználtak arra, hogy rekonstruálják az aktiválásait egy adott osztálynak [4]. A legvégére képesek voltak vizualizálni, hogy a modell milyen képeket "gondolt" annak az adott osztálynak. Ekkor kezdődött el egy teljesen új tudományi ág, olyan képek generálása amelyeket még nem látott senki. Néha kép szintézisként is fogok rá hivatkozni. A kép generálás feladatára, a legelterjedtebb modellek, az úgynevezett mély generatív modellek, amelyek a felügyeletlen tanítás kategóriába tartoznak.

A kép generálásnak egy specifikus témájára fókuszálok, az emberi arckép generálására. Az emberi arcképek általában követik ugyan azt a rejtett struktúrát, van egy fej, arc, szemek és így tovább. De ahhoz képest, hogy hasonló struktúráról beszélünk, valahogy mindegyik arc egyedi. Emberi arcok emellett még nagyon sok különböző attribútummal rendelkezhetnek mint nem, kor, rassz és ez is sorolható. Habár most nem arc felismerésről lesz szó, ezek az attribútumok fontosak lesznek arc szintézisnél is. Az arckép generálásnak még egy kihívása lesz, hogy mi emberek nagyon jók vagyunk, más emberek felismerésében. Ha esetleg a modell által generált kép már egy kicsit is az "uncanny valley" terében van, már akkor észrevesszük, hogy itt valami nincs rendben. Az "uncanny valley" az egy angol kifejezés amely a borzongások völgyére fordítható le, ezt az érzetet eredetileg a humanoid robotokra mondták, ahol az emberi agy képes egyszerűen megmondani, hogy ez a robot nem ember. Ez sok más kép generálás feladatnál előfordulhat, hogy elő se jön. A célom a szakdolgozatomban, új emberi arcképek generálása amelyek nem teljesen egyeznek meg az adathalmazban lévő mintákkal és nem esnek bele a borzongások völgyébe.

Ian Goodfellow fantasztikus könyvét vettem alapul, nagyon sok résznél, ennek a szakdolgozat megírásában és többször is hivatkozni fogok rá [5]. Goodfellow leírása szerint egy neurális hálózathoz kombinálnunk kell egy adathalmaz specifikációt, egy veszteségfüggvényt, egy optimalizációs eljárást és egy modellt. Ezeket mind átfogok menni a szakdolgozatomban, egy kép szintézis szempontból és áttekintem a generatív modell architektúráit amelyet fellehet használni arc szintézisre.

2. Irodalomkutatás

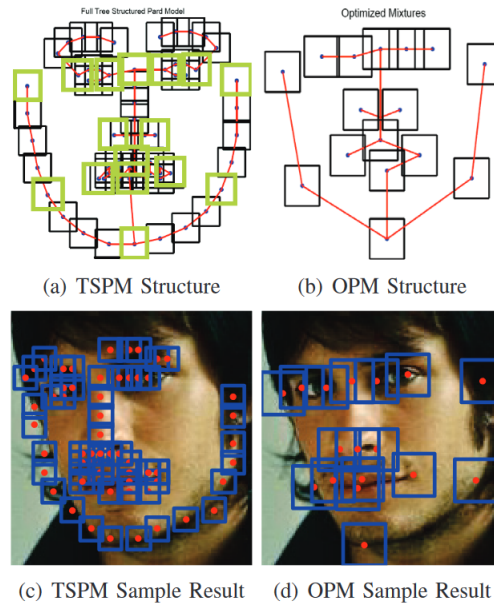
Az irodalomkutatás részben, szó lesz az arcképek felépítéséről, ezeknek a generálásáról és kihívásairól. Ezután pedig áttekintem az arcképek generálására alkalmas rendszereket és technológiákat. Végül pedig a neurális hálózatok architektúráinak különböző részeiről fogok írni.

2.1. Arcképek felépítése

A szakdolgozatomban az arckép generálásról van szó, de olyan arcképek generálása amely nem része a kiindulási adathalmaznak. Tehát teljesen új, eddig még nem látott emberi arcképek generálása, amelyek egyedi arc attribútumokkal rendelkeznek. Egy emberi arcnak az attribútumai alatt, a biológiai jellegzetességekről van szó, amelyek a különböző személyek közötti, egyéni variációt adó, arcjellemzőket képviselnek. Minden ember rendelkezik valamilyen egyedi arc attribútummal, amelyet képesek vagyunk felhasználni a személynek az azonosítására. Ezek az arc attribútumok képesek egy kis lokális, vagy esetleg globális változást okozni az arcon. Globális változás alatt olyan attribútumokra kell gondolni amelyek az egész arcot vagy fejet képesek befolyásolni. Ilyen attribútumok egy személynek az életkora, a neme és a rassza.

Lokális változás alatt pedig olyan attribútumokról van szó, amelyek csak egy részét határozzák meg a fejnek. Lokális attribútum például a haj, amely lehet különböző típusú, hosszú, sűrűségű, színű, továbbá a kopaszságnak a mértéke is ide tartozhat. Arcszőrzet is hasonlóan a hajhoz, lehet különböző típusa, hossza, sűrűsége és színe. A személynek a hangulata is képes lokális változást okozni, boldogság képes mosolygást és nevetést okozni, mérgesség a szemöldököket összehúzni, undorodás pedig az orrunkat felfele húzni. A fej póza is fontos lehet mivel ez képes a fejnek adott részeit kisebbiteni vagy nagyobbítani esetleg teljesen kitakarni. A személy hordhat valamilyen fajta sminket is. Emellett beszélhetünk valamilyen fajta ruháról is, mint kalap, maszk, szemüvegek és ékszerek, amelyek főleg csak kiegészítik a létező arcot.

Arcunk struktúráját pedig néhány úgynevezett "face landmark" pont alkotja, amelyek az arcunknak különböző részein találhatóak. A face landmark pontok mozgatásával képesek vagyunk az arc struktúráját átalakítani, ezek a pontok pedig megszámlálhatatlan mennyiségű variációt képesek adni a különböző arcok között. Az [2-1](#) ábrán látható TSPM modell



2-1. ábra. TSPM és OPM modellek, amelyek a "Facial landmark modellek" családjába tartozik és képesek detektálni az emberi arcokon található face landmark pontokat(piros) és lokális foltokat(kék) [6]

fantasztikusan demonstrálja az állkapocsvonalunkon található face landmark pontokat. A szánkon található 2 pont a bal és jobb végén, emellett felvetődik a felső és alsó ajak amelyek képesek nagy variációra. Orrunkon is sok ilyen pont található, bal és jobb orrgyökér, orrlyuk, orrhegy és az orrnyergek. Szemeinknek lehet a külsejére vagy a pupillájára és a szemöldöknek a pozíciójára.

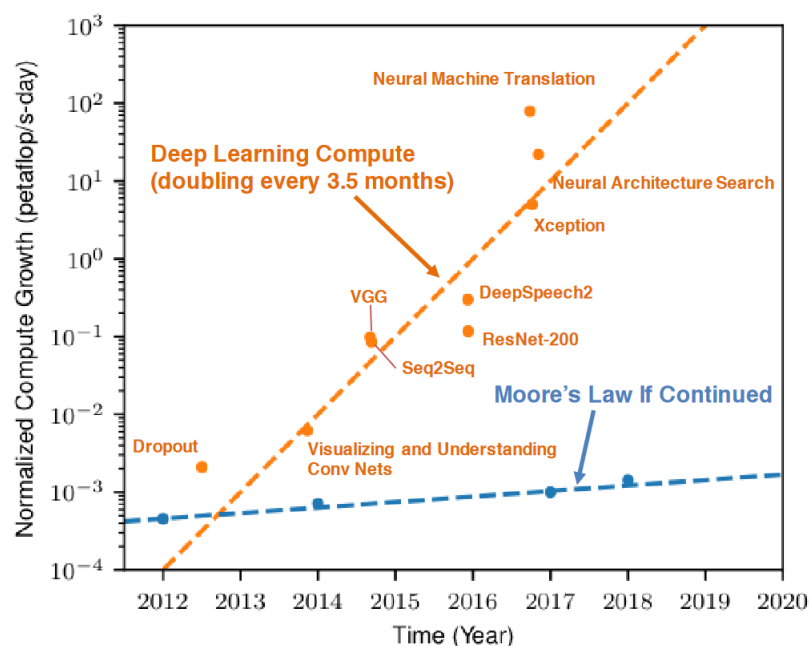
Kép minőség és felhasználhatóság terén pedig beszélhetünk, vizuális zajról amely belekerülhet a képbe. A képnek a minősége és egyben a képen lévő arcképnek a láthatósága is hatalmas faktor lehet, főleg ha egy saját adathalmazt készítünk. Esetleg ki vannak-e takarva az arcképnek adott részei. A háttérben lévő objektumok is rendelkezhetnek információval. A szakdolgozatomban lévő neurális hálózatban a háttérben lévő objektumokra nem fogok fókuszálni, hanem a cél az csak az előtérben lévő arcok generálása különböző arc struktúrákkal, arc attribútumokkal amely teljesen egyedi arcképeket lesz képes generálni.

2.2. Arcképek generálása

Szükségünk van egy olyan hálózatra vagy rendszerre amely képes megtanulni a felépítését és a komplexitását is az arcoknak. Emellett ezen kívül legyen képes egyedi arcképeknek a generálására. Ehhez a különböző tulajdonságait az arc attribútumoknak és a pozícióit az arc struktúrának, meg kell tanulnia. A leírt tanulási procedúrát végig kell futtatni több 1000, később láthatjuk hogy több 10000-100000 képen, számtalan epoch során. Egy epoch az egy általunk előredefiniált levágási pontja a tanítási fázisnak, amely lebontja több kisebb részre a tanítást. Ez a lebontás, fontos az eredmények naplózására és kiértékelésére. Szakirodalomban 100-300 epoch a feltanított modelleknek a szokásos tanítási ideje. Neurális hálózatoknak a tanítására és használására nagyon sok számítási teljesítményre van szükség. A képek felbontása is elég magas, illetve a jellemző attribútumokat is ki kell nyernünk belőlük, ami további számítási lépéseket igényel.

Neurális hálózatokat nagyon sok ideig csak egy elméleti síkon működő rendszerek voltak, mivel nem rendelkezünk megfelelő hardverrel, amelyek elég számítási kapacitással rendelkeztek. Ez a tény még ennél is igazabb volt az arcképeknek a generálására mivel az általam leírt követelmények sokkal komplexebbek mint egy egyszerű neurális hálózat. Nem csak számítási erővel voltak kihívások hanem még tárolóhely meg a memóriába betölthető adatoknak a mennyiségével is. A napjainkban használt adathalmazok már 2 TB felett is lehetnek, amelyeket memóriába lehetetlen betölteni. Neurális hálózatok viszont 2012-ben a mély tanulás(deep learning) miatt és egyéb fejlesztések miatt, mint gyors GPU implementációk, exponenciálisan több számítási teljesítményre voltak képesek. Az exponenciális fejlődés még a további évekre is igaz lett, amely látható az [2-2](#) ábrán.

A mély tanulásnak a számítási sebességének növelése lehetővé tette a képek és egyben az arcképeknek a generálását is. Ezt a kihívást főleg a generatív modellekkel oldották meg, ezeknek pedig egy új metódus családjára fókuszáltak, a mély generatív modellre. A mély generatív modell kombinálja a generatív modellt és a mély neurális hálózatokat. A mai napjainkban a state of the art minőséget a mély generatív modellek hálózatai érik el.



2-2. ábra. Mély tanulás számításának a növekedése 2012-től kezdve [7]

2.3. Generatív modell

2.3.1. Felügyeletlen tanítás

Tipikusan, felügyelt tanításhoz szükségünk van egy (x, y) adatpárra. x valamilyen bemeneti halmazt definiál, y pedig az elvárt kimenetet tartalmazó halmaz, sokszor címként(label) fogok rá hivatkozni. A célunk ezekkel az adatokkal hogy betanítsunk egy modellt ami képes x bemeneti adatokról egy prediktált y kimenetet előállítani. A modellünk megpróbálja az elvárt kimenet létezése nélkül prediktálni, hogy szerinte mi lehetett az elvárt kimenet. Ezután ezeket az adatokat felhasználjuk a modell tanítására, ahol összehasonlítjuk az elvárt y és a prediktált y értékeit. A felügyelt tanításos algoritmusok olyan adathalmazokat tartalmaznak ahol vannak z jellemzők de minden példája emellett még egy y címkével is össze van kötve [5]. Egy klasszikus példája a felügyelt tanításos algoritmusoknak a kép klasszifikáció, emellett még nagyon sok olyan problémára használt, ahol valamilyen osztály vagy számszerű predikcióra van szükségünk.

A felügyeletlen tanításban viszont csak egy x bemeneti adat halmazára van szükségünk. Az előzőleg említett y címkékre nincs is szükségünk. A felügyeletlen tanításos algoritmusok olyan adathalmazokat tapasztalnak, amelyek sok jellemzőt tartalmaznak, ezután pedig hasznos tulajdonságait tanulja meg az adathalmaz struktúrájának. Mivel nincsen

szükségünk címkézésre ezért nagyobb adathalmazokat sokkal egyszerűbb kreálni. Ez olyan módon is nagyon hasznos adathalmaz gyűjtés során, hogy kevesebb idő megy el az adatok címkézésével, ehelyett gyűjthetünk több adatot vagy finomítjuk a hálónak más komponenseit.

2.3.2. Mély generatív modell

A mély generatív modell (Deep generative model), egy osztálya a felügyeletlen tanításnak, amelynél adott tanítási adatokkal $p_{data}(x)$ képes generálni új mintákat $p_m(x)$ ugyan abból a valószínűség disztribúcióból. Így $p_m(x)$ hasonló lesz a $p_{data}(x)$ -hez. Néhány architektúra, a valószínűség disztribúció funkciót részletesen kiértékelik, néhány más pedig nem engedi hogy kiértékeljük a valószínűség disztribúció funkciót, de támogat más módszereket ennek a következtetésére, hasonló módon mint amikor példákat húzunk a disztribúcióból. Ez azért nagyon fontos mivel ilyenre nem képesek a felügyelt tanítási algoritmusok. Még egy előnye a mély generatív modelleknek, hogy a neurális hálózat amelyet generatív modellnek használunk, jelentősen kevesebb paraméterrel rendelkezik mint az adatok amiken tanítják őket.

Sok különböző feladatra képesek a generatív modellek, mint például képek felbontásának növelése, képek beszínezése, de legfőképpen a szakdolgozatoknak a nagy minőségű képek szintézise vagy generálása. Legtöbb esetben a képek szintézisére és az emberi arckép szintézisére, ugyan azokat a modelleket szokás használni.

A generatív modell ugyebár képes új mintákat generálni egy adathalmazból. A képek is adatok, minden egyes pixelt lehet reprezentálni egy szám értékkel. Tehát egy 128x128-as felbontású színes kép az valójában egy 128x128x3 méretű adattömb. A 3. dimenzió azok az RGB (piros, zöld, kék) csatornák értékei. Ha esetleg szürkeárnyaltos képeket vizsgálnánk akkor erre nem lenne szükségünk, de a szakdolgozatomban, a nagyobb kép minőség érdekében színes képekről lesz szó.

Nagyon sok fajta mély generatív modell létezik képek generálására, de 3 legelterjedtebb architektúra a VAE [8], PixelRNN [9] és a Generatív ellenséges hálózatok [10] (generative adversarial network).

2.4. Hasonló rendszerek és technológiái

2.4.1. Variational autoencoderek

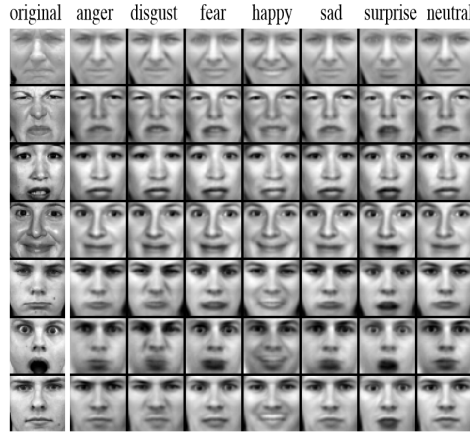
Ezek a Variational autoencoderek(VAE) hasonlóak az autoencoderekhez, de azért néhány részben mégis eltérőek.

Az autoencodereket nem adat generálásra használjuk mint a VAE-ket, hanem ezeket általában arra használjuk, hogy valamely jellemzőt (feature) z tudjon ábrázolni az x bemeneti adatból. Ha x bemeneti adatok emberi arcok, akkor z jellemzői lehetnek különböző attribútumok, esetleg milyen hangulata van a személynek (mosolyog, mérges, szomorú lásd Ábra 2-3). Az autoencoderek emellett felügyeletlen tanításos megközelítést használnak, tehát a bemeneti adatoknál nincsenek címkék. Ebben a rendszerben lesz egy encoderünk ami leképezi a jellemzőt valamilyen bemeneti adatból. Mivel a z az leképez az x adatokból ezért dimenzió csökkentésen megy át. Ezt azért csináljuk hogy csak a legfontosabb jellemzőket ragadja meg a bemeneti adatoknak. Nagyon sok ideig nem is mély neurális hálózatoknál használták hanem lineáris modelleknél, már sok ideje létezik ez a modell [11].

Egy karakterisztikája az autoencodereknek, hogy van egy decoder is amely segítségével lehetséges az eredeti bemeneti adat körülbelüli helyreállítása csak a jellemzők felhasználásával. Decodereknek is hasonló történelme van mint az encodereknek. Érdekes módon, ami után feltanítottuk a modellünket a decoderre már nincs is szükségünk, mivel csak az volt a célja hogy helyreállítsa az eredeti bemeneti adatot és ennek a segítségével képesek legyünk valamilyen veszteségfüggvényt definiálni. Ezután lehetőségünk van egy klasszifikációs modellt is hozzáadni, a decoder helyére, amely segít az encoder teljesítményének a finomítására. De ahogy eddig is említettem, ez nem képes új minta adatok generálásában és itt jön képbe a VAE hálózat.

A VAE hálózat kimeneti értéként egy valószínűség eloszlást adnak ki mindegyik lehetséges tulajdonságra a rejtett térben, ehhez képest az autoencoderek csak a lehetséges státusz tulajdonságot adja kimeneti adatként. A z jellemzők módosításával a VAE hálózat képes ugyan azt az arcot legenerálni csak különböző tulajdonságokkal, mint mérges, szomorú, stb.(Ábra 2-3)

A VAE hálózatok képesek implementálni egy egyenletes és állandó ábrázolását a rejtett térnek(latent space). Egy elég nagy problémába ütközhetünk az ilyen hálózatoknál mivel



2-3. ábra. A bal oldali oszlop egy minta a tesztelési adathalmazból, tőle jobbra pedig lehet látni a generált arcokat, különböző arckifejezésekre, amelyet a [12] papírban egy VAE hálózattal a TFD [13] adathalmazon készítettek el.

megoldhatatlan lesz az adat valószínűségének(data likelihood) a kiszámításában [8].

$$p_0(x) = \int p_0(z)p_0(x|z)dz$$

Nem vagyunk képesek kiszámítani vagy megkülönböztetni a peremeloszlást (marginal likelihood). Kiderült, hogy ez a probléma nem csak az adat valószínűségének a kiszámításában fordul elő hanem a $p_0(z|x)$ is megoldhatatlan. Emiatt nem vagyunk képesek egyértelműen optimalizálni a modellt. Egyik megoldása az hogy készítünk még egy encoder hálózatot ami hozzávetőlegesen képes kiszámítani a $p_0(z|x)$. Ennek a segítségével kitudjuk számítani az alsó korlátot az adat valószínűsége, ez már kiszámítható. Mivel kiszámítható ezért optimalizációnál maximalizálni kell a valószínűség alsó korlátját. Tanítás után elhagyhatjuk az encoder hálókat és képesek leszünk adat generálásra csak a decoder használatával.

A VAE hálózatokat képesek vagyunk feltanítani komplex képek generálására, például emberi arcokra. De, mint minden másik generatív modell implementációnak, vannak hátrányai. Mivel a valószínűség alsó korlátját maximalizálja ezért a képek elmosottak és kisebb minőségűek lesznek. Nagyon könnyen tanulnak pedig variációt a különböző arcok között. Összehasonlítva a GAN hálózatokhoz képest, a mai napig jelentős különbség van a minőségük között. A megoldás ahol a valószínűség alsó korlátját számítja ki, természetesen jobb mint ha nem lenne, de még sem jobb mint néhány más becslés módszer például a PixelCNN/PixelRNN-eknek.

2.4.2. Autoregressive generative model

Ezek a modellek a "fully-visible belief network" (FVBN) hálózatok családjába tartoznak. Az ilyen hálózatok a lánc-szabályt használják ahhoz, hogy kiszámítsák a többdimenziós valószínűségeloszlást amelyet a modellre $p_m(x)$ -el jelölünk. Ennek kiszámításához a feltételes eloszlást kell felírunk az összes pixelre amelynek a formulája [9]:

$$p_m(x) = \prod_1^{n^2} p_m(x_i | x_1, \dots, x_{i-1})$$

Itt a $p_m(x_i | x_1, \dots, x_{i-1})$ értéke megmutatja a valószínűségét az i -edik pixelnek az összes előtte lévő pixel alapján és ezt természetesen addig folytatjuk amíg átdolgozzuk az összes pixelt. Szakirodalmanként változik hogy n vagy n^2 darab iteráción megyünk keresztül, ez azon múlik hogy milyen módon definiáljuk az n -et. Én most úgy definiálom hogy egy $n \times n$ felbontású képnél n^2 lépést csinálunk és az első $p(x_1)$ mivel nincsen előtte lévő pixel amelyet meg kéne vizsgálni ezért kihozhatjuk a product elé:

$$p_m(x) = p_m(x_1) \times \prod_2^{n^2} p_m(x_i | x_1, \dots, x_{i-1})$$

Így hogy a formula már egyértelművé válik lehet látni hogy ez nagyon sok lépés lehet egy nagyobb méretű képnél. De emellett még csak egy grayscale képet dolgoztunk át, még hozzá kell rendelni 3 értéket az RGB szín-értékek miatt, minden egyes x_i értékhez. Tehát az eloszlást módosítanunk kell minden x_i értékre, ezt a következő módon valósítjuk meg [9]:

$$p_m(x_{i,R} | x < i) p_m(x_{i,G} | x < i, x_{i,R}) p_m(x_{i,B} | x < i, x_{i,R}, x_{i,G})$$

Ez azt fogja jelenteni hogy nem csak az ez előtti pixelek befolyásolják a valószínűséget hanem még ez az extra 3 szín csatorna is. A fully visible belief hálózatoknak van 2 elterjedt implementációjuk, a PixelRNN [9] és a PixelCNN [9].

PixelRNN a nevéből adódóan egy RNN azaz rekurrens neurális hálózattal van modellezve. Amikor generálunk egy képet a PixelRNN modellel akkor az egyik sarokban lévő első pixelt kezdjük el generálni, tipikusan a bal felső sarokból való kiindulás szokásos. Ezután szekvenciálisan generálunk tovább és diagonálisan haladunk előre. Mivel kihasználjuk az RNN architektúrát és egy LSTM modellt ezért képesek vagyunk az előző pixelek alapján generálni amely a lényege volt az FVBN hálózatoknak.

Hatalmas hátránya a PixelRNN hálózatoknak hogy a szekvenciális generálás az nagyon

lassú. Lehetett érezni a valószínűségeloszlás számításánál is, hogy ez nem pont egy gyors generálást fog eredményezni és ehhez hozzátevé még nem is vagyunk képesek párhuzamosítani a folyamatot mivel szekvenciális generálást használunk. Természetesen ez azt is fogja jelenteni hogy 128x128 méretű képek generálása felé nagyon sok ideig képtelen volt az architektúra mivel túl sok ideig tartana az egész folyamat.

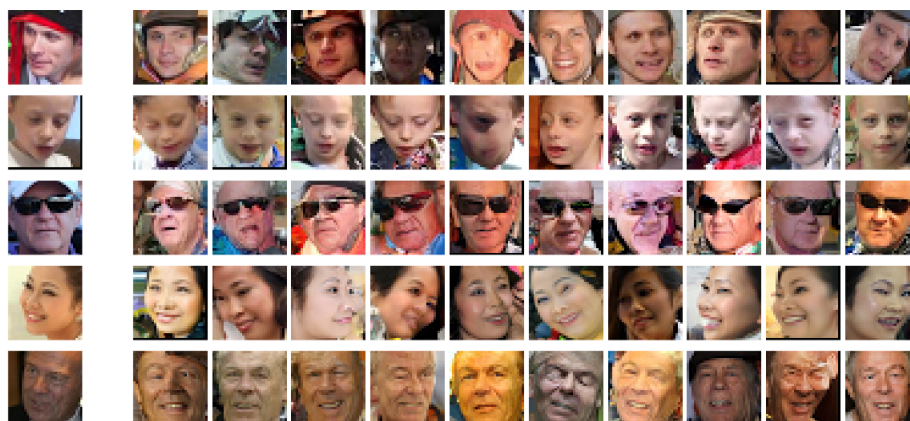
PixelCNN sok szempontból hasonló a PixelRNN architektúrához, de most nem egy RNN hanem egy CNN azaz konvolúciós neurális hálózat architektúrát alkalmaz. Ugyanúgy egy sarokból kezd el generálni és ugyan úgy szekvenciálisan generál. Egy előnye ennek az architektúrának a PixelRNN-hez képest hogy a tanítása az sokkal gyorsabb mivel azt képesek vagyunk párhuzamosítani. De általában jobban szokott teljesíteni a PixelRNN.

A PixelCNN++ [14] papírban még tovább fejlesztették ezt az architektúrát, gyorsabb tanítással, elhagyják a subpixel számítást az RGB szín csatornákra és egész pixelekkel dolgoznak. Downsampling-et használnak, hogy könnyebben észrevegyen a modell struktúrákat különböző méretekben és még dropout [15] szabályozást is elkezdtek használni. Habár egy nagy előnye a tanítás egyszerűsége, mivel a valószínűséget egyértelműen ki tudjuk számítani. De ez még mindig nem ideális, a neurális hálózatoknál egy nagyon fontos filozófia az lenne, hogy törekedjünk a gyors generálásra és a hosszabb tanításra.

Amikor arcképek generálásáról beszélünk akkor az autoregressive modellek nagyon jól működnek pixel szintű részletek generálására, de nem képesek igazán megragadni az emberi arcoknak a struktúráját. Ezt a jelenséget lehet látni a [16] papírban ahol egy conditional PixelCNN hálózatot betanítottak, hogy ugyan azt az arcot generálja le más pózokban (Ábra 2-4).

Az autoregressive generatív modelleket általában hibrid modellekben szokták használni. Ezek a hibrid modellek általában kombinálják az egyik autoregressive generatív modellt, egy másfajta generatív modellel. Egyik legjobban teljesítő hibrid modell a variational autoencoderek és az autoregressive modell kombinálása. Ebben a hibrid modellben a VAE hálózat feladata a főstruktúrával foglalkozik, az autoregressive modell feladata pedig a minőség javítása, ahogy említettem ebben nagyon jól működnek. A VAE hálózat nagyon jól képes globális információt megragadni, a PixelCNN/RNN pedig lokális információban teljesít jól. Ez a VAE hálózatoknak a problémáját is megoldja ahol csak az alsó korlátot volt képes megbecsülni, de a hibrid modelleknél képesek vagyunk a másik modellnek a becslés módját használni.

Egy hibrid modell megvalósítása a VQ-VAE-2 [17] amely képes gyönyörű state-of-the-



2-4. ábra. A bal oldali oszlop egy minta a tesztelési adathalmazból, tőle jobbra pedig lehet látni új képeket ugyan arról az arcról, csak más pozíciókból, amelyet a [16] papírban egy conditional PixelCNN generálta le.

art minőségű 1024x1024 emberi arcokat generálni(Ábra 2-5). Ennél jobb megvalósításra csak a GAN hálózatok képesek, habár nem rendelkeznek a GAN hálózatok problémáival, mint a mode collapse és a tanítás nehézsége.



2-5. ábra. VQ-VAE-2 [17] által generált 1024x1024 felbontású képek.

2.4.3. GAN

A generatív ellenséges hálózatok (generative adversarial networks), rövidítve GAN [10], egy teljesen másfajta megközelítése ezeknek a generatív modelleknek amit Ian Goodfellow, a generatív ellenséges hálózatoknak a keresztapja, a "game theoretic scenario" angol kifejezéssel definiálta. Ez a game theoretic scenario vagy approach magyarul annyit jelent hogy egy játékszerű megközelítés. Ebben a megközelítésben nem csak 1 modellünk van hanem 2 neurális hálózatot tartalmaz, a generátor hálózatot G (generator network) és a diszkriminátor hálózatot D (discriminator network), amelyek egymás ellen úgymond, versengenek [5].

A generátor hálózatnak, a nevéből adódóan is, minták készítése $x = g(z; 0^{(g)})$ a feladata amelyet továbbad majd a diszkriminátor hálózatnak. A generált adatokat általában random zajból kiindulva szokta elvégezni. Ennek a diszkriminátor hálózatnak a feladata pedig, a generátor által megkapott minta és a tanítási adatok között megkülönböztetni, hogy melyik adat jött a tanítási adatokból és melyik a generált. A döntést, hogy melyik a tanítási adat és melyik a generált, egy valószínűség érték alapján állapítja meg amit $d(x; 0^{(d)})$ határozza meg. A $d(x; 0^{(d)})$ az x éppen vizsgálat alatt álló adat, tehát a formula azt határozza meg hogy mekkora eséllyel lehet x egy tanítási adat inkább mint egy generátor által készített minta.

Az eredeti generatív ellenséges hálózat elképzelésben a tanítás egy zero-sum játék volt ahol egy funkció F meghatározza a diszkriminátornak a végeredményét és a generátor pedig megkapta $-F$ a saját végeredményének. Tanítás közben, mindkettő modell megpróbálja maximalizálni a saját végeredményét. A funkciónk pedig ahol z a generált adat [5]:

$$F = E_{x \sim p_{data}} \log D(x) + E_{x \sim p_{model}} \log(1 - D(G(z)))$$

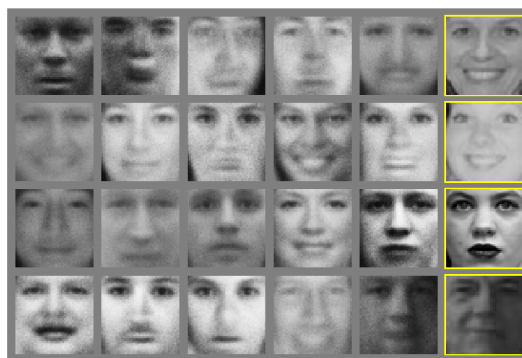
Ez a megközelítés megpróbálja betanítani a diszkriminátort, hogy képes legyen pontosan meghatározni, hogy melyik minta az igazi és melyik a generált. Eközben a generátor megpróbálja átverni a diszkriminátort, hogy elhiggye, hogy az általa generált minták az igaziak.

Egy nagyon érdekes karakterisztikája a generatív ellenséges hálózatoknak, hogy ha konvergenciát elértünk a két háló között, tehát a generátor által készített minták már nem megkülönböztethetőek, az eredeti tanítási adatoktól és a diszkriminátor mindig 50-50%-ot ad ki a klasszifikáció során, akkor elhagyható a diszkriminátor. Így a diszkriminátor használata nélkül képesek vagyunk minták generálására.

Sajnos ezt, implementáció során sokkal egyszerűbb elképzelni mint elérni. A generatív ellenséges hálózathoz a tanítása rendkívül nehéz és a mai napig kutatott. Sokszor a hálózat nem konvergens és emiatt a hálózat képes alul-tanulni (underfit). Ez a gépi tanulásban azt jelenti hogy a modell nem képes megragadni a különböző összefüggéseket, amelyek az adathalmazban vannak. Általában 2 modellnek a gradiens ereszkedésének a kiszámításánál, nem túl valószínű hogy elér egy teljesen kiegyensúlyozott státuszt, egy úgynevezett Nash-egyensúlyt. Volt ennek a megoldásnak egy alternatívája amit Goodfellow 2014-ben felfedezett amely nem használt zero-sum megoldást, de sajnos gyakorlatban nem segített a konvergenciában.

A legjobban teljesítő implementációban, nem használtak zero-sumot itt se, a generátor célja a diszkriminátornál annak a valószínűségét növelni hogy a diszkriminátor a rossz döntést hozza, ellentétben az eddigi implementációban ahol az volt a célja a generátornak, hogy csökkentse a valószínűségét annak hogy a diszkriminátor jó döntést hoz.

Érdemes megemlíteni hogy ez az új implementáció sem tette hihetetlenül stabilá ezeket a modelleket. Legtöbb esetben azért rendes architektúra választással és hiperparaméter optimalizációkkal, nagyon jól képes teljesíteni a generatív ellenséges hálózat. Egy rendszeren optimalizált modell képes volt tényleg olyan példák generálására amelyek nem csak memorizálták a modell disztribúciót és ténylegesen olyan arcokat generált amelyek nem voltak benne a tanítási adathalmazban (Ábra 2-6). Habár még csak grayscale arcképeket volt képes generálni és a képek felbontása 32x32-es volt, sikeresen random emberi arcokat készített. Sőt még a generálási sebesség is gyors, amely egy neurális hálózathoz nagyon fontos.



2-6. ábra. GAN modell által generált arcok a TFD [13] adathalmazon.[10]

2.4.4. Továbbfejlesztett GAN modellek

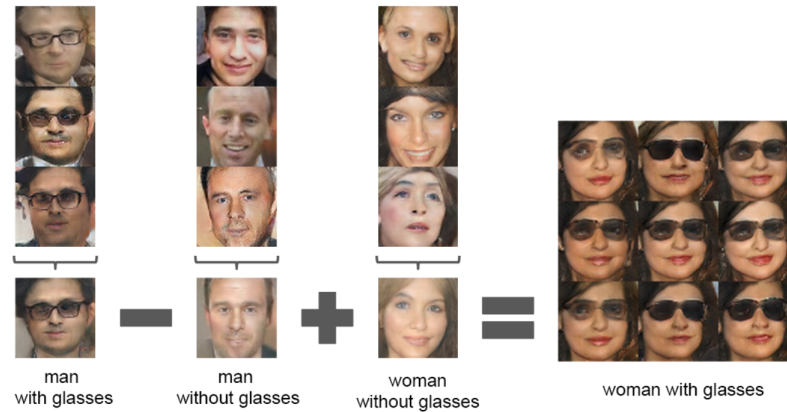
Volt néhány probléma a GAN hálózatokkal amelyet javítani kellett, vagy teljesen meg kellene változtatni. A 2 modell párhuzamos tanítása az nagyon instabil lehet, ha rossz hiperparaméterek állítunk be. Egy jobban funkcionáló veszteségfüggvényre is szükség van. A GANoknak tendenciája van egy úgynevezett mode collapse jelenségre, ahol a generátor túloptimalizál a diszkriminátorra és ilyen helyzetekben a diszkriminátor nem képes tanulni. Emiatt a generátor nem képes nagyon sok egyedi kép generálására. Az eltűnő gradiens probléma is gyakran megjelent a GAN hálózatokban. Ezeket a kihívásokat megpróbálták megoldani sok különböző architektúrával és megközelítéssel.

Az évek során nagyon sok fajta architektúra jelent meg, különféle használatra. Több 100 fajta GAN architektúrát javasoltak különböző kutatási tanulmányok, ezért is "GAN zoo" elnevezés lett kitalálva rá, amely kifejezi a bőségét a generatív ellenséges hálózat architektúrák számának. Néhány híresebb GAN modellt fogok beszélni amelyek vagy hozzájárultak a GAN hálózatoknak a stabilitásában vagy valamilyen módon hozzájárultak az emberi arckép generálásának minőségéhez.

Egyik legfőbb ötlet az a konvolúciós neurális hálózatoknak (CNN) integrálása a GAN hálózatokba, hogy egy még mélyebb modellt legyenek képesek felépíteni, de Mehdi Mirzának [18] egy másik megközelítése volt. Kimutatták hogy a generatív ellenséges hálózatnak a tanítási problémáját letudjuk bontani sok kisebb részre. Itt megint megjelenik az y címke, ez lehet akármilyen fajta információ, osztály címkék vagy valamilyen modalitás. Ez arckép generálásnál lehetett valamilyen arc attribútum vagy esetleg face landmark is. Ebben az architektúrában létezik még egy bemeneti réteg, amely a generátorba és a diszkriminátorba is fog etetni y adatot, ezt a módszert kondicionálásnak nevezzük. A generátornál a random zaj meg az y kombinálva lesz egy rétegbe. Egyértelműen jobb eredményeket mutatott mint egy standard generatív ellenséges hálózat, de azért az extra munka ami belemerne az adatok címkézésébe, egy adathalmaz kreálásánál problémás tud lenni a projekt mérete alapján.

A mély konvolúciós generatív ellenséges hálózatok DCGAN [19] olyan generatív ellenséges hálózatok amelyek sikeresen implementálták a konvolúciós neurális hálózatokat a generátor és a diszkriminátor hálózatba. A DCGAN drasztikusan növelte a minőségét a generált mintáknak. Az eredeti papírban [19] be is mutatták, hogy vektor művelet végzésével képesek a generált adatokat módosítani. Az összes olyan z vektort, amelyek a bemeneti értékei a generátornak, összeadták és átlagozták amelyek tartalmaztak egy fér-

fit szemüveggel. Ebből a z vektorból pedig kivonták az összes "férfi szemüveg nélkül" vektort és hozzáadták a "nő szemüveg nélkül" vektort és ezt az új Y vektort bemenetként áadták a generátornak. A generátor ezután pedig olyan arcképeket generált amelyen csak nők voltak szemüveggel(Ábra 2-7).

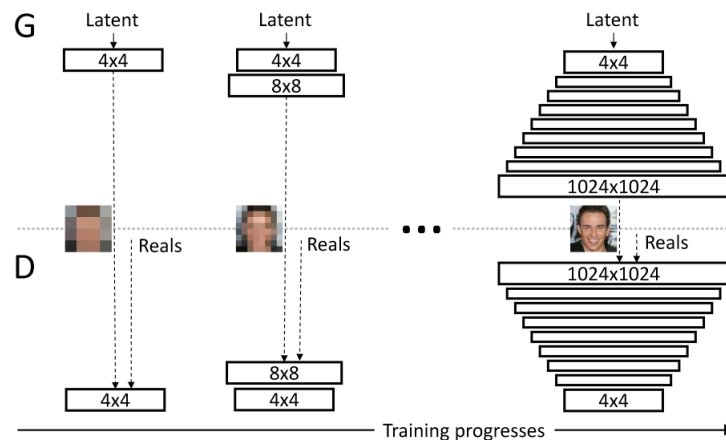


2-7. ábra. DCGAN modell által generált arcokon végzett vektor műveletek az eredeti DCGAN cikkben [19].

A DCGAN modell nagyon sokat javított a minőségén és a stabilitásán is a GAN modell-hez képest. De még mindig képes volt a mode collapse jelenségre, ezért még mindig nem volt teljesen stabil.

A GAN modelleknek szükségük van egy új veszteségfüggvényre, hogy stabilabban tudjon a modell tanulni és hogy a mode collapse jelenség ne forduljon elő. Eredetileg a Jensen-Shannon divergenciát használta Ian Goodfellow [10] a távolsági metrikára amely nem segít a jó megoldás konvergenciájára. A WGAN [20] pont ezeket a problémákat oldja meg. Az úgynevezett wasserstein vagy EM távolságot implementálja a JS divergencia helyett. Ez a változás képes volt egy jobban működő veszteségfüggvényt implementálni, amely egy sokkal stabilabb modellt fog megalkotni. Az eltűnő gradiens problémát is megoldotta és a mode collapse jelenség is eltűnt. De azért még mindig néha kicsi minőségű mintákat generált vagy nem volt képes rendesen konvergálni. A diszkriminátorra pedig szükség volt a k -Lipschitz-tulajdonságot alkalmazni, amelyet súlykorlátozással értek el. Kiderült, hogy a súlykorlátozás optimalizációs nehézségekhez vezethet ezért is a WGAN-GP [21] papírban egy alternatívát találtak ki erre a módszerre amely sokkal stabilabb megoldás lett. Ahelyett hogy a súlyokat korlátoznák, inkább bünteti a diszkriminátor gradiensének normáját, a bemenetéhez képest. A gradiens büntetés(gradients penalty) nagyon jól teljesített, olyan körülmények között is ahol nagyon kevés hiperparaméter optimalizáláson esett át a modell.

A WGAN-GP által felvetett javítások képesek voltak a GAN modelleket sokkal stabilabbá tenni. Ezután, hogy elértünk egy stabilabb hálózatot, a fő cél a képek minőségének növelése és a képek közötti variációnak a bővítése. Eddig a pontig főleg csak 64×64 felbontású képek generálására voltak képesek a hálózatok. Hatalmas ugrás volt minőségben és még stabilitásban is, a Progressive Growing of GAN [22] papírban leírt architektúra, amely képes volt 1024×1024 felbontású képek generálására. A legfőbb változás amelyet ebben a papírban felvetettek, az egy új tanítási metodológia GAN modellekre, ahol alacsony minőségű képek generálásával kezdenek és folyamatosan növelik a felbontást, tanítás közben. Ezt úgy érik el hogy rétegeket adnak a generátor és a diszkriminátor hálózatokhoz a tanítás folyamán (Ábra 2-8). Ha egy réteget hozzáadunk egy modellhez akkor ez egy "sudden shock" lehet a rendszeren feltanított kisebb felbontású rétegeknek. Emiatt a jelenség miatt inkább szabályosan beleolvasztják a hálózatba.



2-8. ábra. A folyamatosan növekvő GAN avagy a Progressive Growing of GAN [22] modell rétegeinek a bővítése, amelyet tanítás folyamán végez.

A felbontásnak az inkrementális növelése lehetővé teszi a tanítás elején, hogy a modell képes legyen megtanulni a nagyobb globális struktúráit a tanítási adathalmaznak, és később pedig a lokális részleteknek a finomításán. Ez a folyamat stabilabbá is teszi a modellt, mivel nem annyira nagy feladat megoldására van szükség mint az egész képnek a feltérképezése rejtett vektorokból. Még egy előnye a tanítási időnek a lecsökkentése. A folyamatosan növekvő GAN modelleknek a legtöbb iterációi kisebb felbontáson történnek, amely a papír [22] szerint ugyan olyan minőségű képekre, $2-6 \times$ gyorsabb is lehet. Emellett a modell bevezet még több fontos implementációt, mint a "minibatch standard deviation" réteget amely növeli a variációt a generált minták között és a diszkriminátorba helyeznek el. Egy másik implementáció a kiegyenlített tanítási ráta és a pixel alapú jellemző vektorok normalizálása amelyek segítenek a tanítás stabilizálásában. Külön ké-

szítettek egy adathalmazt a CelebA-HQ, amelyet a CelebA [23] adathalmaz alapján készítettek el. A CelebA-HQ adathalmaz 1024x1024 felbontású és ezt használták fel a modell tanítására. Modell által generált képeket lehet látni a 2-9. ábrán.



2-9. ábra. A folyamatosan növekvő GAN avagy a Progressive Growing of GAN [22] modell által generált arcképek.

A Progressive Growing of GAN sokkal stabilabb volt és state of the art képek generálására volt képes, de azért a generátor még mindig egy gyenge pontja volt a hálózatnak. A StyleGAN [24] papírban továbbfejlesztik a Progressive Growing of GAN modellt, ahol csak a generátort módosították. A diszkriminátor és a veszteségi függvény ugyan olyan állapotban maradt. A módosított generátor a stílus átvitel [25] alapján lett elkészítve. Ennek a módosításával képesek a kép generálásnak a folyamatát kontrollálni. A generátor fel lett bontva 2 hálózattá, a mapper hálózatra és a szintézis hálózatra. A mapper hálózatnak bemeneti adatként érkeznek a normalizált rejtett z kódok, amelyet leképez egy köztes W rejtett térbe, ez a W tér fogja kontrollálni a generátornak mindegyik konvolúciós rétegét, adaptive instance normalization (AdaIN) segítségével, amely az instance normalization [26] alapján lett elkészítve. A szintézis hálózat gyakorlatilag ugyan az mint a Progressive Growing of GAN, legfőbb eltérései, hogy egy konstans értéket kap az első rétegének a bemeneteként, a z helyett és a w integrálódik a konvolúciós rétegeibe.

Arckép szintézis szempontból, a képek minőségének javításán kívül, a StyleGAN modellben a generátor zaj képeket kap bemenetként amely lehetővé teszi hogy a generátor képes legyen különböző fajta sztochasztikus variációt generálni az arcképekre. A modell képes megtanulni a pontos elhelyezkedését a hajnak, szeplőknek és különböző bőrpórusoknak, a generátor kapacitásának az elvesztegetése nélkül. Emellett még egy teljesen új adathalmazt használtak amelyet Faces-HQ (FFHQ) neveztek el, ez az adathalmaz sokkal

több variációt tartalmaz kor, rassz és a kép hátterei között mint a CelebA-HQ adathalmaz amelyen a Progressive Growing of GAN modellre alkalmazták.



2-10. ábra. A StyleGAN [24] modell által generált arcképek, az FFHQ adathalmazon.

Összességében arc szintézis feladatra a GAN modellek teljesítenek legjobban. A legjobban teljesítő modell a StyleGAN2 [27] papírban a StyleGAN módosításával érték el. Analizálták ebben a papírban a StyleGAN modellnek a hiányosságait és gyenge pontjait. A StyleGAN modellnek volt egy jelensége ahol apró úgynevezett artifactokat lehetett észrevenni a generált képeken, amely az AdaIN normalizálás miatt történt. A StyleGAN2 papírban az architektúrát is újratervezték amely nem használt progresszívan növekvő generátor és diszkriminátort. Néhány új szabályozási technikát is használnak benne mint a "lazy regularization" és a "path length regularization". A papírban egy kapacitás problémát is azonosítottak, ahol kiderült, hogy nagyobb modellek, további minőséget képesek hozzáadni a generált képekhez.



2-11. ábra. A StyleGAN2 [27] modell által generált arckép, az FFHQ adathalmazon.

2.5. Aktivációs függvények

Egy mesterséges neuronba beérkező adatokat $x_i \in \mathbb{R}$, multiplikáljuk a súlyainkkal $w_i \in \mathbb{R}$, ehhez az értékhez hozzá adunk valamilyen eltolást(bias) $b \in \mathbb{R}$ és ez átmegy egy f aktivációs függvényen:

$$f\left(\sum_{1 \leq i \leq n} w_i x_i + b\right) \quad \text{ahol } i \in [1, \dots, n]$$

Ez az alapja a mesterséges neuronok működésének. Ezek a f aktivációs függvények, differenciálható funkciók, amelyeknek az a célja hogy a hálózatunkhoz adjanak nemlinearitást, így képesek lesznek sokkal bonyolultabb mintákat leírni mint egy egyenes, lineáris vonal. Régebben machine learning problémákra volt használva lineáris funkció is, mivel nagyon egyszerű volt maga a funkció és a kiszámítása is, de nem ideális komplex kihívásokra.

Mivel a neuronban szorzásokat végzünk el ami miatt sokszor magas lehet a számértékünk, az aktivációs függvény ezeket a nagyobb értékeket általában összenyomja 1 és 0 vagy esetleg -1 közé. Nagyon fontos része egy neurális hálózatnak, ezért az is nagyon fontos hogy a mi modellünkhöz kedvező funkciót válasszunk ki.

2.5.1. Sigmoid és hasonló függvények

A sigmoid függvény σ , az egyik legelterjedtebb aktivációs függvény amely a számokat összenyomja a $[0, 1]$ skálára.

$$\sigma(x) = \frac{1}{1 + e^x}$$

Habár régen nagyon sokszor használt volt, elég sok problémába ütközhetünk a használásánál. A sigmoid nagyon híres arról hogy az eltűnő gradiens(vanishing gradient) problémát okozhatja. Ez tanítási időben történhet meg és a neuronok telítettek(saturated) lesznek és nem képesek utána tanulni, vagy csak nagyon alacsony tanítási rátával. Maga a funkció kiszámítása is relatív lassú a e^x miatt.

A funkció kimenete mindig pozitív ez azt fogja jelenteni hogy a funkciónak nem nulla a középpontja. Emiatt a súlyok frissítése sokkal nehezebb lesz mivel ha egy gradiens 0, akkor nem egyértelmű hogy milyen irányba kell frissíteni [5]. Viszont ha valószínűséget kell számolnunk akkor a sigmoid elég jó választás mivel ott nincsen szükségünk negatív számokra. Főleg akkor használt amikor csak két osztályos klasszifikációra van szükségünk. Ezért egy GAN hálózatnál a mai napig használatban van a diszkriminátornál.

Sok másik aktivációs függvény a szigmoidon alapul mint például a $\tanh$ [28] és a softmax [29]. A $\tanh(x)$ skálája $[-1, 1]$ és a középpontja nullán helyezkedik el ami már egy nagy előny a frissítés lépésénél. De sok ugyan olyan problémával rendelkezik mint a szigmoid, e^x számításigényes, ugyan úgy hajlamos az eltűnő gradiens problémára.

A softmax függvények legtöbbször egy osztályozónak a kimenetére használják, hogy valószínűség disztribúciót mutasson több osztályos klasszifikációkra. A funkció formulájában Z a bemeneti vektor, a z_i pedig a kimeneti vektor, k pedig az összes osztálynak a száma amely benne van a többosztályos osztályozóban. Olyan $i \in [1, \dots, k]$ és $j \in [1, \dots, k]$ számokra nézve a formula:

$$\text{Softmax}(Z)_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

Ez igazából egy fajta $\max$ funkció, ahol nem a legmagasabb számértéket választja ki hanem a disztribúció legnagyobb részének a maximál elemét. De ugyan azokkal a problémákkal rendelkezik mint a szigmoid függvény.

2.5.2. ReLU

A ReLU vagy angol megfelelőjén Rectified Linear Unit aktivációs függvény, már a nevéből adódóan is félig helyesbített(Rectified), akkor amikor a beérkező adat negatív akkor a funkció kinullázza és emiatt lesz félig helyesbített, alulról természetesen. Ha az érték nulla vagy nullánál nagyobb akkor változatlanul átmegy a funkción.

$$R(x) = \max(0, x)$$

A ReLU nem rendelkezik a szigmoid problémájával ahol telítettek lesznek a neuronok, így nem kell foglalkoznunk az eltűnő gradiens problémával, de ez természetesen csak a pozitív régióban igaz. A negatív régióban ugyan úgy megöli a gradienst ami visszaterjesztés(backpropagation) közbe azt fogja jelenteni hogy nem lesz teljesen optimális a frissítési lépés. A funkció kiszámítása sokkal egyszerűbb mivel nem tartalmaz kalkulációkat mint a szigmoidban a e^x . Emellett még sok neuront nem aktivál egyszerre mint más funkciók, mivel kinullázza őket, ezért számítása egyszerűbb. A szigmoid σ meg $\tanh(x)$ -hez képest be is lett bizonyítva hogy gyorsabban konvergál, akár még hatszor gyorsabban is gyakorlatban. Viszont lehet látni hogy ennek a funkciónak sem a középpontja a 0 és ezért egy zig-zag utat fog csinálni optimalizáció közbe, amely több időbe kerülhet [5]. Rossz inicializációval az is lehetséges hogy a ReLU-nk egy úgy nevezett halott ReLU lesz ami sohase fog aktiválódni és ebből következően frissíteni se.

2.5.3. ReLU alternatívák

A halott ReLU probléma ellen van néhány intézkedés ami segít a helyzeten, egyik ilyen a neuronok inicializálásánál az eltolásnak egy pozitív számot megadni, de még emellett is inkább nulla eltolással szokás inicializálni. Egy sokkal konkrétabb megoldás a Leaky ReLU [30] amely megpróbálja a halott ReLU problémát megoldani. Ahelyett hogy lenul-lázzuk a negatív értékeket, egy nagyon kicsi értékkel beszorozzuk őket. Ennek az értéke változó, de 0.01 az elterjedt.

$$LeakyR(x) = \max(0.01x, x)$$

Ennél a funkciónál az eltűnő gradiens probléma már a negatív régióban sem fordul elő és segít hogy a hatásköre nagyobb legyen, mivel nem csak pozitív számaink lesznek. De ugyan úgy nem nulla a középpontja. Ezek a legelterjedtebbek de van milliányi más, gyakorlatban ReLU-t érdemes szinte mindig használni, főleg a tervezés legelején. Még így is hogy ennyi fajta aktivációs funkció van, egyik se volt képes teljesen lecserélni, mivel inkonzisztensek a jól bevált ReLU-hoz képest [31]. Később érdemes kipróbálni a Leaky ReLU, ELU és a Maxout aktivációs funkciókat, hogy milyen módon befolyásolja a hálónkat. Nem árt megemlíteni, hogy még olyan esetekben is amikor jobb lenne ReLU-nak az alternatívája, elég apró különbségek szoktak lenni, ezért is nem annyira használtak az alternatívák. Hiába kapunk egy nagyon kicsit jobb eredményt, ha közben a rendszerünk kicsit inkonzisztensebb lett, nem egyértelmű, hogy megéri a váltás.

2.6. Paraméter inicializálás

Egy megfelelő inicializálása a súlyoknak egy nagyon fontos lépése a neurális hálózatok felépítésének. Hogy rendes konvergenciát érjen el a modellünk, a súlyoknak az inicializálása kritikus része a modell felépítésének. A fő célja a súlyok inicializálásának az, hogy megakadályozza a rétegek aktivációs függvények kimeneteinek a felrobbanását vagy eltűnését [32]. Ha akármelyik megtörténik akkor a gradiensnek, úgy is mondhatjuk, hogy haszontalanok lesznek a modell tanításában, mivel vagy túl nagyok lesznek vagy túl kicsik. Legjobb esetben képesek lesznek a konvergenciára, de csak nagyon lassan, legrosszabb esetben pedig semmilyen mértékben sem fognak konvergálni.

Ezek az inicializációs problémák előfordul, hogy egy néhány réteges modellnél nem

annyira észrevehető, de minél több réteggel rendelkezik egy modell annál nagyobb problémát jelenthet egy rosszul inicializált modell. Mivel egyfolytában mátrix szorzásokat végzünk a rétegek között, egyre nagyobb és nagyobb probléma lesz belőle. Találni kell egy olyan stratégiát amely konzisztensen képes felkészíteni a súlyainkat hogy rendesen képesek legyenek konvergálni egy adott aktivációs függvényre.

A legalapabb stratégia a random inicializálás, de ez természetesen a felrobbanó és eltűnő gradiens problémát fogja okozni, ennél kicsit komplexebb megoldásra van szükség. Valahogy szükségünk lenne egy normalizált inicializálásra és pont ezt akarták elérni a normalizált inicializációval vagy mostanság jobban ismert a Xavier vagy Glorot inicializáció néven [33]. A Xavier inicializációban a rétegeknek a súlyait beállítja egy random értékre amely ennek a formulának a disztribúciójából jön, ahol $rand(-1, 1)$ egy random számértéket ad vissza $[-1, 1]$ között:

$$rand(-1, 1) * (\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}})$$

Ez a stratégia fantasztikusan működött nem lineáris aktivációs függvényeken mint például $\tanh(x)$ függvényen. Ki is mutatták hogy sokkal gyorsabban konvergál a modell. De ez csak olyan aktivációs függvényeknél működött ahol az értékünk $[-1, 1]$ között van. De a Xavier inicializáció nem működött annyira jól olyan függvényeknél mint a ReLU [32], amely pont nagyon fontos a GAN hálózatoknak. A deriválása azon alapszik hogy az aktivációs függvény az lineáris, de ez nem igaz ReLURA.

Szükségünk van egy olyan inicializálási stratégiára, amelyek ReLUkkal is működnek. Kaiming [34] demonstrálta a Kaiming vagy He inicializációt, amely képes nagyon mély modelleknél is konvergálni. Ennél a megoldásnál az eltolások(bias) inicializálása nulla értékű lesz és a súlyok inicializálása pedig a következő formula lesz, ahol n a beérkező egységeknek a száma lesz, $rand(n, m)$ pedig egy random disztribúció:

$$rand(n, m) * \frac{\sqrt{2}}{\sqrt{n}}$$

2.7. Veszteségfüggvények

Amikor veszteség függvény vagy néha veszteség funkcióról beszélünk akkor általában egy olyan értékre gondolunk amely számszerűsíti azt, hogy a neurális háló modellünk mennyire ad jó eredményeket. Ez általában úgy történik hogy a neurális hálózatunknak a kapott és az elvárt kimenetén valamilyen műveletet végzünk. Ez a különbség minél kisebb

annál "jobban" teljesít a hálózat. Természetesen ez gyakorlatban nem ennyire egyszerű, sokszor előfordulhat komplex modelleknél, hogy 2 modell közül, a rosszabb veszteség függvény eredményekkel rendelkező modell, jobb minőségű eredményeket ad vissza.

Egy nagyon fontos építőelem egy neurális hálónak az elkészítésében az, hogy milyen veszteségi függvényt választunk. Szerencsénkre a függvények, amelyekre szükségünk van egy ilyen neurális hálózat felépítésénél, szinte ugyan azok mint más gépi tanulási feladatoknál. Neurális hálónál létezik egy eredeti veszteségfüggvény(primary loss function) és egy teljes veszteségfüggvény (full loss function) amely kombinálja az eredeti függvényünket valamilyen szabályozási technikával. Ebben a szekcióban csak egy olyan eredeti veszteségi függvényről beszélek amely nem tartalmaz szabályozást, ezt majd egy későbbi szekcióban tárgyalom.

2.7.1. Veszteségfüggvény optimalizálása

Legtöbb algoritmusban egy fontos lépés az optimalizáció de ez még annál is igazabb egy ilyen deep learning algoritmusnál. Ezeknek az algoritmusoknak a célja az valamilyen funkcionak $f(x)$ vagy a minimalizálása vagy a maximalizálása, úgy hogy módosítja a x értékét. Most nekünk szükségünk van valamilyen olyan optimalizációs algoritmusra amely minimalizálná számunkra a veszteség függvényt. Az értékek amelyeket pedig módosítani kellene nekünk azok a W súlyaink lennének, amelyeknek a finomhangolásával minél kisebb veszteségfüggvényre törekedünk. Neurális hálózatoknál pedig annyira sok ilyen paraméterünk(súly és bias) értékünk van, hogy muszáj valami optimális módszert találni megfelelő értékek kiszámolásához.

Egyik stratégia amely papíron létezik de semmilyen szituációban sem hasznos a random keresés lenne, ahol random értékeket választunk ki a súlyunkra és kiszámoljuk ezekre az értékekre a veszteséget. Ha ez a veszteség érték "jobb" vagyis alacsonyabb mint az eddig lementett veszteség értékünk akkor lementjük az értéket és ezt megint végrehajtjuk adott iteráción keresztül. Ez egy szörnyű algoritmus, de még mindig jobb annál, mint ha semmilyen algoritmust se használnánk.

Egy sokkal elterjedtebb stratégiában az úgynevezett gradiens ereszkedés módszerben (gradient descent), mindig arra törekszünk, hogy az értékeket csak egy kis lépéssel hozzuk közelebb a jó értékhez. Ezt egy iterációs módszernek nevezzük. Úgy is leírható mintha egy domb tetején lennénk és leszeretnénk menni a dombról. Akkor nyilvánvalóan lefele

fogunk elindulni mert le akarunk érni. Ez nem azt fogja jelenteni, hogy ha lefele megyünk akkor a legalacsonyabb globális pontra érkezünk, de belefogunk érni egy lokális minimumra. Globális minimumot rendkívül nehéz megtalálni. Azt hogy merre van lefele úgy kapjuk meg hogy kiszámoljuk a gradienst $\nabla C(x, y)$ amely megadja azt az irányt ahol a legmeredekebb a funkció. Ha vesszük a negatívját a gradiensnek akkor megkapjuk azt az irányt amerre legjobban csökken a funkció. Ebből nem csak azt tudjuk meg, hogy milyen irányba hanem azt is, hogy mennyire nagy mértékben. Tehát már minden készen áll, ennek a stratégiának a használásához, mivel ez a módszer nem változik akkor se ha 12 vagy 12000 paraméterünk van, természetesen csak olyan módon, hogy több számítási időbe kerül az elvégzése.

2.8. Szabályozás

Egy fő probléma gépi tanulásban az, hogy milyen módon építünk fel egy algoritmust ami képes ideálisan teljesíteni, nem csak olyan bemenetekre amely segítségével fel lett tanítva, hanem teljesen új bemeneti adatokon. Irdatlan mennyiségű olyan felépített algoritmus létezik, amely kitűnő eredményeket képes produkálni a tanítási adatokon, de ha új adatokra teszteljük akkor nem tudja megragadni ugyan azt a teljesítményt. Sok stratégia létezik a gépi tanulásban a tesztelési hibák redukálásához, habár ezek sokszor a tanítási hibákat viszont növelik. Az itt leírt stratégiák azok amelyeket együttesen a szabályozásnak [35] (regularization) definiálunk.

Gépi tanulásból adódóan számtalan fajta szabályozás létezik, melyet képesek vagyunk felhasználni a mély tanulásban. Ahogy már ezt említettem sok típusú stratégia létezik, néhány korlátozásokat rak rá a modellünkre, olyanok amelyek megszorításokat raknak valamilyen paraméter értékekre. Innen is lehet érzékelni, hogy ezeket nagyon óvatosan kell kiválasztani. Igen, ezek képesek javítani a modell teljesítményén, de emellett ott van a kockázat is, hogy olyan megszorítást rakunk a modellre ami kicsit túlságosan is megszorítja. Innen tökéletesen fellevezethető 2 gépi tanulási probléma az alul-tanulás (underfitting) és az túlilleszkedés (overfitting).

2.8.1. Alul-tanulás és túlilleszkedés

Általában 2 fő tényező szokta megállapítani, hogy mennyire megfelelően működik egy gépi tanulós algoritmusunk. Egyik ebből az hogy mennyire képes az algoritmusunk a

tanítási hibát minél alacsonyabbá tenni. Mindig törekednünk kell erre, hogy az algoritmus minél optimálisabb legyen. A másik pedig az, hogy az algoritmusunk mennyire képes a tesztelési hibát minél közelebb hozni a tanítási hibához. Ha nagy különbség van a kettő között, akkor érdemes az algoritmusunkat megvizsgálni.

A 2 fő tényező azért is nélkülözhetetlenek mivel ezek konkrétan az alul-tanulás és a túlilleszkedésnek a definíciói. Akkor beszélünk alul-tanulásról amikor a modellünk nem képes elfogadhatóan alacsony tanítási hiba értéket elérni. Ez akkor szokott előfordulni ha a modell vagy adott paraméterek nem eléggé komplexek. Túlilleszkedésről pedig akkor beszélünk ha túl nagy a különbség a tanítási és a tesztelési hiba értékek között. Ezekben az esetekben a modellünk nem volt képes optimálisan hozzáilleszkedni az adatokhoz.

2.8.2. Dropout

A dropout [15] szabályozás, számítástechnikailag olcsó de erős módszert szolgáltat sok különböző felépítésű modelleszaladokra. A dropout olyan módon működik hogy egy teljes csomópontot eltüntet egy hálózatból. Ezt úgy éri el, hogy a kimenetét egy ilyen egységnek beszorozza nullával és akkor effektíven el is tűnt a legtöbb modern hálózatból. Ez az érték lehet természetesen más értékű is, ha a hálózatunknak más kellene, változtatható az érték. A dropout kiszámítása már így is láthatóan nagyon kevés kiszámítási időbe telik, mivel csak nullával szorzás. Ezután hogy csomópontokat eltávolít, az új alhálózatok együttesét tanítja.

Másik jelentős előnye a dropout szabályozásnak, hogy nincsen semmilyen korlátja vagy előfeltétele, hogy implementálható legyen egy modellbe. Még egy másik előnye, hogy lehet használni a sztochasztikus gradiens ereszkedés módszerrel együtt, ami az egyik legelterjedtebb módszer optimalizálásra. Célszerű megjegyezni, hogy más szabályozási stratégiának aminek hasonló hatásossága van, meglepően több számítási erőbe kerül és komolyabb korlátokkal rendelkezik a modell architektúrájára nézve [15].

Mivel dropout egy szabályozási technika ezért mégiscsak limitálja valamilyen módon a modellnek az effektív kapacitását. Kompenzálás képpen a modell méretét érdemes megnövelni. Ez viszont tényleg lehet egy nagy negatívuma a dropoutnak, ha nem rendelkezünk végtelen számítási erővel, ez egy nagy negatívum lehet. Emellett, hatalmas adathalmazoknál, a szabályozás elég csekély csökkentésre képes az általánosítási hibánál. Ezeknél az eseteknél, nem biztos hogy megéri az ilyen modellek használata.

2.8.3. Kötegelt normalizálás

Mély neurális hálózatok tanításánál eléggé oda kell figyelnünk a paraméter inicializációra, meg hiperparaméterek rendes beállítására, mint a tanítási ráta. Erre azért van szükségünk mivel nagyon mély modellek rendelkeznek számos funkcióval és rétegekkel. A gradiensnek az a dolga hogy meghatározza milyen módon frissítsük a paramétereket. Eddig minden rendben van de ez csak akkor működne ha nem változnának a rétegeink. Gyakorlatban minden réteget egyidejűleg frissítünk és ez elvezethet minket sok váratlan jelenségre. Az alacsony tanítási ráta és a paraméter inicializálás, lelassítja a tanításunknak a lehetséges sebességét. Ezt a jelenséget, angolul internal covariate shiftnek nevezzük [36] és a kötegelt normalizálás(batch normalization) pont ezzel a problémával próbál megbirkózni.

A kötegelt normalizálás ezt úgy éri el, hogy mindegyik egység bemenetét mean and unit variance[nem találtam magyar fordítás].

A kötegelt normalizálás megfelelő alkalmazásánál, használhatunk magasabb tanítási rátákat mint amit eddig megengedhettünk a modellnek. Hasonló lesz a helyzetünk a paraméter inicializációra. Ki is lett mutatva, hogy néhány helyzetben még a dropout szabályozásra se lesz szükségünk [36]. A kötegelt normalizálás stratégia nem pont egy szabályozási technika de hasonló effektjei is lehetnek. Mély modelleknél segít a gradiens áramlásán is. Ez nem minden esetben igaz, de DCGAN architektúrára ki lett mutatva hogy egy nagyon fontos építő komponense [19]. A normalizálás nagyon fontos lett, hogy a modell egyáltalán elkezdjen tanulni. De azért nem szabad ezt a technikát az összes rétegre használni, mivel túl instabil lesz a modell. Ezt úgy oldották meg, hogy mind 2 modell használja mindegyik rétegen kivéve a generátor kimeneti és a diszkriminátor bemeneti rétegeire.

2017-ben még tovább lett fejlesztve a kötegelt normalizálás, úgynevezett Batch Renormalization [37] nevet kapta meg, amely ugyan azokkal a pozitívumokkal rendelkezik mint a kötegelt normalizálás, de most már kisebb kötegekre(mini-batch) is jó eredményeket képes produkálni.

2.9. Hiperparaméterek optimalizálása

A hiperparaméter egy olyan paraméter amelynek értéke képes a tanítási folyamatot vezérelni. Minden gépi tanulós rendszer rendelkezik hiperparaméterekkel és ezeknek az optimális beállítása nagyon drasztikus különbségeket jelenthet. Főleg a mély neurális

hálózatoknál ahol létfontosságúan függnék a különféle hiperparaméterektől, amelyek felépítik az architektúrát, szabályozási módszert meg az optimalizációt. 2 alap szemléletmód létezik a hiperparaméterek kiválasztására. Vagy automatikusan választjuk ki a hiperparamétereinket, valamilyen optimalizáló algoritmussal vagy manuális választjuk ki.

Ha manuálisan szeretnénk kiválasztani ezeket a hiperparamétereket, akkor a modellünknek és a különböző hiperparamétereknek egy nagyon magas szintű felfogására van szükségünk. Az automatikus hiperparaméter kiválasztáshoz pedig nincsen szükségünk egy nagyon magas szintű megértésre, így a tervezés során egyszerűbbé teszi a dolgunkat. Viszont ezek az algoritmusok általában számítás igényesek.

Szóval a legfőbb hiperparaméterek amelyeket optimalizálhatunk, azok a hálózatunk architektúrája, a tanítási rátát, tanítási rátának az elhalását (learning rate decay) és a szabályozásokat, például a dropoutnak az erősségét. Ezek közül a tanítási ráta a legfontosabb[1] mivel ez sokkal erősebben befolyásolja a modellünk effektív kapacitását, mint más hiperparaméterek. Más hiperparamétereknek a változtatásához nagyon szorosan kell figyelni a tesztelési meg a tanítási hibákat is, hogy megállapítsuk vajon a modellünk alul-tanult vagy túlilleszkedik és ezután módosítsuk megfelelően. Ha a hibánk a tanítási adathalmazon magasabb mint a cél hiba rátánk akkor egyetlen lehetőségünk a modell kapacitásának a növelése.

Olyan modell amely sok hiperparaméterrel rendelkezik, mindig érdemes egyszerre csak 1 vagy 2 hiperparamétert optimalizálni. Egyik oka az, hogy minél többet próbálunk egyszerre optimalizálni annál hosszabb lesz a számítási idő. Másik pedig az, hogy általában egy hiperparaméter konfigurálásával, nem szokás nagy kihatással lenni a többi hiperparaméterre, ez főleg igaz a tanítási rátára. Ezért is érdemes a tanítási ráta optimalizálásával kezdeni.

Ha automatikusan szeretnénk kiválasztani a tanítási rátát, a leghíresebb az Adam [38] amely egy adaptív tanítási ráta optimalizációs algoritmus, amely pontosan mély neurális hálózatok optimalizálására lett készítve. Ez az optimalizációs algoritmus a sztochasztikus gradiens ereszkedés módszerrel szokott egyidejűleg használva lenni.

2.9.1. Sztochasztikus gradiens ereszkedés módszer

Szinte az összes mély tanulásos modell, egy fontos algoritmussal van felépítve, a sztochasztikus gradiens ereszkedés módszerrel(SGD) [5]. Ez egy bővített verziója az előbb

említett gradiens ereszkedés módszernek. A sztochasztikus gradiens egyszerű és effektív, habár nagyon óvatosan kell kiválasztanunk a különböző hiperparamétereket, főleg a tanítási rátát és a paraméter inicializálást.

A sztochasztikus gradiens ereszkedés módszerben ahelyett hogy 1 példán mennénk keresztül inkább egy kisebb köteget(mini-batch) dolgozunk fel egyszerre. Ez azért hasznos mert így a köteg veszteség gradiense egy becslése az egész tanítási adathalmaznak és ennek segítségével mindig a köteg méretének növelésével képesek vagyunk a becslés minőségét korrigálni. Emellett pedig az egyszerűen paralizálható videokártyák felhasználásával sokkal több adatot tudunk feldolgozni egyszerre. Ez viszont nagyon sok memóriát fog igényelni, és egy nagyon erős NVIDIA videokártyát is, a modell komplexitásától függően. Ezeknek a kötegeknek a száma mindig valamilyen 2^x szokott lenni a paralizálhatóság miatt. Az ajánlott kötegméret általában 32 és 256 között szokott lenni. Érdemes megemlíteni hogy a kicsi kötegméretetek képesek egy szabályozási módszerként is működni.

SGD módszernek az egyik legelterjedtebb algoritmus pedig az Adam. Adam [38] egy egyszerű algoritmus melynek kiszámítása hatékony,gradiens alapú sztochasztikus funkciók optimalizálására. Eredetileg gépi tanulási problémákra volt fejlesztve, amelyeknek nagy méretű adathalmazai vannak és ez természetesen mély tanulási felhasználásra is alkalmas. Az algoritmus hiperparamétereinek vannak intuitív interpretációi és tipikusan nagyon kevés finomításra van szükségünk az implementáció után.

3. Követelmény specifikáció

A neurális hálózat amelyet specifikálni fogok egy generatív ellenséges hálózat amely 2 modellből lesz felépítve, a G generátorból és D diszkriminátorból.

G Generátor, egy konvolúciós progresszív hálózat, bemeneti adatként random generált számokat fog megkapni, amelyeket mély konvolúciók segítségével átalakítja egy $X = [x, x, 3]$ méretű tömbbé, amely egy képként is elképzelhető. x a felbontásunk lesz, a 3 pedig RGB csatornák adatai. Ez a X tömb lesz a kimeneti adatunk.

D Diszkriminátor ugyan úgy egy konvolúciós progresszív hálózat, bemeneti adatként megkapja a X generátor által készített "képet" és a tanítási adathalmazból egy Y példányt. Diszkriminátorunk feladata megkülönböztetni az eredeti és a generált képet. Kimeneti adatként egy valószínűség értéket ad, szükségünk lesz mind a kettő képnek a valószínűségére. Ez tanítás folyamán fel lesz használva.

Tanítás folyamán a generátor, backprop felhasználásával, nem csak random zajt fog generálni képként hanem egyre jobban fog hasonlítani a tanítási adathalmazra, pedig nem interaktál vele.

A game theoretic approach miatt, a generátor meg a diszkriminátor optimalizálása egyidejűleg fog történni. Az optimális szituáció a Nash egyensúly elérése lenne, de gyakorlatban nem biztos, hogy megtörténik. Még egy elvárás a tanítási és a tesztelési veszteség között ne legyen hatalmas különbség.

Ha rendesen fel lett tanítva a modellünk akkor elhagyható a diszkriminátor és képek generálására lesz képes.

A Modell letesztelése MNIST adathalmazon, amely $28 \times 28 \times 1$ felbontású képeket tartalmaz. Mielőtt még mélyebb konvolúciókat használnánk, érdemes itt letesztelni, mivel emberi arcokhoz szükség lesz nagyobb felbontású képekre, amelyekhez mélyebb konvolúciók is kellenek majd. Fontos, hogy minden a modellel rendben legyen, mielőtt ki lenne bővítve egy mélyebb hálózatra.

A generátor képes legyen az emberi arc rejtett struktúrájának a generálására és emellett olyan arcok generálására amelyet a generátor nem csak betanult a tesztelési adathalmazból.

4. Tervezés

4.1. Mély tanuláshoz használt szoftver és hardver

4.1.1. Hardver mély tanulásra

hardver az egyik legfontosabb elvárás, a tudáson kívül, egy neurális hálózatnak a tanításában. Vannak arra példák, hogy régebbi hardveren is tanítanak neurális hálózatokat, de ilyenkor fontos feltérképezni, hogy mennyire mély modellről volt szó, mennyi egységgel és mennyi kalkulációval. Az évek során a neurális hálók egyre bonyolultabbak lettek, egyre több réteggel, komplikáltabb algoritmusokkal és komplexebb adat feldolgozásával.

A legelterjedtebb hardver amellyel szinte mindenki rendelkezik mostanában, az a CPU és a GPU lenne, amelyek általános felhasználásra van használva. CPU-val is képesek vagyunk tanítani neurális hálózatokat, de mivel nagyon sok mátrix szorzást kell elvégeznünk és a processzorok kevés maggal rendelkeznek, ezért eléggé lassúak mélyebb és komplexebb modellekhez. Ekkor jönnek be a videó kártyák (GPU) amelyek több ezer gyengébb maggal rendelkeznek mint a processzorok, de ezeknek a magoknak a száma jelentősen felgyorsítja a tanítás folyamatát, mivel a mátrix szorzásokat nagyon egyszerű párhuzamosítani.

Fontos megemlíteni, hogy ha videokártyát szeretnénk használni akkor csak NVIDIA kártyákat érdemes használni CUDA meg cuDNN miatt. AMD nagyon le van maradva gépi tanulós feladatokra.

Léteznek specializált hardver opciók is, ha van rá pénzünk és igény is. Legfontosabb az ASIC vagy Application Specific Integrated Circuits amely adott logikát tartalmaz, ez természetesen csak adott applikációknál fog működni, de olyan szituációkban pedig nagyon jól működik. Ilyen például a Google TPU, amely gépi tanulásra a leggyorsabb hardver, főleg a TPUv4 amely elvileg hamarosan elérhető lesz.

Én rendelkezek egy NVIDIA RTX 2070 Super kártyával, amelyet felfogok használni a modellem tanítására. Ha esetleg nem bírná ki a kártyám a terhelést, akkor online bérelhető mindenféle munkaállomás, amelyben sokkal gyorsabb videó kártya lenne.

4.1.2. Szoftver választása

Minden kód amit írunk a GPUra az a CUDA nyelven fog megtörténni amely hasonló C programozási nyelvhez. De rendes CUDA kódot írni nagyon nehéz és fontos, hogy minél optimálisabb legyen a kódom mély tanulásos rendszerekben. NVIDIA észrevette ezt a problémát és különböző libraryket adtak ki, mint a cuDNN libraryt, amely implementál fontos mély tanulásos koncepciókat. Ezek a libraryk már az évek során, optimalizálva lettek NVIDIA által, ezért nincs értelme nem használni őket.

Léteznek mély tanulásos frameworkök is, amelyek képesek futtatni ezeket a libraryket optimálisan a videó kártyákon, így neurális hálózat tanítása közben nem kell az apróságokkal foglalkoznunk. A legtöbb framework eléggé hasonló, de azért mégis van néhány különbség közöttük [39]. Persze ezek a frameworkök egyfolytában változnak, ezt főleg lehet látni a [39] papírban ahol, összehasonlították az akkori leggyakrabban használt frameworköket

Mostanában a leghasználtabb Frameworkök a Pytorch [40], Caffé [41] és a Tensorflow [42]. Tensorflow-t választottam személyesen mivel statikus gráfokat használ, ezek a gráfok kicsit optimálisabbak, mint a dinamikusak, mivel csak egyszer kell őket felépíteni. Képesek vagyunk a statikus gráfok súlyait lementeni és használni anélkül, hogy futtatnánk a kódot. Ez azt is jelenti, hogy a neurális háló, amit felépítünk, lehetne használni valamilyen más projektben is. Még egy kis apróság az, hogy Tensorflowra van nagyon sok fajta magas szintű wrapper, egyik ilyen, amelyet már ismerek a Keras [43]. Kerasnak a legfőbb szándéka a kódot minél olvashatóbbá tenni az emberi szemnek, és ezt fogom használni a projekthez. Eredetileg nem terveztem a Keras használatát, de Tensorflow 2.0+ verzió használatára, ajánlott. Mindemellett nagyon alapos dokumentációval rendelkezik a Tensorflow és a Keras is.

4.2. Adathalmaz specifikáció

Kettő adathalmazra lesz szükségem, egyik ezekből egy "sanity check" miatt fog kellelni, a második pedig a rendes implementációra. A legelterjedtebb tesztelési adathalmaz az MNIST adathalmaz amelyet az architektúra rendes működésének a tesztelésére fogok használni. Az MNIST adathalmazra azért is szükség van mivel nagyon egyszerű struktúrákat tartalmaz, az arcképekhez képest. Emiatt a különböző architektúrai problémákat sokkal hamarabb diagnosztizálni lehet, mint például ha arcképeket próbálnánk generálni kisebb felbontásban. Emelett pedig az arcképek túl sok információval rendelkeznek amelyek 28x28-as felbontásban nem lennének képesek teljes mértékű generálására az egyszerű számokat rendelkező adathalmazhoz képest. A GAN hálózatok amelyeket én választottam, képesek az adathalmaz disztribúció struktúrájának a megtanulását és ha ebben a struktúrában valamilyen inkonzisztenciát észreveszünk az nagy valószínűséggel egy másik adathalmaz disztribúcióra is igaz. Ilyenkor viszont amikor adathalmazt váltunk természetesen a súlyokat nem tudjuk újra használni egy másik adathalmazhoz, ezért a tanítást előlről kellene kezdeni. De pont ezért tervezem az MNIST adathalmaz tesztelése után a hálózatot bővíteni, mivel alapból el kellene dobni a súlyokat az adathalmaz váltáskor. Az MNIST adathalmaz egy [28,28,1] felbontású adathalmaz ahol, [szélesség, magasság, csatornák], ez sokat fog segíteni a mély konvolúciók tesztelésében, hogy a modell rendesen megtanulja-e a globális és a lokális struktúrákat. A rendes implementációhoz pedig egy nagy minőségű, RGB csatornás, arckép adathalmazra van szükségem, amelynek beszerzésére 2 lehetőségem van.

- Saját adathalmaz gyűjtése, ahol oda kell figyeljek a képeknek a konzisztens reprezentációjára, a képnek a különböző attribútumára mint kontraszt és emellett ez nagyon sok monoton munkaórába kerülne. Még egy probléma lehet, maga a képeknek a licencezése.
- Másik lehetőség egy már létező adathalmaz felhasználása, amely akár több 100 munkaórát megtakarítana.

Mind 2 lehetőségnek megvannak az előnyei és a hátrányai. Szakirodalomból ahogy olvastam az ilyen projekteknél általában nagyobb csapatról van szó mint egy személy. Ez azt is fogja jelenteni hogy több munkaerővel különböző feladatokat képesek felbontani a csapat között. Az adathalmaz gyűjtésre legalább mindig egy vagy kettő személy szokott lenni, mivel ez maga is egy külön tudományág, hogy milyen módon kell rendesen felhasználni.

ható adathalmazt létrehozni, amely nem a szakdolgozatomnak a témája. Ezek az indokok miatt és azért, hogy fókuszálhassak az architektúrára, egy már létező adathalmazt fogok használni. Az ilyen adathalmazoknak tartalmazniuk kell egy engedélyt(license), amely megengedi az adathalmaz ingyen felhasználását és adaptálását non-profit célokra.

Az implementációhoz szükséges lenne egy [128,128,3] vagy esetleg egy [256,256,3] felbontású arckép adathalmazra. Nagyobb adathalmazt is feltudok használni, [1024,1024,3] minőségűt is, de nem rendelkezek 8 google TPUval ezért a konvolúciók csak [128,128,3] vagy [256,256,3] felbontásig fognak felmenni. Legelterjedtebb arckép adathalmaz a CelebA [23] adathalmaz, amely 202 ezer képet tartalmaz [178,218,3] minőségben. A nem teljesen konvencionális 178x218 felbontás nem probléma a numpy [44] miatt, képes átalakítani a reshape módszerrel. Sok papírban használták, de számomra már egy kicsit alacsony minősége van az én projektemhez. Létezik a CelebA-HQ [22] amelyet a Progressive Growing of GAN modellnél is használtak, de nem szeretném ugyan azt az adathalmazt használni az ugyan úgy folyamatosan növekvő hálózatomhoz. Még egy lehetőség az FFHQ [24] adathalmaz, amelyre nem találtam PGGAN papírt [128,128,3] vagy [256,256,3] minőségre, ezért is ezt választom.

4.2.1. Adathalmaz előkészítése

Első lépés az adathalmaz letöltése valamilyen adattároló eszközre. Ha van lehetőség rá akkor egy SSDre, ne hogy az adatok olvasása legyen a bottleneck a hálózatban. Hasonló indok miatt az adatok bebetöltése a hálózatba sokszor a memóriából történik mivel az gyorsabb mint a tároló eszköz. Az adatok betöltésénél fontos, hogy a processzor segítsen az adatok párhuzamosított betöltésével. Amikor már letöltöttem az adatokat a numpy segítségével átfogom őket alakítani a tervezett felbontásra.

Amikor már letöltöttem az arcképeket, ezeket még át kell alakítani egy rendes adathalmazzá, amelyet a tf.data.Dataset API segítségével fogok. Ezt a tensorflow [42] API támogatja a hatékony bemeneti pipeline-okat. Az API dokumentáció szerint ez készít egy forrás adathalmazt a bemeneti adatokból, ahol ezután feldolgozzuk az adatokat és végül iterálunk az adathalmazon, és feldolgozza az elemeit. Az iterálás nem egyszerre történik az egész adathalmazon, tehát nem kell az egész adathalmaznak beleilleszkednie a memóriába. Betöltés után normalizálni kell a képek adatait olyan módon, hogy az alapértelmezetten [0, 255] intervallumra eső képcsatorna értékek a [-1,1] intervallumra essenek. Ez segít a számítás gyorsításában és az aktivációs függvényeknek is egyszerűbb lesz.

4.3. Fejlesztés tervezett menete

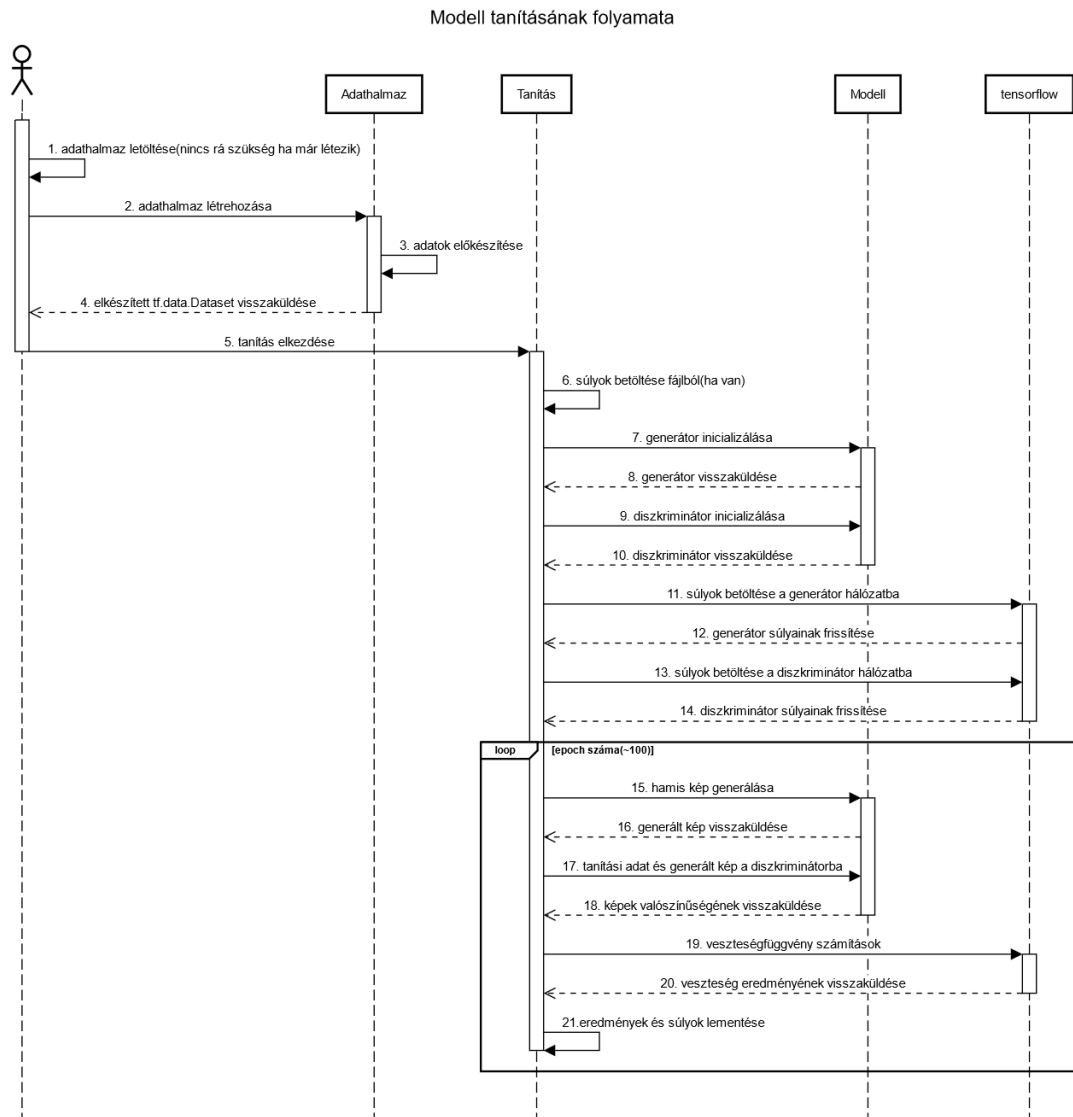
A rendszer fejlesztésének tervezett menetére készítettem egy Gantt-diagramot amelyet a 4-12 ábrán láthatunk. A fejlesztésnek egyik legfontosabb lépése az maga az irodalomkutatás, főleg egy ilyen komplex kihívásnál mint arckép generálás. Ezután fontos az adathalmazunkat összegyűjteni, mivel a neurális hálózatot ehhez képest kell felépíteni. Következő lépés az architektúra kiválasztása amely képes lesz rendesen funkcionálni az adathalmazunkon és a lényege ennek az architektúrának az alapjainak implementálása. Itt akármilyen szabályozás vagy optimalizálásos lépéseket még nem szükséges implementálni, csak az alapokat mint a 2 hálózatot, veszteségfüggvényt és aktivációs függvényt. Az elkészített modellnek a feltanítása és tesztelése az MNIST adathalmazon, amely tökéletes lesz a rendszerben különböző bugoknak az elemzésére és kijavítására. Ha a teszteléssel végeztünk akkor a hálózatoknak a rétegeit bővíteni kell, hogy a választott adathalmaz feldolgozására alkalmas legyen. Ilyenkor is belesik egy tesztelési periódusba, hogy biztosan minden rendben funkcionál. Ezután pedig a hiperparamétereknek az optimalizálása és további optimalizációs lépések elkezdődnek. Hiperparaméterek optimalizálása nagyon sok másik hálózathoz képest a leghosszabb folyamat, tehát fontos ennek minél több időt adni, de előfordulhat, hogy nem lesz rá kellő idő.

	1	2	3	4	5	6	7	8	9	10	11	12	13
Irodalomkutatás	■	■	■										
Adathalmaz keresése, összegyűjtése			■	■									
Architektúra kiválasztása és alapjainak implementálása				■	■	■							
Modell tanítása és tesztelése MNIST adathalmazon						■	■	■					
Hálózatok változtatása, ha szükséges tesztelés után							■	■					
Hálózat bővítése								■	■				
Hálózat tesztelése nagyobb felbontású adathalmazokon									■	■	■		
Hiperparaméterek optimalizálása											■	■	■
További optimalizációs lépések												■	■

4-12. ábra. A rendszer fejlesztésének tervezett menete, Gantt-diagrammon ábrázolva

4.4. Rendszer funkcionalitása

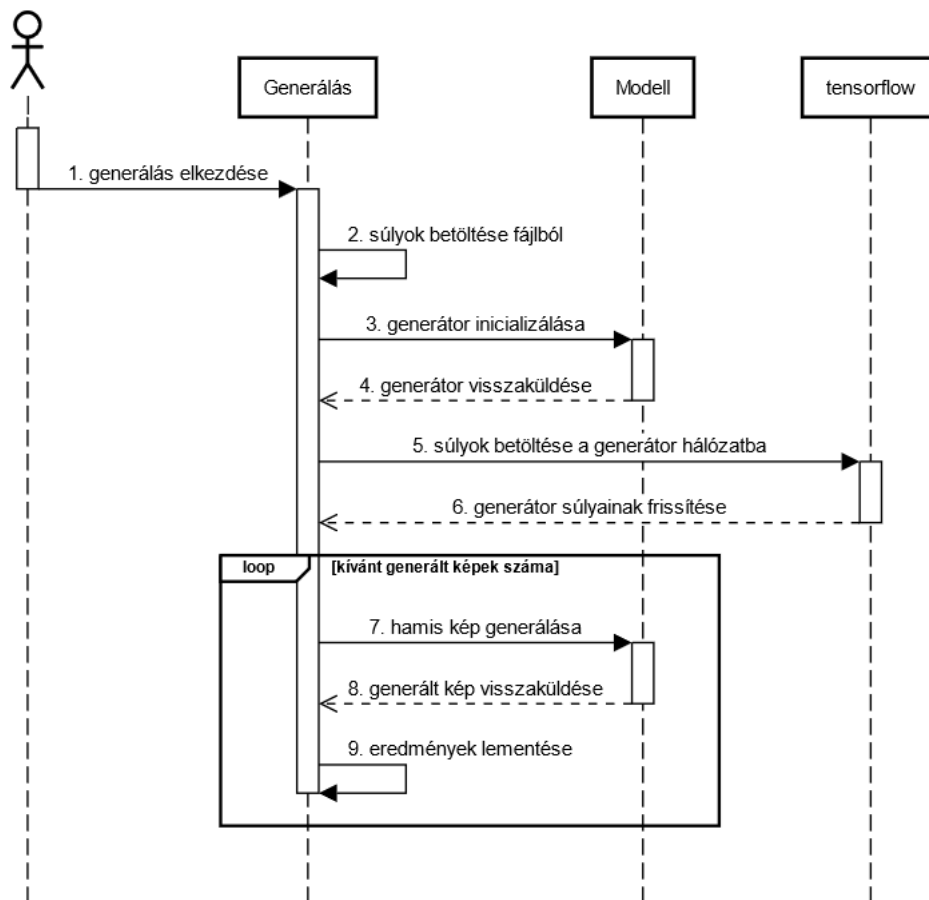
A rendszer különböző funkcionális részletének a tisztázása érdekében készítettem két szekvenciadiagramot. A generátor és a diszkriminátor tanításának a folyamatát lehet látni a 4-13 ábrán.



4-13. ábra. A rendszer tanításának a szekvenciadiagramja

A 4-14 ábrán látható szekvenciadiagramon pedig a már feltanított hálózatnak a szekvencia diagramját láthatjuk. Nagy előnye a GAN hálózatoknak, hogy a diszkriminátor teljesen elhagyható és így képeknek a generálása rendkívül gyors lesz.

Feltanított modellnél arckép generálása



4-14. ábra. Egy feltanított rendszerenél, arcképek generálásának a szekvenciadiagramja

5. Fejlesztés folyamata

A fejlesztés eredetileg tervezett menetének megfelelően, az irodalomkutatás után keresni kell egy tesztelési és egy végleges adathalmazt. Mivel ennek a kutatása és összegyűjtése sikeresen megtörtént már a tervezési fázisban, ezt követően pedig az architektúrának specifikálására és implementálására lesz szükség.

5.1. Architektúra

Az architektúra felépítésekor, még mielőtt sokkal alaposabb irodalomkutatást tettem volna, feltanítottam egy nagyon standard mély konvolúciós architektúrát amely eredetileg a tervezett architektúrámmá lett volna, csak kevesebb paraméterrel és szabályozási technikák nélkül. Ehhez a konfigurációhoz "prototípus" vagy esetleg "konfig *A*"-ként fogok referálni. A modell feltanítása után, az eredmények kiértékelésekor, viszont nem voltam az eredményekkel megelégedett. Ezt az első architektúrát áttekintem, eredményeit kiértékelem, ezután továbbfejlesztési lehetőségekről beszélek és végül elkészítem a rendes architektúrámat a MORKGAN vagy "konfig *B*", amely ennek a kis prototípusnak a hibáiból tanul.

5.1.1. Prototípus

Prototípushoz használt fontosabb függőségek, amelyeket nem feltétlenül fogok változtatni, maximum csak frissíteni, ha ez valamilyen teljesítménynövelő hatást ad:

- Editor: PyCharm 2021.2.1 (Community Edition)
- Python(3.9)
- Keras(2.8.0)
- Keras-Preprocessing(1.1.2)
- tensorflow(2.8.0)
- numpy(1.22.3)
- matplotlib(3.5.1)
- NVIDIA Graphics Driver(471.41)
- NVIDIA CUDA toolkit(11.4.1)
- cuDNN(8.2.4.1)

A prototípus mély konvolúciós rétegzést használ, amely a DCGAN [19] papíron alapszik, a generátornak és a diszkriminátornak is láthatjuk a teljes felépítését a 5-15 és 5-16 ábrán. Az eredeti DCGAN papírban az első réteg a generátorban ahol a random zajt generálunk kétszer több paraméterrel rendelkezett mint az én prototípusom a számítási kapacitás miatt, ezt később természetesen megdupláztam volna. Még egy változás az eredeti papírhoz képest, hogy upscale rétegeket tartalmaz a generátor modell, amely miatt jobb képminőséget fog elérni mint a papírban használt konvolúciók, de viszont nagyobb számítási kapacitást is igényel. Prototípus implementációs információi:

- Aktivációs függvény rétegek között: ReLU a generátorban és LeakyReLU($\alpha = 0.2$) a diszkriminátorban, [19] ajánlásával
- Utolsó rétegen a DCGAN [19] papír ajánlása miatt egy másfajta aktivációs függvényt használtam, a papírban $\tanh(x)$ függvényt ajánlottak én Sigmoidot használtam, az irodalomkutatásban talált okok miatt
- Veszteségfüggvény: binary crossentropy/log loss
- Optimalizációs eljárás mind a kettő hálózatra Adam [38] $\alpha = 0.0001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$, $minibatch_{size} = 32$ értékekkel
- Adathalmaz a CelebA [23] volt amelynek a képei downscale-ezve lett 64x64x3 felbontásra, nearest neighbor interpoláció használatával, számítási kapacitás miatt
- 80 epochig lett tanítva, otthoni videokártyán, minden egyes epoch 1400 másodperc körül volt
- Nem történt hiperparaméter optimalizálás, amely pedig az egyik leghosszabb folyamat egy hasonló modell feltanításában a [24] papír szerint, de a papírban található hiperparamétereket alkalmaztam, amelyek nem teljesen optimálisak, de egy jó kezdőpont

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 8192)	1056768
reshape (Reshape)	(None, 8, 8, 128)	0
conv2d_transpose (Conv2DTran	(None, 16, 16, 128)	262272
re_lu (ReLU)	(None, 16, 16, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 32, 32, 256)	524544
re_lu_1 (ReLU)	(None, 32, 32, 256)	0
conv2d_transpose_2 (Conv2DTr	(None, 64, 64, 512)	2097664
re_lu_2 (ReLU)	(None, 64, 64, 512)	0
conv2d (Conv2D)	(None, 64, 64, 3)	38403
Total params: 3,979,651		

5-15. ábra. Prototípus generátor rétegzésének az összefoglalása

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 64)	3136
leaky_re_lu (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	131200
leaky_re_lu_1 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 8, 8, 128)	262272
leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 1)	8193
Total params: 404,801		

5-16. ábra. Prototípus diszkriminátor rétegzésének az összefoglalása

5.1.2. Prototípus továbbfejlesztési lehetőségei

A prototípus továbbfejlesztésének első legfontosabb lépése az a modell tanításának a stabilizálása. Veszteségfüggvény javítására az irodalomkutatásban már említett Wasserstein GAN [20] és ennek a WGAN-GP variánsa. Az eredeti PGGAN papírban ACGAN [45] által behozott változtatások a veszteségi függvényt is befolyásolta, de én ezt nem fogom implementálni mivel későbbi papírok mint a StyleGAN [24] nem támogatják a használa-

tát.

A DCGAN papírban található rétegzés nagyon sok stabilitási problémát okozhat, mint például a mode collapse. Azt a mérnöki döntést hoztam, hogy a PGGAN [22] papírban leírt rétegzést fogom használni. Itt ahelyett, hogy egyszerre próbálja megtanulni az egész felépítését az arcoknak, a PGGAN vagy folyamatosan növekvő GAN hálózatok alacsony minőségen kezdenek el tanulni és lassan megtanulják a képek struktúráját, addig amíg el nem érik a kívánt minőséget. Emellett pedig sok más szabályozási technikát fogok használni a PGGAN, StyleGAN és StyleGAN2 papírokból. A PGGAN megközelítése lehetővé teszi a gyorsabb tanítást is olyan módon hogy nagyobb minőségben is generálunk, a folyamatosan növekvő megközelítés miatt. Esetleg ha nem váltanék rétegzést akkor van néhány lehetőségem mint Dropout vagy Batchnorm szabályozás. Néhány helyen láttam a LeakyReLU használatát a generátor hálózatban is.

Nem használtam paraméter inicializálást, ez segíthet a tanítás legelejét felgyorsítani. Hyperparaméter optimalizálás az egyik legfontosabb lépés az ilyen hálózatoknál és nagyon sok ideig tarthat amelyre nem biztos hogy van számítási kapacitásom vagy időm se esetleg. Azt viszont olvastam hogy a WGAN gradiens penalttyvel az sokat segít olyan modelleken ahol kevés hiperparaméter optimalizálást végeztek. Emellett pedig a PGGAN papírban alkalmazott hiperparamétereket fogom újrahasználni egy ilyen kezdőértékként, amelyeket az architektúra specifikációnál adok meg, "()" között. Minden felhasznált paraméter az implementációmban a config.py fájlban az *opt* dictionary-ben található.

A stabilitás után a legfontosabb lépés az maga a minőségnek a növelése, amelyre a PGGAN papírban leírt rétegzés és szabályozási technikák sokat fognak segíteni. Habár az ilyen progresszívan növekvő architektúrákban, vagy kicsit pontosabban az upsampling és downsampling rétegek használatakor mindig valamilyen apró artifactokkal találkozni fogunk, amelyekre még nem biztos, hogy képes leszek javításokat találni. A StyleGAN2 papírban ki lett mutatva hogy egy nagyobb, vagy mélyebb rétegzésű modell az nagyon nagy minőségi javításra képes de erre nem feltétlenül lesz lehetőségem mivel otthoni GPUval tanítom a modellt és nem 4 darab GOOGLE TPU vagy NVIDIA Titan V felhasználásával. Esetleg, ha még lesz időm a félévben és már minden más lépést elvégeztem akkor lehet, hogy online bérelnék valahonnan egy munkaállomást, amely sokkal erősebb mint az otthoni videokártyám. Utoljára pedig adathalmazt is váltok, amelyet már specifikáltam a tervezés fázisban.

5.1.3. Architektúra specifikáció

Sok tanulással és adathalmazzal kapcsolatos paraméterről csak a tényleges implementációnál lesz szó, de a legfontosabb architektúrával kapcsolatos információk, itt találhatóak:

- Rétegzés: PGGAN
- Diszkriminátor veszteségfüggvény: $\text{WGAN-GP}(\lambda = 10, \text{drift}_{\text{weight}} = 0.001)$
- Generátor veszteségfüggvény: $\text{Adverisal}(\text{formula} = -\Delta \text{predikcio}_{\text{hamis}})$
- Optimalizációs eljárás mindkettő hálózatra: $\text{Adam}(\alpha = 0.001, \beta_1 = 0.0, \beta_2 = 0.99, \epsilon = 10^{-8})$
- Szabályozási technikák: WeightScaling , $\text{PixelNorm}(\epsilon = 10^{-8})$
- Általános rétegek aktivációs függvényei: $\text{LeakyReLU}(\alpha = 0.2)$
- Utolsó réteg aktivációs függvényei: $\tanh(x)$ vagy lineáris
- Adathalmaz: FFHQ(MNIST tesztelés)

5.2. Modell implementálása

A prototípusom implementálása után rájöttem arra hogy a kód amelyet magamtól megírtam nem úgy volt felépítve és leködölve, hogy könnyen bővíthető és felhasználható legyen más adathalmaz vagy esetleg más konfigurációkra is. Egy nagyon fontos tervezési szempont számomra a skálázhatóság és olyan implementáció készítése amely könnyen bővíthető és újrahasználható különböző konfigurációkra. A GAN hálózatoknál ez egy kifejezetten fontos lépés mivel nagyon instabil hálózatok és a konfigurációt egyfolytában változtatni is kell. Ezzel a hozzáállással lett elkészítve a MORKGAN, és később ha lesz rá időm akkor ezt be is mutatom más adathalmazoknál. Ennek érdekében létrehoztam egy config.py fájlt, amelyben minden fontos konfigurációs információ 3 "dict" objektumra lett lebontva. Dict objektumok Pythonban képesek akármilyen objektumot eltárolni, amelyeknek meg kell adni egy kulcsszavat is, ennek meghívásával pedig visszakapjuk az objektumot. Ez a 3 objektum pedig az "opt" amely az Adam hiperparamétereit tartalmazza és a veszteségi függvényekkel kapcsolatos paramétereket. "training" pedig a tanítással és a rétegzéssel kapcsolatos információkat tárol. "dataset" pedig az adathalmazzal kapcsolatos információkat mint, név, színcsatornák száma, és progresszívan növekedés felbontásai tömb formában. Később egy "path" dict fájlal is kiegészítettem, hogy egyszerűbben le lehessen menteni különböző eredményeket.

5.2.1. Adathalmaz előkészítése

Első lépésként az adathalmazzal kapcsolatos implementálási kihívásokkal kezdtem. Minden adathalmazt letöltök előre tárolóeszközre, egy SSD-re, hogy ne legyen a hálózat bottle-neckelve, a tárolóegység sebessége vagy esetleg internet letöltési sebessége miatt. Egy probléma ami feljött az, az volt, hogy a különböző adathalmazok, különféle tárolási módokat használnak. Ezért különféle módon kell betölteni és átdolgozni ezeket, ezért is lett létrehozva a "dataset" dict, amely tárolja ezeket az információkat. Emellett pedig progresszívan növekvő hálózatról van szó, tehát ezeket az előre elkészített adathalmazokat, mindig betöltéskor át kell méretezni az éppen kívánt felbontásra, ezt főleg a numpy és tensorflow kezeli. Képek átméretezésére létezik sok beépített metódus tensorflowban, én a "Nearest Neighbor interpolation" metódust választottam a fejlesztés elején, mivel gyors, de néha eléggé "kockás" eredményeket tud okozni. Később csak is kizárólag "Bilinear interpolation"-t használok mivel már a sebesség nem fog túlságosan számítani, csak az eredmény minősége. Ezután pedig a betöltött pixel adatok módosítására van szükség, ahol x maga a betöltött adatok(lebegő pontos 32bites) halmaza és y egy az elkészített adathalmaz:

$$x' = \text{RandomFlipLeftRight}(x)$$
$$\text{normalized} = (x' - 127.5)/127.5$$
$$y = \text{ShuffleandBatch}(\text{normalized})$$

Első lépésként a betöltött képeket randomba balra vagy jobbra fordítjuk, ezután pedig normalizáljuk az adatokat [0,255] intervallumról, [-1,1] intervallumra(standardized), hogy a globális közepe 0 legyen és ne 0.5 mint [0,1] implementációknál. Normalizálás az egy technika a kép feldolgozásban ahol egy adott intervallumra csökkentjük a pixel adatokat. Ezen a normalizált halmazon pedig két technikát fogok alkalmazni a "Shuffle" amely a halmaz értékeit random felcseréli egy paraméter alapján. "Batch" pedig kötegekre bontja az adathalmazt, amely eredetileg egy statikus tömbből, de most már dinamikusan lesz kiszámolva a kívánt felbontásra az én videokártyámra(6 GB VRAM), nehogy kifusson a GPU videó memóriából(VRAM). Itt beleütközök még egy problémába, mivel Windows 10 csak a 90% át engedi a videokártya videomemóriájának felhasználására, és tensorflow ennek az értéknek is csak a 90%-át engedi. Így maradunk 8 GB VRAM-ból közel 6 GB VRAM-mal. Ezt vagy Linux platformra váltással lehet megkerülni, vagy specializált videokártyák használatával, de egyenlőre maradok ugyan annál a konfigurációnál. Az idáig

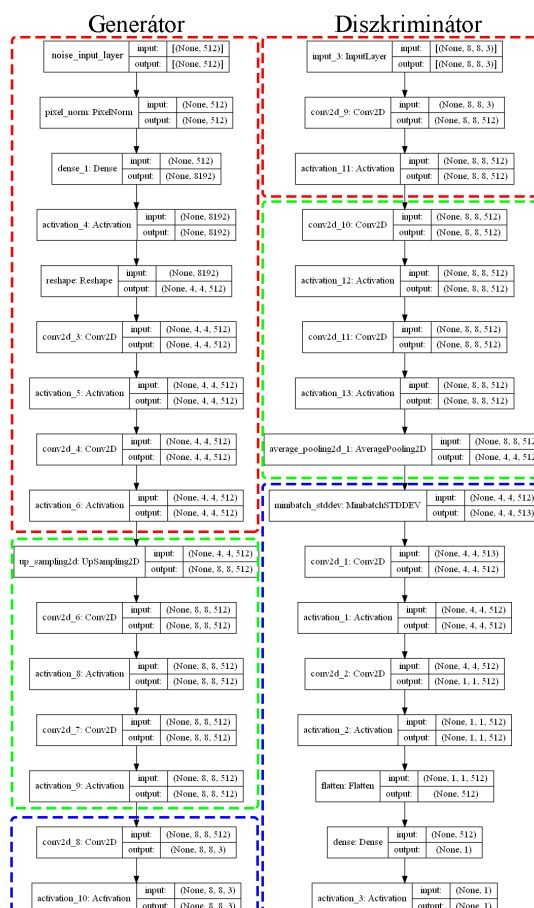
elkészített adathalmaz egy BatchDataset típusú adathalmaz lesz, amelyen nagyon egyszerű lesz iterálni, kötegekre lebontva. De általában egy felbontáson több lépést teszünk meg mint a hossza az adathalmaznak. Ha a végére érünk az adathalmaznak akkor fontos, hogy ismétlésre legyen képes, amelyet a repeat metódus tesz lehetővé. Tanítás közbe mindig limitálni kell a lépéseket vagy különben ez örökké fordulna lenne. A fent leírt lépések után pedig készen is állunk az adathalmazzal kapcsolatos kihívásokkal.

5.2.2. Hálózati blokkok és rétegzés implementálása

A MORGGAN rétegzése a PGGAN papírban használt rétegzésen alapul, de természetesen nem teljesen ugyan az lesz, sok faktor miatt mint, más framework, kevesebb VRAM amely miatt muszáj átgondolni a hálózatban található konvolúciós rétegeket. Ahogy már eddig is említettem a GAN hálózatok 2 neurális hálóból állnak a generátor és a diszkriminátor hálózatokból. Az első hálózat amelyen elkezdtem dolgozni az a generátor volt, mivel nagyon egyszerűen kiértékelhető, hogy a funkcionalitását képes elvégezni, amely most még csak random zaj generálása adott felbontáson. Diszkriminátornak a célja pedig valószínűségérték visszaküldése egy bemeneti adathalmazra.

Eredetileg a modelljeim rétegzése statikusan megvolt írva adott felbontásokra, mint 4^2 , 8^2 , 16^2 és az előző felbontásról mindig csak betöltötte a súlyokat. Ez nagyon egyszerűen dedukálható, hogy több ezer olyan sor leködölására van szükség amely igazából nem kötelező, emellett nagyon könnyen hibázhatok ennyire sok hálózat leírásában és nagyon nehezen lehetne problémát diagnosztizálni. Ezért a PGGAN papírban található rétegzést átvizsgáltam és 3 különböző blokkra lehetne ezeket bontani, amely sokat segítene a hálózatok dinamikus felépítésében. Ha ezt sikeresen implementálnám akkor skálázható lenne a rendszer, amely csak hardver számítási kapacitásával és VRAM mennyiségével lenne limitálva. A generátor esetében ezek a blokkok lennének az "input", "upsample" és a "toRGB" blokkok. A hálózat inicializálásánál csak az "input" és a "toRGB" blokkokra van szükség, és minden egyes progresszív növekedésnél egy "upsample" blokkot kell elhelyezni a "toRGB" blokk elé, ezeknek felépítését lehet látni az [5-17](#) ábrán.

A diszkriminátor is ugyan úgy lebontható 3 blokkra, ezek az "output", "downsample" és a "fromRGB" rétegek. Inicializálásnál csak a "fromRGB" és "output" blokkra van szükségünk ahol a "fromRGB" az első blokk és a "downsample" blokkok mindig a "fromRGB" után lesznek beillesztve progresszív növekedésnél, "output" pedig az utolsó blokk, ezeknek felépítését lehet látni az [5-17](#) ábrán.



5-17. ábra. Progresszívan növekvő generátor és diszkriminátor hálózat, 8^2 felbontásra, ahol 4^2 felbontáson kezdett el tanulni. Pirossal jelölt blokkok a generátor esetén az input blokkhoz tartoznak, diszkriminátor esetén pedig a fromRGB blokkhoz. A zölddel jelöltek pedig az up- és downsample blokkok. Kékkel jelöltek pedig a generátor esetén a toRGB, diszkriminátor esetén pedig az output. Ebben a modellben nem alkalmaztam weightscaling-et és pixelnormot minden rétegen mivel nem fért volna ki egy ábrára.

Ezek a blokkok is lebonthatóak kisebb elemekre, amelyeket rétegeknek fogok nevezni. Egy konvolúciós réteg amelyet a 5-17 ábrán láthatunk nem csak egy konvolúcióból áll, hanem a hasonló rendszerek szekcióban leírt kiegyenlített tanítási ráta(weight scaling) és a generátorban pedig a pixel alapú jellemző vektorok normalizálása(pixelnorm). Természetesen aktivációs függvényt is tartalmaznak ezek a rétegek, a normalizációs technikák előtt alkalmazom őket.

Még egy karakterisztikája van a PGGAN-alapú hálózatoknak, nem csak a progresszív növekedésen alapuló "felbontás csere" létezik, ahol ugye felskálázuk a hálózatot egy nagyobb felbontásra, egy új block beillesztésével, hanem egy úgynevezett modell csere is

létezik. A felbontás csere eljárásnál egy olyan hálózatot készítünk ahol, a generátornál, ahogy eddig említettem egy upsample blokkot helyezünk el a toRGB blokk elé, de emellett létrehozunk ennek a hálózatnak az olyan verzióját ahol az upsample blokkból csak az upsample réteg van benne, a konvolúciós rétegek pedig nem kerülnek bele. Itt fog elindulni a "fade-in" folyamat ahol beolvasztjuk az új konvolúciós rétegeket a hálózatba. Ahhoz hogy elkerüljük a modell felrobbanását az új konvolúciós rétegekkel rendelkező kimenet, egy alpha érték alapján lesz befolyásolva, amelynek a kiindulási helyzetben nulla lesz az értéke és lassan a "fade-in" folyamat során egyre jobban közelít az 1 értékhez. Az alpha értéke kötegenként lesz frissítve egy Keras callback segítségével, amely nincsen számítva, ha nem "fade-in" folyamat közbe vagyunk, optimalizációs okok miatt. A hálózat amely pedig nem rendelkezik az extra konvolúciós rétegekkel, a kimenete pedig $1 - \alpha$, lesz. Amint eléri az alpha az 1 értéket, modell csere folyamat történik és a csak a konvolúciós rétegekkel rendelkező modell marad és elkezdődik a stabilizálás folyamat, amely ugyan annyi képen keresztül fog tanulni mint a "fade-in" folyamat, de itt nem lesz semmilyen alpha érték, csak szokásos tanítás. A modell súlyai pedig megmaradnak felbontás vagy modell csere között.

Mind a kettő hálózat egy úgynevezett MORCKGAN nevezetű osztályban lesz felépítve és eltárolva. Ez a MORCKGAN egy alosztálya a Kerasnak a Model ősosztályának és a feladata a különböző hálózati funkciók elvégzése lesz. Annyit fog ez a döntés jelenteni, hogy néhány beépített Keras funkciót nem leszek képes használni, mint a checkpoint rendszert, mivel nem egy eredeti Model osztályom van hanem egy alosztálya. Viszont ez lehetővé fogja tenni a kettő modellnek a tanítását egyszerre kezelni és különböző konfigurációs adatokat egy példányban tárolni.

5.2.3. Tanítás implementálása

A tervezésben található szekvenciadiagram alapján, amely a modell tanítását tartalmazza, első lépéseink nagy része a különböző komponensek inicializálása, amely a main.py scriptben található. Mivel az adathalmazok már előre le vannak töltve ezért csak az adathalmaz betöltésére és előkészítésére van szükség. Ezután pedig sok konfigurációs információt kell beállítani amely kell a generátor és diszkriminátor felépítéséhez, ezután pedig a tanításhoz. Itt még csak felépítjük, teljesen még nem konfiguráltuk a hálózatokat.

Tanítás metódus belépésénél példányosítjuk a generátornak és a diszkriminátornak az optimalizációs eljárását. Erre azért is van szükségünk, mivel amikor a hálózatot konfi-

guráljuk a ".compile" keras metódussal, akkor ide paraméterként be is adhatjuk ezeket. Ezenkívül viszont a modelleknek az optimalizációs eljárásait minden egyes modell "cserénél" vissza kell állítani az eredeti állapotára. Elég trükkös visszaállítani egy optimalizációs eljárás állapotát ezért az optimalizációs eljárás lokális példánya sokat segít ezen.

Konfiguráció után ideje tanítani a modellt. A modell tanítása egy felbontásra sokat különböztethet konfigurációs adatok alapján. Általános konfigurációk alapján viszont ahol 800 ezer képet lát a modell minden egyes modell "csere" előtt, olyan 50 epoch és 1000 lépésen fog átmenni a modell. Keras beépített tanítási lépéseit felül írrom egy egyéni tanítási lépés metódussal, ahol a veszteség függvény értékeit az adott hálózatra és ez alapján a gradienseket is számolni fogom. Veszteségfüggvényeknél egyetlen dolog amit érdemes megemlíteni, hogy a diszkriminátornak a wgan-gp veszteségfüggvénye eltér egy kicsit az eredetitől. Egy úgynevezett "drift" értéket mindig hozzáadunk a teljes veszteségfüggvényhez. Ez a "drift" egy nagyon kicsi értékű súly amelyet azért adunk hozzá a veszteségfüggvényhez, hogy a diszkriminátor kimenete "ne távolodjon el túlságosan a nullától", a PGGAN papír alapján.

Ahogy már említettem a beépített Keras checkpoint rendszert nem leszek képes használni, de azért ugyan úgy szeretnék valamilyen tanítás folytató megoldást készíteni. A GAN rendszerek feltanításának ideje még a legerősebb GPU-k használatával is képes több hétbe telni, nagy felbontásokon. Előfordulhat, áramkimaradás vagy valamilyen operációs rendszer szintű hiba miatt, hogy a használt gép hirtelen leáll vagy újraindul. Kutatás után találtam egy másik Keras megoldást, ahol csak az architektúra súlyai vannak lementve. Ez működőképes, de a hálózatot mindig addig az állapotig újra kell építeni és csak modell cserék között lesz súlyok mentése.

5.3. Modell tanítása MNIST adathalmazon

5.3.1. Tanítási adatok

Egyetlen egy nagy különbség az eredetileg tervezett felbontásban található. Eredetileg [28,28,1] méretű képeket szerettem volna generálni, ahol az architektúra által generált képek 7->14->28 módon működne. Generálás közben mivel páratlan kezdeti felbontással kezdek, ezért a generátor által készített képekben a jobb alsó sor és oszlop teljesen fehér lett, tehát ezeken a pontokon a konvolúciók mindig aktiváltak és nem változnak. Ezt egy extra konvolúciós réteggel meg is lehet oldani, de ha már ilyen lépéseket teszek,

Generátor	Akt. Fgvny.	Kimenet	Paraméterek	Diszkriminátor	Akt. Fgvny.	Kimenet	Paraméterek
Zaj vektor	-	1 x 1 x 256	-	Bemeneti kép	-	32 x 32 x 1	-
Conv 4 X 4	LReLU	4 x 4 x 256	1.05M	Conv 1 X 1	LReLU	32 x 32 x 256	590K
Conv 3 X 3	LReLU	4 x 4 x 256	590K	Conv 3 X 3	LReLU	32 x 32 x 256	590K
Upsample	-	8 x 8 x 256	-	Conv 3 X 3	LReLU	32 x 32 x 256	590K
Conv 3 X 3	LReLU	8 x 8 x 256	590K	Downsample	-	16 x 16 x 256	-
Conv 3 X 3	LReLU	8 x 8 x 256	590K	Conv 3 X 3	LReLU	16 x 16 x 256	590K
Upsample	-	16 x 16 x 256	-	Conv 3 X 3	LReLU	16 x 16 x 256	590K
Conv 3 X 3	LReLU	16 x 16 x 256	590K	Downsample	-	8 x 8 x 256	-
Conv 3 X 3	LReLU	16 x 16 x 256	590K	Conv 3 X 3	LReLU	8 x 8 x 256	590K
Upsample	-	32 x 32 x 256	-	Conv 3 X 3	LReLU	8 x 8 x 256	590K
Conv 3 X 3	LReLU	32 x 32 x 256	590K	Downsample	-	4 x 4 x 256	-
Conv 3 X 3	LReLU	32 x 32 x 256	590K	Minibatch stddev	-	4 x 4 x 257	-
Conv 1 X 1	Tanh	32 x 32 x 1	257	Conv 3 X 3	LReLU	4 x 4 x 256	592K
Összes tanítható paraméter			5.2M	Conv 4 X 4	LReLU	1 x 1 x 256	1.05M
				Fully connected	Tanh	1 x 1 x 1	257
				Összes tanítható paraméter			6.23M

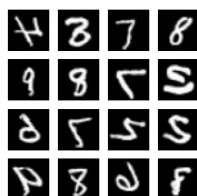
5-18. ábra. Progresszívan növekvő hálózat rétegzése, MNIST adathalmazra, [32,32,1] képek generálására

amely egyébként teljesen haszontalan lenne a végleges adathalmazhoz, gondoltam előre is letesztelhetem a 4->8->16->32 generálást amely az alapja lesz a bővített hálózatnak.

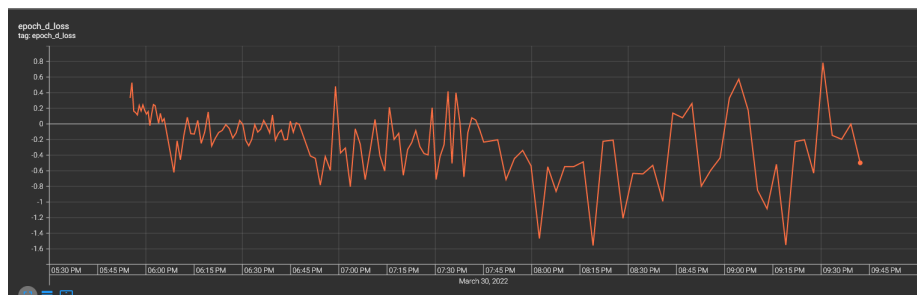
Az MNIST adathalmazt pedig meghagytam az eredeti formájában ahol tanításra 60000 képet használtam fel, tesztelési folyamatokra pedig 10000 képet tartalmazó adathalmazt használtam.

5.3.2. Eredmények kiértékelése

Az MNIST adathalmazon való tesztelésnek a lényege az a kód oldali implementáció funkcionalitásának tesztelése. A GAN modellek nagyon komplexek és instabilak, ezért az implementációjuk során sok hibába befuthatunk, amelybe sikerült bele is fussak. Szinte minden részében a hálózatnak vagy tanításnak javítottam ki implementációs hibákat vagy elírásokat amelyeket ezen az adathalmazon való gyors tesztelés miatt voltam képes diagnosztizálni. A hibák között volt a modellnek a nem tanulása, nem konvergálása, divergens veszteség függvény eredmények és sok apró architektúrai hiba.



5-19. ábra. MNIST adathalmazon, Konfig E által generált 16 random kép.



5-20. ábra. MNIST adathalmazon, Konfig F diszkriminátorának veszteségfüggvény értékeknek(WGAN-GP) epochonkénti ábrázolása.

Legtöbb mély tanulást alkalmazó modell rendelkezik egy objektív veszteség függvénnyel amely korrelációban van a modellnek a teljesítményével. GAN hálózatok tesztelése és az eredmények kiértékelése egy nagyon rettentő feladatot jelent, mivel nem objektív veszteség függvénnyel rendelkeznek. Emiatt nagyon nehéz objektívan kiértékelni a tanítási folyamat közben látható fejlődést, a generátor által készített képek minőségét és a képek között található variációt. Szerencsére az évek során volt néhány papír mint például a GAN Evaluation Measures [46], amely átvizsgálja a "quantitative" vagy mennyiségi és "qualitative" vagy minőségi értékelési módszereket. A mennyiségi és minőségi értékelési módszerek mellett még létezik a manuális értékelési módszer, amelyet legtöbbször felhasználtam a modell implementálása és tanítása közben.

A manuális értékelési módszer során a generátor által készített képeket kell átvizsgálni. A saját módszerem az általában a tanítás folyamat közben történt, ahol tanítási epochok legvégén generáltam 16 képet, amelyeket lementek lokális tárolóegységre, hogy később összehasonlítsam a különböző epochok által generált képeket. Az epochok végén generált képeket, pedig mindig ugyan azzal a generátorba beadott zaj input vektorral generáltam, hogy az eredményeket könnyebb legyen kiértékelni. Ez a módszer viszont nagyon szubjektív és tudni kell hogy egy adott ponton mit várunk el a modelltől és limitált is, mivel nem generálok több mint 9 mintát, emellett pedig ha többet is generálnék mégiscsak limitál az idő, hogy mennyit vagyok képes átnézni. Viszont egy nagyon jó kezdőpont a modellnek a kiértékelésére, de persze nem egy jó megoldás mint végső kiértékelés. Pont erre a feladatra léteznek a mennyiségi és minőségi értékelési módszerek. A képek vizsgálása mellett még alkalmaztam egy manuális tesztelési módszert, ahol felhasználtam a Keras által generált modell tervrajzokat, hogy diagnosztizáljak különböző architektúrai problémákat. A modell tervrajzokon lehet látni mindegyik modell elnevezését, az adatok bemenetének és kimenetének formáját. A tervrajzoknak a vizsgálásával

sokszor képes voltam diagnosztizálni architektúrai problémákat, mint például a rétegek nem a megfelelő pozícióban vannak, vagy esetleg a bemeneti és kimeneti adatoknak a formája nem megfelelő. A tervrajzokat lehet látni a 5-17 ábrán, habár itt csak 8x8-as felbontásról van szó, minél nagyobb a modell annál komplexebb is lesz.



5-21. ábra. Manuális tesztelésre példák az MNIST adathalmazon

Tesztelési módszer alapjaként az FID és IS számszerűsítő módszereket használtam, mivel találkoztam velük sok GAN papírban. Ezeket a tesztelési módszereket csak a manuális tesztelés után kezdtem el használni, amikor már rendesen működött a modell, tehát hatalmas artifactok vagy tanítási problémák egyikben sem voltak jelen. Ezeknek a módszereknek a használására egy úgynevezett InceptionV3, előre feltanított modellt kell használni, amely [299,299,3] méretű képeket vár bemenetként és képes a tulajdonságait értékelni a generált képeknek. Mivel a tesztelési adathalmazom 10000 méretű, ehhez nagy mennyiségű RAM-ra van szükségem, amely sajnos nem lehetséges számomra. Ezért a számítások során betöltöm az egész adathalmazt mielőtt még átméretezném [299,299,3] felbontásra, összekeverem az elemeket és kiválasztom az első 2000 elemet. Ekkora méretre még rendelkezek elegendő memóriával, ezt a folyamatot pedig lefuttatom 10-szer, hogy az eredmények ne legyenek túl randomok, mivel lehetséges hogy ha csak egyszer futtatnám le, akkor egyik konfigurációnak az "alsó" eredményei egyenlőek lehetnek egy másik "felső" eredményeivel, de az átlaga pedig sokkal különbözőbb. Az itt leírt módszertant FID-2K és IS-2K-nak neveztem el, és ezt is fogom használni, az eredmények kiértékelésére, látható is 5-22 ábrán.

Ki is értékelem, hogy az 5-22 ábrán látható eredmények mit is jelentenek. Az összes konfiguráció amelyet láthatunk az a Konfig B, alap konfiguráción alapul és a konfigok utáni látható leírás pedig az apró változtatások amelyek az alap konfigurációtól eltér. Fontos megemlíteni, hogy a GAN hálózatok és a tensorflow is, egy nagyon sztochasztikus környezet. Különböző futások között akár még 10% különbség is lehet minőségben, akármi-lyen alterációk nélkül. Nekem sajnos nem lesz időm többször lefuttatni mindent a legjobb lehetséges eredmények eléréseért, ezért tesztelés során csak akkor fogok valamilyen kon-

Konfiguráció	FID-2K ↓	IS-2K ↑	Tanítási idő ↓
Tanítási adathalmaz	7.27	1.367 ± 0.035	-
Konfig A: DCGAN	36.65	1.118 ± 0.027	8 óra 18 perc
Konfig B: Alap MORKGAN	26.06	1.253 ± 0.017	2 óra 30 perc
Konfig C: Cross-entropy	28.04	1.308 ± 0.023	1 óra 32 perc
Konfig D: 5 Diszkriminátor lépés, kötegenként	30.05	1.262 ± 0.020	8 óra 4 perc
Konfig E: Nearest neighbor interpolation	15.24	1.336 ± 0.026	2 óra 48 perc
Konfig F: "Lassú" tanítás	25.09	1.240 ± 0.021	3 óra 47 perc
Konfig G: Nearest+Magasabb konv filterek+hosszabb tanítás	13.55	1.464 ± 0.016	21 óra 22 perc

5-22. ábra. MNIST adathalmazon, FID-2K, IS-2K és tanítási idő elemzésének eredményei. FID-2K és tanítási idő esetén az érték minél alacsonyabb annál jobb, IS-2K esetén pedig minél magasabb.

figurációt használni ha láthatóan jobb eredményeket ér el.

Nem generált képekre, nem a legjobban működnek ezek a számszerűsítési technikák, minden módszernek megvannak a gyenge pontjai, de képesek vagyunk ezeket kiértékelni az adathalmazon is, amely egy egész jó lehetséges hosszú távú célt adhat, hogy milyen értékeket szeretnék elérni. Pontosan ezért ezeket a számszerűsítési technikákat végig futtattam a tanítási adathalmazon is, hogy adjon valamilyen kiinduló pontot az eredményeimnek. Ez látható is az IS-2K eredményeknél ahol a konfiguráció G "jobb" eredményt ért el mint a tanítási adathalmaz. A DCGAN modellt is hozzáadtam, hogy adjon egy kis perspektívát a prototípus modellről.

Konfiguráció C-ben megpróbáltam egy másfajta veszteségi függvényt használni mint a WGAN-GP, hátha jobb eredményeket kapnék vele, egy időben pedig divergens vagy nan eredményeket kaptam tanítás közben és azt gondoltam, hogy lehet a WGAN-GP implementációmban lenne probléma, de ez nem lett igaz. Tanítása viszont nagyon gyors volt a Konfiguráció C-nek de az irodalomkutatásban leírt okok miatt továbbá is használni fogom a WGAN-GP veszteségfüggvényt.

Konfiguráció D-ben egy olyan módszert próbáltam ki amely az eredeti PGGAN papírban is benne van, ahol tanítás során egy tanítási lépésen belül a diszkriminátor adott x -szer lesz tanítva nem csak egyszer mint egy normális implementációnál. Itt $x = 5$ hiperparamétert használtam mint az eredeti papírban, de a magasabb tanítási idő és rosszabb minőségű

eredmények miatt nem fogom használni. Később viszont ez azt fogja jelenteni, hogy a diszkriminátor nagyobb felbontáson képes lesz a gradienseinek a "felrobbanására", de ez idővel helyre fog állni.

Konfig *E*-ben pedig, amelyet már ezelőtt megemlítettem az adathalmaz implementációnál, leteszteltem a különbséget a "Nearest Neighbor interpolation" és "Bilinear Interpolation" használata között, a generátorban és az adathalmaz átméretezésekor. Nagyon egyértelműen a "Nearest Neighbor interpolation" adott jobb eredményeket ezért használni is fogom az FFHQ adathalmaznál is.

Konfig *F*-ben, a "lassú" tanítás az a kötegek hiperparaméter csökkentését jelenti. Jelenleg a StyleGAN papír alapján ajánlott köteg méreteket kezdtem el használni, amelyet csökkentenem is kellett a VRAM hiány miatt, ahol a kezdő 4x4-es felbontáson 128 képet dolgozunk fel a 16 helyett amely a PGGAN papírban volt. Idő optimalizálás miatt kezdtem el használni a StyleGAN implementáció által használt köteg hiperparamétereket és összehasonlítottam itt, hogy milyen implikációi vannak a képek minőségére és futási időre. FID-2K értékei 4%-al jobbak, IS-2K értékei pedig nagyon apró különbségeket tartalmaz. A minőségi értékek eltérése nagyon egyszerűen lehet a sztochasztikus környezet miatt is, ezért nem számítom szignifikáns különbségnek. Tanítási időben viszont közel 50%-al lassabb a Konfig *F* ezért, ezentúl is a StyleGAN hiperparamétereit fogom használni.

A Konfig *E* tesztelése után, egyértelművé vált, hogy ezt fogom alapul venni a további teszteléshez, a Konfig *G* pontosan ezt veszi alapként és emellett pedig a konvolúciós filtereket megdupláztam, mint a StyleGAN és PGGAN papírokban és kétszer annyi képen tartott egy modell csere. Ezek alapján pedig sikerült elérni a legmagasabb eredményeket az eddigi összes konfigurációból. Viszont, jelentős tanítási idő különbséget eredményezett, de mivel a szakdolgozatom célja a minőségi képek generálása, ezért használni fogom ezeket a hiperparamétereket.

5.4. Modell tanítása FFHQ adathalmazon

5.4.1. Tanítási adatok

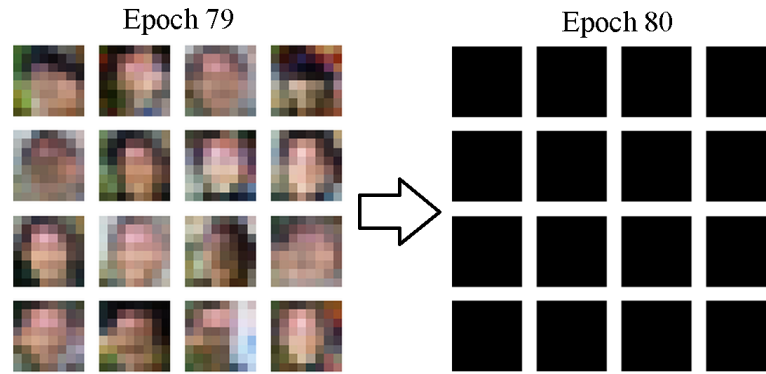
A tanítás az FFHQ adathalmazon, már a manuális tesztelési folyamatban is elkezdődött és hasonlóan sok architektúrai problémát ki is tudtam javítani miatta. Itt viszont nem

Generátor	Akt. Fgvny.	Kimenet	Paraméterek	Diszkriminátor	Akt. Fgvny.	Kimenet	Paraméterek
Zaj vektor	-	1 x 1 x 512	-	Bemeneti kép	-	128 x 128 x 3	-
Conv 4 X 4	LReLU	4 x 4 x 512	4.2M	Conv 1 X 1	LReLU	128 x 128 x 128	512
Conv 3 X 3	LReLU	4 x 4 x 512	4.19M	Conv 3 X 3	LReLU	128 x 128 x 128	148K
Upsample	-	8 x 8 x 512	-	Conv 3 X 3	LReLU	128 x 128 x 256	295K
Conv 3 X 3	LReLU	8 x 8 x 512	2.36M	Downsample	-	64 x 64 x 256	-
Conv 3 X 3	LReLU	8 x 8 x 512	2.36M	Conv 3 X 3	LReLU	64 x 64 x 256	590K
Upsample	-	16 x 16 x 512	-	Conv 3 X 3	LReLU	64 x 64 x 512	1.18M
Conv 3 X 3	LReLU	16 x 16 x 512	2.36M	Downsample	-	32 x 32 x 512	-
Conv 3 X 3	LReLU	16 x 16 x 512	2.36M	Conv 3 X 3	LReLU	32 x 32 x 512	2.36M
Upsample	-	32 x 32 x 512	-	Conv 3 X 3	LReLU	32 x 32 x 512	2.36M
Conv 3 X 3	LReLU	32 x 32 x 512	2.36M	Downsample	-	16 x 16 x 512	-
Conv 3 X 3	LReLU	32 x 32 x 512	2.36M	Conv 3 X 3	LReLU	16 x 16 x 512	2.36M
Upsample	-	64 x 64 x 512	-	Conv 3 X 3	LReLU	16 x 16 x 512	2.36M
Conv 3 X 3	LReLU	64 x 64 x 256	1.18M	Downsample	-	8 x 8 x 512	-
Conv 3 X 3	LReLU	64 x 64 x 256	590K	Conv 3 X 3	LReLU	8 x 8 x 512	2.36M
Upsample	-	128 x 128 x 256	-	Conv 3 X 3	LReLU	8 x 8 x 512	2.36M
Conv 3 X 3	LReLU	128 x 128 x 128	295K	Downsample	-	4 x 4 x 512	-
Conv 3 X 3	LReLU	128 x 128 x 128	148K	Minibatch stddev	-	4 x 4 x 513	-
Conv 1 X 1	Linear	128 x 128 x 3	387	Conv 3 X 3	LReLU	4 x 4 x 512	2.364M
Összes tanítható paraméter			27.13M	Conv 4 X 4	LReLU	1 x 1 x 512	4.19M
				Fully connected	Linear	1 x 1 x 1	513
				Összes tanítható paraméter			22.93M

5-23. ábra. Progresszívan növekvő hálózat rétegzése, FFHQ-128 adathalmazra, [128,128,3] képek generálására

a nagy minőségű 1024x1024-es képes verzióját használtam hanem a kisebb 128x128-es "thumbnail"-eket tartalmazó adathalmazt amit FFHQ-128-nak fogok nevezni. Erre azért volt szükség mivel sokkal több lesz a tanítási idő, a képek betöltése miatt és mivel én maradtam a [128,128,3] képek generálásánál, ezért nem létfontosságú a nagyobb FFHQ-1024 adathalmaz.

Emellett pedig egy architektúrai hibába futottam bele, látható az 5-24 ábrán, amelyet nem voltam képes teljesen diagnosztizálni. Amelyet viszont képes voltam dedukálni az a nagyobb felbontásokon történő model failure, ahol hirtelen divergens értékeket produkált a diszkriminátor veszteségfüggvénye. Ezt manuális hiperparaméter optimalizálással sikerült elkerüljem, ahol az utolsó réteg aktivációs függvényét megváltoztattam $\tanh(x)$, lineárisra. Még egy újítást implementáltam a StyleGAN papírból, amelyet "variable learning rate", VRR vagy változó tanítási rátának nevezek. A változó tanítási ráta folyamat során, adott felbontásokon az Adam optimalizációs eljárásának a tanítási ráta hiperparaméterét növelem.



5-24. ábra. Divergens veszteségfüggvény eredménye

5.4.2. Eredmények kiértékelése

Konfiguráció	FID-2K ↓	IS-2K ↑	Tanítási idő ↓
Tanítási adathalmaz	17.53	1.042 ± 0.006	-
Konfig G: Magas filterek	54.08	1.046 ± 0.004	6 nap 14 óra
+ 30k adathalmaz	46.63	1.064 ± 0.004	5 nap 3 óra
Konfig H: 5 diszkr lépés + 1200k kép/model csere	49.08	1.041 ± 0.004	9 nap 9 óra

5-25. ábra. FFHQ-128 adathalmazon, FID-2K, IS-2K és tanítási idő elemzésének eredményei. FID-2K és tanítási idő esetén az érték minél alacsonyabb annál jobb, IS-2K esetén pedig minél magasabb.

Az IS-2K és FID-2K elemzéseket az FFHQ-128 adathalmazon tanított modellekre is elvégeztem, amely látható az 5-25 ábrán. Az itt található Konfig G ugyan az a magas filter konfiguráció, amelyet az MNIST tesztelési szekcióban lehet találni. Egy változtatást próbáltam ki, ahol az FFHQ-128 adathalmazból csak 30000 képből tanul a modell, nem a teljes 60000-ből. Eredeti PGGAN implementációban is csak egy 30000 képes adathalmazt használtak. Erre a lépésre azért volt szükség, mivel láthatóan nem volt képes rendesen megtanulni az ilyen nagy adathalmaznak a struktúráit. Nem adott rossz eredményeket, de nem optimálisakat ezért lefeleztem a képeket amelyből tanulnia kell. Minőségben jó eredményt is adott ez változtatás lehet is látni az 5-25 ábrán. Ez viszont azt fogja jelenteni, hogy kevesebb variáció lesz a képek között mintha a teljes 60000 képből tanulna. Ennek ellenére is ez lett az alap konfigurációm amelyet deployolás közben használni fogok. Az alap konfigurációból lehet látni 16 általam kiválasztott képet az 5-26 ábrán, amely mutatja mennyire sok fajta arcképet képes generálni a hálózat.

Volt időm még egy konfigurációnak lefuttatására, a Konfig *H*-ra, amelynek az eredményei jól demonstrálják a fontosságát a hosszabb tanításnak. Ahogy ez már előbb ki lett mutatva az extra diszkriminátor lépések, az én hálózatomban rosszabb minőségű eredményeket produkál, de mivel hosszabb tanításon esett át, ezért az eredmények jobb minőségűek lettek.



5-26. ábra. FFHQ-128 adathalmazon, manuálisan kiválasztott 128^2 felbontáson Konfig *G*-30k adathalmaz által generált képek.

5.5. Feltanított modell használata webapplikációban

A végső MORPGAN modellt sikeresen feltanítottam, feltanítás után generáltam 10000 képet, eredményeit kiértékeltem IS-2k és FID-2k elemzésekkel. Viszont ezen felül én még szerettem volna ezt a modellt felhasználni valamilyen applikációban is. Eredetileg MVVM alapú applikációban gondolkoztam, de rövid kutatás után találkoztam a Flask web framework-el, amely ugyan úgy pythonban lett írva, ezért az implementálás nagyon egyszerű lesz. Eléggé primitív framework, sokszor "micro web framework"-nek nevezik,

de egy nagyon egyszerű weboldal megoldására tökéletes, ahol csak demonstrálni szeretném a modellt betöltését és generálását.

Még van néhány optimalizációs lépés, amelyet deployment előtt el kéne végezni. Az egész MORKGAN hálózatnak a súlyai mentésre kerülnek a tanítási procedura során. Ezt checkpoint rendszerként használtam tanítás során, meg a gyors betöltéshez, ha generálni szeretnék, de ez a súlyokat tartalmazó fájl sok olyan súly értékkel is rendelkezik, amelyek köze sincsen a generáláshoz. Az így keletkezett .ckpt (checkpoint fájl) mérete közel 600 MB. Külön ezért lementettem csak a generátornak a súlyait, és a betöltés során az egész diszkriminátor modellt el lesz hagyva, mivel már nincsen szükség rá. Ezután a procedura után pedig a súly fájl mérete mindössze 103 MB lesz, amely hatod akkora mint az eredeti. Emellett képes minden számomra szükséges funkciót elvégezni. Ha ezt a fájlt még kisebbre szeretném lebontani akkor egy úgynevezett "neural network pruning" avagy neurális hálózat metszés folyamaton keresztül lehetőség lenne. Ez a folyamat kitöröl paramétereket a hálózathoz, amely során az a célja, hogy ne romoljon a hálózat pontossága, de növelje a hatékonyságát.

A weboldal backendjének elindulásakor mindig felépítem és betöltöm a súlyokat. Ha egy felhasználó pedig megnyitja a weboldalt akkor egy új kép generálódik mindig. Ha frissíti az oldalt is egy új kép lesz. Sajnos ennél többet nem volt már időm rajta dolgozni, de demonstrálni képes a modellem működését.

6. Továbbfejlesztési lehetőségek

Összességében boldog vagyok a hálózat eredményeivel, de bőven lenne mit továbbfejleszteni, mivel ez egy nagyon komplex téma. Ahogy ezt már említettem a szakdolgozatom során, nagyon kulcsfontosságú az ilyen hálózatoknál a modell mélysége és emiatt egyben a hardver amely elérhető számomra. Egyszerűen kijelenthető, hogy sokkal jobb minőségű és értékű eredményeket tudtam volna elérni, ha erősebb hardverrel rendelkeztem volna. Linux rendszerre váltás is sokat segíthetett volna, több szoftver használata miatt, meg az említett VRAM gond miatt. Létezik néhány automatikus hiperparaméter optimalizáló szoftver amelyet csak Linux disztribúcióra találtam, példaként.

A hiperparaméterek optimalizálásának folytatásával és a modell továbbtanítása/felbontás növelése is egy nagyon jó kiindulópont lenne a modellnek a továbbfejlesztésére. Mivel a kód oldala úgy volt megírva hogy 1024x1024-ig felskálázható a hálózat, ez csak számítási idő kérdése. Hasonló módon az FFHQ-1024 adathalmaz használata az FFHQ-128 helyett is jobb eredményeket hozott volna, amely közelebb is vihetné state-of-the-art megoldásokhoz. Nagyon sok kutatási papír létezik a témában, sok újítás a StyleGAN vagy StyleGAN2 modellből, ide is implementálható lehetne. Egyik ilyen a style transfer és AdaIN normalizáció lenne, amely ide is beilleszthető lehetne. Architektúra továbbkutatása is jó eredményeket adhat, nincsen még túloptimalizálva. StyleGAN2 is ezért váltott a progresszív növekedésről, de most számomra az időhiány miatt nagyon jó döntés volt, a PGGAN alapnál maradni.

A végső MORCKGAN hálózat képes nagyon szép képek generálására, de sajnos átlagosan nem ad annyira minőségi képeket. Ezt próbáltam is diagnosztizálni, de nem jutottam egy egyértelmű válaszra, ezt még továbbá kellene vizsgálni. Egyértelműen valahol a tanítási folyamat során van valamilyen meghibásodás, én csak a veszteségfüggvényre tudtam gondolni. Kicsit továbbolvasás során találkoztam az alábbi papírral "Which Training Methods for GANs do actually Converge?" [47] amely átvizsgál néhány alternatívát a WGAN-GP helyett. A papír alapján, a veszteségfüggvény váltása R_1 vagy R_2 regularizációra, egy jó lépés lenne ennek megoldására.

Néhány apróság amelyet még lehetne továbbfejleszteni, az a mód ahogy kezelem az adathalmazokat. Előre elkészíteni TFRecord fájlokat és azokat használni, apró előnyt képes adni. Ez meg is oldaná az egyik problémát, MNIST adathalmaz előkészítése során az előkészítési folyamat lassú volt.

Hivatkozások

- [1] N. Jamil, S. Lqbal, and N. Iqbal, „Face recognition using neural networks,” in *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, pp. 277–281, IEEE, 2001.
- [2] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, „Show, attend and tell: Neural image caption generation with visual attention,” in *International conference on machine learning*, pp. 2048–2057, PMLR, 2015.
- [3] R. Zhang, P. Isola, and A. A. Efros, „Colorful image colorization,” in *European conference on computer vision*, pp. 649–666, Springer, 2016.
- [4] A. Mordvintsev, C. Olah, and M. Tyka, „Inceptionism: Going deeper into neural networks,” 2015.
- [5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [6] X. Yu, J. Huang, S. Zhang, and D. N. Metaxas, „Face landmark fitting via optimized part mixtures and cascaded deformable model,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 11, pp. 2212–2226, 2015.
- [7] P. R. Prucnal, A. N. Tait, M. A. Nahmias, T. F. De Lima, H.-T. Peng, and B. J. Shastri, „Multiwavelength neuromorphic photonics,” in *CLEO: Applications and Technology*, pp. JM3M–3, Optical Society of America, 2019.
- [8] D. P. Kingma and M. Welling, „Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [9] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, „Pixel recurrent neural networks,” in *International Conference on Machine Learning*, pp. 1747–1756, PMLR, 2016.
- [10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, „Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.
- [11] G. E. Hinton and R. S. Zemel, „Autoencoders, minimum description length, and helmholtz free energy,” *Advances in neural information processing systems*, vol. 6, pp. 3–10, 1994.

- [12] B. Cheung, J. A. Livezey, A. K. Bansal, and B. A. Olshausen, „Discovering hidden factors of variation in deep networks,” *arXiv preprint arXiv:1412.6583*, 2014.
- [13] H. G. Susskind JM, Anderson AK, „The toronto face database,” *Department of Computer Science, University of Toronto, Toronto, ON, Canada, Tech Rep 3*, 2010.
- [14] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, „Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications,” *arXiv preprint arXiv:1701.05517*, 2017.
- [15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, „Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [16] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, „Conditional image generation with pixelcnn decoders,” *arXiv preprint arXiv:1606.05328*, 2016.
- [17] A. Razavi, A. van den Oord, and O. Vinyals, „Generating diverse high-fidelity images with vq-vae-2,” in *Advances in neural information processing systems*, pp. 14866–14876, 2019.
- [18] M. Mirza and S. Osindero, „Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.
- [19] A. Radford, L. Metz, and S. Chintala, „Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
- [20] M. Arjovsky, S. Chintala, and L. Bottou, „Wasserstein generative adversarial networks,” in *International conference on machine learning*, pp. 214–223, PMLR, 2017.
- [21] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, „Improved training of wasserstein gans,” *arXiv preprint arXiv:1704.00028*, 2017.
- [22] T. Karras, T. Aila, S. Laine, and J. Lehtinen, „Progressive growing of gans for improved quality, stability, and variation,” *arXiv preprint arXiv:1710.10196*, 2017.

- [23] Z. Liu, P. Luo, X. Wang, and X. Tang, „Deep learning face attributes in the wild,” in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- [24] T. Karras, S. Laine, and T. Aila, „A style-based generator architecture for generative adversarial networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.
- [25] X. Huang and S. Belongie, „Arbitrary style transfer in real-time with adaptive instance normalization,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1501–1510, 2017.
- [26] V. Dumoulin, J. Shlens, and M. Kudlur, „A learned representation for artistic style,” *arXiv preprint arXiv:1610.07629*, 2016.
- [27] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, „Analyzing and improving the image quality of stylegan,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119, 2020.
- [28] B. Kalman and S. Kwasny, „Why tanh: choosing a sigmoidal function,” in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 4, pp. 578–581 vol.4, 1992.
- [29] R. A. Dunne and N. A. Campbell, „On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function,” in *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, vol. 181, p. 185, Citeseer, 1997.
- [30] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, „Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, p. 3, Citeseer, 2013.
- [31] P. Ramachandran, B. Zoph, and Q. V. Le, „Searching for activation functions,” *arXiv preprint arXiv:1710.05941*, 2017.
- [32] S. K. Kumar, „On weight initialization in deep neural networks,” *arXiv preprint arXiv:1704.08863*, 2017.
- [33] X. Glorot and Y. Bengio, „Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

- [34] K. He, X. Zhang, S. Ren, and J. Sun, „Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.
- [35] L. Chen, *Deep Learning and Practice with MindSpore*. Springer Nature, 2021.
- [36] S. Ioffe and C. Szegedy, „Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pp. 448–456, PMLR, 2015.
- [37] S. Ioffe, „Batch renormalization: Towards reducing minibatch dependence in batch-normalized models,” *arXiv preprint arXiv:1702.03275*, 2017.
- [38] D. P. Kingma and J. Ba, „Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [39] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, „Comparative study of deep learning software frameworks,” *arXiv preprint arXiv:1511.06435*, 2015.
- [40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, „Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.
- [41] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, „Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.
- [42] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, „TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.

- [43] F. Chollet *et al.*, „Keras.” <https://keras.io>, 2015.
- [44] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, „Array programming with NumPy,” *Nature*, vol. 585, pp. 357–362, Sept. 2020.
- [45] A. Odena, C. Olah, and J. Shlens, „Conditional image synthesis with auxiliary classifier gans,” in *International conference on machine learning*, pp. 2642–2651, PMLR, 2017.
- [46] A. Borji, „Pros and cons of gan evaluation measures,” *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.
- [47] P. Dhar, A. Bansal, C. D. Castillo, J. Gleason, P. J. Phillips, and R. Chellappa, „How are attributes expressed in face dcnn’s?,” *CoRR*, vol. abs/1910.05657, 2019.

Ábrajegyzék

2-1. TSPM és OPM modellek, amelyek a "Facial landmark modellek" családjába tartozik és képesek detektálni az emberi arcokon található face landmark pontokat(piros) és lokális foltokat(kék) [6]	4
2-2. Mély tanulás számításának a növekedése 2012-től kezdve [7]	6
2-3. A bal oldali oszlop egy minta a tesztelési adathalmazból, tőle jobbra pedig lehet látni a generált arcokat, különböző arckifejezésekre, amelyet a [12] papírban egy VAE hálózattal a TFD [13] adathalmazon készítettek el.	9
2-4. A bal oldali oszlop egy minta a tesztelési adathalmazból, tőle jobbra pedig lehet látni új képeket ugyan arról az arcról, csak más pozíciókból, amelyet a [16] papírban egy conditional PixelCNN generálta le.	12
2-5. VQ-VAE-2 [17] által generált 1024x1024 felbontású képek.	12
2-6. GAN modell által generált arcok a TFD [13] adathalmazon.[10]	14
2-7. DCGAN modell által generált arcokon végzett vektor műveletek az eredeti DCGAN cikkben [19].	16
2-8. A folyamatosan növekvő GAN avagy a Progressive Growing of GAN [22] modell rétegjeinek a bővítése, amelyet tanítás folyamán végez.	17
2-9. A folyamatosan növekvő GAN avagy a Progressive Growing of GAN [22] modell által generált arcképek.	18
2-10. A StyleGAN [24] modell által generált arcképek, az FFHQ adathalmazon.	19
2-11. A StyleGAN2 [27] modell által generált arckép, az FFHQ adathalmazon.	19
4-12. A rendszer fejlesztésének tervezett menete, Gantt-diagrammon ábrázolva	35
4-13. A rendszer tanításának a szekvenciadiagramja	36
4-14. Egy feltanított rendszernél, arcképek generálásának a szekvenciadiagramja	37
5-15. Prototípus generátor rétegzésének az összefoglalása	40
5-16. Prototípus diszkriminátor rétegzésének az összefoglalása	40
5-17. Progresszívan növekvő generátor és diszkriminátor hálózat, 8^2 felbontásra, ahol 4^2 felbontáson kezdett el tanulni. Pirossal jelölt blokkok a generátor esetén az input blokkhoz tartoznak, diszkriminátor esetén pedig a fromRGB blokkhoz. A zölddel jelöltek pedig az up- és downsample blokkok. Késsel jelöltek pedig a generátor esetén a toRGB, diszkriminátor esetén pedig az output. Ebben a modellben nem alkalmaztam weightscaling-et és pixelnormot minden rétegen mivel nem fért volna ki egy ábrára.	45

5-18. Progresszívan növekvő hálózat rétegzése, MNIST adathalmazra, [32,32,1] képek generálására	48
5-19. MNIST adathalmazon, Konfig <i>E</i> által generált 16 random kép.....	48
5-20. MNIST adathalmazon, Konfig <i>F</i> diszkriminátorának veszteségfüggvény értékeknek(WGAN-GP) epochonkénti ábrázolása.	49
5-21. Manuális tesztelésre példák az MNIST adathalmazon	50
5-22. MNIST adathalmazon, FID-2K, IS-2K és tanítási idő elemzésének eredményei. FID-2K és tanítási idő esetén az érték minél alacsonyabb annál jobb, IS-2K esetén pedig minél magasabb.	51
5-23. Progresszívan növekvő hálózat rétegzése, FFHQ-128 adathalmazra, [128,128,3] képek generálására	53
5-24. Divergens veszteségfüggvény eredménye.....	54
5-25. FFHQ-128 adathalmazon, FID-2K, IS-2K és tanítási idő elemzésének eredményei. FID-2K és tanítási idő esetén az érték minél alacsonyabb annál jobb, IS-2K esetén pedig minél magasabb.	54
5-26. FFHQ-128 adathalmazon, manuálisan kiválasztott 128^2 felbontáson Konfig <i>G</i> -30k adathalmaz által generált képek.....	55