Tapio Siitonen

Kuvien generointi neuroverkkojen avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

30.11.2022

Tiivistelmä

Tekijä:	Tapio Siitonen
Otsikko:	Kuvien generointi neuroverkkojen avulla
Sivumäärä:	31 sivua
Aika:	30.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaaja:	Lehtori Vesa Ollikainen

Generatiiviset koneoppimisen mallit pystyvät nykyisin generoimaan monimutkaisia ja fotorealistisia kuvia. Yksi tällainen koneoppimisen malli on generatiivinen kilpaileva verkosto (GAN). Työn tavoitteena oli toteuttaa sellainen generatiivinen kilpaileva verkosto, joka kykenee tuottamaan fotorealistisia kasvokuvia. Toisena tavoitteena oli perehdyttää lukija generatiivisen kilpailevan verkoston toimintaperiaatteisiin.

Insinöörityön teoriaosuudessa käytiin läpi koneoppimista, neuroverkkoja, konvoluutioverkkoja ja generatiivinen kilpaileva verkosto. Ennen toteutukseen menemistä esiteltiin myös erilaisia GAN-malleja, jotta saatiin parempi käsitys niiden potentiaalista.

Toteutusosiossa käytiin läpi kasvoja generoiva DCGAN-toteutus osa kerrallaan ja koodiesimerkkien tukemana. Lopputuloksena DCGAN pystyi generoimaan kasvokuvia, joista parhaat vaikuttivat aidoilta.

DCGAN-toteutuksen kouluttaminen todettiin aikaa vieväksi suhteutettuna generoituihin kuviin, jotka olivat melko pieniä ja huonoja laadultaan verrattuna edistyneempien GAN-mallien tuotoksiin. Toteutus antoi kuitenkin hyvän pohjan tuleviin koneoppimisen projekteihin.

Avainsanat: koneoppiminen, generatiivinen kilpaileva verkosto

Abstract

Author:	Tapio Siitonen
Title:	Generating Images with Neural Networks
Number of Pages:	31 pages
Date:	30 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisor:	Vesa Ollikainen, Senior Lecturer

Generative machine learning models are currently capable of generating complex and photorealistic pictures. One of the machine learning models is the generative adversarial network (GAN). The aim of this study was to implement a GAN that can generate photorealistic images of faces. The other aim of the study was to familiarize the reader with the principles of a generative adversarial network.

The theory part goes through machine learning, neural networks, convolutional networks, and generative adversarial networks. Before getting into the implementation, different kinds of GAN-models are also introduced to get a better understanding of their potential.

The implementation part of the study consisted of the face generating DCGAN implementation one part at a time and is supported by example code. In the end, the DCGAN could generate faces of which the best ones seemed real.

Training the DCGAN implementation was found to be too time consuming related to the photos that were quite small and lacking in quality compared to what other GAN implementations can produce. However, the implementation provided a good basis for future machine learning projects.

Keywords: Machine Learning, Generative Adversarial Network

Sisällys

Lyhenteet

ANN: *Artificial neural network*. Keinotekoiset neuroverkot koostuvat neuroneista ja niiden välisistä yhteyksistä. Algoritmeja käyttämällä ne voivat oppia datassa piileviä suhteita.

CGAN: *Conditional generative adversarial network*. Ehdollinen GAN generoi dataa perustuen haluttuun luokkaan.

CNN: *Convolutional neural network*. Konvoluutioverkko on keinotekoinen neuroverkko, joka kykenee erottamaan piirteitä kuten reunoja kuvista käyttämällä suodattimia.

DCGAN: *Deep convolutional generative adversarial network*. GAN, joka käyttää konvoluutiota luokittelijassa ja transponoitua konvoluutiota generaattorissa.

GAN: *Generative adversarial network*. Generatiivinen kilpaileva verkosto koostuu generaattorista ja luokittelijasta, jotka kilpailevat keskenään.

ProGAN: *Progressively growing generative adversarial network*. GAN, jonka generaattori ja luokittelija kasvavat hiljalleen luoden ensin pienen resoluution kuvia, jotka kasvavat jokaisella askeleella.

ReLU: *Rectified linear unit*. Neuroverkkojen käyttämä aktivointifunktio, joka palauttaa syötteen sen ollessa positiivinen ja nollan syötteen ollessa negatiivinen.

SRGAN: *Super resolution generative adversarial network*. GAN, joka ottaa pienemmän resoluution kuvan ja muodostaa siitä isomman resoluution kuvan.

1 Johdanto

Koneoppimista ja generatiivisia malleja hyödyntämällä pystytään nykyisin luomaan monimutkaisia ja fotorealistisia kuvia. Midjourney-tekoälyllä generoitu taideteos palkittiin sinisellä nauhalla Colorado State Fair -tapahtuman vuosittaisen taidekilpailun nousevien digitaalisten taiteilijoiden sarjassa elokuussa 2022 [1]. New Yorkissa sen sijaan vuonna 2018 tekoälyn generoima taideteos myytiin huutokaupassa 432 500 dollarilla [2].

Yksi näistä generatiivisista malleista on generatiivinen kilpaileva verkosto (GAN). Tämän insinöörityön tavoitteena on toteuttaa sellainen generatiivinen kilpaileva verkosto, joka kykenee generoimaan fotorealistisia kasvokuvia. Toisena tavoitteena on perehdyttää lukija generatiivisen kilpailevan verkoston toimintaperiaatteisiin.

Ensiksi teoriaosuudessa tutustutaan lyhyesti koneoppimisen käsitteeseen ja sen eri muotoihin. Tämän jälkeen perehdytään neuroverkkojen (ANN) rakenteisiin ja oppimisprosessiin. Konvoluutioneuroverkkojen (CNN) osuudessa käydään läpi konvoluutiokerroksen ja transponoidun konvoluutiokerroksen toimintaperiaatteet. Teoriaosuuden viimeisessä osassa käsitellään generatiivinen kilpaileva verkosto, DCGAN-arkkitehtuuri (deep convolutional generative adversarial network) ja generatiivisen kilpailevan verkoston mahdolliset heikkoudet.

Ennen toteutusosaa tutustutaan vielä erilaisiin GAN-malleihin, jotka ovat mahdollistaneet kuvien generointia eri näkökulmista sitten vuoden 2014. Tässä luvussa käydään läpi lyhyesti CGAN-, SRGAN-, CycleGAN-, ProGAN- ja StyleGAN-mallit, että saadaan laajempi kuva generatiivisten kilpailevien verkostojen potentiaalista.

Toteutusosassa käydään läpi yksi generatiivisen kilpailevan verkoston toteutus, joka noudattaa DCGAN-arkkitehtuuria. Toteutukseen tutustutaan koodiesimer-

kein, joiden ohessa pyritään perustelemaan tehtyjä ratkaisuja. Lopuksi arvioidaan vielä generoituja kuvia ja toteutetun mallin käytännöllisyyttä kuvien generoinnissa.

2 GAN koneoppimisessa

Koneoppiminen on tekoälyn osa-alue. Terminä se sai alkunsa Arthur Samuelin vuonna 1959 tehdystä tutkimuksesta, jossa hän kehitti tammea pelaavaa tekoälyä. Kyseinen tekoäly oppi pelaamalla itseään vastaan, mitkä pelilaudan tilanteet johtavat todennäköisemmin voittoon käyttämällä heuristista arviointifunktiota ja minimax-algoritmia. Tekoäly voittikin itseään tammimestariksi kutsuvan Robert Nealeyn vuonna 1962. [3; 4.]

Koneoppimisella viitataankin järjestelmiin, jotka parantavat suorituskykyään annetussa tehtävässä saamalla lisää dataa tai kokemusta. Tähän suorituskyvyn parantamiseen on eri lähestymistapoja, jotka voidaan karkeasti jakaa ohjattuun oppimiseen, ohjaamattomaan oppimiseen ja vahvistusoppimiseen. Myös käsitettä puoliohjattu oppiminen käytetään, kun on kyse osittain ohjatusta ja osittain ohjaamattomasta oppimisesta. [5.]

Ohjatun oppimisen esimerkkinä voisi olla järjestelmä, joka koulutetaan tunnistamaan kissoja ja koiria antamalla sille valmiiksi luokiteltuja kuvia kissoista ja koirista. Tämän jälkeen järjestelmän suoriutuminen mitataan katsomalla, kuinka hyvin se luokittelee uusia kuvia kyseisistä eläimistä. [4; 5.]

Ohjaamattomassa oppimisessa data on luokittelematonta ja tehtävänä on selvittää sen rakennetta esimerkiksi jakamalla samankaltaisia alkioita joukkoihin. Myös tässä insinöörityössä toteutettu GAN on yleensä osa ohjaamatonta oppimista. [4; 5.]

Vahvistusoppimisessa ei tarvita tietoaineistoja oppimiseen vaan se oppii toimimalla ympäristössä, jossa se saa palautetta hyvistä ja huonoista ratkaisuista. Esimerkiksi robottiauto voi oppia ajamaan tätä kautta. [4; 5.]

Puoliohjattu oppiminen voi olla hyvä valinta tilanteessa, jossa on olemassa dataa, josta vain pienempi osa on luokiteltua. Tällä pienemmällä osalla voidaan ohjata oppimista, joka auttaa löytämään yhtäläisyyksiä isommasta luokittelemattomasta datasta. [4.]

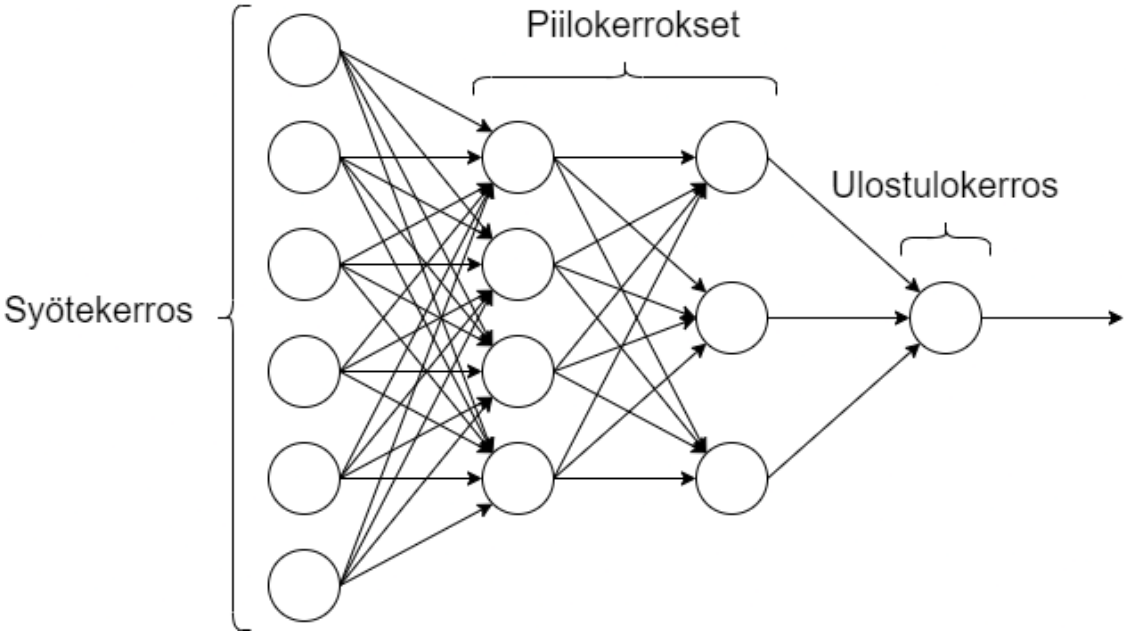
Edellä mainittuja lähestymistapoja toteutetaan käytännössä lukuisilla eri malleilla. Näitä ovat esimerkiksi klusterointi, lineaarinen ja logistinen regressio, neuroverkot ja päätöspuu. Tässä työssä keskitytään neuroverkkoihin, joita on myös työn toteutusvaiheessa käytetty.

2.1 Keinotekoiset neuroverkot

Keinotekoiset neuroverkot (ANN) on koneoppimisen osa-alue, joka on ottanut inspiraatiota biologisista neuroverkoista eli aivoista. Neuroverkkojen yhteydessä puhutaan usein myös syväoppimisesta, joka voidaan ymmärtää isompien monikerroksisien neuroverkkojen opettamisena.

Keinotekoiset neuroverkot koostuvat neuroneista, jotka saavat sisään yhden tai useamman arvon ja tuottavat yhden ulostuloarvon. Perseptroni on yksi tällainen 1950- ja 1960-luvulla kehitetty neuroni, jonka ulostulo on nolla tai yksi. Perseptronin jokaisella sisään tulevalla arvolla on painoarvo, joka ilmaisee sen tärkeyden perseptronin ulostulolle. Sisään tulevat arvot kerrotaan niiden painoarvoilla, jonka jälkeen ne summataan yhteen. Tätä kutsutaan painotetuksi summaksi. Jos tämä summa ylittää perseptronille asetetun raja-arvon, on ulostulona yksi. Muissa tapauksissa perseptroni antaa ulos arvon nolla. Toisin ilmaistuna perseptronin ulostulo lasketaan sisään tulevista arvoista koostuvan vektorin ja painoarvoista koostuvan vektorin pistetulona. Tähän pistetuloon lisätään harha (bias), joka ilmaisee, kuinka helposti perseptroni aktivoituu eli antaa ulos arvon yksi. Koska raja-arvoa ei ole, riittää aktivointiin tulos, joka on isompi kuin nolla. Todella suuri harha tarkoittaa silloin, että perseptroni aktivoituu helposti ja päinvastoin hyvin pieni harha tarkoittaa, että aktivointi on vaikea saavuttaa. [6.]

Neuroneista voidaan muodostaa erilaisia verkkoja, mutta tässä työssä keskitymme myötäkytkentäverkkoihin. Tällöin neuroneista muodostetaan kerroksia kuvan 1 osoittamalla tavalla. Informaatio liikkuu myötäkytkentäverkossa (feed-forward network) eteenpäin jokaisen kerroksen läpi tekemättä toistoja. Ensimmäinen kerros on syötekerros, jonka jälkeen on yksi tai useampi piilokerros. Viimeistä kerrosta kutsutaan ulostulokerrokseksi. Syötekerroksen ja ulostulokerroksen suunnittelu ovat usein suoraviivaisia. Esimerkiksi mustavalkoisen kuvan ollessa verkon syötteenä voidaan syötekerros muodostaa yhtä monesta neuronista kuin kuvassa on pikseleitä. Jokaisen syötekerroksen neuronin arvo laitetaan sitten vastaamaan yhtä kuvan pikselin intensiteettiä. Jos kuvat ovat esimerkiksi käsin kirjoitetuista numeroista, ja verkon tehtävänä on arvioida, onko kuvassa numero kolme, on yksi neuroni ulostulokerroksessa riittävä ilmaisemaan tuloksen. Piilokerrosten suunnittelua ei sen sijaan voida tiivistää helppoihin sääntöihin. Niiden rakenteen selvittäminen vaatii enemmän testausta, tai vaihtoehtoisesti valmiiden ratkaisuiden hyväksikäyttöä. [6.]

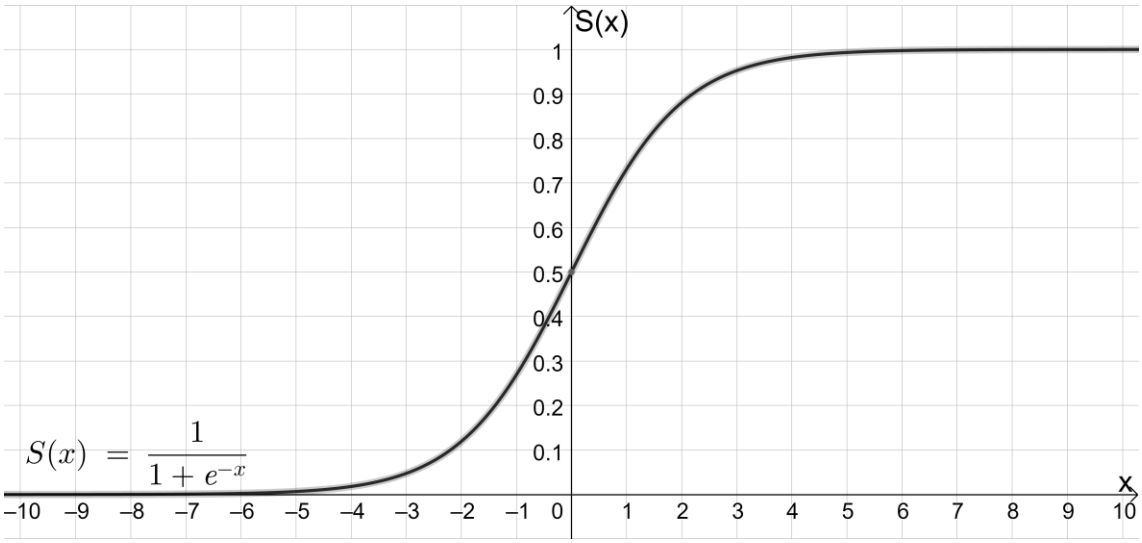


Kuva 1. Neljän kerroksen myötäkytkentäverkko, jossa on kaksi piilokerrosta.

Kuvasta 1 voidaan myös nähdä, että jokainen neuroni on aina yhteydessä jokaiseen seuraavan kerroksen neuroniin. Tällaisia kerroksia kutsutaan täysin kytketyiksi kerroksiksi. Neuroneilla on näistä yhteyksistä huolimatta vain yksi ulostuloarvo, kuten aiemmin kerrottiin. Tämä tarkoittaa sitä, että jokainen seuraavan kerroksen neuroni saa jokaisen edellisen kerroksen neuronin ulostuloarvon syötteenä.

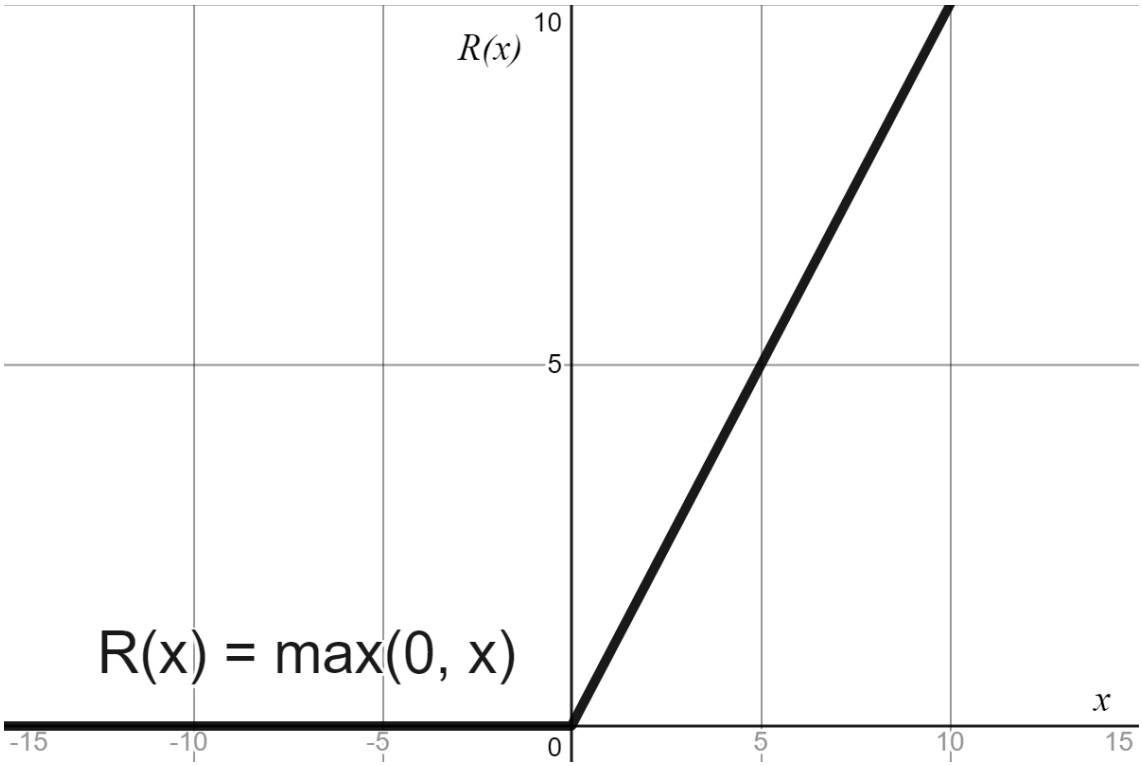
Nykyisin perseptroneja ei oikeastaan käytetä neuroverkkojen rakentamisessa. Perseptronien heikkoudet tulevat esille neuroverkkojen kouluttamisvaiheessa, jossa halutaan usein pienillä painojen tai harhan muutoksilla vaikuttaa neuroverkon lopputulemaan. Perseptronista ulos tuleva arvo ei kykene kuitenkaan pieniin muutoksiin, vaan se on aina nolla tai yksi. Tällaisella arvon vaihtumisella voi olla monimutkaisia seurauksia koko muun verkon käyttäytymisessä, mikä tekee perseptroneista koostuvan verkon kouluttamisesta hankalaa. Ongelman voi ratkaista käyttämällä perseptronien sijasta sigmoid-neuronia, jonka ulostulona on arvo nollan ja yhden välillä. Nyt pienet muutokset painoissa tai harhassa aiheuttavat vain pienen muutoksen neuronin ulostulossa. Lisäksi arvoilla, jotka ovat nollan ja yhden välillä, voidaan ilmaista enemmän asioita kuin pelkällä nollalla ja ykkösellä, kuten esimerkiksi kuvassa olevan pikselin intensiteettiä. [6.]

Sigmoid-neuronilla tarkoitetaan oikeastaan neuronia, joka käyttää kuvassa 2 olevaa sigmoidista aktivointifunktiota. Aktivointifunktio saa syötteenä neuronin sisään tulevan painotetun summan, johon on lisätty harha, ja laskee sen perusteella neuronin ulostuloarvon. Neuroverkko ilman aktivointifunktioita on pelkkä painotettujen summien ja harhojen yhteenlaskusta koostuva lineaarinen funktio, joka ei ole tarpeeksi monimutkainen vaativien epälineaaristen ongelmien ratkaisuun. Aktivointifunktion tärkein tehtävä onkin epälineaarisuuden lisääminen neuroverkkoon. [7.]



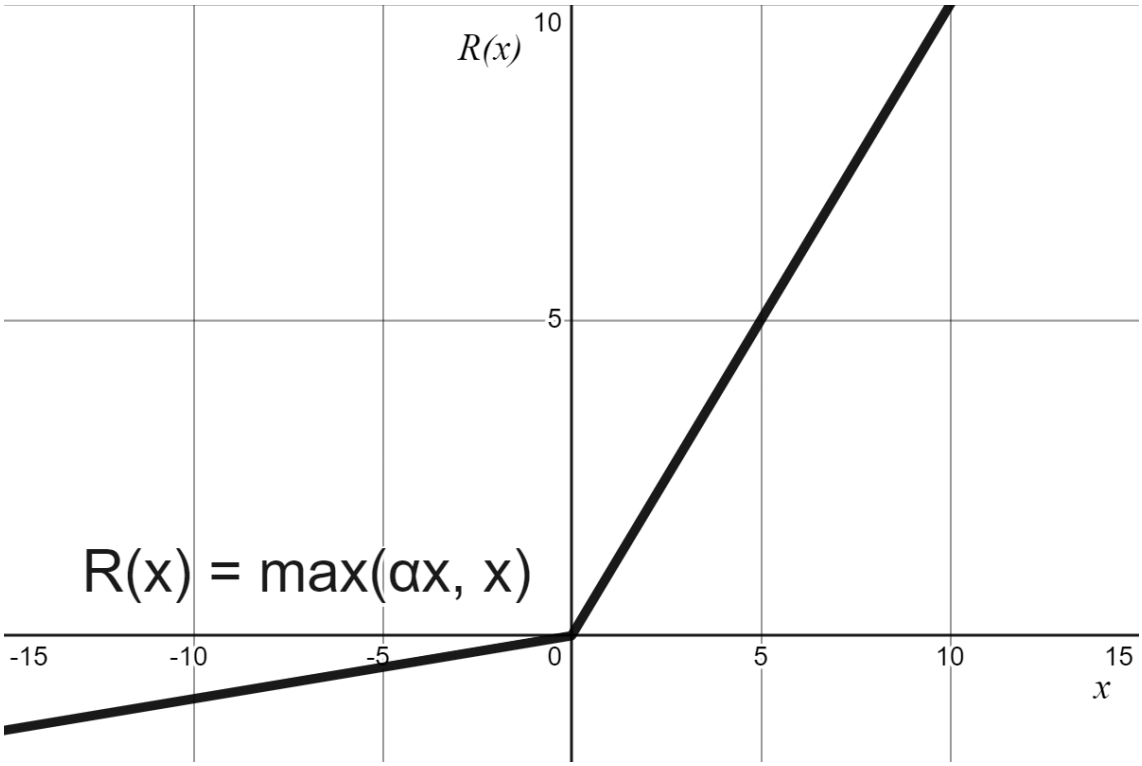
Kuva 2. Sigmoidi aktivointifunktio saa syötteen x ja palauttaa arvon S(x), joka on neuronin ulostuloarvo.

Useimmat nykyiset mallit eivät kuitenkaan käytä sigmoidista aktivointifunktiota syvissä neuroverkoissa, vaan sen on korvannut ReLU (rectified linear unit). ReLU-aktivointifunktio on ensinnäkin laskennallisesti kevyempi. Lisäksi sigmoidin aktivointifunktion ongelmana on, että suurilla ja pienillä syötteillä sen arvo on lähes vakio, jolloin sen merkitys koulutusvaiheessa on mitätön. ReLU-aktivointifunktiossa samankaltainen ongelma voi esiintyä vain nollaa pienemmillä syötteillä, kuten kuvasta 3 voi nähdä. ReLU palauttaa syötteen sen ollessa positiivinen ja nollan syötteen ollessa negatiivinen. [8.]



Kuva 3. ReLU-aktivointifunktio saa syötteen x ja palauttaa arvon R(x), joka on neuronin ulostuloarvo.

Leaky ReLU -aktivointifunktion käyttäminen korjaa mahdolliset ongelmat liittyen ReLU-aktivointifunktion nollaantumiseen negatiivisilla syötteillä [8]. Leaky ReLU alustetaan parametrilla alfa, joka määrittää sen jyrkkyyden negatiivisilla arvoilla. Yleensä alfan arvoksi määritellään melko pieni luku kuten kuvassa 4, jossa sen arvo on 0,1. Alfan arvon määrittäminen ykköseksi muuttaa funktion lineaariseksi, joka tekee koko funktiosta turhan.



Kuva 4. Leaky ReLU -aktivointifunktio toimii kuten ReLU, mutta se palauttaa negatiivisia arvoja nollaa pienemmillä syötteillä.

Neuroverkkojen oppimista on jo hieman sivuttu viittaamalla painoarvojen muutoksiin. Perehdytään seuraavaksi siihen, miten esimerkiksi kuvia luokittelevan verkon oppimisprosessi toimii. Verkko alustetaan ensin satunnaisilla painoarvoilla, jonka jälkeen sille syötetään kuvia luokiteltavaksi. Tarkoituksena oppimisessa on pyrkiä minimoimaan hukkafunktio (loss function), joka antaa numeerisen arvon siitä, kuinka hyvin verkko luokittelee kuvia. Hukkafunktion arvon ollessa iso se luokittelee kuvia huonolla tarkkuudella. [6.]

Hukkafunktion minimoimiseksi käytetään gradienttimenetelmää kuten stokastisen kaltevuuden laskemista, jonka jälkeen painoarvot tai harha muuttuvat niin, että tavallaan laskeudutaan askel kohti jotain minimiä. Tämän askeleen suurus riippuu määritellystä oppimisnopeudesta. Monikerroksisissa neuroverkoissa tämä yhdistetään vastavirta-algoritmiin, joka käy neuroverkkoa toiseen suuntaan lähtien ulostulokerroksesta. Toiveena on, että ottamalla tarpeeksi askelia

saavutetaan sellainen minimi, että neuroverkko luokittelee kuvat oikein riittävän hyvällä tarkkuudella.

Neuroverkkojen koulutuksessa kannattaa olla tietoinen ylisovittamisesta (overfitting). Tällä tarkoitetaan tilannetta, jossa neuroverkko suoriutuu esimerkiksi luokitteluongelmasta hyvin, kun kyseessä on harjoitteluun käytetty data, mutta suoriutuu heikosti uuden samankaltaisen datan luokittelusta. Ylisovittaminen johtuu siitä, että neuroverkko löytää harjoitteludatasta kohinaa, jonka se oppii olevan osa datan rakennetta. Tämä opittu kohina saattaa kuitenkin olla vain kyseisessä harjoitusjoukossa. [9.]

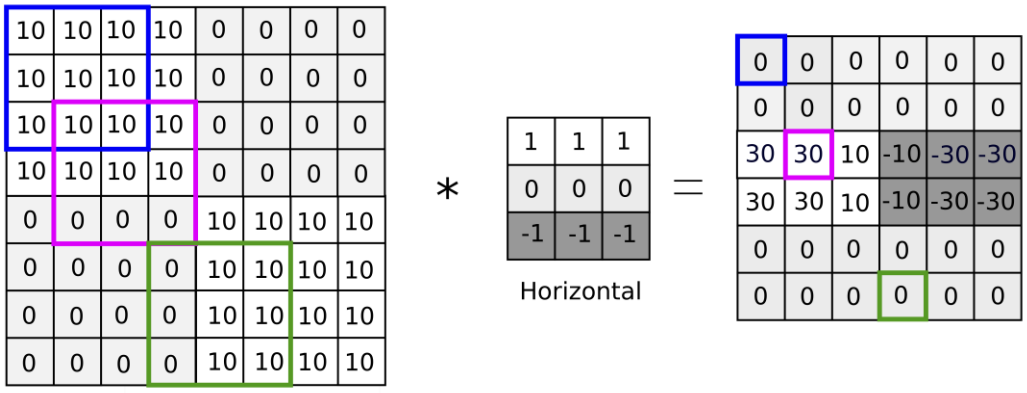
Syynä ylioppimiseen saattaa olla neuroverkon monimutkaisuus, jonka takia se kykenee poimimaan datasta kaikenlaiset sattumanvaraiset ja virheellisetkin datan ominaisuudet. Jos data esitetään pisteinä koordinaatistossa, ylisovitetun mallin kuvaaja on sellainen, että se kävisi poimimassa datan pisteitä yksitellen löytämättä oikeasti dataa kuvaavaa funktiota. Ratkaisuna ongelmaan on erilaiset regularisoinnin menetelmät, kuten L2 ja pudotusmenetelmä (dropout). L2-regularisointi kutistaa neuronien painoarvoja kohti nollaa, mikä pienentää niiden vaikutusta koko verkon toimintaan. Pudotusmenetelmä sen sijaan pudottaa harjoittelun aikana neuroneja jollain määrätyllä todennäköisyydellä nollaksi. Molemmilla malleilla neuroverkkoa yksinkertaistetaan, mikä ehkäisee ylisovittamista. [9.]

2.2 Konvoluutioverkot

Edellisessä aliluvussa puhuttiin täysin kytketyistä kerroksista, joissa jokainen kerroksen neuroni oli aina yhteydessä jokaiseen seuraavan kerroksen neuroniin. Tällaiset neuroverkot voivat toimia kuvien luokittelussa, mutta kannattaa silti harkita siirtymistä konvoluutiokerroksien käyttöön. Konvoluutiokerrokset käyttävät suodattimia (filter), joilla saadaan kuvasta tehtyä piirteiden irrotusta (feature extraction), kuten reunojen tunnistusta [10]. Suodattimista käytetään myös nimitystä kerneli.

Otetaan esimerkiksi kuva, joka on kahdeksan pikseliä korkea ja leveä. Kuva tulee syötteenä konvoluutiokerrokseen, jossa on yksi suodatin, joka on kolme kertaa kolme -matriisi. Kuvan 5 tapaan suodatinta liikutetaan kuvan päällä vasemmalta oikealla ja ylhäältä alas. Jokaisen suodattimen liikutuksen jälkeen otetaan sen syötteen alueen ja suodattimen pistetulo ylös koko kerroksesta tulevaan ulostulomatriisiin, jota kutsutaan myös piirrekartaksi (feature map). Kerroksen ulostulona ovat siis suodattimen irrottamat piirteet kuvasta. Kuva 5 esittää, miten vaakasuoria reunoja erottava suodatin onnistuu irrottamaan piirteet kuvasta. On myös hyvä ottaa huomioon, että ulostulona on niin monta matriisia kuin suodattimiakin. Tätä matriisien lukumäärää kutsutaan myös kanavien lukumääräksi.

Kuvasta 5 voidaan myös huomata, että ulostulona oleva matriisi on syötettä pienempi kooltaan. Syötteeseen voidaan halutessa lisätä nolla-arvoilla oleva reunus, jos halutaan pitää ulos tuleva matriisi samanmuotoisena. Suodattimen liikuttaminen voidaan toteuttaa myös isommilla askeleilla, jolla on päinvastainen vaikutus ulostulon kokoon. [10.]



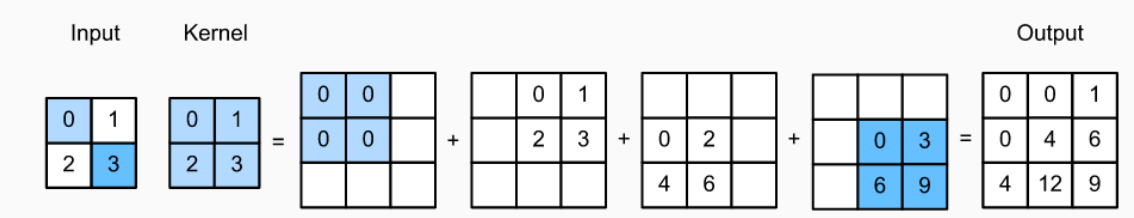
Kuva 5. Syötteen ja suodattimen pistetuloista saatu piirrekartta, jossa näkyy suodattimen kuvasta irrottamat piirteet eli vaakasuorat reunat [11].

Konvoluutiokerrosten toinen etu verrattuna täysin kytkettyihin kerroksiin on optimoitavien parametrien vähyys. Tämä johtuu parametrien jakamisesta sekä harvoista yhteyksistä. Suodattimet, jotka ovat hyödyllisiä yhdessä paikkaa, ovat luultavasti hyödyllisiä myös toisessa kuvan paikassa. Suodattimet pitävät myös

huolen, että jokainen arvo ulostulomatriisissa on riippuvainen vain pienestä määrästä syötearvoja. [12.]

Transponoidut konvoluutiokerrokset tekevät sen sijaan interpolointia ylöspäin (upsampling), jossa suodattimilla kasvatetaan syötteenä saadun piirrekartan kokoa. Transponoidun konvoluution ongelmana on kuviin jäävä ruutukuvio, joka johtuu päällekkäisyyksistä. Tätä voidaan kuitenkin välttää määrittelemällä suodattimen koko sellaiseksi, että se on jaollinen omalla askeleellaan. [13.]

Kuva 6 esittää, miten transponoitu konvoluutio suoritetaan alusta loppuun, kun syötteenä on kaksi kertaa kaksi -matriisi, jota haluamme kasvattaa kokoon kolme kertaa kolme. Suodattimena on käytössä myös kaksi kertaa kaksi -matriisi, jonka askeleen pituus on yksi. Aloitamme syötteen vasemmasta yläkulmasta, jossa olevalla arvolla kerromme jokaisen suodattimessa olevan arvon. Asetamme saadun matriisin paikalleen ulostuloksi tulevan matriisin vasempaan yläkulmaan. Toistamme tämän muille arvoille, jotka asetamme kulmiin kuvan osoittamalla tavalla. Operaatiota suorittaessa voidaan huomata, että osa arvoista menee päällekkäin. Näissä tapauksissa päällekkäin menevät arvot vain summataan keskenään. [13.]



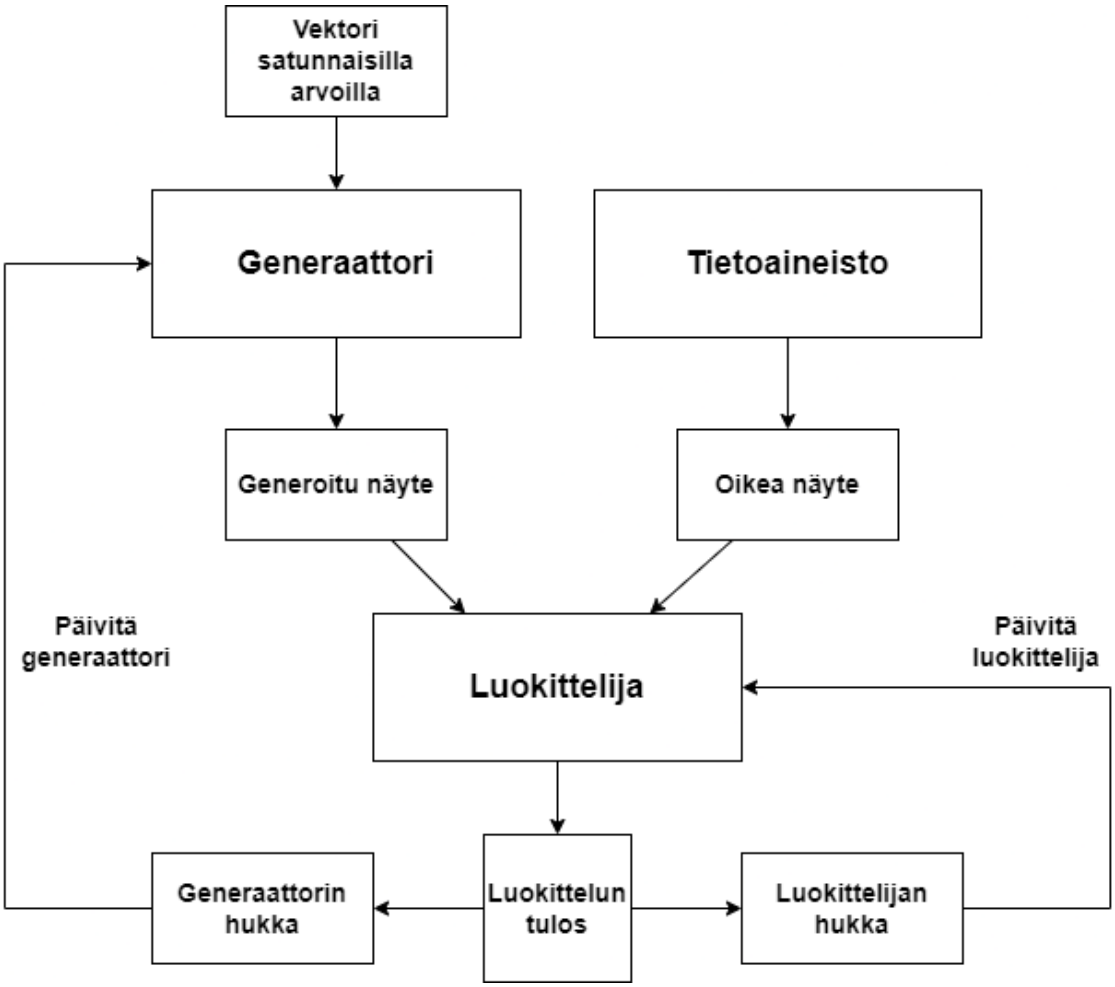
Kuva 6. Transponoitu konvoluutio -operaatio kokonaisuudessaan [13].

2.3 Generatiivinen kilpaileva verkosto

Vuonna 2014 Ian Goodfellow ja hänen kollegansa ehdottivat uutta koneoppimismenetelmää, jossa koulutetaan samanaikaisesti kahta mallia laittamalla ne kilpailemaan toisiaan vastaan. Menetelmä koostuu generaattorista ja luokittelijasta

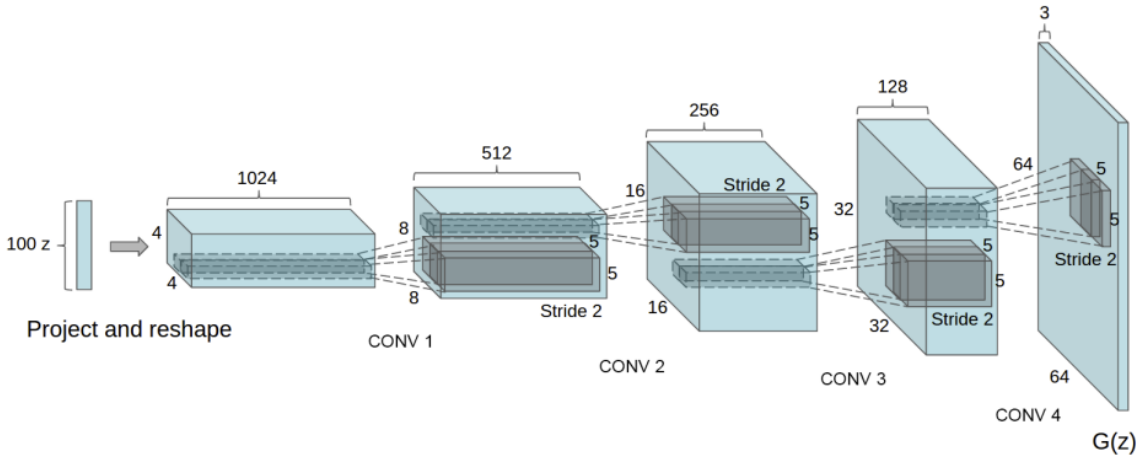
(discriminator). Generaattori luo uutta dataa, jonka se haluaa luokittelijan hyväksyvän oikeaksi dataksi. Luokittelija sen sijaan pyrkii erottamaan generaattorin tekemän datan oikeasta datasta. Kun generaattori ja luokittelija on muodostettu monikerroksisista neuroverkoista, voidaan niiden kouluttaminen suorittaa vastavirta-algoritmilla. [14.]

Generatiivisen kilpailevan verkoston toiminta on hahmoteltu kuvaan 7. Lähtökohtana verkoston toiminnalle on, että on olemassa jotain dataa, jonka perusteella halutaan generoida uutta samankaltaista dataa. Generointi tapahtuu antamalla generaattorille syötteeksi satunnaisista arvoista koostuvan vektorin, josta se luo uuden näytteen. Luokittelija saa puolet ajasta oikeita näytteitä ja puolet ajasta generaattorin tekemiä näytteitä. Todellisuudessa näytteet eivät ole yksittäisiä, vaan ne on koottu näytejoukkoihin. Luokittelija tekee arvion kuvien aitoudesta, jonka avulla lasketaan generaattorille ja luokittelijalle hukka. Generaattorin hukka on siis myös kiinni siitä, miten luokittelija arvioi generoituja kuvia. Hukan laskettuaan generaattorin ja luokittelijan painoja päivitetään uutta kierrosta varten.



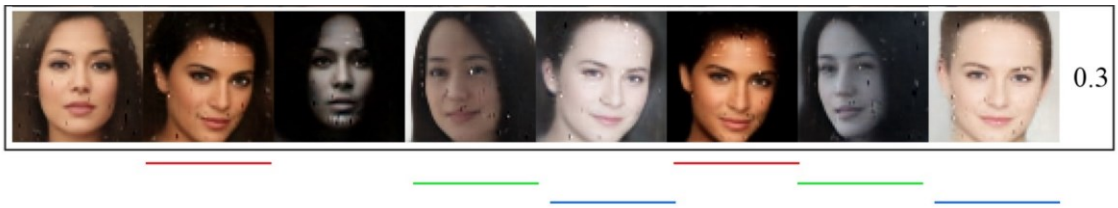
Kuva 7. Generatiivisen kilpailevan verkoston rakenne.

Toteutusosassa käytetty DCGAN-arkkitehtuuri, joka esiteltiin ensimmäisen kerran vuonna 2015, noudattaa edellä mainittuja käytäntöjä, mutta siihen on olemassa lisäohjeita. Luokittelijassa tulee käyttää konvoluutiokerroksia, ja generaattorissa transponoituja konvoluutiokerroksia. Lisäksi täysin kytketyt piilokerrokset tulee poistaa. Kuva 8 antaa mallin DCGAN-generaattorin rakentamiseen. Syötteenä kuvassa on vektori, joka muotoillaan uudelleen matriiseiksi, jotka transponoidut konvoluutiokerrokset interpoloivat ylöspäin aina kuvaan, joka on 64 pikseliä korkea ja 64 pikseliä leveä. [15.]



Kuva 8. DCGAN-generaattori [15].

Generatiiviset kilpailevat verkostot ovat tunnetusti hankalia kouluttaa johtuen generaattorin ja luokittelijan kilpailevasta rakenteesta, jossa toisen häviö on toisen voitto. Ennen verkkojen koulutusta on hyvä olla tietoinen siihen liittyvistä yleisimmistä ongelmista. Ensimmäinen näistä on se, että malli ei koskaan lähenny tasapainotilaa vaan sen parametrit heilahtelevat epävakaasti. Ei siis koskaan saavuteta tilaa, johon voitaisiin olla tyytyväisiä. Toinen ongelma on mallin sortuminen, jossa generaattori luo dataa, joka on suurelta osin vain saman toistoa. Täydellinen sortuminen ei ole yleistä, joten ongelma jää usein huomaamatta. Voi esimerkiksi olla, että generaattori luo useampaa erilaista, mutta toistuvaa dataa, kuten kuvassa 9. Kolmantena ongelmana on luokittelijan ylivertaisuus generaattorin nähden, jolloin generaattori lopettaa oppimisen. [16.]



Kuva 9. Samanvärisellä tai tasoisella viivalla merkityt kuvat alkavat muistuttamaan toisiaan, mikä on merkki mallin osittaisesta sortumisesta [16].

3 GAN-mallien kehitys

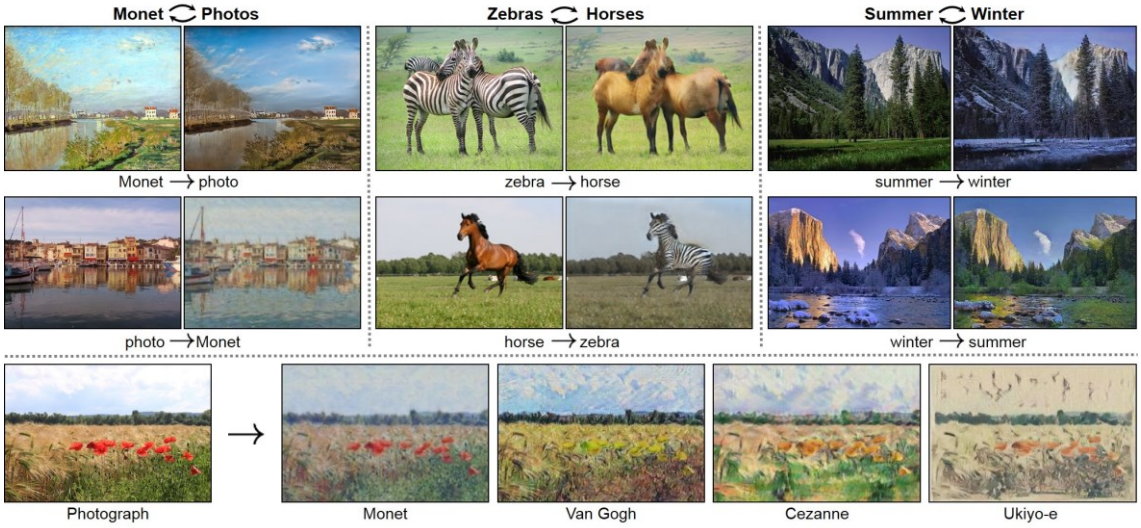
Generatiiviset kilpailevat verkostot ovat kehittyneet paljon sitten vuoden 2014. Ennen toteutukseen perehtymistä onkin hyvä tutustua erilaisiin GAN-malleihin, jotta saadaan laajempi näkemys niiden tarjonnasta.

Nostetaan ensin esille CGAN (conditional GAN), joka on ehdollinen generatiivinen kilpaileva verkosto. Se esiteltiin myös vuonna 2014. Mallin ideana on generoida dataa perustuen luokkiin, jotta voidaan periaatteessa käyttää samaa GAN-mallia esimerkiksi kissojen ja koirien generoimiseen ilman, että tulee niiden sekoituksia [17]. Ehdollinen GAN ei ole vain oma mallinsa, vaan se voidaan ottaa käyttöön erilaisissa GAN-ratkaisuissa, joissa halutaan saada enemmän kontrollia generoituun dataan.

Vuonna 2016 esiteltiin SRGAN (super resolution GAN), jonka tarkoituksena on ottaa pienemmän resoluution kuvia ja tehdä niistä isomman resoluution kuvia. SRGAN-generaattori saa sisään pienemmän resoluution kuvan, jonka resoluutiota se kasvattaa. Luokittelija yrittää sen sijaan erottaa sen kuvan, jonka resoluutiota on suurennettu, sille syötettävistä oikeista kuvista [18]. Vielä parempi tuloksia lupaava ESRGAN (enhanced SRGAN) esiteltiin vuonna 2018 [19].

CycleGAN, joka esiteltiin ensimmäisen kerran vuonna 2017, tekee kuvasta kuvaksi -kääntämistä. Kuvasta kuvaksi -kääntäminen on koneoppimisen ongelma, jonka päämääränä on kääntää kuva lähdejoukosta ulostulojoukkoon pitäen kiinni kuvan omasta sisällöstä. Sillä tarkoitetaan esimerkiksi kuvan hevosen muuttamista seepraksi tai valokuvan muuttamista valitun kuvataiteilijan tyylille kuten kuvassa 10. Tämän saavuttaminen vaatii normaalisti kohdistettuja kuvapareja. CycleGAN lupaa kuitenkin tekevänsä tämän mistä tahansa kahdesta järjestyksettömästä kuvajoukosta eli kuvien ei tarvitse esittää samaa asiaa. CycleGAN koostuu kahdesta generaattorista ja kahdesta luokittelijasta. CycleGAN toimii niin, että yksi generaattoreista ottaa kuvan joukosta X ja yrittää kääntää sen joksikin kuvaksi Y. Toinen generaattori sen sijaan ottaa kuvan joukosta Y ja

yrittää kääntää sen joksikin kuvaksi X. Molemmilla joukoilla X ja Y on sitten omat luokittelijansa, jotka yrittävät erotella aidot generaattorien tuotoksista. [20.]



Kuva 10. CycleGAN-mallin suorittamaa kuvasta kuvaksi -kääntämistä [20].

ProGAN (progressively growing GAN), joka on myös vuodelta 2017, kykenee generoimaan hyvälaatuisia ison resoluution kuvia. Samalla se yrittää vastata generatiivisten kilpailevien verkkojen epävakaisiin harjoitteluongelmiin. Sen ideana on ensin generoida pieniä neljä kertaa neljän pikselin kuvia, jotka kasvavat hiljalleen aina isoihin resoluutioihin. Generaattoriin ja luokittelijaan lisätään harjoittelun aikana lisää kerroksia tulosten parantuessa. Tämän ideana on, että pienillä kuvilla oppiminen on huomattavasti nopeampaa. Kuvien kokoa lisätään myös hiljalleen, että pienemmän resoluution kuvilla opitut asiat eivät mene hukkaan. [21.]

StyleGAN on Nvidian vuonna 2018 julkaisema malli, jossa käytettiin samanlaista progressiivista kasvumallia kuin ProGAN-mallissa, jonka lisäksi siihen oli yhdistetty tyylin siirtoa [22]. StyleGAN generoi tunnetusti laadukkaita ja isoja fotorealistisia kasvokuvia. Vuonna 2019 avautui StyleGAN-mallilla tehty kuuluisa ”This Person Does Not Exist” -sivusto, jossa jokaisella sivun latauksella sai

eteensä uuden fotorealistisen kasvokuvan. StyleGAN on myös kehittynyt vuosien saatossa. Vuonna 2020 julkaistiin StyleGAN-2, jota seurasi vuonna 2021 julkaistu StyleGAN-3.

4 DCGAN-toteutus

Tämä DCGAN-toteutus on tehty Python-ohjelmointikielellä. Kielen valinta oli luonnollinen, koska Pythonille löytyy koneoppimisen alalta laaja käyttäjäkunta ja kirjastot. Tässä työssä käytetty koneoppimisen kirjasto on TensorFlow, jonka on kehittänyt Google Brain Team. TensorFlow'n nimi tulee operaatioista, joita neuroverkot suorittavat moniulotteisille taulukoille. Tensori on eräänlainen vektorien ja matriisien yleistys. Ensimmäinen versio julkaistiin vuonna 2015. Päivitetty versio 2.0 julkaistiin syyskuussa 2019. Keras, jota käytetään neuroverkkojen rakentamisessa, on sen sijaan TensorFlow'n päällä pyörivä ohjelmointirajapinta. TensorFlow otetaan käyttöön esimerkkikoodissa 1. Muiden kirjastojen liittäminen toteutukseen ei ole pakollista. Tarvitsemme OS-moduulia vain myöhemmin tallennusoperaatiossa.

```
import os
import tensorflow as tf
print(tf.__version__)
```

Esimerkkikoodi 1. TensorFlow-versio 2.10.0 otetaan käyttöön ohjelmassa.

4.1 Data

Luokittelija tarvitsee oikeita kasvokuvia, joihin koko verkostolta haluttu toiminta perustuu. Tässä toteutuksessa käytetyt oikeat kuvat ovat CelebA-tietoaineistosta, joka sisältää 202 599 kasvokuvaa eri julkisuuden henkilöistä [23].

Esimerkkikoodi 2 esittää yhden tavan kuvien käyttöönottamisesta, kun kuvat ovat paikallisessa hakemistossa. Toimenpiteen seurauksena muuttuja dataset saa arvokseen tietoaineisto-olion, joka sisältää luokittelemattomia kolmen värikanavan kuvia (RGB). Samalla kuvien kokoa pienennetään alkuperäisestä, jotta kouluttamiseen vaadittavat resurssit eivät nouse liian suuriksi. Näytejoukko

(batch) asetetaan myös sopivaan kokoon. Pienten näytejoukkojen on todettu nopeuttavan oppimista joissakin tapauksissa riippuen mallista ja tietoaineistosta [24]. Lopuksi kuvat normalisoidaan niin, että kaikkien pikselien arvot ovat nollan ja yhden välillä. Suuret pikseliarvot voivat aiheuttaa ongelmia ja hidastumista koulutusvaiheessa [25].

```
dataset = tf.keras.utils.image_dataset_from_directory(
 directory = 'celeb_a',
 label_mode = None,
 color_mode = 'rgb',
 batch_size = 16,
 image_size = (128, 128),
 shuffle = True
).map(lambda x: x / 255.0)
```

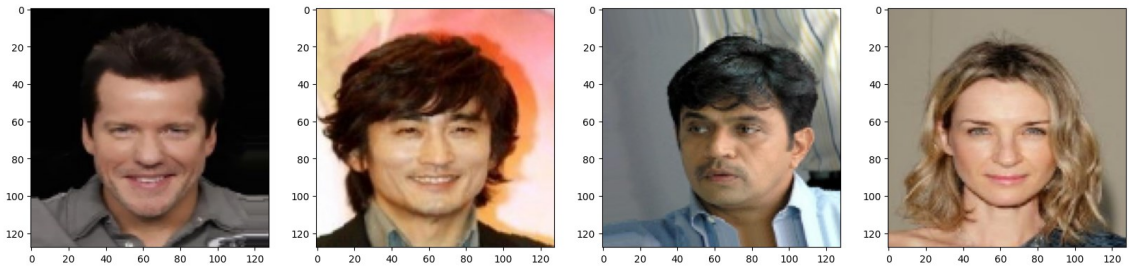
Esimerkkikoodi 2. Paikallisessa hakemistossa olevat kuvat valmistellaan käyttöön otettaviksi.

Dataa on hyvä visualisoida ennen koulutusta, että varmistutaan kuvien olevan oikeassa muodossa. Kuvien muodon ja pikselien arvot saadaan suorittamalla esimerkkikoodi 3. Koodi tulostaa ensin kuvien muodon, joka koostuu näytejoukon koosta, korkeudesta, leveydestä ja kanavista. Tämän jälkeen tulostetaan taulukkoina pikseliarvot, joiden kuuluu olla nollan ja yhden välillä.

```
print(dataset.as_numpy_iterator().next().shape)
print(dataset.as_numpy_iterator().next())
```

Esimerkkikoodi 3. Tarkistetaan, että kuvat ovat oikeassa muodossa.

Ottamalla otoksia tietoaineistosta sen jälkeen, kun esimerkkikoodi 2 on suoritettu, saadaan kuvassa 11 olevia värillisiä kasvokuvia. Kuvat näyttävät olevan valmiina luokittelijalle syötettäväksi.



Kuva 11. Ote tietoaineistossa olevista kuvista sen jälkeen, kun esimerkkikoodi 2 on suoritettu.

4.2 Generaattori

Generaattorin rakentamisessa on mukailtu DCGAN-generaattoria, joka oli aiemmin kuvassa 8. Mallin rakentamiseen on käytetty TensorFlow'n sekvenssimallia (Sequential model). Sekvenssimallissa on pino kerroksia, jossa jokaisella kerroksella on täsmälleen yksi tensori syötteenä ja yksi tensori ulostulona.

Esimerkkikoodi 4 luo generaattorin, joka ottaa syötteenä vektorin, joka muokataan kahdeksan kertaa kahdeksan -matriisiksi, jolla on 128 kanavaa. Neljä transponoitua konvoluutiokerrosta sitten interpoloivat ylöspäin niille tulevaa syötettä. Kerroksissa olevien suodattimien koko on jaollinen niiden askeleella, jonka pitäisi auttaa ruutukuvion välttämisessä. Viimeisenä oleva konvoluutiokerros palauttaa lopulta 128 pikseliä korkean ja 128 pikseliä leveän kolmen värikanavan kuvan, jonka pikseliarvot ovat sigmoidin aktivointifunktion takia nollan ja yhden välillä. Muissa paitsi viimeisessä kerroksessa käytetään Leaky ReLU -aktivointifunktiota, jolle määritellään suluissa nollaa pienemmillä arvoilla olevan vuodon suuruus. Näin myös nollaa pienemmillä arvoilla tapahtuu oppimista.

```
generator = tf.keras.Sequential([
 tf.keras.layers.Dense(8*8*128, input_shape = (128,)),
 tf.keras.layers.Reshape((8, 8, 128)),
 tf.keras.layers.Conv2DTranspose(
  filters = 1024,
  kernel_size = 4,
  strides = 2,
  padding = 'same'
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Conv2DTranspose(
  filters = 512,
  kernel_size = 4,
  strides = 2,
  padding = 'same'
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Conv2DTranspose(
  filters = 256,
  kernel_size = 4,
  strides = 2,
  padding = 'same'
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Conv2DTranspose(
  filters = 128,
  kernel_size = 4,
  strides = 2,
  padding = 'same'
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Conv2D(
  3,
  kernel_size = 5,
  padding = 'same',
  activation = 'sigmoid'
 )
])
```

Esimerkkikoodi 4. Generaattorin verkko koostuu neljästä transponoidusta konvoluutiokerroksesta, jotka interpoloivat ylöspäin eli kasvattavat pikselien määrää.

Generaattorin koodista on suoraan vaikea hahmottaa neuroverkon rakennetta. Virheiden välttämiseksi voidaan neuroverkosta tulostaa tiivistelmä esimerkkikoodin 5 mukaisesti. Tällä saadaan tietoon muun muassa kerrosten ulostulojen muoto ja opittavien parametrin määrät.

```
generator.summary()
```

Esimerkkikoodi 5. Tulostetaan tiivistelmä neuroverkon rakenteesta.

4.3 Luokittelija

Luokittelija, joka määritellään esimerkkikoodissa 6, käyttää generaattorin tapaan sekvenssimallia. Näytejoukon kuvat, jotka luokittelija saa syötteenä, ovat 128 pikseliä korkeita ja 128 pikseliä leveitä. Lisäksi kuvilla on kolme värikanavaa. Tästä lähtötilanteesta luokittelijan pitäisi saada lopputulokseksi arvo nollan ja yhden väliltä, joka on luokittelijan arvio kuvan aitoudesta.

DCGAN-arkkitehtuurin mukaisesti luokittelija koostuu konvoluutiokerroksista. Koska luokittelijalta halutaan lopulta ulos vain yksi arvo, litistetään moniulotteinen tensori yhteen ulottuvuuteen ennen viimeistä yhden neuronin ulostulokerrosta. Luokittelijalle on myös määritelty pudotuskerros. Sen tarkoituksena on harjoittelun aikana tiputtaa satunnaisesti syötteiden arvoja nollaksi. Annetulla parametrin arvolla tämä todennäköisyys on 20 prosenttia. Pudotuskerroksen ideana on estää luokittelijan ylisovittaminen dataan. Toiveena on, että tämä auttaisi generaattoria onnistumaan paremmin, koska luokitteluongelma on yksinkertaisempi.

```
discriminator = tf.keras.Sequential([
 tf.keras.layers.Conv2D(
  filters = 64,
  kernel_size = 4,
  strides = 2,
  padding = 'same',
  input_shape = (128, 128, 3)
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Conv2D(
  filters = 128, kernel_size = 4, strides = 2, padding = 'same'
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Conv2D(
  filters = 256, kernel_size = 4, strides = 2, padding = 'same'
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Conv2D(
  filters = 512, kernel_size = 4, strides = 2, padding = 'same'
 ),
 tf.keras.layers.LeakyReLU(0.2),
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(1, activation = 'sigmoid')
])
```

Esimerkkikoodi 6. Luokittelijan verkko koostuu neljästä konvoluutiokerroksesta, joiden jälkeen moniulotteinen tensori litistetään yhteen ulottuvuuteen.

4.4 Verkoston kouluttaminen

Generaattori ja luokittelija opetetaan samassa koulutussilmukassa, joka on määritelty esimerkkikoodissa 7. Tämä suoritetaan niin monta kertaa kuin koko tietoaineisto halutaan käydä läpi. Kerran koko tietoaineiston läpi käymistä kutsutaan yhdeksi epookiksi. Molemmat verkot oppivat jokaisella opetusaskeleella (train step), joka tarkoittaa yhden näytejoukon läpikäymistä.

```
def train(dataset, epochs):
 for epoch in range(epochs):
  for image_batch in dataset:
   train_step(image_batch)
```

Esimerkkikoodi 7. Koulutussilmukasta voi nähdä, että yhden epookin aikana käydään läpi koko tietoaineisto, ja jokaisella opetusaskeleella käydään läpi yksi näytejoukko [26].

Ennen kuin katsotaan opetusaskeleen sisälle, on hyvä tietää, mitä optimointialgoritmia ja hukkafunktiota siellä käytetään. Esimerkkikoodin 8 mukaan määritellään generaattorille ja luokittelijalle optimointialgoritmi erikseen. Molemmat käyttävät Adam-algoritmia, joka on tarkoitettu stokastisen kaltevuuslaskun optimointiin. Adam-algoritmin parametrina on sille annettu oppimisnopeus. Hukkafunktioksi määritellään binäärinen ristientropia, joka on tarkoitettu luokitteluongelmille, jossa on kaksi kategoriaa. Binäärinen ristientropia ennustaa jommankumman kahdesta kategoriasta luokitteluongelmassa [27].

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
loss_function = tf.keras.losses.BinaryCrossentropy()
```

Esimerkkikoodi 8. Generaattorille ja luokittelijalle määritellään omat optimointialgoritmit. Molemmat käyttävät häviöfunktionaan binääristä ristientropiaa [26].

Opetusaskeleessa esimerkkikoodissa 9 luodaan ensin satunnaisia vektoreita, joita käytetään generaattorissa uusien kuvien luontiin. Sen jälkeen näytejoukko oikeita, ja näytejoukko generaattorin kuvia syötetään luokittelijalle, jonka palauttamat arvot otetaan talteen. Näiden perusteella lasketaan binäärinen ristientropia ja lopulta gradientit. Luokittelijalle on summattu yhteen hukka oikeista ja generoiduista kuvista. Viimeisessä vaiheessa Adam-algoritmi hoitaa arvojen päivityksen molemmille verkoille.

```
@tf.function
def train_step(image_batch):
 latent_vectors = tf.random.normal([BATCH_SIZE, latent_dimension])
 with tf.GradientTape() as g_tape, tf.GradientTape() as d_tape:
  generated_images = generator(latent_vectors, training = True)
  fake_output = discriminator(generated_images, training = True)
  real_output = discriminator(image_batch, training = True)
  g_loss = loss_function(tf.ones_like(fake_output), fake_output)
  d_loss = loss_function(tf.ones_like(real_output), real_output)
  d_loss += loss_function(tf.zeros_like(fake_output), fake_output)
 g_grads = g_tape.gradient(g_loss, generator.trainable_variables)
 d_grads = d_tape.gradient(d_loss, discriminator.trainable_variables)
 generator_optimizer.apply_gradients(
  zip(g_grads, generator.trainable_variables)
 )
 discriminator_optimizer.apply_gradients(
  zip(d_grads, discriminator.trainable_variables)
 )
```

Esimerkkikoodi 9. Verkkojen oppiminen tapahtuu opetusaskeleen sisällä [26].

Verkoston kouluttamisessa voi mennä hyvin pitkiä aikoja, joten niiden suorittaminen yhdessä sessiossa ei välttämättä onnistu. Esimerkkikoodi 10 näyttää, miten määritellään tallennuspiste niin, että kouluttamista voidaan jatkaa myöhemmin. Koodissa määritellään tallennuspiste, joka ottaa talteen generaattorin, luokittelijan ja niiden optimointialgoritmit.

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(
 generator_optimizer = generator_optimizer,
 discriminator_optimizer = discriminator_optimizer,
 generator = generator,
 discriminator = discriminator
)
```

Esimerkkikoodi 10. Määritellään tallennuspiste [26].

Mitään ei ole vielä kuitenkaan tallennettu. Esimerkkikoodissa 11 suoritetaan seuraavaksi pisteen tallennus. Tekemällä tallennuksen koulutussilmukassa, voimme ottaa pisteen talteen esimerkiksi kerran epookissa. Näin varmistumme, että emme menetä kaikkea opetusdataa järjestelmän kaatuessa.

```
checkpoint.save(file_prefix = checkpoint_prefix)
```

Esimerkkikoodi 11. Tallennuspiste on nyt aiemmin määritellyssä hakemistossa [26].

Tallennuspisteestä voidaan aloittaa, kun se palautetaan ennen koulutuksen jatkamista esimerkkikoodin 12 mukaisesti. Jos samaa tallennusta käytetään edelleen, uusi tallennus ei tule ylikirjoittamaan vanhaa pistettä, joten kaikki pisteet jäävät talteen.

```
checkpoint.restore(
 tf.train.latest_checkpoint('./training_checkpoints')
)
```

Esimerkkikoodi 12. Ladataan viimeisin tallennuspiste koulutuksen jatkamista varten [26].

4.5 Tulokset ja mallin arviointi

Generatiivisen kilpailevan verkoston monitorointi ei ole yhtä suoraviivaista kuin esimerkiksi pelkän kuvia luokittelevan verkon tarkkuuden mittaaminen johtuen verkkojen kilpailusta keskenään [28]. Generaattorin ja luokittelijan hukkaa seuraamalla voidaan päätellä joitakin ongelmia kuten tilanteita, joissa toinen osapuoli dominoi toista. Suoraviivaisin tapa monitoroida mallin hyvyyttä on kuitenkin tarkastella kuvia, joita generaattori luo koulutuksen eri vaiheissa. Esimerkkikoodi 13 näyttää, miten generaattori saadaan tuottamaan uusia kuvia, jotka voidaan sitten tallentaa monitorointia varten.

```
predictions = generator(tf.random.normal([16, 128]), training = False)
for i in range(16):
 img = tf.keras.utils.array_to_image(predictions[i])
 img.save(f'./generated_images/img_epoch_{epoch+1}_{i+1}.png')
```

Esimerkkikoodi 13. Generaattorilta pyydetään uusia kuvia. Upottamalla koodin koulutussilmukkaan saadaan kuvia jokaiselta epookilta monitorointia varten.

Koulutusta tehtiin 100 epookin verran, johon asti oli havaittavissa generoitujen kuvien laadun parantumista. Saadut tulokset ovat lupaavia, kuten kuvasta 12 voidaan havaita. Jos parhaat kuvat poimitaan joukosta, aletaan lähestyä tietoaineistosta löytyviä oikeita kuvia. Jostain syystä generoiduissa kuvissa näyttää olevan lähinnä naisten kasvokuvia. On vaikea sanoa, onko toteutuksessa vikaa vai itse tietoaineistossa. Käyttämällä CGAN-toteutusta, jossa jaetaan sukupuolet omiksi luokikseen, saataisiin mahdollisesti enemmän vaihtelua kuviin.

Kuva 12. Generaattorilta saatuja kuvia 100 epookin jälkeen.

Verkkojen rakenne voisi varmasti olla parempikin, mutta ajallisesti eri ratkaisujen kokeileminen on haastavaa. Ennen lopullista mallia generaattorissa olevissa transponoiduissa konvoluutiokerroksissa suodattimien lukumäärät olivat toisinpäin eli ensimmäisessä kerroksessa niiden määrä oli pienin. Yhden epookin pituinen harjoitteluaika oli tuolloin yli kolme tuntia. Ottamalla enemmän mallia Alec Bradfordin ja hänen kollegoidensa alkuperäisestä DCGAN-esityksestä päädyttiin nykyiseen ratkaisuun [15]. Suodattimien lukumäärän ollessa korkein

ensimmäisessä kerroksessa yhden epookin pituisessa harjoittelussa meni samalla laitteella enää vajaat 40 minuuttia. Aikaisemmassa vaiheessa olleita mallin romahduksiakaan ei ollut enää havaittavissa.

5 Yhteenveto

Työn ensimmäiseksi tavoitteeksi oli asetettu generatiivisen kilpailevan verkoston toteutus, joka kykenee generoimaan fotorealistisia kasvokuvia. Haasteena oli resurssien vähyys ja olematon kokemus neuroverkkojen parissa työskentelystä. Toteutetussa ratkaisussa päädyttiin siksi melko yksinkertaiseen ja kuvien generoinnissa jo nopean kehityksen takia vanhentuneeseen arkkitehtuurin nimeltä DCGAN.

Generoidut kuvat vastasivat odotuksia, jotka perustuivat muiden saavuttamiin tuloksiin vastaavilla verkoilla. Poimimalla sopivat kuvat joukosta saadaan luotua joukko kuvia, joita voisi luulla oikeiksi. Kuvat olivat lopulta kuitenkin hyvin pieniä kooltaan ja niille sopivat käyttötarkoitukset rajallisia. DCGAN-arkkitehtuurin käyttäminen isompien kuvien generoimiseen ei vain ollut realistista käytössä olleilla resursseilla.

Toisena tavoitteena oli perehtyä generatiivisen kilpailevan verkoston toimintaperiaatteisiin, joka tarkoitti tietenkin samalla perehtymistä neuroverkkoihin sekä konvoluutioverkkoihin johtuen valitusta toteutuksesta. Koin saavani hyvän pohjan mahdollisia tulevia syväoppimisprojekteja varten, olivat ne sitten GAN-projekteja tai eivät. Asioiden syvempi ymmärtäminen jäi tosin välillä heikommalle pohjalle, koska ne vaativat taustalla olevan matematiikan hallitsemista. Työssä ei tästä syystä lähestytty asioita matematiikan kautta.

Tämän työn jälkeen en näe enää syytä palata DCGAN-toteutukseen ainakaan kasvokuvien generointia varten. Generatiiviset mallit ovat kehittyneet nopeaa tahtia viimeisten vuosien aikana, joten seuraavaksi kannattaa tutustua johonkin uudempaan ratkaisuun. Jos haluaa jatkaa generatiivisten kilpailevien verkoston

käyttämistä, kannattaa tutustua ProGAN- tai StyleGAN-arkkitehtuureihin. Internetistä löytyy lukuisia sivuja, jotka antavat ulos StyleGAN-toteutuksella generoituja hyvälaatuisia kuvia eri asioista. Tässä työssä toteutettu DCGAN ei pysty näiden kanssa kilpailemaan.

Diffuusioon perustuvat generatiivisen koneoppimisen mallit ovat olleet myös suuressa suosiossa nykypäivänä, ja ne suoriutuvatkin joistain tehtävistä paremmin kuin generatiiviset kilpailevat verkostot. Ajatellaan, että meillä on esimerkiksi kasvokuva, johon lisätään asteittain kohinaa niin, että lopulta kuva on pelkkää kohinaa. Diffuusiomallin perusideana on, että voidaan aloittaa kohinaa sisältävästä kuvasta, josta kohinaa poistetaan hiljalleen, minkä seurauksena saataisiin lopulta taas kasvokuva. Diffuusiota käyttävät muun muassa DALL-E 2, Imagen ja Stable Diffusion. [29.]

Lähteet

1 Roose, Kevin. 2022. An A.I.-Generated Picture Won an Art Prize. Artists Aren't Happy. The New York Times 2.9.2022. Verkkoaineisto. <https://www.nytimes.com/2022/09/02/technology/ai-artificial-intelligence-artists.html>. Luettu 2.10.2022.

2 Falcon, William. 2018. What Happens Now That An AI-Generated Painting Sold For $432,500? Forbes 25.10.2018. Verkkoaineisto. <https://www.forbes.com/sites/williamfalcon/2018/10/25/what-happens-now-that-an-ai-generated-painting-sold-for-432500/?sh=299e4e50a41c>. Luettu 27.10.2022.

3 Gabel, Frank. 2019. Artificial Intelligence for Games: Seminar. Verkkoaineisto. <https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/636026949/report_frank_gabel.pdf>. Luettu 26.10.2022.

4 Machine Learning. IBM. Verkkoaineisto. <https://www.ibm.com/cloud/learn/machine-learning>. Luettu 3.10.2022.

5 The types of machine learning. Helsingin yliopisto. Verkkoaineisto. <https://course.elementsofai.com/4/1>. Luettu 27.10.2022.

6 Nielsen, Michael. 2019. Verkkoaineisto. <http://neuralnetworksanddeeplearning.com/chap1.html>. Luettu 28.10.2022.

7 Jain, Vandit. 2019. Everything you need to know about "Activation Functions" in Deep learning models. Verkkoaineisto. <https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253>. Luettu 1.11.2022.

8 Thakur, Ayush. 2022. ReLU vs. Sigmoid Function in Deep Neural Networks Verkkoaineisto. <https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--VmlldzoyMDk0MzI>. Luettu 30.10.2022.

9 Oppermann, Artem. 2020. Regularization in Deep Learning — L1, L2, and Dropout. Verkkoaineisto. <https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036>. Luettu 1.11.2022.

10 Vo, Anh. 2018. Deep Learning – Computer Vision and Convolutional Neural Networks. Verkkoaineisto. <https://anhvnn.wordpress.com/2018/02/01/deep-learning-computer-vision-and-convolutional-neural-networks>. Luettu 31.10.2022.

11 Reynolds, Anh. 2019. Convolutional Neural Networks (CNNs). Verkkoaineisto. <https://anhreynolds.com/blogs/cnn.html>. Luettu 2.11.2022.

12 Ng, Andrew. 2017. C4W1L11 Why Convolutions. Verkkoaineisto. <https://www.youtube.com/watch?v=ay3zYUeu-yhU&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF>. Katsottu 15.10.2022.

13 Mishra, Divyanshu. 2020. Transposed Convolution Demystified. Verkkoaineisto. <https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>. Luettu 30.10.2022.

14 Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron & Bengio, Yoshua. 2014. Generative Adversarial Nets. Verkkoaineisto. <https://arxiv.org/pdf/1406.2661.pdf>. Luettu 2.10.2022.

15 Radford, Alec; Metz, Luke & Chintala, Soumith. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. Verkkoaineisto. <https://arxiv.org/pdf/1511.06434.pdf>. Luettu 27.10.2022.

16 Hui, Jonathan. 2018. GAN — Why it is so hard to train Generative Adversarial Networks! Verkkoaineisto. <https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b>. Luettu 31.10.2022.

17 Mirza, Mehdi & Osindero, Simon. 2014. Conditional Generative Adversarial Nets. Verkkoaineisto. <https://arxiv.org/pdf/1411.1784.pdf>. Luettu 2.11.2022.

18 Ledig, Christian; Theis, Lucas; Huszar, Ferenc; Caballero, Jose; Cunningham, Andrew; Acosta, Alejandro; Aitken, Andrew; Tejani, Alykhan; Totz, Johannes; Wang, Zehan & Shi, Wenzhe. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. Verkkoaineisto. <https://arxiv.org/pdf/1609.04802v5.pdf>. Luettu 6.11.2022.

19 Wang, Xintao; Yu, Ke; Wu, Shixiang; Gu, Jinjin; Liu, Yihao; Dong, Chao; Loy, Chen; Qiao, Yu & Tang, Xiaoou. 2018. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. Verkkoaineisto. <https://arxiv.org/pdf/1809.00219.pdf>. Luettu 6.11.2022.

20 Zhu, Jun-Yan; Park, Taesung; Isola, Phillip & Efros, Alexei. 2018. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. Verkkoaineisto. <https://arxiv.org/pdf/1703.10593v6.pdf>. Luettu 2.11.2022.

21 Karras, Tero; Aila, Timo; Laine, Samuli & Lehtinen, Jaakko. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. Verkkoaineisto. <https://arxiv.org/pdf/1710.10196.pdf>. Luettu 30.10.2022.

22 Karras, Tero; Laine, Samuli & Aila, Timo. 2019. A Style-Based Generator Architecture for Generative Adversarial Networks. Verkkoaineisto. <https://arxiv.org/pdf/1812.04948.pdf>. Luettu 30.10.2022.

23 Liu, Ziwei; Luo, Ping; Wang, Xiaogang & Tang, Xiaoou. 2015. Large-scale CelebFaces Attributes (CelebA) Dataset. Verkkoaineisto. <https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html>. Luettu 23.10.2022.

24 Shen, Kevin. 2018. Effect of batch size on training dynamics. Verkkoaineisto. <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e>. Luettu 24.10.2022.

25 Brownlee, Jason. 2019. How to Manually Scale Image Pixel Data for Deep Learning. Verkkoaineisto. <https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning>. Luettu 26.10.2022.

26 TensorFlow. Deep Convolutional Generative Adversarial Network. Verkkoaineisto. <https://www.tensorflow.org/tutorials/generative/dcgan>. Luettu 22.11.2022.

27 Brownlee, Jason. 2019. A Gentle Introduction to Cross-Entropy for Machine Learning. Verkkoaineisto. <https://machinelearningmastery.com/cross-entropy-for-machine-learning>. Luettu 2.11.2022.

28 Brownlee, Jason. 2019. How to Evaluate Generative Adversarial Networks. Verkkoaineisto. <https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks>. Luettu 30.10.2022.

29 Seff, Ari. 2022. What are Diffusion Models? Verkkoaineisto. <https://www.youtube.com/watch?v=fbLgFrlTnGU>. Katsottu 30.10.2022.