jamk.fi

Ilmakuvien analysointi digitaalisesti Tensorflow ohjelmistolla

Niko Taari

Opinnäytetyö
Toukokuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), tietotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Taari, Niko	Opinnäytetyö, AMK	Toukokuu 2017
	Sivumäärä 51 + 6	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Ilmakuvien analysointi digitaalisesti TensorFlow ohjelmistolla

Tutkinto-ohjelma
Tietotekniikka

Työn ohjaaja(t)
Kokkonen Tero, Rantonen Mika

Toimeksiantaja(t)
Maaseutu 2.0 projekti, Weckström Petteri

Tiivistelmä

Opinnäytetyön toimeksiantaja oli Maaseutu 2.0-hanke ja tavoitteena oli selvittää onko mahdollista hyödyntää tekoälyä hankkeessa yhden käyttötapauksen avulla. Käyttötapauksena on myrskyvaurioiden tunnistaminen ilmakuvista. Tätä varten opinnäytetyössä luotiin järjestelmä, joka havaitsisi kaatuneita puita lennokeilla otetuista ilmakuvista. Ilmakuvista tehtävän muutosanalyysin luokittelu tehtiin TensorFlow ohjelmalla.

Opinnäytetyön teoriaosuus keskittyy neuroverkkojen teoriaan, mutta koneoppimiseen ja TensorFlow ohjelmistoon liittyvää teoriaa käsitellään myös.

Opinnäytetyön toteutusvaiheessa pystytettiin järjestelmä omalle palvelimelle yhdelle virtuaalikoneelle. Toteutusvaihe aloitettiin alkuvalmisteluilla, josta edettiin varsinaisen järjestelmän pystytykseen. Järjestelmän rakentamisen jälkeen siirryttiin testaamaan järjestelmää.

Opinnäytetyössä todettiin että tekoälyä voidaan hyödyntää Maaseutu 2.0-hankkeen tarpeisiin. Käyttötapausta varten luodussa järjestelmässä jäi jatkokehityksen varaa ja näitä jatkokehitys tarpeita on opinnäytetyössä kartoitettu.

Avainsanat (asiasanat)
TensorFlow, Neuroverkko, Koneoppiminen

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Taari, Niko	Bachelor's thesis	May 2017
		Language of publication: Finnish
	Number of pages 51 + 6	Permission for web publication: x

Title of publication

Digital analyzing of aerial images using TensorFlow

Degree programme

Information Technology

Supervisor(s)

Kekkonen Tero, Rantonen Mika

Assigned by

Maaseutu 2.0 project, Weckström Petteri

Abstract

The bachelor's thesis was assigned by Maaseutu 2.0 project and the purpose of the thesis was to find out if artificial intelligence could be utilized in the project and this was done with one use case. The use case was to recognize storm damage from aerial images. This was done by building a system capable of recognizing fallen trees from aerial images. In this bachelor's thesis, a machine learning program TensorFlow was used to label the changes in the aerial images.

Theory part of the bachelor's thesis focuses on neural networks; however, the theory of machine learning and TensorFlow is discussed too.

In the thesis, the system was built in a virtual machine on a private server. Building the system started with installing all the prerequisites, after which it was built. After the building phase, the system underwent a testing phase.

The results showed that artificial intelligence could be utilized by the Maaseutu 2.0 project in some way. The system built for the use case had room for improvement. These improvements are also discussed in the thesis.

Keywords/tags (subjects)

TensorFlow, Neural Network, Machine learning

Miscellaneous

Sisältö

Kuviot

Taulukot

Lyhenteet

CNN	Convolutional Neural Network
FNN	Feedforward Neural Network
GDAL	Geospatial Data Abstraction Library
GeoTIFF	Georeferenced Tagged Image File Format
HRV	High Resolution Visible
JPEG	Joint Photographic Experts Group
LSTM	Long Short Term Memory
PNG	Portable Network Graphics
RDMA	Remote Direct Memory Access
RNN	Recurrent Neural Network
TPU	Tensor Processing Unit

1 Johdanto

1.1 Toimeksiantaja

Opinnäytetyö on osa Maaseutu 2.0-hanketta, jonka tarkoituksena on löytää uusia liiketoimintamahdollisuuksia maatalouteen tietoteknisiä ratkaisuja hyödyntäen. Nämä ratkaisut voivat esimerkiksi säästää maanomistajien aikaa. Yksi mahdollisista tarjottavista ratkaisuista voisi olla metsänomistajille mahdollisuus tarkastella tiluksiaan tietokoneelta käsin. Tämän ratkaisun voisi toteuttaa joko satelliittikuvien tai ilmakuvien avulla. Ratkaisun johdosta metsänomistajilla säästyy aikaa ja mahdollisesti rahaa, mikäli he muuten tarkistaisivat tiluksiaan autolla. (Maaseutu 2.0 siirtää metsänomistajan digiaikaan. 2016.)

Maaseutu 2.0-hankkeen keskesimpänä tuloksena olisi luoda validoitava integraatioväylän prototyyppi. Kyseinen prototyyppi muuntaa luonnonvara-alan keskeisiä digitaalisia paikkatietoaineistoja yhteismitallisen standardin mukaisiksi työkohteiksi. Nämä työkohteet olisi sitten saatavilla eri toimijoille tietyssä muodossa avoimen rajapinnan kautta. Hanke alkoi vuoden 2016 helmikuussa ja se kestää vuoden 2018 tammikuun loppuun asti. Hankkeen toteuttajia ovat Jyväskylän ammattikorkeakoulun lisäksi Bitcomp Oy, Suomen metsäkeskus ja MHY Keski-Suomi ry. Hankkeen rahoittavat Keski-Suomen ELY-keskus (Manner-Suomen maaseudun kehittämisohjelma 2014 – 2020), Metsä Group ja Nordkalk Oy.(Maaseutu 2.0. N.d.)

1.2 Opinnäytetyön tavoitteet ja kuvaus

Opionnäytetyön tutkimusmenetelmänä käytetään konstruktiivista tutkimusta. Opinnäytetyön tavoitteena oli kartoittaa voiko tekoälyä hyödyntää Maaseutu 2.0-hankkeessa. Tähän pyrittiin löytämään vastauksia luomalla järjestelmä, jolla pystyy tekemään muutosanalyysiä värillisille ilmakuville ja sen on pystyttävä havaitsemaan kaatuneet puut. Järjestelmä voisi mahdollisesti tehdä kaatuneista puista jonkin ilmoituksen. Järjestelmää ajavan tietokoneen käyttöjärjestelmänä on Ubuntu. Järjestelmässä tullaan testaamaan TensorFlow nimistä ohjelmistoa, jolla muutoksen luokittelu toteutettiin. Järjestelmässä oltaisiin voitu käyttää muitakin avoimen

lähdekoodin koneoppimisohjelmistoja, mutta TensorFlow ohjelmisto valittiin toimeksiantajan ehdotuksesta.

Järjestelmään syötettävät värilliset ilmakuvat tulevat järjestelmälle GeoTIFF tiedostomuotoisina ja järjestelmää varten niiden tiedostomuoto muutetaan TensorFlow'n tukemaan tiedostomuotoon erillisen skriptin avulla. Kaksi uusinta ilmakuvaa laitetaan syötteiksi TensorFlow'n python skriptiin. Skriptin ensimmäinen osa vertaa niitä keskenään. Kun havaitaan eroavaisuus uusimmasta kuvasta, irrotetaan muuttunut alue ja syötetään se skriptin toiseen osaan, missä tehdään objektin tunnistus. Järjestelmää testataan omilla laitteilla ja testausvaiheen jälkeen järjestelmä siirretään myöhemmin valittavaan pilvipalveluun. Pilvipalveluun siirtämistä ei dokumentoida tähän opinnäytetyöhön.

Opinnäytetyön teoriaosuudessa selvitetään mitä tarkoitetaan koneoppimisella ja keinotekoisilla neuroverkoilla. Koneoppimisen yhteydessä käsitellään myös koneoppimisohjelmistojen oppimisprosessia. Neuroverkkojen yhteydessä kartoitetaan erityyppisiä neuroverkkoja ja hieman niiden matemaattista taustaa. Näitä asioita selvitetään luvuissa 2 ja 3. Luvussa 4 kerrotaan TensorFlow ohjelmistosta tarkemmin ja käydään läpi sen arkkitehtuuri. Siinä kartoitetaan muita TensorFlow'n tukemia koneoppimismalleja.

Opinnäytetyön toteutusvaihe koostui kolmesta vaiheesta: suunnittelusta, toteutuksesta ja testauksesta. Toteutusvaiheesta ensimmäisenä käydään läpi järjestelmän suunnittelu ja toteutus luvussa 5. Tässä vaiheessa myös kartoitetaan, mitä eri tapoja joidenkin osien toteuttamiseen on ja selitetään miksi päädyttiin valittuun tapaan. Toetuksessä käytetyt skriptit on lisätty opinnäytetyön liitteiksi. Järjestelmän testausta käydään läpi luvussa 6.

1.3 Kaukokartoitus nykypäivänä

Kaukokartoitus käsitteellä tarkoitetaan tiedon hankkimista kohteesta analysoimalla sellaista aineistoa, jota ottava laite ei ole ollut kosketuksissa tutkittavan kohteen kanssa. Käsitteeseen sisältyy esimerkiksi kohteen mittaus, aineiston siirto, aineiston analysointi ja jatkokäsittely. Kaukokartoituksella tutkittavia kohteita voivat olla esimerkiksi jotkin maa-alueet, meret tai kasvillisuus. Kaukokartoituksessa tehty mittaus

voidaan suorittaa eri korkeuksilta. Mittaaminen voidaan tehdä maanpinnalta, lentokoneesta tai jopa satelliiteista käsin.(Eskelinen. 2001. 1)

Nykypäivänä luonnonmaantieteessä hyödynnetään alati kehittyviä teknologioita, joista mm. spatiaalinen analytiikka ja kaukokartoitus ovat nousseet keskeisiksi menetelmällisiksi kulmakiviksi. Näiden ja muidenkin teknologioiden avulla luonnonmaantieteilijät pystyvät selvittämään mm. luonnonresurssien käyttömahdollisuuksia ja luonnon kestokyvyn määrää. Tiedon analysoimisilla ja mallinnuksella tieteilijät pystyvät tekemään ennustuksia esimerkiksi siitä minkälaisia vaikutuksia erilaisilla toimenpiteillä on ympäröivään luontoon.(Luonnonmaantiede. 2016.)

Kaukokartoitustekniikat ovat kehittyneet paljon viime vuosikymmenien aikana. Samalla entistä enemmän dataa on helposti saatavilla. Näiden syiden takia kaukokartoituksesta on tullut yhä suositumpi tutkimusmenetelmä ympäristön havainnointiin ja tutkimiseen. Kaukokartoitusmenetelmä mahdollistaa laajojen alueiden tehokkaan mittaamisen ja sillä pystytään havaitsemaan globaalisia ilmiöitä satelliittien avulla. Koska dataa on saatavilla useamman vuosikymmenen ajalta, on mahdollista vertailla tutkittavaa kohdetta menneisyyden ja nykypäivän välillä. Yksi kaukokartoitusmenetelmiä soveltava tutkimusaloista on globaalien muutosten, esimerkiksi ilmastonmuutoksen, havainnointi.(Keto. 2017. 6)

Kaukokartoituksessa käytettävät laitteet tuottavat valokuvia tai digitaalista dataa, joka usein visualisoidaan tietojenkäsittelyohjelmia käyttäen. Nykypäivänä nämä laitteet tuottavat pääasiallisesti digitaalisia kaukokartoituskuvia. Näiden kuvien pikselit kuvaavat tietyn kokoista maa-aluetta ja sen mitattuja ominaisuuksia. Sensorin spatiaalisella resoluutiolla tarkoitetaankin tätä tutkittavan maanpinnan yksityiskohtien erottamiskykyä eli toisin sanoen pikselin sisältämää maapinta-alaa. Esimerkiksi käytöstä poistettujen SPOT-1, -2 ja -3 kaukokartoitussatelliittien HRV-instrumentit (High Resolution Visible) operoivat 20 metrin spatiaalisella resoluutiolla.(Mts. 9.)

2 Koneoppiminen

2.1 Yleistä

Koneoppiminen (engl. Machine learning) on yksi tekoälyn ydinosa-alueista. Koneoppiminen käsitteenä tarkoittaa tietokoneen ohjelmoimista oppimaan käyttäen algoritmeja eikä suoraan ohjelmoida sitä ratkaisemaan haluttua ongelmaa. Toisin sanoen tietokoneella yritetään simuloida ihmiselle ominaista kykyä oppia uutta hyväksikäyttäen mennyttä dataa. Ilman sitä olisi erittäin hankalaa rakentaa minkäänlaista älykästä järjestelmää, koska silloin pitäisi ohjelmoijan ottaa huomioon jokainen mahdollinen ongelmaan liittyvä muuttuja. Koneoppimisen avulla jokaista muuttujaa ei tarvitse ottaa ohjelmoidessa huomioon olettaen, että käytettävät algoritmit ovat toimivia. (Schapire, 2008, 1.)

Kun henkilö kerää paljon dataa jonkin ongelman selvittämiseksi, yleensä datassa alkaa esiintymään jonkinlainen kaava. Riippuen siitä minkälaista dataa ja kuinka paljon sitä on, voi tuo kaava olla liian monimutkainen yhden ihmisen tunnistettavaksi. Tällaisissa tapauksissa ihminen voidaan korvata tietokoneella. Se voi tarkastella tosi suurta määrä dataa ja luoda koodin, jolla voidaan tunnistaa tuo kaava uudesta datasta. Tätä kutsutaan koneoppimiseksi. Tuota koodia voidaan sitten hyödyntää muissa sovelluksissa ja tehdä niistä viisaampia esimerkiksi ne voivat tehdä parempia ehdotuksia siitä, minkä elokuvan asiakas voisi haluta katsoa seuraavaksi.(Chappel, 2015, 3.)

Kun lähdetään ongelmaa ratkaisemaan, on parempi hyödyntää ongelmaan liittyvää dataa, mikäli sitä on saatavilla. Esimerkiksi tarkoituksena olisi luoda sovellus, jonka tarkoituksena olisi erotella vilpilliset rahansiirrot oikeista korkealla onnistumisprosentilla. Yksi vaihtoehto ongelman ratkaisemiseen olisi kerätä muutama älykäs henkilö ratkomaan ongelmaa ja luoda sovellus sen perusteella, mitä he yhdessä ovat keksineet. Tämä lähestymistapa voisi toimia, mutta todennäköisesti ongelma ratkeaisi paremmin dataa tutkimalla. Tosin on otettava huomioon datan määrä, koska mikäli sitä on vähän, ei siitä todennäköisesti saa selville oikeata kaavaa. Käyttäen jo mainittua vilpillisten rahansiirtojen tunnistusesimerkkiä, tutkitaan taulukkoa 1. Heti voidaan huomata datan vähäisyyden takia yhtäläisyys molempien vilpillisten rahansiirtojen kesken. Yhtäläisyys on yksinkertaisesti se, että heidän nimensä alkavat N-kirjaimella.

Tämä todennäköisesti ei päde suuremmassa mittakaavassa, joten kyseinen kaava on suurella todennäköisyydellä virheellinen.(Mts. 3.)

Taulukko 1. Koneoppimisen ensimmäinen esimerkki

Sukunimi	Määrä	Vilpillinen
Kuusinen	2052 €	Ei
Niemi	1600 €	Kyllä
Nuhtamoinen	1004 €	Kyllä
Pouta	8000 €	Ei

Datan määrän kasvaessa tarkemman kaavan löytäminen varmistuu, mutta samalla se todennäköisesti on monimutkaisempi ja hankalampi löytää. Taulukko 2 pitää sisällään huomattavasti paljon enemmän tietoa rahasiirroista, ja niitä on enemmän kuin taulukossa 1. Taulukon ”Missä myönnetty”-sarake tarkoittaa, missä henkilön pankkikortti on myönnetty, ja ”missä käytetty” -sarake tarkoittaa, missä maassa rahansiirto on tehty. Siinä myös esiintyy neljä vilpillistä rahansiirtoa, mutta mikähän niissä on yhtenäistä? Ei ainakaan aiemman taulukon perusteella löytynyt kaava, eli kaikkien vilpillisten rahansiirtojen tekijöiden nimet eivät ala N-kirjaimella. Henkilöillä, joiden ikä on 20 – 30 vuotta, on aika suuri vilpillisyysprosentti, mutta kaikki heistäkään eivät ole tehneet vilpillisiä rahansiirtoja. Tästäkin määrästä dataa on aika helppo löytää kaava. Rahansiirto on vilpillinen, mikäli kortin omistajan ikä on 20 - 30 vuotta, kortti on myönnetty Suomessa ja käytetty Venäjällä ja rahaa on siirretty enemmän kuin 1000€. (Mts. 3 – 4.)

Taulukko 2. Koneoppimisen toinen esimerkki

Sukunimi	Määrä	Missä myönnetty?	Missä käytetty?	Ikä	Vilpillinen
Kuusinen	2052 €	FIN	USA	21	Ei
Niemi	1600 €	FIN	RUS	28	Kyllä
Nuhtamoinen	1004 €	FIN	RUS	29	Kyllä
Pouta	8000 €	SWE	RUS	65	Ei
Niiranen	300 €	RUS	FIN	49	Ei
Mäkelä	2500 €	USA	FIN	46	Ei

Saarinen	7300 €	FIN	RUS	23	Kyllä
Toivonen	7100 €	FIN	FIN	35	Ei
Rantala	600 €	FIN	RUS	24	Ei
Rautiainen	6500 €	FIN	RUS	20	Kyllä

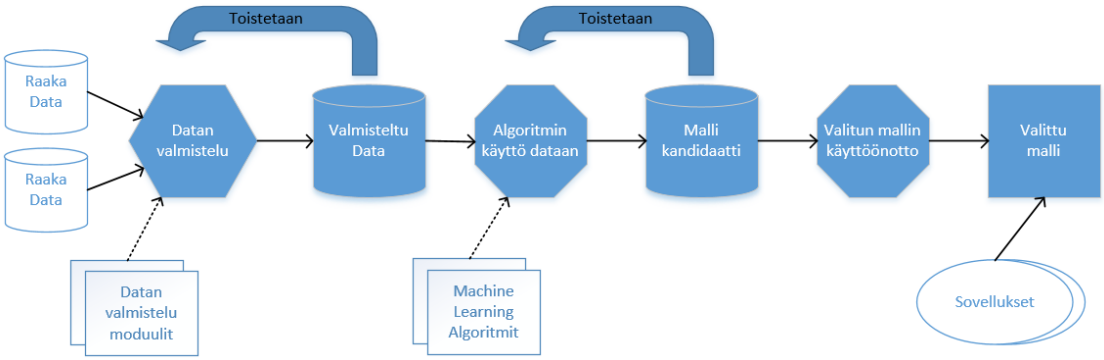
Näistä esimerkeistä voi jo huomata, että mitä enemmän dataa on, sitä kauemmin kaavan löytämisessä kestää. Toki tämä myös pätee tietokoneella kaavan löytämiseenkin, mutta on se silti huomattavasti nopeampaa kuin ihmisellä. Tämä käy varsinkin selväksi, kun dataa on esimerkiksi tuhat kertaa enemmän kuin mitä oli taulukossa 2. Suuren datamäärän tutkimiseen ihmisen on käytettävä tilastotieteellisiä tekniikoita, sillä pelkästään datan katsomisella ei todennäköisesti löydy oikeaa kaavaa, kun sitä on tosi paljon.(Mts. 4.)

Koneoppiminen on juuri sitä: tietokone/ohjelmisto käyttää tilastotieteellisiä tekniikoita suuren datamäärän läpikäymiseen etsiäkseen siitä parasta kaavaa ongelman ratkaisemiseen. Kaavan löytämisen jälkeen se generoi koodia, joka pystyy tunnistamaan sen kaavan. Tätä koodia kutsutaan malliksi, ja muut sovellukset pystyvät kutsumaan sitä käyttöönsä ratkoakseen halutun ongelman. Esimerkiksi aiemmin läpi käyty vilpillisyyden tunnistaminen, kutsuvan sovelluksen on tarjottava kaikki tarpeelliset tiedot rahansiirrosta mallille: henkilön ikä, missä kortti on myönnetty, missä sitä on käytetty ja kuinka paljon rahaa siirrettiin. Malli näiden tietojen perusteella palauttaa vastauksena onko tämä siirto todennäköisesti vilpillinen vai ei.(Mts. 4)

Koneoppimista voidaan käyttää paljon muuhunkin kuin pelkästään petoksen tunnistamiseen. Koneoppimisen käyttötarkoituksilla on yhteistä se, että ne vaativat paljon ratkaistavana olevaan ongelmaan liittyvää dataa. Ongelmasta riippuen tämä data voi olla esimerkiksi kuvia, tekstiä ja videoita. Jokainen käyttötarkoitus myös vaatii jonkin koneoppimisohjelmiston, jolla pystytään ongelmaa varten luomaan oikeanlainen algoritmi. Muita koneoppimisen käyttötarkoituksia on esimerkiksi mustavalkoisten kuvien värittäminen, äänien lisääminen videoihin ja tekstin kääntäminen toiselle kielelle.(Brownlee. 2016.)

2.2 Oppimisprosessi

Oppimisprosessi on sama jokaisella koneoppimisohjelmistolla. Tätä prosessia on kuvattu kuviossa 1. Prosessissa on muutama vaihe, ja kahta niistä toistetaan, kunnes ollaan tyytyväisiä lopputulokseen.(Chappel, 2015, 5.)



Kuvio 1. Oppimisprosessi

Kuten kuviosta nähdään oppimisprosessi alkaa raa'alla datalla. Mitä enemmän dataa on saatavilla, sitä parempi vastaus todennäköisesti saadaan. Koska big datan suosio kasvaa koko ajan, on meillä saatavilla paljon dataa eri alueilta. Tämä mahdollistaa koneoppimisen hyödyntämisen useampaan ongelmaan. Mitä tahansa dataa ei kuitenkaan kannata syöttää ohjelmalle vaan ainoastaan se data, joka voi vaikuttaa selvitettävänä olevaan ongelmaan. Esimerkiksi suihkumoottorin huoltoajankohdan määrittämiseen voisi esimerkiksi tarvita seuraavanlaista dataa: kuinka pitkiä matkoja sillä on lennetty, kuinka kauan sillä on lennetty, missä säässä on lennetty ja kuinka usein se on sekä käynnistetty että sammutettu.(Mts. 5.)

Haluttu data ei välttämättä ole heti oikean muotoista järjestelmällemme, joten se pitää muotoilla uudestaan. Yleensä koneoppimisohjelmistot tarjoavat datan valmistelu moduuleita, joilla tämän pystyy tekemään. Tämä datan valmistelu voi esimerkiksi tarkoittaa duplikaatti tietojen poistamista tai, jos on tarkoitus käsitellä kuvia, kuvien koon muuttamista. Datasta voidaan myös poistaa turhaksi osoittautuvia asioita, oli ne sitten esimerkiksi vääränlaisia kuvia tai kuvia, jotka pitävät sisällään vääriä asioita. Datan valmisteluun voi kulua paljon aikaa, mikäli raakaa dataa pitää muotoilla use-

asti. Datan valmisteluakaan ei kannata jättää kesken, sillä algoritmit yleensä on suunniteltu vastaan ottamaan tietyn muotoista dataa ja kohdatessaan erimuotoista dataa voivat ne antaa erikoisia tuloksia.(Altinas, I., Nguyen, M. N.d.)

Kun data on muotoiltu oikeanlaiseksi, voidaan siihen ruveta käyttämään algoritmeja. Algoritmit yleensä tekevät jonkinlaista tilastotieteellistä analyysiä annettuun dataan. Nämä voivat pitää sisällään esimerkiksi suhteellisen yleisiä asioita kuten regressio ja mahdollisesti paljon monimutkaisempiakin kuten kaksitasoisia päätöspuita. Mahdollisesti ei ole varmaa, mikä algoritmi on paras ongelman ratkaisemiseen. Tällaisissa tapauksissa kokeillaan useampaa algoritmia dataan ja selvitetään mikä niistä antaa parhaimman tuloksen.(Chappel, 2015, 6.)

Kun testidataan on ajettu valittu algoritmi, syntyy malli. Malli on siis algoritmin synnyttämää koodia, jota käytetään datassa olevan kaavan tunnistamiseen. Tämä malli antaa vastauksen kysyttyyn ongelmaan ja muut sovellukset voivat kysyä siltä esimerkiksi: ”Onko tämä rahansiirto vilpillinen?”, minkä jälkeen malli antaa vastauksen. Malli ei kylläkään anna suoraa kyllä-ei vastausta, vaan se palauttaa todennäköisyyden 0 – 1 välillä. Todennäköisyyden tulkitseminen jää siten ihmisten vastuulle. Esimerkiksi malli palauttaa rahansiirrolle 0.8 ”vilpillisyystodennäköisyyden”, eli voimme olla kohtuullisen luottavaisia sen vilpillisyydestä. Jos se palauttaisi rahansiirrolle 0.5, olisi se paljon hankalampi kategorisoida.(Mts. 6.)

Kuten algoritmien yhteydessä mainittiin, ensimmäinen saatu malli tuskin on se kaikkein tehokkain. Tämän takia koneoppimisalgoritmeja vaihdellaan ja datan muotoilusääntöjä muutetaan. Kun tehokkain malli on löytynyt, seuraava vaihe on sen käyttöönotto. Ilman käyttöönottoa oli aiemmin tehty työ turhaa. Käyttöönoton jälkeen voidaan mallia hyödyntää uuteen dataan, jolloin ei ihmisen välttämättä tarvitse dataa tutkia niin tarkasti.(Mts. 7.)

3 Neuroverkot

3.1 Yleistä

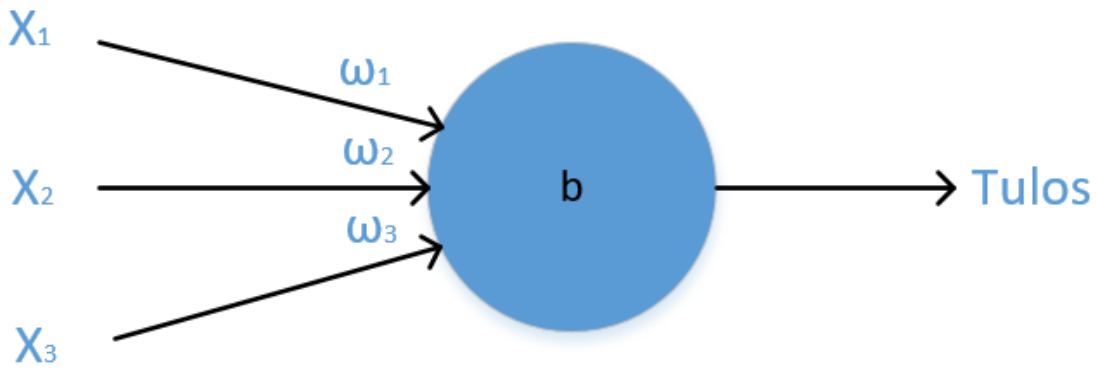
Keinotekoisella neuroverkko on suunniteltu prosessoimaan tietoa samantapaisesti kuin ihmisen aivot. Aivojen kyky oppia kokemuksista todistaa, että ongelmat, joita

nykyteknologialla ei voi ratkoa, on ratkottavissa. Tämä tiedon prosessointimalli ajatellaan olevan seuraava suuri edistysaskel tiedonkäsittelyssä. Nimittäin jopa yksinkertaiset eläimen aivot pystyvät toimintoihin, jotka ovat mahdottomia nykytietokoneille. Tämä ero aivojen ja tietokoneiden välillä ainakin osittain johtuu tiedontallennustapojen erosta. Aivot nimittäin tallentavat tietoa malleina. Jotkin näistä malleista ovat tosi monimutkaisia, mikä mahdollistavaa esimerkiksi ihmisten tunnistaa henkilöiden kasvot monesta eri kulmasta.(Artificial Neural Networks Technology. N.d.)

Ensimmäiset tutkimukset keinotekoisiin neuroverkkoihin tehtiin 1940-luvulla, jolloin McCulloh ja Pitts esittelivät ensimmäisen neuroverkko tiedonkäsittelymallin. 1950-luvulla neuroverkkojen kehitys jatkoi ja silloin Rosenblatt sai aikaan kaksi tasoisen neuroverkon ja perceptron niminen neuroni, mikä pystyi oppimaan tiettyjä luokitteluja muuttamalla algoritmin painoja. Vaikka perceptron oli toimiva tiettyjen mallien tunnistamisessa, sillä oli myös rajoitteita ja nämä rajoitteet aiheuttivat neuroverkkojen kiinnostuksen heikkenemiseen. 1980-luvun alussa tutkijoiden kiinnostus alaa kohtaa heräsi uudestaan. Viimeisimmät neuroverkkoihin liittyvät tutkimustulokset sisältävät Boltzmannin kone, Hopfieldin verkko ja monen kerroksen verkot.(Russel. 1996.)

3.2 Sigma neuroni

Sigma neuroni on kehittyneempi neuroni aiemmin mainittuun perceptron neuroniin verrattuna. Nämä neuronit ovat keskenään hyvin samanlaisia, mutta yhden asian takia sigma on parempi näistä kahdesta. Se liittyy näiden neuronien antamaan tulokseen. Perceptron voi antaa vain tulokset 0 ja 1, kun taas sigma antaa tuloksen 0 ja 1 välillä. Tämä ei välttämättä äkkiseltään näytä hirveän tärkeältä erolta, mutta tutkitaanpa neuronien toimintaa ja vertaillaan näitä kahta. Kuviossa 2 on yksinkertaistettu neuronin toimintaa. Neuronille voi tulla yksi tai useampi syöte riippuen toteutuksesta. Kuvion esimerkissä sille tulee kolme X_1, X_2 ja X_3. Näihin syötteisiin lisätään jokin painoarvo, joka on eri jokaiselle syötteelle. Niihin lisätään myös neuronille ominainen vinouma b (engl. bias). Näiden kolmen arvon avulla neuroni antaa tuloksen.(Nielsen. 2017.)



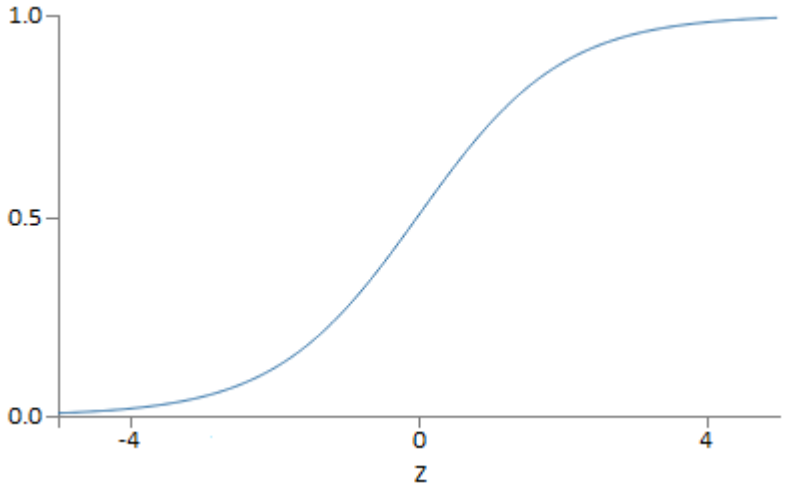
Kuvio 2. Sigma neuronin toiminta

Kun neuronit oppivat ne muuttavat syötteille annettuja painoarvoja ja omaa vinoumaansa. Tästä voi seurata ongelmia perceptronien kanssa, sillä pieni muutos näissä arvoissa voi kääntää sen antaman tuloksen nollasta ykköseen tai toisinpäin. Tämä voi aiheuttaa odottamattomia ongelmia perceptron pohjaisissa neuroverkoissa. Sigma neuronien kanssa ei ole tällaista ongelmaa painoarvojen ja vinouman muuttamisen kanssa, koska niiden muutokset eivät muuta tulosta dramaattisesti. Se muuttaa neuronin antamaa tulosta vain hiukan. Tämä fakta tekee sigmapohjaisista neuroverkoista helpommin opetettavia. Sigma neuronit pystyvät myös vastaanottamaan syötteitä, jotka ovat nollan ja yhden välissä.(Mts.)

Sigma neuronit antavat tuloksen seuraavan kaavan perusteella σ(w * x + b). σ symboli tarkoittaa sigma funktiota, mikä on kaavan 1 mukainen. (Sigmoid Function. N.d.)

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{1}$$

Tämä matemaattinen kaava voi vaikutta monimutkaiselta, varsinkin kun z:n paikalle laitetaan tuo aiemmin mainittu sarja, varsinkin silloin kun neuronille tulee monta eri syötettä. Näiden neuronien yhteydessä on kuitenkin parempi muistaa minkä näköinen kuvaaja syntyy logistisesta funktiosta, mikä on esitetty kuviossa 3. Kuten kuviossa näkyy, funktio ei ikinä tule saamaan arvoja yksi tai nolla. Se kuitenkin menee äärettömän lähelle kumpaakin. Kuviosta myös huomaa, että funktion kuvaaja ei muutu radikaalisti missään välissä, mistä pohjautuu aiemmin mainittu sigma neuronien etu.(Mts.)

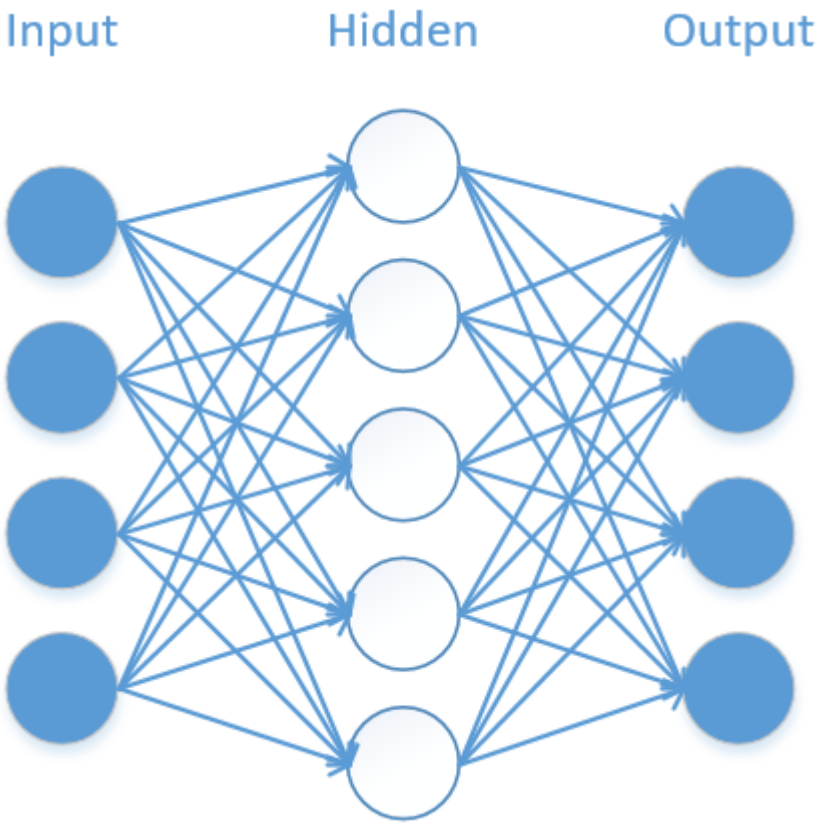


Kuvio 3. Logistisen funktion kuvaaja

3.3 Neuroverkkomallit

3.3.1 Myötäkytkentä neuroverkko

Kuviossa 4 on kuvattu myötäkytkentä neuroverkon (engl. Feedforward neural network, FNN) rakennetta. Neuroverkon rakenne jaetaan kolmeen kerrokseen: syöte, piilotettu ja ulostulo. Myötäkytkentä verkoissa edellisen kerroksen tulosta käytetään seuraavan kerroksen syötteenä. Piilotetun kerroksen sisällä voi olla yksi tai useampikin neuronirivi. Syötekerros nimensä mukaisesti vastaanottaa neuroverkkoon syötettävää tietoa. Syötekerroksen neuronit laskevat niihin tulevasta tiedosta jonkin tuloksen ja siirtävät sen jokaiselle piilotetun kerroksen ensimmäisen neuronirivin neuroneille. Piilotetun kerroksen ensimmäisen neuronirivin neuronit laskevat syötekerroksen neuronien antamien tulosten avulla omat tulokset. Nämä tulokset ne siirtävät joko seuraavalle piilotetun kerroksen neuroniriville tai ulostulokerrokselle, mikäli neuroverkon piilotetussa kerroksessa on vain yksi neuronirivi. Ulostulokerros antaa nimensä mukaisesti neuroverkon tuottaman tuloksen. (Nielsen. 2017.)



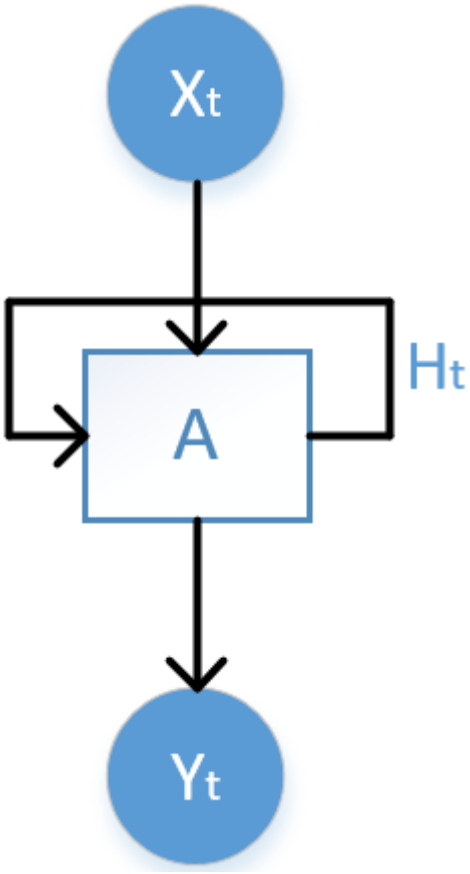
Kuvio 4. Myötäkytketyn neuroverkon rakenne

Kuten kuviosta huomaa kaikki paitsi ulostulokerroksen neuronit syöttävät tuloksensa jokaiselle seuraavan neuronirivin neuronille. Jokaiselle saamalleen syötteelle neuroni antaa jonkin painoarvon kuten aiemmin mainittiin, mistä johtuu neuroverkon kouluttamisen suuri ajankulutus. Verkon monimutkaisuuden lisääminen kasvattaa verkon tarkkuutta, mutta samalla se vaatii enemmän resursseja sitä ajavalta tietokoneelta.(Mts.)

3.3.2 Toistuva neuroverkko

Myötäkytkentä neuroverkot eivät huomioi edellistä syötettä seuraavan syötteen käsittelyssä, mikä aiheuttaa ongelmia esimerkiksi lauseeseen kuuluvan seuraavan sanan ennustamisessa. Toistuva neuroverkko (engl. recurrent neural network, RNN) on suunniteltu ottamaan huomioon edellisen syötteen, kun se käsittelee seuraavaa syötettä. Tämä saadaan aikaan luomalla silmukoita verkkoon. Kuviossa 5 on kuvattu, miten toistuva neuroverkko käsittelee saamiaan syötteitä (X_t). Se ilmoittaa tuloksen Y_t

ja samalla tallentaa niistä muistiin arvon H_t. Tämän muistissa olevaa arvoa neuroverkko hyödyntää kun se käsittelee seuraavaa syötettä X_{t+1}. Neliöllä A tarkoitetaan koko verkkoa.(Olah. 2015.)

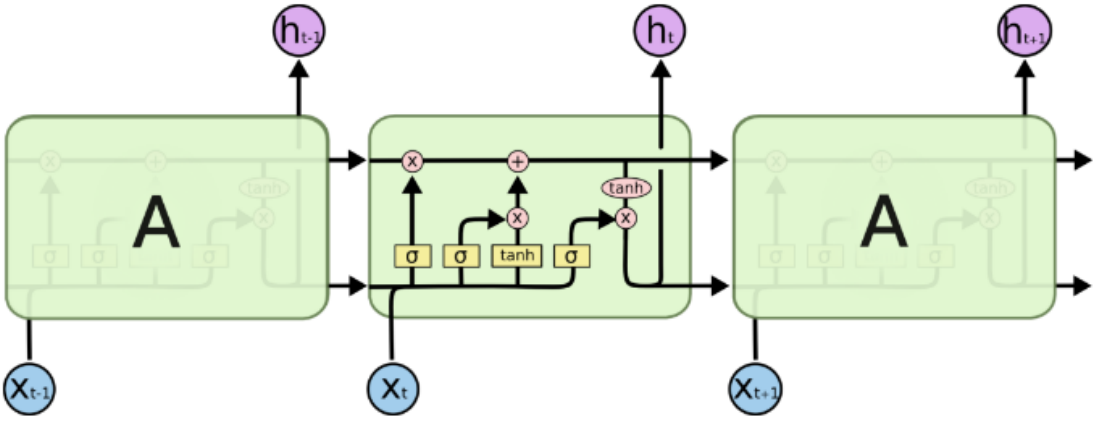


Kuvio 5. Toistuva neuroverkko

Normaaleilla toistuvilla neuroverkoilla on kuitenkin rajoitteita, kuinka pitkälle ne voivat muistaa entisiä syötteitä. Tämä käy hyvin selväksi, jos tällaisia verkkoja käytetään esimerkiksi lauseen seuraavan sanan ennustamisessa. Verkko pärjää lyhyiden yksittäisten lauseiden kanssa erittäin hyvin, koska niistä käy riittävän hyvin ilmi konteksti. Ongelmia ilmenee, kun verkolle syötetään esimerkiksi tekstikappale, jonka ensimmäisessä lauseessa on kerrottu, että henkilö asuu Saksassa, ja viimeinen lause olisi ”Hän puhuu…”. Verkko todennäköisesti tunnistaa, että seuraavan sanan pitäisi olla jokin kieli, mutta se ei todennäköisesti osaa antaa juuri saksan kieltä. Teoriassa tämän ei pitäisi olla ongelma toistuville neuroverkoille, mutta käytäntö on osoittanut päinvastaista. Yoshua Bengio, Patrice Simard ja Paolo Frasconi olivat tutkineet tätä ongelmaa vuonna 1994. Ongelman ymmärtämistä varten he yrittivät rakentaa järjestelmän,

joka pystyy täyttämään kolme vaatimusta. Sen pitää pystyä tallentamaan tietoa satunnaisen mittaisen ajanjakson ajan. Sen pitää myös pystyä sietämään sekä satunnaista syötteiden vaihtelua että syötteen epäoleellista vaihtelua. Järjestelmän kouluttaminen ei saa myöskään kestää liian kauan. He päätyivät tulokseen, että joko järjestelmä on herkkä syötteiden vaihtelulle tai törmätään ”vanishing gradient” nimiseen ongelmaan, eli yksinkertaistettuna verkko antaa paljon enemmän painoarvoa viimeisimmille tuloksille kuin vanhemmille tuloksille.(Bengio, Frasconi & Simard. 1994.)

Yksi ratkaisu juuri mainittuun ongelmaan ovat pitkät lyhytaikaisen muistin verkot (engl. Long Short Term Memory Network, LSTM). Tällaiset verkot pystyvät oppimaan pitkäaikaisia riippuvuuksia. Ensimmäisen tällaisen toistuvan neuroverkon variantin esittelivät Hochreiter ja Schmidhuber vuonna 1997. Ensiesittelyn jälkeen LSTM-verkon kehittelyä on jatkettua, ja nykyään siitä on monta erilaista varianttia. Kaikille niille on kuitenkin yhteistä se, että ne kaikki väistävät normaalien toistuvien neuroverkkojen pitkäaikaisriippuvuusongelman ja niiden oletuskäyttäytyminen on muistaa tietoa pitkiä ajanjaksoja. Näiden verkkojen rakenne muistuttaa normaalien toistuvien neuroverkkojen tapaista ketjumaista rakennetta. Kuviossa 6 on esitetty tämä LSTM verkon perusrakenne. (Olah. 2015.)



Kuvio 6. LSTM verkon rakenne

LSTM:n rakenteen tärkein osa on solun tilaa(engl. cell state) ja se kulkee suoraan koko verkon läpi. Sen läpi tietoa voi kulkea muuttumattomana koko verkon läpi, mutta tarpeen vaatiessa siihen voidaan myös lisätä tai poistaa tietoa. Kuviossa 6 olevat keltaiset laatikot tarkoittavat neuronikerroksia ja punaiset ympyrät tarkoittavat pistettäisiä operaatioita. (Mts.)

Rakenteen ensimmäisen pystyviivan sigma neuronikerrosta kutsutaan unohdusporttikerrokseksi(engl. forget gate layer) ja kuten nimestä voi päätellä, kyseinen kerros tekee päätöksen siitä, mitä verkko haluaa ”unohtaa” eli poistaa solun tilasta. Unohdusportti laskee arvojen X_t(syöte) ja h_{t-1}(piilotettu tila) avulla tuloksen, joka saa arvoja 0 ja 1 välillä, jokaiselle solu tilassa sijaitsevalle arvolle. Arvolla 0 solun tilassa oleva arvo poistetaan kokonaan ja arvolla 1 kyseinen solun tilassa oleva arvo säilytetään kokonaan. Seuraavaksi on päätettävä, mitä uutta tietoa tallennetaan solun tilaan. Tämä operaatio tehdään kaksiosaisesti. Ensimmäisenä päätetään mitä arvoja solun tilassa päivitetään ja tämän operaation hoitavaa sigma neuronikerrosta kutsutaan syöteporttikerrokseksi(engl. input gate layer). Toisena osana luodaan vektori uusista arvoista, jotka voitaisiin lisätä soluntilaan ja tämän operaation tekee hyperbolinen tangentti neuronikerros(kuviossa tanh kerros).(Mts.)

Kun on tehty päätös siitä, mitä halutaan solun tilasta unohtaa ja mitä siihen halutaan lisätä, on aika päivittää solun tila. Entinen soluntila ensin kerrotaan unohdusporttikerroksen tuloksella ja sen jälkeen siihen lisätään syöteporttikerroksen ja hyperbolisen tangenttikerroksen tulo. Tämän voi mieltää helposti seuraavalla tavalla: Muistista poistetaan ne asiat, jotka halutaan unohtaa ja muistiin lisätään uusia asioita, jotka halutaan muistaa.(Mts.)

Solun tilan päivittämisen jälkeen on vielä annettava jokin tulos. Tätä varten yksi sigma neuronikerros laskee mitä osia solun tilasta halutaan antaa tulokseksi, minkä lisäksi solun tila ajetaan hyperbolisen tangentin läpi. Funktion läpiajaminen siirtää solun tilassa olevien arvojen siirtymisen -1 ja 1 välille. Funktion läpi käymisen jälkeen solun tilassa olleet arvot kerrotaan sigma neuronikerroksen tuloksen kanssa.(Mts.)

Nämä operaatiot voidaan myös esittää funktio muodossa, jotka voivat olla yksinkertaisempia ymmärtää kuin kuvioversio. Kaava 2 esittää kuviossa olevaa unohdusporttikerrosta f_t. Kaavassa verkon piilotetusta tilasta ja syötteistä luodaan matriisi. Tämä matriisi kerrotaan funktion painoarvolla ja siihen lisätään vinouma. Näiden laskutoimitusten jälkeen tulos syötetään kaavan 1 mukaisen sigma funktion z:n paikalle.(Srihari. N.d. 8.)

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \quad (2)$$

Seuraavat kaavat 3 ja 4 liittyvät tiedon lisäämiseen solun tilaan. Kaava 3 on syöteporttikerrosta kuvaava funktio ja kaavalla 4 lasketaan uudet arvot, jotka lisätään solun tilaan. Syöteporttikerroksen kaava on lähes samanlainen kuin unohdusporttikerroksen kaava, mutta sen paino arvo ja vinouma ovat erikokoiset. Tämän eron lisäksi uusien arvojen kaavassa sigma korvataan hyperbolisella tangentilla.(Mts. 9.)

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \quad (3)$$

$$\tilde{C}_t = \tanh(W_C * [h_{t-1}, x_t] + b_C) \quad (4)$$

Kaavalla 5 päivitetään se solun tila vastaamaan nykytilannetta. Kertauksena solun tila kerrotaan unohdusporttikerroksen funktion tuloksella ja siihen lisätään syöteporttikerros funktion ja uusien arvojen funktion tuloksella.(Mts. 10.)

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (5)$$

Kaavoilla 6 ja 7 lasketaan annettava tulos. Kaavalla 6 lasketaan mitä solun tilasta halutaan antaa tulokseksi ja kaavalla 7 lasketaan se annettava tulos. Kaava 6 on taas lähes sama, kuin aiemmat sigma funktiot, mutta siinäkin sen paino arvo ja vinouma ovat erikokoiset. Tuloksen saamiseen solun tila syötetään hyperboliseen tangenttiin ja se kerrotaan kaavan 6 tuloksella.(Mts. 11.)

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \quad (6)$$

$$h_t = o_t * \tanh(C_t) \quad (7)$$

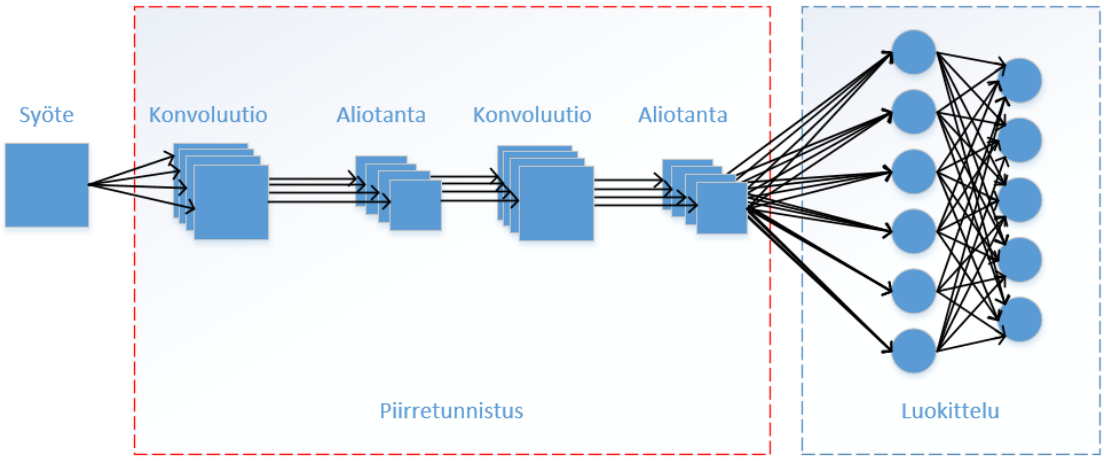
3.3.3 Konvoluutio neuroverkko

Konvoluutio neuroverkot (engl. convolutional neural network, CNN) ovat moniker-roksisia myötäkytkentä neuroverkkoja. Ne kuitenkin eroavat normaaleista FNN verkoista siten, että CNN verkoissa joidenkin kerroksien neuronit eivät ole kaikkien seuraavien kerroksien neuronien kanssa yhteyksissä. CNN verkkojen arkkitehtuuri perustuu Hubel ja Wiesel nimisten henkilöiden tekemään tutkimukseen. Tämä tutkimus koski kissojen näköaivokuoressa tapahtuvaan visuaaliseen prosessointiin. Tästä joh-

tuen CNN:iä käytetään yleensä kuvien tarkasteluun ja niistä objektien tunnistamiseen, mutta on kuitenkin todistettu niiden toimivan muissakin erilaisissa käyttötapauksissa.(Vojt. 2016. 16.)

CNN:ssä olevat neuronit käsittelevät vain pientä osaa kuvasta. Näistä pienistä osista neuronit etsivät niille ennalta määriteltyjä piirteitä, joita voi olla esimerkiksi pysty-, vaakaviivat ja ympyrät. Jokainen kerros on ohjelmoitu etsimään eri piirteitä ja kun kuvan osat ovat kulkeneet verkon läpi, näitä piirrekokonaisuuksia käytetään kuvan luokitteluun.(Mts. 16.)

CNN verkon rakenne on suhteellisen yksinkertainen, mikä on esitetty kuviossa 7. CNN verkko on jaettavissa kahteen osaan: piirteiden tunnistus- ja luokitteluosaan. Molempien osien pituudet riippuvat siitä, kuinka syvä verkko halutaan luoda. Piirteiden tunnistus osan viimeisen aliotantakerroksen jälkeen seuraa yksi tai useampi täysin yhdistetty kerros, mitkä kuuluvat verkon luokitteluosaan. CNN verkonkin tapauksessa verkon viimeistä kerrosta kutsutaan tuloskerrokseksi. (Mts. 16.)



Kuvio 7. CNN verkon rakenne

CNN:ssä on kahden tyyppisiä piirteiden tunnistuskerroksia: konvoluutio- (engl. convolutional layer) ja aliotantakerroksia (engl. subsampling layer). Konvoluutiokerroksen tarkoituksena on tehdä piirteiden tunnistaminen syötekuvista. Se käyttää siihen monia piirrekarttoja (engl. feature map), jotka tunnistavat eri piirteitä. Konvoluutiokerroksen tekemän piirteiden tunnistamisen voi ajatella samanlaisena kuin ajaisi sen

osa kuvan suodattimen läpi. Suodatin joko korostaa tai vaimentaa pikselin potentiaalia riippuen sen ympärillä olevista pikseleistä. Suodattaminen tehdään kerroksen painoja muuttamalla ja painojen muuttuminen tapahtuu oppimisprosessin aikana. Konvoluutiokerrokselle tulevat kuvan osat ovat hieman päällekkäin, minkä ansiosta se kestää pientä kuvan vääristymistä.(Mts. 16–18)

Aliotantakerros voi olla verkon ensimmäinen kerros (syötekerros), mutta se on aina konvoluutiokerroksen jälkeen ja se pitää sisällään saman verran piirrekarttoja kuin sitä edeltävässä konvoluutiokerroksessa. Aliotantakerroksen tarkoituksena on pienentää piirrekarttojen kokoa, jotta tunnistetut piirteet olisivat yksinkertaisempia ja yleistettyjä. Aliotantakerroksen piirrekarttojen syötteenä käytetään konvoluutiokerroksen vastaavia piirrekarttoja, minä johdosta jokainen konvoluutiokerroksen neuroni on yhdistetty vain yhteen aliotantakerroksen neuroniin. Aliotantakerrokselle tulevat kuvan osat eivät mene ollenkaan päällekkäin.(Mts. 16,19.)

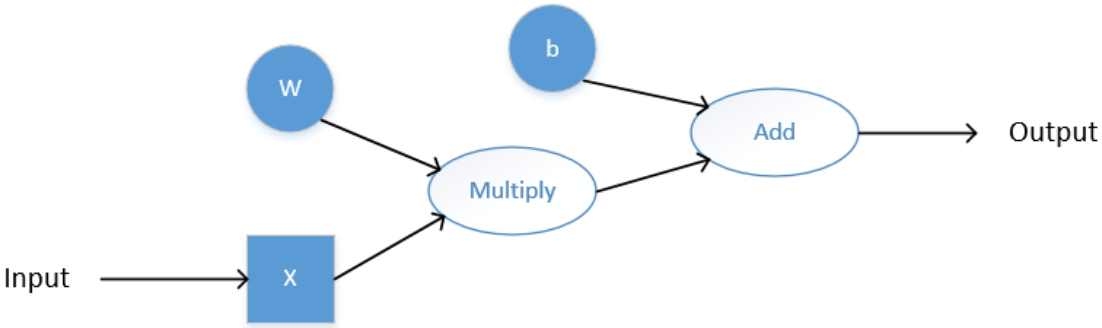
Koska CNN verkot ovat erikoisvariantteja FNN verkoista, on niiden kouluttaminen mahdollista samalla backpropagation algoritmilla. Kyseisen algoritmin tarkoituksena on pienentää verkon väärien tuloksien määrää käyttäen gradient descend nimistä tekniikkaa. Kyseistä tekniikkaa käyttäen kerrosten painoarvoja lähdetään muuttamaan verkon viimeisestä kerroksesta lähtien. Tämä kouluttaminen on valvottua kouluttamista, sillä jokaiselle koulutussyötteelle on tiedossa haluttu tulos. Toisin sanoen esimerkiksi jokaisen syötekuvan lisäksi verkolle annetaan tieto siitä, mikä kuvassa oleva objekti on. Kaavan 8 mukaan pystymme laskemaan virheen tuloskerroksen jokaiselle neuronille. Kaaviossa m_k on tuloskerroksella olevien neuronien määrä. Kirjain j tarkoittaa tarkasteltavaa neuronia, y^p tarkoittaa neuronilta saatua tulosta ja d^p tarkoittaa haluttua tulosta.(Mts. 12.)

$$E_p = \frac{1}{m_k} \sum_{j=1}^{m_k} \left(e_j^p\right)^2 = \frac{1}{m_k} \sum_{j=1}^{m_k} \left(y_j^p - d_j^p\right)^2 \quad (8)$$

4 TensorFlow

4.1 Yleistä

TensorFlow on Googlen kehittämä avoimen lähdekoodin koneoppimisohjelmisto. Se pystyy jakamaan laskutehtäviä sekä monelle eri tietokoneelle että yhden tietokoneen sisällä prosessorille ja näytönohjaimille. Se myös pystyy hyödyntämään erikoisvalmistettuja ASIC-piirejä, joita kutsutaan Tensor Processing Uniteiksi (TPU). TensorFlow käsittelee tietoa tietovuokaavion (engl. data flow graph) mukaisesti. Kuviossa 8 on esitetty yksinkertainen tietovuokaavio. Tietovuokaaviossa solmut tarkoittavat matemaattisia operaatioita ja nuolet tarkoittavat moniulotteisia taulukoita, joita kutsutaan tensoreiksi. Tensori voi olla mm. vektori tai matriisi. Kyseisellä tietovuokaaviolla on esitetty, miten TensorFlow ymmärtää funktion f= W*X + b. (Abadi, Barham, Chen, Chen, Davis, Dean, Devin, Ghemawat, Irving, Isard, Kudlur, Levenberg, Monga, Moore, Murray, Steiner, Tucker, Vasudevan, Warden, Wicke, Yu & Zheng, 2016)



Kuvio 8. Tietovuokaavio

TensorFlow'n saapuva syöte sijoitetaan erikoismuuttujaan nimeltä paikanpitäjä (engl. placeholder), jota on kuvattu X:llä. W ja b ovat normaaleja muuttujia. Normaalit muuttujat tarvitsevat jonkin aloitusarvon, kun taas paikanpitäjämuuttuja ei sitä tarvitse. Paikanpitäjämuuttujalle pitää kuitenkin määrittää, minkä muotoista dataa se säilyttää. Näiden kahden muuttujan käyttötarkoitukset myös eroavat toisistaan. Normaaleja muuttujia käytetään koneoppimisessa sellaisten arvojen säilyttämiseen, mitä käytettävä malli kouluttaa oppimisprosessin aikana. Paikanpitäjämuuttujia käytetään

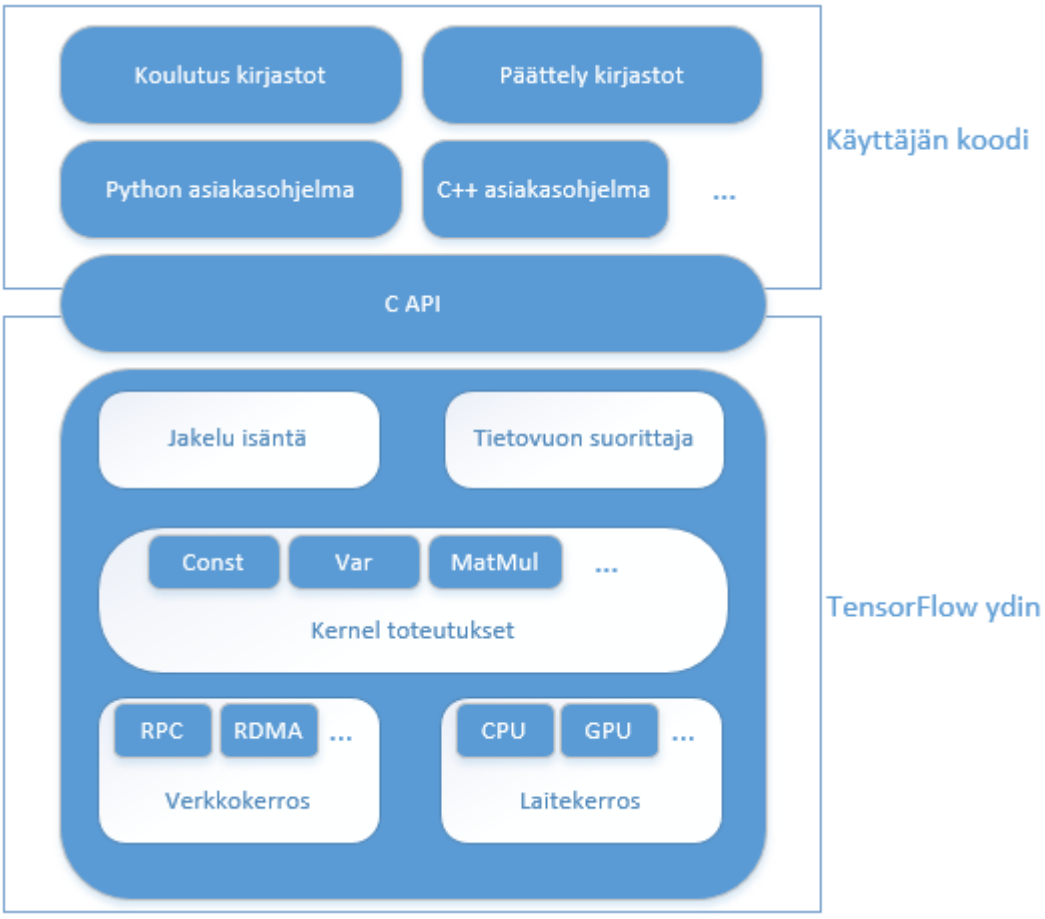
malliin syötettävien syötteiden väliaikaiseen säilyttämiseen.(TensorFlow Architecture. 2017.)

4.2 Arkkitehtuuri

TensorFlow:n voi jakaa rakenteeltaan kahteen osaan: käyttäjän luomaan omaan koodiin ja TensorFlow'n ytimeen. Käyttäjän koodin osuudessa TensorFlow'lle kerrotaan, minkälainen tietovuokaavio halutaan. Tämän ohjelmoinnin voi tehdä muutamalla eri kielellä, mutta yleisimmät niistä ovat Python ja C++. Näistä kahdesta Pythonille on tarjolla enemmän koulutuskirjastoja kuin C++:lle. Tietovuokaavion ”rakentamisen” jälkeen käyttäjän koodiosuus käynnistää sen suorittamisen. C API toimii jakajana TensorFlow'n rakenteessa ja se välittää käyttäjän koodin osuudelta tulevat käskyt TensorFlow'n ytimelle.(Abadi ym. 2016)

TensorFlow:n ydin on kirjoitettu C++:lla. Jakeluisäntä vastaanottaa nämä käskyt ja tarkastaa tietovuokaavion. Siitä se ”irrottaa” sen osion, mitä halutaan suorittaa (voi olla myös koko tietovuokaavio) ja jakaa sen pienempiin osiin, joita voidaan suorittaa samanaikaisesti. Nämä osat se sitten jakaa tietovuon suorittajille, jotka ajoittavat osion vaatimien kerneleiden suorittamisen. Se pystyy ajoittamaan niiden suorittamista esimerkiksi useammalle prosessorin ytimelle tai näytönohjaimelle.(Mts.)

Laitetasolla TensorFlow pystyy siirtämään paikallisten prosessorien ja näytönohjaimen välillä tietoa ja laskutoimituksia cudaMemcpyAsync() API:n avulla. Eri tehtävien välillä tapahtuva tiedonsiirto on toteutettu monella eri protokollalla mukaan lukien gRPC over TCP ja RDMA(Remote Direct Memory Access) over Converged Ethernet. TensorFlow:n sisäistä arkkitehtuuria on kuvattu kuviossa 9.(Mts.)



Kuvio 9. TensorFlow:n arkkitehtuuri

4.3 TensorFlow'n muita koneoppimismalleja

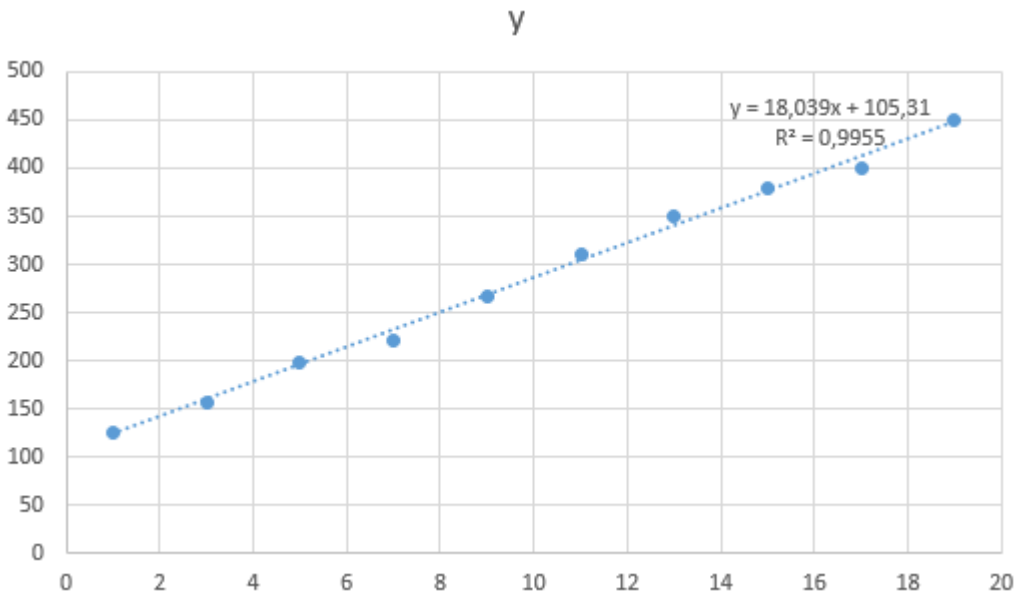
TensorFlow on monipuolinen koneoppimisohjelmisto, jonka kirjastoja laajennetaan jatkuvasti. Tämän takia sen tukemien koneoppimismallien määrä kasvaa koko ajan. Opinnäytetyön kirjoittamisen hetkellä se tukee teoriaosuudessa mainittujen neuroverkkomallien lisäksi esimerkiksi lineaarisia malleja ja tukivektorikoneita. Näiden yksittäisten mallien lisäksi se tukee joitain koneoppimismalleja, joissa yhdistetään muita yksittäisiä malleja. Esimerkkinä tällaisista malleista on syvä ja leveä verkkomalli, joka yhdistää lineaarisen mallin ja myötäkytkentä neuroverkkomallin.(Godbout. 2016.)

4.3.1 Lineaarinen malli

Lineaariset mallit kuvaavat vastausmuuttujaa yhden tai useamman ennemuuttujan funktiona. Niitä voidaan käyttää esimerkiksi taloudellisen datan analysoinnissa. Lineaarisen mallin luomisessa voidaan käyttää muutamia tilastotieteellistä menetelmiä, joista yksi on nimeltään lineaarinen regressio. Lineaarinen regressio on kaavan 9 mukainen. Se kuvaa riippuvan muuttujan y suhdetta yhteen tai useampaan riippumattomaan muuttujaan X_i. β_0 on vakiotermi, jolla merkataan mistä kohtaa y-akselilla kaavasta syntyvä suora poikkeaa origosta eli koordinaattiakselin nollakohdasta. β_i on riippumattoman muuttujan X_i regressiokerroin. Yhden riippumattoman muuttujan tapauksessa se suoraan kertoo kaavasta syntyvän suoran kaltevuuden eli toisin sanoen se on suoran kulmakerroin. ϵ_i on kaavassa virhetermi. Sillä kompensoidaan aineistosta syntyvien koordinaatistopisteiden mahdollisia virheitä.(Linear Model)

$$y = \beta_0 + \sum \beta_i X_i + \epsilon_i \tag{9}$$

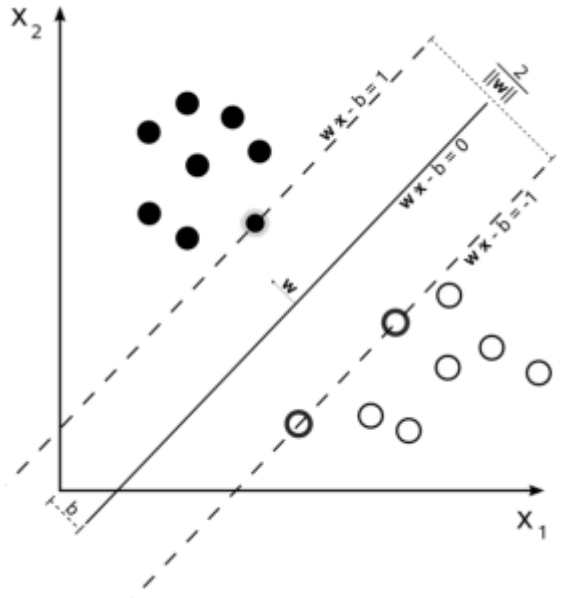
Kuviossa 10 on esitetty lineaarisen regression kuvaaja. Kuviossa näkyvät aineistopisteet on valittu mielivaltaisesti. Kuvaajan kaava tarkasteltaessa huomataan, että tässä on kyse yhden riippumattoman muuttujan tapauksesta. Voimme siis suoraan sanoa, että β_0 vastaa arvoa 105,31. β_i vastaa arvoa 18,039 ja ϵ_i (kuviossa R^2) on 0,9955.



Kuvio 10. Lineaarinen regression

4.3.2 Tukivektorikone

Tukivektorikonemallissa aineistosta syntyvät koordinaattipisteet pyritään jakamaan kahtia. Näiden kahden koordinaattipisteryhmien väliin pyritään sovittamaan sellainen taso, jonka molemmille puolille sijoitettujen samansuuntaisten marginaalitasojen välimatka olisi maksimaalinen. Samalla pyritään välttämään yhdenkään koordinaattipisteen jäämistä näiden tasojen väliin. Tukivektoreiksi kutsutaan niitä näytevektoreita, jotka rajoittavat näiden marginaalitasojen välimatkaa. Yksinkertaisimmassa tapauksessa aineistosta syntyvät koordinaattipistejoukot ovat lineaarisesti erottuvia. Tätä on esitetty kuviossa 11. Kuviossa olevat ulommat viivat ovat juuri mainitut marginaalitasot.(Tukivektoriluokittelija. N.d.. 1.)



Kuvio 11. Tukivektorikone lineaarisesti erottuvat luokat

Kuviossa 11 olevaa keskimmäistä viivaa kutsutaan päätöstasoksi ja, kuten aiemmin mainittiinkin, kahta ulommaista viivaa kutsutaan marginaalitasoiksi. Kaavalla 10 kuvataan ensimmäistä marginaalitasoa, kaavalla 11 kuvataan päätöstasoa ja kaavalla 12 kuvataan toista marginaalitasoa. Kaavoissa w^T tarkoittaa kyseisen tason normaalivektoria ja b on jokin vakio. Näiden w^T ja b arvot on valittava siten että marginaalitasojen välinen matka olisi mahdollisimman suuri.(Mts. 2.)

$$w^T x + b = 1 \quad (10)$$

$$w^T x + b = 0 \quad (11)$$

$$w^T x + b = -1 \quad (12)$$

Marginaalitasojen välimatkan maksimoiminen helpottuu, kun niiden kaavat yhdistetään kaavan 13 mukaiseksi. (Mts. 2.):

$$y_i(w^T x_i + b) \geq 1 \quad (13)$$

Tällöin tarvitsee vain minimoida funktio, jota kuvastaa kaava 14. Tämä tapa on helpoin, koska minimoitava funktio on neliöllinen ja se omaa vain yhden minimikohdan. Tämä mahdollistaa neliöllisen ongelmoinnin menetelmien käyttämisen kyseisen minimikohdan löytämiseen. (Mts. 2.)

$$\frac{1}{2} ||w||^2 \quad (14)$$

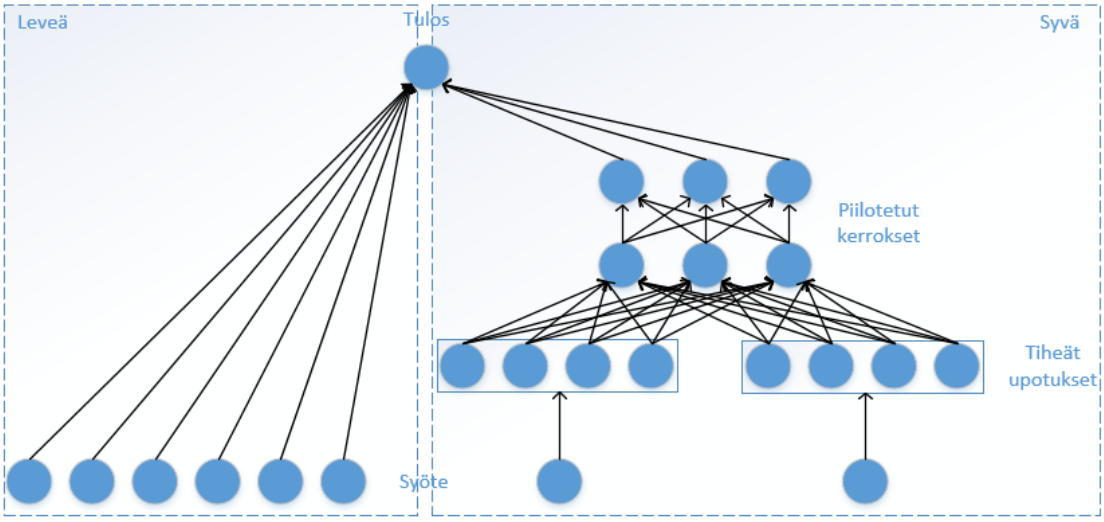
Joissain tapauksissa aineistosta ei synny lineaarisesti erottuvia luokkia. Jotta tällaiset tilanteet saataisiin tukivektorikoneella ratkaistua, on marginaalitasojen yleistä kaavaa hieman muutettava. Siihen on lisättävä niin sanotut slack-vakiot y_i kaavan 15 mukaisesti. Nämä vakiot mahdollistavat sen, että jotkin aineiston koordinaattipisteistä jäävät marginaalien sisälle. Tätä toteutustapaa kutsutaan joustavan marginaalin tavaksi.(Mts. 3.)

$$y_i(w^T x_i + b) \geq 1 - y_i \quad (15)$$

4.3.3 Leveä ja syvä verkkomalli

Leveä ja syvä verkkokoneoppimismalli yhdistää myötäkytkentä neuroverkon ja lineaarisen mallin. Kuviossa 12 näkyy kyseisen verkkomallin rakenne. Verkon leveä-komponentilla tarkoitetaan lineaarista mallia ja syvällä-komponentilla tarkoitetaan myötäkytkentä neuroverkkoa. Tätä mallia voidaan käyttää esimerkiksi asioiden suosittelu-

järjestelmissä, kuten Google Play kaupan sovellusten suosittelujärjestelmä.(Anderson, Anil, Aradhye, Chai, Chandra, Cheng, Corrado, Haque, Harmsen, Hong, Ispir, Jain, Koc, Liu, Shah, Shaked, 2016, 1.)



Kuvio 12. Leveä ja syvä verkkomalli

Google Playn suosittelujärjestelmässä lineaariseen malliin tulevat syötteet ovat esimerkiksi ”user_installed_app=facebook” arvolla 1, mikäli käyttäjä on asentanut Facebook sovelluksen. Kyseinen syöte on tietenkin arvolla 0, mikäli kyseistä sovellusta ei ole asennettu. Leveä-komponentti pystyy yhdistämään syötteitä ryhmiksi, mikäli sellainen ryhmä on ollut mukana koulutusdatassa. Tämän takia sen kyky tehdä yleisiä syötepareja on rajoittunut. Kyseisen rajoitteen takia verkossa on myös syvä-komponentti eli FNN. Se on paljon parempi yleisten syöteparien/-ryhmien luomisessa kuin lineaarinen malli, koska se pystyy yhdistämään syötteitä, vaikka sille ei ole koulutettu samanlaisia syöteryhmiä aiemmin.(Mts. 1)

Leveä ja Syvä verkkomalleja voidaan kouluttaa kahdella eri tavalla: joko koko verkkoa koulutetaan samaan aikaan tai molempia puoliskoja koulutetaan erikseen. Mikäli puoliskoja halutaan kouluttaa erikseen, on molempien mallien oltava riittävän suuri. Muuten verkon antaman ennustuksen tarkkuus ei todennäköisesti olisi riittävän korkea. Jos halutaan koko verkkoa kouluttaa samaan aikaan, ei molempien puoliskojen tarvitse olla hirveän suuri. Leveän-komponentin tarvitsee olla vain niin suuri, että se riittää kompensoimaan syvän-komponentin heikkouksia. Eli sille riittää, että se osaa tunnistaa pienen määrän syötepareja.(Mts. 2-3.)

5 Järjestelmän suunnittelu ja toteutus

5.1 Järjestelmän suunnittelu

Opinnäytetyön toteutus tullaan toteuttamaan konstruktiivista tutkimusmenetelmää käyttäen. Konstruktiivisessa tutkimusmenetelmässä ratkaistava ongelma on tiedossa, mutta tapa ratkaisuun pääsemiseen ei ole etukäteen tiedossa. Opinnäytetyössä ratkottavana ongelmana on kaatuneiden puiden havaitseminen ilmakuvista. Ongelman ratkaisemiseksi opinnäytetyössä luodaan järjestelmä, joka pystyy tämän tekemään.(Lukka. 2001.)

Järjestelmää varten tarvitaan vähintään yksi kuvasarja ennalta määrittelyltä alueelta ja kuvasarjassa pitää olla vähintään kaksi kuvaa. Valittu kuvasarja pitää sisällään juurikin kaksi kuvaa ja kuvat on otettu sellaiselta alueelta, jossa on jo valmiiksi kaatuneita puita. Tämä helpotti järjestelmän testausvaihetta ja kyseistä kuvasarjaa on mahdollista käyttää myös mahdollisissa opinnäytetyön jatkokehityksissä. Kuvasarjassa olevat kuvat on otettu eri vuodenaikaan. Kuvasarjan ensimmäinen kuva on otettu vuoden 2016 toukokuussa ja toinen kuva on otettu saman vuoden lokakuussa. Kuvasarjan kuvat on ladattu erikseen ”DroneDeploy” nimisestä palvelusta.

Tästä palvelusta peräisin olevat ilmakuvat on otettu käyttäen pieniä lennokkeja. Tämän palvelun kuvat päivittyvät nopeampaan tahtiin kuin Maanmittauslaitoksen karttapalvelun kuvat, minkä takia päädyttiin käyttämään kyseistä palvelua. Tämä mahdollistaa myrskytuhon varalta paremman reaktionopeuden, sillä ei tarvitse odottaa yhtä kauan uuden kuvan saamista. Maanmittauslaitoksenkin kuvia pystytään järjestelmässä käyttämään, mutta niitä varten pitää järjestelmän bash-skriptiä muokata hieman, sillä sieltä saatavat kuvat on jp2 tiedostomuotoisia ja niiden tiedostomuodon muuttamiseen tarvitaan erillinen ohjelma.

Palvelusta ladattavien kuvien tiedostomuodon voi valita kahdesta vaihtoehdosta: GeoTIFF (Georeferenced Tagged Image File Format) ja JPEG (Joint Photographic Experts Group). Molemmilla tiedostomuodoilla on omat etunsa. JPEG tiedostojen tiedostokoko on huomattavasti pienempi kuin vastaava GeoTIFF muotoinen kuva. GeoTIFF kuva taas voi pitää sisällään esimerkiksi koordinaattitietoja, joiden avulla

pystytään kuvassa esiintyvä alue sijoittaa kartalle. Tämän ominaisuuden takia päädyttiin käyttämään GeoTIFF tiedostomuotoisia kuvia. Koordinaattitietojen perusteella voimme olla varmoja kuvien muutosanalyysiä tehdessä, että vertaamme saman alueen kuvia keskenään.

Järjestelmä toimii kahden skriptin avulla. Ensimmäisen skriptin tehtävänä on muokata valitun kuvasarjan kuvat oikeanlaisiksi vertailua varten. Kyseinen skripti on bash-kielellä toteutettu skripti, koska kyseisellä kielellä oli helpointa nivottaa useampaa erillistä ohjelmaa yhteen. Skriptillä käytetään halutut kuvat kahden eri ohjelmiston läpi: GDAL (Geospatial Data Abstraction Library) ja ImageMagick. GDAL ohjelmistolla leikattiin jokaisesta kuvasarjan kuvasta koordinaattien perusteella sama alue, jonka jälkeen niistä luotiin omat kuvansa. Kuvien koordinaatistot eivät ole samat kuin mitä Suomen alueen karttakoordinaatistona on käytössä. Kuvien koordinaatisto on EPSG:4326 ja Suomen alueen karttakoordinaatisto on EPSG:3067. Normaalisti näiden koordinaatistojen välillä tarvitsisi tehdä muutoksia, jotta kuvista saisimme rajattua oikean alueen irti. Onneksi kuvasarjan toiseen kuvaan on jo alue rajattu valmiiksi, joten siitä saimme valmiiksi EPSG:4326 koordinaatiston mukaiset koordinaatit. ImageMagick ohjelmistolla hoidetaan loput kuvankäsittelytoimenpiteet. Näihin toimenpiteisiin kuuluu kuvien tiedostomuodon muuttaminen PNG (Portable Network Graphics) tiedostomuotoon, kuvista mahdollisen läpinäkyvyyden poistaminen ja kuvien jakaminen 8 pienempään osaan. Viimeisenä toimenpiteenä se käynnistää järjestelmän toisen skriptin.

Järjestelmän toinen skripti on kirjoitettu pythonilla ja se on vastuussa kuvan muutosanalyysissä. Sen tarkoituksena on ensin verrata kuvasarjan kuvien vastaavia osakuvia keskenään ja yrittää havaita niistä muutoskohtia. Tämä muutoskohta sitten syötetään Tensorflow'lla toteutettuun neuroverkkoon luokittelua varten. Neuroverkon valinnassa on monia vaihtoehtoja, voidaan joko valita yksi monesta TensorFlow'n tarjoamasta valmiiksi rakennetuista ja koulutetuista neuroverkoista tai voidaan myös luoda uusi. Taulukossa 3 on esitetty osa TensorFlow'n tarjoamista neuroverkoista. Jokaiselle taulukossa esiintyvälle mallille on merkattu niiden Top-1 ja Top-5 tarkkuudet. Top-1 tarkkuudella tarkoitetaan sitä, että neuroverkko antaa vain yhden luokitteluvaihtoehdon kuvalle ja Top-5 tarkoitetaan sitä, että neuroverkko antaa viisi eri luokit-

teluvaihtoehtoa kuvalle. Esimerkiksi Inception V4 on oikeassa 80,2 % ajasta antaessaan vain yhden vaihtoehdon ja 95,2 % ajasta antaessaan viisi eri vaihtoehtoa.(TensorFlow-Slim image classification library. N.d.)

Taulukko 3. TensorFlow'n valmiit mallit.

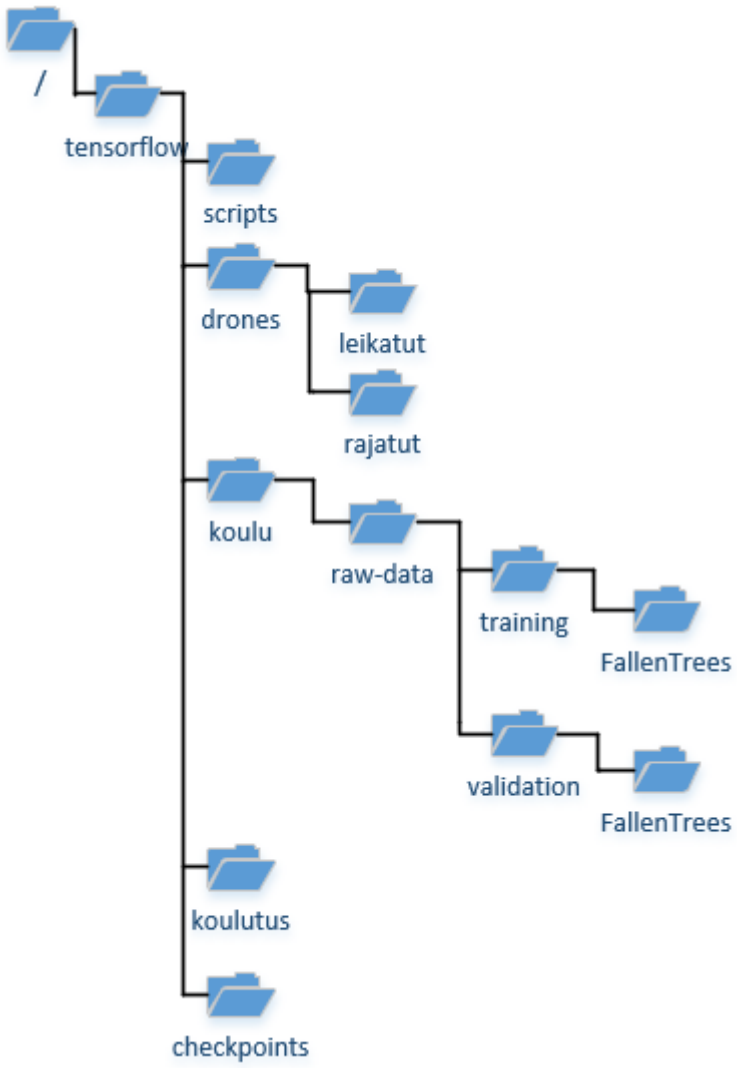
Mallin nimi	Top-1 tarkkuusprosentti	Top-5 tarkkuusprosentti
Inception V4	80.2	95.2
Inception-ResNet-v2	80.4	95.3
ResNet V1 152	76,8	93,2
ResNet V2 152	77,8	94,1
VGG 16	71,5	89,8
VGG 19	71,1	89,8

Oma neuroverkko voisi olla paras vaihtoehto, mikäli olisi paljon aikaa käytettävissä sekä sen luomiseen, kouluttamiseen, toiminnan analysointiin että muutoksien tekemiseen analysoinnin pohjalta. Tämän ajan tarpeen takia päädyttiin käyttämään jo valmiiksi rakennettua ja koulutettua neuroverkkoa. Inception V4 on Googlen työntekijöiden luoma ja kouluttama neuroverkko ja sillä on toiseksi korkein tarkkuusprosentti. Sen ero tarkimman neuroverkon kanssa on tosi pieni, joten sen tarkkuus on riittävä opinnäytetyön tarpeisiin. Valittu neuroverkko piti kuitenkin kouluttaa tunnistamaan kaatuneet puut, joten tuota tarkkuuslukemaa ei voi pitää absoluuttisena totuutena. Näiden syiden perusteella päädyttiin valitsemaan Inception V4 neuroverkko.

5.2 Alkuvalmistelut

Aluksi on luotava jokin kansiorakenne, johon sijoitetaan kaikki toteutukselle tarpeellinen materiaali. On hyvä jakaa materiaali eri kansioihin, koska toteutuksessa syntyy monta eri kuvaa ja se vaatii montaa eri TensorFlow moduulia. Opinnäytetyössä käytetään seuraavaa kansiorakennetta. Toteutusta varten luodaan juureen tensorflow niminen kansio, jonka sisälle kaikki materiaali sijoitetaan omiin kansioihinsa. Molemmille skripteille luodaan yhteinen kansio nimeltä ”scripts”. Ilmakuvia varten luodaan ”drones” niminen kansio, jonka sisään luodaan kansiot ”leikatut” ja ”rajatut”. Kouluttamista varten luodaan kaksi kansiota ”koulu” ja ”koulutus”. ”Koulu” kansio sisältää

neuroverkon kouluttamiseen käytettävät kuvat ja sen sisälle luodaan ”raw-data” niminen kansio. ”Raw-data” kansion sisälle luodaan kaksi eri kansiota ”training” ja ”validation”, molempien kansioiden sisälle sijoitetaan yksittäinen kansio ”FallenTrees”. Tällainen koulutusdatakansiorakenne tekee kouluttamisesta yksinkertaisempaa, sillä ”FallenTrees” kansio kuvaa sitä nimilappua, jonka haluamme neuroverkon yhdistävän koulutuskuviimme. ”Koulutus” kansion sisälle ei sijoiteta muita kansioita, sillä sinne tulee sijoittumaan kaikki koulutuksen aikana syntyvät tiedostot. Viimeisenä luodaan ”checkpoints” kansio, jonne sijoitetaan neuroverkkomme tallennuspiste. Kuviossa 13 on pyritty selventämään toteutuksen kansiorakennetta.



Kuvio 13. Toteutuksen kansiorakenne

Seuraavaksi asennetaan kaikki tarvittavat ohjelmat ja niiden vaatimat muut kirjastot. Näitä ohjelmistoja oli GDAL, ImageMagick ja TensorFlow. GDAL ohjelmiston asentamista varten tarvitsee lisätä uusi tietolähde käyttöjärjestelmälle. On myös päivitettävä Ubuntun pakettilistaukset, jotta voimme asentaa GDAL'n. Nämä asiat hoituvat seuraavilla komennoilla:

```
sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get install gdal-bin
```

Seuraavaksi on asennettava ImageMagick ohjelmisto. Tämän ohjelmiston asentaminen vaatii vain apt-get komennon, samalla tavalla kuin GDAL.

```
sudo apt-get install imagemagick
```

Viimeisenä asennetaan TensorFlow. TensorFlow'n asentamiseen on muutama eri tapa, mutta tässä opinnäytetyössä se asennetaan virtuaaliympäristö asennustavan mukaisesti. Toteutuksessa käytettään Pythonin 2.7 versiota, joten TensorFlow'n asennuksessa on otettu se huomioon. Seuraavilla komennoilla TensorFlow'n asentaminen tapahtuu. "Pip install" komennoista vain toista käytetään ja se riippuu siitä, onko laitteella käytössä näytönohjainta. Jos laitteella on näytönohjain ja sen resursseja halutaan antaa TensorFlow'n käyttöön, käytetään "tensorflow-gpu" komentoa.

```
sudo apt-get install python-pip python-dev python-virtualenv
virtualenv –system-site-packages /home/tensor/tensorflow
source /home/tensor/tensorflow/activate
pip install –upgrade tensorflow
pip install –upgrade tensorflow-gpu
```

Mikäli halutaan käyttää järjestelmässä Maanmittauslaitoksen karttapalvelun ilmakuvia, on virtuaalikoneelle vielä asennettava yksi ohjelma OpenJPEG. Tällä ohjelmalla voidaan muuttaa aiemmin mainittujen jp2 tiedostomuotoisten kuvien tiedostomuotoa. Sen voi asentaa seuraavalla komennolla.

```
sudo apt-get install openjpeg-tools
```

5.3 Kuvien muokkaaminen

Kuvien muokkaamista varten oleva skripti vaatii käyttäjältä kahta syötettä käynnistettäessä. Se haluaa tiedon kahden vertailuun haluttavan kuvan tiedostosijainnit, ensimmäisenä annetaan uudemman kuvan tiedostosijainti ja toisena vanhemman kuvan tiedostosijainti. Kuviossa 14 on esitetty komento, millä käynnistetään tämä skripti. Kuviosta huomataan uudemman kuvan tiedostosijainniksi ”/tensorflow/drones/Konnevesikuvio22_Orthomosaic_export_ThuMay11.tif” ja vanhemman kuvan tiedostosijainniksi ”/tensorflow/drones/Konnevesi52016_Orthomosaic_export_ThuMay11.tif”

```
tensor@Tensor:/tensorflow$ scripts/convertkuvat.sh /tensorflow/drones/Konnevesik
uvio22_Orthomosaic_export_ThuMay11.tif /tensorflow/drones/Konnevesi52016_Orthomo
saic_export_ThuMay11.tif
```

Kuvio 14. Kuvan muokkaamisen käynnistäminen

Kuviossa 15 on esitetty kuvien muokkaamista varten olevan skriptin alkua. Ensin tulee muutama kommenttikenttä, joilla pyritään selventämään skriptin etenemistä. Skriptissä ensimmäisenä ilmoitetaan kaksi muuttujaa ”uusi” ja ”vanha”, mihin sijoitetaan skriptin käynnistyksen yhteydessä annetut tiedostosijainnit. Tiedostosijaintien tallentamisen jälkeen molempia muuttujia käytetään kahdessa seuraavassa komennoissa. Kyseisillä komennoilla rajataan GeoTIFF kuvista koordinaattien perusteella ennalta määritelty alue. –T_srs lipun jälkeen määritellään kuvissa käytössä oleva koordinaatisto. –Te lipun jälkeen määritetään kuvan koordinaatiston mukaiset halutun alueen koordinaatit. Ensimmäinen arvo on x_min, toinen arvo on y_min, kolmas arvo on x_max ja viimeinen arvo on y_max. Koordinaattien jälkeen komennossa ilmoitetaan muuttujalla, mistä kuvasta haluttu alue rajataan. Komennon lopussa kerrotaan rajaamisesta syntyvän kuvan tallennuspaikka ja -nimi.

```
#!/bin/bash
#ilmakuvien muokkaaminen järjestelmälle sopiviksi, jonka jälkeen ne syötetään järjestelmään.

#Laitetaan talteen skriptin käynnistyksessä annetut tiedostosijainnit
uusi=$1
vanha=$2

#Molemmista kuvista leikataan alue koordinaattien perusteella
gdalwarp -t_srs EPSG:4326 -te 26.3400650 62.6694936 26.3501071 62.6745768 $uusi /tensorflow/drones/rajatut/LeikattuUusi.tif
gdalwarp -t_srs EPSG:4326 -te 26.3400650 62.6694936 26.3501071 62.6745768 $vanha /tensorflow/drones/rajatut/LeikattuVanha.tif
```

Kuvio 15. Tiedostosijaintien vastaanotto ja koordinaattirajaus

Seuraavaksi skripti muuttaa kuvan tiedostomuodon PNG muotoiseksi kahdesta syystä. Tästä eteenpäin järjestelmällä ei ole enää tarvetta koordinaattitiedoille ja TensorFlow ei tuo GeoTIFF tiedostomuotoa. Kuten on aiemmin mainittu, tiedostomuodon muuttaminen toteutetaan ImageMagick ohjelmistolla. Tiedostomuodon muuttaminen tapahtuu yksinkertaisella komennolla: ”convert mistä mihin”. Kuviossa 16 on esitetty tiedostomuodon muuttamiseen käytetty komento. Kuviossa olevat kaksi ensimmäistä komentoa muokkaavat juuri rajattujen kuvien tiedostomuodot oikeaksi. Seuraavat kaksi komentoa poistavat molemmista kuvista läpinäkyvät osat. Komennoissa ”-alpha off” lipulla hoidetaan läpinäkyvät poistaminen. Tämä ei ole tarpeellista, mutta toisessa kuvasarjan kuvassa läpinäkyvyyttä löytyy ja sen poistaminen tarkoittaa sitä, että sen tilalle tulee mustaa aluetta. Tätä me hyödynnämme muutosanalyysi skriptissä.

```
#Muutetaan molempien kuvien tiedostomuodot
convert /tensorflow/drones/rajatut/LeikattuUusi.tif /tensorflow/drones/rajatut/LeikattuUusi.png
convert /tensorflow/drones/rajatut/LeikattuVanha.tif /tensorflow/drones/rajatut/LeikattuVanha.png

#Poistetaan molemmista kuvista mahdolliset näkymättömällä värillä merkatut alueet
convert /tensorflow/drones/rajatut/LeikattuUusi.png -alpha off /tensorflow/drones/rajatut/LeikattuAlueUusi.png
convert /tensorflow/drones/rajatut/LeikattuVanha.png -alpha off /tensorflow/drones/rajatut/LeikattuAlueVanha.png
```

Kuvio 16. Tiedostomuodon muuttaminen

Seuraavana on vuorossa molempien kuvien paloitteleminen pienempiin osiin. Kuviossa 17 on esitetty kuvien paloittelu komento ja myös sekä TensorFlow’n että muutosanalyysi skriptin käynnistäminen. Kuvien paloittelukomennossa ”-crop” lipun jälkeen määritetään miten laajasti paloitellaan kuva. Paloittelun koko voidaan määrittää pikseleissä tai prosenteissa. Toteutuksissa päädyttiin käyttämään prosentteja, koska haluttiin saada jaettua kuva tasaisesti ja ei ole tarvetta jakaa kuvaa tosi moneen osaan. Nyt komento jakaa kuvat leveyssuunnassa neljään osaan ja korkeussuunnassa kahteen osaan eli kuvista syntyy yhteensä kahdeksan osaa. Lopuksi skripti

käynnistää TensorFlow'n virtuaaliympäristön ja käynnistää muutoksen analysoinnin. Läpikäyty skripti on kokonaisuudessaan liitetty liitteeksi 1.

```
#Leikataan molemmat kuvat pienempiin osiin
convert -crop 25%x50% /tensorflow/drones/rajatut/LeikattuAlueUusi.png /tensorflow/drones/leikatut/LeikattuUusi-jaoteltu.png
convert -crop 25%x50% /tensorflow/drones/rajatut/LeikattuAlueVanha.png /tensorflow/drones/leikatut/LeikattuVanha-jaoteltu.png

#Varmistetaan että slim moduuli on asennettu, käynnistetään tensorflow ja viimeisenä ajetaan muutoksen tarkastelu python skripti
source /home/tensor/tensorflow/bin/activate
python /tensorflow/scripts/analyysi.py
```

Kuvio 17. Kuvien jakaminen pienempiin osiin

5.4 Muutosanalyysi ja muutoksen luokittelu

Muutosanalyysi skripti, kuten aiemmin mainittiin, on kirjoitettu Python'lla ja järjestelmän käytössä oleva Python versio on 2.7. Python skriptin alussa tuodaan tarvittavia kirjastoja ja TensorFlow'n moduuleita. Nämä on esitetty kuviossa 18. Ennen ensimmäistä kirjastoa on kirjoitettava kommentiksi ”coding=utf-8”, koska ilman sitä voi ilmentyä erikoisia virheitä. Ensimmäisenä kirjastona tuodaan NumPy niminen kirjasto, joka tuo lisää mahdollisia laskutoimituksia. Kirjasto ”os” tuo Pythonille käyttöön tiettyjä käyttöjärjestelmäkohtaisia komentoja esimerkiksi tiedostojen kirjoittaminen. Muutosanalyysi skriptille on myös tuotava TensorFlow, jotta voimme käyttää neuroverkkoja muutoksen luokitteluun. Matplotlib kirjasto mahdollistaa kuvioiden luomisen datasta ja tässä toteutuksessa se ei tule avaamaan ikkunoita luomistaan kuvioista, sillä se vaatisi virtuaalikoneelta X-windows tukea. Itsessään se ei olisi ongelma, mutta skriptiä tullaan ajamaan palvelimella olevalla virtuaalikonetta ja sille ei haluttu sitä asentaa. Mikäli on mahdollisuus ja halutaan ikkunoiden avautuminen, on skriptistä vain poistettava tuo ”matplotlib.use” rivi. Import rivien jälkeen python skriptiin tuodaan PIL kirjastosta kaksi komentoa from lauseilla. Viimeisillä neljällä rivillä tuodaan TensorFlow'n slim moduulista neuroverkon toimimiselle oleellisia toimintoja.

```
#coding=utf-8

import numpy as np
import os
import tensorflow as tf
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

from PIL import Image
from PIL import ImageChops
from datasets import dataset_utils
from datasets import imagenet
from nets import inception
from preprocessing import inception_preprocessing
```

Kuvio 18. Python skriptiin tuodut kirjastot

Kuviossa 19 on esitetty python skriptin kuvien vertailuosuus. Kuvion ylin rivi pakottaa käymään tämän skriptin läpi kahdeksan kertaa, jotta se voisi käydä jokaisen kuvan osan kerrallaan läpi ajamatta sitä useampaan otteeseen tai tekemättä skriptistä älyttömän pitkää. ”Image.open” rivit avaavat tutkittavana olevat kuvan osat(ensimmäisellä läpikulkukerralla kuvan osa 0, toisena 1, kolmantena 2, jne.). Diff muuttujaan tallennetaan havaitun muuttuneen kohdan sijainti. Tämä tulostetaan näytölle seuraavalla print komennolla.

```
for x in range(8):
    print(x)
    im1 = Image.open("/tensorflow/drones/leikatut/LeikattuUusi-jaoteltu-{}.png" .format(x))
    im2 = Image.open("/tensorflow/drones/leikatut/LeikattuVanha-jaoteltu-{}.png" .format(x))
    diff = ImageChops.difference(im2, im1).getbbox()
    print "Kuvion osassa on ",diff," kohdassa muutos"
```

Kuvio 19. Kuvan osien vertailu

Kun muuttunut kohta on tiedossa, on se seuraavaksi irrotettava uusimmasta kuvasta ja siitä on sitten luotava oma kuvansa. Tämän asian hoitavat komennot on esitetty kuviossa 20. Tässäkin muutoskuvan nimen numerointi on juokseva, eli joka läpikulkukerralla se saa nimeensä seuraavan numeron. Tämä muutoskohtakuvan tiedostosijainti ja –nimi tallennetaan muuttujaan myöhempää käyttöä varten. Kuvion toinen if-rivi tarkistaa onko muutoskuvassa pelkkää mustaa. Jos kuvio havaitaan olevan täynnä mustaa(esimerkiksi johtuen aikaisemman skriptin poistaessa läpinäkyvyyden), muutoskohtaa ei siirretä neuroverkolle tutkittavaksi.

```
if diff != None:
    muutosnimi = ("/tensorflow/drones/muutos-{}.png".format(x))
    muutos = iml.crop((diff))
    muutos.save(muutosnimi,"PNG")
    if muutos.getbbox() == None:
        print ('Kuvan osa {} sisaltaa pelkkaa mustaa, sitä on turha syottaa neuroverkkoon' .format(x))
```

Kuvio 20. Muuttuneen kohdan tallennus

Else lauseen jälkeen ruvetaan valmistelemaan neuroverkkomme. Nämä näkyvät kuviossa 21. Ensin määritettään käyttöön TensorFlow'n slim moduuli, joka sisältää monia työkaluja esimerkiksi eri laajojen neuroverkkojen kouluttamiseen. Seuraavana määritetään neuroverkkoon syötettävien kuvien koko. Kuvan kooksi määritetään Inception V4 neuroverkon vakio kuvakoko, joka on 299 x 299. Sitten on määritettään neuroverkon tallennuspistetiedoston sijainti jonka avulla tallennetaan tiedot neuroverkon parametrien tämän hetkiset arvot. Seuraavaksi aloitetaan TensorFlow'n kaavion piirtäminen. Sen alle luodaan aluksi tyhjä muuttuja "tarim", jonka sisään tallennetaan muutoskuvan sisältö. Sen jälkeen muuttujan sisältämä PNG kuvadata puretaan tensoriksi. Sen jälkeen siitä prosessoidaan neuroverkolle haluttuun muotoon.

```
else:
    slim = tf.contrib.slim
    image_size = inception.inception_v4.default_image_size
    checkpoints_dir = '/tensorflow/checkpoints'
    with tf.Graph().as_default():
        tarim = ''
        with open (muutosnimi) as f:
            tarim = f.read()
        tarka = tf.image.decode_png(tarim, channels=3)
        prosessointi = inception_preprocessing.preprocess_image(tarka, image_size, image_size, is_training=False)
        prosessoitu = tf.expand_dims(prosessointi, 0)
```

Kuvio 21. Neuroverkon valmistelut

Kuviossa 22 Python skriptille kerrotaan kaikki tarvittava Inception V4 neuroverkosta. Sille kerrotaan mitä eri luokitteluja sillä on datalle ja miten se vertaa meidän prosessoitua muutoskuvaa niihin. Samalla sille kerrotaan mistä se katsoo todennäköisyydet jokaiselle muutoskuvalle. Kun sillä on luokat tiedossa, on sille kerrottava missä neuroverkon parametrit ovat tallessa. Nämä tiedot tallennetaan "init_fn" nimiseen muuttujaan. Kun kaikki tiedot on kerrottu skriptiin, on aika suorittaa TensorFlow koodi. TensorFlow'n suorittaminen tapahtuu istunnon sisällä ja koodissa se merkitään "with tf.Session" lausekkeella. Sen alle määritetään parametri muuttuja, joka ajetaan istunnon sisällä. Myös tuloksien todennäköisyydet tarkastetaan istunnon sisällä ja ne tallennetaan listaksi.

```
with slim.arg_scope(inception.inception_v4_arg_scope()):
        logits, _ = inception.inception_v4(prosessoitu, num_classes=1001, is_training=False)
todennakoisyys = tf.nn.softmax(logits)

init_fn = slim.assign_from_checkpoint_fn(
        os.path.join(checkpoints_dir, 'inception_v4.ckpt'),
        slim.get_model_variables('InceptionV4'))

with tf.Session() as sess:
        init_fn(sess)
        np_image, todennakoisyys = sess.run([tarka, todennakoisyys])
        todennakoisyys = todennakoisyys[0, 0:]
        sorted_inds = [i[0] for i in sorted(enumerate(-todennakoisyys), key=lambda x:x[1])]
```

Kuvio 22. Neuroverkon luonti ja session käynnistäminen

Kuviossa 23 on esitetty miten luotu Python skripti loppuu. Siinä on muutama Matplotlib kirjaston komento kuvaajien piirtämiseen. ”Plt.imshow” komento avaisi tutkittavan muutoskuvan uutena ikkunana, mikäli järjestelmässä olisi käytössä ”X-windows” ja jokin työpöytä tai terminaali ohjelmassa on ”X11 forwarding” laitettu päälle. "Plt.axis(’off’) komento poistaa syntyvistä kuvioista reunukset. ”Nimi” muuttujan avulla muutetaan neuroverkon luokat helpommin ymmärrettäviksi. Sen jälkeen ”for” loopin avulla tulostetaan näytölle neuroverkon viisi ehdotusta kuvan luokittelulle ja niiden todennäköisyydet. Skriptin alin ”else” lausekkeen avulla voidaan jättää väliin jokin skriptin läpikulku, jos verrattavat kuvat olivat samoja. Koko muutosanalyysi skripti on lisätty opinnäytetyöhön liitteenä 2.

```
            plt.figure()
            plt.imshow(np_image.astype(np.uint8))
            plt.axis('off')
            plt.show()
            print ("Kuvan osan {} sisältämässä muutoskohdassa on järjestelmän mielestä:".format(x))
            nimi = imagenet.create_readable_names_for_imagenet_labels()
            for i in range (5):
                index = sorted_inds[i]
                print("Todennakoisyys %0.2f%% => [%s]" % (todennakoisyys[index], nimi[index]))

else:
    print ('Käsittelyssä olleeseen kuvan osaan {0} ei ole tapahtunut muutosta' .fomrat(i))
```

Kuvio 23. Muutoskohtien luokittelu

5.5 Neuroverkon koulutus

Valitsemaamme neuroverkkoa voidaan kouluttaa kahdella eri tavalla: joko kouluttaa kokonaan alusta asti itse tai kouluttaa ainoastaan viimeisen kerroksen arvoja eli tehdä neuroverkon parametrien arvoille pientä hienosäätöä. Jos neuroverkon koulut-

taisi kokonaan uudestaan, voisimme yhdistää oman koulutusdatan imagenet nimiseen valmiiseen datasettiin ja saada ehkä tarkoitukseen toimiva järjestelmä. Tämä tapa vie tosi paljon aikaa, se nimittäin voisi viedä useampia päiviä tai jopa enemmänkin riippuen kuinka tehokas tietokone on järjestelmää pyörittämässä. Koska tätä järjestelmää testataan omalla laitteella, joka ei ole hirveän tehokas ja kiintolevytila on rajallinen, tätä toteutustapaa ei voi käyttää. Joten ainoastaan voimme kokeilla hienosäädöllä saada järjestelmää tunnistamaan kaatuneet puut.

Neuroverkon koulutus on aloitettava luomalla koulutus ”dataset”. Tämä koulutusdata koostuu kuvista. Koska järjestelmän tarkoituksena oli pystyä tunnistamaan kaatuneita puita, on näiden koulutusdatan kuvien oltava kaatuneista puista. Mieluiten näiden pitäisi olla suurin piirtein samanlaisesta perspektiivistä otettuja, kuin oikea data, eli ilmasta käsin otettuja kuvia. Helpoiten tällaisen kuvakokoelman saa kasaan Googlen kuvahaun avulla ja ensin ottaa sieltä muutaman suuremman kuvan. Näistä suuremmista kuvista sitten leikkaa pienempiä alueita, joissa kaatuneita puita esiintyy. Kuvia pitäisi olla tosi monta, varsinkin jos halutetaan kunnolla kouluttaa laajaa neuroverkkoa. Päädyttiin kuitenkin kokoamaan vain 35 kuvan sarja, joista viisi sijoitetaan arviointi kuviksi ”validation/FallenTrees” kansioon ja loput sijoitetaan koulutuskuviksi ”training/FallenTrees” kansioon. Samalla on luotava ”labels.txt” tiedosto, jonne kirjoitetaan haluttu luokan nimi eli tässä tapauksessa ”FallenTrees”.

Koulutus ”datasetin” luominen tapahtuu komennolla, joka on esitetty kuvion 24 yläreunassa. Komento on selkeyden takia jaettu useammalle riville. Komento vaatii erilaisilla lipuilla tiedon, missä mikäkin data on. ”-Traini_directory” lipulla ilmoitetaan missä koulutusdata on, mainitsematta luokan nimellä olevaa kansiota. ”-validation_dir” lipulla ilmoitetaan missä tiedostosijainnissa arviointi dataset on. ”-Output_directory” lipulla ilmoitetaan, mihin halutaan kaikki komennosta syntyvä data sijoitettavan. ”-Label-file” lipulla kerrotaan missä luomamme luokittelu tiedosto sijaitsee. Kaikki sen jälkeen näkyvät rivit ovat tulostetta, jotka kertovat datasetin luomisen edistymisestä.

```
(tensorflow) tensor@Tensor:/tensorflow$ python models/inception/inception/data/build_image_data.py \
> --train_directory="/tensorflow/koulu/raw-data/training" \
> --validation_directory="/tensorflow/koulu/raw-data/validation" \
> --output_directory="/tensorflow/koulu" \
> --labels_file="/tensorflow/koulu/raw-data/labels.txt"
Saving results to /tensorflow/koulu
Determining list of input files and labels from /tensorflow/koulu/raw-data/validation.
Found 5 JPEG files across 1 labels inside /tensorflow/koulu/raw-data/validation.
Launching 2 threads for spacings: [[0, 2], [2, 5]]
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE3
se are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4
hese are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4
hese are available on your machine and could speed up CPU computations.
2017-05-16 11:51:19.709035 [thread 0]: Wrote 2 images to /tensorflow/koulu/validation-00000-of-00002
2017-05-16 11:51:19.709164 [thread 0]: Wrote 2 images to 2 shards.
2017-05-16 11:51:19.714690 [thread 1]: Wrote 3 images to /tensorflow/koulu/validation-00001-of-00002
2017-05-16 11:51:19.714808 [thread 1]: Wrote 3 images to 3 shards.
2017-05-16 11:51:20.702826: Finished writing all 5 images in data set.
Determining list of input files and labels from /tensorflow/koulu/raw-data/training.
Found 30 JPEG files across 1 labels inside /tensorflow/koulu/raw-data/training.
Launching 2 threads for spacings: [[0, 15], [15, 30]]
2017-05-16 11:51:20.905391 [thread 0]: Wrote 15 images to /tensorflow/koulu/train-00000-of-00002
2017-05-16 11:51:20.905601 [thread 0]: Wrote 15 images to 15 shards.
2017-05-16 11:51:20.910164 [thread 1]: Wrote 15 images to /tensorflow/koulu/train-00001-of-00002
2017-05-16 11:51:20.910316 [thread 1]: Wrote 15 images to 15 shards.
2017-05-16 11:51:21.726729: Finished writing all 30 images in data set.
```

Kuvio 24. Koulutusdatan luominen

Koulutus ”datasetin” luomiskomennon jälkeen ”-output_directory” lipulla ilmoitetun kansion sisältö näyttää kuvion 25 mukaiselta.

```
tensor@Tensor:~$ ls /tensorflow/koulu
labels.txt  train-00000-of-00002  validation-00000-of-00002
raw-data    train-00001-of-00002  validation-00001-of-00002
```

Kuvio 25. Koulutusdatakomennon tulos

Tämän jälkeen voi alkaa kouluttamaan järjestelmää. Hienosäätökouluttaminen tapahtuu komennolla joka on esitetty kuviossa 26. Se vaatii muutaman lipun enemmän kuin koulutusdatasetin luominen. ”-Model_name” lipulla ilmoitetaan mitä mallia käytetään. ”-Max_number_of_steps” lipullla määritetään kuinka monta koulutusaskelta komento käy läpi(komennon läpikäyminen tällä määrällä kesti noin kaksi tuntia). ”-Checkpoint_exclude_scopes” lipulla määritetään mitä parametrien arvoja ei ladata tallennuspisteestä. ”-Trainable_scopes” lipulla kerrotaan minkä parametrien arvoja halutaan kouluttaa.

```
(tensorflow) tensor@Tensor:~$ python /tensorflow/models/slim/train_image_classifier.py \
> --train_dir="/tensorflow/koulutus" \
> --dataset_dir="/tensorflow/koulu" \
> --dataset_split_name="train" \
> --model_name="inception_v4" \
> --max_number_of_steps=100 \
> --checkpoint_path="/tensorflow/checkpoints/inception_v4.ckpt" \
> --checkpoint_exclude_scopes=InceptionV4/Logits,InceptionV4/AuxLogits \
> --trainable_scopes=Inceptionv4/Logits,InceptionV4/AuxLogits
```

Kuvio 26. Hienosäätökouluttamisen komento

Kuviossa 27 on esitetty koulutuskomennosta syntyvää syötettä. Komento antaa jokaisen 10 koulutusaskeleen jälkeen ilmoituksen häviöfunktion arvosta. Mitä enemmän koulutusdataa on käytettävissä, sitä useampi koulutusaskel on annettava komennossa. Kun komento on käynyt sille määritellyn määrän koulutusaskelia läpi, se tallentaa mallin muutokset levylle.

```
INFO:tensorflow:Starting Session.
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global_step/sec: 0.0145009
INFO:tensorflow:global step 10: loss = 7.8028 (132.13 sec/step)
INFO:tensorflow:global_step/sec: 0.0145425
INFO:tensorflow:global step 20: loss = 7.8397 (60.90 sec/step)
INFO:tensorflow:global_step/sec: 0.0142032
INFO:tensorflow:global step 30: loss = 7.8032 (60.37 sec/step)
INFO:tensorflow:global_step/sec: 0.0144298
INFO:tensorflow:global step 40: loss = 7.7828 (60.88 sec/step)
INFO:tensorflow:global_step/sec: 0.0144327
INFO:tensorflow:global step 50: loss = 7.7892 (60.14 sec/step)
INFO:tensorflow:global_step/sec: 0.0143013
INFO:tensorflow:global step 60: loss = 7.7487 (60.39 sec/step)
INFO:tensorflow:global_step/sec: 0.0145132
INFO:tensorflow:global_step/sec: 0.0136741
INFO:tensorflow:global step 70: loss = 7.7987 (135.68 sec/step)
INFO:tensorflow:global_step/sec: 0.0142796
INFO:tensorflow:global step 80: loss = 7.7233 (61.41 sec/step)
INFO:tensorflow:global step 90: loss = 7.8259 (60.62 sec/step)
INFO:tensorflow:global step 100: loss = 7.7831 (60.66 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
```

Kuvio 27. Koulutuskomennon antama tuloste

6 Järjestelmän testaus

Järjestelmän testaaminen aloitettiin testaamalla kuvan muokkaamista varten olevaa skriptiä vaiheittain. Aloitettiin koordinaattien rajaamisen testauksella. Koska tiedämme että uudemmassa kuvassa on jo valmiiksi rajattu haluttu alue, voimme verrata vanhemmasta kuvasta tehtyä rajattua kuvaa suoraan uuteen. Tämä vertailu on esitetty kuviossa 28. Kuviossa vasemman puolen kuva on vanhempi kuva ja oikean

puoleinen kuva on uudempi. Kuten kuviosta voidaan todeta, ovat ne samasta alueesta. Tästä myös huomaa, että kuvien valoisuudet ovat hyvin erilaiset. Tämä todennäköisesti johtaa siihen, että järjestelmä kategorioi nämä kuvat lähes kokonaan erilaisiksi. Vanhemman alueen kuvassa olevat metsäalueet näyttävät olevan laajempia kuin uudemmassa kuvassa.

Kuvio 28. Rajattujen kuvien vertailu

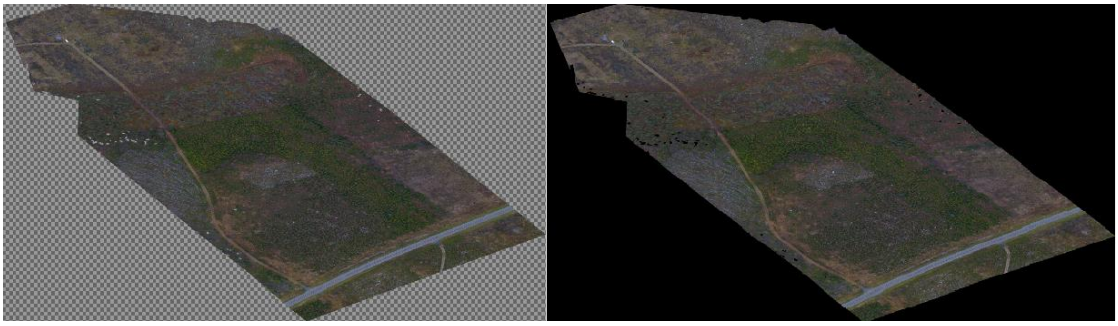
Seuraavaksi piti testata, että kuvien tiedostomuotojen muuttaminen toimi eli käyttöjärjestelmä tunnistaa ne oikeiksi PNG kuviksi. Tiedostomuodon oikeellisuuden testaamiseen on monia eri tapoja, mutta testauksessa haluttiin välttää uusien ohjelmien asennuksia ja mahdollisesti käyttää ubuntu käyttöjärjestelmään kuuluvia ohjelmia tai komentoja. Tämän takia päädyttiin kuviossa 29 näkyvää komentoa. Kuviossa näkyvällä ”file” komennolla pystytään tarkistamaan erilaisten tiedostojen tiedostomuodot. Komento kertoo tiedostosta erilaisia tietoja riippuen tiedoston tiedostomuodosta. Esimerkiksi ”file” komento kertoo PNG kuvista tiedostomuodon lisäksi niiden koon ja värisyvyyden. Kun komennolla testattiin molempia PNG kuvia, tulostetta tutkittaessa huomataan että kuvat ovat hieman erikokoisia.

```
tensor@Tensor:~$ file /tensorflow/drones/rajatut/LeikattuUusi.png
/tensorflow/drones/rajatut/LeikattuUusi.png: PNG image data, 39778 x 20135, 8-bit/color RGBA, non-interlaced
tensor@Tensor:~$ file /tensorflow/drones/rajatut/LeikattuVanha.png
/tensorflow/drones/rajatut/LeikattuVanha.png: PNG image data, 38607 x 19543, 8-bit/color RGBA, non-interlaced
```

Kuvio 29. Kuvien tiedostomuodon muuton testaus

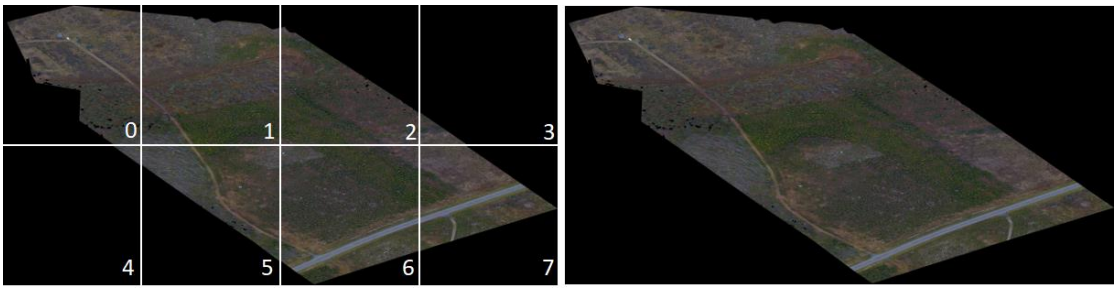
Järjestelmän testaaminen jatkuu tarkistamalla toimiiko läpinäkyvyyden poistamiseen tarkoitettu skriptin osa. Tämän epäonnistuminen ei ole suuri ongelma, koska kyseinen ominaisuus ei ole käytössä olevalla kuvasarjalle tarpeellinen. Tämän toiminnan voi testata tarkastelemalla uudempaa kuvasarjan kuvaa. Alkuperäisessä kuvassa on tallella läpinäkyvyyttä, joten siihen on hyvä verrata. Tarkasteluun sopii mikä tahansa

kuvankäsittelyohjelma, jolla läpinäkyvän osion saa näkyviin. Tässä testauksessa päädyttiin käyttämään ”Gimp” nimistä kuvienkäsittely ohjelmaa. Kuviossa 30 verrataan alkuperäistä kuvaa(vas.) siihen kuvaan, josta läpinäkyvyys on poistettu(oik.). Tästä vertailusta huomaamme, että vasemmassa kuvassa näkyy selvästi läpinäkyvä alue(kuvassa harmaalla) ja oikeassa kuvassa kaikki harmaa alue on korvattu mustalla. Tästä voimme päätellä ominaisuuden toimivan.



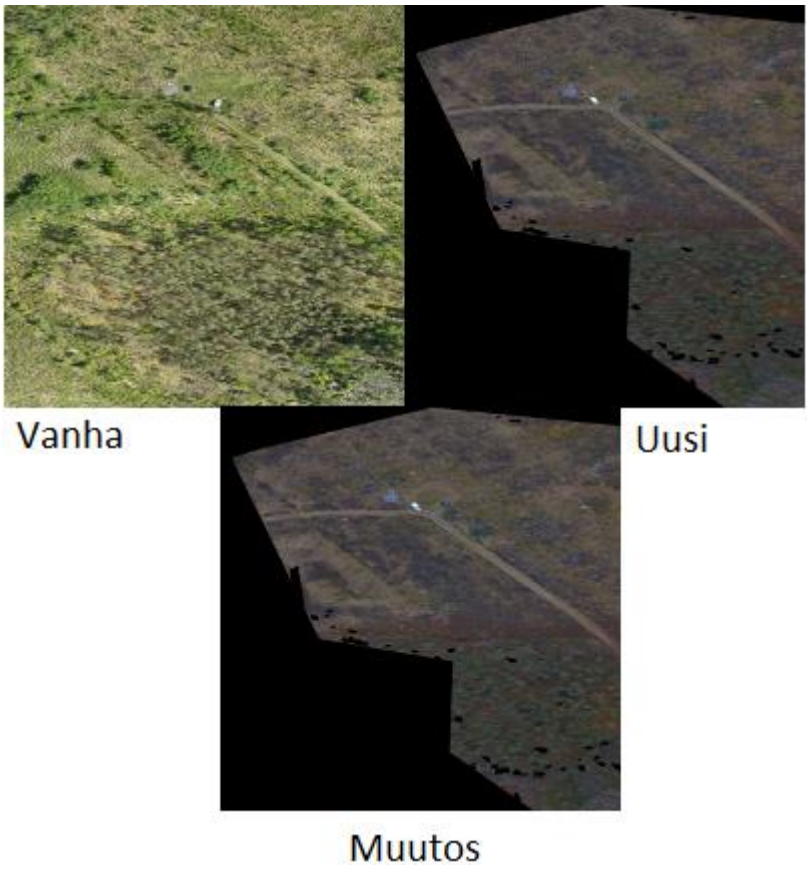
Kuvio 30. Läpinäkyvyyden poiston testaus

Läpinäkyvyyden poistamisen jälkeen järjestelmän tehtävänä on jakaa kuva kahdeksaan pienempään osaan. Tämän toiminnon testaaminen on helpointa verrata leikattavaa kuvaa ja siitä saatua kahdeksaa osaa keskenään. Näistä osista pyritään kasaamaan alkuperäinen kuva laittamalla ne lähekkäin oikeassa järjestyksessä. Kuviossa 31 on esitetty kuvan palaset vasemmalla ja kokonainen kuva oikealla. Kuvan palasiin on myös merkattu niiden nimessä esiintyvä numero. Numerointi kertoo sen, millä periaatteella ImageMagick ohjelma jakaa kuvan osiin. Testin tuloksena voimme todeta, että paloittelu toimii niin kuin pitääkin, eli kuvan paloista pystyy kasaamaan alkuperäisen kuvan ja siitä ei ole mitään aluetta jätetty pois.



Kuvio 31. Palottelutestin tulos

Muutosanalyysi skripti oli seuraavana testivuorossa. Tämän toiminnan testaamisessa aluksi tarkastellaan muutoskohta kuvia. Muutoskohtakuvista tarkastetaan vain, että ne ovat eri osakuvista otettuja. Opinnäytetyön testaus osioon otetaan esille kuitenkin vain yksi esimerkki näistä muutoskohdista. Tässä verrataan kuvasarjan kuvien yhtä osakuvaa ja sen osakuvan muutoskohta kuvaa. Kuviossa 32 verrattiin kuvien osia, joiden numero oli 0, ja kyseisen alueen muutoskohta kuvaa. Kuten aiemmin havaitsimmekin, valoisuuseron takia järjestelmämme luulee koko kuvan olevan muuttunut. Toki silmämääräiselläkin tarkastelulla huomataan kuvasarjan kuvien olevan hyvin erilaiset vaikka samalta alueelta ovatkin. Tämä testi osoittaa, että tämä muutoksen huomaamistapa ei ole toimiva ainakaan tälle kyseiselle kuvasarjalle. Tämä toteutustapa on kuitenkin toimiva, mikäli vertailtavat kuvasarjan kuvat ovat lähes samanlaisia.

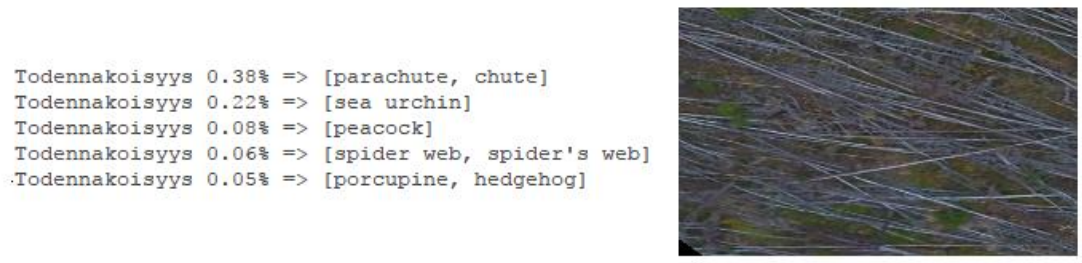


Kuvio 32. Muutoskohdan vertailua alkuperäiseen kuvasarjaan

Muutoskuvien testaamisen jälkeen oli testattava koko skriptiketjun toiminta. Tähän mennessä ne oli aina ajettu erillään, jotta mahdolliset ongelmat ja virhekohdat olisi helpompi rajata. Tällä testillä saatiin kunnon kuva siitä, kuinka paljon aikaa kuluu

tuon skriptiketjun ajamiseen. Kuvien muokkaamisskriptin ajamisessa kesti yli tunti. Python skriptin ajaminen kesti vain noin 15 minuuttia omalla palvelimella olevalla virtuaalikoneella. Molempien skriptien suorittamiseen kuluvat ajat olisivat paljon pienempiä, mikäli käytettävänä olisi enemmän laitteistoresursseja kuten prosessointitehoa. TensorFlow'kin toimisi paljon nopeammin, mikäli olisi käytettävänä tehokas näytönohjain.

Koulutusta testattiin useampaan otteeseen, mutta järjestelmä antaa samoilla muutoskuvilla aina samat luokitteluvaihtoehdot. Kuviossa 33 on esitetty minkä luokituksen järjestelmä antoi kaatuneita puita sisältävälle kuvalle kun sellainen syötettiin neuroverkolle luokiteltavaksi. Tätä testiä varten muokattiin käytössä olevaa python skriptiä, jotta saatiin varmasti syötettyä kaatuneita puita sisältävä kuva neuroverkolle. Tämä skripti on lisätty liitteenä 3. Kaatuneita puita sisältävä kuva on leikattu kuvasarjan uudemmasta kuvasta ja sitä ei ole käytetty järjestelmän kouluttamisessa. Kuvion vasemmassa reunassa on listattu viisi vaihtoehtoa muutoskohdan luokittelulle. Neuroverkolle koulutettu luokittelu ”FallenTrees” ei ole vaihtoehtolistauksessa.



Kuvio 33. Kaatuneiden puiden luokittelu

7 Yhteenveto

7.1 Tulosten analysointi

Kuvien muokkaaminen toimii niin kuin pitääkin. Ainoa huono puoli on se, että omilla laitteilla tehtynä, kuvien muokkaamisskriptin ajamisessa kestää tosi kauan. Toki kuvatkin ovat todella suuri kokoisia, joten niiden muokkaaminen tulee viemään aikaa. Ajan tarvetta lisää myös kuville tehtävien operaatioiden määrä. Tämän bash kielellä

kirjoitetun skriptin suorittamiseen kuluvaa aika saadaan pienennettyä antamalla virtuaalikoneelle enemmän laitteistoresursseja. Etenkin prosessointitehoa lisäämällä saadaan aikaa säästettyä, koska kuvienmuokkaamisoperaatiot näin suurilla tiedostokoilla vaativat paljon prosessointitehoa.

Python skriptin ajaminen kulutti yllättävän vähän aikaa, mutta silläkin on hintansa. Tämän hetkinen Python skripti vaatii virtuaalikoneelta paljon RAM-muistia. Sen pystyy suorittamaan viidellä gigatavulla RAM-muistilla, mutta ei ole varmuutta riittääkö vain neljä gigatavua RAM muistia. Testien perusteella ainakaan kolme gigatavua RAM muistia ei riitä sen suorittamiseen. Python skriptissä on myös toinen ongelma ja se liittyy muutoskohtien löytämiseen kuvista. Tämä toteutustapa on liian herkkä kuvien ”vääristymiselle” eli esimerkiksi jos värit on hieman pielessä normaalista. Silloin tämä tapa ajattelee sitä kuvaa kokonaan uutena.

Neuroverkon hienosäätökouluttamisella ei näytä olevan vaikutusta neuroverkon toimintaan. Ainoa mitä huomaa on se, että häviöfunktion arvo vaihtelee hieman eri koulutuskertojen aikana. Tämä antaa vaikutuksen siitä, ettei koulutusta ole tehty oikein. Se voi olla että pitäisi luoda ensin oma dataset siten, että slim moduulikin tunnistaa sen. Siinä vaan on ongelmana, ettei oikein löydy hirveästi siihen liittyviä ohjeita. Toinen vaihtoehto on se että kouluttaa neuroverkon alusta asti itse imagenet datasetin avulla ja lisää kaatuneita puita varten olevat koulutuskuvat imagenet koulutuskuvien sekaan. Tätä vaihtoehtoa ei tosin omilla laitteilla pysty testaamaan, koska ei riitä laitteistoresurssit. Kolmas vaihtoehto on myös se, että slim moduulin sisäisten neuroverkkomallien kouluttamista ei ole ymmärretty oikein.

Tulosten perusteella voidaan todeta, että TensorFlow’lla pystytään toteuttamaan myrskytuhojen tunnistamiseen toimiva järjestelmä. Se tosin vaatii kunnollista perehtymistä TensorFlow’n toimintaan, neuroverkkoihin ja matematiikkaan. Tämän tapaiset toteutukset vaativat paljon matemaattista älykkyyttä.

7.2 Pohdinta ja jatkokehitys

Opinnäytetyössä oli tarkoituksena saada selville, voidaanko tekoälyä käyttää Maaseutu 2.0-hankkeessa hyödyksi. Tätä kokeiltiin luomalla järjestelmä, jolla saataisiin

ilmakuvista tunnistettua kaatuneita puita. Eli yritettiin luoda järjestelmä myrskyvaurioiden tunnistamista varten. Järjestelmään valittiin testattavaksi Googlen avoimen lähdekoodin koneoppimisohjelmisto nimeltään TensorFlow. Muitakin avoimenlähdekoodin vaihtoehtoja kuitenkin on, mutta päädyttiin testaamaan tätä toimeksiantajan ehdotuksesta. Kyseiseen ohjelmaan oli myös tutustuttava teorian puolella ja samalla oli tutustuttava sen taustalla olevien käsitteiden koneoppimisen ja neuroverkkojen teoriaan.

Opinnäytetyön tulokset antavat hyvän kuvan TensorFlow'n ominaisuuksista ja hyödynnettävyysmahdollisuuksista. Kun TensorFlow'ta osataan hyödyntää oikein, saadaan sillä toteutettua toimiva myrskyvaurioiden tunnistamiseen tarkoitettu järjestelmä. Myös muunlaisiin tarkoituksiin TensorFlow'ta voidaan käyttää.

Selvänä jatkokehityskohteena tälle opinnäytetyölle on neuroverkon laajempi kouluttaminen. Se kannattaa suosiolla tehdä joko jossain pilvipalvelussa tai tehokkaalla palvelin koneella varsinkin jos aikomuksena on kouluttaa neuroverkko kokonaan uusiksi. Jos järjestelmää halutaan hienosäätö kouluttaa, niin ”slim” moduuli tekee siitä hankalaa. Se vaatii erikoisia toimenpiteitä mihin pitäisi neuroverkkoa kouluttaessa perehtyä. Toinen jatkokehitysmahdollisuus olisi luoda järjestelmälle oma mukautettu neuroverkko. Tämä toki voi mennä jo normaalin opinnäytetyön laajuuden yli, varsinkin jos oltaisiin kehittämässä konvoluutioneuroverkkoa ja pyritään muutoskohtienkin löytäminen tekemään kyseisellä neuroverkolla. Tämä kuitenkin voisi avata muitakin vaihtoehtoja siitä, mihin kyseistä järjestelmää voitaisiin käyttää ja niitä voitaisiin jossain opinnäytetyössä kartoittaa.

Tulevissa opinnäytetyössä voitaisiin myös verrata kuinka hyvin muut valmiiksi koulutetut neuroverkkomallit toimisivat tässä järjestelmässä. Tämä toki vaatisi sen, että ensin on joko opinnäytetyöllä tai muuten löydetty toimiva neuroverkon koulutustapa. Tässä jatkokehityksessä voitaisiin esimerkiksi keskittyä neuroverkkojen luokittelujen tarkkuuksiin.

Lähteet

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X. 2016. Tensorflow: A System for Large-scale Machine Learning. Viitattu 2.4.2017. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

Altinas, I., Nguyen, M. N.d. Lecture 6 - Goals and Activities in the Machine Learning Process. Videomateriaalia koneoppimisesta coursera koulutus verkkosivulla. Viitattu 21.5.2017. https://www.coursera.org/learn/big-data-machine-learning/lecture/S7QKN/goals-and-activities-in-the-machine-learning-process

Anderson, G., Anil, R., Aradhye, H., Chai, W., Chandra, T., Cheng, H-T., Corrado, G., Haque, Z., Harmsen, J., Hong, L., Ispir, M., Jain, V., Koc, L., Liu, X., Shah, H., Shaked, T. 2016. Wide & Deep Learning for Recommender Systems. Viitattu 16.4.2017. https://arxiv.org/pdf/1606.07792.pdf

Artificial Neural Networks Technology. N.d. Artikkeli keinotekoisista neuroverkoista. Viitattu 1.3.2017. http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural2.html

Bengio, Y., Frasconi, P., Simard, P. 1994. Learning Long-Term Dependencies with Gradient Descent is Difficult. Viitattu 25.3.2017. http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf

Brownlee, J. 2016. 8 Inspirational Applications of Deep Learning. http://machinelearningmastery.com/inspirational-applications-deep-learning/

Chappel, D. 2015. Introducing Azure Machine Learning. Viitattu 27.2.2017. http://download.microsoft.com/download/3/B/9/3B9FBA69-8AAD-4707-830F-6C70A545C389/Introducing_Azure_Machine_Learning.pdf

Eskelinen, M. 2001. Kaukokartoituksen peruskäsitteet ja vesialueiden tilan tutkiminen. Viitattu 20.5.2017. http://tupa.gtk.fi/raportti/arkisto/rs_2001_1.pdf

Godbout, C. 2016. TensorFlow in a Nutshell – Part Three: All the Models. Viitattu 11.4.2017. https://hackernoon.com/tensorflow-in-a-nutshell-part-three-all-the-models-be1465993930

Keto, K. 2017. Kasvillisuuden kaukokartoitus. Oulun yliopistolla tehty kandidaatintyö. Viitattu 20.5.2017. http://jultika.oulu.fi/files/nbnfioulu-201703311417.pdf

Linear Model. N.d. Artikkeli MathWorks:in verkkosivuilla. Viitattu 12.4.2017. https://se.mathworks.com/discovery/linear-model.html

Lukka, K. 2001. Kari Lukka: Konstruktiivinen tutkimusote. Artikkeli konstruktiivisesta tutkimusotteesta Metodix verkkosivulla. Viitattu 20.5.2017. https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/

Luonnonmaantiede. 2016. Artikkeli Luonnonmaantieteestä Oulun yliopiston sivulla. Viitattu 20.5.2017. http://www.oulu.fi/maantiede/luonnonmaantiede

Maaseutu 2.0. N.d. Artikkeli Maaseutu 2.0-hankkeesta Aito maaseutu verkkosivulla. Viitattu 20.5.2017. https://www.aitomaaseutu.fi/hankkeet/maaseutu-2-0

Maaseutu 2.0 siirtää metsänomistajan digiaikaan. 2016. Artikkeli Maaseutu 2.0.-hankkeesta Jyväskylän ammattikorkeakoulun verkkosivuilta. Viitattu 28.2.2017. https://www.jamk.fi/fi/Tietoa-JAMKista/Materiaalit/asiakaslehti-12016/maaseutu-2.0-siirtaa-metsanomistajan-digiaikaan/

Nielsen, M. 2017. Using neural nets to recognize handwritten digits. Viitattu 7.3.2017. http://neuralnetworksanddeeplearning.com/chap1.html

Olah, C. 2015, Understanding LSTM Networks. Viitattu 25.3.2017. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Russel, I. 1996. Brief History of Neural Networks. Viitattu 1.3.2017. http://uhaweb.hartford.edu/compsci/neural-networks-history.html

Schapire, R. 2008. COS 511: Theoretical Machine Learning. Viitattu 28.2.2017. http://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf

Sigmoid Function. N.d. Sigma funktion selitys Wolfram Mathworld nimisestä matematiikan tietokannasta. Viitattu 1.3.2017. http://mathworld.wolfram.com/SigmoidFunction.html

Srihari, S. N.d. Long-Short Term Memory. Viitattu 25.3.2017. http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

TensorFlow Architecture. 2017. Artikkeli TensorFlow'n verkkosivuilla. Viitattu 4.4.2017. https://www.tensorflow.org/extend/architecture

TensorFlow-Slim image classification library. N.d. Tensorflow'n slim moduulin ohjeet GitHub-sivustolla. Viitattu 16.5.2017. https://github.com/tensorflow/models/tree/master/slim

Tukivektoriluokittelija. N.d. Oulun yliopisto. Viitattu 12.4.2017. http://www.ee.oulu.fi/research/tklab/courses/521497S/pruju/SVM.pdf

Vojt, J. 2016. Deep neural networks and their implementation. Diplomityö. Prahan Charles yliopisto matematiikan ja fysiikan laitos. Viitattu 6.4.2017 https://is.cuni.cz/webapps/zzp/download/120229251/?lang=cs

Liitteet

Liite 1. Kuvien muokkaamista varten oleva skripti.

```
#!/bin/bash

#ilmakuvien muokkaaminen järjestelmälle sopiviksi, jonka jälkeen ne syötetään järjestelmään.

#Laitetaan talteen skriptin käynnistyksessä annetut tiedostosijainnit

uusi=$1

vanha=$2

#Molemmista kuvista leikataan alue koordinaattien perusteella

gdalwarp -t_srs EPSG:4326 -te 26.3400650 62.6694936 26.3501071 62.6745768 $uusi /tensorflow/drones/rajatut/LeikattuUusi.tif

gdalwarp -t_srs EPSG:4326 -te 26.3400650 62.6694936 26.3501071 62.6745768 $vanha /tensorflow/drones/rajatut/LeikattuVanha.tif

#Muutetaan molempien kuvien tiedostomuodot

convert /tensorflow/drones/rajatut/LeikattuUusi.tif /tensorflow/drones/rajatut/LeikattuUusi.png

convert /tensorflow/drones/rajatut/LeikattuVanha.tif /tensorflow/drones/rajatut/LeikattuVanha.png

#Poistetaan molemmista kuvista mahdolliset näkymättömällä värillä merkatut alueet

convert /tensorflow/drones/rajatut/LeikattuUusi.png -alpha off /tensorflow/drones/rajatut/LeikattuAlueUusi.png

convert /tensorflow/drones/rajatut/LeikattuVanha.png -alpha off /tensorflow/drones/rajatut/LeikattuAlueVanha.png

#Leikataan molemmat kuvat pienempiin osiin

convert -crop 25%x50% /tensorflow/drones/rajatut/LeikattuAlueUusi.png /tensorflow/drones/leikatut/LeikattuUusi-jaoteltu.png

convert -crop 25%x50% /tensorflow/drones/rajatut/LeikattuAlueVanha.png /tensorflow/drones/leikatut/LeikattuVanha-jaoteltu.png


#Varmistetaan että slim moduuli on asennettu, käynnistetään tensorflow ja viimeisenä ajetaan muutoksen tarkastelu python skripti

source /home/tensor/tensorflow/bin/activate

python /tensorflow/scripts/analyysi.py
```

Liite 2. Muutosanalyysi skripti.

```
#coding=utf-8
import numpy as np
import os
import tensorflow as tf
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from PIL import Image
from PIL import ImageChops
from datasets import dataset_utils
from datasets import imagenet
from nets import inception
from preprocessing import inception_preprocessing
for x in range(8):
        print(x)
        im1 = Image.open("/tensorflow/drones/leikatut/LeikattuUusi-jaoteltu-{}.png" .format(x))
        im2 = Image.open("/tensorflow/drones/leikatut/LeikattuVanha-ja-oteltu-{}.png" .format(x))
        diff = ImageChops.difference(im2, im1).getbbox()
        print "Kuvion osassa on ",diff," kohdassa muutos"
        if diff != None:
                muutosnimi = ("/tensorflow/drones/muutos-{}.png".format(x))
                muutos = im1.crop((diff))
                muutos.save(muutosnimi,"PNG")
                if muutos.getbbox() == None:
                        print ('Kuvan osa {} sisaltaa pelkkaa mustaa, sitä on turha syottaa neuroverkkoon' .format(x))
                else:
                        slim = tf.contrib.slim
```

```
                image_size = inception.inception_v4.de-
fault_image_size

                checkpoints_dir = '/tensorflow/checkpoints'
                with tf.Graph().as_default():
                        tarim = ''
                        with open (muutosnimi) as f:
                                tarim = f.read()
                        tarka = tf.image.de-
code_png(tarim, channels=3)
                        prosessointi = inception_prep-
rocessing.preprocess_image(tarka, image_size, image_size, is_training=False)
                        prosessoitu =
tf.expand_dims(prosessointi, 0)


                        with slim.arg_scope(incep-
tion.inception_v4_arg_scope()):
                                lo-
gits, _ = inception.inception_v4(prosessoitu, num_classes=1001, is_training=False)
                        todennakoisyys = tf.nn.soft-
max(logits)


                        init_fn = slim.as-
sign_from_checkpoint_fn(
        os.path.join(checkpoints_dir, 'inception_v4.ckpt'),

        slim.get_model_variables('InceptionV4'))


                        with tf.Session() as sess:

        init_fn(sess)

        np_image, todennakoisyys = sess.run([tarka, todennakoisyys])
                                to-
dennakoisyys = todennakoisyys[0, 0:]
                                sor-
ted_inds = [i[0] for i in sorted(enumerate(-todennakoisyys), key=lambda x:x[1])]
```

```
                                                    plt.figure()
                                                    plt.imshow(np_image.as-
type(np.uint8))
                                                    plt.axis('off')
                                                    plt.show()
                                                    print ("Kuvan osan {} sisältä-
mässä muutoskohdassa on järjestelmän mielestä:".format(x))
                                                    nimi = imagenet.create_reada-
ble_names_for_imagenet_labels()
                                                    for i in range (5):
                                                                        in-
dex = sorted_inds[i]

                print("Todennakoisyys %0.2f%% => [%s]" % (todennakoisyys[index],
nimi[index]))


                else:

                                print ('Käsittelyssä olleeseen kuvan osaan {0} ei ole tapah-
tunut muutosta' .fomrat(i))
```

Liite 3. Kaatuneiden puiden luokittelu testaus skripti.

```
import numpy as np
import os
import tensorflow as tf
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

from datasets import dataset_utils
from datasets import imagenet
from nets import inception
from preprocessing import inception_preprocessing

slim = tf.contrib.slim
image_size = inception.inception_v4.default_image_size
checkpoints_dir = '/tensorflow/checkpoints'

with tf.Graph().as_default():
    tarim = ''
    with open ('/tensorflow/drones/testi.png') as f:
        tarim = f.read()
    tarka = tf.image.decode_png(tarim, channels=3)
    prosessointi = inception_preprocessing.preprocess_image(tarka, image_size,
image_size, is_training=False)
    prosessoitu = tf.expand_dims(prosessointi, 0)

    with slim.arg_scope(inception.inception_v4_arg_scope()):
        logits, _ = inception.inception_v4(prosessoitu, num_classes=1001, is_trai-
ning=False)
    todennakoisyys = tf.nn.softmax(logits)

    init_fn = slim.assign_from_checkpoint_fn(
```

```
            os.path.join(checkpoints_dir, 'inception_v4.ckpt'),
            slim.get_model_variables('InceptionV4'))

    with tf.Session() as sess:
        init_fn(sess)
        np_image, todennakoisyys = sess.run([tarka, todennakoisyys])
        todennakoisyys = todennakoisyys[0, 0:]
        sorted_inds = [i[0] for i in sorted(enumerate(-todennakoisyys), key=lambda
x:x[1])]

    plt.figure()
    plt.imshow(np_image.astype(np.uint8))
    plt.axis('off')
    plt.show()

    nimi = imagenet.create_readable_names_for_imagenet_labels()
    for i in range (5):
        index = sorted_inds[i]
        print("Todennakoisyys %0.2f%% => [%s]" % (todennakoisyys[index], nimi[in-
dex]))
```