jamk.fi

Kasvilajien tunnistaminen neuroverkon avulla

Toni Janatuinen

Opinnäytetyö
Toukokuu 2020
Tekniikan ala
Insinööri (AMK), sähkö- ja automaatiotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Janatuinen, Toni	Opinnäytetyö, AMK	Toukokuu 2020
	Sivumäärä 38	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Kasvilajien tunnistaminen neuroverkon avulla

Tutkinto-ohjelma
Insinööri (AMK), sähkö- ja automaatiotekniikan tutkinto-ohjelma

Työn ohjaaja(t)
Juho Riekkinen, Ari Kuisma

Toimeksiantaja(t)
Jyväskylän ammattikorkeakoulu

Tiivistelmä

Neuroverkoilla on mahdollista tunnistaa kuvista lukuisia asioita hyvin tarkasti, kuten keuhkokuumetta röntgenkuvista tai luomien pahanlaatuisuutta. Tämä on avannut kuluttajille mahdollisuuden tunnistaa arkipäiväisiä asioita erilaisilla neuroverkkoja hyödyntävillä sovelluksilla.

Työn tavoitteena oli tuottaa neuroverkkosovellus, joka tunnistaa kymmenen asiakasyrityksen määrittelemää kasvilajia. Kuvat kasvilajeista neuroverkon opetusprosessia varten saatiin asiakasyritykseltä. Toisena tavoitteena oli tutkia, päästäänkö konvoluutioverkolla riittävään tarkkuuteen hyödyntäen myös siirrettyä oppimista. Konvoluutioverkko on syvä neuroverkko, jossa on konvoluutiokerroksia, ja sitä opetettiin syöttämällä kuvia valituista kasvilajeista

Tuloksena saavutettiin toimiva malli, joka tunnisti kaikki kuusi testikuvaa jokaisesta kymmenestä lajista. Neuroverkko ei oppinut tunnistamaan kasvilajeja, kun sitä opetettiin ilman siirrettyä oppimista. Kuitenkin siirretyn oppimisen avulla päästiin haluttuun tulokseen. Siirretyssä oppimisessa käytettiin hyödyksi toisessa neuroverkkomallissa opittuja representaatioita, joiden avulla on mahdollista vähentää tarvittavaa opetusdatan määrää.

Johtopäätöksenä voidaan todeta, että kasvilajeja pystytään tunnistamaan neuroverkon avulla myös silloin, kun opetuskuvia on erityisen vähän, kunhan käytetään siirrettyä oppimista. Tämä avaa yrityksille lukuisia mahdollisuuksia toteuttaa neuroverkkoja hyödyntäviä sovelluksia kuluttajille.

Avainsanat (asiasanat)
Kuvantunnistus, neuroverkko, konvoluutioverkko, tekoäly

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Janatuinen, Toni	Bachelor's thesis	May 2020
		Language of publication: Finnish
	Number of pages: 38	Permission for web publication: x

Title of publication

Recognizing plant species with neural network

Degree programme

Bachelor's degree program in Electrical and Automation Engineering

Supervisor(s)

Juho Riekkinen, Ari Kuisma

Assigned by

Jyväskylän ammattikorkeakoulu

Abstract

With neural networks, it is possible to very accurately recognize numerous things from images, like pneumonia from x-ray images or if a nevus is malignant. This has opened the possibility for consumers to use different applications that utilize neural networks to recognize everyday objects

The goal was to develop a neural network application, which recognizes ten plant species that the client company specified. Images of the plant species for teaching the neural network were received from the client company. The second goal was to research if it is possible to achieve sufficient accuracy, while also utilizing transfer learning. A convolutional neural network is a deep neural network that has convolutional layers. It is taught by feeding it pictures of the selected plant species.

As a result, a working model was achieved, that recognized all six test images from all ten plant species. The neural network did not learn to recognize the species when trained without transfer learning. With transfer learning however, the desired result was achieved. Transfer learning utilized representations learned from another neural network model, which makes it possible to reduce the amount of training data required.

In conclusion, it could be determined plant species with a neural network can be recognized, even when very small training image dataset is available, as long as transfer learning is used. This opens numerous possibilities for companies to develop applications that utilize neural networks for the consumers.

Keywords/tags (subjects)

Image recognition, neural network, convolutional neural network, artificial intelligence

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

Neuroverkot ja tekoäly ovat olleet viime aikoina paljon esillä eri yhteyksissä, kuten kuvantunnistuksessa tai itsestään ajavien autojen parissa. Tarve työlle tuli asiakasyrityksen vielä toistaiseksi salassa pidettävistä projekteista. Lisäksi toteutus on tärkeä osa asiakasyrityksen neuroverkoihin liittyviä projekteja.

Työn päätavoitteena oli selvittää, kuinka neuroverkon opettaminen onnistuu Jyväskylän ammattikorkeakoulun (JAMK) asiakasyrityksen tarjoamilla kuvilla ja pystyykö siitä saatua mallia hyödyntämään kasvilajien tunnistamisessa. Tavoitteena oli myös tuottaa neuroverkkosovellus, joka tunnistaa kymmenen asiakasyrityksen määrittelemää kasvilajia.

Asiakasyritykselle tuli toteuttaa myös neuroverkko, jota se voi opettaa jatkossa myös useammilla kasvilajeilla omia projektejaan varten. Lisäksi asiakasyritykselle tuli tuottaa erillinen ohjeistus neuroverkon käyttöä varten. Tavoitetulokseksi määriteltiin 95 %:n tarkkuus kasvilajien tunnistuksessa.

Raportissa pohditaan myös jatkokehitysmahdollisuuksia ja mallin skaalautuvuutta yli kymmenelle kasvilajille.

Toimeksiantajana toimi Jyväskylän ammattikorkeakoulu (JAMK), joka on vuonna 1994 perustettu korkeakoulu. JAMK:ssa opiskelee n. 8500 opiskelijaa ja niistä valmistuu vuosittain yli 1500 opiskelijaa. JAMK tuottaa korkeakouluun johtavaa koulutusta, ammatillista opettajakoulutusta, täydennyskoulutusta ja avoimia korkeakouluopintoja. (Tutustu JAMKiin 2019.)

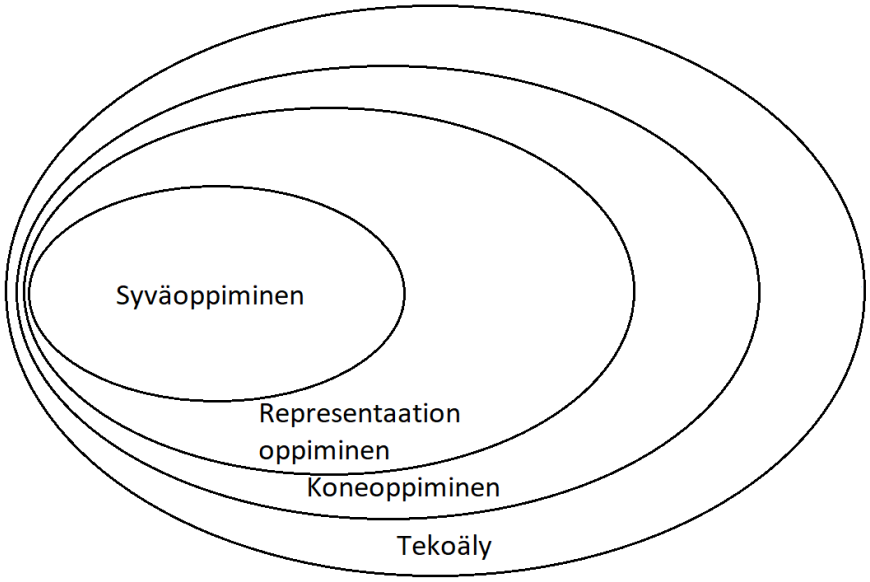
2 Tekoäly ja sen osa-alueet

2.1 Tekoäly

Tekoäly on yksi tieteen ja tekniikan uusimmista aloista, ja sillä pyritään ymmärtämään älyn toimintaa ja rakentamaan älyllisiä sovelluksia. Tekoälyllä on lukuisia sovelluksia yleisistä ongelmista rajattuihin aiheisiin, kuten shakin pelaaminen, matemaattisten teorioiden todistaminen, runojen kirjoittaminen, auton ajaminen ja sairauksien diagnosointi. Tekoäly on siis olennainen mihin tahansa älyllistä toimintaa vaativaan tehtävään. (Russell & Norvig 2016, 1.) Nykypäivänä tekoälyllä voidaan esimerkiksi ajaa autoa, tunnistaa puhetta, suunnitella ja aikatauluttaa, pelata pelejä, suodattaa roskapostia, suunnitella logistiikkaa, kääntää kieliä ja ohjata robotteja. Tekoälyllä on myös lukuisia muita sovelluksia jo nykyään. (Russell & Norvig 2016, 28-29.)

Tekoälyn toiminnan määrittelyssä voidaan käyttää esimerkkinä neljää kategoriaa: Inhimillinen ajattelu, järkiperäinen ajattelu, inhimillinen toiminta ja järkiperäinen toiminta. Inhimillisellä ajattelulla ja toiminnalla verrataan tarkkuutta ihmisten toimintaan. Järkiperäisellä ajattelulla ja toiminnalla puolestaan verrataan tarkkuutta ideaaliseen toimintaan. (Russell & Norvig 2016, 1-2.)

Tekoälyn osa-alueisiin kuuluvat syväoppiminen, representaation oppiminen, koneoppiminen ja tekoäly. Syväoppiminen on osa representaation oppimista, joka taas on osa koneoppimista, joka puolestaan on osa tekoälyä (Goodfellow, Bengio & Courville 2016, 8-9). Tekoäly puolestaan on osa tietojenkäsittelytieteitä (Elements of AI: Mitä tekoäly on? 2019). Kuviosta 1 nähdään esimerkki, miten tekoälyn eri osa-alueet ovat osa toisiaan.



Kuvio 1. Venn-diagrammi tekoälyn osa-alueista

2.2 Koneoppiminen

Tekoälyjärjestelmillä on kyky pystyä hakemaan itse tietoa muodostamalla kaavoja raakadatasta. Tätä toimintoa kutsutaan koneoppimiseksi. Koneoppiminen mahdollistaa sellaisten ongelmien ratkaisun, jotka vaativat informaatiota oikeasta maailmasta. Yksi yksinkertainen koneoppimisen algoritmi on logistinen regressio, jolla voidaan esimerkiksi suositella keisarileikkauksen tekemistä tai tekemättä jättämistä. Toinen yksinkertainen koneoppimisen algoritmi on Naiivi bayes, jolla esimerkiksi voidaan erotella roskaposti tärkeästä sähköpostista. Koneoppimisalgoritmien suorituskyky perustuu pitkälti sen datan laatuun, jota niille syötetään. Algoritmille syötettävä data muodostaa piirteitä ja piirteet luovat kokonaiskuvan, jota kutsutaan representaatioksi. (Goodfellow ym. 2016, 3.)

Logistisella regressiolla ei kuitenkaan onnistu järkevien ennustusten tekeminen esimerkiksi magneettikuvista. Yksittäisten pikselien data ei tuo sellaista informaatiota, että logistisesta regressiosta olisi hyötyä. Tätä varten voidaan käyttää koneoppimista omien piirteiden ja representaatioiden oppimiseen. Opitut representaatiot tuottavat

usein parempia tuloksia kuin manuaalisesti suunnitellut representaatiot. Oppiminen myös mahdollistaa mukautumisen uusiin tehtäviin, joihin silti tarvitaan vähäistä ohjausta ihmiseltä. Yksinkertaisissa ongelmissa representaation oppimisalgoritmi löytää sopivat piirteet ongelman ratkaisemiseen minuuteissa. Kuitenkin jos ongelma on haastava, siihen voi mennä tunneista kuukausiin. Kokonaisella tutkijaryhmällä voi kulua vuosikymmeniä haastavan ongelman manuaaliseen piirteiden etsimiseen. (Goodfellow ym. 2016, 3-4.)

Koneoppimisalgoritmit voidaan jakaa karkeasti kahteen osa-alueeseen, eli ohjattuun oppimiseen ja ohjaamattomaan oppimiseen. Osa-alueet määrittyvät esimerkiksi sen mukaan, miten data valmistellaan algoritmia varten. Ohjatussa oppimisessa opetusdata nimetään ja lajitellaan, kun taas ohjaamattomassa oppimisessa tuodaan data ilman nimikkeitä tai lajittelua. Voidaan siis ajatella, että ohjatussa oppimisessa on ohjaaja tai opettaja ja ohjaamattomassa oppimisessa ei kumpaakaan. Ohjattua oppimista voidaan käyttää esimerkiksi kuvantunnistuksessa, jossa opetuskuvat lajitellaan nimikkeitä vastaaviin polkuihin. Kun vaikka kurjenmiekka -kasveista luokitellaan lajit erikseen opetusta varten, pystytään ohjatulla oppimisella tunnistamaan kurjenmiekan eri lajit. Ohjaamattomalla oppimisella puolestaan haetaan hyödyllistä tietoa datasetin rakenteesta, kuten todennäköisyyksien jakauma. Ohjaamattomalla oppimisella voidaan tehdä myös muunlaisia tehtäviä, kuten klusterointia. Klusteroinnissa keskenään samankaltainen data jaetaan erillisiin ryhmittymiin. (Goodfellow ym. 2016, 102-103.)

Koneoppimisalgoritmien suorituskykyä mitataan usein mallin tarkkuudella, joka ilmoittaa, kuinka moni ennustetuista tuloksista on oikein suhteessa kaikkiin tuloksiin. Vastaavasti voidaan mitata mallin häviötä, joka ilmoittaa väärin arvattujen tulosten määrän. Suorituskykyä mitattaessa halutaan nähdä, miten koneoppimisalgoritmit suoriutuvat datasta, jota ne eivät ole aikaisemmin nähneet. Sitä varten suorituskykyä mitataan erillisellä testidatajoukolla, jota ei ole käytetty opettamisessa. (Goodfellow ym. 2016, 101-102.) Käytettävissä oleva data voidaan jakaa kolmeen eri joukkoon, opetusdatajoukkoon, testidatajoukkoon ja validointidatajoukkoon. Opetusdatajoukko on varsinainen data, jota käytetään mallin opetuksessa. Validointidatajoukkoa

voidaan käyttää parametrointiin ja mallin suoriutumisen arviointiin opetuksen aikana, mutta malli ei koskaan opi suoraan validointidatasta. Testidatajoukkoa käytetään valmiin mallin suorituskyvyn testaamiseen. Validointidatajoukko ja testidatajoukko voidaan ottaa opetusdatasta, jolloin opetusdatan määrä vähenee. Usein data jaetaan aluksi kahteen osaan, opetusdatajoukkoon ja testidatajoukkoon. Tämän jälkeen testidatajoukko voidaan pitää sellaisenaan ja opetusdatajoukosta siirretään tietty määrä, esim. 20 %, validointidataksi. Joskus opetusdataa voi olla niukasti, jolloin validointidatan määrää voi vähentää tai validointidatajoukon voi jättää kokonaan pois. (Shash 2017.)

Joskus ratkottava ongelma on luonteeltaan sellainen, ettei mallin tarkkuus ole paras mahdollinen mittari mittaamaan suorituskykyä. Tällainen ongelma on esimerkiksi erittäin harvinaisen sairauden havaitseminen, jota sairastaa kaksi miljoonasta ihmisestä. Jos neuroverkko luulee kaikkien olevan terveitä, sillä saavutetaan erittäin suuri tarkkuus, koska vain kaksi miljoonasta arvauksesta menee väärin. Tällaisia ongelmia ratkottaessa on parempi käyttää sisäistä tarkkuutta ja sensitiivisyyttä mittaamaan mallin suorituskykyä. Sisäinen tarkkuus on oikein arvattujen tulosten suhde kaikkiin tuloksiin. Sensitiivisyys puolestaan on oikein arvattujen tulosten suhde ennustettuihin tuloksiin. Näissä tilanteissa, jos neuroverkko uskoisi, ettei kenelläkään ole kyseistä sairautta, saavutettaisiin lähes täydellinen tarkkuus, mutta huono sensitiivisyys ja sisäinen tarkkuus. Jos taas neuroverkko ilmoittaisi kaikilla olevan kyseinen sairaus, sisäinen tarkkuus olisi erittäin huono ja sensitiivisyys puolestaan täydellinen. On siis tärkeää pyrkiä mahdollisimman suureen sensitiivisyyteen, mutta samalla kiinnittää huomiota sisäiseen tarkkuuteen. (Goodfellow ym. 2016, 418.) Näitä voidaan havainnollistaa muutamalla laskuesimerkillä. Sisäinen tarkkuus lasketaan yhtälöllä 1.

$$\frac{op}{op+vp} * 100 = st \qquad (1)$$

missä op = oikeat positiiviset
vp = väärät positiiviset
st = sisäinen tarkkuus

Sensitiivisyys lasketaan yhtälöllä 2.

$$\frac{op}{op+vt} * 100 = Sensitiivisyys \quad (2)$$

missä op = oikeat positiiviset

vn = väärät negatiiviset

Tarkkuus lasketaan yhtälöllä 3.

$$\frac{Oikeat\ arvaukset}{Kaikki\ arvaukset} * 100 = Tarkkuus \quad (3)$$

Edellä esitetyssä harvinaisen sairauden tapauksessa sisäinen tarkkuus tilanteessa, jossa neuroverkko ennustaa kaikkien olevan sairaita, on laskettuna seuraava:

$$\frac{2}{2 + 1000000} * 100 \approx 0{,}0002\%$$

Sensitiivisyys samassa tapauksessa on laskettuna puolestaan seuraava:

$$\frac{2}{2 + 0} * 100 = 100\%$$

Tarkkuus samassa tapauksessa on laskettuna seuraava:

$$\frac{2}{1000000} * 100 = 0{,}0002\%$$

Sitä vastoin sisäinen tarkkuus tapauksessa, jossa neuroverkko ennustaa kaikkien olevan terveitä, on laskettuna seuraava:

$$\frac{0}{0} * 100 = 0\%$$

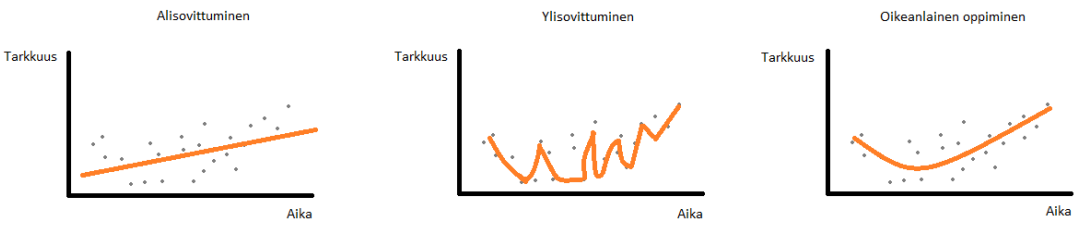
Sensitiivisyys tässä tapauksessa on laskettuna seuraava:

$$\frac{0}{0+2} * 100 = 0\%$$

Ja tarkkuus laskettuna tapauksessa, jossa neuroverkko ennustaa kaikkien olevan terveitä, on seuraava:

$$\frac{999998}{1000000} * 100 \approx 100\%$$

Yksi koneoppimisen keskeisimmistä haasteista on algoritmien suoriutuminen uuden datan parissa, eli testidatan, jota ei ole käytetty opettamisessa. Kun algoritmit suoriutuvat hyvin uuden datan tunnistamisessa, sitä kutsutaan kyvyksi yleistää. Koneoppimisalgoritmeja käytettäessä pyritään parantamaan tuloksia ja pienentämään virheitä muuttamalla parametreja opetusten välissä. Parametrien säätämisellä pyritään välttämään alisovittumista ja ylisovittumista. Ylisovittumista tapahtuu, kun opetusvirhe on pieni ja testivirhe kasvaa liian suureksi. Alisovittumista puolestaan tapahtuu, kun opetusvirhe ja testivirhe kasvavat molemmat liian suuriksi. Käytännössä ylisovittumisella tarkoitetaan tilannetta, kun malli oppii tunnistamaan piirteitä opetuskuvista, jotka eivät hyödytä testikuvien tunnistamisessa, eli opetustarkkuus on hyvä mutta validointitarkkuus huononee. Esimerkiksi keskittyminen kuvan taustan tekstuureihin tunnistettavan objektin sijasta on ylisovittumista. (Goodfellow ym. 2016, 108-110.) Kuviossa 2 nähdään esimerkki, miltä oikeanlainen neuroverkon oppiminen, ylisovittuminen ja alisovittuminen voi näyttää.



Kuvio 2. Esimerkki neuroverkon sovittumisesta

2.3 Representaation oppiminen

Representaation oppimisessa koneoppimisalgoritmien kyky löytää sopivat piirteet yksinkertaista tehtävää varten minuuteissa mahdollistaa myös tekoälyjärjestelmien nopean mukautumisen uusiin tunnistustehtäviin vähäisellä ihmisten avulla. Tyypillinen esimerkki representaation oppimisalgoritmista on autoenkooderi. (Goodfellow ym. 2016, 4.)

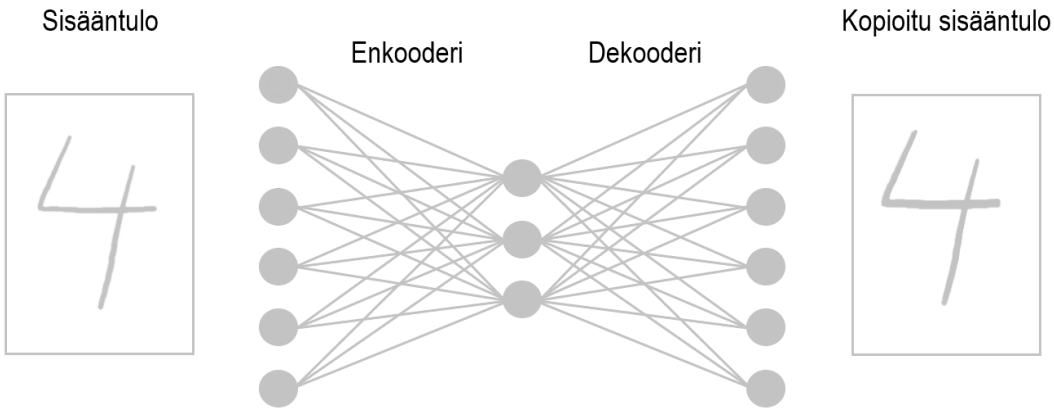
Eteenpäin syöttävien neuroverkkojen toiminnan voidaan ajatella olevan tavallaan representaation oppimista. Neuroverkon viimeinen kerros on tyypillisesti lineaarinen luokittelija, kuten softmax -funktio. Muu osa neuroverkosta oppii tuottamaan representaation viimeisen kerroksen funktiolle. Jokaiselle piilossa olevalle kerrokselle muodostuu omanlainen representaatio, jota syötetään aina seuraavalle kerrokselle. Nämä representaatiot sisältävät piirteitä, jotka edesauttavat tunnistustehtävissä. (Goodfellow ym. 2016, 525.)

2.3.1 Autoenkooderi

Autoenkooderi on enkooderin ja dekooderin yhdistelmä, jossa enkooderi muuttaa sisääntulodatan uuteen representaatioon ja dekooderi muuttaa sen takaisin alkuperäiseen muotoon. Erilaisia autoenkoodereita käytetään eri ominaisuuksien ja tulosten saavuttamiseen. (Goodfellow ym. 2016, 4.) Käytännössä autoenkooderi on neuroverkko, joka opetetaan kopioimaan sisääntulo ulostuloon. Kuitenkin jos autoenkooderi kopioisi sisääntulon täydellisesti ulostuloon, se ei olisi kovinkaan hyödyllinen. Tämän takia ne on suunniteltu niin, ettei kopiointi ole täydellistä. Sen saavuttamiseksi ne voivat yleensä kopioida ainoastaan likiarvoja datasta. Autoenkooderit usein oppivat tärkeitä piirteitä datasta, koska ne priorisoivat tiettyjä puolia mitä sisääntulodatasta kopioidaan. (Goodfellow ym. 2016, 499.)

Perinteisesti autoenkoodereita on käytetty dimensioiden pienentämiseen tai piirteiden oppimiseen ja viime aikoina sitä on käytetty myös generatiivisissa neuroverkoissa. Autoenkoodereita voidaan pitää erikoistapauksena eteenpäin syöttävistä

neuroverkoista ja niitä voidaan opettaa samoilla tekniikoilla kuin normaaliakin eteenpäin syöttävää neuroverkkoa. Toisin kuin normaali eteenpäin syöttävä neuroverkko, autoenkooderia voidaan opettaa myös ns. kierrätyksellä. Kierrätyksellä tarkoitetaan oppimisalgoritmia, jonka toiminta perustuu alkuperäisen sisääntulodatan ja kopioidun datan vertailuun keskenään. (Goodfellow ym. 2016, 499-500.) Kuviossa 3 nähdään periaatteellinen kuva autoenkooderin arkkitehtuurista.



Kuvio 3. Esimerkki autoenkooderista

2.3.2 Siirretty oppiminen

Ihmisillä on kyky hyödyntää oppimiaan asioita uusissa tehtävissä. Se tieto, mitä saadaan kerättyä yhdestä tehtävästä, voi olla hyödyllinen toisessa. Esimerkiksi se, että osaa ajaa moottoripyörällä, voi auttaa oppimaan auton ajamista. Perinteiset koneoppimisalgoritmit ovat toimineet eristyksissä ja suorittaneet vaan tiettyjä tehtäviä. Siirretyllä oppimisella liikutaan kuitenkin pois siitä. Se, että neuroverkkoja ei voisi opettaa ilman miljoonia luokiteltuja kuvia, on nykyään myytti. Todellisuudessa voidaan oppia hyödyllisiä representaatioita toisista kuvista ja käyttää siirrettyä oppimista, jotta päästään parempiin tuloksiin. (Sarkar 2018.)

Siirretyssä oppimisessa opitaan representaatioita tietyssä tilanteessa ja niitä hyödynnetään toisessa tilanteessa yleistämisessä. Esimerkiksi voidaan aluksi opettaa sovellus tunnistamaan kissoja ja koiria. Siitä saatua representaatiota taas käytetään eri kategorian opetuksessa, kuten muurahaisten ja mehiläisten tunnistamisessa. Jos ensimmäisessä opetustilanteessa on ollut huomattavasti enemmän opetusdataa, se voi auttaa toisessa opetustilanteessa representaatioiden oppimisessa, jos opetusdataa on vähäisesti. (Goodfellow ym. 2016, 534.)

Syvissä neuroverkoissa on useita kerroksia, jotka oppivat erilaisia piirteitä ja ne kerrokset on lopussa yhdistetty viimeiseen kerrokseen, joka ohjatussa oppimisessa on täysin yhdistetty kerros. Useiden kerrosten arkkitehtuuri mahdollistaa tiettyjen kerrosten, kuten vaikka viimeisen täysin yhdistetyn kerroksen, pois jättämisen esiopetetuista malleista. Tämä mahdollistaa esiopetetun mallin ja sen oppimien representaatioiden hyödyntämisen muissa tunnistustehtävissä. (Sarkar 2018.)

2.4 Syväoppiminen

Ensimmäiset neuroverkot kehitettiin 1940-luvulla. Ensimmäisissä malleissa oli ainoastaan pieni määrä virtuaalisia neuroneita ja muutamia painoarvoja, jotka yhdistivät neuronit toisiinsa. Painoarvot määrittivät, miten tieto kulkee neuronien välillä. 1980-luvulla alkoi tulemaan syvempiä malleja, joissa oli yli kaksi kerrosta. Vuosien 1985 ja 1990 välillä oli niin sanottu AI-talvi, jonka aikana tekoälyn kehitys hidastui suosion hiipumisen takia. Siihen aikaan tuli yksinkertaisempia ja tehokkaampia ratkaisuja, joitten takia valtiot ja yritykset lopettivat neuroverkkojen tukemisen. Viime vuosikymmenen aikana on syvien neuroverkkojen saralla tehty lukuisia läpimurtoja, jotka ovat taas herättäneet akateemisten alojen kiinnostuksen, kuten myös teollisuuden kiinnostuksen. (Di, Bhardwah & Wei 2018, 10-12.)

Syväoppimisessa koneet ottavat opikseen omista kokemuksistaan ja luovat käsitystä maailmasta konseptien hierarkialla. Yksinkertaisemmat konseptit muodostavat monimutkaisempia konsepteja ja lopulta nämä konseptit muodostavat erittäin syvän kerroskokonaisuuden. Syvän kerroskokonaisuuden takia sitä kutsutaan syväoppimiseksi. (Goodfellow ym. 2016, 1.) Syväoppimisella pyritään matkimaan neuronien toimintaa

aivoissa. Yksi suurimmista syväoppimisen hyödyistä on kyky oppia representaatioita monella tasolla. Se mahdollistaa monimutkaisten funktioiden luomisen suurimmaksi osaksi ilman ihmisten apua. Syväoppiminen luo myös mahdollisuuden esiopettamiselle, jolla tarkoitetaan representaation oppimista eri opetusdatasta ja hyödyntämällä sitä toisessa tilanteessa. (Di ym. 2018, 8.)

Syväoppimismalli koostuu useista kerroksista, jotka toimivat yhdessä luodakseen parannellun piirreavaruuden. Ensimmäinen kerros oppii piirteitä, kuten väri ja reunat, kun taas toinen kerros oppii ylemmän tason piirteitä, kuten kulmia, ja kolmas kerros oppii pienempiä yksityiskohtia ja tekstuureita. Piirteet syötetään vielä viimeiseen kerrokseen, jossa hoidetaan esimerkiksi kuvan luokittelu. (Di ym. 2018, 9.)

3 Neuroverkot

3.1 Neuroverkkojen perusteet

Jotkut ensimmäisistä koneoppimisalgoritmeista pyrkivät mallintamaan biologista oppimista, eli miten aivot oppivat uutta tietoa. Tämän takia syväoppimista on kutsuttu myös keinotekoiseksi neuroverkoksi. Joillain neuroverkoilla on pyritty ymmärtämään aivojen toimintaa, mutta niitä ei yleensä ole suunniteltu realistisiksi malleiksi aivojen toiminnasta. (Goodfellow ym. 2016, 13.)

Neuroverkot ovat joutuneet kilpailemaan tekoälytutkimuksissa symbolisen mallintamisen kanssa, eli esimerkiksi sääntöjärjestelmien käytön kanssa. Sääntöjärjestelmällä voidaan esimerkiksi selvittää, onko kuvassa olevalla ihmisellä suupielet ylöspäin ja sen perusteella päätellä hymyileekö henkilö. Neuroverkko puolestaan oppii esimerkeistä, joita sille syötetään. (Honkela n.d.)

Neuroverkkojen oppiminen voi olla joko ohjattua tai ei ohjattua oppimista. Oppimiskyky ja oikean maailman ilmiöiden mallintaminen ovat johtaneet nopeaan leviämiseen erilaisten tehtävien parissa. (Honkela n.d.)

3.2 Neuroverkkojen rakenne

Neuroverkkojen rakenne koostuu keinotekoisista hermosoluista ja niitä yhdistävät keinotekoiset synapsit. Neuroverkon oppiminen tapahtuu keinotekoisten hermosolujen välisten liitosten voimakkuuksien muutoksina. Vaikka neuroverkkoja ei suoraan ohjelmoida vastaamaan syötteisiin jollain tavalla, kuitenkin menetelmä, jolla liitosten voimakkuudet muuttuvat, on ohjelmoitu. (Honkela n.d.)

Tyypillisesti neuroverkkojen rakenteeseen kuuluu kolmea erilaista kerrosta keinotekoisia neuroneita: syötekerros, piilokerros ja ulostulokerros. Kuviossa 4 nähdään esimerkki neuroverkon kerroksien kytkeytymisestä. Syötekerros on ensimmäinen kerros ja siihen tuodaan kaikki data, jota halutaan käyttää, kuten pikselien arvot kuvissa. Jokainen syötekerroksen neuroni kytkeytyy seuraavan kerroksen, eli piilokerroksen neuroneihin. Datan arvojen siirtyessä piilokerrokseen, ne kerrotaan painoarvoilla ja summataan piilokerroksen sisällä. Tätä kutsutaan painotetuksi summaksi, jonka tulos voidaan laskea yhtälöllä 5. Painotettu summa puolestaan siirtyy ulostulokerrokselle aktivaatiofunktion kautta. Ulostulokerroksessa on usein ainoastaan yksi neuroni, joka ilmoittaa ulostulon arvon. (Sivanadam, Sumathi & Deepa 2006, 368.)

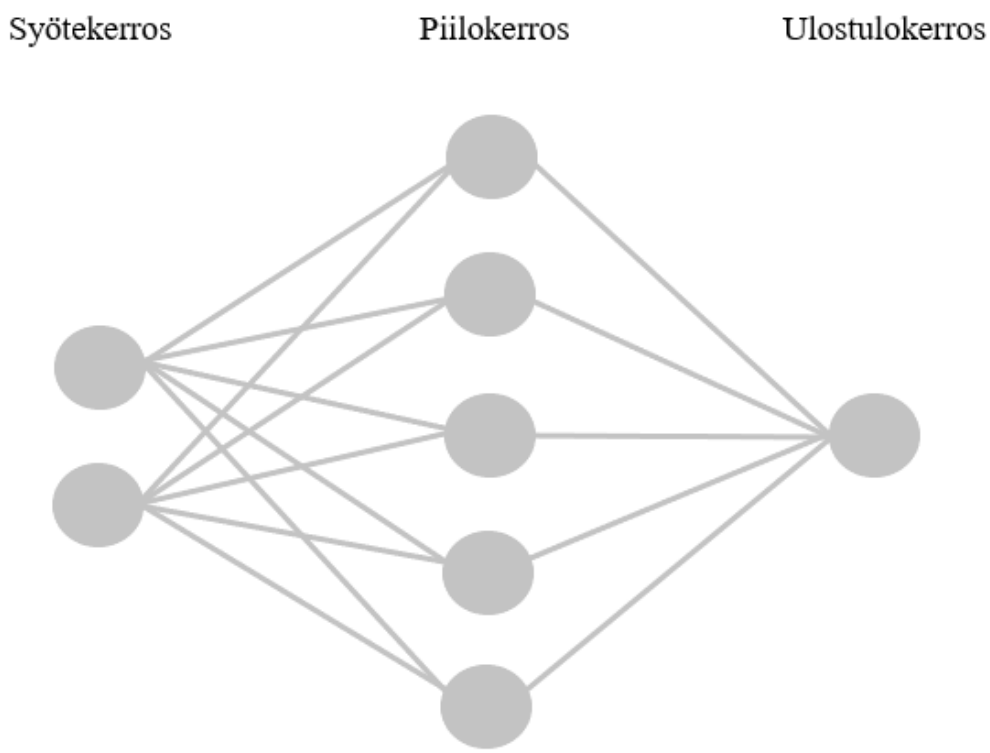
$$\sum_{i=1}^{n} x_i w_i + b \qquad (5)$$

missä b = vakiotermi
x = neuronille tuleva arvo
w = neuronin painoarvo

Esimerkki painotetun summan laskemisesta keksityillä arvoilla on seuraava:

$$0{,}2 * 0{,}5 + 0{,}4 * 0{,}3 + 0{,}7 * 0{,}9 + 0 = 0{,}85$$

Esimerkistä saatu tulos viedään eteenpäin aktivaatiofunktiolle, esimerkiksi ReLU -aktivaatiolle. Koska sisääntulo on suurempi kuin 0, ulostulon arvo on sama kuin sisääntulo, eli tässä tapauksessa 0,85.



Kuvio 4. Esimerkki neuroverkon kerroksista

3.3 Yleisimmät aktivaatiofunktiot

ReLU-funktio

Suurin osa nykypäivän neuroverkoista käyttää aktivointifunktiona rectified linear unit, eli ReLU -funktiota (Goodfellow ym. 2016, 15). ReLU:n sisääntulon ollessa 0 tai pienempi, sen ulostulo on myös 0. Sisääntulon ollessa suurempi kuin 0, sen ulostulo on sama kuin sisääntulo. ReLU -funktio on huomattavasti nopeampi kuin sigmoid -funktio ja suorittaa yksinkertaisia laskelmia. (Sharma 2019.)

Sigmoid-funktio

Sigmoid -funktio muuttaa kaikki ulostulot 0 ja 1 välille. Tästä syystä sitä käytetään erityisen paljon malleissa, joissa ennustetaan todennäköisyyksiä. Huonona puolena sigmoid -funktiolla on datan katoaminen suuresta datan pakkaamisesta johtuen.

Mitä syvempi neuroverkko on kyseessä, sitä enemmän menetetään alkuperäisestä datasta. (Sharma 2019.) Sigmoid -funktion tulos on mahdollista laskea yhtälöllä 6.

$$s(x)\frac{1}{1+e^{-x}} \quad (6)$$

missä x = sisääntulon arvo

Softmax-funktio

Softmax -funktio muuttaa kaikki ulostulot 0 ja 1 välille kuten myös sigmoid -funktio tekee. Sen lisäksi softmax -funktio jakaa kaikki ulostulot niin, että niiden summaksi tulee 1. Softmax -funktion ulostulo koostuu jokaisen luokan todennäköisyydestä ja sitä käytetään useiden luokkien luokitteluun, kun taas sigmoid -funktiota käytetään binääriseen luokitteluun. (Sharma 2019.) Softmax -funktion tulos on mahdollista laskea yhtälöllä 7.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad (7)$$

missä x = sisääntulon arvo

4 Konvoluutioverkot

4.1 Konvoluutioverkkojen perusteet

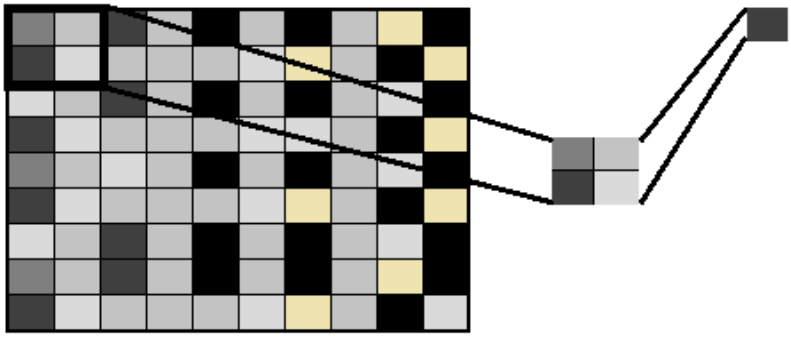
Konvoluutioverkot (myös konvoluutioneuroverkot, CNN) ovat olleet tärkeässä asemassa syväoppimisen historiassa ja ovat ensimmäisten syväoppimismallien joukossa, jotka ovat suoriutuneet hyvin jo ennen kuin syviä malleja pidettiin käyttökelpoisina. Ne ovat myös ensimmäisten neuroverkkojen joukossa, joilla toteutettiin tärkeitä kaupallisia sovelluksia. Esimerkiksi 1990 -luvulla AT&T toteutti neuroverkkosovelluksen, jolla pystyttiin tunnistamaan sekkejä ja 90 -luvun loppuun mennessä sitä käytettiin tunnistamaan n. 10% kaikista yhdysvaltojen sekeistä.

Konvoluutioverkoilla on myös voitettu lukuisia kilpailuja, kuten ImageNet hahmontunnistushaaste. (Goodfellow ym. 2016, 365-366.)

Konvoluutioverkot ovat erikoistuneet taulukkotyylisen datan prosessointiin. Taulukkotyylistä dataa on esimerkiksi kuvadata joka on kaksiulotteinen taulukko pikseleitä tai aikasarjadata joka on yksiulotteinen taulukko, jossa on otteita tietyin aikavälein. Konvoluutioverkon nimi tulee matemaattisesta konvoluutio -operaatiosta. Konvoluutioverkossa käytetään vähintään yhdellä kerroksella konvoluutioita matriisien kertomisen sijasta. (Goodfellow ym. 2016, 326.)

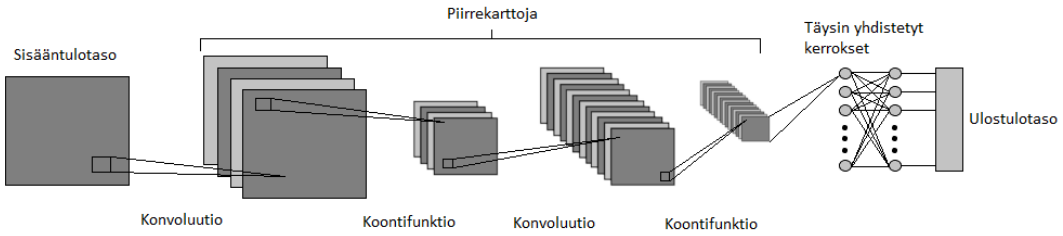
Kuvia otetaan usein useissa eri perspektiiveissä ja tavarat ovat yleisesti ottaen erikokoisia eri kuvissa. Yksi tyypillisimmistä tavoista parantaa konvoluutioverkkojen suorituskykyä ja ratkoa edellä mainittuja ongelmia on opetuskuvien esikäsittely. Kuvien esikäsittelyyn voi kuulua esimerkiksi kuvien peilaaminen, leikkeiden ottaminen, kokomuutokset, rotaatiot ja värimuunnokset. (Di ym. 2018, 97-98.)

Konvoluutioverkon tyypillinen kerros koostuu kolmesta vaiheesta. Ensimmäisessä vaiheessa kerros suorittaa useita konvoluutioita rinnakkain, joista tulee joukko lineaarisia aktivaatioita. Toisessa vaiheessa jokainen lineaarinen aktivaatio menee ei-lineaarisen aktivaatiofunktion läpi, kuten esimerkiksi ReLU -funktion läpi. Tämän avulla poistetaan mallin lineaarisuutta. Kolmannessa vaiheessa käytetään koontifunktiota, jolla vähennetään muuttujien määrää. Esimerkiksi MaxPooling -funktiolla, eli koontifunktiolla, voidaan ottaa 2x2 kokoiselta pikselialueelta näyte ja sen alueen suurin pikseliarvo edustaa sitä seuraavassa tasossa. Tämä operaatio toistetaan kunnes kuvan kaikki pikselit on käyty läpi ja seuraavalle tasolle muodostuu pienempi kuva. 2x2 -kokoisella suodattimella ja kahden pikselin suuruisella asekeleella kuvasta tulee puolet pienempi. Askel kertoo kuinka monta pikseliä siirrytään sivulle jokaisen näytteen jälkeen. (Goodfellow ym. 2016, 335-336.) Kuviossa 5 näkyy esimerkki koontifunktion toiminnasta.



Kuvio 5. Koontifunktion toiminnan esimerkki

Viimeisenä kerroksena konvoluutioverkoissa on täysin yhdistetty kerros, jonka jokainen neuroni on yhdistetty edellisen kerroksen kaikkiin neuroneihin. Kahta täysin yhdistettyä kerrosta käytetään usein kahtena viimeisenä kerroksena ennen softmax tai sigmoid -funktiota, joilla luokittelu tapahtuu. (Goodfellow ym. 2016, 335-336.) Kuviosta 6 nähdään yksinkertainen esimerkki yhdenlaisesta konvoluutioverkon rakenteesta.



Kuvio 6. Esimerkki yksinkertaisesta konvoluutioverkon rakenteesta.

4.2 Konvoluutioverkkojen käytännön sovelluksia

New Yorkin yliopistossa toteutettiin sovellus, joka löytää, luokittelee ja tunnistaa tavaroita kuvista konvoluutioverkon avulla. Kyseisellä neuroverkolla pystytään tunnistamaan esimerkiksi konserttikuvasta esiintyjät, lamput, mikrofonit kitarat ja muut soittimet. Tarkkuudeksi neuroverkolla tuli noin 66%, jossa on vielä parannettavaa. (Sermanet, Eigen, Zhang, Mathieu, Fergus & LeCun 2014.)

Jyväskylän ammattikorkeakoulussa on toteutettu erilaisia kuvantunnistusdemoja konvoluutioverkoilla, kuten kissan ja koiran tunnistus kuvasta tai iän ja sukupuolen tunnistaminen kasvojen perusteella. Myös keuhkokuumetta on onnistuttu tunnistamaan 99% sensitiivisyydellä ja 91% sisäisellä tarkkuudella röntgen kuvista. (Tekoälyn ja data-analytiikan toteutukset 2019.)

Stanfordin yliopistossa tutkijaryhmä toteutti sovelluksen, joka tunnistaa pahanlaatuisia luomia hyvänlaatuisista luomista. Dermatologit tunnistivat luomet noin 66% tarkkuudella, mutta neuroverkolla päästiin 72% tarkkuuteen. Sovelluksessa käytettiin esiopetettua Inception V3 -mallia. (Esteva, Kurpel, Novoa, Ko, Swetter, Blau & Thrun 2017.)

Vuonna 2012 konvoluutioverkko AlexNet saavutti suuressa kuvantunnistushaasteessa ykkössijan 85% -tarkkuudella, joka oli 11% parempi kuin toinen sija. Vuoteen 2015 mennessä oli jo useita konvoluutioverkkoja, jotka ylittivät 95% tarkkuuden. (Di ym. 2018, 28.)

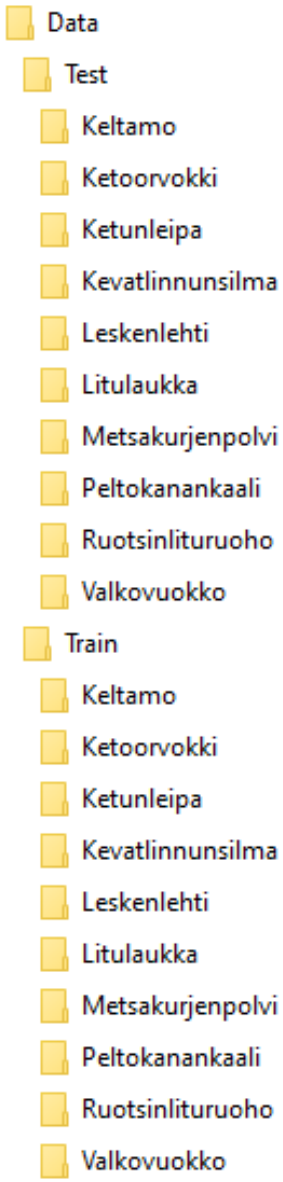
5 Toteutus

Kehitysympäristöksi valittiin avoimen lähdekoodin Anaconda-ympäristö. Python-ohjelmointi ja neuroverkon opetus toteutettiin jupyter notebookilla, joka on avoimen lähdekoodin sovellus, sekä osa Anaconda-jakelua. Keras valittiin ylemmän tason oh-

jelmointirajapinnaksi. Anaconda valittiin mm. helpon kehitysympäristöhallinnan, kirjastojen hallinnan ja monipuolisten kehityssovellusten takia. Python valittiin suuren suosion takia koneoppimisyhteisön keskuudessa. Suosion takia saatavilla on myös erittäin monipuoliset neuroverkkokirjastot ja tietoa on myös saatavilla erittäin laajasti. Keras puolestaan valittiin omien kokemusten, koneoppimsyhteisön suositusten, modulaarisuuden ja helppokäyttöisyyden perusteella. Toteutuksessa käytettyjen kirjastojen ja ohjelmistojen versiot löytyvät liitteestä 3.

5.1 Esikäsittely

Kuvia varten luotiin hakemistorakenne, josta Keras löytää kuvat suoraan. Rakenteeseen kuuluu Data -kansio, jonka alla on Train ja Test -kansiot. Train ja Test -kansioiden alla on jokaiselle kasvilajille omat kansionsa ja kuvat löytyvät niiden alta kuten näkyy kuviosta 7. Kuvat tulivat valmiiksi lajiteltuina, joten ne voitiin siirtää suoraan kasvilajeja vastaaviin kansioihin.



Kuvio 7. Data -kansion hakemistorakenne

Kuville tehtiin kuvankäsittelyä scriptissä ennen kuin ne syötettiin neuroverkon opetukseen. Aluksi kuvien pikselien arvot skaalattiin väliltä 0-255 välille 0-1, koska 0-255 on liian suuri skaala neuroverkon opettamiseen ja 0-1 -väliä käytetään perinteisesti normalisoinnissa. Kuvia zoomattiin satunnaisesti 0 - 20%, pyöritettiin satunnaisesti 0 - 20° sekä peilattiin satunnaisesti pystysuunnassa. Nämä esikäsittelyn toimenpiteet kasvattavat keinotekoisesti opetusdatan määrää, koska lähes jokainen kerta, kun sama kuva tulee opetukseen, se on hieman erilainen. Kuvan esikäsittelyfunktiot näkyvät kuviossa 8.

```
train_datagen = ImageDataGenerator(
    rescale=1. / 255,                   #Scaling RGB values from "0-255" to "0-1"
    zoom_range=0.2,                     #Random zoom between 0 and 0.2
    rotation_range=20,                  #Random rotation of 20 degrees
    horizontal_flip=True)               #Random horizontal flip
```

Kuvio 8. Kuvien esikäsittelyfunktion konfigurointi.

5.2 Koulutus

Alussa ladattiin pohjaksi imagenet -datasetillä esikoulutetun mallin painoarvot, joista jätettiin pois ylin täysin yhdistetty kerros omaa koulutusta varten. Esikoulutetuksi malliksi valittiin Xception -malli, jonka arkkitehtuurilla on saatu hyviä tuloksia hahmontunnistuksessa. Parempiakin tuloksia on saavutettu, mutta Xception on sopiva yhdistelmä suorituskykyä ja koulutusnopeutta. Latauksen jälkeen kutsuttiin tarvittavat kirjastot ja määriteltiin muuttujia. Määriteltäviä muuttujia olivat kuvan koko, opetuskuvien hakemistopolku, testikuvien hakemistopolku, opetuskuvien määrä, testikuvien määrä, epookkien määrä, opetusjoukon koko ja tallennuspisteen nimi. Nämä kaikki löytyvät kuviosta 9.

```
from keras.applications.xception import Xception
conv_base = Xception(weights='imagenet', include_top=False, input_shape=(299,299,3))
```

```
from keras.preprocessing.image import ImageDataGenerator          #For image preprocessing
from keras.models import Sequential                                #For sequential model
from keras.layers import Conv2D, MaxPooling2D                      #For conv layers
from keras.layers import Activation, Dropout, Flatten, Dense       #For conv layers
from keras import backend as K                                     #Bringing tensorflow backend
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint     #LR reducing on plateau and saving best model
from keras.optimizers import Adam                                  #Bringing optimizer
from mlxtend.plotting import plot_confusion_matrix                 #For plotting Confusion matrix
from sklearn.metrics import confusion_matrix                       #Bringing confusion matrix
import numpy as np                                                 #Bringing numpy
import keras_metrics                                               #Bringing Custom metrics to measure performance
```

```
img_width, img_height = 299, 299    #Defining picture height and width
train_data_dir = 'Data/Train'       #Training data directory
validation_data_dir = 'Data/Test'   #Validation data directory
nb_train_samples = 1254             #Number of training samples
nb_validation_samples = 60          #Number of validation samples
epochs = 100                        #Number of epochs
batch_size = 9                      #Batch size
checkpointfile = 'weights.{acc:.2f}-{precision:.2f}-{recall:.2f}.h5' #Checkpoint filename
```

Kuvio 9. Kirjastot ja muuttujien määritys.

Muuttujien jälkeen tarkistettiin inputin muoto, koska sequential -mallin täytyy tietää, missä muodossa data tulee. Muodon tarkistuksen jälkeen määritettiin sequential -malli ja syötettiin esiopetetun mallin painoarvot työssä olevaan malliin. Tätä seuraa arkkitehtuuri, johon kuuluu seuraavanlaiset kerrokset. Flatten -kerros, joka litistää piirrekartan yhdeksi pylvääksi. Pari dense -kerrosta, jotka ovat täysin yhdistettyjä kerroksia. ReLU aktivaatio -kerros, jolla poistetaan neuroverkon lineaarisuutta. Dropout -kerros, jolla ehkäistään mallin ylisovittumista. Lopuksi vielä täysin yhdistetty softmax aktivaatio -kerros, jolla muodostetaan todennäköisyydet jokaiselle kasville. Nämä näkyvät kuviossa 10 ja neuroverkon arkkitehtuurin rakenne sisään- ja ulostulojen parametrien määrän kanssa on nähtävissä kuviossa 12.

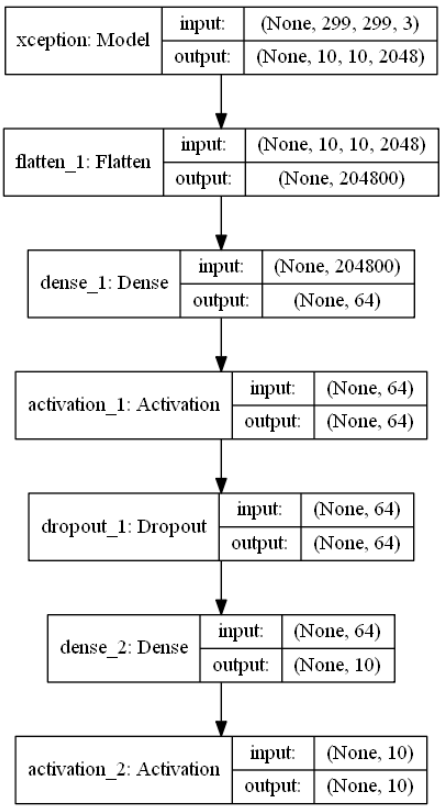
```
#Sequental model needs to know the input_shape.

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)


#Loading pretrained model to our network

model = Sequential()
model.add(conv_base)
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))
```

Kuvio 10. Sisääntulon muodon määritys ja neuroverkon kerrokset.

Arkkitehtuurin määrityksen jälkeen malli konfiguroitiin opetusta varten. Loss -funktiona käytettiin categorical_crossentropya, optimisaattorina toimi adam ja suorituksen mittareiksi määriteltiin ulkoinen tarkkuus, tarkkuus ja herkkyys, jotka näkyvät kuviossa 11.

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.0001),
              metrics=['accuracy',
                       keras_metrics.categorical_precision(),
                       keras_metrics.categorical_recall()])
```

Kuvio 11. Mallin konfigurointi opetusta varten.



Kuvio 12. Neuroverkon arkkitehtuuri parametrien kanssa.

Mallin konfiguroinnin jälkeen suoritettiin esikäsittelyt -osiossa läpi käyty kuvankäsittely, jolla saatiin mm. keinotekoisesti paisutettua opetusdatan määrää. Kuvankäsittelyn jälkeen kuvien data tuotiin opetus ja testidatan generaattorille, jolla generointiin muokattuja joukkoja opetus- ja testikuvista. Testikuville ei tehty kuvamanipulaatiota, lukuun ottamatta pikseliarvojen skaalausta neuroverkkoa varten.

Generaattoreita varten piti määritellä hakemistopolku, jossa kuvat sijaitsevat, kuvan koko, kuvajoukkojen koko ja luokkien määrä, jotka näkyvät kuviossa 13. Luokkien määrän määrittelyksi generaattorille riittää joko binary, jos on vain kaksi luokkaa tai categorical, jos luokkia on yli kaksi.

```
#Generates batches of image data for training

train_generator = train_datagen.flow_from_directory(
    train_data_dir,                                  #Training data directory
    target_size=(img_width, img_height),             #Image size
    batch_size=batch_size,                           #Amount of training samples in one pass
    class_mode='categorical')                        #Amount of classes, binary if 2 classes, else categorical


validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,                             #Training data directory
    target_size=(img_width, img_height),             #Image size
    batch_size=batch_size,                           #Amount of training samples in one pass
    class_mode='categorical')                        #Amount of classes, binary if 2 classes, else categorical
```

Kuvio 13. Opetus- ja testidatageneraattorien konfiguraatio.

Generaattorien jälkeen konfiguroitiin opetusnopeuden alennusfunktio, joka alentaa opetusnopeutta, mikäli edistystä ei tapahdu tarkkailtavalla mittarilla. Pienempi opetusnopeus aiheuttaa painoarvoissa pienempiä muutoksia, joka taas mahdollistaa tarkkuuden parantumisen. Alusta asti pientä opetusnopeutta käyttäessä opetus tapahtuu hitaammin, eikä haluttuun tulokseen välttämättä päästä realistisessa ajassa. Määriteltäviä muuttujia funktiossa ovat monitor, factor, patience, cooldown, min_lr ja verbose, jotka näkyvät kuviossa 14. Monitor -muuttujaan tulee mitä suorituksen mittaria tarkkaillaan, factor -muuttujaan tulee kerroin, jolla muutetaan opetusnopeutta, patience -muuttujaan tulee epookkien määrä ilman parannusta ennen kuin

opetusnopeutta alennetaan, min_lr -muuttujaan tulee opetusnopeuden alaraja ja viimeiseksi verbose -muuttujalla määritetään päivitysviestit päälle tai pois päältä.

```
#Lowering learning rate if no progress is made.

reduce_learning_rate = ReduceLROnPlateau(monitor='loss',
                                         factor=0.1,
                                         patience=2,
                                         cooldown=2,
                                         min_lr=0.00001,
                                         verbose=1)
```

Kuvio 14. Opetusnopeuden alennusfunktion konfiguraatio.

Opetusnopeuden alennusfunktion jälkeen määriteltiin tallennuspiste, joka tallentaa aina parhaan mallin painoarvot erilliseen tiedostoon, josta ne voidaan ladata ennustusten tekemistä varten. Määriteltäviä muuttujia checkpoint -funktioon ovat checkpointfile, monitor, verbose, save_best_only, save_weights_only ja mode. Checkpointfile -muuttujaan tulee tallennuspisteen tiedostonimi, joka määriteltiin koodin alussa muiden muuttujien kanssa. Monitor -muuttujalla määritellään mitä suorituksen mittaria tarkkaillaan, verbose -muuttujalla määritetään päivitysviestit päälle tai pois päältä, save_best_only -muuttujalla määritetään, kirjoitetaanko edellisten painoarvojen päälle parannuksen tapahtuessa ja mode muuttujalla määritetään, halutaanko tarkkailtavalta suorituksen mittarilta mahdollisimman suurta vai pientä arvoa. Checkpointin jälkeen määriteltiin callbacks, joka on joukko aikaisemmin määriteltyjä funktioita, jotka kutsutaan kesken opetuksen. Kuviosta 15 löytyy checkpoint ja callbacksin määritykset.

```
#Defining checkpoint that saves the best model or models
#weights based on the value that we are monitoring.

checkpoint = ModelCheckpoint(checkpointfile,
                             monitor='acc',
                             verbose=1,
                             save_best_only=True,
                             save_weights_only=True,
                             mode='max')


#Defining callbacks which includes checkpoint and
#reducing learning rate on plateau.

callbacks = [checkpoint, reduce_learning_rate]
```

Kuvio 15. Checkpoint -funktion ja callbacksin konfiguraatio.

Lopussa toteutettiin mallin opetus fit_generator -funktiolla. Generaattorin muuttujia ovat generator, steps_per_epoch, epochs, validation_data, validation_steps ja callbacks. Generator -muuttujan tilalle tulee aikaisemmin määritetty opetusdatan generaattori, joka on tässä tapauksessa train_generator. Steps_per_epoch -muuttujalla määritetään askelten määrä yhdessä epookissa, joka tässä tapauksessa oli opetuskuvien määrä jaettuna kuvajoukkojen koolla. Koska opetuskuvien määrä ei ole jaollinen kuvajoukkojen koolla, on viimeisen kuvajoukon koko ainoastaan kolme, normaalin yhdeksän sijasta. Epochs -muuttujaan tulee epookkien määrä, eli kuinka monta kertaa opetusdata käydään kokonaisuudessaan läpi. Validation_data -muuttujaan tulee validointidatan lähde, joka on tässä tapauksessa validation_generator. Validation_steps -muuttuja toimii vastaavanlaisesti kuin steps_per_epoch, paitsi validointikuvien määrällä ja callbacksilla määritetään aikaisemmin määritellyt callbacks -funktiot. Kuviossa 16 näkyy fit_generatorin konfiguraatio.

```
#Training CNN

model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size,
    callbacks=callbacks)
```

Kuvio 16. Fit_generator -funktion konfiguraatio.

Neuroverkkoa opetettiin sadan epookin verran ja epookit kestivät noin 57 sekuntia per epookki. Kokonaisuudessaan siihen kului noin 1,5 tuntia. Parhaat painoarvot mallille löytyivät kuitenkin jo ennen sadannetta epookkia. Kuten kuviosta 17 nähdään, validointitarkkuus (val_acc) alkoi matalalta, mutta nousi jo kolmen epookin jälkeen yli 50% -tarkkuuteen. Myös opetuskuvien tunnistustarkkuudessa (acc) nähdään huomattavaa parannusta ensimmäisen kolmen epookin aikana.

```
Found 1254 images belonging to 10 classes.
Found 60 images belonging to 10 classes.
Epoch 1/100
139/139 [==============================] - 63s 450ms/step - loss: 2.3056 - acc: 0.1511 - precision: 0.0000e+00 - recall: 0.0000
e+00 - val_loss: 1.8895 - val_acc: 0.2963 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00

Epoch 00001: acc improved from -inf to 0.15108, saving model to weights.0.15-0.00-0.00.h5
Epoch 2/100
139/139 [==============================] - 57s 409ms/step - loss: 2.0472 - acc: 0.2222 - precision: 0.0000e+00 - recall: 0.0000
e+00 - val_loss: 1.5039 - val_acc: 0.4118 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00

Epoch 00002: acc improved from 0.15108 to 0.22329, saving model to weights.0.22-0.00-0.00.h5
Epoch 3/100
139/139 [==============================] - 56s 402ms/step - loss: 1.8853 - acc: 0.3046 - precision: 0.0000e+00 - recall: 0.0000
e+00 - val_loss: 1.2510 - val_acc: 0.5882 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
```

Kuvio 17. Neuroverkon koulutustilanne.

5.3 Ennustusten tekeminen

Kuvien tunnistamista varten tehtiin erillinen python scripti, joka piti sisällään opetuksessa käytetyn arkkitehtuurin ja sen päälle ladattiin koulutetun mallin painoarvot.

Painoarvojen latauksen jälkeen määriteltiin kasvit -joukko, johon listattiin tunnistettavien kasvilajien nimet aakkosjärjestyksessä, kuten nähdään kuviossa 18.

```
#Classes that you are predicting in alphapetical order.

kasvit =['Keltamo', 'Ketoorvokki', 'Ketunleipa', 'Kevatlinnunsilma',
         'Leskenlehti', 'Litulaukka', 'Metsakurjenpolvi', 'Peltokanankaali',
         'RuotsinliturUoho', 'Valkovuokko']
```

Kuvio 18. Kasvit -listan määritys.

Syötteenä scriptille piti antaa tunnistettavan kuvan tiedostonimi, jota scripti kysyi ajettaessa. Tämän jälkeen kuva skaalattiin, tulostettiin ruudulle, muokattiin neuroverkolle sopivaksi ja syötettiin kerasin predict_classes -funktiolle, joka käytti opetettua mallia tunnistamaan mikä kasvilaji on kyseessä. Lopuksi ruudulle tulostui numerona kasvin paikka kasvit -listalta ja kasvin nimi. Kuviossa 19 näkyy tunnistamiseen käytetty koodi.

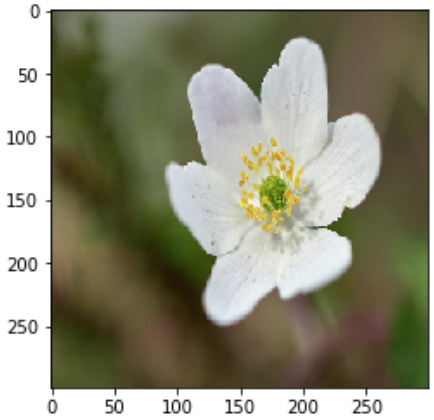
```
text = input("Tunnistettavan kuvan nimi (mukaanlukien tiedostopääte, esim. kukka.jpg): ")

img = image.load_img(text, target_size = (299, 299))
plt.imshow(img)
img = image.img_to_array(img)
img = np.expand_dims(img, axis = 0)
img /= 255.

prediction = model.predict_classes(img)
print(prediction)

print(kasvit[int(prediction)])
```

Kuvio 19. Kuvan syöttö ennustamista varten ja ennustamisen tekeminen.

6 Tulokset

Tuloksena JAMKin servereille toteutettiin salasanasuojattu demo asiakasta varten. Asiakasyritys testasi demoa omilla kuvillaan ja oli tyytyväinen tulokseen. Lopullinen tarkkuus parhaassa mallissa oli opetuskuvien tunnistamisessa 99,76% ja testikuvien tunnistamisessa 100%. Testikuvien tunnistustarkkuutta ei kuitenkaan voida pitää realistisena huomattavasti suuremmilla kuvamäärillä, kuten vaikka tuhannella kuvalla. Kyse on kuvista, jotka neuroverkko on nähnyt lukuisia kertoja verrattuna kuviin, joita neuroverkko ei ole koskaan nähnyt. Tästä syystä testikuvilla ei ole realistista päästä parempaan tarkkuuteen kuin opetuskuvilla. Tämä olisi voitu testata käytännössä, jos käytössä olisi ollut enemmän testikuvia. Kuviossa 20 nähdään esimerkki kasvilajin tunnistuksesta.

```
Python 3.6.8 |Anaconda, Inc.| (default, Dec 30 2018, 18:50:55) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Toni/Kasvilajit/Ennustus.py', wdir='C:/Users/Toni/Kasvilajit')
Using TensorFlow backend.

Tunnistettavan kuvan nimi (mukaanlukien tiedostopääte, esim. kukka.jpg): kukka.jpg
[9]
Valkovuokko
```



Kuvio 20. Valkovuokon tunnistaminen kuvasta.

Koulutettu malli pystyi tunnistamaan jokaisen kuudestakymmenestä testikuvasta. Testikuvien lisäksi se tunnisti myös kuvia, joilla testattiin ennustus scriptiä. Koulutus ei onnistunut tyhjällä mallilla, vaikka siihen kokeiltiin erilaisia arkkitehtuureita ja muuttujia. Opetuskuvien määrän ollessa erittäin vähäinen, alun perinkään ei uskottu neuroverkon oppivan tunnistamaan kasvilajeja, kun sitä opetetaan tyhjästä mallista.

Opinnäytetyö luo hyvän pohjan jatkokehitykselle kasvilajien tunnistuksen parissa ja saatua tietoa voidaan hyödyntää muidenkin asioiden tunnistuksessa. Asiakasyritykselle toimitettiin koulutuksessa ja tunnistuksessa käytetyt koodit heidän omia projektejaan varten. Myös erillinen ohjeistus mallin koulutuksen toteuttamisesta Jupyter Notebookilla ja kasvilajien tunnistuksen tekemisestä toimitettiin asiakasyritykselle. Neuroverkon koulutuksen koodi on kokonaisuudessaan nähtävillä liitteessä 1 ja kasvilajien tunnistuksessa käytetty koodi näkyy liitteessä 2.

Mallilla ei yritetty tunnistaa muita lajeja, joita sille ei ole opetettu. Jos kuitenkin olisi yritetty syöttää kuva jostain muusta kuin opetetuista kasvilajeista, malli olisi valinnut jonkun niistä kymmenestä kasvilajista, jonka piirteet ovat lähimpänä syötettyä kuvaa. Testikuvissa oli ainoastaan yksi kasvilaji kuvaa kohden, mutta testattavassa kuvassa saattoi olla useampi kappale samaa kasvilajia. Jatkossa on mahdollista myös selvittää mitä tapahtuu, jos kaksi eri kasvilajia opetetuista kasvilajeista on samassa kuvassa.

7 Pohdinta

Työssä päästiin tarkkuustavoitteeseen, joka oli määritelty 95% -tarkkuuteen. Vaikka tavoitteeseen päästiin pelkästään siirretyn oppimisen avulla, on mallin opetus realistista toteuttaa myös täysin tyhjistä, kunhan opetusdataa on enemmän. Jos samaan tavoitteeseen halutaan päästä ilman siirrettyä oppimista, tulisi opetuskuvien määrän oltava huomattavasti suurempi kuin noin sata kuvaa per kasvilaji. Konvoluutioverkkojen suorituskyky on verrannollinen opetuskuvien määrään. Opetuskuvien määrä kymmenelle kasvilajille voisi olla tuhansia kuvia per kasvilaji, niin voitaisiin nähdä jo jonkinlaisia tuloksia tyhjän mallin opetuksessa.

Tuloksia voidaan pitää luotettavina, mikäli suorituksen mittareiden toiminta on ymmärretty oikein. Vaikka niiden toiminta olisi ymmärretty väärin ja ne mittaisivatkin eri asioita kuin luultu, testeissä neuroverkko silti tunnisti kaikki kuvat jotka sille syötettiin. Testit ja asiakkaan palaute myös tukee suurta tarkkuutta. Työssä saadut tarkkuudet ovat Keras -kirjaston tulostamia mallin koulutuksen yhteydessä.

Neuroverkkoa voi myös käyttää pohjana muiden asioiden tunnistamisessa, kuten vaikka sienilajien tai autolajien tunnistamisessa. Kuitenkin jos muita asioita halutaan tunnistaa hyvällä tarkkuudella, joutuu todennäköisesti tekemään muutoksia neuroverkon arkkitehtuuriin ja parametreihin. Konkreettisina jatkokehitysmahdollisuuksina voisi olla mobiili- tai web-sovelluksen kehitys useammille kasvilajeille.

Neuroverkko skaalaantuu suuremmillekin kasvilajimäärille lisäämällä opetus- ja testikuvia hakemistorakenteeseen uusiin kansioihin, sekä muuttamalla dense -tason ulostulojen määrän vastaamaan kasvilajien määrää ja lisäämällä uudet kasvilajit ennustuksen tekemisessä kasvit -joukkoon. Myös opetuskuvien määrää per kasvilaji olisi hyvä lisätä, mikäli aikoo lisätä kasvilajeja opetukseen. Suuremmat kasvilajimäärät voivat vaatia muutoksia parametrointiin sekä mahdollisesti neuroverkkoarkkitehtuuriin.

Koneoppiminen on ollut paljon esillä viimeaikoina eri yhteyksissä. Itsestään ajavat autot ovat pikkuhiljaa yleistymässä ja verkossa löytyy kokoajan enemmän neuroverkkosovelluksia pelien resoluutioiden skaalauksesta asuntomarkkinoiden ennustamiseen ja lääketieteellisiin tarkoituksiin. Kuvantunnistuksissakin on jo saavutettu parempia tuloksia neuroverkkojen avulla kuin asiantuntijoiden avulla.

Teknologiateollisuudessakin koneoppiminen on ollut jo pidemmän aikaa käytössä mm. robotiikan merkeissä. Muidenkin alojen yritykset ovat alkaneet kiinnostumaan koneoppimisen sovelluksista ja sitä kautta löytyi opinnäytetyöllekin aihe kasvilajien tunnistuksen parista. Samankaltaisia sovelluksia voisi hyödyntää myös muissa tunnistustehtävissä, kuten sienien tai vaikka autojen tunnistuksessa. Työssä tehtyä

neuroverkkoa voisi myös kokeilla opettaa muilla asioilla kuin kasveilla ja katsoa millaiseen tarkkuuteen päästäisiin.

Tekoälyn yleistyessä tulee kuitenkin kiinnittää erityisesti huomiota opetusdataan ja data-analytiikkaan. Huonosta opetusdatasta johtuen on ollut jo puhetta ns. rasistisista tekoälyistä, jotka luokittelevat ihmisiä väärin perustein. Esimerkiksi jos opetuskuvajoukon kaikki jäätelöstä pitävät ihmiset ovat vaaleaihoisia, niin neuroverkko erittäin todennäköisesti liittää jäätelöstä pitämisen vaaleaan ihon väriin. Tästä johtuen huomion kiinnittäminen datan laatuun on erityisen tärkeää, kuitenkaan esikäsittelyä unohtamatta.

Lähteet

Di, W., Bhardwaj, A. & Wei, J. 2018. Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling. Birmingham: Packt Publishing.

Goodfellow, I., Bengio, Y. & Courville, A. 2016. Deep Learning. Cambridge: The MIT Press.

Esteva, A., Kurpel, B., Novoa, R., Ko, J., Swetter, S., Blau, H. & Thrun, S. 2017. Skin cancer classification with deep learning. Macmillan Publishers Limited. Viitattu 8.1.2020. https://cs.stanford.edu/people/esteva/nature.

Elements of AI: Mitä tekoäly on?. N.d. Helsingin yliopisto. Viitattu 8.9.2019. https://course.elementsofai.com.

Honkela, T. N.d. Neuroverkot: johdatus moderniin tekoälyyn. Viitattu 5.1.2020. http://users.ics.aalto.fi/tho/stes/step96/honkela2.

Russell, S. & Norvig, P. 2016. Artificial Intelligence: A Modern Approach. Harlow: Pearson Education Limited.

Sarkar, D. 2018. A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. Viitattu 8.1.2020. https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. & LeCun, Y. 2014. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. Viitattu 08.01.2020. https://arxiv.org/pdf/1312.6229.pdf.

Sharma, H. 2019. Activation Functions : Sigmoid, ReLU, Leaky ReLU and softmax basics for Neural Networks and Deep learning. Viitattu 28.12.2019. https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e.

Shash, T. 2017. About Train, Validation and Test Sets in Machine Learning. Viitattu 10.01.2020. https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

Sivanadam, S., Sumathi, S. & Deepa S. 2006. Introduction to Neural Networks using MATLAB 6.0. New Delhi. Tata McGraw-Hill Publishing Company Limited.

Tekoälyn ja data-analytiikan toteutukset. N.d. Viitattu 1.6.2019. https://www.jamk.fi/fi/Tutkimus-ja-kehitys/projektit/tekoaly-ja-data-analytiikka.

Tutustu JAMKiin. N.d. Viitattu 1.6.2019. https://www.jamk.fi/fi/Tietoa-JAMKista/Tutustu-JAMKiin.

Liitteet

Liite 1. Neuroverkon koodi

```
from keras.applications.xception import Xception
conv_base = Xception(weights='imagenet', include_top=False, input_shape=(299,299,3))

from keras.preprocessing.image import ImageDataGenerator             #For image preprocessing
from keras.models import Sequential                                  #For sequential model
from keras.layers import Activation, Dropout, Flatten, Dense         #For conv layers
from keras import backend as K                                       #Bringing tensorflow backend
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint       #LR reducing on plateau and saving best model
from keras.optimizers import Adam                                    #Bringing optimizer
import keras_metrics                                                 #Bringing Custom metrics to measure performance


img_width, img_height = 299, 299    #Defining picture height and width
train_data_dir = 'Data/Train'       #Training data directory
validation_data_dir = 'Data/Test'   #Validation data directory
nb_train_samples = 1254             #Number of training samples
nb_validation_samples = 60          #Number of validation samples
epochs = 100                        #Number of epochs
batch_size = 9                      #Batch size
checkpointfile = 'weights.{acc:.2f}-{precision:.2f}-{recall:.2f}.h5' #Checkpoint filename


#Sequental model needs to know the input_shape.

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)


#Loading pretrained model to our network

model = Sequential()
model.add(conv_base)
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))


#Configuring the model for training, defining optimizer and performance metrics.

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.0001),
              metrics=['accuracy', keras_metrics.categorical_precision(), keras_metrics.categorical_recall()])


#Preprocessing parameters for training images.

train_datagen = ImageDataGenerator(
    rescale=1. / 255,                   #Scaling RGB values from "0-255" to "0-1"
    zoom_range=0.2,                     #Random zoom between 0 and 0.2
    rotation_range=20,                  #Random rotation of 20 degrees
    horizontal_flip=True)               #Random horizontal flip

test_datagen = ImageDataGenerator(rescale=1. / 255) #Scaling RGB values from "0-255" to "0-1".


#Generates batches of image data for training

train_generator = train_datagen.flow_from_directory(
    train_data_dir,                                 #Training data directory
    target_size=(img_width, img_height),            #Image size
    batch_size=batch_size,                          #Amount of training samples in one pass
    class_mode='categorical')                       #Amount of classes, binary if 2 classes, else categorical

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,                            #Training data directory
    target_size=(img_width, img_height),            #Image size
    batch_size=batch_size,                          #Amount of training samples in one pass
    class_mode='categorical')                       #Amount of classes, binary if 2 classes, else categorical


#Lowering learning rate if no progress is made.

reduce_learning_rate = ReduceLROnPlateau(monitor='loss',
                                         factor=0.1,
                                         patience=2,
                                         cooldown=2,
                                         min_lr=0.00001,
                                         verbose=1)
```

```
#Defining checkpoint that saves the best model or model's weights based on the value that we are monitoring.

checkpoint = ModelCheckpoint(checkpointfile,
                             monitor='acc',
                             verbose=1,
                             save_best_only=True,
                             save_weights_only=True,
                             mode='max')


#Defining callbacks which includes checkpoint and reducing learning rate on plateau.

callbacks = [checkpoint, reduce_learning_rate]


#Training CNN

model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size,
    callbacks=callbacks)
```

Liite 2. Mallista ennustamisen koodi

```
from keras.applications.xception import Xception
conv_base = Xception(weights='imagenet', include_top=False, input_shape=(299,299,3))

import keras_metrics
from keras.models import Sequential
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam


#Using the same architecture and metrics that were used while training the model, including pretrained Xception -model.

model = Sequential()
model.add(conv_base)
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.0001),
              metrics=['accuracy', keras_metrics.categorical_precision(), keras_metrics.categorical_recall()])


#Loading our weights to the model.

model.load_weights('weights.0.9976-1.00-1.00.h5') #Use the weights file that you generated.


#Classes that you are predicting in alphapetical order.

kasvit =['Keltamo','Ketoorvokki', 'Ketunleipa', 'Kevatlinnunsilma', 'Leskenlehti',
         'Litulaukka', 'Metsakurjenpolvi', 'Peltokanankaali', 'Ruotsinliturуoho',
         'Valkovuokko']


from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt

text = input("Tunnistettavan kuvan nimi (mukaanlukien tiedostopääte, esim. kukka.jpg): ")

img = image.load_img(text, target_size = (299, 299))
plt.imshow(img)
img = image.img_to_array(img)
img = np.expand_dims(img, axis = 0)
img /= 255.

prediction = model.predict_classes(img)
print(prediction)

print(kasvit[int(prediction)])
```

Liite 3. Kirjastojen ja ohjelmistojen versiot

Keras 2.2.4

Keras-preprocessing 1.0.5

Keras-applications 1.0.6

keras_metrics 1.1.0

numpy 1.15.4

matplotlib 3.0.2

Tensorflow 1.12.0

Spyder 3.3.4

jupyter_client 5.2.4

jupyter_core 4.4.0