TEKOÄLYN HYÖDYNTÄMINEN IHOSYÖVÄN TUNNISTUKSESSA

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäen kampus, Tieto- ja viestintätekniikka

Kevät, 2020

Mikko Siukola

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Riihimäki

Tekijä Mikko Siukola **Vuosi** 2020

Työn nimi Tekoälyn hyödyntäminen ihosyövän tunnistuksessa

Työn ohjaaja/t Petri Kuittinen

TIIVISTELMÄ

Tämän opinnäytetyön tavoitteena oli tutkia teoreettista ja käytännön prosessia, jolla tekoälyn voisi opettaa tunnistamaan vaarallisia ihosyöpäluomia valokuvista.

Opinnäytetyössä esitellään keinotekoisen neuroverkon teoria, tekoälyn lyhyt historia ja käydään läpi, kuinka avoimeen lähdekoodiin perustuva TensorFlow- koneoppimis framework voitaisiin mahdollisesti opettaa luokittelemaan kuvia. Oppimismetodina keskitytään syväoppimisen menetelmään hyödyntäen konvolutionaalista neuroverkkoa.

Lopuksi tehtiin kokeellinen konvoluutioneuroverkko, jota opetettiin erottamaan terveet luomet sairaista, mutta sen toimivuus jäi kyseenalaiseksi johtuen mahdollisesti liian pienestä opetusmateriaalin määrästä, tai muusta syystä, joka ei selvinnyt opinnäytetyötä tehtäessä.

Avainsanat AI, diagnosointi, ihosyöpä, syväoppiminen, tekoäly

Sivut 33 sivua, joista liitteitä 3 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Author Mikko Siukola **Year** 2020

Subject Utilization of artificial intelligence in the identification of skin cancer

Supervisors Petri Kuittinen

ABSTRACT

The aim of this thesis was to study the theoretical and practical process by which artificial intelligence could be taught to identify dangerous skin cancer moles from photographs.

This thesis presents the theory of artificial neural network, a short history of artificial intelligence and a description of how the open source, TensorFlow- machine learning framework, could possibly be taught to classify images. The learning method focuses on the deep learning method utilizing a convolutional neural network.

Finally, an experimental convolutional neural network was made, which was taught to distinguish healthy moles from sick ones, but its functionality was questioned, possibly due to insufficient amount of teaching material, or for some other reason that could not be solved while writing the thesis.

Keywords Artificial intelligence, AI, deep learning, diagnose, skin cancer

Pages 33 pages including appendices 3 pages

SISÄLLYS

1 JOHDANTO

Keinoälyn nopea kehitys viime vuosina on johtanut sen käytön lisääntymiseen monilla eri tekniikan ja elämän alueilla. Keinoäly mahdollistaa monia asioita, jotka perinteisillä tietoteknisillä menetelmillä ovat olleet hyvin haastavia toteuttaa. Keinoälyllä voidaan tehdä onnistuneita päätelmiä todellisen elämän ongelmista, jossa analysoitava aineisto ei koskaan ole sisällöttään samanlaista tai muotoista vaan muuttuu koko ajan. Kuvan tunnistus ja luokittelu on yksi esimerkki, joka sopii hyvin tekoälyn menetelmään.

Sairauksien diagnosointi on alue, jossa keinoälyä on käytetty onnistuneesti monien sairauksien kohdalla lisääntyvin määrin ja tämän trendin on oletettu jatkuvan edelleen tekniikan ja opetusmateriaalien laadun kehittyessä.

Ihosyövät ovat asiantuntijoiden mukaan lisääntyneet viime aikoina huolestuttavasti, mutta niiden aikainen havaitseminen on edelleen ongelma, sillä siihen tarvitaan aina ammattilaisen tietotaitoa ja monesti ammattilaiselle ymmärretään mennä vasta, kun sairaus on päässyt jo pahenemaan.

Keinoälyä hyödyntämällä voitaisiin ehkä tehdä jonkinlainen mobiiliapplikaatio, jolla voitaisiin tehdä alustava diagnoosi kotioloissa. Ohjelmalla otettaisiin kuva epäilyttävästä ihomuutoksesta, joka sitten syötetäisiin tekoälyn analysoitavaksi. Jos ohjelma arvioi ihomuutoksen epäilyttäväksi, tulisi hakeutua lääkäriin, jossa tehtäisiin varsinainen diagnoosi.

Tässä opinnäytetyössä ei ole tarkoituksena opettaa keinoälyä tunnistamaan ihosyöpien eri tyyppejä vaan pelkästään kokeilla, kykeneekö tekoäly erottamaan terveet luomet sairaista luomista. Tarkempi erottelu on varmasti myös mahdollista, mutta se vaatii huomattavasti suurempaa opetusmateriaalia, jota ei tällä hetkellä ole yleisesti saatavilla.

Johtuen aihepiirin laajuudesta on neuroverkoista koskeva osio rajattu koskemaan vain perinteistä eteenpäin kytkettyä monikerroksista neuroverkkoa ja konvoluutio-osiossa keskitytään vain 2d-kerneliin. Myöskään kaikkia konvoluutioverkon osia ei käydä läpi vaan keskitytään vain sen kaikista yleisimmin käytettyihin osiin.

2 MITÄ ON TEKOÄLY?

Tekoäly on ohjelma tai tietokone, joka kykenee suorittamaan niin sanottuja älykkäitä toimintoja, jotka ovat tyypillisiä lähinnä ihmismäiselle ajattelulle. Keinoälyllä on kyky oppia suorittamaan uusia tehtäviä opettamisen kautta ja se kykenee tulkitsemaan oikein sille syötettyä informaatiota. (Pietikäinen & Silvén, 2019, ss. 5-6)

Tekoälyä luokitellaan yleisesti kolmeen eri luokkaan, heikkoon, vahvaan ja supertekoälyyn. Heikolla tekoälyllä ei ole tietoisuutta ja se suorittaa toimintoja hyvin kapea-alaisella alueella ymmärtämättä mitään suorittamansa toiminnon merkityksestä. Heikko tekoäly kykenee esimerkiksi puheen ja tekstin tunnistamiseen ja pelaamaan pelejä. (Pietikäinen & Silvén, 2019, s. 23)

Vahva tekoäly tarkoittaa älykkyyttä, joka vastaa käsityskyvyltään lähelle ihmisen ajattelukykyä ja se kykenee tiedostamaan olemassaolonsa jollain tasolla. Esimerkiksi vahva tekoäly kykenisi keskustelemaan ihmisen kanssa luontaisesti ymmärtäen samalla keskustelun sisällön. Heikko tekoäly ei tähän kykenisi. (Pietikäinen & Silvén, 2019, s. 23)

Supertekoäly tarkoittaa keinoälyä, joka kykenee suurempaan älykkyyteen kuin maailman älykkäimmät ihmiset. Supertekoäly voisi teoriassa syntyä siten, että vahva tekoäly alkaisi kehittämään itseään koko ajan paremmaksi ja älykkäämmäksi. Supertekoäly ei ole nykytietämyksen mukaan mahdollista toteuttaa. (Pietikäinen & Silvén, 2019, ss. 23-24)

Itse keinoälyn toteutusmenetelmät jaetaan usein myös kahteen luokkaan, koneoppimiseen ja syväoppimiseen (Microsoft, 2020).

2.1 Tekoälyn historia lyhyesti

Tekoäly käsitteenä ei ole uusi keksintö, vaan jo antiikin filosofit ennen ajanlaskun alkua ovat leikkineet ajatuksella mekaanisesta, keinotekoisista ihmisistä alkaen jo muinaisesta Egyptistä. Nykymuotoisen, tietokoneisiin pohjautuvan tekoälyn alun voi sanoa alkaneen Brittiläisen tietokonetutkija, Alan Turingin ja hänen kollegoidensa ideoimasta ja toteuttamasta alkeellisesta tietokoneesta, jota käytettiin toisessa maailmansodassa purkamaan saksalaisten käyttämää enigma-koodia. (Lewis, 2014)

1950-luvun alussa Von Neuman ja Alan Turing kehittivät eteenpäin tietokoneen toimintatapaa ja ottivat käyttöön binäärisen laskennan luoden samalla teknisen pohjan nykyisen kaltaisen tietokoneen toiminnalle. Alan Turing myös julkaisi artikkelin vuonna 1950, jossa hän leikki ajatuksella keinotekoisesta älystä, joka yrittäisi huijata ihmistä luulemaan sitä myös ihmiseksi, kun molemmat kirjoittelisivat toisilleen näkemättä kuitenkaan,

onko toisella puolella kone vai ihminen. Tämä ajatus on saanut nimen Turingin testi. (Council of europe, 2020)

Tekoäly omana virallisena tieteellisenä tutkimusalueena syntyi kuitenkin vasta vuonna 1956 Yhdysvalloissa, Dartmouth:in yliopistossa järjestetyssä conferenssissa, jossa myös termi tekoäly syntyi (Council of europe, 2020).

Konferenssin innostamana moni alaa seuraava tutkija oli vakuuttunut, että tekoälyn haasteet saadaan ratkaistua hyvinkin pian, jopa kymmenessä vuodessa, mutta kun tutkimus ei kuitenkaan käytännössä edennyt odotetulla tavalla, tekoälyn tutkimuksen rahoitusta leikattiin rajusti vuosiksi 1974–1980 ja tämä aika on saanut nimen ensimmäinen AI-talvi (Lewis, 2014).

Ensimmäisten mikroprosessoreiden tullessa 1970-luvun lopulla tekoälyn tutkimus koki taas uuden nousukauden, kun asiantuntijajärjestelmät tulivat käyttöön. Ne imitoivat ihmisen käyttämää logiikkaa järkeillessään asioita. Kuitenkin 1980-luvun loppuun mennessä myös asiantuntijajärjestelmien kehitys oli osoittautunut vaikeammaksi kuin luultiin, ja tekoälyn tutkimus vaipui taas uuteen talveen. Jopa termistä tekoäly oli tullut jonkinlainen tabu ja sen käyttöä pyrittiin välttämään. (Council of europe, 2020)

2010-luvun alusta tekoälyn tutkimus lähti taas uuteen nousuun pitkän tauon jälkeen. Tämä johtui tietokoneiden laskentatehon kehittymisestä ja myös siitä, että järkevän hintaisia, mutta silti tehokkaita näytönohjaimia kyettiin valjastamaan tekoälyn laskentaan. Myös tekoälyn opettamiseen tarvittava suuri määrä dataa oli usein nopeasti ja helposti saatavilla netin hakukoneiden avulla, toisin kuin aikaisemmin, jolloin usein kaikki opetusdata piti kerätä itse. Toisin kuin edellisen tekoälyaallon asiantuntijajärjestelmissä, uudet tekoälysovellukset oppivat itse luokittelemaan asioita ilman ihmisen apua, suuren laskutehon ja opetusdatan avulla. (Council of europe, 2020)

Nykyisin lupaavimmaksi tekoälyn toteuttamisen malliksi monissa tehtävissä on tullut ihmisen aivojen neuroverkon toimintaa löyhästi mallintava syväoppiminen, koska parhaat tulokset eri sovelluksissa on usein saatu sillä (Council of europe, 2020).

2.2 **Koneoppiminen**

Koneoppiminen on termi, joka kuuluu tekoälyn alaisuuteen ja se tarkoittaa kaikkia menetelmiä, joilla kone tai ohjelma voidaan saada oppimaan haluttuja asioita. Sen toiminta perustuu datasta oppiviin algoritmeihin. Koneoppimisessa ohjelma opetetaan joko valmiiksi luokitellulla tai luokittelemattomalla opetusdatalla ja se kykenee itsenäiseen oppimiseen ilman valmiita toimintamalleja, toisin kuin perinteiset ohjelmat. (Gollapudi, 2016, ss. 1-5)

Tosin, vaikka ohjelma on kykenevä oppimaan syötetystä datasta, se tarvitsee kuitenkin usein ohjausta ja parantelua ihmisen toimesta toimiakseen merkittävästi paremmin. Algoritmeilla aikaansaadun ennusteen tarkkuus määräytyy pitkälle käytössä olevan opetusdatan määrän ja laadun mukaan. Koneoppiminen sopii menetelmäksi parhaiten silloin kun opetusdatan määrä ei ole kovin suurta. Koneoppimisen menetelmä ei tarvitse niin suurta laskentatehoa kuin esimerkiksi syväoppiminen. (Microsoft, 2020)

2.3 **Syväoppiminen**

Syväoppiminen kuuluu koneoppimisen alaisuuteen ja se perustuu aivoissa olevien neuroverkkojen toiminnan jäljentämiseen. Neuroverkko perustuu neuroneihin, jotka on kytketty toisiinsa ja ne välittävät dataa toisilleen. Neuroverkoista on luotu kerroksia, jotka myös välittävät dataa keskenään. (Wiley, 2016, ss. 1-5)

Kuin koneoppimisessa, myös syväoppimisessa neuroverkko oppii sille syötetyn opetusdatan avulla. Ero on siinä, että syväoppimisessa piirteiden suodattaminen syötteestä ja myös havaittujen piirteiden luokittelu ovat vapaita ihmisen läsnäolosta, ne oppivat toimimaan halutusti opetusprosessin edetessä riittävästi. (Wiley, 2016, ss. 68-71)

Verrattuna perinteiseen koneoppimiseen, neuroverkon kunnollinen opettaminen vaatii huomattavasti suuremman määrän opetusdataa ja se vie paljon laskentatehoa, mutta se on myös yleensä tarkempi päätelmissään. Syväoppimisella saadaan tehtyä sovelluksia hyvin moniin eri tarkoituksiin, ja tietokoneiden laskentatehon kasvaessa on odottavissa menetelmän kehittyminen ja yleistyminen. (Mathworks, n.d.)

3 KEINOTEKOISET NEUROVERKOT

Keinotekoiset neuroverkot perustuvat ihmisen aivoissa olevan, aivosoluista muodostuvan hermoverkon toimintaan. Luonnollisesti keinotekoiset neuroverkot ovat huomattavasti yksinkertaisempia kuin biologiset vastaavat, mutta niillä voidaan silti suorittaa monia erilaisia tehtäviä, jotka vaativat vain aivoille tyypillisiä ominaisuuksia kuten ennen näkemättömän kuvan sisällön luokittelun. Tällaista ominaisuutta kutsutaan myös kyvyksi yleistää. Normaalille tietokoneohjelmalle esimerkiksi kuvien luokittelu on hyvin vaikea tehtävä, mutta hyvin opetetulle neuroverkolle se voi olla helppo. (Hemanth & Estrela, 2017, ss. 27-29)

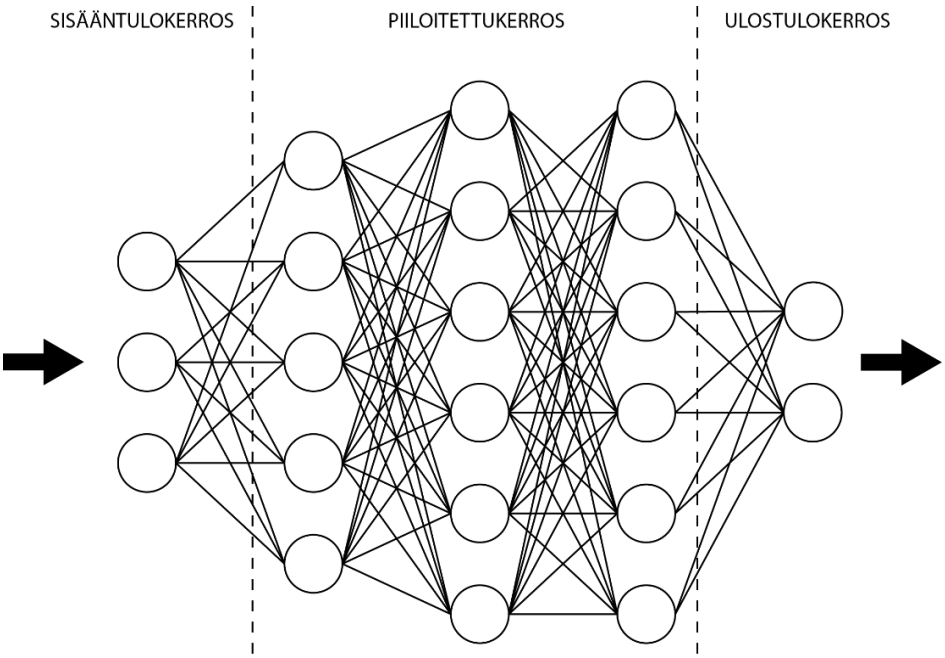
Neuroverkot oppivat tunnistamaan toistuvia kuvioita opetusdatasta ilman erillistä ohjelmointia neuroverkon eri kerrosten avulla, joista jokainen tunnistaa kuvioita ja hahmoja eri tarkkuudella ja se lopulta tekee päätelmän syötetyn materiaalin sisällöstä (Hemanth & Estrela, 2017, ss. 30-32).

Toisin kuin normaali tietokoneohjelma, neuroverkko pitää aina opettaa uuteen tehtäväänsä opetusmateriaalin avulla. Opetusmateriaali voi olla lähes mitä vain, jota voidaan esittää digitaalisessa muodossa ja sitä on saatavilla riittävästi. Hyvälaatuisen opetusmateriaalin puute onkin yksi suurimmista neuroverkkojen kehittämistä hidastavista tekijöistä, sillä esimerkiksi kohtalaisesti onnistuvaan kuvan luokitteluun vaaditaan yleensä vähintään tuhansia kuvia jokaisesta kuvaluokasta, mielellään kymmeniä tuhansia tai enemmän. (Brownlee, 2017)

Itse opetustapahtuma on usein laskennallisesti erittäin raskas tehtävä ja käytännössä kaikissa vähänkään monimutkaisemmassa mallissa käytetään apuna näytönohjain kiihdytystä. Käyttämällä näytönohjainta suorittamaan verkon laskutoimitukset saadaan laskenta-aikaa lyhennettyä huomattavasti, jopa 20-osaan verrattuna siihen, että laskenta suoritettaisiin täysin pelkästään tietokoneen prosessorin avulla. (Zaccone, Karim & Menshawy, 2017, ss. 210-211)

Kun malli on saatu onnistuneesti opetettua tehtävään, voi valmiin mallin tallentaa ja sitä voi käyttää esimerkiksi luokittelemaan kuvien sisältöä kohtalaisen pienillä laskentaresursseilla (Mahapatra, 2018).

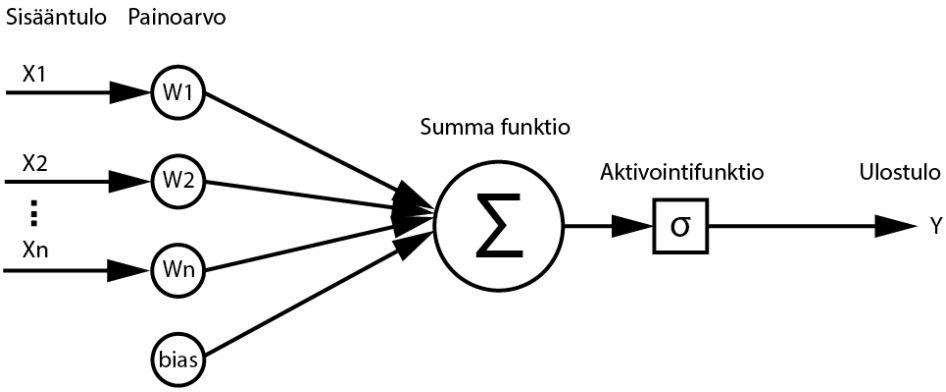
Kuvassa 1 on esitetty perinteisen neuroverkon yksinkertainen malli. Neuroneista rakentuva neuroverkko rakentuu pääpiirteisesti kolmesta eri verkon kerroksesta: sisääntulo kerroksesta, piilotetusta ja ulostulokerroksesta ja informaatio verkossa liikkuu vastaavassa järjestyksessä eteenpäin. Tällaista verkkoa kutsutaan eteenpäin kytketyksi. Sisääntulo kerros ottaa datan vastaan vektorimuodossa, piilotetussa, monesti monta tasoa sisältävässä, täysin kytketyssä kerroksessa tapahtuu itse laskenta ja ulostulokerros antaa vastauksen. (Gollapudi, 2016, s. 318)



Kuva 1. Neuroverkko

3.1 **Neuronin toiminta**

Neuronit ovat yksinkertaisesti vain funktioita, joihin syötetään arvoja sisään kerrottuna tietyllä painokertoimella ja niiden summa syötetään eteenpäin aktivointifunktion kautta muille neuroneille. Kuvassa 2 on esitelty neuronin malli, joista koko neuroverkko rakentuu. (Zocca, Spacagna, Slater & Roelants, 2017, ss. 40-42)



Kuva 2. Neuroni

Neuronin toimintaa kuvaa kaava1:

$$f(x) = b + \sum_i w_i x_i \qquad (1)$$

Jossa b=bias-arvo, joka määrittelee neuronin herkkyyttä, w_i on sisääntulon painoarvo ja x_i sisääntulon arvo (Zocca ym., 2017, s. 23).

Kun neuroverkko alustetaan, niin yleinen tapa on antaa kaikkien neuronien sisääntulojen painoarvoille satunnainen arvo (Kharkovyna, 2019).

Jokaisella neuronilla on myös niin kutsuttu aktivointifunktio, joka määrää millaisen arvon ulostulo saa sisääntulojen ja niiden painokertoimien summaamisen jälkeen (Zocca ym., 2017, s. 46).

Yksi aktivointifunktion tärkeä tehtävä luoda verkkoon epälineaarisuutta, jota ilman verkko ei voisi käytännössä oppia. Jos aktivointifunktio olisi esimerkiksi niin kutsuttu binäärinen funktio, jossa ulostulo voi saada vain kaksi arvoa, 1 tai 0, riippuen asetetusta laukaisukynnyksen tasosta, ei sillä olisi mahdollista toteuttaa luokittelijaa, sillä yksittäisten sisääntulojen eri arvojen vaikutusta ei voisi havaita ulostulossa. Sillä ei ole myöskään derivaattaa, jota tarvitaan, kun verkon neuronien sisääntulojen painoarvoja päivitetään. Siksi on tärkeää, että aktivointifunktio tuottaa ulostulossa epälineaarisen vasteen, josta voidaan päätellä jotain neuronille tulleiden syötteiden arvoista ja painokertoimista verkon opettamista varten. (Ognjanovski, 2019)

Aktivointifunktioita on monia erilaisia ja tässä työssä esitellään kaksi ehkä yleisimmin käytettyä, sigmoid- ja ReLU-funktiot.

Sigmoid-funktio

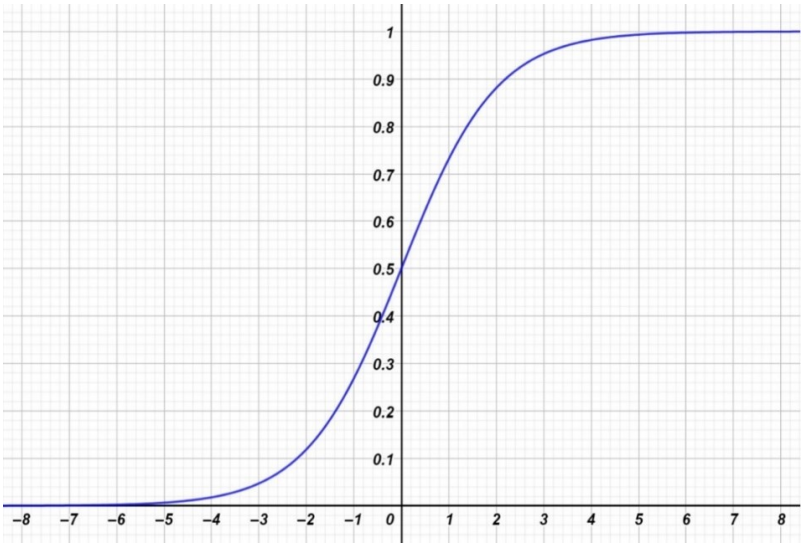
Sigmoid-funktio on vanha, mutta edelleen hyvin yleisesti käytetty.
Funktiota kuvaa kaava 2 (Gulli, 2017, s. 14).

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad (2)$$

Sigmoid-funktio, jonka käyrä on esitelty kuvassa 3, tuottaa ulostulon, jonka arvo on aina välillä 0 ja 1 (Gulli, 2017, s. 14).

Funktiota käytetään yleisimmin binääriluokitteluun tarkoitetuissa neuroverkoissa ulostulokerroksen neuronina, jossa ennusteena voi olla vain kaksi luokkaa (Sharma, 2019).

Käyrän muodon takia, kohtalaisen pienillä x:n muutoksilla Y:n arvo on helposti aina lähellä jompaa kumpaa ääripäätä, tämä selventää saadun ennustuksen luettavuutta. Negatiivisena puolena on se, että hyvin pienellä tai isolla X:n arvolla Y:n arvo muuttuu vain vähän ja tämä voi heijastua ongelmana, kun verkkoa opetetaan. (Sharma, 2019)



Kuva 3. Sigmoid-funktio

Rectified Linear Units (ReLU)-funktio

ReLU-funktio on ehkä yleisin neuroverkoissa käytetty aktivointifunktio, jonka käyrä on esitelty kuvassa 4.

Funktiota kuvaa kaava 3 (Gulli, 2017, s. 15).

$$\sigma(x) = \max(x, 0) \tag{3}$$

Funktio antaa arvon 0, kun sisääntulojen arvo on 0 tai vähemmän ja sen yläpuolisilla arvoilla se antaa ulos saman arvon kuin sisääntulojen yhteenlaskettu arvo on (Gulli, 2017, s. 15).

ReLUn toiminta on kaikista aktivointifunktioista lähimpänä oikean neuronin vastaavaa toimintaa.

Kuva 4. ReLU-funktio

3.2 Neuroverkon opettaminen

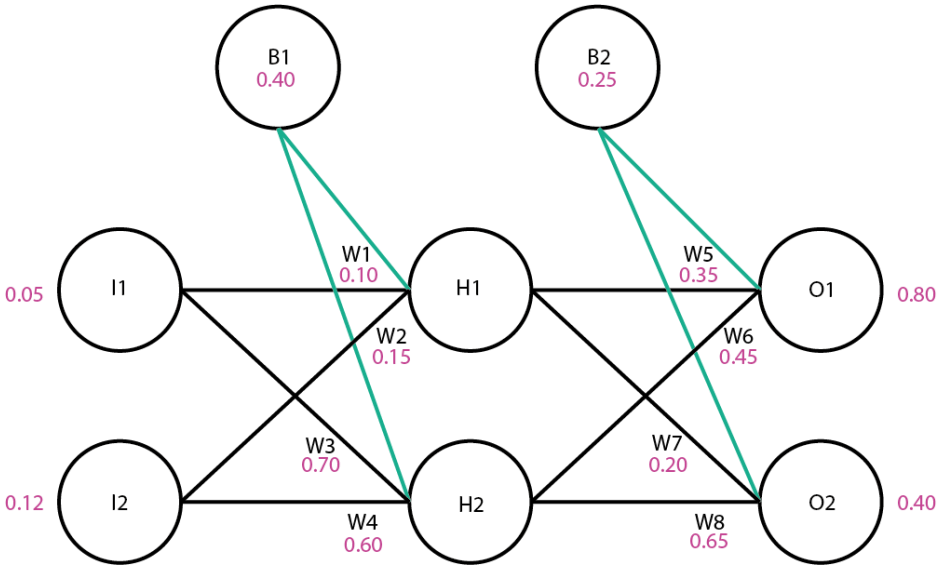
Jotta neuroverkko voisi oppia, sille tulee syöttää suuri määrä mahdollisimman monipuolista ja laadukasta opetusdataa ja aina jokaisen opetuskerran välillä tulee kaikkien neuroverkon neuronien sisääntulojen painoarvoja päivittää perustuen laskettuun virheeseen annetun syötteen ja saadun ennustuksen perusteella. Tätä prosessia kutsutaan verkon optimoimiseksi. Itse opetuskertoja voi olla jopa useita satoja.

Yleinen metodi tähän on nimeltään vastavirta-algoritmi, jossa lasketaan neuronien aktivointifunktion antaman arvon perusteella osittaisderivaatta, josta johdetaan alkuperäiset neuronien syötteet ja neuronien painoarvot päivitetään perustuen niiden osuuteen kokonaisvirheestä. Metodia käyttäen käydään läpi kaikki neuroverkon neuronit yksi kerrallaan, alkaen ulostulosta läpi piilotettujen kerrosten, päätyen lopulta sisään tuleviin neuroneihin. Tämä prosessi tehdään aina jokaisella opetuskerralla virheen minimoimiseksi. (Mazur, 2015)

Alla on annettu esimerkki algoritmin toiminnasta, jossa ensin lasketaan annetuilla sisääntulo arvoilla verkon ulostulo, jonka jälkeen lasketaan vastavirta-algoritmia käyttäen uudet painoarvot verkon neuroneille.

Matt Mazur esittelee julkaisussaan (Mazur, 2015) yksinkertaisen neuroverkon eteenpäin laskennan ja vastavirtalaskennan esimerkin ja alla oleva laskelma perustuu siihen suoraan, vain neuronien painoarvot, syötteet ja halutut vasteet on vaihdettu toisiin.

Kuvan 5 esimerkkiverkko koostuu kahdesta sisääntuloneuronista, kahdesta piilotetun kerroksen neuronista ja kahdesta ulostuloneuronista. Jokaiselle neuronille on annettu satunnainen sisääntulon painoarvo ja myös satunnainen bias-arvo. Yksinkertaistamisen vuoksi bias-arvojen päivitystä ei huomioida vastavirtalaskennassa. Haluttuna ominaisuutena neuroverkolla olisi antaa kuvassa olevilla sisääntuloarvoilla ulostulona kuvassa olevat halutut arvot. Verkossa aktivointifunktiona käytetään sigmoid-funktiota.



Kuva 5. Esimerkkiverkko

Lasketaan verkon antama arvo annetuilla syötteillä.

Neuronin H1 saama syöte saadaan kaavasta

$$f(x) = b + \sum_i w_i x_i \quad (4)$$

$$net_{h1} = w1 * i1 + w2 * i2 + b1$$

$$net_{h1} = 0.10 * 0.05 + 0.15 * 0.12 + 0.40$$

$$net_{h1} = 0.423$$

Lisätään aktivointifunktio, jotta saadaan selville neuronin h1 ulostulon arvo

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.423}} = 0.604201$$

Lasketaan syöte samalla tavalla neuronille H2

$$net_{h2} = w3 * i1 + w4 * i2 + b1$$

$$net_{h2} = 0.70 * 0.05 + 0.60 * 0.12 + 0.40$$

$$net_{h2} = 0.507$$

Lisätään aktivointifunktio, jotta saadaan selville neuronin h2 ulostulon arvo

$$out_{h2} = \frac{1}{1 + e^{-net_{h2}}} = \frac{1}{1 + e^{-0.507}} = 0.624103$$

Seuraavaksi lasketaan ulostulokerroksen neuronien O1 ja O2 sisääntulo arvot käyttämällä hyväksi aikaisemmin laskettuja piilokerroksen neuronien H1 ja H2 ulostuloarvoja

$$net_{o1} = w5 * out_{h1} + w6 * out_{h2} + b2$$

$$net_{o1} = 0.35 * 0.604201 + 0.45 * 0.624103 + 0.25$$

$$net_{o1} = 0.742317$$

Lisätään aktivointifunktio, jotta saadaan selville neuronin O1 ulostulon arvo

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-0.742317}} = 0.677502$$

Lasketaan syöte samalla tavalla neuronille O2

$$net_{o2} = w7 * out_{h1} + w8 * out_{h2} + b2$$

$$net_{o2} = 0.20 * 0.604201 + 0.65 * 0.624103 + 0.25$$

$$net_{o2} = 0.776507$$

Lisätään aktivointifunktio, jotta saadaan selville neuronin O2 ulostulon arvo

$$out_{o2} = \frac{1}{1 + e^{-net_{o2}}} = \frac{1}{1 + e^{-0.776507}} = 0.684927$$

Jotta saadaan selville, kuinka laskemalla saadut tulokset 01 ja 02-ulostuloissa vastaavat haluttuja ulostulon arvoja, tulee molempien ulostuloneuronien virhe laskea käyttämällä kaavaa

$$E_{total} = \sum \frac{1}{2}(target - output)^2 \quad (5)$$

Neuronin O1 virhe

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.80 - 0.677502)^2 = 0.007503$$

Neuronin O2 virhe

$$E_{02} = \frac{1}{2}(target_{o2} - out_{o2})^2 = \frac{1}{2}(0.40 - 0.684927)^2 = 0.040592$$

Lasketaan neuronien virheet yhteen, jolloin saadaan kokonaisvirhe

$$E_{total} = E_{o1} + E_{o2} = 0.007503 + 0.040592 = 0.048095$$

Vastavirta-algoritmin soveltaminen ulostulokerroksen neuronien painoarvojen päivitykseen

Jotta neuronien painoarvoja voidaan optimoida, tulee laskea jokaisen neuronin sisääntulon painoarvon vaikutus kokonaisvirheeseen. Tämä voidaan tehdä laskemalla osittaisderivaatta kokonaisvirheestä E_{total} suhteessa haluttuun painoarvoon. Suhdetta kuvaa kaava

$$\frac{\partial E_{total}}{\partial w_x} \quad (6)$$

Neuroni $\boldsymbol{o_1}$,painoarvo $\boldsymbol{w_5}$

Haluttaessa esimerkiksi selvittää painoarvon w5 vaikutusta kokonaisvirheeseen voidaan soveltaa ketjusääntöä, josta arvo voidaan johtaa.

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial w_5} \quad (7)$$

Ensin pitää kuitenkin selvittää kokonaisvirheen suhde neuronin O1 ulostuloon laskemalla osittaisderivaatta, joka saadaan derivoimalla ∂out_{01}-suhteen kaavasta

$$E_{total} = \frac{1}{2}(target_{01} - out_{01})^2 + \frac{1}{2}(target_{02} - out_{02})^2 \quad (8)$$

josta saadaan, huomioiden että laskettaessa osittaisderivaattaa kokonaisvirheen suhteesta O1 ulostuloon out_{01} ,tulee O2 arvoksi nolla

$$\frac{\partial E_{total}}{\partial out_{01}} = -(target_{01} - out_{01}) = -(0.80 - 0.677502) = -0.122498$$

Seuraavaksi tulee selvittää, kuinka paljon on neuronin O1 ulostulon out_{01} suhde sen saamaan kokonaissyötteeseen net_{01}. Tämä saadaan selville osittaisderivoimalla neuronin O1 aktivointifunktio

$$out_{01} = \frac{1}{1 + e^{-net_{01}}}$$

joka saadaan kaavasta

$$\frac{\partial out_{01}}{\partial net_{01}} = out_{01}(1 - out_{01}) \quad (9)$$

$$\frac{\partial out_{01}}{\partial net_{01}} = out_{01}(1 - out_{01}) = 0.677502(1 - 0.677502) = 0.218493$$

Tämän jälkeen voidaan selvittää mikä on neuronin O1 syötteen w_5 suhde O1 kokonaissyötteeseen osittaisderivoimalla kokonaissyötteen kaava (10) syötteen w_5-suhteen

$$net_{o1} = w5 * out_{h1} + w6 * out_{h2} + b2 * 1 \quad (10)$$

ottaen huomioon, että neuroni H1 syöttää painoarvoa w_5

$$\frac{\partial net_{01}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.604201$$

Soveltamalla aikaisemmin esiteltyä ketjusääntöä

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial w_5} \quad (11)$$

saadaan

$$\frac{\partial E_{total}}{\partial w_5} = -0.122498 * 0.218493 * 0.604201 = -0.016171$$

Jotta virhettä saataisiin pienennettyä, tulee edellä saatu tulos vähentää painoarvosta w_5 kertomalla se vielä ennalta määrätyllä oppimiskertoimella (tässä η=0.5), jonka tehtävänä on vähentää isoja painoarvojen muutoksia jokaisella opetuskerralla ylisovittamisen (verkko oppii opetusdatan hyvin mutta ei osaa tehdä onnistuneita yleistyksiä uudesta datasta) ehkäisemiseksi.

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} \quad (12)$$

$$w_5^+ = 0.35 - 0.5 * -0.016171 = 0.358086$$

Vastaavalla tavalla voidaan laskea uudet painoarvot myös muille ulostulokerroksen neuroneille.

$$w_6^+ = 0.458352$$

$$w_7^+ = 0.181425$$

$$w_8^+ = 0.630813$$

Piilotetun kerroksen sisääntulojen arvojen päivitys

Neuroni $\boldsymbol{h_1}$, painoarvo $\boldsymbol{w_1}$

Siirryttäessä laskemaan piilotetun kerroksen neuronin H1 painoarvoa w_1, muuttuu ketjusääntö muotoon

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{\mathrm{h1}}} * \frac{\partial out_{\mathrm{h1}}}{\partial net_{\mathrm{h1}}} * \frac{\partial net_{\mathrm{h1}}}{\partial w_1} \quad (13)$$

koska piilotetun kerroksen neuronin H1 ulostulo vaikuttaa kahteen muuhun neuroniin (O1 ja O2) ja myös niiden virheeseen, tulee tämä ottaa huomioon laskettaessa osittaisderivaattaa, joten kaavassa ketjusäännön kaavan kohdassa

$$\frac{\partial E_{total}}{\partial out_{\mathrm{h1}}}$$

tulee ottaa huomioon sen vaikutus molempiin ulostuloneuroneihin, jolloin se muuttuu muotoon

$$\frac{\partial E_{total}}{\partial out_{\mathrm{h1}}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} \quad (14)$$

Yhtälöstä tulee selvittää molemmat osat alkaen

$$\frac{\partial E_{o1}}{\partial out_{h1}}$$

joka saadaan kaavasta

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} \quad (15)$$

ja tämän arvo

$$\frac{\partial E_{o1}}{\partial net_{o1}}$$

saadaan kaavalla

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} \quad (16)$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = -0.122498 * 0.218493 = -0.026765$$

jossa on hyödynnetty jo aikaisemmin laskettuja arvoja

$$\frac{\partial E_{total}}{\partial out_{\mathrm{o1}}} = -(target_{\mathrm{o1}} - out_{\mathrm{o1}}) = -(0.80 - 0.677502) = -0.122498$$

ja

$$\frac{\partial out_{\mathrm{o1}}}{\partial net_{\mathrm{o1}}} = out_{\mathrm{o1}}(1 - out_{\mathrm{o1}}) = 0.677502(1 - 0.677502) = 0.218493$$

seuraavaksi laskettava arvo

$$\frac{\partial net_{o1}}{\partial out_{h1}}$$

on oltava sama kuin w_5 sillä kun tiedetään että osittaisderivaatta saa arvon

$$\frac{\partial net_{01}}{\partial w_5} = 1 * out_{h1} * w_5{}^{(1-1)} + 0 + 0 = out_{h1}$$

on oltava

$$\frac{\partial net_{01}}{\partial out_{h1}} = w_5 = 0.35$$

Nyt saaduilla arvoilla voidaan laskea yhtälö

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = -0.026765 * 0.35 = -0.009368$$

Sitten lasketaan sama osittaisderivaatalle

$$\frac{\partial E_{o2}}{\partial out_{h1}}$$

jolloin kaava muuttuu muotoon

$$\frac{\partial E_{o2}}{\partial out_{h1}} = \frac{\partial E_{o2}}{\partial net_{o2}} * \frac{\partial net_{o2}}{\partial out_{h1}} \quad (17)$$

josta selvitetään ensin

$$\frac{\partial E_{o2}}{\partial net_{o2}}$$

kaavalla

$$\frac{\partial E_{o2}}{\partial net_{o2}} = \frac{\partial E_{o2}}{\partial out_{o2}} * \frac{\partial out_{o2}}{\partial net_{o2}} \quad (18)$$

Josta tulee ensin selvittää

$$\frac{\partial E_{o2}}{\partial out_{\mathrm{o2}}} = -(target_{\mathrm{o2}} - out_{\mathrm{o2}}) = -(0.40 - 0.684927) = 0.284927$$

Jonka jälkeen lasketaan

$$\frac{\partial out_{o2}}{\partial net_{o2}} = out_{o2}(1 - out_{o2}) = 0.684927(1 - 0.684927) = 0.215802$$

Lasketaan arvo

$$\frac{\partial E_{o2}}{\partial net_{o2}} = \frac{\partial E_{o2}}{\partial out_{o2}} * \frac{\partial out_{o2}}{\partial net_{o2}} = 0.284927 * 0.215802 = 0.061488$$

Ja vielä yksi puuttuva arvo

$$\frac{\partial net_{o2}}{\partial out_{h1}} = w_7 = 0.20$$

Saadaan tulokseksi

$$\frac{\partial E_{o2}}{\partial out_{h1}} = \frac{\partial E_{o2}}{\partial net_{o2}} * \frac{\partial net_{o2}}{\partial out_{h1}} = 0.061488 * 0.20 = 0.012298$$

Ja nyt voidaan laskea kokonaisvirhe neuronin H1-suhteen

$$\frac{\partial E_{total}}{\partial out_{\mathrm{h1}}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = -0.009368 + 0.012298 = 0.00293$$

Jotta saataisiin virhearvo w_1:lle, tarvitsee laskea vielä kaksi arvoa

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.604201(1 - 0.604201) = 0.239142$$

$$\frac{\partial net_{\mathrm{h1}}}{\partial w_1} = i_1 = 0.05$$

Nyt voidaan laskea uusi virhearvo w_1:lle

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{\mathrm{h1}}} * \frac{\partial out_{\mathrm{h1}}}{\partial net_{\mathrm{h1}}} * \frac{\partial net_{\mathrm{h1}}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.00293 * 0.239142 * 0.05 = 0.000035$$

Sitten päivitetään painoarvo $w_1$saadun virhearvon mukaan kaavalla

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.10 - 0.5 * 0.000035 = 0.099983$$

Lasketaan vastaavalla tavalla uudet painoarvot myös muille piilotetun kerroksen neuroneille.

$$w_2^+ = 0.149958$$

$$w_3^+ = 0.699836$$

$$w_4^+ = 0.599607$$

Koska nyt kaikki painoarvot (paitsi bias) ovat päivitetty, voidaan laskea verkon antamat ulostulot uudestaan ja verrata niiden virhettä tilanteeseen ennen painoarvojen päivitystä.

Hyödyntämällä aikaisemmin esitettyä virheen laskumetodia saadaan uudeksi virhearvoksi

$$E_{total} = 0.046407$$

Josta voidaan havaita, että kokonaisvirhe on laskenut hieman alkuperäisestä arvosta (0.048095) painoarvojen päivityksen myötä. Kun painoarvojen päivityskierroksia jatkettaisiin riittävän monta kertaa, pienenisi kokonaisvirhe niin paljon, että aikaisemmin määritellyillä verkon syötteillä ulostulo olisi lähellä haluttua arvoa.

3.3 **Opetusmateriaali**

Opetusmateriaalin hankkiminen on yksi suurimmista ongelmista neuroverkkojen opettamisessa, sillä opetuskuvia tarvitaan yleensä vähintään tuhansia, mielellään kymmeniätuhansia tai enemmän. Tarvittava määrä opetuskuvia riippuu myös itse luokittelun kohteena olevasta kuvasta ja opetettavan neuroverkon monimutkaisuudesta. Lähtökohtaisesti kuvien luokittelemiseen tarvitaan vähintään 1000 kuvaa jokaisesta luokasta. (Theophano, 2019)

On myös tärkeää, että luokiteltavissa luokissa on mahdollisimman sama määrä kuvia, sillä kun kuvamäärät eivät ole balanssissa niin luokittelun tarkkuus alkaa heikentymään. Yleensä käy niin, että mallin tekemät ennusteet kääntyvät enemmän siihen luokkaan, jonka kuvia on eniten. Lähtökohtaisesti, kun kyse on syväoppivista neuroverkoista, voi tehdä oletuksen, että ennustusten tarkkuus paranee opetusmateriaalin määrän kasvaessa. (Theophano, 2019)

3.4 **Neuroverkkojen alttius vääriin päätelmiin**

Vaikka syväoppivat neuroverkot ovat osoittautuneet luokittelemaan ja tekemään ennusteita erittäin hyvin, ne eivät kuitenkaan ole vapaita virhearvioinneista. On nimittäin havaittu, että hyvinkin kehittyneitä neuroverkkoja pystytään yllättävän yksinkertaisesti huijaamaan antamaan vastaukseksi täysin jotain muuta, kuin voisi ennalta olettaa. Tämä onnistuu esimerkiksi sijoittamalla tunnistettavaan kuvaan ylimääräisiä tunnistusta hämääviä merkkejä tai sitten äänen tunnistuksen ollessa kyseessä sekoittamalla tunnistettavaan äänisignaaliin valkoista kohinaa. Joskus kuvan tunnistamisen ollessa kyseessä, riittää jopa muutaman kuvapikselin muokkaaminen tietyllä tavalla väärän tuloksen aikaansaamiseksi. (Heaven, 2019)

Tekoälyä käytetään yhä enemmissä määrin erilaisissa yhteyksissä, jossa niiden luotettavuus pitää olla huippuluokkaa, kuten itseajavissa autoissa ja sairauksien diagnosoinnissa. Keinotekoisten neuroverkkojen alttiudella manipulointiin, jolla voidaan saada aikaan virheellisiä johtopäätöksiä, voi olla vakavia seuraamuksia ja se muodostaa todellisen riskitekijän. (Heaven, 2019)

Koska hyvin opetettu tekoäly voi olla erittäin hyvä siinä mihin se on opetettu ja se tavallaan omaa kykyä tehdä päätelmiä ihmismäisesti, usein ajatellaan, että tekoäly ymmärtää näkemänsä, mutta tämä ei pidä paikkaansa. Tekoäly ei ymmärrä lainkaan sille syötetyn materiaalin todellista merkitystä, toisin kuin ihminen. Se vain kykenee oppimaan syötetystä datasta erilaisia toistuvia muotoja ja luokittelemaan niitä, ja koska ymmärrys käsiteltävän datan merkityksestä puuttuu, on myös sen huijaaminen kohtalaisen helppoa. (Heaven, 2019)

Alttius tahalliseen tai tahattomaan manipulointiin koskee kaikkia syväoppivia verkkoja, eikä vain kuvien luokitteluun tehtyjä neuroverkkoja. Käyttämällä erilaisia huijausmenetelmiä on jopa mahdollista saada tiettyyn tarkoitukseen kehitetty ja opetettu oppiva neuroverkko muokattua toimimaan pysyvästi täysin eri tavalla, kuin sen oli tarkoitus toimia. Tätä voisivat esimerkiksi hakkerit hyödyntää monilla eri tavoilla. (Heaven, 2019)

Yksi keino saada vähennettyä neuroverkon alttiutta vääriin tulkintoihin on opettaa neuroverkkoa isolla määrällä materiaalia, joissa osaa materiaalista on tarkoituksella muokattu niin, että ne aikaansaisivat väärän päätelmän, ja lopulta riittävän suuren opetusmäärän ja virheiden korjauksen jälkeen, verkko oppisi olemaan välittämättä tarkoituksellisista tai tahattomista häiriötä aiheuttavista asioista. Mitään varmaa keinoa, jolla saataisiin tehtyä tekoälystä täysin virhevapaa, koski se sitten tahallista manipulointia tai verkon omia luonnollisesti syntyneitä vääriä päätelmiä, ei kuitenkaan ole. (Heaven, 2019)

4 KONVOLUTIONAALISEN NEUROVERKON RAKENNE

Konvolutionaalinen neuroverkko on syväoppimisen algoritmi, joka kykenee erityisen hyvin oppimaan ja erottamaan syötetystä kuvasta piirteitä, joiden perusteella se voi myös tehdä luokituksia kuvan sisällön suhteen. Konvolutionaalinen neuroverkko oppii itse tunnistamaan halutut ominaisuudet kuvasta opetusdatan avulla ilman ihmisen apua, joten sen käyttöönotto uutta tehtävää varten on kohtalaisen helppoa, toisin kuin monissa muissa tunnistamiseen tehdyissä algoritmeissa, joissa kuvasta ominaisuuksia kartoittavien filtterien suunnittelu tulee tehdä ihmisen toimesta. (Saha, 2018)

Konvolutionaalisen neuroverkon toimintatapa perustuu ihmisen näköaivokuoren toimintaan, jossa jokainen neuroni reagoi vain pieneen osaan reseptiivistä kenttää ja yhdistämällä kaikkien neuronien informaatio voidaan muodostaa käsitys koko kuvasta. Yksittäisten neuronien muodostama kuva ei ole myöskään tarkkarajainen, vaan jokaiset vierekkäiset kuvat ovat hieman päällekkäin toistensa kanssa. (Saha, 2018)

Konvoluutioverkko teknisesti perustuu keinotekoisista neuroneista rakennettuun perinteiseen verkkoon, jossa data(kuva) syötetään ennen varsinaista neuroverkkoa niin kutsuttuun kerneliin, joka eriyttää kuvasta erilaisia muotoja käyttäen konvoluutio operaatioon perustuvia suodattimia, joita on yleensä monia rinnakkain monessa eri tasossa. Ensimmäiset konvoluutiosuodattimet eriyttävät alhaisemman tason muotoja, viivoja ym. Ylemmän tason suodattimet puolestaan monimutkaisempia muotoja ja hahmoja. Suodattimilta saadut muodot tallennetaan niin kutsuttuun hahmokarttaan. (Saha, 2018)

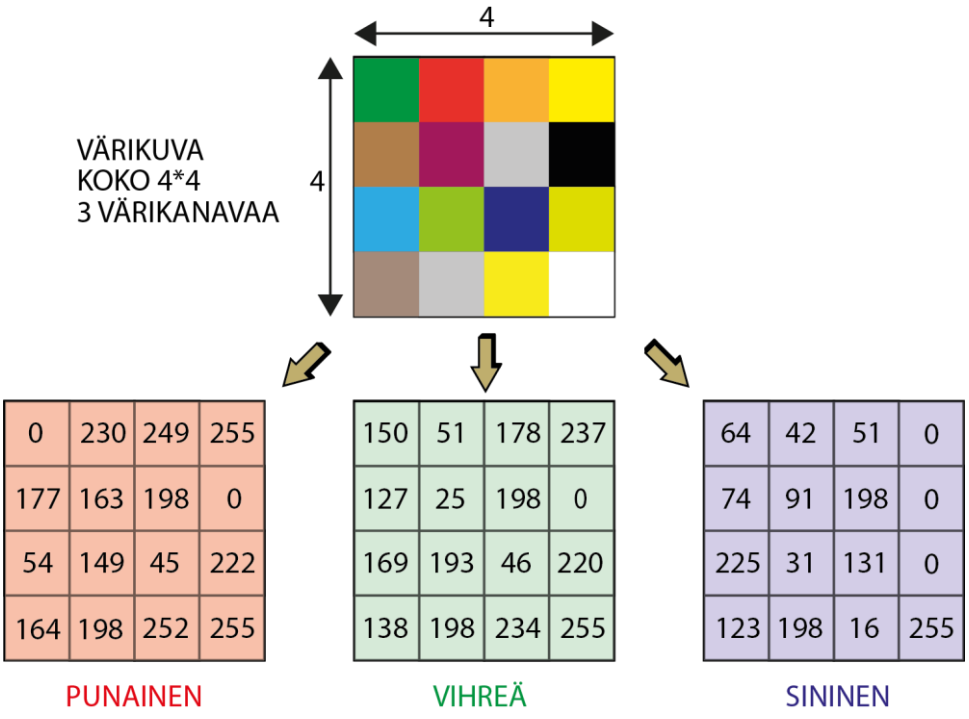
Erittäin tärkeä osa kerneliä on myös pienentää myöhemmin prosessoidun datan määrää. Hahmokartoista saatu data muutetaan yksiulotteisiksi lineaarivektoreiksi ja syötetään piilotettuun kerrokseen, joka tekee luokittelun. Tämän jälkeen ulostulokerroksesta voidaan lukea lopputulos, johon verkko päätyi. Jokaisen opetuskerran jälkeen lasketaan vastavirta algoritmilla verkon kaikille painoarvoille, myös suodattimen arvoille, uudet arvot virheen pienentämiseksi. (Saha, 2018)

Syötetty kuva voitaisiin myös syöttää suoraan neuroverkkoon ilman kerneliä, mutta johtuen kuvien suuresta koosta ja mahdollisesti kolmesta värikanavasta, jotka värikuvissa on, tulisi luokittelijasta erittäin monimutkainen sekä laskennallisesti erittäin raskas, sillä jokaista sisään tulevaa pikseliä(värikuvassa*3) varten tulisi olla yksi sisääntuloneuroni. Toisekseen hahmontunnistuksen onnistumisessa on olennaisempaa tehdä luokittelu perustuen kuvassa havaittuihin muotoihin sen sijaan, että tutkittaisiin tarkasti yksittäisten pikseleiden suhdetta toisiinsa ja yritettäisiin siitä tehdä päätelmiä. (Saha, 2018)

Seuraavaksi esiteltävät osat ovat yleisimpiä konvoluutioverkon osia ja myös osa niistä, joita on käytetty työn soveltavassa osassa.

4.1 Syötettävä kuva

Konvoluutioverkossa sisään tuleva kuva koostuu kuvapikseleistä kuvan 6 mukaan, joilla on leveys, korkeus ja syvyys ja joista viimeinen määräytyy kuvan värikanavien lukumäärän mukaan (Zaccone ym., 2017, ss. 114-115).



Kuva 6. Värikuvan muodostuminen

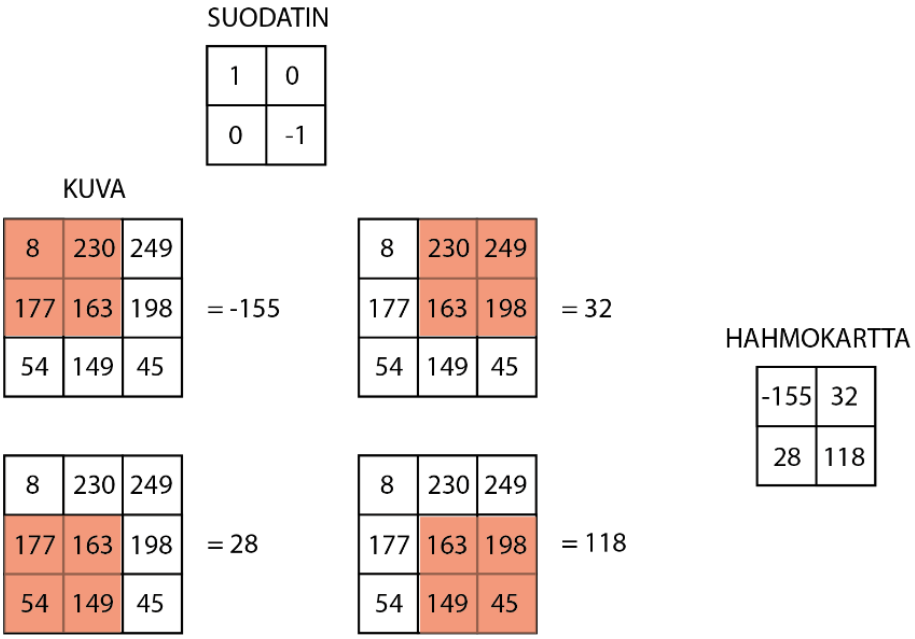
4.2 Konvoluutiokerros

Kuvassa 7 on esitelty konvoluutiosuodatuksen periaate. Syötettyyn kuvaan käytetään konvoluutiometodia, jossa kuvan päälle asetetaan halutut määritteet omaava suodatin. Suodattimen arvot ovat aloitustilanteessa täytetty satunnaisnumeroilla, joita sitten opetuksen edetessä päivitetään vastavirta-algoritmilla. Syötetyn kuvan ja suodattimen pikseleiden välillä tehdään laskutoimitus, jossa syötetyn kuvan pikselit kerrotaan suodattimen vastaavilla pikselin arvoilla ja saatu tulos tallennetaan hahmokarttaan, jonka koko on yleensä alkuperäistä pienempi. Värikuvien kyseessä ollessa (kuva 8), suodatus tehdään jokaiselle värikanavalle erikseen ja niistä saadut arvot summataan yhteen hahmokarttaan. (Saha, 2018)

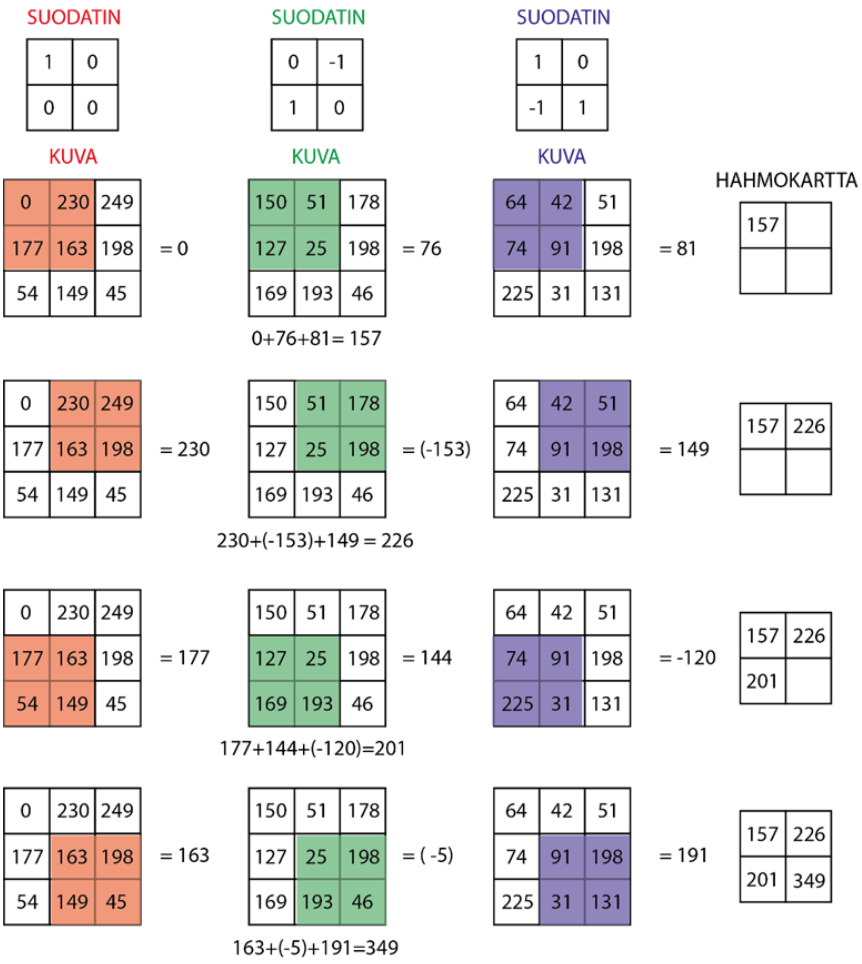
Suodatin liikkuu kuvan päällä, alkaen vasemmasta yläreunasta ja liikkuen oikealle pikseli kerrallaan. Kun kuvan oikea reuna tulee vastaan, suodatin liikkuu yhden pikselin alas, palaa takaisin vasempaan reunaan alkaen taas

liikkeen oikealle ja käyden lopulta läpi kaikki kuvan pikselit. Suodattimen liikkumiselle voidaan myös antaa lisämääritteitä, esimerkiksi meneekö suodatin yhden askeleen kerrallaan eteenpäin vai hyppääkö se isomman askeleen. Syötettävää kuvaa voidaan myös esi-täyttää halutuilla arvoilla reunan ulkopuolisilta alueilta, jossa osa suodattimen kattamasta alueesta voisi olla tyhjää ja tämä voisi vaikuttaa saatuun tulokseen. Riippuen konvoluutio suodattimen määritteistä, voidaan kuvasta eriyttää erilaisia asioita. (Zocca ym., 2017, ss. 141-149)

Jokainen konvoluutiokerros sisältää yleensä monia eri suodattimia, joista jokainen käsittelee syötettyä kuvaa erilaisilla suodatinarvoilla ja tekee oman hahmokartan, joka toimii sitten sisääntulona seuraavan kerroksen suodattimille tai piilokerrokselle (Zocca ym., 2017, ss. 141-149).



Kuva 7. Konvoluutio operaatio



Kuva 8. Värikuvan konvoluutio operaatio

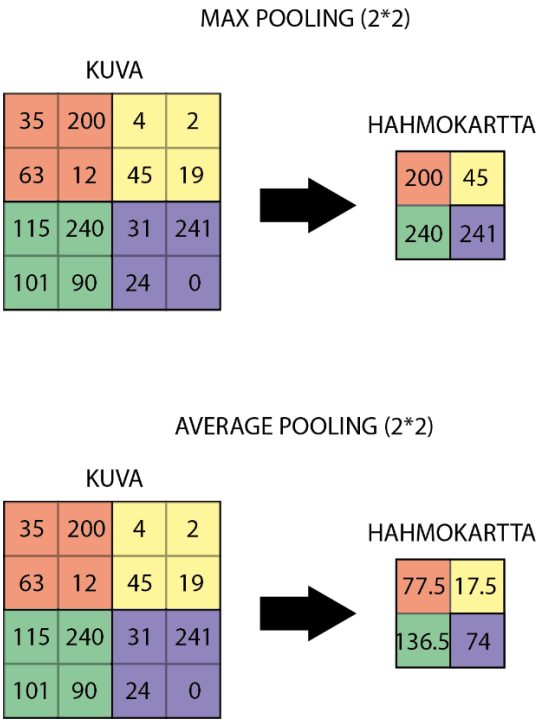
4.3 **Pooling-kerros**

Pooling-kerros, jota käytetään usein konvoluutiokerroksen jälkeen, on hieman samantapainen kuin konvoluutiokerros ja sillä on kaksi päätarkoitusta: vähentää edelleen neuroverkolle syötettävän datan määrää ja tehdä konvoluutiosuodattimien eriyttämistä muodoista riippumaton niiden sijainnista itse kuvassa (Brownlee, 2019).

Konvoluutiokerroksen suodattimet tallentavat myös paikan, jossa kuvassa erottui havaittu muoto ja jos uudessa kuvassa, jota suodattimelle syötetään, on haluttu muoto eri paikassa, saa tämä aikaan erilaisen hahmokartan, joka ei välttämättä toimi niin hyvin. Pooling-kerroksen metodit vähentävät tätä ongelmaa. (Brownlee, 2019)

Kuvassa 9 on esitelty pooling-kerroksen toiminta. Kaksi yleisintä Pooling-metodia on Max-pooling ja Average-pooling. Max-pooling palauttaa suurimman arvon, joka on senhetkisessä kernelin kattamassa alueessa ja Average-pooling palauttaa arvon, joka on kernelin alueella olevien kaikkien arvojen keskiarvo. (Zocca ym., 2017, ss. 150-151)

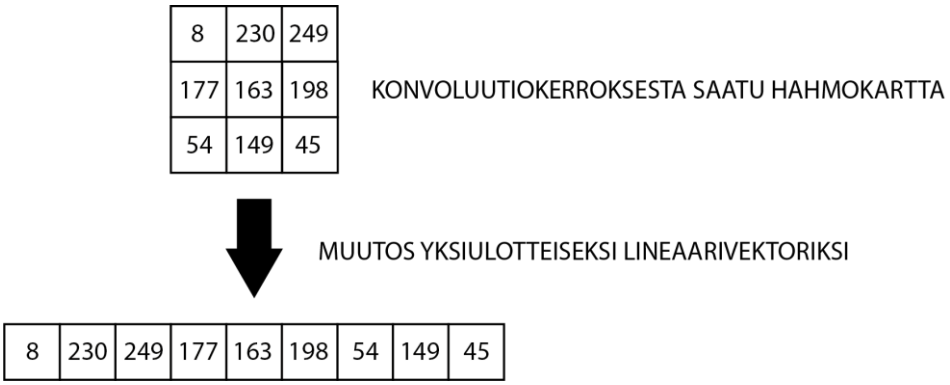
Pooling-kernelissä käytetään yleisesti kernelin kokona 2*2pikseliä ja askelluksena 2, jolloin pooling-kernelin tuottama hahmokartta on puolet pienempi kuin siihen konvoluutiokerroksesta syötetty hahmokartta (Zocca ym., 2017, ss. 150-151).



Kuva 9. Pooling operaatio

4.4 **Flattening**

Kun halutut pooling-kerrosten muotokartat halutaan syöttää luokittelun tekevälle neuroverkolle, tulee ne ensin muuttaa yksiulotteisiksi lineaarivektoreiksi kuvan 10 mukaan (SuperDataScience, 2018).



Kuva 10. Vektorimuutos

Yksinkertaistettuna kaikki kuvan pikseliarvot laitetaan peräkkäin jonoon muodostaen yhden pitkän lukujonon (SuperDataScience, 2018).

5 SUUNNITTELUOHJELMISTOT

5.1 Tensorflow

TensorFlow on googlen kehittämä vapaan levityksen symbolinen matematiikkakirjasto (AI-alusta), jota voi käyttää syväoppivien neuroverkkojen suunnitteluun ja toteutukseen (Tensorflow, n.d).

Varsinainen ohjelmointikieli Tensorflow:ssa on Python, mutta se tukee myös javascriptiä ja node.js:ää. Normaalin version lisäksi on saatavilla kevyt versio mobiililaitteille ja tuotannolliseen vaativampaan käyttöön tarkoitettu versio. Tensorflow tukee myös grafiikkaohjain-kiihdytystä, joka on lähes välttämätön neuroverkkojen opetuksessa. (Tensorflow, n.d)

Tensorflow:lle on saatavilla erittäin hyvin dokumentaatiota alkaen yksinkertaisista harjoituksista monimutkaisiin malleihin. Lopputyötä kirjoitettaessa uusin Tensorflow-versio on 2.0. (Tensorflow, n.d)

5.2 Keras

Keras on François Chollet:in kehittämä vapaan levityksen API, joka toimii monen eri AI-alustan päällä. Keras yksinkertaistaa huomattavasti vaadittavaa ohjelmointityötä neuroverkkoja suunniteltaessa, joka nopeuttaa suunnitteluprosessia ja tekee siitä myös helpommin ymmärrettävää. Kerasin ohjelmointikieli on Python. (Keras, n.d.)

Kerasissa on tuki monille yleisesti käytetyille neuroverkkomalleille ja niiden osille ja myös konvoluutionaalisille ja takaisinpäin kytketyille neuroverkoille. Myös näytönohjain kiihdytys on tuettu. (Keras, n.d.)
Lopputyötä kirjoitettaessa uusin Keras-versio on 2.3.0

6 SOVELTAVA PROJEKTI

Soveltavana projektina oli tarkoitus testata, kuinka konvolutionaalinen neuroverkko hyödyntäen Tensorflow:ta ja Keras:ia oppii tunnistamaan sairaat luomet terveistä luomista.

6.1 Mallin suunnittelu ja toteutus

Jotta opetus onnistuisi suunnitellusti, tulee neuroverkolla olla riittävästi laadukasta opetusmateriaalia käytössään. Sivustolta nimeltään isic-archive.com löytyi kootusti tuhansittain kuvia erilaisista syöpäluomista ja myös terveistä luomista. Osassa kuvista oli muutakin kuin itse luomi, (värimerkkejä ym.) joten kaikki kuvat tuli käydä läpi ja poistaa ne, jotka eivät kelpaa opetukseen.

Lopullinen varsinainen opetusdata koostui yhteensä 5580 kuvasta, 1580 kuvaa terveistä luomista ja 4000 kuvaa sairaista luomista.

Kuvia sairaista luomista olisi ollut mahdollista ottaa käyttöön noin 10000, mutta koska terveiden luomien kuvia oli vain 1580, olisi tämä johtanut liian suureen eroon luokkien välillä, jolloin neuroverkko todennäköisesti oppisi tunnistamaan toisen luokan huomattavasti paremmin kuin toisen. Opetuksessa myös tarvittavia validointikuvia puolestaan oli yhteensä 1395, 395 kuvaa terveistä luomista ja 1000 kuvaa sairaista luomista.

Luokittelija toteutettiin hyödyntämällä konvolutionaalista neuroverkkoa, joka sopii hyvin kuvien analysointiin. Koska luokkia on vain kaksi, on kyseessä binäärinen luokittelu ongelma.

Frameworkina oli Tensorflow, jonka päälle vielä asennettiin Keras helpottamaan sen ohjelmointia. Ohjelman koodi on esitelty liitteessä 1.

Ohjelmointikieleksi valittiin Python, sillä Keras tukee vain sitä ja myös koska suurin osa opetusmateriaaleista ja esimerkeistä on tehty sillä. Koska näytönohjaimen hyödyntäminen neuroverkon opetuksessa on käytännössä välttämätöntä, oli se myös saatava toimimaan järjestelmässä.

Suurena yllätyksenä tuli se, että kaikkien ohjelmien saaminen toimimaan yhdessä näytönohjainlaskennan kanssa oli hyvin haastavaa, sillä kaikista ohjelmista ja myös näytönohjaimen ajureista tarvittiin tietyt ohjelmistoversiot, jotta ne toimisivat toistensa kanssa. Tarkkaa tietoa oikeista versioista oli vaikea löytää ja meni paljon aikaa, että kaikki rupesi toimimaan niin kuin pitää.

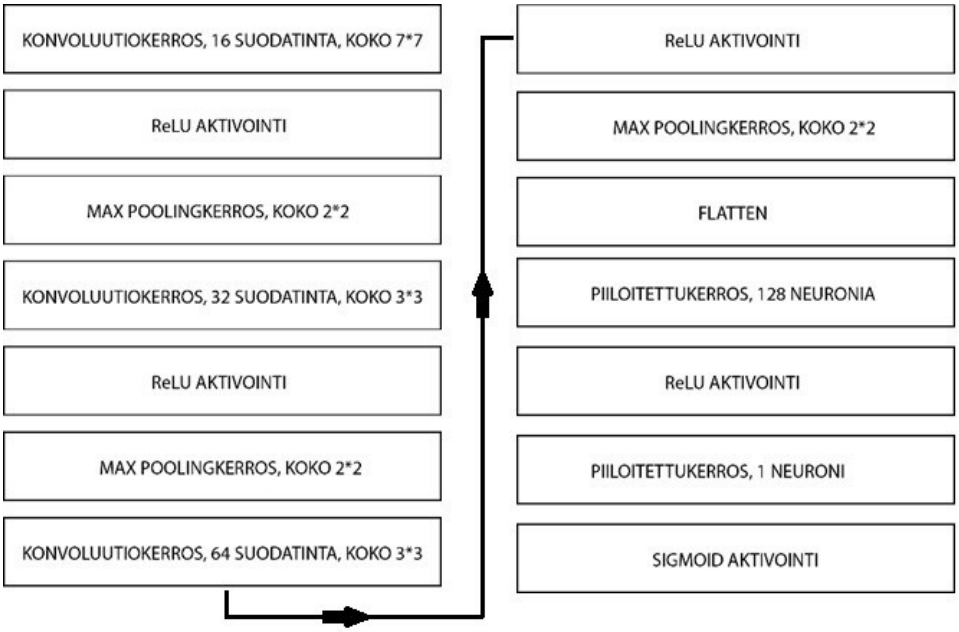
Kun lopulta kaikki toimi niin kuin piti, niin päästiin vihdoin rakentamaan itse neuroverkkomallia. Mitään suoraa ohjetta miten neuroverkkomalli tulisi kyseisessä tapauksessa toteuttaa ei ollut, joten mallin pohja rakennettiin yhdistelemällä monia eri konvoluutioverkon esimerkkejä, joita löytyi internetin välityksellä.

Mitään suoraa ohjetta mallin optimaaliseen hienosäätöön ei myöskään ollut saatavilla, joten parhaan tuloksen antanut malli optimoitiin puhtaasti kokeilemalla hyvin monia erilaisia asetuksia neuroverkkomallin eri osissa. Yksityiskohtaisten hienosäätöohjeiden puutos johtuu mahdollisesti siitä,

että optimaalinen neuroverkon rakenne riippuu todennäköisesti aina vahvasti luokittelun kohteena olevien kuvien informaatiosisällöstä, joten kun tehdään tunnistusta kuvilla mitä varten ei olla aikaisemmin rakennettu onnistunutta neuroverkkomallia, parhaat asetukset löytyvät usein vain kokeilemalla.

Aluksi mallia rakentaessa oli oletus, että mitä suurempaa ja monimutkaisempaa neuroverkkoa käytetään, sitä parempaa tulosta voi odottaa. Tämä ei yllättäen pitänytkään paikkaansa vaan vasta kun päätettiin kokeilla päinvastaista metodia, verkon pienentämistä ja yksinkertaistamista, rupesi ennustustarkkuus nousemaan huomattavasti. Onkin mahdollista, että monimutkainen verkko oppii nopeasti tunnistamaan sille opetetut kuvat, mutta samalla se rupeaa menettämään kykyään tehdä onnistuneita yleistyksiä kuvista, mitä se ei ole aikaisemmin nähnyt.

Toteutuneen, parhaan tuloksen antaneen mallin topologiaksi tuli kuvan 11 mukainen malli.



Kuva 11. Konvoluutioverkon topologia

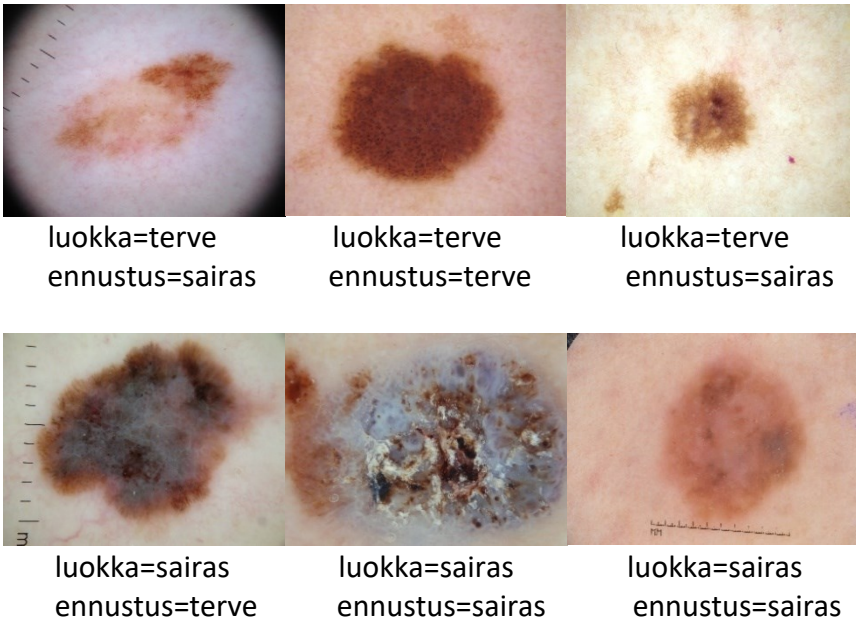
6.2 **Tulokset**

Pitkän testaamisen jälkeen optimaaliseksi määräksi opetuskertoja käytössä olevalla opetusaineistolla tuli 10 ja suurin tarkkuus johon neuroverkko päätyi (Kerasin automaattinen tarkkuuden testaus) oli 80 %. Tämä tarkkuus vaihtelee aina hieman, jos mallin opettaa uudestaan, sillä jokaisella kerralla, kun verkko alustetaan, sen kaikkien neuronien painoarvot saavat uuden satunnaisarvon.

Jotta mallin todellisesta tarkkuudesta saisi paremman käsityksen, valmiille mallille syötettiin 20 kuvaa (10 tervettä ja 10 sairasta), joille se antoi todennäköisyyden kuulumisesta luokkaan yksi, joka tässä tapauksessa on terveet luomet. Kyseisiä kuvia ei ole käytetty verkon opetusdatassa. Malli luokitteli kuvat antamalla arvon alueella 0-1 jolloin ennustus<0.5 edustaa sairasta luomea ja ennustus>0.5 tervettä.

Kuten ennakkoon oli oletettavissa, opetuksen aikana saatu tarkkuus ei pitänyt paikkaansa. Malli ennusti oikein terveistä luomista 50 % ja sairaista 80 %, joka tekee yhdistetyksi tarkkuudeksi 65 %. Tosin, kun testin ajoi uusilla kuvilla niin tunnistuksen tarkkuus vaihteli täysin satunnaisesti ja mallin toimivuus jäi kyseenalaiseksi.

Kuvassa 12 on esitetty esimerkkinä 6-kappaletta mallin antamia ennusteita molemmista luokista.



Kuva 12. Esimerkkejä ennustuksista

7 POHDINTAA

Syväoppivan konvoluutioverkon rakentaminen ja opettaminen onnistuvat kohtalaisen hyvin, koska nykypäivänä on käytössä Tensorflow'n ja Kerasin kaltaisia kirjastoja, jotka tekevät mallien rakentamisesta ja opettamisesta paljon helpompaa kuin aikaisemmin.

Alkuperäinen kokeellinen tavoite, terveiden luomien erottaminen sairaista ei onnistunut luotettavasti. Välillä ennusteet menevät paremmin, mutta hyvin usein ne jäävät kyseenalaisiksi. Syytä tähän ei selvinnyt.

Vaikka Kerasin ilmoittama mallin tunnistustarkkuus on yllättävän korkea, jää jälkikäteen testattu todellinen tarkkuus paljon pienemmäksi. Jos opetusdataa olisi huomattavasti enemmän, vähintään kymmeniätuhansia kuvia molemmissa luokissa, kasvaisi tunnistustarkkuus todennäköisesti myös paljon paremmaksi.

On myös todennäköistä, että terveiden luomien huonompi tunnistustarkkuus johtuu ainakin osittain siitä, että niiden kuvia on huomattavasti vähemmän kuin sairaiden luomien kuvia.

Kun luomien kuvia sisältävät tietokannat oletettavasti suurenevat tulevaisuudessa, pääsee paremmin testaamaan mallin toimivuutta, kun on käytössä luokkien suhteen balanssissa oleva opetusaineisto.

Voisi myös olettaa, että johtuen molempien luokkien kuvien samanlaisuudesta, tarvitaan opetuskuvia erittäin paljon ennen kuin päästään hyvään tarkkuuteen, verrattuna luokitteluongelmiin, jossa luokiteltavissa kuvissa on selviä usein toistuvia eroja.

Sinänsä se, että tunnistaminen ei onnistunut luotettavasti ei ole kovinkaan iso yllätys, sillä tälläkin hetkellä moni yritys, joilla on iso budjetti, suunnittelee vastaavaa, tekoälyyn perustuvaa syöpäluomien tunnistinta ja siltikään mitään toimivaa tuotetta ei ole tullut markkinoille. Jos näillä resursseilla, joilla opinnäytetyötä tehtiin, olisi saanut aikaan luotettavan tunnistusjärjestelmän, olisi se ollut maailmanluokan sensaatio.

Vaikka itse kokeellinen neuroverkko ei toiminut, niin kuin alun perin oli tarkoitus, opetti työn tekeminen aihealueesta todella paljon ja työtä oli hyvin mielenkiintoista tehdä.

LÄHTEET

Brownlee, J. (2017). How Much Training Data is Required for Machine Learning? Haettu 25.5.2020 osoitteesta https://machinelearningmastery.com/much-training-data-required-machine-learning/

Brownlee, J. (2019). A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. Haettu 25.5.2020 osoitteesta https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

council of europe. (2020). History of Artificial Intelligence. Haettu 25.5.2020 osoitteesta https://www.coe.int/en/web/artificial-intelligence/history-of-ai

Gollapudi, S. (2016). Practical Machine Learning. Birmingham: Packt Publishing.

Gulli, A. (2017). Deep Learning with Keras. Birmingham: Packt Publishing.

Heaven, D. (2019). Why deep-learning AIs are so easy to fool. Haettu 25.5.2020 osoitteesta https://www.nature.com/articles/d41586-019-03013-5

Hemanth, D. & Estrela, V. V. (2017). Deep Learning for Image Processing Applica-tions. Amsterdam: IOS Press.

keras. (n.d.). Haettu 25.5.2020 osoitteesta https://keras.io/

Kharkovyna, O. (2019). A Comprehensive Guide to Neural Networks for Beginners. Haettu 25.5.2020 osoitteesta https://towardsdatascience.com/a-comprehensive-guide-on-neural-networks-for-beginners-a4ca07cee1b7

Lewis, T. (2014). A Brief History of Artificial Intelligence. Haettu 25.5.2020 osoitteesta https://www.livescience.com/49007-history-of-artificial-intelligence.html

mathworks. (n.d.). What is Deep Learning? Haettu 25.5.2020 osoitteesta https://se.mathworks.com/discovery/deep-learning.html

Mahapatra, S. (2018). Why Deep Learning over Traditional Machine Learning? Haettu 25.5.2020 osoitteesta https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063

Mazur, M. (2015). A Step by Step Backpropagation Example. Haettu 25.5.2020 osoitteesta https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

microsoft. (2020). Deep learning vs. machine learning. Haettu 25.5.2020 osoitteesta https://docs.microsoft.com/en-us/azure/machine-learning/concept-deep-learning-vs-machine-learning

Ognjanovski, G. (2019). Everything you need to know about Neural Networks and Backpropagation. Haettu 25.5.2020 osoitteesta https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

Pietikäinen, M. & Silvén, O. (2019). Tekoälyn haasteet-koneoppimisesta ja konenäöstä tunnetekoälyyn. Oulu: Konenäön ja signaalianalyysin keskus. Oulun yliopisto.

Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks. Haettu 25.5.2020 osoitteesta https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Sharma, H. (2019). Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax ba-sics for Neural Networks and Deep Learning. Haettu 25.5.2020 osoitteesta https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e

SuperDataScience. (2018). Convolutional Neural Networks (CNN): Step 3 - Flattening. Haettu 25.5.2020 osoitteesta https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

tensorflow. (n.d.). Haettu 25.5.2020 osoitteesta https://www.tensorflow.org/

Theophano, M. (2019). How Do You Know You Have Enough Training Data? Haettu 25.5.2020 osoitteesta https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee

Wiley, J. F. (2016). R Deep Learning Essentials. Birmingham: Packt Publishing.

Zaccone, G. Karim, R. & Menshawy, A. (2017). Deep Learning with TensorFlow. Bir-mingham: Packt Publishing.

Zocca, V. Spacagna, G. Slater, D. & Roelants, P. (2017). Python Deep Learning. Bir-mingham: Packt Publishing Ltd.

Liite 1

OHJELMAKOODI

```
import numpy
import h5py
import tensorflow as tf
import keras

from keras import Sequential
from keras.layers import Dense, Dropout, Flatten, BatchNormalization, Activation
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
from keras.layers import Input, Dense
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K

#Määritellään asetuksia näytönohjainta varten
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config)

# Määritellään kuvien koko neuroverkkoa varten
img_width, img_height = 256, 256

# Määritellään datakansiot jossa kaikki kuvat ovat
train_data_directory = 'C:/lopputyo/data/train'
validate_data_directory = 'C:/lopputyo/data/validate'

#Määritellään kuvien määrät
nb_train_samples = 5580
nb_validation_samples = 1395
epochs = 10
batch_size = 1

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


#Mallin luominen
model = Sequential()
model.add(Conv2D(16, (7, 7), padding="valid", input_shape=input_shape))
```

```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis = -1))
model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), padding ="same"))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis = -1))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding ="same"))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization(axis = -1))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128))
model.add(Activation('relu'))
model.add(BatchNormalization(axis = -1))
model.add(Dropout(0.5))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
          optimizer='rmsprop',
          metrics=['accuracy'])

# Sovelletaan kuviin automaattisia muokkauksia
train_datagen = ImageDataGenerator(
   rescale=1. / 255,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True)

# Validointikuviin käytetään vain skaalausta
validate_datagen = ImageDataGenerator(rescale=1. / 255)

#Opetuskuvien lataus
train_generator = train_datagen.flow_from_directory(
   train_data_directory,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='binary')
```

```
#Validointikuvien lataus
validation_generator = validate_datagen.flow_from_directory(
   validate_data_directory,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='binary')

#Mallin opetus
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)

#Lasketaan mallin tarkkuus
scores = model.evaluate_generator(validation_generator, steps=5)
print("Accuracy: %.2f%%" % (scores[1]*100))

#Tallennetaan malli
model.save("testimalli.h5")
```