Tunteiden tunnistaminen tietokonenäön perusteella

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, insinööri (AMK), Riihimäen kampus

Kevät, 2021

Jani Liiti

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka Tiivistelmä

Riihimäki

Tekijä	Jani Liiti	Vuosi 2021
Työn nimi	Tunteiden tunnistaminen tietokonenäön perusteella	
Ohjaaja	Petri Kuittinen	

TIIVISTELMÄ

Tämän opinnäytetyön tavoitteena oli selvittää, miten ihmisten kasvonilmeillään osoittamia tunteita voidaan tunnistaa käyttäen tietokonenäköä, syväoppimista ja konvoluutioneuroverkkoja. Opinnäytetyö koostuu teoriaosuudesta ja käytännön projektista.

Työssä käytiin läpi tekoälyn, tietokonenäön, sekä tunteiden tunnistamisen teoriaa. Teoriaosuudessa tutustuttiin tarkemmin myös konvoluutioneuroverkkojen toimintaan, sekä tunteiden tunnistamisen käyttökohteisiin. Lopuksi tehtiin käytännön projekti, jossa luotiin järjestelmä, jolla ihmisen kasvonilmeillään osoittamia tunteita voidaan tunnistaa reaaliajassa käyttäen web-kameraa.

Järjestelmässä käytettiin OpenCV-ohjelmistokirjastoa tietokonenäön toteuttamiseen, ja TensorFlow-koneoppimisalustaa, sekä Keras-ohjelmointirajapintaa tunteiden tunnistamiseen. Tulokseksi saatiin hyvin toimiva järjestelmä, joka pystyy tunnistamaan tunteita optimaalisissa kuvausolosuhteissa melko tarkasti ottaen huomioon aiheen monimutkaisuuden.

Avainsanat Tietokonenäkö, tekoäly, syväoppiminen, tunteet

Sivut 31 sivua ja liitteitä 6 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Abstract

Author Jani Liiti
Year 2021
Subject Identifying Emotions with Computer Vision
Supervisor Petri Kuittinen

ABSTRACT

The aim of this thesis was to find out how the emotions a person reflects on their face can be recognized using computer vision, deep learning, and convolutional neural networks. The thesis consists of a theory part and a project.

The thesis studies theory of artificial intelligence, computer vision and emotion recognition. The theory part of this thesis also further explores how convolutional neural networks function and how emotion recognition is used in real life situations. In the end a project was conducted to create a system for recognizing the emotions a person conveys with their facial expressions in real-time using a webcam.

The system was created using OpenCV software library for computer vision, and TensorFlow machine learning platform and Keras deep learning API for the emotion recognition. The end result was a system that can recognize emotions in optimal filming conditions fairly accurately. The result was satisfactory considering the complexity of the subject.

Keywords Computer vision, artificial intelligence, deep learning, emotion
Pages 31 pages and appendices 6 pages

Sisällys

Kuvat, taulukot ja kaavat

Liitteet

1 Johdanto

Tämän opinnäytetyön tavoitteena on selvittää, miten ihmisen ilmaisemia tunteita on mahdollista tunnistaa käyttäen tietokonenäköä, syväoppimista ja konvoluutioneuroverkkoja. Työssä keskitytään tunteiden tunnistamiseen kasvonilmeistä. Kasvonilmeitä tunnistavia järjestelmiä on jo jonkin verran käytössä, esimerkiksi älypuhelimien kamerasovelluksissa.

Tekoälyn kehitys on kiihtynyt huomattavasti viime vuosikymmenellä, johtuen laajalti prosessointitehon nopeasta kehittymistahdista ja saatavilla olevan opetusdatan nopeasta lisääntymisestä. Tietokonenäköä käytetään maailmalla paljon muun muassa videovalvontajärjestelmissä, teollisuudessa ja robotiikassa. Erilaisten tekoälyratkaisujen kehittyessä tulevat tietokonenäön mahdolliset käyttökohteet laajenemaan entistä enemmän.

Nykypäivänä ihmisistä pyritään keräämään hyvin paljon erilaista tietoa, jotta heille voidaan tarjota tietoa heitä kiinnostavista aiheista tai markkinoida tiettyjä tuotteita. Tunteiden tunnistamista käyttäen voitaisiin tulevaisuudessa kerätä tietoa ihmisten reaktioista esimerkiksi kadulla vastaan tulevaan mainokseen. Täten yritykset saisivat alueellista dataa esimerkiksi siitä, miten ihmiset reagoivat tietyn tyyppiseen markkinointiin ilman, että yritysten täytyisi tehdä erillisiä kyselyitä.

2 Tekoäly

Tekoälyksi voidaan Euroopan komission määritelmän mukaan kuvailla järjestelmää, joka osoittaa älykkyyttä vaativaa käyttäytymistä analysoimalla ympäristöään ja tekemällä jollain tasolla autonomisia toimintoja saavuttaakseen tietyn päämäärän (Boucher, 2020, s. 3). Tekoälyn tutkimukseen on käytössä kaksi eri lähestymistapaa. Symbolinen tekoäly ja dataohjattu tekoäly. (Pietikäinen & Silvén, 2019, s. 5)

Symbolisen tekoälyn ajatuksena on, että jotakin asiaa, ilmiötä tai ominaisuutta voidaan kuvata symboleilla. Symbolisen tekoälyn toiminta perustuu laajoihin tietämyskantoihin ja sääntöihin siitä, miten tarkasteltavissa olevaan aiheeseen liittyvä maailma toimii. Järjestelmä vertaa tunnistamiaan asioita ja ominaisuuksia sen tietämyskannasta löytyviin käsitteisiin ja niiden suhteisiin, ja tekee niiden perusteella päätelmän. Symbolisen tekoälyn opettaminen on hyvin työlästä, sillä säännöt ja tiedot täytyy usein lisätä manuaalisesti. Positiivista symbolisessa tekoälyssä on, että kaikki sen osat ovat muodossa, joka ihmisen on helppo ymmärtää, mikä helpottaa vianhakua ja jatkokehitystä. (Pietikäinen & Silvén, 2019, ss. 6, 48)

Dataohjattu tekoäly perustuu keinotekoisiin neuroverkkoihin. Keinotekoinen neuroverkko koostuu laskentaelementeistä, jotka jäljittelevät ihmisen neuroneita. Näistä keinotekoisista neuroneista muodostetaan ihmisen hermokudosta muistuttava neuroverkko. Dataohjattu tekoäly pystyy oppimaan ja oppiminen on mahdollista automatisoida. Oppimisen ollessa automatisoitua virheiden jäljittäminen muuttuu kuitenkin hyvin vaikeaksi. Jotta dataohjatun tekoälyn on mahdollista antaa oikeita tuloksia, on oppimisvaiheessa yleensä oltava valtava määrä dataa. (Pietikäinen & Silvén, 2019, ss. 6, 48)

2.1 Tekoälyn historia

Alan Turing oli ensimmäinen ihminen, joka nosti esiin älykkään koneen luomisen vuonna 1950 julkaisemassaan artikkelissa "Computing Machinery and Intelligence". Artikkelissa Turing kuvailee kuuluisaa ”Turingin testiä”, jossa ihmisen täytyy erottaa keskusteleeko hän ihmisen vai koneen kanssa. (Council of Europe, n.d.) Testin mukaan, jos ihminen ei huomaa

keskustelevansa koneen kanssa, voidaan kone luokitella älykkääksi. Monien asiantuntijoiden mielestä testi ei kuitenkaan ole hyvä älykkyyden mittari. (Lewis, 2014)

Tekoälyn tieteenhaaran katsotaan syntyneen vuoden 1956 kesällä pidetyssä konferenssissa, johon oli kutsuttu paljon eri alojen parhaita tutkijoita keskustelemaan tekoälystä (Council of Europe, n.d.; Rockwell, 2017). Vaikka kaikki tutkijat eivät osallistuneet konferenssiin kovin aktiivisesti, eikä tieteenhaaran standardeista päästy yhteisymmärrykseen, olivat tutkijat kuitenkin yhtä mieltä siitä, että tekoäly on mahdollista toteuttaa (Rockwell, 2017).

Vuodesta 1957 eteenpäin tekoälyn kehitys kukoisti. Tietokoneet kehittyivät ja niiden käytöstä tuli halvempaa. Myös koneoppimis-algoritmit kehittyivät ja ihmiset oppivat käyttämään niitä tehokkaammin. Ennen pitkää kehitys kuitenkin hidastui huomattavasti. Tietokoneissa ei vielä ollut tarvittavia määriä tallennustilaa, eivätkä ne pystyneet prosessoimaan dataa tarpeeksi nopeasti. Pian rahoitukset alkoivat pysähtyä ja täten tutkimus hidastui. (Rockwell, 2017) Tekoälyn kehitys oli täten osoittautunut huomattavasti odotettua vaikeammaksi ja vuosina 1974–1980 elettiin ajanjaksoa, joka tunnetaan nimellä ”AI-talvi” eli tekoälytalvi (Lewis, 2014).

Tekoälyn kehitys lähti takaisin nousuun, kun ensimmäiset mikroprosessorit tulivat käyttöön 1970-luvun lopussa. Tällöin kehityksen keskiössä olivat asiantuntijajärjestelmät mitkä jäljittelevät ihmisen loogista päättelykykyä. Asiantuntijajärjestelmien kehityshuuma kesti noin 1980-luvun loppuun. Järjestelmien ohjelmoiminen oli hyvin työlästä ja niiden toimintalogiikka oli osittain epäselvä. Kehittäminen ja virheiden korjaaminen oli täten hankalaa. Samaan aikaan nopeampia ja halvempia, sekä monilla tavoin yksinkertaisempia järjestelmiä oli jo mahdollista toteuttaa. 1990-luvulla tekoälyn kehitys oli niin heikkoa, että termiä tekoäly ei enää oikein käytetty. (Council of Europe, n.d.)

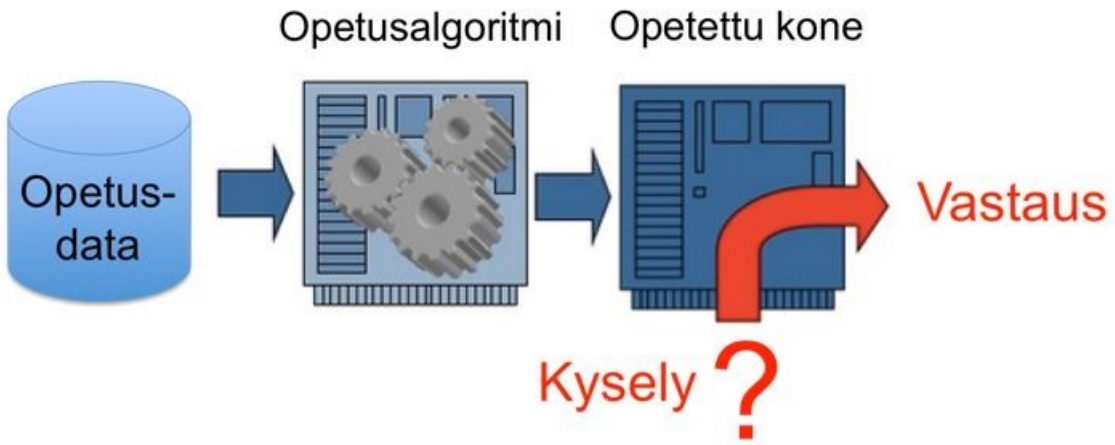
Vuodesta 2010 eteenpäin tekoälyn kehitys on taas kiihtynyt. Nykyään valmiiksi käyttökelpoista dataa on saatavilla valtavia määriä, mikä mahdollistaa esimerkiksi kuvien luokittelualgoritmien luomisen ilman, että kuvanäytteitä tarvitsee ottaa itse. Valtavat datamäärät ovat myös mahdollistaneet sen, että nykyään tekoälyn annetaan oppia asioita itse sen sijaan, että kaikki määriteltäisiin säännöillä, niin kuin vanhemmissa asiantuntijajärjestelmissä. Toinen tärkeä kehitys on, että tietokoneiden näytönohjaimia

voidaan käyttää oppimisalgoritmien laskelmoinnissa, mikä on nopeuttanut prosessia huomattavasti. 2000-luvulla syväoppiminen nousi keskeiseksi tekniikaksi varsinkin tekstin, äänen, ja kuvan tunnistuksessa. (Council of Europe, n.d.)

2.2 Koneoppiminen

Koneoppiminen on tekoälyn osa-alue, jossa luodaan analyyttisiä malleja data-analytiikan avulla. Mallien tehtävänä on oppia ja tunnistaa yhtäläisyyksiä mahdollisimman autonomisesti. Koneoppiminen syntyi ajatuksesta, että tietokoneet voisivat oppia suorittamaan tiettyjä toimintoja ilman niiden varsinaista ohjelmointia. (SAS, n.d.-b) Koneoppimisen algoritmit opetetaan löytämään yhtäläisyyksiä ja ominaisuuksia suuresta datamäärästä, jotta kun algoritmille syötetään uutta dataa, se pystyy tehdä ennusteita aikaisemmin oppimiensa asioiden perusteella. Koneoppimisalgoritmien tarkkuus päätöksenteossa paranee sitä mukaa, kun algoritmi käsittelee enemmän dataa. Kuvassa 1 on kuvattu koneoppimisen toimintaperiaatetta. (IBM Cloud Education, 2020a)

Kuva 1. Koneoppimisen toimintaperiaate (Apell, 2020).



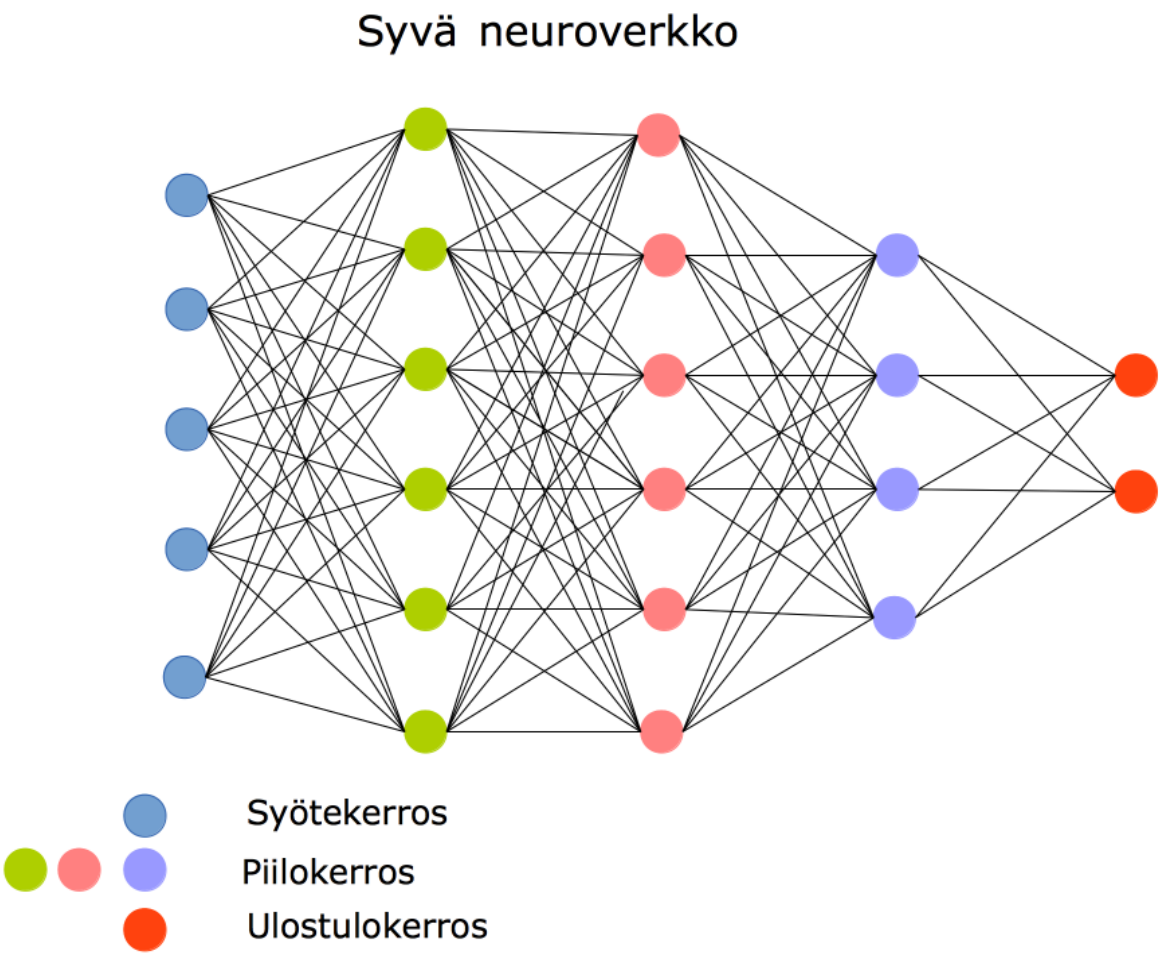
Koneoppimisalgoritmien opetustyylejä on neljä: Ohjattu koneoppiminen, ohjaamaton koneoppiminen, osittain ohjattu koneoppiminen ja vahvistusoppiminen. Näistä käytetyimpiä ovat ohjattu ja ohjaamaton koneoppiminen. Ohjatussa oppimisessa mallia opetetaan syöttämällä sille valmiiksi luokiteltua dataa, jotta malli voi verrata ennustustaan oikeaan

vastaukseen ja korjata virheitä. Ohjaamattomassa oppimisessa mallille ei kerrota oikeita vastauksia, vaan sen on selvitettävä ne täysin itse. Osittain ohjatussa oppimisessa käytetään sekä luokiteltua, että luokittelematonta dataa. Vahvistusoppimisessa algoritmia ei opeteta käyttäen esimerkkidataa, vaan algoritmi pyrkii löytämään parhaat tulokset tuottavat ratkaisut ”Yritys ja erehdys”-periaatteella. (SAS, n.d.-b; ks. myös IBM Cloud Education, 2020a)

2.3 Syväoppiminen

Syväoppiminen on tekoälyn osa-alue, jossa luodaan laajoja neuroverkkomalleja, jotka kykenevät tekemään dataohjattuja päätöksiä tutkimalla niille syötettyä dataa ja arvioimalla parhaan lopputuloksen (Kelleher, 2019, ss. 14–17). Neuroverkkomallit koostuvat useista neuroverkkokerroksista, jotka jäljittelevät ihmisaivojen rakennetta (Kelleher, 2019, ss. 78–81). Ensimmäinen kerros on aina syötekerros, jonka jälkeen tulee yksi tai useampi piilokerros. Silloin kun piilokerroksia on useampia, puhutaan syvästä neuroverkosta. Verkon viimeinen kerros on aina ulostulokerros. Syvän neuroverkon rakennetta on kuvattu kuvassa 2. (Tuominen & Neittaanmäki, 2019) Neuroverkkomallien tiedonkäsittelytapoja ei tarvitse erikseen määritellä, vaan mallit määrittelevät ne itse. Syväoppiminen jäljittelee huomattavasti ihmisen luontaista oppimistapaa. (Suomen koodikoulu, 2018, s. 23)

Kuva 2. Syvän neuroverkon rakenne (Tuominen & Neittaanmäki, 2019).



Syväoppimista käytetään etenkin, kun tulkittava data on monimutkaista ja kun aiheeseen liittyvää tietoa tai materiaalia on laajasti saatavilla. Syväoppimista käytetäänkin nykypäivänä hyvin laajasti. Kaikki älypuhelimet käyttävät syväoppimista muun muassa puheen sekä kasvojen tunnistukseen. Syväoppimista käytetään myös paljon terveydenhoitoalalla erilaisten kuvien prosessointiin ja diagnoosien tekemiseen. (Kelleher, 2019, ss. 14–17)

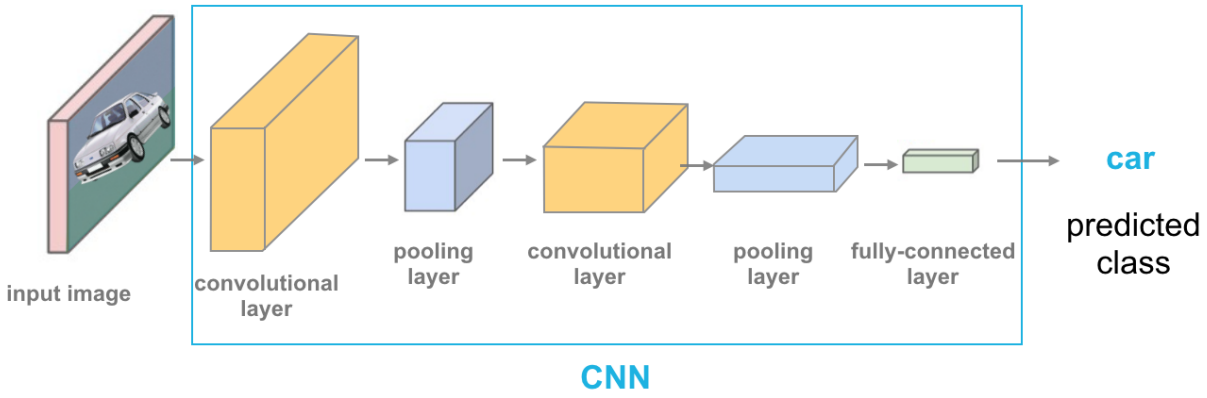
Yksi suurimpia saavutuksia syväoppimisen saralla oli, kun DeepMind-yrityksen kehittämä AlphaGo-tekoäly voitti vuosina 2016 ja 2017 kaksi Go-pelin parhaimmistoon kuuluvista pelaajista. Tekoälyn ei ollut odotettu oppivan peliä huipputasolla vielä useaan vuoteen, Go-pelin ollessa paljon monimutkaisempi peli kuin shakki, jossa DeepBlue-tekoäly päihitti shakin maailmanmestarin jo vuonna 1997. Huomion arvoista on, että shakkiohjelmien kehitys normaalin pelaajan tasolta maailmanmestaritasolle kesti 30-vuotta, kun Go-ohjelmien kesti

vain seitsemän vuotta. Tämä on osoitus siitä, miten suuren kehityksen tekoälyssä syväoppiminen on mahdollistanut. (Kelleher, 2019, ss. 14–20)

2.4 Konvoluutioneuroverkot

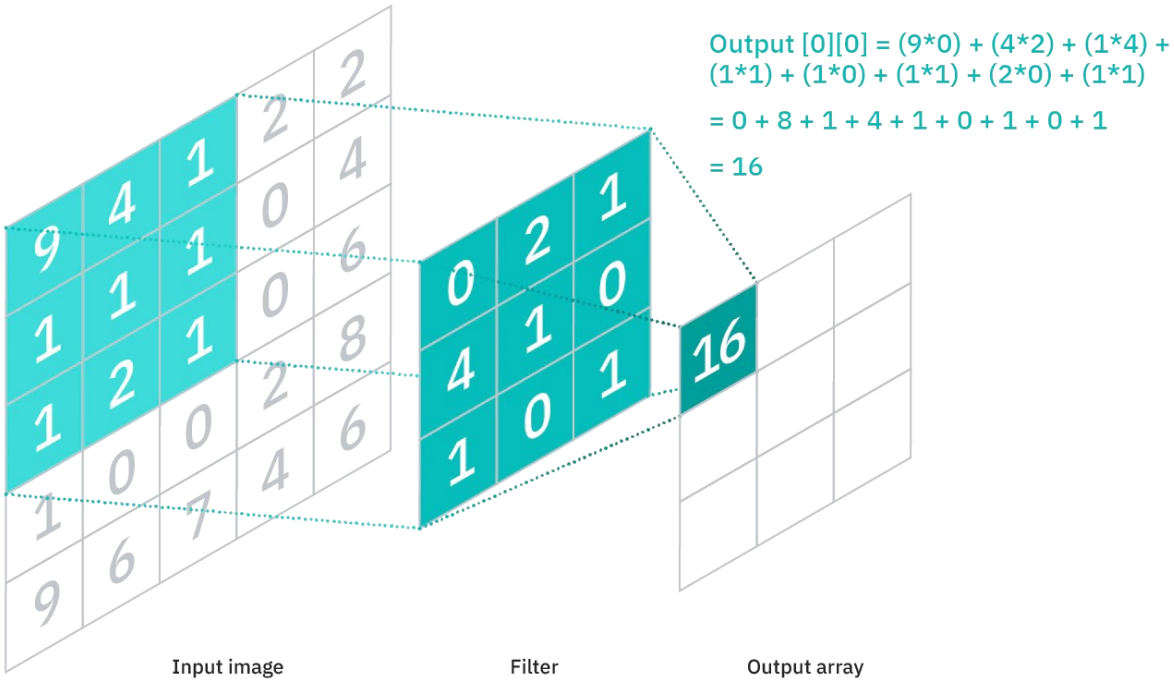
Konvoluutioneuroverkot ovat neuroverkkotyyppi, jota käytetään etenkin kuvien, puheen ja äänisignaalien käsittelyssä. Konvoluutioneuroverkkoon kuuluu kolmen tyyppisiä kerroksia. Verkko alkaa aina konvoluutiokerroksella (Convolutional layer), välissä on alinäytteistyskerroksia (Pooling layer) ja viimeisenä on aina täysin kytketty kerros (Fully-connected layer). Kaikkia kerroksia voi olla verkossa useita. (IBM Cloud Education, 2020b) Konvoluutioneuroverkon rakennetta on kuvattu kuvassa 3.

Kuva 3. Konvoluutioneuroverkon rakenne (Camacho, 2018).



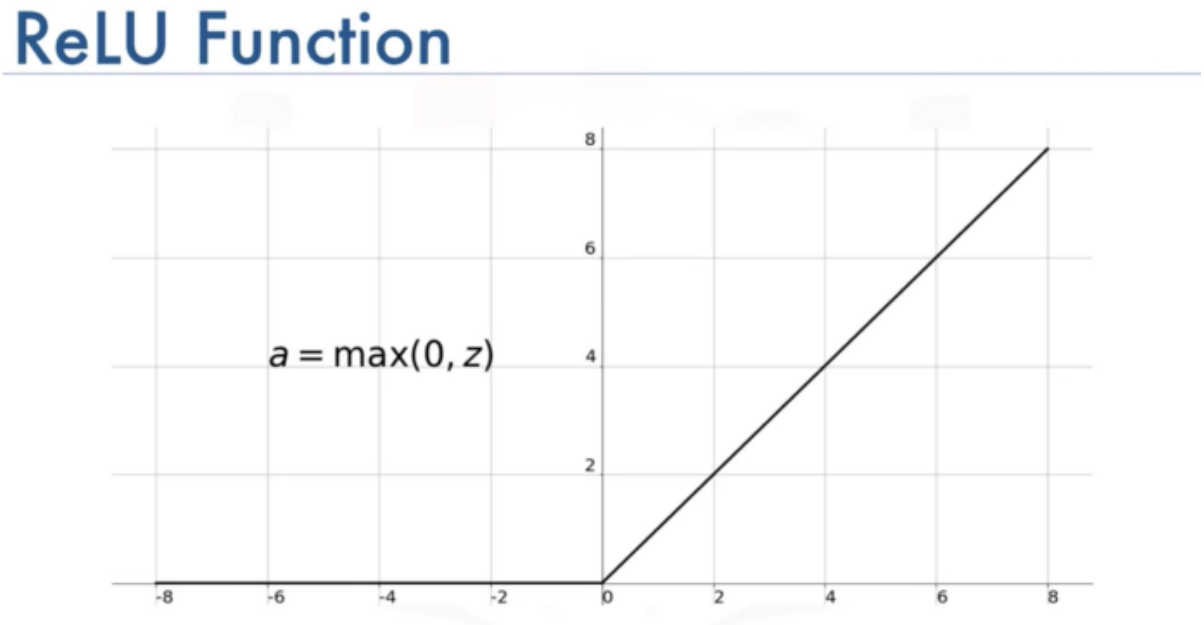
Konvoluutiokerros on konvoluutioneuroverkon keskeisin osa. Kerros koostuu kolmesta osasta. Ensimmäinen osa on kerrokselle syötetty data, esimerkiksi kuva. Toisena on suodatin, jolla data käydään läpi etsien tiettyjä piirteitä, joita datasta halutaan löytää. Tätä piirteiden etsintä prosessia kutsutaan konvoluutioksi. Kuvassa 4 näkyy, miten suodatinta käyttäen syötetty kuva käsitellään alue kerrallaan, palauttaen yhdeltä alueelta aina yhden arvon. Konvoluutiokerroksen kolmas osa on piirrekartta, johon on merkitty sijainnit löydetyille piirteille. (IBM Cloud Education, 2020b)

Kuva 4. Konvoluutiokerros (IBM Cloud Education, 2020b).



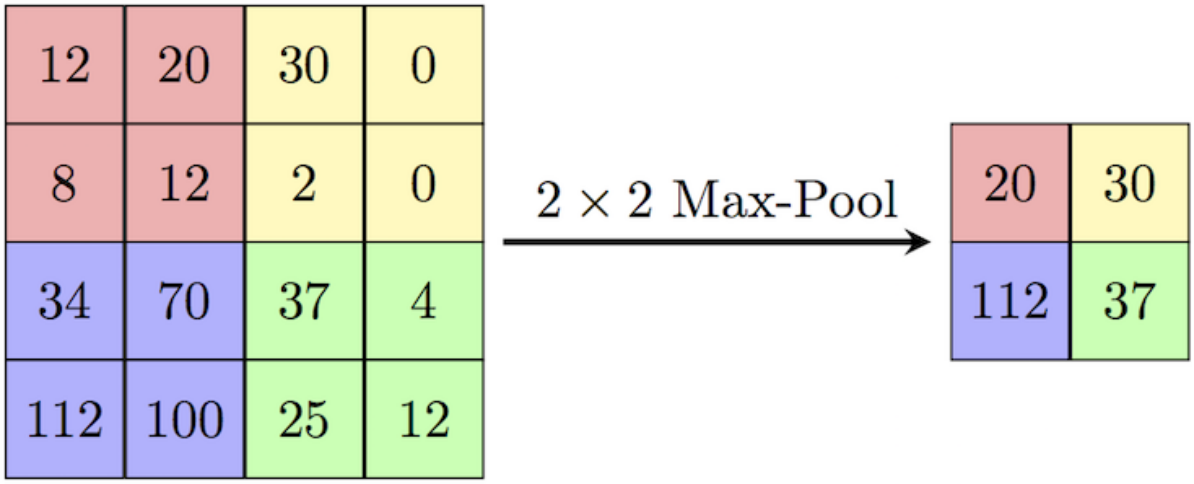
Jokaisen konvoluutio-operaation jälkeen piirrekarttaan käytetään ReLU-aktivointifunktiota, jolla verkkoon saadaan epälineaarisuutta (IBM Cloud Education, 2020b). Yksinkertaisuudessaan ReLU-funktio muuttaa kaikki piirrekartan negatiiviset arvot nolliksi (Kelleher, 2019, ss. 178–180). ReLU-funktion kaava ja käyrä on esitetty kuvassa 5. Konvoluutiokerroksen päätehtävänä on muuttaa syötetty data numeerisiksi arvoiksi, jotta neuroverkko voi tulkita sitä ja erotella olennaiset osat (IBM Cloud Education, 2020b). Käyttäen useita suodattimia tai konvoluutiokerroksia voidaan erotella useita eri piirteitä (Kelleher, 2019, ss. 180–184).

Kuva 5. ReLU-funktion kaava ja käyrä (Toprak, 2020).



Pooling- eli alinäytteistyskerroksia käytetään piirrekarttaan, johon on jo käytetty aktivointifunktiota (Kelleher, 2019, ss. 178–180). Alinäytteistyksessä piirrekartta käydään läpi yleensä 2x2-pikselijoukko kerrallaan, joista palautetaan aina yksi arvo. Arvo voi olla joko lukujen keskiarvo (Average Pooling) tai yleisemmin tarkastellun alueen suurin luku (Max Pooling). Esimerkki Max Poolingista kuvassa 6. (Brownlee, 2019) Alinäytteistyksellä saadaan vähennettyä parametrien määrää ja monimutkaisuutta, jotta verkon toiminta on tehokkaampaa (IBM Cloud Education, 2020b). Alinäytteistyksessä poistetaan usein piirteiden sijaintitiedot, jotta piirre pystyttäisiin tunnistamaan sen sijainnista huolimatta (Kelleher, 2019, ss. 175–178).

Kuva 6. Max Pooling (Computer Science Wiki, n.d.).



Täysin kytketty kerros on nimensä mukaisesti kerros, jonka kaikki keinotekoiset neuronit ovat yhteydessä kaikkiin edellisen kerroksen neuroneihin. Täysin kytketyssä kerroksessa edellisistä kerroksista ja suodattimista poimittujen piirteiden perusteella päätetään mihin luokkaan käsiteltävä data kuuluu. Konvoluutioneuroverkossa dataa käsitellään siis pieni osa kerrallaan, joista poimitaan löydettyjä ominaisuuksia. Myöhemmin kaikkien tunnistettujen ominaisuuksien pohjalta pystytään luoda kokonaiskuva ja tunnistaa mihin luokkaan data kuuluu. (IBM Cloud Education, 2020b)

3 Tietokonenäkö

Tietokonenäkö on tekoälyn osa-alue, jossa pyritään jäljittelemään ihmisen näkökykyä. Tarkoituksena on, että kone pystyy näkemisen lisäksi myös ymmärtämään näkemäänsä. (Suomen koodikoulu, 2018, s. 29)

3.1 Historia

Tietokonenäön osa-alueen tutkinta alkoi 1960-luvulla. Tällöin ihmisen näkökyvyn monimutkaisuutta aliarvioitiin vielä huomattavasti. Vuonna 1966 ajateltiin, että koulun kesäprojektina pystytään toteuttamaan järjestelmä, jonka tuloksena tietokone pystyy kuvailemaan, mitä se näkee. Näkökyky osoittautui kuitenkin yhdeksi suurimmista haasteista

tekoälyn alueella. (Planche & Andres, 2019, A brief history of computer vision; Motion Metrics, 2019)

Tietokonenäön perusidea on pysynyt lähtökohtaisesti samana alusta lähtien. Tarkoituksena on jäljitellä ihmisen näkökykyä ja havainnointia, erotellen kuvasta ensin merkityksellisiä piirteitä ja verraten niitä jo valmiiksi tunnettuihin piirteisiin ja täten tunnistaa mitä kuvassa näkyy. Tutkimus keskittyi aluksi kuvasta löytyvien rajojen ja viivojen tunnistamiseen, joilla saatiin luotua käsitys esimerkiksi tilan geometrisestä rakenteesta. Myöhemmin mukaan tulivat tekstuurien ja valoisuuden käsittely, mikä johti ensimmäisiin objektien luokittelujärjestelmiin. (Planche & Andres, 2019, A brief history of computer vision)

Tietokonenäön kehittyessä monia erilaisia tekniikoita otettiin käyttöön, mutta kaikissa oli sama toimintaperiaate. Poimittuja piirteitä verrattiin toisten kuvien piirteisiin. Pian kuitenkin huomattiin, että piirteiden tunnistaminen on vain osa työstä. Kahdesta samaan luokkaan kuuluvasta kuvasta tunnistetut piirteet voivat näyttää hyvin erilaisilta, esimerkiksi koiria vertailtaessa. 1990-luvulla luokitteluun alettiin soveltaa koneoppimista, ja kehitys pääsi etenemään. (Planche & Andres, 2019, A brief history of computer vision)

Nykyään tietokonenäössä käytetään enimmäkseen syväoppimista ja neuroverkkoja. Neuroverkkojen käyttöä tutkittiin ensimmäistä kertaa jo 1950-luvulla, konvoluutioneuroverkkojen syntyessä 1980-luvun alkupuolella. Konvoluutioneuroverkot olivat kuitenkin tällöin vielä laskennallisesti liian raskaita. 2000-luvulla, kun tietokoneet alkoivat olla tarpeeksi tehokkaita ja kuva- ja videodataa oli runsaasti saatavilla, oli syväoppimista mahdollista käyttää tehokkaasti. (Planche & Andres, 2019, A brief history of computer vision)

3.2 Käyttökohteet

Tietokonenäköä käytetään kulujen laskemiseksi, turvallisuuden parantamiseksi ja käyttökokemuksen kehittämiseksi muun muassa kaupan, teollisuuden, autoteollisuuden ja terveydenhuollon aloilla (SAS, n.d.; Faggella, 2020). Kaupan alalla tietokonenäköä voidaan käyttää hävikin minimoimiseen ja valvomaan tuotteiden hyllymääriä. Tietokonenäön avulla voidaan tunnistaa varkausyrityksiä ja epäilyttävästi käyttäytyviä ihmisiä. Amazonin Amazon

Go-kaupassa asiakkaan ei tarvitse skannata tuotteita, vaan asiakas vain kerää tuotteet mukaansa ja järjestelmä tunnistaa kameroiden avulla mitä tuotteita asiakas otti ja laskuttaa ne automaattisesti asiakkaan poistuessa kaupasta. (Saik, 2020)

Teollisuudessa tietokonenäköä käytetään muun muassa laadunvalvontaan ja järjestelmien ylläpitoon. Tietokonenäköjärjestelmällä voidaan tunnistaa laitteistovikoja ennen kuin ne aiheuttavat mittavia vahinkoja tai kuluja. Samalla tavalla tuotteissa olevat valmistusviat voidaan tunnistaa automaattisesti ilman ihmisen läsnäoloa. Tietokonenäköä käytetään myös lukemaan viivakoodeja, käsin kirjoitettuja merkintöjä ja täytettyjä lomakkeita, joiden avulla voidaan esimerkiksi ohjata tuotteita eri linjoille. (USM, 2020)

Uusimmissa autoissa tietokonenäköjärjestelmiä on jo hyvin laajasti käytössä, ja ne ovat erittäin keskeisessä osassa itsestään ajavissa autoissa. Niiden avulla tunnistetaan muita tienkäyttäjiä, liikennemerkkejä ja kaistoja. Myös monissa aivan tavallisissa peruutuskameroissa käytetään tietokonenäköä varoittamaan lähestyvistä objekteista. (Brooke, 2020)

Terveydenhoitoalalla tietokonenäköä käytetään muun muassa poikkeavuuksien tunnistamiseen erilaisista kuvista, kuten röntgenkuvista, sekä verenvuodon tarkkailuun erilaisten kirurgisten operaatioiden aikana. Stanfordin yliopiston tekoälylaboratoriossa tutkijat loivat konvoluutioneuroverkkoja käyttämällä mallin, johon käytettiin 120 000-ihosyöpäkuvaa. Tuloksena oli malli, joka pystyi tunnistamaan ihosyövän yhtä tarkasti, kuin sertifioitu ihotautilääkäri. (Beatrice, 2020; ks. myös: Roth, 2019)

3.3 Haasteet

Tietokonenäön tavoitteena on jäljitellä ihmisen näkökykyä. Tietokonenäön tutkimuksen pääongelmana onkin, että emme laajoilta osin tiedä miten ihmisen näköaisti toimii. (Dawson-Howe, 2014, s. 2) Ongelmat eivät kuitenkaan ole rajoittuneet vain järjestelmien toimivuuteen, vaan myös eettisiin ongelmiin. Jos itsestään ajava auto joutuu kolariin, kuka on vastuussa? Tietokonenäköä käytetään myös paljon terveydenhuollon alalla. Jos järjestelmän tehtävänä on tunnistaa syöpäsoluja, kuka on vastuussa, jos järjestelmä

epäonnistuu? Tästä syystä järjestelmät toimivat toistaiseksi pääasiassa ihmisten tukena. (Dawson-Howe, 2014, s. 5)

Yksi hyvin keskeinen ongelma on myös ihmisten oikeus yksityisyyteen. Tässä eri maiden lait, yritysten säännöt, ja ihmisten mielipiteet vaihtelevat hyvin suuresti. Iso osa ihmisistä ei pidä ajatuksesta, että heitä voitaisiin tarkkailla jatkuvasti eri järjestelmien kautta. (Dawson-Howe, 2014, s. 5)

4 Tunteiden tunnistaminen

Kasvojen ilmeet ovat yksi keskeisimmistä ihmisen tunnetilaa kuvaavista ominaisuuksista, koska ne välittävät hyödyllistä tietoa ympärillä oleville. Ihmisen tunteet on jaettu kuuteen eri luokkaan. Nämä luokat ovat viha, suru, onnellisuus, inho, pelko ja yllättyneisyys. (Li ym., 2020)

Tunteiden tunnistaminen on aktiivinen tutkimusalue tietokonenäön alalla. Kone- ja syväoppimistekniikat ovat tuoneet älykkäät järjestelmät, jotka voivat tunnistaa tunteita, lähemmäs todellisuutta. Tutkimusten edetessä ongelma on kuitenkin osoittautunut erittäin monimutkaiseksi. Ihmiset osoittavat tunteitaan ilmeiden lisäksi muun muassa mikroilmeillä, eleillä, sekä äänensävyllä. Asiayhteydellä on myös usein merkitystä, kun pyritään tulkitsemaan ihmisen tuntemia tunteita. (Canedo & Neves, 2019, s. 1)

Tunteiden tunnistaminen nykyisillä kone- ja syväoppimisjärjestelmillä on ongelmallista myös siksi, että kuvien valoisuudet, taustat ja asennot vaihtelevat. (Li ym., 2020)

4.1 Toimintaperiaatteet

Kasvonilmeiden tunnistamisjärjestelmässä on yleisesti neljä osaa: Kuvan hankinta, kuvan esikäsittely, piirteiden erottelu ja luokittelu. Jotta luokittelu onnistuisi, on tärkeää, että luokittelussa käytetään kaikista olennaisinta ja mahdollisimman selkeää dataa. Tästä syystä järjestelmälle syötettyä kuvaa pitää yleensä esikäsitellä. Pääsääntöisesti syötetystä kuvasta eritellään ensimmäisenä kasvot, ja muu data poistetaan. Jo tämä vaihe saattaa tuottaa ongelmia hallitsemattomassa ympäristössä, jossa voi olla muun muassa liikettä,

epäoptimaalinen valaistus ja kameran epäoptimaalinen suuntaus suhteessa kohteeseen. (Canedo & Neves, 2019, ss. 1–2)

Kasvojen rajauksen jälkeen kuvalle voidaan tehdä vielä lukuisia muita esikäsittelyyn liittyviä toimenpiteitä. Normalisoinnilla voidaan säätää kuvan valoisuutta ja kohinasuodattimella voidaan poistaa kohinaa. Lisäksi voidaan tehdä muun muassa aineiston täydennystä, jossa kuvaan tehdään erilaisia muokkauksia. Kuvaa voidaan esimerkiksi pyörittää tai skaalata, se voidaan kääntää, tai sen kirkkautta voidaan muuttaa. Näin yhdestä kuvasta saadaan luotua monta hieman erilaista kuvaa, joilla saadaan lisättyä opetusdatan määrää. Esikäsittelyn jälkeen kuvasta erotellaan tiettyjä piirteitä, joiden avulla kuva pitäisi pystyä luokittelemaan. (Canedo & Neves, 2019, ss. 2, 8)

4.2 Käyttökohteet ja tulevaisuus

Ihmisten kasvonilmeiden tunnistamista käytetään jo hyvin monilla eri aloilla. Jopa FBI- ja TSA-agenteille on koulutettu tunteiden tunnistamista, jotta he pystyisivät paremmin tunnistamaan terroristeja. Tunteiden tunnistamista käytetään myös paljon psyykesairauksien diagnosoinnissa ja hoidossa. Autismipotilaita opetetaan usein tunnistamaan ihmisten valokuvissa osoittimia tunteita. Oikeudessa syytetyn osoittamat tunteet vaikuttavat huomattavasti tuomarin ja valamiehistön johtopäätöksiin siitä, onko syytetty syyllinen ja katuuko hän. (Barret ym., 2019, ss. 2–3)

Tunteiden tunnistamisjärjestelmien kehittyessä alkaa niiden käyttö lisääntyä edellä mainituissa, sekä täysin uusissa käyttötarkoituksissa. Ihmiset käyttävät paljon erilaisia teknologisia laitteita, kuten autoja, puhelimia, televisioita ja erilaisia kodinkoneita. Tunteiden tunnistaminen mahdollistaa kaikkien laitteiden käyttökokemuksen parantamisen. Jos laite pystyy tunnistamaan käyttäjän kokemia tunteita, se voi reagoida käyttäjän tarpeisiin automaattisesti. Joissain autoissa on jo esimerkiksi järjestelmiä, jotka tunnistavat, jos kuljettaja meinaa nukahtaa. Monet yritykset haluavat kaikin mahdollisin tavoin saada tietää tyydyttävätkö heidän tuotteensa, palvelunsa ja markkinointinsa heidän asiakkaitaan. Liiketoiminta-ala on erittäin kiinnostunut järjestelmistä, joiden avulla voitaisiin mitata

asiakkaan reagointia ja kiinnostusta käyttäen webkameroita. (Pilarczyk & Wieczorek, 2014, ss. 4–5)

4.3 Haasteet

Ihmisten osoittamien tunteiden tunnistaminen kasvoilta on saanut myös kritiikkiä. Psykologisen tieteen järjestö (Association for psychological science) julkaisi 2019 tutkimuksen, jossa tutkittiin tunteiden tunnistamisen haasteita tulkittaessa kasvonilmeitä. Tutkimuksen mukaan ihmisten tapa osoittaa eri tunteita vaihtelee merkittävästi muun muassa kulttuurin ja tilanteen mukaan. Tutkimuksessa todetaan, että esimerkiksi onnellisuutta ei voida todeta hymystä, eikä surullisuutta silmäkulmien kurtistuksesta. Silloin, kun kasvonliikkeet ilmaisevat tunnetilaa, niiden todetaan olevan huomattavasti vaihtelevampia ja riippuvaisia kontekstista, kuin nykyään käytössä olevan ajattelutavan mukaan. Tutkimuksen lopussa todetaan, että nykyisellä teknologialla ei voida tehdä johtopäätöksiä siitä, mitä ihmiset tuntevat pelkästään kasvonilmeiden perusteella. (Barret ym., 2019, ss. 1, 46, 48)

5 Käytetyt ohjelmistot

Kaikki projektissa käytetyt ohjelmistot olivat opinnäytetyön kirjoitushetkellä ilmaiseksi saatavilla. Ohjelmistojen valintaan vaikutti eniten, että miten hyvät ohjeet niiden käyttämiseen oli, ja oliko niillä rakennettu vastaavanlaisia järjestelmiä ennen. Valitut ohjelmistot vaikuttivat olevan yleisimmin käytettyjä, kun rakennetaan vastaavanlaisia järjestelmiä.

5.1 OpenCV

OpenCV on avoimen lähdekoodin ohjelmistokirjasto tietokonenäölle ja koneoppimiselle. Kirjasto sisältää yli 2500 tietokonenäön ja koneoppimisen algoritmia, joita käyttäen voidaan muun muassa tunnistaa kasvoja, objekteja, ja liikkeitä, sekä seurata liikkuvia kappaleita. OpenCV:llä on yli 47 000-käyttäjää ja sitä on arvion mukaan ladattu yli 18 miljoonaa kertaa. Kirjastoa käytetään laajasti myös yrityksissä ja tutkimuksissa. Joitakin tunnettuja yrityksiä,

joissa ohjelmistokirjasto on käytössä ovat muun muassa Google, Microsoft, Intel, IBM, Sony ja Toyota. OpenCV:tä käytetään muun muassa tunkeutumisten tunnistamiseen Israelilaisissa videovalvontajärjestelmissä, kaivosvälineiden seurantaan Kiinassa, uima-altailla hukkumisten tunnistamiseen Euroopassa, sekä jätteiden tunnistamiseen kiitoteillä Turkissa. OpenCV on kirjoitettu C++-ohjelmointikielellä, mutta se tukee myös Pythonia, Javaa, sekä MATLABia. Tuettuja käyttöjärjestelmiä ovat Windows, Linux, Android, ja MacOS. (OpenCV, n.d.)

5.2 TensorFlow

TensorFlow on avoimen lähdekoodin koneoppimisalusta (TensorFlow, n.d.). Google loi alustan alun perin tutkijoiden ja kehittäjien koneoppimistutkimuksia varten. TensorFlow:n tarkoitus on yksinkertaistaa koneoppimisratkaisujen käyttöönottoa ja se tukee prosessoreita, näytönohjaimia, mobiililaitteita, sekä selaimia. Se sisältää myös paljon hyödyllisiä toimintoja koneoppimismallien luomiseen. TensorFlow-ohjelmat kirjoitetaan Pythonilla, mutta järjestelmä suorittaa komennot C++:lla. (Planche & Andres, 2019, Getting started with TensorFlow 2 and Keras)

5.3 Keras

Keras on TensorFlow:n päällä toimiva Pythonilla kirjoitettu ohjelmointirajapinta syväoppimisratkaisuja varten. Samoin kuin TensorFlow, myös Keras on suunniteltu niin, että kehitys ideasta lopputulokseen olisi mahdollisimman nopeaa ja yksinkertaista. Keras tukee prosessoreita, näytönohjaimia, sekä koneoppimiseen suunnattuja tensorisuorittimia. Valmiita Keras-malleja on mahdollista käyttää myös selaimessa tai mobiililaitteessa. (Keras, n.d.-a)

6 Projekti

Opinnäytetyön soveltavan osuuden tavoitteena oli selvittää, miten tunteiden tunnistaminen tietokonenäöllä voidaan käytännössä toteuttaa ja miten tarkasti tunteita pystytään tunnistamaan.

6.1 Ohjelmistoihin tutustuminen ja tiedonhaku

Ensimmäisenä täytyi päättää mitä ohjelmistoja, ja mitä kuvatietokantaa haluttiin käyttää. TensorFlow ja Keras osoittautuivat olevan hyvin yleisessä käytössä konvoluutioneuroverkkoja luotaessa. Niiden käyttöön oli myös saatavilla hyvät dokumentaatiot ja paljon erilaisia tutoriaaleja internetissä. Molempien käyttöönotto oli myös hyvin yksinkertaista ja vastaan tulleisiin ongelmiin oli suhteellisen helppoa löytää ratkaisut. Suurin työ ohjelmistojen toimimaan saamisessa oli NVIDIA:n cuDNN-kirjaston (CUDA Deep Neural Network) ja CUDA-ohjelmointirajapinnan asentamisessa, joita tarvitaan, että TensorFlow voi käyttää näytönohjainta ohjelmien suorittamiseen.

Kuvatietokantoja oli myös monia erilaisia saatavilla, joista osassa on jopa miljoonia kuvia. FER-2013–tietokanta osoittautui kuitenkin kaikkein sopivimmaksi tähän projektiin. FER-2013 julkaistiin 2013 pidettyä kilpailua varten, jossa osallistujien piti luoda malli, joka tunnisti kuvissa näkyvät tunteet mahdollisimman tarkasti. Osallistujien saavuttamat tulokset ovat julkisesti nähtävillä, mikä auttaa arviomaan tässä projektissa saavutettavia tuloksia. Tuloksien mittausta varten on omat testikuvat. FER-2013 koostuu 48x48-kokoisista mustavalkokuvista. Kuvat oli valmiiksi jaettu opettamis- ja validointiosuuksiin ja niitä oli sopiva määrä, jotta mallien luominen onnistui järkevässä ajassa. Kuvat oli jaettu seitsemän eri tunteen mukaan, jotka olivat: Viha, inho, pelko, ilo, neutraali, suru ja yllättyneisyys. Opetuskuvia oli 28 709, ja validointikuvia 3 589.

Ennen ensimmäisten ohjelmaversioiden kirjoittamista tutustuttiin internetistä löytyviin muiden tekemiin vastaavanlaisiin projekteihin ja kokeiltiin joidenkin toimivuutta. Huomattavissa oli, että ohjelmissa käytettävät mallit vaihtelivat suuresti. Joissain malleissa ohjelma myös tunnisti neutraalin ilmeen helposti surullisena. Ennuste inhon ilmeestä oli kaikissa malleissa hankalin toteuttaa, joissakin se ei onnistunut ollenkaan. Tämä ei kuitenkaan ollut yllättävää, koska kaikille muille tunteille opetuskuvia oli tuhansia, kun inholle kuvia oli vain 436.

6.2 Ohjelmien kirjoittaminen

FER-2013–kuvatietokanta oli ladattavissa csv-tiedostona, jossa yhdellä rivillä oli aina yhden kuvan pikseliarvot, sen osoittama tunne ja kuuluiko se opetus-, validointi- vai testisarjaan. Ensimmäinen tehtävä oli siis luoda ohjelma, jolla kuvat saatiin eriteltyä omiin kansioihinsa. Tämän jälkeen kirjoitettiin ohjelma mallin luomiselle. Tässä vaiheessa luotiin vain hyvin yksinkertainen malli, jolla pystyttiin toteamaan ohjelman toimivuus. Kun ensimmäinen versio mallista oli saatu luotua, tehtiin ohjelma, jolla tunteita pyrittiin tunnistamaan kasvoilta käyttäen web-kameraa. Tässä vaiheessa malli oli vielä niin yksinkertainen, että se ei pystynyt tunnistamaan tunteita oikein käytännössä ollenkaan. Pystyttiin kuitenkin toteamaan, että molemmat ohjelmat toimivat halutulla tavalla, ja pystyttiin alkaa keskittyä mallin syventämiseen ja kuvien esikäsittelyyn.

Mallin syventämiseen ja kuvien esikäsittelyyn ei ollut mitään oikeita tapoja tai oikeita asetuksia, vaan ne piti löytää ajamalla mallinluomisohjelmaa uudestaan ja uudestaan kokeillen erilaisia asetuksia. Tämä prosessi vei todella runsaasti aikaa. Lopulta mallin tarkkuus alkoi olla halutulla tasolla ja mallin rakenteeseen ja opetusasetuksiin tehdyillä muutoksilla alkoi olla hyvin minimaalisia vaikutuksia, usein myös täysin huomaamattomia. Suodattimien reilu lisääminen ei ollut mahdollista myöskään siksi, että mallin opettamisprosessi olisi kestänyt aivan liian kauan käytössä olevalla laitteistolla. Monilla määrityksillä lopullisessa tarkkuudessa oli huomattavasti enemmän vaihtelevuutta ajaessa ohjelmaa useamman kerran samoilla asetuksilla, joten kun vakailta vaikuttavat määritykset löytyivät, oli hyvä aika lopettaa mallin kehittäminen. Monimutkaisemmasta mallista ei vaikuttanut olevan merkittävää hyötyä, ja erilaisia opetusasetuksia oli käyty kattavasti läpi.

6.3 Mallin luontiohjelman läpikäynti

Mallin luontiohjelmassa ensimmäisenä tuotiin kaikki tarvittavat luokat ja metodit (koodiesimerkki 1).

Koodiesimerkki 1. Luokkien ja metodien tuonti.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.layers import Activation, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Seuraavaksi määriteltiin asetukset näytönohjaimen muistin käytölle. Ilman asetuksien määrittelemistä ohjelma ei toiminut, koska ohjelma yritti käyttää kaikkea näytönohjaimessa olevaa muistia, vaikka osa muistista oli jo muiden ohjelmien käytössä (koodiesimerkki 2).

Koodiesimerkki 2. Näytönohjaimen asetukset.

```
#Asetukset näytönhojaimen muistin käytölle
gpus = tf.config.experimental.list_physical_devices('GPU')
config = tf.config.experimental.set_memory_growth(gpus[0], True)
```

Koodiesimerkissä 3 määriteltiin ensin kuvakansioiden sijainnit ja kuvien määrät. Kuvat jaettiin opetuskuviin, validointikuviin ja testikuviin. Testikuvia käytettiin valmiin mallin tarkkuuden määrittämiseen. Testikuvat antoivat paremman kuvan mallin tehokkuudesta, koska niitä ei ollut käytetty mallin opettamiseen, toisin kuin opetus- ja validointikuvia. Lopuksi määriteltiin minkä kokoisissa erissä kuvat käsiteltiin, ja kuinka monta kertaa kaikki kuvat enimmillään käytiin läpi.

Koodiesimerkki 3. Kuvien määritykset ja asetukset.

```
#Kuvakansiot
dir_training = 'data/Training'
dir_validation = 'data/PublicTest'
dir_testing = 'data/PrivateTest'

#Kuvien määrät
training_images = 28709
validation_images = 3589
testing_images = 3589

#Asetuksia mallin opettamista varten
batch_size = 32
epochs = 50
```

Tämän jälkeen määriteltiin kuville esikäsittelyasetuksia. Kaikkien kuvien kokoa pienennettiin ja opetuskuville määriteltiin myös satunnaista kiertoa ja zoomausta opetustuloksen parantamiseksi. Lopuksi lisättiin mahdollisuus myös kuvan vaakasuoralle käännölle. Kuvien esikäsittely näkyy koodiesimerkissä 4.

Koodiesimerkki 4. Kuvien esikäsittely.

```
#Kuvien esikäsittely
training_datagenerator = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 10,
    zoom_range = 0.2,
    horizontal_flip = True
    )
validation_datagenerator = ImageDataGenerator(rescale = 1./255)

testing_datagenerator = ImageDataGenerator(rescale = 1./255)
```

Koodiesimerkissä 5 kuvat ladataan muuttujiin, määritellään niiden koko, jaetaan ne eriin ja muutetaan ne mustavalkoisiksi.

Koodiesimerkki 5. kuvien lataus ja määritykset.

```
#Kuvien jakaminen eriin
training_generator = training_datagenerator.flow_from_directory(
    dir_training,
    color_mode = 'grayscale',
    target_size = (48,48),
    batch_size = batch_size,
    class_mode = 'categorical'
)

validation_generator = validation_datagenerator.flow_from_directory(
    dir_validation,
    color_mode = 'grayscale',
    target_size = (48,48),
    batch_size = batch_size,
    class_mode = 'categorical'
)

testing_generator = testing_datagenerator.flow_from_directory(
    dir_testing,
    color_mode = 'grayscale',
    target_size = (48,48),
    batch_size = batch_size,
    class_mode = 'categorical'
)
```

Seuraavaksi määriteltiin mallin kerrokset koodiesimerkin 6 mukaisesti. Jokaisen konvoluutiokerroksen jälkeen tehtiin ReLU-aktivointi, jonka jälkeen tehtiin eränormalisointi. Konvoluutiokerroksia tuli malliin lopulta kahdeksan, joista aina kahdessa peräkkäisessä

kerroksessa oli sama määrä suodattimia. Aina kahden konvoluutiokerroksen, niiden aktivoinnin ja normalisoinnin jälkeen tuli alinäytteistyskerros. Alinäytteistyskerroksen jälkeen tuli ”Dropout”-kerros, jossa osa opitusta tiedosta tiputetaan pois ylioppimisen välttämiseksi (Keras, n.d.-c).

Seuraavaksi edellisistä kerroksista luotu piirrekartta valmisteltiin täysin kytkettyä kerrosta varten (Dense-kerros). Tähän käytettiin ”Flatten”-kerrosta, joka muutti piirrekartan yksiulotteiseksi vektoriksi (Jeong, 2019). Tämän jälkeen tuli ensimmäinen täysin kytketty kerros, jonka jälkeen vielä viimeinen ReLU-aktivointikerros ja Dropout-kerros. Lopuksi toinen täysin kytketty kerros, tällä kertaa ulottuvuudella seitsemän, koska tunnistettavia tunteita oli seitsemän. Viimeisenä vielä ”Softmax”-aktivointi, jonka tuloksena saatiin todennäköisyysjakauma eri tunteille (Keras, n.d.-b).

Koodiesimerkki 6. Malliin käytetyt kerrokset.

```
#Mallin luonti
model = Sequential()

model.add(Conv2D(32, kernel_size = (3,3), padding="same", input_shape = (48,48
,1)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Conv2D(64, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Conv2D(128, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
```

```
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Conv2D(256, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(7))
model.add(Activation('softmax'))
```

Ennen mallin opettamisprosessin aloittamista määriteltiin vielä muutamia asetuksia. Opetusprosessi määriteltiin päättymään, jos mallin oppimisessa ei tapahdu kehitystä kuuden kierroksen aikana. Yhden kierroksen aikana kaikki opetuskuvat käydään läpi kerran. Tässä kohdassa määriteltiin myös tiedostonimi ja muut tallennusasetukset. Kolmantena määriteltiin oppimisnopeus pienenemään, jos oppimisessa ei tapahtunut muutosta kahteen kierrokseen. Kaikki määritellyt asetukset näkyvät koodiesimerkissä 7.

Koodiesimerkki 7. Mallin opettamisen asetukset.

```
#Asetukset opettamisen lopettamiselle,
# mallin tallentamiselle ja opetusnopeuden säätämiselle
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor = 'val_loss',
        min_delta = 0,
        patience = 6,
        verbose = 1,
        mode = "min",
        restore_best_weights = True
    ),

    tf.keras.callbacks.ModelCheckpoint(
        filepath = 'emotionRecognitionModel.h5',
        monitor = 'val_loss',
        mode = 'min',
        save_best_only = True,
        verbose = 1
```

```
        ),

    tf.keras.callbacks.ReduceLROnPlateau(
        monitor = 'val_loss',
        factor = 0.5,
        patience = 2,
        verbose = 1,
        mode = "min",
        min_delta = 0.0001
        )
]
```

Koodiesimerkissä 8 näkyy compile-toiminnolla tehdyt konfiguroinnit, kuten oppimisnopeus. Fit-toiminnolla mallin opettaminen lopulta tapahtuu.

Koodiesimerkki 8. Mallin opettaminen.

```
#Valmistellaan malli opetusta varten
model.compile(loss='categorical_crossentropy',
 optimizer = Adam(lr=0.001), metrics = ['accuracy'])

#Mallin opettaminen
model_history = model.fit(
    training_generator,
    epochs = epochs,
    callbacks = my_callbacks,
    steps_per_epoch = training_images//batch_size,
    validation_data = validation_generator,
    validation_steps = validation_images//batch_size
)
```

Mallin opettamisen jälkeen tehtäväksi jäi vain mallin tarkkuuden testaaminen kuvilla mitä ei ollut käytetty opettamiseen. Tämä tehtiin käyttäen evaluate-toimintoa ja testikuvia koodiesimerkin 9 mukaisesti. Koko ohjelmakoodi löytyy liitteestä 1.

Koodiesimerkki 9. Mallin tarkkuuden arviointi.

```
#Mallin tarkkuuden arviointi kuvilla mitä ei ole käytetty mallin luomiseen
score = model.evaluate(testing_generator)
print("Accuracy: {:.2f}".format((score[1]*100)))
```

6.4 Tunteiden tunnistamisohjelman läpikäynti

Tunnistamisohjelman toiminta oli melko yksinkertainen. Videokuvasta napattiin kuva, josta pyrittiin etsimään kasvot käyttäen OpenCV:n valmista ”Haar Cascade”-luokittelijaa. Jos kuvasta löytyi kasvot, ne eroteltiin omaksi kuvakseen, joka muutettiin samaan kokoon, mitä mallin opettamiseen käytetyt kuvat olivat. Tämän jälkeen kuva syötettiin tunteiden tunnistamismallille, joka palautti arviot siitä, millaisella todennäköisyydellä kuvan kasvot osoittivat mitäkin tunnetta. Näistä arvioista voitiin tämän jälkeen valita se tunne, jonka todennäköisyys oli suurin. Tunteiden tunnistamiseen käytetty silmukka on nähtävissä koodiesimerkissä 10. Koko ohjelmakoodi löytyy liitteestä 2.

Koodiesimerkki 10. Tunteiden tunnistamisen silmukka.

```
while True:
    #Luetaan kuva kameran syötteestä
    _, frame = capture.read()
    #Muutetaan kuva mustavalkoiseksi
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    #Erotellaan kuvasta kasvot
    faces = faceCascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame,(x, y),(x+w,y+h),(255, 0, 0), 3)
        face_gray = gray[y:y+h,x:x+w]
        #Muutetaan kuva samaan kokoon kuin mallin luomiseen käytetyt kuvat
        face = cv2.resize(face_gray, (48, 48))
        face = face/255
        face = img_to_array(face)
        face = numpy.expand_dims(face, axis = 0)
        #Ennustetaan kasvoilla näkyvä tunne palauttamalla eri tunteiden painoa
rvot
        predictions = model.predict(face)
        #Valitaan tunne, jonka painoarvo on suurin
        label = emotions[predictions.argmax()]
        #Lisätään kuvaan teksti, jossa lukee ennustettu tunne
        cv2.putText(frame,label,(x,y),cv2.FONT_HERSHEY_DUPLEX, 1, (255, 255, 2
55), 2)
    cv2.imshow('Tunteiden tunnistaja', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

6.5 Tulokset

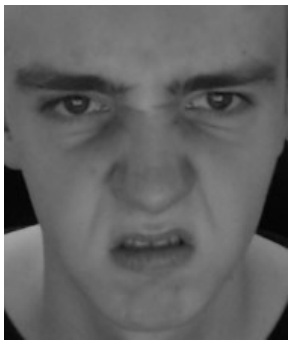
Lopullisen mallin tarkkuudeksi tuli testikuvilla 66.79 %. Tarkkuus vaihtelee uutta mallia opettaessa aina hieman, koska opetus alkaa aina satunnaisarvoista. Jos tämä malli olisi ollut mukana 2013 järjestetyssä ”Challenges in Representation Learning: Facial Expression Recognition Challenge”-kilpailussa, jota varten FER-2013–kuvatietokanta julkaistiin, olisi malli tullut sijalle 5.

Kuvassa 7 näkyy esimerkkikuvat kaikista tunnistettavista tunteista. Mallilla pystytään siis tunnistamaan kaikkia tunteita. Ilman ongelmia kaikkia ennusteita ei kuitenkaan pystytty tuottamaan. Suurin ongelma tuli vastaan kasvojen tunnistuksessa. Kuvista haluttiin saada mahdollisimman selkeitä ja tasaisesti valaistuja. Jo ikkunoista tuleva luonnonvalo teki taustasta liian valoisan ja esti kasvojentunnistuksen normaalin toiminnan, mikä tuotti haasteita kasvojen valaisemiseen ilman, että taustasta tulee liian kirkas.

Ainut ennuste minkä tuottamisessa oli haasteita, oli surullisuuden tunne, minkä ohjelma tunnisti suurimman osan ajasta neutraalina. Kaikki muut ennusteet toimivat jopa erittäin hyvin. Kasvonilmeiden on kuitenkin oltava huomattavia. Pienistä kasvonilmeistä ohjelma ei kykene tekemään tarkkoja johtopäätöksiä. Ohjelma tunnisti erot pelokkaan ja yllättyneen ilmeen välillä tehokkaasti, eikä sekoittanut niitä toisiinsa. Inhon ilmeen ohjelma tunnisti välillä vihaisena, mutta suurimman osan ajasta ennusteet olivat oikein. Kuvassa 8 on esimerkki inhonilmeestä, jonka ohjelma tunnisti vihaiseksi.

Kuva 7. Esimerkkikuvat tunnistettavista tunteista.

Kuva 8. Inhonilme, jonka ohjelma tunnisti vihaiseksi.



7 Yhteenveto

Tunteita tunnistavia järjestelmiä on jo melko yksinkertaista toteuttaa, koska niiden luomiseen käytettäviä ohjelmistoja on vapaasti saatavilla ja niiden käyttöönottoon on olemassa selkeät ohjeet ja mahdollisiin ongelmiin on helppo löytää apua. Tunteiden tunnistaminen on myös melko aktiivinen tekoälyn tutkimusalue, joten aiheesta on runsaasti tietoa saatavilla.

Järjestelmien on mahdollista tunnistaa tunteita jo melko tehokkaasti. Tämä kuitenkin vaatii ainakin toistaiseksi optimaalisia kuvausolosuhteita, eli oikeanlaisen valaistuksen, sekä kameran kohtisuoran suuntauksen kohteeseen. Lisäksi kohteen osoittamien kasvonilmeiden on oltava hyvin selkeitä. Ohjelma ei pysty erottamaan tunteita tehokkaasti, jos ilmeet eivät eroa neutraalista huomattavasti. Harva ihminen kuitenkaan osoittaa kaikkia tuntemiaan tunteita aina hyvin voimakkaasti kasvonilmeillään, joten tällaisen järjestelmän sovittaminen johonkin oikean elämän käyttötarkoitukseen on haastavaa.

Projektin tulokset ovat kuitenkin erittäin positiivisia. Tunteiden tunnistamiseen pystyttiin rakentamaan toimiva järjestelmä ja mallin tarkkuudessa onnistuttiin pääsemään hyvälle tasolle normaalilla kuluttajakäyttöön tarkoitetulla laitteistolla, vaikka kyse on monimutkaisen konvoluutioneuroverkon luomisesta, jonka opettamiseen käytettiin kymmeniä tuhansia kuvia. On myös oletettavaa, että suurten yritysten ja alan osaavimpien ammattilaisten on mahdollista jo nykyisellä teknologialla toteuttaa huomattavasti tehokkaampia järjestelmiä. Jos järjestelmän kehitystä haluaisi jatkaa, sitä voisi laajentaa tunnistamaan esimerkiksi käsien ja jalkojen asentoa tai liikkeitä, tai ottamaan huomioon kasvojen värin muutokset. Työn tekeminen opetti paljon tekoälyn ja tietokonenäön toiminnasta. Tunteiden tunnistaminen oli myös aihealueena erittäin mielenkiintoinen.

Lähteet

Apell, P. (12.1.2020). *Kone + oppiminen = koneoppiminen – Mitä se oikein tarkoittaa? [kuva]*. Haettu 16.3.2021 osoitteesta Piilo-osaajat: https://piilo-osaajat.com/2017/08/30/kone-oppiminen-koneoppiminen-mita-se-oikein-tarkoittaa/

Barret, L. F., Adolphs, R., Marsella, S., Martinez, A. M., & Pollak, S. D. (2019). *Emotional Expressions Reconsidered: Challenges to Inferring Emotion From Human Facial Movements.* Association for psychological science. doi:https://doi.org/10.1177/1529100619832930

Beatrice, A. (27.11.2020). *Computer Vision: The Doctor's Eye of Healthcare Industry*. Haettu 15.2.2021 osoitteesta Analytics Insight: https://www.analyticsinsight.net/computer-vision-the-doctors-eye-of-healthcare-industry/

Boucher, P. (2020). *Artificial intelligence: How does it work, why does it matter, and what can we do about it?* European Parliament. Brussels: Scientific Foresight Unit. Noudettu osoitteesta https://www.europarl.europa.eu/RegData/etudes/STUD/2020/641547/EPRS_STU(2020)641547_EN.pdf

Brooke, S. (13.6.2020). *The impact of computer vision on the automotive industry*. Haettu 15.2.2021 osoitteesta Innovation Enterprise: https://channels.theinnovationenterprise.com/articles/the-impact-of-computer-vision-on-the-automotive-industry

Brownlee, J. (5.7.2019). *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. Haettu 3.2.2021 osoitteesta Machine Learning Mastery: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

Camacho, C. (3.6.2018). *Convolutional neural networks [kuva]*. Haettu 9.3.2021 osoitteesta https://cezannec.github.io/Convolutional_Neural_Networks/

Canedo, D.;& Neves, A. J. (2019). *Facial Expression Recognition Using Computer Vision: A Systematic Review.* doi:https://doi.org/10.3390/app9214678

Computer Science Wiki. (n.d.). *Max Pooling [kuva]*. Haettu 8.3.2021 osoitteesta https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

Council of Europe. (n.d.). *History of Artificial Intelligence*. Haettu 27.1.2021 osoitteesta https://www.coe.int/en/web/artificial-intelligence/history-of-ai

Dawson-Howe, K. (2014). *A Practical Introduction to Computer Vision with OpenCV.* John Wiley & Sons, Incorporated.

DeepAI. (n.d.). *ReLU*. Haettu 9.3.2021 osoitteesta DeepAI: https://deepai.org/machine-learning-glossary-and-terms/relu

Faggella, D. (14.3.2020). *Computer Vision Applications – Shopping, Driving and More*. Haettu 9.2.2021 osoitteesta Emerj: https://emerj.com/ai-sector-overviews/computer-vision-applications-shopping-driving-and-more/

IBM Cloud Education. (15.7.2020a). *Machine Learning*. Haettu 2.3.2021 osoitteesta IBM: https://www.ibm.com/cloud/learn/machine-learning

IBM Cloud Education. (20.10.2020b). *Convolutional Neural Networks*. Haettu 3.2.2021 osoitteesta IBM: https://www.ibm.com/cloud/learn/convolutional-neural-networks

Jeong, J. (24.1.2019). *The Most Intuitive and Easiest Guide for Convolutional Neural Network*. Haettu 29.3.2021 osoitteesta towards data science: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480

Kelleher, J. D. (2019). *Deep Learning.* London, England: The Massachussetts Institute of Technology.

Keras. (n.d.-a). *About*. Haettu 3.3.2021 osoitteesta Keras: https://keras.io/about/

Keras. (n.d.-b). *Layer activation functions*. Haettu 29.3.2021 osoitteesta Keras: https://keras.io/api/layers/activations/

Keras. (n.d.-c). *Dropout layer*. Haettu 29.3.2021 osoitteesta Keras: https://keras.io/api/layers/regularization_layers/dropout/

Lewis, T. (4.12.2014). A Brief History of Artificial Intelligence. *Live Science*. Haettu 27.1.2021 osoitteesta https://www.livescience.com/49007-history-of-artificial-intelligence.html

Li, K., Jin, Y., Waqar Akram, M., Han, R. & Jiongwei, C. (2020). *Facial expression recognition with convolutional neural networks via a new face cropping and rotation strategy.* Springer. Haettu 15.2.2021 osoitteesta https://www.researchgate.net/publication/330301581_Facial_expression_recognition_with_convolutional_neural_networks_via_a_new_face_cropping_and_rotation_strategy

Motion Metrics. (16.5.2019). *How Artificial Intelligence Revolutionized Computer Vision: A Brief History*. Haettu 9.2.2021 osoitteesta Motion Metrics:

https://www.motionmetrics.com/how-artificial-intelligence-revolutionized-computer-vision-a-brief-history/

OpenCV. (n.d.). *About*. Haettu 3.3.2021 osoitteesta OpenCV: https://opencv.org/about/

Pietikäinen, M. & Silvén, O. (2019). *Tekoälyn haasteet – koneoppimisesta ja konenäöstä tunnetekoälyyn.* Suomi: Konenäön ja signaalianalyysin keskus. Noudettu osoitteesta http://jultika.oulu.fi/files/isbn9789526224824.pdf

Pilarczyk, M. & Wieczorek, T. (9.2014). How to measure emotions? Latest research, future possibilities. doi:https://doi.org/10.13140/2.1.1193.8564

Planche, B. & Andres, E. (2019). *Hands-On Computer Vision with TensorFlow 2.* Packt Publishing.

Rockwell, A. (28.8.2017). *The History of Artificial Intelligence*. Haettu 27.1.2021 osoitteesta Science in the news, Harvard University: http://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

Roth, M. (5.8.2019). *Computer Vision in Healthcare - Current Applications*. Haettu 15.2.2021 osoitteesta Emerj: https://emerj.com/ai-sector-overviews/computer-vision-healthcare-current-applications/

Saik, M. (16.7.2020). *Use cases of computer vision in retail*. Haettu 9.2.2021 osoitteesta Future Mind: https://www.futuremind.com/blog/use-cases-computer-vision-retail

SAS. (n.d.-a). *Computer Vision*. Haettu 9.2.2021 osoitteesta SAS: https://www.sas.com/en_us/insights/analytics/computer-vision.html

SAS. (n.d.-b). *Machine Learning*. Haettu 2.3.2021 osoitteesta SAS: https://www.sas.com/en_us/insights/analytics/machine-learning.html

Suomen koodikoulu. (7.12.2018). *Johdatus tekoälyyn.*

TensorFlow. (n.d.). Haettu 3.3.2021 osoitteesta TensorFlow: https://www.tensorflow.org/

Toprak, M. (14.6.2020). *Activation Functions for Deep Learning [kuva]*. Haettu 25.3.2021 osoitteesta Medium: https://medium.com/@toprak.mhmt/activation-functions-for-deep-learning-13d8b9b20e

Tuominen, H. & Neittaanmäki, P. (2019). Keinotekoiset neuroverkot. Teoksessa *Tekoälyn perusteita ja sovelluksia.* Noudettu osoitteesta https://tim.jyu.fi/view/kurssit/tie/tiep1000/tekoalyn-sovellukset/kirja#keinotekoiset-neuroverkot

USM. (5.10.2020). *Top 7 Use Cases of Computer Vision in Manufacturing*. Haettu 9.2.2021 osoitteesta usmsystems: https://usmsystems.com/use-cases-of-computer-vision-in-manufacturing/

Liite 1: Mallin luomisen ja opettamisen ohjelmakoodi

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.layers import Activation, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#Asetukset näytönhojaimen muistin käytölle
gpus = tf.config.experimental.list_physical_devices('GPU')
config = tf.config.experimental.set_memory_growth(gpus[0], True)

#Kuvakansiot
dir_training = 'data/Training'
dir_validation = 'data/PublicTest'
dir_testing = 'data/PrivateTest'

#Kuvien määrät
training_images = 28709
validation_images = 3589
testing_images = 3589

#Asetuksia mallin opettamista varten
batch_size = 32
epochs = 50

#Kuvien esikäsittely
training_datagenerator = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 10,
    zoom_range = 0.2,
    horizontal_flip = True
    )
validation_datagenerator = ImageDataGenerator(rescale = 1./255)

testing_datagenerator = ImageDataGenerator(rescale = 1./255)

#Kuvien jakaminen eriin
training_generator = training_datagenerator.flow_from_directory(
    dir_training,
    color_mode = 'grayscale',
    target_size = (48,48),
    batch_size = batch_size,
    class_mode = 'categorical'
)

validation_generator = validation_datagenerator.flow_from_directory(
```

```
    dir_validation,
    color_mode = 'grayscale',
    target_size = (48,48),
    batch_size = batch_size,
    class_mode = 'categorical'
)

testing_generator = testing_datagenerator.flow_from_directory(
    dir_testing,
    color_mode = 'grayscale',
    target_size = (48,48),
    batch_size = batch_size,
    class_mode = 'categorical'
)

#Mallin luonti
model = Sequential()

model.add(Conv2D(32, kernel_size = (3,3), padding="same", input_shape = (48,48,1)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Conv2D(64, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Conv2D(128, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Conv2D(256, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, kernel_size = (3,3), padding="same"))
model.add(Activation('relu'))
```

```
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding="same"))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(7))
model.add(Activation('softmax'))

#Asetukset opettamisen lopettamiselle,
# mallin tallentamiselle ja opetusnopeuden säätämiselle
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor = 'val_loss',
        min_delta = 0,
        patience = 6,
        verbose = 1,
        mode = "min",
        restore_best_weights = True
    ),

    tf.keras.callbacks.ModelCheckpoint(
        filepath = 'emotionRecognitionModel.h5',
        monitor = 'val_loss',
        mode = 'min',
        save_best_only = True,
        verbose = 1
        ),

    tf.keras.callbacks.ReduceLROnPlateau(
        monitor = 'val_loss',
        factor = 0.5,
        patience = 2,
        verbose = 1,
        mode = "min",
        min_delta = 0.0001
        )
]

#Valmistellaan malli opetusta varten
model.compile(loss='categorical_crossentropy',
 optimizer = Adam(lr=0.001), metrics = ['accuracy'])

#Mallin opettaminen
model_history = model.fit(
    training_generator,
    epochs = epochs,
    callbacks = my_callbacks,
```

```
    steps_per_epoch = training_images//batch_size,
    validation_data = validation_generator,
    validation_steps = validation_images//batch_size
)


#Mallin tarkkuuden arviointi kuvilla mitä ei ole käytetty mallin luomiseen
score = model.evaluate(testing_generator)
print("Accuracy: {:.2f}".format((score[1]*100)))
```

Liite 2: Tunteiden tunnistamisen ohjelmakoodi

```
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
import numpy
import cv2

#Asetukset näytönhojaimen muistin käytölle
gpus = tf.config.experimental.list_physical_devices('GPU')
config = tf.config.experimental.set_memory_growth(gpus[0], True)

#Ladataan kasvojentunnistus-malli
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

#Ladataan tunteidentunnistus-malli
model = load_model('emotionRecognitionModel.h5')

#Tunneluokat
emotions = ['Vihainen','Inho','Pelokas','Iloinen','Neutraali','Surullinen','Yllatty
nyt']

capture = cv2.VideoCapture(0)


while True:
    #Luetaan kuva kameran syötteestä
    _, frame = capture.read()
    #Muutetaan kuva mustavalkoiseksi
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    #Erotellaan kuvasta kasvot
    faces = faceCascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame,(x, y),(x+w,y+h),(255, 0, 0), 3)
        face_gray = gray[y:y+h,x:x+w]
        #Muutetaan kuva samaan kokoon kuin mallin luomiseen käytetyt kuvat
        face = cv2.resize(face_gray, (48, 48))
        face = face/255
        face = img_to_array(face)
        face = numpy.expand_dims(face, axis = 0)
        #Ennustetaan kasvoilla näkyvä tunne palauttamalla eri tunteiden painoarvot
        predictions = model.predict(face)
        #Valitaan tunne, jonka painoarvo on suurin
        label = emotions[predictions.argmax()]
        #Lisätään kuvaan teksti, jossa lukee ennustettu tunne
        cv2.putText(frame,label,(x,y),cv2.FONT_HERSHEY_DUPLEX, 1, (255, 255, 255),
2)
```

```
    cv2.imshow('Tunteiden tunnistaja', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
capture.release()
cv2.destroyAllWindows()
```