KASVONTUNNISTUS NEUROVERKOLLA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Sähkö- ja automaatiotekniikka, insinööri (AMK)

Syksy 2022

Juho Holopainen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Sähkö- ja automaatiotekniikka
Tekijä Juho Holopainen
Työn nimi Kasvontunnistus neuroverkolla
Ohjaaja Juhani Henttonen

Tiivistelmä
Vuosi 2022

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa konvoluutio-neuroverkkoon pohjautuva kasvontunnistussovellus. Sovelluksessa käytettävä neuroverkko tuli kouluttaa erottamaan henkilöllisyyksiä toisistaan kasvokuvien avulla.

Työssä valittiin valmis neuroverkkoarkkitehtuuri ja sen koulutuksessa hyödynnettiin siirto-oppimista. Tunnistustehtävän ratkaisemiseksi esiteltiin kirjoittajan suunnittelema häviöfunktio, jota käyttämällä saavutettiin demosovellukseen riittävä tunnistustarkkuus. Sovelluksen koodi kirjoitettiin Python-ohjelmointikielellä.

Neuroverkon oppimisprosessi oli stabiili ja saavutettu tarkkuus kohtalaisen hyvä. Koulutuksessa ei havaittu tapahtuneen ylisovittumista. Lopputuloksena saatiin aikaan yksinkertainen, mutta toimiva sovellus, jota käyttämällä saatiin aikaan verrattain onnistuneita tunnistuksia ottaen huomioon lyhyen koulutusajan, mallin pienen koon ja aineiston suppeuden.

Avainsanat neuroverkko, kasvontunnistus, syväoppiminen
Sivut 45 sivua ja liitteitä 1 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Electrical & Automation Engineering
Author Juho Holopainen
Subject Face recognition using a neural network
Supervisors Juhani Henttonen

Abstract
Year 2022

The objective of this thesis was to design and implement a face recognition application based on a convolutional neural network. The neural network used by the application was required to distinguish persons apart based on the images of their faces.

A readily available neural network architecture was chosen, and transfer learning was used in the training process. To achieve the learning goal, a new loss function was introduced by the author. The use of the new loss function enabled a sufficient accuracy of the demo application. The code of the application was written in Python.

The learning process of the model was stable, and a moderate accuracy was reached. Signs of overfitting during training were not detected. The resulting application was simple, but it provided relatively accurate predictions considering the short training time, small size of the model and the constricted source data.

Keywords neural network, face recognition, deep learning
Pages 45 pages and appendices 1 pages

Sisällys

Liitteet

Liite 1 Linkki kirjoitettuun koodiin

Kuvat, kaavat ja taulukot

1 Johdanto

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa konvoluutio-neuroverkkoon pohjautuva kasvontunnistussovellus. Sovelluksen tuli löytää videokuvasta poimituille kasvokuville vastaavuus tietokannasta.

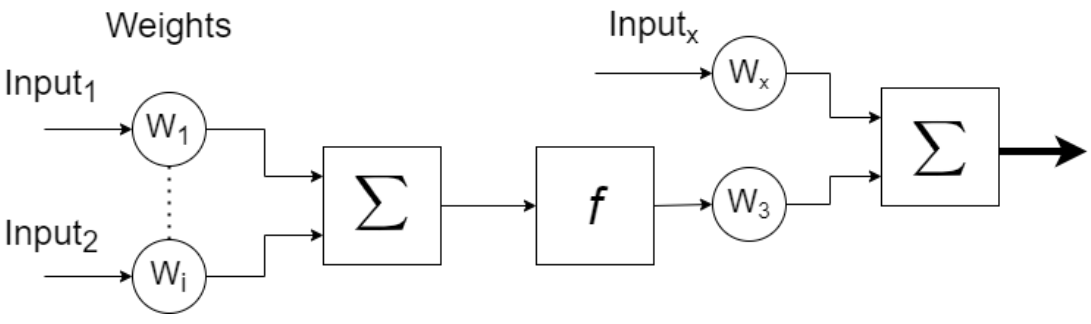
Hakuprosessiin käytettävä neuroverkko tuli kouluttaa henkilöiden erottamiseen toisistaan. Tavoitteena oli valita sopiva neuroverkkoarkkitehtuuri, tehtävään soveltuva aineisto ja koulutusstrategia. Neuroverkon laatu ja käytettävyysaste piti varmistaa analysoimalla sen toimintaa.

2 Neuroverkot

Neuroverkot koostuvat toisiinsa kytketyistä keinotekoisista neuroneista, joiden toiminnalla on yhtäläisyyksiä aivojen hermosolujen kanssa. Neuronit ottavat vastaan toisilta neuroneilta saapuvaa signaalia ja syöttävät sitä eteenpäin joko vahvana tai heikkona ulostulona. (Kelleher, 2020/2019, s. 65)

Keinoneuronin toiminta voidaan jakaa kahteen vaiheeseen. Ensiksi lasketaan neuronin syötteiden painotettu summa. Sen jälkeen tulos siirretään aktivointifunktiolle, joka muodostaa neuronin varsinaisen ulostuloarvon. (Kelleher, 2020/2019, s. 69) Kuvassa 1 esitellään kahden peräkkäisen keinoneuronin kytkeytyminen toisiinsa.

Kuva 1. Keinoneuronin toiminta

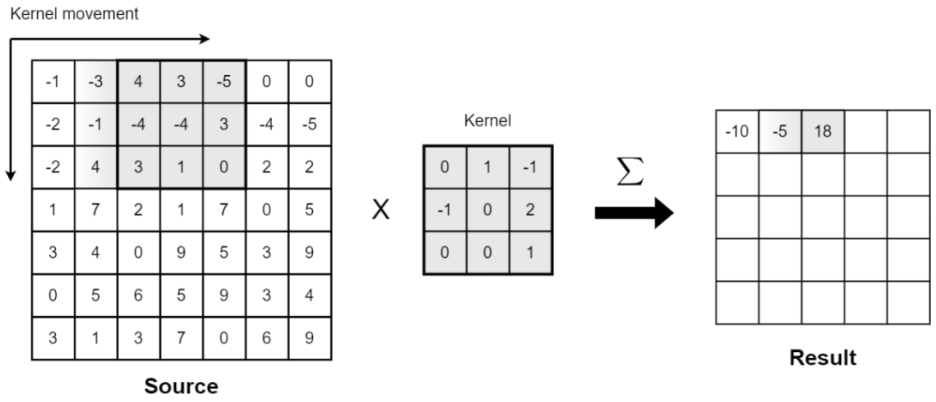


2.1 Konvoluutioneuroverkot

Konvoluutioneuroverkot luotiin ratkaisemaan kuvantunnistusongelmia. Konvoluutioneuroverkko kykenee oppimaan eristämään syötekuvasta paikallisia piirteitä varhaisissa kerroksissa ja yhdistelemään niitä myöhemmissä kerroksissa korkean tason piirteiksi. Neuroverkon viimeiset kerrokset yhdistelevät syntyneet korkean tason piirteet lopulliseksi tunnistukseksi. (Kelleher, 2020/2019, s. 145)

Konvoluutioneuroverkossa konvoluutio muodostetaan liu'uttamalla suodinmaskia syötteen yli tietyllä askelpituudella. Konvoluution tulos saadaan laskemalla suotimen ja sen sijaintia lähteessä vastaavan pikselinaapuruston painotettu summa. Tulos sijoitetaan tulostematriisiin. Operaation painoina toimivat suotimen painot ja naapuruston koon määrää suodinmaskin koko ja muoto. (Rossi, 2008) Kuvassa 2 visualisoidaan konvoluution soveltaminen 2-uloitteiseen lähteeseen.

Kuva 2. Konvoluutio askelpituudella yksi ja suodinkoolla 3x3



Konvoluutioneuroverkot koostuvat pääosin konvoluutiokerroksista, koontikerroksista ja täysin kytketyistä kerroksista. Konvoluutiokerros huolehtii neuronien aktivoinnista päivittämällä suotimien painoarvoja. Koontikerros pienentää syötteen tilavuuksia sitä seuraaville kerroksille. Täysin kytketty kerros pyrkii syötteen luokitukseen sen saamien aktivointien perusteella. (O'Shea & Nash, 2015)

Eri syvyyksillä olevien konvoluutiokerrosten tulosteita voidaan visualisoida kanava kerrallaan. Kuvassa 3 havainnollistetaan kerrosten synnyttämiä aktivaatioita. Pikselien kirkkauden taso kuvastaa signaalin vahvuutta kyseisessä neuronissa. Mitä alemmas mennään kuvan rivejä, sitä syvemmällä konvoluutiokerros on neuroverkossa.

Kuva 3. Konvoluutiokerrosten satunnaisia tulosteita

2.2 Residuaalineuroverkko

Residuaali-arkkitehtuuri on Microsoftin kehittäjien esittämä ratkaisu syvien neuroverkkojen koulutukseen. Residuaalilohkoja hyödyntävää neuroverkkoa kutsutaan residuaalineuroverkoksi.

Residuaalilohko koostuu rinnankytketystä konvoluutiolohkosta ja sen kiertävästä identiteettipolusta, jossa ei ole kerroksia tai painoja. Identiteettipolku ja konvoluutiolohkon tulosteet summataan yhteen niitä seuraavassa summakerroksessa. Painoton Identiteettipolku mahdollistaa vahvan gradientin välittymisen myös residuaalilohkoa edeltäville kerroksille (He, Zhang, Ren & Sun, 2016)

Identiteettipolun ja konvoluutiolohkon summaamisen edellytyksenä on, että konvoluutiolohkon ja sitä edeltävän kerroksen ulottuvuudet ovat identtiset. Identtisyys saavutetaan käyttämällä kolmea konvoluutiokerrosta, joissa suodinmaskin koot ovat 1x1, 3x3 ja 1x1. Ensimmäinen suodin laventaa tensorin, seuraava toimii pullonkaulana, jossa tensorista muodostetaan vähemmän informaatiota sisältävä representaatio ja viimeinen suodin muuntaa tensorin takaisin alkuperäiseen kokoonsa. (He ym. 2016)

Residuaalilohkon vahvuus on siinä, että konvoluutioverkon lohkojen ei tarvitse oppia mallintamaan yhdessä kokonaisfunktiota H(x) vaan jokainen kerros voi oppia itsenäisesti kokonaisfunktion osan: jäännösfunktion F(x) + x. (He ym. 2016)

2.3 Käänteinen residuaalirakenne

Käänteinen residuaalirakenne on Googlen kehitystiimin esittämä ratkaisu syvien neuroverkkojen soveltamiseen laskentateholtaan rajoitetuissa laitteissa. Kehitystiimin julkaisemassa tutkimuksessa esitellään konvoluutioneuroverkko, jota kutsutaan MobileNetV2:ksi. Verkon rakenne perustuu syvyyssuuntaisiin konvoluutioihin residuaalirakenteissa.

Käänteinen residuaalirakenne perustuu residuaalilohkojen käyttöön, joissa identiteettipolut sijaitsevat ohuiden pullonkaulakerrosten välissä. Lohkossa syöte laajennetaan ensin suurempaan ulottuvuuteen, jonka jälkeen käytetään syvyyssuuntaista konvoluutiota. Syvyyssuuntainen konvoluutio tuo rakenteeseen epälineaarisuutta, mikä parantaa verkon suorituskykyä. Tulos palautetaan lopuksi alkuperäiseen muotoon käyttämällä lineaarista konvoluutiota. Tutkimuksessa rakennetta kutsutaan residuaaliseksi pullonkaulakerrokseksi. (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018).

2.4 Siirto-oppiminen

Neuroverkon kouluttamista ei tarvitse aina aloittaa jostain satunnaisesti alustetusta pisteestä. Käsiteltävää ongelmaa voidaan lähteä ratkaisemaan siirto-oppimisen avulla. Työssä hyödynnettiin siirto-oppimista käyttämällä ImageNet-aineistoon koulutetun mallin painoja.

ImageNet on WordNet:in perustuva tietoaineisto, jonka pyrkimyksenä on tarjota koneoppimisen tutkijoille tietoaineisto, jossa jokaista WordNetin konseptia (synset) kohti tarjotaan 500–1000 kuvaa. WordNetissä on yhteensä yli 100 000 konseptia. ImageNet kehitettiin luokittelutehtäviin kehitetyille malleille vertailujohtamiseen. (ImageNet, n.d.)

Siirto-oppimisessa käytetään käsiteltävän tehtävän ratkaisuun jotain muuta aiemmin koulutettua neuroverkkoa, jolla on ollut samankaltainen oppimistavoite. Siirto-oppimista on käytetty oppimisen nopeuttamiseen. Oppimisen nopeutuminen perustuu siihen, että koulutetun mallin varhaiset kerrokset ovat jo oppineet poimimaan syötteestä matalan tason piirteitä kuten reunoja. Matalan tason piirteet ovat yleensä yleishyödyllisiä muidenkin kuin koulutukseen käytettyjen luokkien tunnistamiseen. (Kelleher, 2020/2019, s. 208)

2.5 Syötön augmentointi

Koneoppivan algoritmin väärä valinta voi aiheuttaa alisovittamista tai ylisovittamista. Alisovittaminen tarkoittaa sitä tilannetta, jossa algoritmin induktiivinen vinouma on liian vahva, eikä algoritmi tavoita todellista aineistoa. Ylisovittaminen taas koskee tilannetta, jossa vinouma on liian salliva. Tällöin algoritmi poimii aineistosta kohinaa, eikä funktio yleisty uuteen aineistoon. (Kelleher, 2020/2019, s. 27)

Neuroverkon kouluttamiseen tarvitaan riittävästi tasapainotettua lähdedataa, jotta ylisovittuminen voitaisiin välttää. Aineiston kerääminen voi olla haasteellista ja aineisto saattaa jäädä suppeaksi. Olemassa olevaa aineistoa voidaan kuitenkin laajentaa muokkaamalla sitä. Aineistomuokkauksen on havaittu parantavan huomattavasti neuroverkkojen tarkkuutta. (Shorten & Khoshgoftaar, 2019)

Jos lähdeaineistona käytetään kuvia, voidaan niitä esimerkiksi venyttää, kiertää tai rajata. Kuviin voidaan myös tehdä ei-geometrisiä muutoksia: kohinan lisääminen tai poistaminen, kuvien yhdistely tai värikanavien sekoittaminen. Myös kuvien kirkkaustasoa voidaan säätää. Muokkaamisen lisäksi voidaan aineiston käsittelyä muuttaa. Aineistoa voidaan sekoittaa tai siitä voidaan valikoida lohkoja käsiteltäväksi. (Shorten & Khoshgoftaar, 2019)

2.6 Käytetyt kirjastot

Opinnäytetyössä käytettiin useita python-kirjastoja, joista tärkeimmät olivat Tensorflow, Albumentations ja Mediapipe. Tensorflowta käytettiin mallin koulutukseen, Albumentations'a kuvien muokkaukseen ja MediaPipen Face Mesh-sovellusta kasvojen eristämiseen ja muokkaukseen kamerakuvasta demosovelluksessa.

Tensorflow on alun perin Google Brain tiimin kehittämä avoimen lähdekoodin ohjelma, joka on erikoistunut korkean tason numeraaliseen laskentaan. Kirjasto tarjoaa koneoppimiseen ja syväoppimiseen tarvittavat työkalut. (Google Inc., 2022)

Albumentations on kuvien manipulointiin tarkoitettu kirjasto, jota käytetään muun muassa syväoppimisessa koulutettavan datan generointiin. Kirjasto tarjoaa laajan valikoiman työkaluja, joita voidaan käyttää suoraan Tensorflown kanssa. (Albumentations, 2022)

MediaPipen Face Mesh on neuroverkkoon pohjautuva kasvojen landmark-pisteitä arvioiva sovellus. Sovellus arvioi kuvan kasvoista 468 kolmiulotteista koordinaattia, joita voidaan hyödyntää MediaPipen tarjoamien muiden sovellusten kanssa. Sovellusta voidaan hyödyntää esimerkiksi augmentoidun todellisuuden teknologioissa. (MediaPipe, 2022)

2.7 Google Colaboratory

Google Colaboratory on Google Research:n tarjoama tuote, jossa voidaan suorittaa python-koodia web-selainta käyttämällä. Colaboratory soveltuu muun muassa koneoppimiseen ja data-analyysiin. Tuote on houstattu Jupyter notebook serverillä, jonka python-ympäristöön on jo valmiiksi asennettu koneoppimiseen tarvittavia yleisimpiä kirjastoja. Colaboratory tarjoaa ilmaista laskentatehoa virtuaalikoneella, mutta sen tarjoamien resurssien vakautta ei taata. Tuotteen käytöllä on sekä päivittäinen että yhtäjaksoisen käytön aikaraja. Yhtäjaksoisen käytön raja on maksimissaan 12 tuntia. (Google, n.d.)

3 Neuroverkon tehtävän määritys

Neuroverkon tehtäväksi asetettiin henkilöllisyyksien erottaminen toisistaan kasvokuvien perusteella. Tehtävän toteuttaminen vaati tutustumista aiemmin tehtyihin tutkimuksiin sekä ongelman ratkaisuun liittyvään kirjallisuuteen.

3.1 Lähimmän naapurin hakuongelma

Lähimmän naapurin hakuongelmassa tarkoituksena on löytää pisteelle q k-määrä tietokannan pisteitä, jotka ovat lähimpänä pistettä q. Pisteet ovat d-ulotteisia, joiden läheisyys lasketaan jollakin samankaltaisuutta arvioivalla funktiolla. Tavoitteena on prosessoida tietokannan S pisteet siten, että niiden joukosta voidaan mahdollisimman nopeasti löytää pistettä q lähimpänä oleva esimerkki. (Kibriya & Frank, 2007)

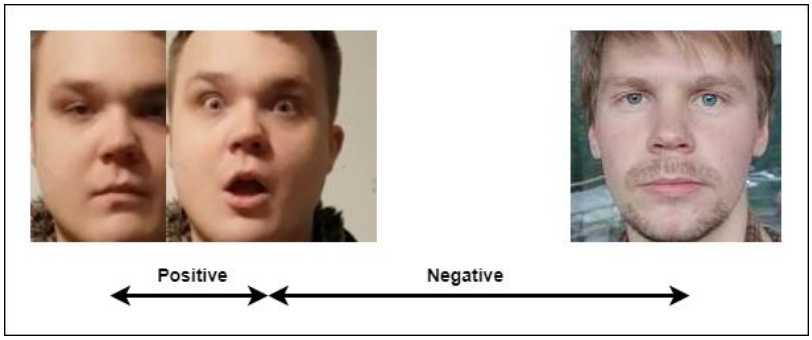
Ongelman ratkaisemiseksi on kehitetty eri algoritmeja, joista yksinkertaisin on lineaarinen menetelmä. Lineaarisessa menetelmässä tietokantaa ei prosessoida, kaikki pisteiden väliset etäisyydet lasketaan ja tulosten joukosta valitaan lähimmät k-naapuria. Menetelmän aikavaativuus joukolle n pisteitä, joiden tilavuus on d, on O(d*n). (Kibriya & Frank, 2007)

Tässä opinnäytetyössä käytetään lineaarista hakumenetelmää esimerkkisovelluksen yksinkertaistamiseksi. Sovelluksen ei katsota käyttävän niin laajaa aineistoa demonstraatiossa, että monimutkaisempien algoritmien käyttö olisi tarpeellista.

3.2 Kolmikoiden louhinta

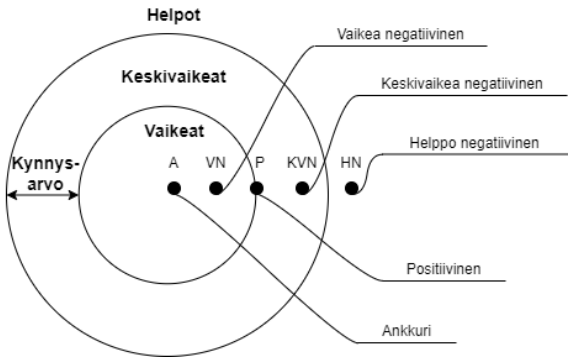
Kolmikoiden louhinta on neuroverkon koulutuksessa käytetty tekniikka, jonka tavoitteena on muodostaa kolmikoita kahdesta samaan luokkaan kuuluvasta positiivisesta- ja yhdestä eri luokkaan kuuluvasta negatiivisesta parista (kuva 4). Tekniikan taustalla on pyrkimys saada samaan luokkaan kuuluvat mallin tulosteet mahdollisimman lähelle toisiaan ja eri luokkiin kuuluvat negatiiviset tulosteet kauaksi toisistaan. Tavoitteena ei kuitenkaan ole luoda tiukkoja klustereita vaan saada negatiiviset etäisyydet kauemmaksi kuin positiiviset. (Schroff, Kalenichenko & Philbin, 2015)

Kuva 4. Positiivinen ja negatiivinen pari



FaceNet-mallin suunnittelijat luokittelivat negatiiviset parit kolmeen eri kategoriaan: helpot-, keskivaikeat- ja vaikeat negatiiviset (kuva 5). Helppojen negatiivisten parien etäisyys ankkurista on suurempi kuin positiivisen parin etäisyyden ja kynnysarvon summa. Keskivaikeat negatiiviset parit sijaitsevat positiivisen parin ja kynnysarvon välissä. Vaikeiden negatiivisten parien etäisyys ankkurista on pienempi kuin positiivisen parin. (Schroff ym. 2015)

Kuva 5. Negatiivisten parien luokittelu



FaceNet-tutkimuksessa mainittiin kaksi erilaista tapaa kolmikoiden muodostamiseen: online ja offline. Online-louhimisessa kolmikot muodostetaan häviöfunktiossa koulutusprosessin aikana. Offline-louhinnassa etsitään sopivia kolmikoita analysoimalla mallin tulosteita jaksojen välissä. Mallin painoja ei päivitetä offline-louhinnan aikana. (Schroff ym. 2015)

FaceNet-mallin koulutuksessa käytettiin Triplet loss- häviöfunktiota (kaava 1). (Schroff ym. 2015) Tässä työssä yritettiin ImageNet:n painoilla varustetun mallin kouluttamista samalla funktiolla, mutta jokaisella yrityksellä kohdattiin katoavan gradientin ongelma ja malli lakkasi oppimasta. Häviötä voitiin käyttää onnistuneesti vasta kun malli oli esikoulutettu kirjoittajan laatimalla häviöllä. Triplet loss-häviön käyttö on siis erittäin riippuvainen painojen oikeasta alustamisesta. Epäonnistuminen Triplet loss-funktion käytössä ajoi kirjoittajan suunnittelemaan uuden häviöfunktion, jota käyttämällä koulutus tapahtui huomattavasti vakaammin (kuva 9).

Kaava 1. Triplet loss

$$loss = \ sum(\max\left(d_p^i - d_n^i + \ \alpha, 0\right)))$$

Jossa d_p^i on positiivinen parin etäisyys, d_n^i on negatiivisen parin etäisyys ja α kynnysarvo.

3.3 Euklidinen etäisyys

Euklidinen etäisyys tarkoittaa kahden pisteen välistä etäisyyttä. Moniulotteisessa etäisyydessä se on neliöjuuri kahden vektorin p ja q alkioiden i erotusten neliöiden summasta. Etäisyys lasketaan Pythagoraan lausetta käyttämällä (kaava 2).

Kaava 2. Euklidinen etäisyys

$$\sqrt{\sum_{i=1}^{n}(p_i \ - \ q_i)^2}$$

jossa p_i on ensimmäisen vektorin alkio ja q_i toisen vektorin alkio.

3.4 L2-normalisointi

L2-normalisointi tarkoittaa vektorin arvojen normalisointia vektorin pituusnormin avulla. Euklidinen vektorinormi määritetään Pythagoraan lauseen mukaisesti, ottamalla neliöjuuri vektorin alkioiden neliöiden summasta (kaava 3).

Kaava 3. Euklidinen vektorinormi

$$\left|\left|x\right|\right|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x_1^2 + x_2^2 + \cdots x_n^2}$$

Vektori normalisoidaan jakamalla sen alkiot vektorinormilla (kaava 4). L2-normalisoidun vektorin alkioiden neliöiden summa on yksi, olettaen, että vektorin pituus ei ole nolla.

Kaava 4. Vektorin l2-normalisointi

$$x_{norm} = \frac{x}{\left|\left|x\right|\right|_2}$$

Tästä seuraa, että kahden normalisoidun vektorin (a ja b), joiden kummankin pituus on suurempi kuin yksi, välinen euklidinen etäisyys on maksimissaan $\sqrt{2}$ (kaava 5).

Kaava 5. Euklidisen etäisyyden ääriarvo

$$d_{max} = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} = \sqrt{2} \approx 1.4142, \text{jossa a on } \{0, 1\} \text{ ja b } \{1, 0\}.$$

Euklidinen etäisyys voidaan nyt normalisoida alueelle 0–1, kun tiedetään sen saama maksimi ääriarvo (kaava 6).

Kaava 6. Euklidisen etäisyyden maksimiarvon määritys

$$d_{norm} = \frac{\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}}{\sqrt{2}} = \frac{(a_1 - b_1)^2 + (a_2 - b_2)^2}{2} = 1$$

4 Neuroverkon kehittäminen

Tässä kappaleessa kuvataan opinnäytetyön projektiosan toteutusta. Työssä käytettiin Googlen Colaboratory-ympäristöä neuroverkon koulutukseen. Ympäristö asetti rajoitteita aineiston sekä koulutettavan neuroverkon arkkitehtuurin valintaan. Ympäristön tarjoamat resurssit vaikuttivat myös rajoittavasti koulutuksen kestoon.

Colaboratory:n työmuisti oli rajallinen eikä kuvia kaikkia kuvia olisi voitu ladata siihen. Kuvat ladattiin sen vuoksi opetusprosessissa yksitellen levyltä. Käytettävä työmuisti rajoitti myös valittavan neuroverkon arkkitehtuurin suhteellisen pieneksi. Ympäristö katkaisi koodin suorituksen päivittäinen käyttörajan tullessa vastaan. Mallin koulutus jakaantui sen vuoksi viiteen jaksoon, joissa mallin koulutusta jatkettiin edellisestä pisteestä.

4.1 Aineiston valinta ja esikäsittely

Neuroverkon koulutukseen haluttiin valita mahdollisimman suuri julkisesti saatavilla oleva tietokanta, jota olisi vielä helppo käsitellä kotitietokoneella. Tietokannan haluttiin sisältävän mahdollisemman paljon henkilöllisyyksiä sekä kohtuullisen paljon kuvia jokaisesta henkilöstä. Käytettäväksi aineistoksi valikoitui lopulta CelebA-tietokanta. Tietokannasta oli julkaistu eri versioita, ja tässä työssä käytettiin versiota, jossa kuvat oli rajattu kasvojen alueelle ja kasvot kohdistettu pystysuoraan. Aineistoon oli sisällytetty henkilöllisyyksien luokitteluun soveltuva tiedosto.

CelebA-tietoaineisto sisältää sen julkaisijoiden mukaan noin 10 000 eri henkilöllisyyttä, joista jokaisesta on noin 20 eri kuvaa. Aineisto sisältää yhteensä noin 200 000 kuvaa. Tietoaineisto oli luotu tutkimusta varten, jossa pyrittiin neuroverkon avulla luokittelemaan kuvia kasvoattribuuttien perusteella. (Liu, Luo, Wang & Tang, 2015)

Ennen neuroverkon koulutusta tietoaineisto tarkastettiin manuaalisesti. Aineistoa tarkastettiin silmämääräisesti henkilöllisyys kerrallaan ja virheelliset luokitukset kirjattiin ylös. Joissakin kuvissa havaittiin korruptiota tai kasvojen olevan liian peitossa. Edellä mainitut kuvat kirjattiin myös ylös, jotta ne voitaisiin jättää koulutuksen ulkopuolelle.

Neuroverkon ensimmäisen koulutuksen jälkeen henkilöllisyyksiä tarkastettiin uudelleen käyttämällä neuroverkkoa. Jokainen kuva syötettiin yksitellen neuroverkkoon ja niiden vektoritulosteet otettiin talteen. Kerättyjen tulostevektorien kesken laskettiin euklidinen etäisyys ja poikkeavien etäisyyksien kanssa korreloivat kuvat tarkastettiin. Jos kuvien luokittelussa havaittiin virheitä, niille etsittiin oikea henkilöllisyys lähimmistä naapureista. Neuroverkko koulutettiin uudelleen alusta jokaisen tarkastuksen jälkeen.

Ylisovittamisen havaitsemiseksi ja ehkäisemiseksi lopullisesta aineistosta erotettiin henkilöllisyydet, joilla oli aineistossa vain yksi kuva. Näitä kuvia käytettiin neuroverkon testaamiseen opetusprosessin aikana.

Tietoaineiston suodatuksen jälkeen opetusdataan jäi 10 132 identiteettiä ja 200 618 kuvaa. Testausdatan kooksi määräytyi 1 777 identiteettiä eli yhteensä 1 777 kuvaa. Hylättyjä kuvia kertyi yhteensä 204 kappaletta (taulukko 1).

Aineiston kuvat olivat kooltaan 218*178, mikä ei soveltunut työssä käytettävälle mallille. Kuvat esikäsiteltiin rajaamalla niiden keskeltä 160*160 kokoinen alue.

Taulukko 1. Kooste koulutusaineistosta

Section	Persons	Images
Train	10 132	200 618
Test	1 777	1 777
Not included	-	204
Total	11 909	202 599

4.2 Neuroverkon valinta

Projektityössä käytettävän neuroverkon valintaan sovellettiin seuraavia kriteereitä; arkkitehtuurin piti olla valmiiksi kehitetty ja soveltuva sekä tehtävään että käytettävän aineiston laajuuteen. Arkkitehtuurit kuten VGG16 ja InceptionV3 osoittautuivat liian suuriksi opettaa Googlen Colaboratory-ympäristössä, mutta mobiiliapplikaatioiden tarpeisiin kehitetyt MobileNet-arkkitehtuurit havaittiin käyttökelpoisiksi. MobileNet-arkkitehtuureista valittiin kirjoitushetkellä viimeiseksi julkaistu MobileNetV2-arkkitehtuuri.

MobileNetV2-mallissa oli mahdollista säätää syötteen kokoa sekä arkkitehtuurin laajuutta syvyys- ja leveyssuunnassa aineistolle sopivammaksi. Malli koulutettiin käyttämällä siirto-oppimista; painoina käytettiin ImageNet-tietoaineiston luokitustehtävästä saatuja painoja. Tensorflow-kirjaston tarjosi painoja eri kuvako'oille, joista valittiin sopivat 160x160x3 kokoisille kuville.

4.3 Häviöfunktio

Neuroverkon koulutus tapahtuu minimoimalla häviö- tai virhefunktion palauttama arvo muuttamalla verkon painoarvoja jokaisen iteraation jälkeen. Työssä käytettävän häviöfunktion tavoitteena on minimoida positiivisten parien ja maksimoida negatiivisten parien välinen euklidinen etäisyys.

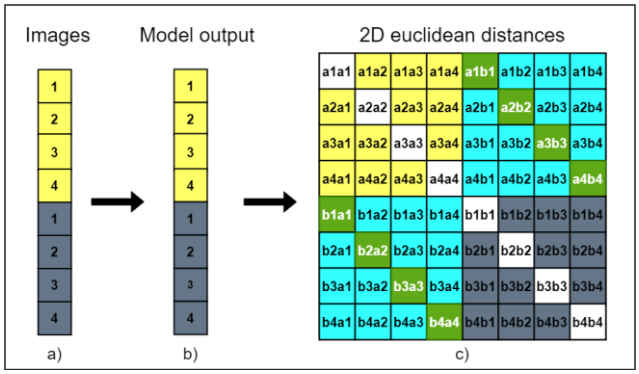
Normalisoitu etäisyys lasketaan kaikkien erän tulosteiden välille käyttämällä euclidean-funktiota. Funktio palauttaa 2-uloitteisen matriisin, jossa arvo koordinaatissa (x, y) vastaa kahden tensorin etäisyyttä mallin tulosteen indekseistä x ja y (kuva 6).

Kuva 6. Euklidisen etäisyyden laskeva funktio

```
def euclidean(p):
    euclidean_2d = tf.reduce_sum(tf.square(p - p[:, None]), axis=-1) / 2.0
    return euclidean_2d
```

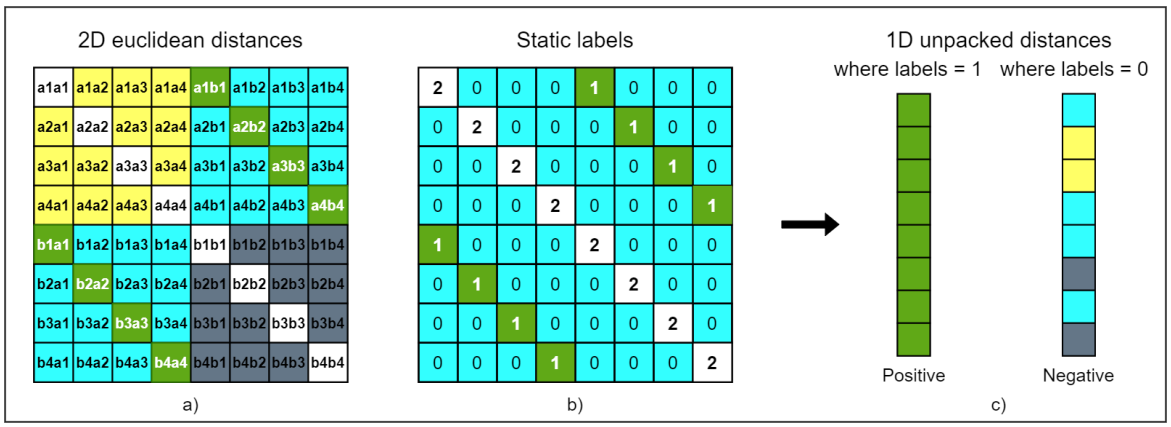
Edellytyksenä matriisin järjestyksen säilymiselle on syötteen järjestyksen varmistaminen. Koulutuserän kuvaparit sijoitetaan syötteen indekseihin siten, että ne ovat aina puoli eräkokoa erossa toisistaan. Järjestämällä syöte edellä mainitulla tavalla helpottuu positiivisten ja negatiivisten etäisyyksien poimiminen myöhemmässä vaiheessa. Kuvassa 7 havainnollistetaan, miten eri vektorien väliset etäisyydet sijoittuvat matriisiin.

Kuva 7. Etäisyysmatriisin muodostus tulosteesta



Etäisyysmatriisista suodatetaan jokaisen rivin pienimmät negatiiviset etäisyydet. Nämä arvot kuvaavat koulutuserän huonoiten luokiteltuja eri identiteeteistä koostuvia pareja. Suodatus tapahtuu käyttämällä erityisesti järjestettyä suodinmaskia. Maskissa arvo nolla kuvaa negatiivista, yksi positiivista ja kaksi identtistä paria. Jokaista mallin tulosteen alkiota kohden on nyt valittuna yksi lähimpänä oleva negatiivinen pari sekä positiivinen pari. Valitussa louhintastrategiassa käytetään siis kaikkia aiemmin mainittuja negatiivisten parien kategorioita (kuva 8).

Kuva 8. Etäisyysmatriisin suodatus



Ensimmäisillä koulutuskerroilla mallin huomattiin hajauttavan negatiiviset parit nopeasti kauas toisistaan, jolloin positiivisten piirteiden lähentyminen vaikeutui. Ongelma pyrittiin ratkaisemaan rajaamalla negatiiviset etäisyydet alueelle 0–0.5 ja sen jälkeen normalisoimalla etäisyydet takaisin alueelle 0–1 (kaava 7).

Kaava 7. Negatiivisten arvojen normalisointi

$$neg = \frac{\min(neg, 0.5)}{0.5}$$

Lopuksi lasketaan Tensorflow-kirjaston binary_crossentropy-funktion avulla lopullinen, binäärinen logistinen regressiohäviö. Kaavassa tavoitearvo y_i saa arvon nolla, jos etäisyys p_i on positiivinen ja arvon yksi etäisyyden ollessa negatiivinen. Häviöfunktio palauttaa lopuksi laskettujen häviöiden keskiarvon (kaava 8).

Kaava 8. Binäärinen häviöfunktio

$$\log_{\text{loss}} = \frac{1}{N}\sum_{i=1}^{N} -\left(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i)\right), jossa\ y_i = \{0 \vee 1\}$$

Häviöfunktio ottaa argumentteina suodinmaskin t sekä mallin syötteen p. Funktiota kutsutaan automaattisesti Tensorflow-opetusympäristön toimesta. Häviöfunktion koodi on esitelty kokonaisuudessaan kuvassa 9.

Kuva 9. Häviöfunktio

```
def loss(t, p):
    # unpack euclidean matrix
    euclidean_2d = euclidean(p)
    # set positives > 1 not to include them in min negatives
    neg = tf.reduce_min(euclidean_2d + t, axis=-1)
    # clip negative to avoid sparsing
    neg = tf.minimum(neg, 0.5) / 0.5
    # positive distances
    pos = tf.reshape(tf.boolean_mask(euclidean_2d, t == 1), [-1])
    # make labels
    pos_lbls, neg_lbls = tf.zeros_like(pos), tf.ones_like(neg)
    # take binary crossentropy
    return binary_crossentropy([pos_lbls, neg_lbls], [pos, neg])
```

4.4 Koulutuksessa käytetty metriikka

Neuroverkon koulutuksessa käytettiin kolmea eri mittarifunktiota keräämään dataa koulutuksen edistymisestä. Ensimmäinen funktio seurasi positiivisten etäisyyksien keskiarvoa, toinen negatiivisten etäisyyksien keskiarvoa ja kolmas neuroverkon tarkkuutta. Funktioita kutsutaan jokaisen koulutuserän lopussa automaattisesti Tensorflow-koulutusympäristön toimesta. Jakson aikana funktioiden arvojen juokseva keskiarvo näytetään käyttäjälle.

Positiivisten etäisyyksien keskiarvo laskettiin mean_pos-funktiolla. Neuroverkon yhtenä tavoitteena oli minimoida positiivisten parien etäisyys ja siksi sen keskiarvon muuttumista haluttiin seurata koulutuksen aikana. Kuvassa 10 esitellään funktion koodi kokonaisuudessaan.

Kuva 10. Positiivisten etäisyyksien keskiarvon laskeva funktio

```
def mean_pos(t, p):
    # unpack euclidean matrix
    euclidean_2d = euclidean(p)
    # get average of positive distances
    return tf.reduce_mean(tf.boolean_mask(euclidean_2d, t == 1))
```

Negatiivisten etäisyyksien keskiarvo laskettiin mean_neg-funktiolla. Neuroverkon toisena tavoitteena oli maksimoida negatiivisten parien etäisyys ja siksi sen keskiarvon muuttumista haluttiin seurata koulutuksen aikana. Kuvassa 11 esitellään funktion koodi kokonaisuudessaan.

Kuva 11. Negatiivisten etäisyyksien keskiarvon laskeva funktio

```
def mean_neg(t, p):
    # unpack euclidean matrix
    euclidean_2d = euclidean(p)
    # get average of negative distances
    return tf.reduce_mean(tf.boolean_mask(euclidean_2d, t == 0))
```

Neuroverkon tarkkuutta mitattiin accuracy-funktiolla. Funktio laskee etäisyysmatriisista rivien keskiarvollisen määrän, joissa rivin pienin etäisyys on positiivinen. Tarkkuusfunktio laskee siis positiivisten parien tunnistustarkkuutta. Kuvassa 12 esitellään funktion koodi kokonaisuudessaan.

Kuva 12. Tarkkuusfunktio

```
def accuracy(t, p):
    # unpack euclidean matrix
    euclidean_2d = euclidean(p)
    temp = euclidean_2d + tf.cast(t == 2, p.dtype)
    # average of rows where positive distance is the smallest value
    acc = tf.equal(tf.argmin(temp, axis=-1), tf.argmax(t == 1, axis=-1)),
    return tf.reduce_mean(tf.cast(acc, p.dtype))
```

4.5 Koulutuksessa käytetyn datan generointi

Työssä käytettävään mallin koulutusrutiiniin sovelletaan kustomoitua datageneraattoria, joka suoratoistaa dataa koulutusprosessiin. Generaattori on iteroitava luokka, jonka tehtävänä on lukea kuvat, tehdä niihin satunnaisia muokkauksia ja lopuksi esikäsitellä niiden pikseliarvot alueelta 0–255 alueelle -1–1. Generaattorin tehtävänä on myös sekoittaa dataa niin että jokainen erä sisältäisi mahdollisimman erilaisen joukon luokkia.

Datageneraattorin init-metodi ottaa argumentteina eräkoon, kuvien datapolun, polun identiteetit sisältävään csv-tiedostoon sekä valinnan siitä käytetäänkö generaattoria testaukseen vai koulutukseen (kuva 13).

Kuva 13. Generaattorin init-metodi

```
class ImageGen:

    def __init__(self, batch_size, path, identites, subset="train"):
        self.path = path
        self.subset = subset
        self.bs = batch_size

        self.lock = threading.Lock()
        self.data = self.unpack_data(identites, path)
        self.rand_gen = self.get_rand_generator()
        self.it = self.get_iterator()
```

Seuraavaksi esitellään metodi, joka purkaa kuvien polut ohjelmamuistiin henkilöllisyyksittäin. Polut talletetaan sanakirja-tietotyyppiin, jonka avaimina käytetään henkilöllisyyksiä. Sanakirjan arvot ovat ikuisesti iteroitavia, satunnaisia kahden polun polkuyhdistelmiä palauttavia generaattoriobjekteja. Testattaessa generaattorit palauttavat saman polun tuplana. Ero johtuu siitä, että testaukseen on varattu vain henkilöllisyyksiä, joilla on aineistossa vain yksi kuva (kuva 14).

Kuva 14. Generaattorin datanpurku-metodi

```
def unpack_data(self, identities_path, unzip_path):
    # read identity_CelebA
    df = pandas.read_csv(identities_path)
    # get filenames for each identity except bad images labelled as "0"
    df = df.loc[df['identity'] != 0]
    # get unique classes
    unique = np.unique(df["identity"])
    # make dictionary of identities and related images
    identity_dict = {}

    for class_name in tqdm(unique):
        filenames = df[np.in1d(df['identity'].values, [class_name])]["filename"]
        imgs = list(map(lambda x: os.path.join(unzip_path, x), filenames))

        # train images
        if self.subset == "train" and len(filenames) >= 2:
            combs = self.chunker(imgs, 2)
            identity_dict[class_name] = combs

        # test images
        if self.subset != "train" and len(filenames) == 1:
            identity_dict[class_name] = cycle([[imgs[0], imgs[0]]])

    return identity_dict
```

Generaattorin lukemia kuvia muokataan ennen niiden siirtämistä neuroverkolle. Muokkaukseen käytetään Albumentations-kirjastoa, jonka avulla voidaan yhdistää satunnaisia muokkausoperaatioita. Operaatioiksi valittiin horisontaalinen kääntö, valkoinen kohina, saturaation muutos, siirto-skaalaus-kierto sekä satunnainen kirkkauden ja kontrastin muutos. Muokkausoperaatioita suoritettiin eri todennäköisyyksillä ja satunnaisilla parametreillä. Kuvassa 15 esitellään funktio, jossa määritellään kuvien muokkaukseen käytetyt operaatiot.

Kuva 15. Kuvanmuokkaukset määrittävä metodi

```
@staticmethod
def get_rand_generator():
    transform = A.Compose([
        A.HorizontalFlip(p=0.5),
        A.GaussNoise(p=0.5),
        A.HueSaturationValue(hue_shift_limit=20,
                             sat_shift_limit=30,
                             val_shift_limit=20,
                             p=0.5),
        A.ShiftScaleRotate(rotate_limit=15,
                           scale_limit=(-0.16, 0.16),
                           shift_limit=0.15,
                           border_mode=cv2.BORDER_CONSTANT,
                           p=1.0),
        A.RandomBrightnessContrast(brightness_limit=0.25, p=1.0)
    ])
    return transform
```

Generaattorin tehtävänä on syöttää neuroverkolle kuvien lisäksi myös leimoja, joiden perusteella häviöfunktio laskee koulutuserän häviön. Tässä työssä esitellyssä koulutusmetodissa ei anneta leimoja yksittäisille syötteen alkioille kuten tavallisessa luokitustehtävässä vaan jokaiselle syötetylle erälle annetaan sama suodinmaskina toimiva eräleima. Eräleimaa käytetään positiivisten ja negatiivisten etäisyyksien erottamiseen häviöfunktion laskemasta etäisyysmatriisista. Kuvassa 16 esitellään funktio, jossa eräleima muodostetaan.

Kuva 16. Eräleiman palauttava metodi

```
def create_static_label(self):
    y = np.tile(np.arange(self.bs), 2)
    y = np.equal(y, y[:, None]).astype(float)
    y[np.diag_indices(self.bs * 2)] = 2
    return y
```

Generaattorilla on itsellään myös sisäinen generaattoriobjekti, jossa koulutusrutiinille esitettävä data valmistellaan ja pakataan. Generaattoriluokka iteroi sisäistä generaattoria aina kun sen next-metodia kutsutaan. Sisäisen generaattorin määrittelevä metodi esitellään kuvassa 17.

Kuva 17. Sisäisen generaattorin määrittelevä metodi

```
def get_iterator(self):

    # data chunker for sampling classes
    chunker = self.chunker(list(self.data.keys()), self.bs)

    # holders for batch data
    x = np.zeros((self.bs * 2, 160, 160, 3), dtype=np.float32)
    y = self.create_static_label()

    for batch in chunker:
        all_paths = [next(self.data[cls]) for cls in batch]
        for index, (path1, path2) in enumerate(all_paths):
            x[index] = self.get_image(path1)
            x[index + self.bs] = self.get_image(path2)

        x = preprocess_input(x)
        yield x, y
```

Neuroverkon oppimisen kannalta on tärkeää sekoittaa dataa siten että jokaisessa erässä olisi mahdollisimman usein mahdollisimman erilainen joukko luokkia. Generaattorin toiminnan yksinkertaistamiseksi oli myös toivottavaa, että eräkoko pysyisi jakson aikana vakiona. Ratkaisuksi kehitettiin metodi, jossa data palautetaan kutsuttaessa eräkoon kokoisissa paloissa. Metodissa kaikki data sekoitetaan aina jakson lopussa (kuva 18). Paloittelu tehdään siirtämällä vanhoja alkioita eräkoon verran vasemmalle ja palauttamalla datan alusta eräkoon verran uusia alkioita. Edellä mainitulla menetelmällä voidaan välttää datan toistuminen jakson aikana. Lisäksi voidaan välttää erikokoisten erien esiintyminen jakson lopussa, jos data ei jakaannu tasan eräkoon kokoisiin paloihin.

Kuva 18. Datan sekoituksesta vastaava metodi

```
@staticmethod
def chunker(lst, bs):
    while True:
        np.random.shuffle(lst)
        for _ in np.arange(np.ceil(len(lst) / bs)):
            lst = np.roll(lst, bs)
            yield lst[:bs]
```

4.6 Neuroverkon koulutus

Opinnäytetyössä käytetyn neuroverkon runkona toimii MobileNetV2-konvoluutioneuroverkko, joka alustetaan ImageNet-aineiston luokittelusta syntyneillä painoilla.

Runkoon on liitetty yksi täysin kytketty kerros, jonka tarkoituksena on pienentää tulosteen tilavuus halutun kokoiseksi. Täysin kytketyssä kerroksessa käytetään epälineaarista relu-saranafunktiota, joka korvaa negatiiviset arvot nolla-arvolla. Ulostulokerroksena käytetään kustomoitua lambda-kerrosta, joka normalisoi syötteensä käyttämällä l2-normalisointia.

Kuvassa 19 esitellään neuroverkon arkkitehtuurin muodostava funktio. Funktio ottaa sisään argumenttina tiedostopolun alustettaviin painoihin. Jos polkua ei ole annettu ladataan oletusarvoisesti ImageNet-painot. Toinen argumentti, jonka funktio ottaa, on täysin kytketyn kerroksen tilavuus, joka tässä työssä on 256. Viimeinen argumentti on kerroin, joka määrittää rungon suhteellisen leveyden kontrolloimalla konvoluutiokerrosten suotimien määrää. Tässä työssä kertoimen arvo on 0,75.

Arkkitehtuurin määrittämisen jälkeen malli kootaan koulutuksessa käytettävällä optimointialgoritmilla, häviöfunktiolla ja metriikalla. Optimoinnissa käytettiin adam-algoritmia, jonka oppimisnopeus asetettiin arvoon 0,001.

Kuva 19. Neuroverkon alustaminen

```
def get_model_v2(weights=None, size=256, alpha=1.0):
    model = MobileNetV2(input_shape=(160, 160, 3), alpha=alpha, include_top=False, pooling="max")
    x = Dense(size, activation="relu")(model.layers[-1].output)
    x = Lambda(lambda d: tf.math.l2_normalize(d, axis=1), name="l2-norm")(x)
    model = Model(inputs=[model.input], outputs=[x])

    if weights:
        model.load_weights(weights, by_name=True, skip_mismatch=True)

    return model


model = get_model_v2(weights=None, size=256, alpha=0.75)
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss=loss,
    metrics=[mean_pos, mean_neg, accuracy])
```

Koulutuksessa käytettävien datageneraattoreiden muodostaminen esitellään kuvassa 20. Testauksessa ja koulutuksessa käytetään kummassakin eräkokoa 100. Generaattorit tuottavat kutsuttaessa 200 kuvaa, jotka kuuluvat 100:aan eri luokkaan.

Kuva 20. Generaattoreiden esittely

```
identities = r"/content/drive/My Drive/celeba/updated_identities.csv"
unzip_path = r"img_align_celeba"
save_path =  r"/content/drive/MyDrive/test3/session_b/session_5"

bs_train = 100
bs_val = 100

tr_gen = ImageGen(bs_train, unzip_path, identities, subset="train")
val_gen = ImageGen(bs_val, unzip_path, identities, subset="test")
```

Oppimisprosessin analysoimiseksi esitellään CSVLogger-objekti, joka tallettaa metriikan csv-tiedostoon jokaisen erän jälkeen (kuva 21). Tallennetun datan perusteella voidaan myöhemmin tarkastella opetuksen onnistumista ja piirtää havainnollistavia kuvaajia.

Malli tallennetaan jaksojen lopussa käyttämällä ModelCheckpoint-objektia. Tallentamisen ehtona on, että testatessa häviöfunktion palauttama arvo on pienempi kuin aikaisemmalla tallennuskerralla (kuva 21).

Kuva 21. ModelCheckpoint ja CSVLogger -objektin esittely

```
mcp = tf.keras.callbacks.ModelCheckpoint(
    filepath= r"%s/{epoch:02d}_{val_loss:.3f}.h5" % save_path,
    monitor="val_loss",
    verbose=1,
    save_best_only=True,
    mode="auto")

csv_log = tf.keras.callbacks.CSVLogger(
    filename=r"%s/history.csv" % save_path,
    separator=",",
    append=True)
```

Neuroverkon koulutus aloitetaan kutsumalla malliobjektin fit-metodia. Metodi ottaa argumentteina koulutus ja testaus -datan, askelten määrän jakson aikana, jaksojen määrän ja listan objekteista, joita kutsutaan koulutuksen aikana. Askelmääräksi on tässä työssä määritelty ylöspäin pyöristetty identiteettien määrän ja eräkoon jakotulos. Fit-metodin kutsuminen esitellään kuvassa 22.

Kuva 22. Mallin koulutuksen aloitus

```
history = model.fit(x=tr_gen,
                    steps_per_epoch=int(np.ceil(len(tr_gen) / bs_train)),
                    validation_data=val_gen,
                    validation_steps=int(np.ceil(len(val_gen) / bs_val)),
                    epochs=500, callbacks=[mcp, csv_log])

model.save(os.path.join(save_path, "final_model.h5"))
```

5 Neuroverkon analysointi

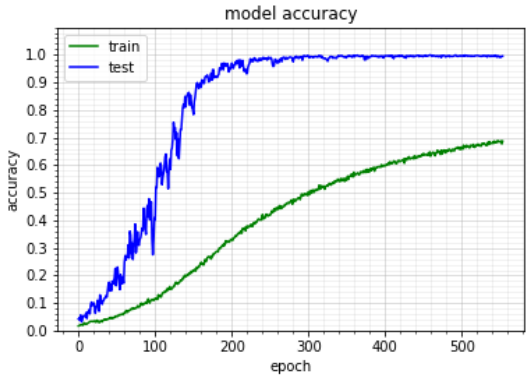
Neuroverkon toimivuutta analysoitiin tarkastelemalla koulutuksen aikana tallennettua dataa. Kerätystä datasta piirrettiin kuvaajia, joita tulkitsemalla pyrittiin saamaan vastaus seuraaviin kysymyksiin: kuinka hyvin malli oppi suoriutumaan tehtävästä, ilmenikö koulutuksen aikana ongelmia sekä olisiko mallia voinut kouluttaa lisää.

Neuroverkon laatua analysoitiin myös internetistä kerätyllä ulkoisella aineistolla. Aineisto käsiteltiin vastaamaan koulutusaineistoa rajaamalla kasvot keskelle kuvaa ja linjaamalla silmät vaakatasoon. Kerätty aineisto sisälsi 523 henkilöllisyyttä ja yhteensä 19992 kuvaa.

5.1 Koulutuksen analysointi

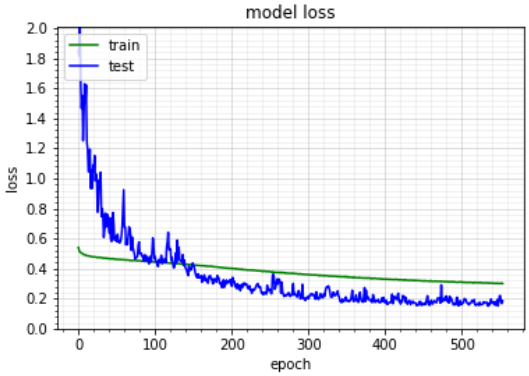
Tarkastelemalla tarkkuusfunktion palauttamaa dataa varmistuttiin oppimisprosessin stabiiliudesta. Tarkkuuden kehittymisestä jaksojen kuluessa piirrettiin kuvaaja (kuva 23). Kuvaajasta ei havaittu selkeitä viitteitä yli tai ali -sovittamista. Testaustarkkuuden huomattiin toisaalta nousseen erittäin lähelle 100 % jo jaksolla 300, mikä vaikeuttaa tarkemman arvion tekemistä. Tarkkuuden havaittiin nousevan jaksojen myötä tasaisesti sekä kouluttaessa että testatessa. Opetusprosessin lopettamishetkellä koulutustarkkuudessa havaittiin vielä kehittymistä, mikä saattaa viitata siihen, että mallin koulutusta olisi vielä voinut jatkaa.

Kuva 23. Tarkkuus opetusprosessissa



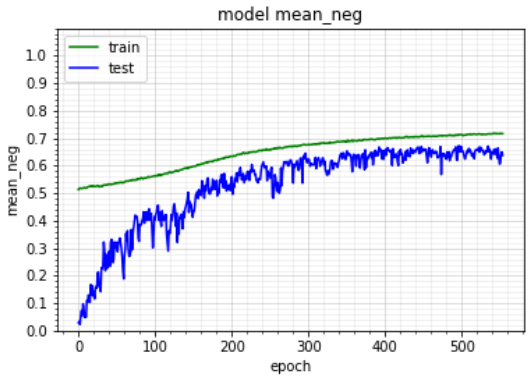
Tarkastelemalla häviöfunktion palauttamaa dataa varmistuttiin oppimisprosessin etenemisestä. Häviön kehittymisestä jaksojen kuluessa piirrettiin kuvaaja (24). Kuvaajasta ei havaittu selkeitä viitteitä yli tai ali -sovittamista. Häviössä tapahtuneet muutokset pienenivät jaksojen kuluessa eli mallin oppimisnopeus hidastui. Opetusprosessin lopettamishetkellä sekä koulutuksen että testauksen aikaisissa häviöissä havaittiin vielä jonkinasteista kehittymistä. Tämä vahvisti aikaisemmin tehtyä arviota mallin jatkokoulutuksen mahdollisuudesta.

Kuva 24. Häviö opetusprosessissa



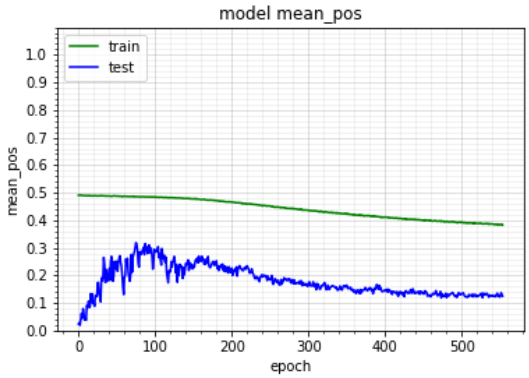
Tarkastelemalla opetusprosessin aikana kerättyä dataa negatiivisten etäisyyksien kehittymisestä voitiin varmistua siitä, että neuroverkko kykeni täyttämään ainakin toisen sille esitetyistä vaatimuksista; negatiivisten etäisyyksien kasvattamisen. Piirretystä kuvaajasta nähtiin etäisyyksien kasvun hidastuvan opetusprosessin loppua kohden (kuva 25). Tästä pääteltiin negatiivisten arvojen rajauksen toimineen häviöfunktiossa.

Kuva 25. Negatiiviset etäisyydet opetusprosessissa



Tarkastelemalla opetusprosessin aikana kerättyä dataa positiivisten etäisyyksien kehittymisestä voitiin varmistua siitä, että neuroverkko kykeni täyttämään toisenkin sille esitetyistä vaatimuksista, positiivisten etäisyyksien pienentämisen. Piirretystä kuvaajasta nähtiin etäisyyksien kutistumisen jatkuvan tasaisena opetusprosessin loppuun (kuva 26). Käyrien ei havaittu kuitenkaan täysin tasoittuneen, mikä vahvisti edelleen arviota jatkokoulutuksen mahdollisuudesta.

Kuva 26. Positiiviset etäisyydet opetusprosessissa



5.2 Tarkkuuden analysointi ulkoisella aineistolla

Neuroverkon käytettävyyttä demosovelluksessa analysoitiin internetistä kerätyllä ulkoisella aineistolla. Analysoimalla mallin toimivuutta pyrittiin löytämään esimerkkisovellukseen sovellettava tunnistuskynnys. Optimaalisen tunnistuskynnyksen käyttämisen arveltiin vähentävän väärien positiivisten tulosten syntymistä.

Kun mallin suoriutumista analysoitiin ensimmäistä kertaa, havaittiin tarkkuuden olevan verrattain pieni. Tarkkuuden kasvattamiseksi tehtiin useita kokeiluja eri algoritmeilla. Kokeilujen perusteella päädyttiin käyttämään algoritmia, jossa kasvontunnistukseen käytetään useita lähdekuvia (kaava 9).

Mallille syötettiin n määrä kuvia kustakin henkilöstä h. Mallin tulostamat d ulotteiset vektorit henkilöstä h summattiin yhteen niin, että ulottuvuus d säilyi. Operaation tuloksena saatu summavektori l2-normalisoitiin. Syntyneitä summavektoreita vertailtiin lopulta euklidisesti toisiinsa ja tulokset kirjattiin ylös. Testauksessa summavektorien muodostamiseen käytettiin kuvien kappalemääriä 1,2,3,5 ja 7. Lähdekuvien kombinaatioista muodostettuja summavektoreita muodostettiin useita kappaleita jokaista luokkaa kohden.

Kaava 9. Summavektori

$$d_{hs} = l2\_normalize(sum(nd_h, axis = 1))$$

Tarkkuuden analysoinnissa käytettiin seuraavia metriikoita: todellinen tarkkuus (accuracy), top5-tarkkuus (top5) sekä turvallisen tunnistuksen tarkkuus (safe). Todellisella tarkkuudella tarkoitetaan niiden esiintymien keskiarvoista määrää, joissa vertailtava piste ja sen lähin vastaavuus kuuluvat samaan luokkaan. Top5-tarkkuudella tarkoitetaan niiden esiintymien keskiarvoista määrää, joissa vertailtavalla pisteellä on samaan luokkaan kuuluva pari lähimpien viiden pisteen joukossa. Turvallisen tunnistuksen tarkkuudella tarkoitetaan niiden esiintymien keskiarvoista määrää, jossa positiivinen etäisyys on korkeintaan yhtä kaukana kuin koko aineiston pienin negatiivinen etäisyys.

Tuloksia tarkastelemalla voitiin päätellä tunnistustarkkuuden kasvavan suhteessa summavektorin luomiseen käytettyjen kuvien määrän kanssa. Tunnistustarkkuus kasvoi kaikilla käytetyillä metriikoilla, kun kuvien määrää lisättiin. Todellinen tarkkuus kasvoi 13,96 prosenttiyksikköä arvosta 85,96 arvoon 99,92 kuvien määrän kasvaessa yhdestä seitsemään. Top5 tarkkuus kasvoi niin ikään 6,71 prosenttiyksikköä arvosta 93,29 % arvoon 100 %. Turvallisen tunnistuksen tarkkuus kasvoi 60,8 prosenttiyksikköä arvosta 20,80 % arvoon 81,60 % (taulukko 2).

Kuvamäärän kasvaessa huomattiin sekä positiivisten että negatiivisten etäisyyksien pienenevän. Tämän arveltiin johtuvan summavektorin muodostamisen keskiarvoistavasta luonteesta. Lähdevektorien määrän kasvaessa summavektorin kohinan magnitudi pienenee, mutta kohina siirtyy arvoihin hajautetusti.

Testauksessa otettiin ylös positiivisten ja negatiivisten etäisyyksien keskiarvo sekä minimi ja maksimi raja-arvot. Selvittämällä negatiivisten arvojen minimi saatiin selville turvallisen tunnistuksen maksimiraja. Tätä arvoa alemmat etäisyydet ovat suuremmalla todennäköisyydellä onnistuneita tunnistuksia.

Taulukko 2. Analysoinnin tulokset

batch size	1	2	3	5	7	Trends
*accuracy	85,96 %	95,27 %	98,32 %	99,55 %	99,92 %	
*top5	93,29 %	98,72 %	99,72 %	99,89 %	100,00 %	
*safe	20,80 %	41,90 %	55,40 %	71,00 %	81,60 %	
pos_{min}	0,001	0,015	0,007	0,008	0,008	
pos_{avg}	0,364	0,224	0,161	0,103	0,073	
pos_{max}	0,819	0,691	0,729	0,448	0,237	
neg_{min}	0,098	0,103	0,099	0,084	0,090	
neg_{avg}	0,600	0,539	0,510	0,482	0,468	
neg_{max}	0,964	0,921	0,911	0,886	0,859	

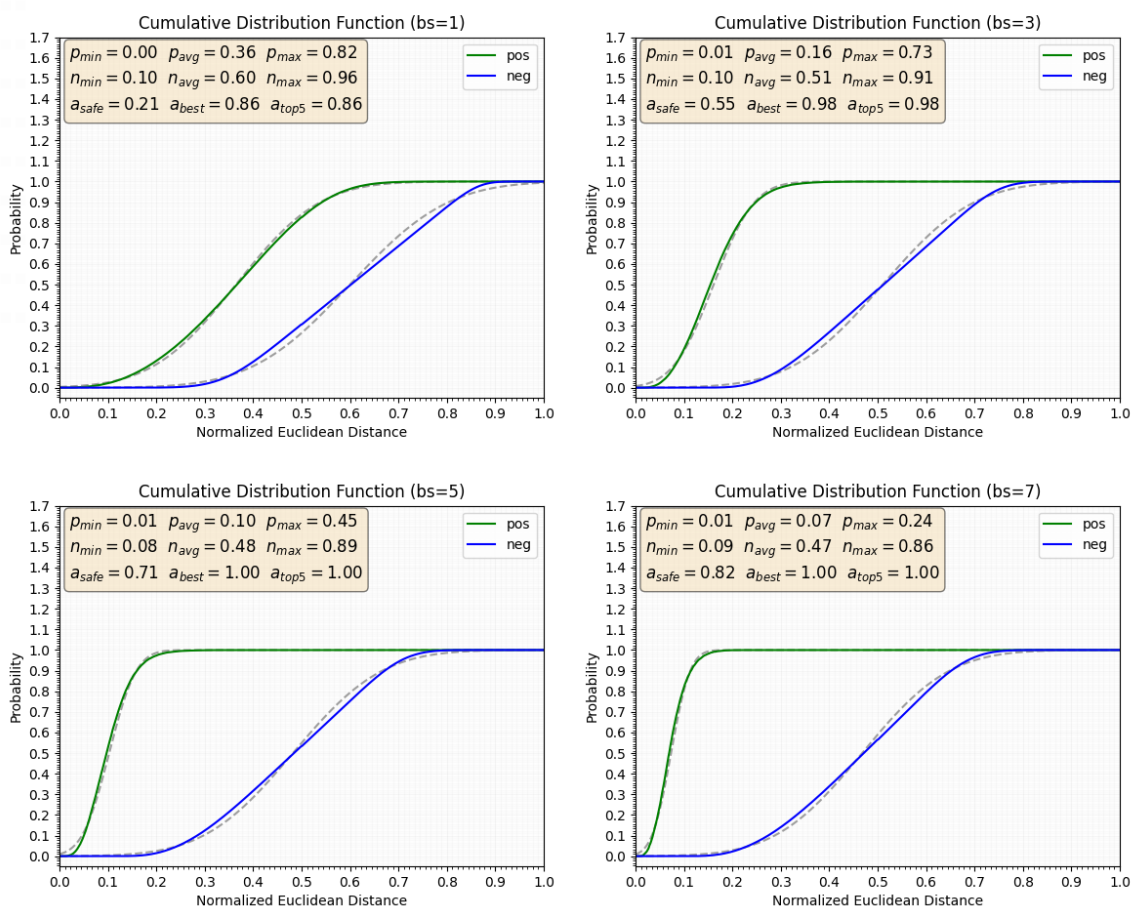
*accuracy: The best match belongs to the same class

*top5: There is a match belonging to the same class in the top 5 results

*safe: Average count of positive distances below minimum negative distance

Analysoinnissa kerätystä datasta piirrettiin kumulatiivisen todennäköisyysjakauman kuvaajat (kuva 27). Kuvaajia vertailemalla havaittiin positiivisten etäisyyksien painottuvan nolla-arvoa kohden nopeammin kuin negatiivisten. Todennäköisyyksien selkeä eriytyminen viittasi vahvasti korrelaatioon summavektorin muodostamisen ja tarkkuuden kasvamisen välillä.

Kuva 27. Kumulatiivinen todennäköisyysjakauma



5.3 Tunnistukseen vaikuttavat tekijät

Mallin koulutuksen jälkeen tutkittiin kasvonilmeiden vaikutusta tunnistusprosessiin. Mallille syötettiin kuusi erilaista kuvaa, joista malli tuotti tulosteita. Tulosteiden väliset euklidiset etäisyydet laskettiin; alettiin tutkia kuvien visuaalisten erojen ja etäisyyksien suhteita (taulukko 3). Valaistus pysyi kuvia otettaessa vakiona, joten sen vaikutusta pidettiin vähäisenä selittämään etäisyyksien eroja. Pään asento pysyi myös suhteellisen samana, mikä rajasi mahdollisen syyn kasvonilmeisiin. Aiemmassa analyysissa saatiin selville positiivisten etäisyyksien keskiarvo (0,364), jota käytettiin viitearvona suhteiden tutkimisessa.

Kahdessa kuvassa (b & f) koehenkilöllä oli suu auki ja neljässä muussa kiinni. Kahdessa kuvassa koehenkilö kurtisti kulmiaan (c & d) ja kahden ilmettä voi kuvata neutraaliksi (a & e). Yhdessä neutraaleista kuvista koehenkilön silmät olivat kiinni (e). Kuvan c etäisyydet olivat kaikista suurimpia ja kolme niistä ylitti viitearvon. Vastaavasti neutraalin ilmeen sisältävä kuva a suoriutui parhaiten. Etäisyydet olivat siis keskimäärin muita pienempiä, eikä yksikään niistä ylittänyt viitearvoa. Kuvan e keskimääräinen etäisyys oli kolmanneksi huonoin, vaikka se myös sisälsi neutraalin ilmeen. Toiseksi parhaiten pärjäsi kuva f, jossa koehenkilön suu ja silmät olivat auki. Vaikka parhaiten pariutuneet kuvat a ja f olivat kasvonilmeiltään erilaisia, olivat kasvojen koko ja sijainti samankaltaisia. Kummassakin kuvassa kasvot sijaitsevat lähellä kuvan keskipistettä.

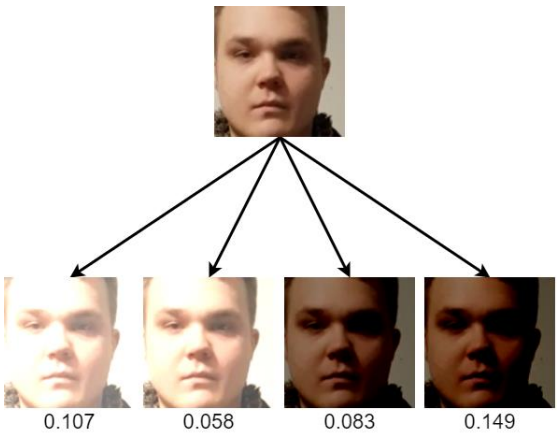
Analysoinnin tuloksena havaittiin, että neutraalit ilmeet keskitettynä kuvan keskelle tuottavat parhaimpia tuloksia. Kiinni olevat silmät, kasvojen kurtistaminen ja avonainen suu vaikeuttavat tunnistusta.

Taulukko 3. Kasvonilmeiden vaikutus euklidiseen etäisyyteen

	a	b	c	d	e	f
a	0.0	0.32	0.28	0.22	0.30	0.17
b	0.32	0.0	0.46	0.43	0.34	0.22
c	0.28	0.46	0.0	0.15	0.38	0.42
d	0.22	0.43	0.15	0.0	0.36	0.39
e	0.30	0.34	0.38	0.36	0.0	0.30
f	0.17	0.22	0.42	0.39	0.30	0.0

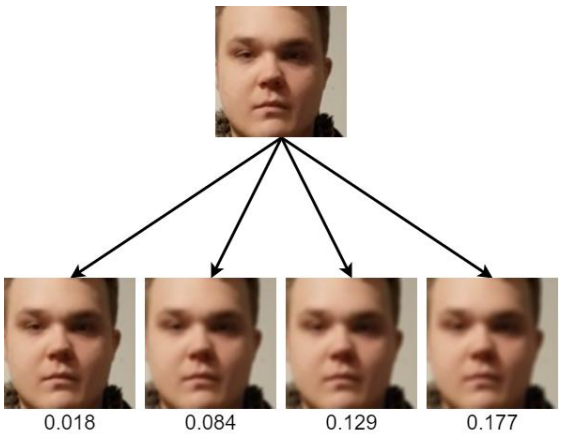
Tutkittiin kuvien kirkkauden vaikutusta tunnistusprosessiin. Ensiksi valittiin vertailukuva. Vertailukuvan kopioihin tehtiin satunnaisia kirkkauden muutoksia ja syntyneet kuvat syötettiin neuroverkolle. Lopuksi laskettiin euklidiset etäisyydet alkuperäisen ja muokattujen kuvien välillä (kuva 28). Etäisyyksiä tutkimalla huomattiin kirkkauden pienenemisen vaikuttavan huomattavasti etäisyyden kasvamiseen; mitä vähemmän kuvassa oli valoa, sitä suurempi oli etäisyys.

Kuva 28. Kirkkauden vaikutus euklidiseen etäisyyteen



Tutkittiin kuvien terävyyden vaikutusta tunnistusprosessiin. Ensiksi valittiin vertailukuva. Vertailukuvan kopioihin tehtiin satunnaisia sumennuksia ja syntyneet kuvat syötettiin neuroverkolle. Lopuksi laskettiin euklidiset etäisyydet alkuperäisen ja muokattujen kuvien välillä (kuva 29). Etäisyyksiä tutkimalla huomattiin sumentumisen vaikuttavan huomattavasti etäisyyden kasvamiseen; mitä sumeampi kuva sitä suurempi oli etäisyys.

Kuva 29. Terävyyden vaikutus euklidiseen etäisyyteen



6 Esimerkkisovellus

Sovelluksen tavoitteena on demonstroida kulunvalvontaa kasvontunnistukseen perustuen. Sovelluksessa on mahdollista lisätä, poistaa ja tunnistaa henkilö kamerakuvasta. Tunnistauduttaessa, henkilöstä kerätään useita kuvia, joista Mediapipe-kirjastoa käyttämällä rajataan kasvot sisältävä alue ja kallistus korjataan. Kuvat syötetään neuroverkolle ja tulosteista muodostetaan summavektori, jonka avulla etsitään vastaavuuksia tietokannasta. Vastaavuus saavutetaan, jos euklidinen etäisyys hakuvektorin ja jonkin tietokannassa olevan tallenteen välillä alittaa raja-arvon. Raja-arvona käytetään mallianalyysistä saatua pienintä negatiivista etäisyyttä.

Hyödyntämällä Mediapipe-kirjaston Face Mesh sovellusta, kamerakuvasta pyrittiin eristämään henkilöiden kasvot taustasta ja kohdistamaan ne. Kasvojen sijainti, koko ja kulma haluttiin pitää mahdollisemman vakaana tunnistustarkkuuden parantamiseksi. Käyttämällä

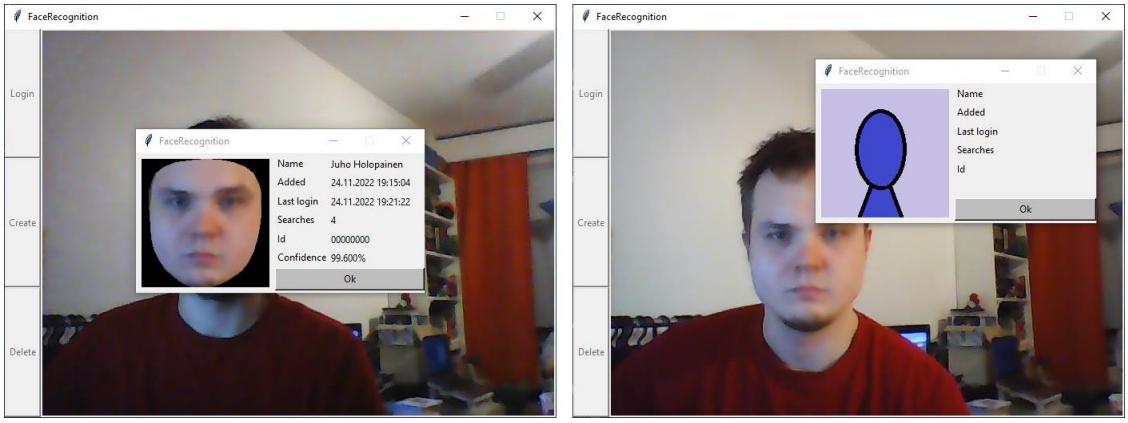
Face Meshiä saatiin selville kasvojen ääriviivat ja silmien sijainti. Ääriviivojen avulla saatiin eristettyä pelkät kasvot taustasta. Silmien sijainnin ja niiden välisen kulman perusteella saatiin kasvot kohdistettua niin että silmät olivat samalla tasolla. Silmien etäisyydestä laskettiin mahdollinen zoomauksen tarve.

6.1 Ohjelmakuvaus

Sovelluksen käynnistyessä ladataan neuroverkko muistiin taustaprosessissa. Samanaikaisesti avataan yhteys laitteen kameraan ja videokuvaa aletaan toistaa käyttäjälle. Kun malli on latautunut taustaprosessissa, avataan näyttöpainikkeiden lukitus ja sovellus on valmis käytettäväksi. Käyttäjälle on näkyvissä kolme painiketta: login, create ja delete.

Login-painikkeella aloitetaan tunnistautuminen. Käyttäjälle palautetaan vastaus haun onnistumisesta. Jos tunnistautuminen on onnistunut, palautetaan näytölle henkilön kuva, nimi sekä muut kirjautumistiedot. Haun epäonnistuessa, palautetaan käyttäjälle tyhjä oletusarvoinen tunnistautumiskortti (kuva 30).

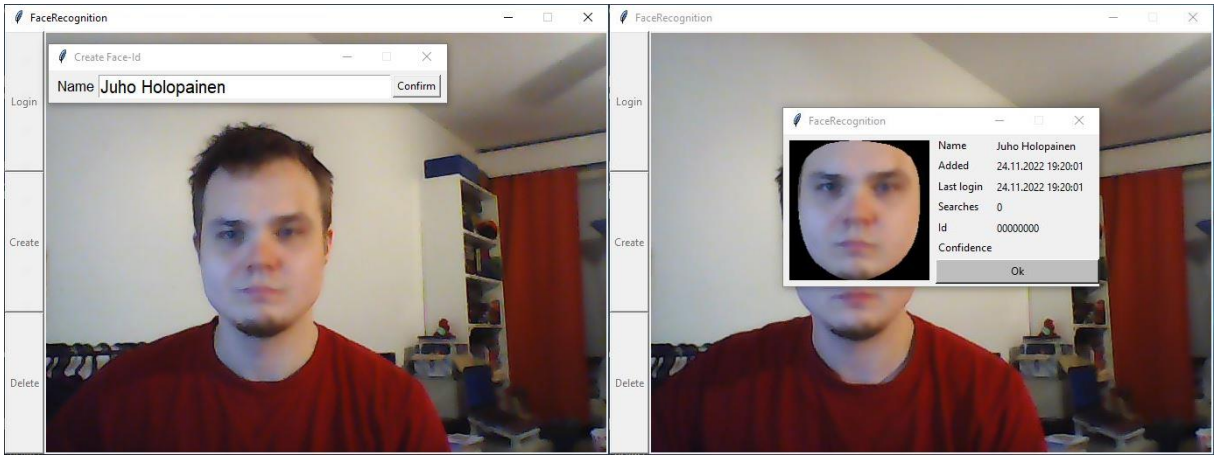
Kuva 30. Henkilön tunnistus



Create-painikkeella lisätään uusi henkilö tietokantaan. Painike avaa ensiksi ikkunan, johon käyttäjä kirjoittaa nimensä. Painamalla Confirm-painiketta sovellus aloittaa kuvien keräämisen käyttäjästä. Kun kuvia on kerätty riittävästi, lähetetään annettu nimi ja kuvat

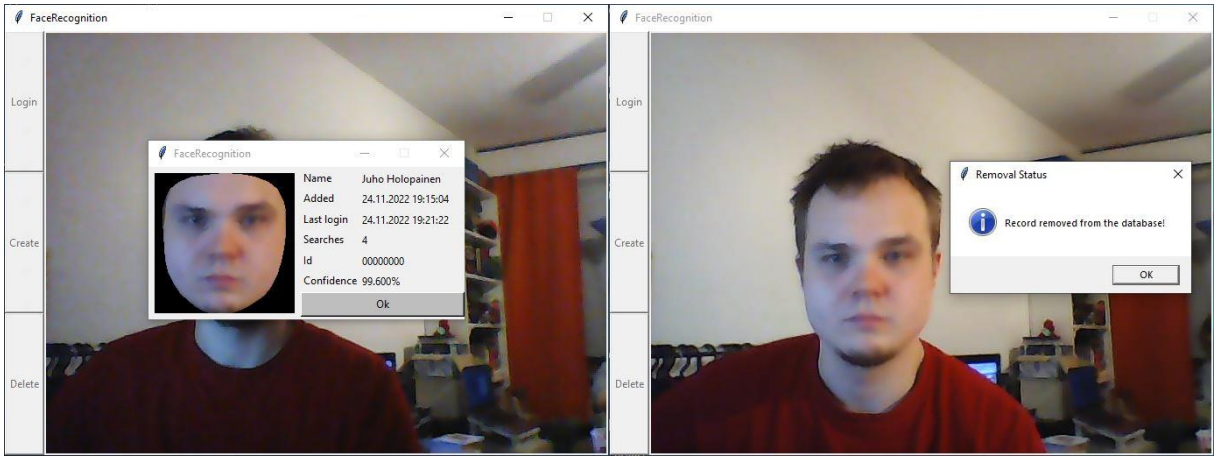
taustaprosessille. Uusi henkilö voidaan lisätä vain, jos tietokannasta ei löydetä vastaavuutta. Jos henkilön lisääminen onnistui, palautetaan näytölle henkilön kuva, nimi sekä muut kirjautumistiedot. Haun epäonnistuessa, palautetaan käyttäjälle tyhjä oletusarvoinen tunnistautumiskortti (kuva 31).

Kuva 31. Henkilön lisääminen



Delete-painikkeella poistetaan henkilö tietokannasta. Painike käynnistää kuvien keräämisen käyttäjästä, jonka jälkeen tietokannasta etsitään vastaavuutta, jos vastaavuus löydetään, voidaan henkilön tiedot poistaa. Onnistunut haku palauttaa näytölle käyttäjän tiedot. Tunnistuskortissa olevalla ok-painikkeella suoritetaan tietojen poisto loppuun. Käyttäjällä on mahdollisuus olla painamatta painiketta. Tietojen poisto keskeytyy, jos ikkuna suljetaan painamalla ikkunan oikeassa yläkulmassa olevaa rastia (kuva 32).

Kuva 32. Henkilön poistaminen



6.2 Kehitysehdotukset ja muut huomiot

Onnistunut kasvontunnistus kameran syötteestä on riippuvainen monesta eri tekijästä. Ongelmia voivat aiheuttaa neuroverkon tarkkuuden lisäksi myös haasteet kuvausympäristössä ja käytetyssä laitteistossa.

Valaistuksen, heijastumien ja kamerakulman negatiivisia vaikutuksia voidaan ehkäistä perinpohjaisella testaamisella asennusvaiheessa. Tunnistuspaikan valaistukseen voidaan tehdä muutoksia, jotta heijastumia ja varjoja syntyisi vähemmän. Valaistuksen suunnittelussa pitää ottaa huomioon ympäristöstä syntyvät häiriöt kuten kiiltävät pinnat ja auringonvalon vaikutus koko vuorokauden ajalta. Kuvakulman aiheuttamia ongelmia voidaan ehkäistä sijoittamalla kamera esimerkiksi kiskoon tai telineeseen, jolloin kuvakulmaa voidaan tarpeen vaatiessa muuttaa. Kuvakulmaa valitessa pitää ottaa huomioon myös eripituiset ihmiset.

Neuroverkkoon liittyviä ongelmia voidaan korjata kouluttamalla malli uudestaan kattavammalla datalla tai käyttämällä eri strategiaa. Uudelleenkoulutuksen lisäksi voidaan neuroverkon rakennetta muuttaa tai vaihtaa se kokonaan toiseen. Uudelleenkoulutuksen tarvetta voidaan arvioida esimerkiksi analysoimalla käyttöjakson aikana tapahtuneita virhetunnistuksia. Mahdollisia virhetunnistuksien aiheuttajia voivat olla esimerkiksi meikki, vaatetus, kampaus, ilmeet ja ihonväri.

Kameran matala resoluutio voi vaikuttaa tunnistukseen. Tarkkoja kasvonpiirteitä ei välttämättä voida erottaa kuvasta edes neuroverkon avulla. Matala kuvataajuus taas kasvattaa tarvittavan kuvamäärän keräämiseen kuluvaa aikaa.

Jotta kameran avulla suoritettava tunnistus olisi järkevää toteuttaa, pitäisi tunnistukseen kuluva aika olla suhteellisen lyhyt. Tunnistuksen kestoon vaikuttavat eniten kuvaamiseen kulutettu aika, tietokantaan sovellettava hakustrategia ja tunnistusvirheiden tiheys. Haun nopeuttamiseksi voisi mahdollisesti soveltaa jotain klusterointimenetelmää. Klusterimalli rajaa haun sen ennustaman luokan alkioihin, jolloin etäisyyksiä ei tarvitsisi laskea kaikkien aineiston alkioiden kanssa.

7 Yhteenveto

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa neuroverkkoa hyödyntävä kasvontunnistussovellus. Työhön kuului olennaisesti sovelluksessa käytettävän neuroverkon valinta, kouluttaminen ja testaaminen.

Työssä esiteltiin erityinen koulutusstrategia, joka koostui kustomoiduista datansyötöstä ja häviöfunktiosta. Analysoimalla lopullisen mallin tarkkuutta sekä koulutuksen aikana kerättyä dataa, huomattiin koulutusstrategian olevan erittäin stabiili ja tarkoitukseen soveltuva.

Tunnistustarkkuuden kasvattamiseksi kehitettiin algoritmi, joka havaittiin toimivaksi. Algoritmissa mallin tulosteet prosessoitiin muodostamalla niistä summavektori, minkä todettiin vähentävän kohinan vaikutusta virhetunnistuksiin.

Mallin todettiin täyttäneen sille annetut tavoitteet: samaan luokkaan kuuluvien tulosteiden euklidisen etäisyyden pienentäminen ja eri luokkiin kuuluvien tulosteiden kasvattaminen. Neuroverkon tarkkuuden katsottiin olevan riittävän hyvä luotettavaan henkilöiden tunnistamiseen.

Lähteet

Albumentations (2022). *Why Albumentations*. Haettu 5.12.2022 osoitteesta https://albumentations.ai/docs/introduction/why_albumentations/

Google. (n.d.). *Colaboratory*. *Frequently Asked Questions*. Haettu 29.11.2022 osoitteesta https://research.google.com/colaboratory/faq.html

Google Inc. (2022). *Tensorflow*. Haettu 28.11.2022 osoitteesta https://pypi.org/project/tensorflow/

He, K., Zhang, X., Ren, S. & Sun J. (2016). *Deep Residual Learning for Image Recognition,* Microsoft Research. https://doi.org/10.48550/arXiv.1512.0338

ImageNet (n.d.). *About ImageNet*. Haettu 1.12.2022 osoitteesta https://www.image-net.org/about.php

Kelleher, J. D. (2020). *Syväoppiminen* (K. Pietiläinen, käänt.). Terra Cognita. (Alkuperäisteos julkaistu 2019)

Kibriya, A. & Frank, E. (2007). *An Empirical Comparison of Exact Nearest Neighbour Algorithms*. Department of Computer Science. University of Waikato. https://doi.org/10.1007/978-3-540-74976-9_16

Liu, Z., Luo, P. & Wang, X. & Tang, Xiaoou. (2015). *Deep Learning Face Attributes in the Wild*. Multimedia Laboratory. https://doi.org/10.48550/arXiv.1411.7766

MediaPipe (2022). *MediaPipe Face Mesh*. Haettu 28.11.2022 osoitteesta https://google.github.io/mediapipe/solutions/face_mesh

O'Shea, K., Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. https://doi.org/10.48550/arXiv.1511.08458

Rossi, T (2008). *Konenäkö ja kuva-analyysi*, Jyväskylän yliopisto. Haettu 26.11.2022 osoitteesta http://users.jyu.fi/~tro/TIES411_10/mat_esitieto.pdf

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. & Chen, L. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks* Google Inc. https://doi.org/10.48550/arXiv.1801.04381

Schroff, F., Kalenichenko, D. & Philbin, J. (2015) *FaceNet: A Unified Embedding for Face Recognition and Clustering.* https://doi.org/10.48550/arXiv.1503.03832

Shorten, C. & Khoshgoftaar, T. (2019). *A survey on Image Data Augmentation for Deep Learning*. Journal of Big Data 60/2019. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0

Liite 1: Linkki kirjoitettuun koodiin

https://github.com/holle1234/MobileFaceNet