Käsieleiden tunnistaminen konenäön avulla

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

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

Kevät, 2020

Taneli Ranne

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintä tekniikka, insinööri (AMK)
Riihimäen kampus

Tekijä Taneli Ranne **Vuosi** 2020

Työn nimi Käsieleiden tunnistaminen konenäön avulla

Työn ohjaaja/t Petri Kuittinen, HAMK

TIIVISTELMÄ

Työn tarkoituksena oli tehdä konenäkösovellus, joka tunnistaa useita käsieleitä käyttäen webkameraa. Lisätoimintona oli sormenpään löytäminen osoittavista eleistä. Työtä voisi käyttää ohjaamaan erilaisia käsieleisiin pohjautuvia sovelluksia, pelejä tai kodin älylaitteita. Työssä tunnistetiin neljä eri elettä suljettu nyrkki, avoin käsi sormet levällään, yhdellä sormella osoittava käsi ja osoittava käsiele, jossa on myös peukalo avattuna.

Työn ensimmäinen vaihe oli kuvien valmisteleminen tunnistus mallin koulutusta varten. Tämä sisälsi noin 850 kuvan ottamista 11 eri ihmisen käsistä. Työn toinen vaihe oli koneoppimisen käyttäminen mallin kouluttamiseen. Tätä mallia käytettiin koulutuksen jälkeen tunnistamaan käsieleitä konenäkösovelluksessa. Työn kolmas vaihe oli kirjoittaa C++ kielellä konenäkökirjastoa hyödyntävä ohjelmistokoodi, joka käyttää koulutettua mallia.

Tunnistimen tarkkuus oli hyvä läheltä, mutta kauempaa tunnistus ei onnistunut ollenkaan. Tämä johtui kaikella todennäköisyydellä koulutuskuvista, jotka olivat kaikki otettu läheltä käsiä. Valittu malli oli myös liian raskas tietokoneelle, jota käytettiin tunnistukseen. Tunnistus nopeus oli 0,3 ruutua sekunnissa.

Avainsanat Konenäkö, koneoppiminen, OpenCV, TensorFlow

Sivut 31 sivua, joista liitteitä 3 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information Technology, engineer (AMK)
Riihimäki campus

Author Taneli Ranne **Year** 2020

Subject Hand gesture recognition using computer vision

Supervisors Petri Kuittinen, HAMK

ABSTRACT

The goal of this thesis project was to create machine vision application that would recognize hand gestures with a webcam. An extra functionality there was finding the fingertip of the pointing finger from pointing gestures. The project could be used to control different hand gesture-based applications, games, or smart home devices. Gestures recognized in the project were to include: a be closed fist, an open hand with fingers spread, a hand pointing with one finger and a hand pointing with one finger and the thumb opened making a gesture that looked like character L in the ABC.

The first phase in the project was to take and prepare pictures for the training of the model. Approximately 850 pictures were taken from the hands of eleven people. The second phase in the project was training the model. This model was used to detect and recognize hand gestures in the application later. The third and last phase was writing of the C++ application that used a computer vision library and the trained model.

The accuracy of the detector was good when the target was near the webcam, but the detector did not detect anything if the target was too far away. The training images were most likely the reason for this, as most of the training pictures were taken from a proximity of the hands. The selected model was also too heavy for the computer used to run the detection. The detection speed was just 0,3 frames per second.

Keywords Computer vision, machine learning, machine vision, OpenCV, TensorFlow.

Pages 31 pages including appendices 3 pages

SISÄLLYS

1 JOHDANTO

Konenäköä käyttävät sovellukset ja laitteet mahdollistavat teollisuudessa ja arkikäytössä erilaisten toimenpiteiden tarkan suorittamisen nopeasti ja tehokkaasti vapauttaen aikaa käyttäjille toisiin tehtäviin. Tulevaisuudessa konenäkö pystyy vapauttamaan käyttäjän kokonaan tehtävästä, esimerkiksi itse ajava henkilöauto vapauttaisi kuljettajan tekemään muita asioita ajomatkan ajaksi.

Opinnäytetyön tarkoitus oli tehdä konenäkösovellus, joka käyttää konenäköä ja koneoppimista tunnistamaan käsieleitä ja sormenpään paikan videolta, jos kädellä tehdään selvä osoittava käsiele. Tunnistettavia käsieleitä oli neljä, jotka ovat avoinkäsi sormet levällään, suljettu nyrkki, yhdellä sormella osoittava käsi ja osoittava käsi, jossa on peukalo avattuna tehden L-merkin. Tunnistaminen tulee tehdä kämmenselkä puolelta. Kuvassa yksi tunnistettavat eleet.

Kuva 1. Avoinkäsi, nyrkki, osoitus ja L-kirjaimen näköinen osoitusele.

Tunnistuksen tulisi tunnistaa käsieleitä käytännöllisellä nopeudella ja tarkkuudella mahdollistaen sovelluksen käyttämistä jatkokehitykseen. Mallin testaamiseen käytetään Python-kieltä käyttävää ohjelmistokoodia mutta lopullinen työ tehdään C++-kielellä.

2 KONENÄKÖ

2.1 Konenäön ja näköaistin ero

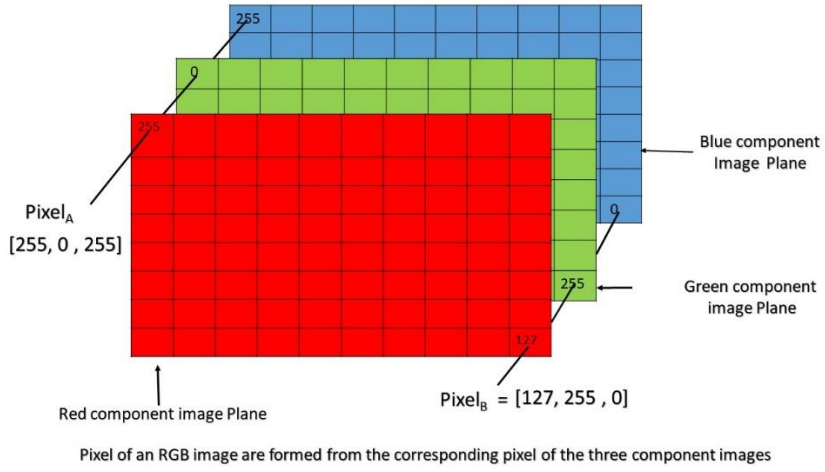
Konenäkö on datan muuttamista kuvasta tai videosta, joko päätökseen tai uuteen esitysmuotoon, jolla halutaan saavuttaa jokin tavoite. Syöttödata voi olla valvontakameran videota tai puhelimen kameran kuva. Päätös voi olla ”kuvassa on henkilö” tai ”kuvassa on neljä pyörää” ja uusi esitysmuoto voi olla kuvan muuttaminen harmaaväriskaalaan. (Bradski & Kaehler, 2008, s. 2)

Ihminen on aisteistaan ehdottomasti riippuvaisin näköaistista, joka tuottaa eniten dataa kaikista aisteista. Silmien kautta jokainen vilkaisu tuottaa megabittejä dataa ja ”jatkuvan” katsomisen data määrä on mahdollisesti jopa yli kymmenen megabittiä sekuntissa (mbit/s). Toisaalta iso osa tästä datasta on tarpeetonta ja sen kompressoi eri tasot näköaivokuoressa, jotta korkeammat aivokeskukset joutuvat käsittelemään vain pienen osan datasta. (Davies, 2012, s. 1)

Yksi ihmisen näkökyvyn keskeinen ominaisuus on tulkinnan helppous. Näemme näkymän sellaisena kuin se on puut ja joet maisemasta, esineitä pöydiltä ja ihmiset kuvasta. Tämä ei vaadi yrittämistä tai ylimääräistä miettimistä ja vastaukset ovat lähes välittömät, normaalisti sekunnin kymmenyksessä. (Davies, 2012, s. 1)

Ihminen pystyy katsoessaan kuvaa laskemaan kuvasta henkilöt, nimeämään heidät ja jopa arvaamaan henkilöiden tunteet kasvojen perusteella. Siinä missä ihmiset näkevät asiat suoraan koneet ”näkevät” vain numeroita matriisissa, joilla tulee selvittää mitä kuvassa on ja kuinka paljon käyttäen algoritmeja. (Szeliski, 2010, s. 1)

Harmaaväriskaala kuvassa matriisi sisältää arvon kuvan jokaisesta pikselistä nollasta 255:een, jossa nolla on musta ja 255 on valkoinen. Värikuvassa matriisi sisältää kolme arvoa jokaista pikseliä kohden, jotka kuvaavat punaisen, vihreän ja sinisen tasoja esim. [255,0,0] on punainen pikseli ja [0,0,255] on sininen pikseli. Kuvassa kaksi näkyy RGB-kuvan matriisin koostuminen kolmesta yksivärisestä matriisista.



Kuva 2. RGB-kuvan rakenne (Ihritik, n.d).

2.2 **Konenäön käyttökohteita**

Konenäköä käytetään tänä päivänä laajasti useilla eri teollisuuden aloilla ja päivittäiskäytössä helpottamaan tai vähentämään ihmisten tarvetta erilaisissa tehtävissä. Konenäkö pystyy joissain tehtävissä vapauttamaan työvoiman kokonaan toisiin tehtäviin.

Szeliskin (2010, s. 5) esimerkkejä konenäön käyttökohteista on listattu alle:

- Konenäöllä tarkistus: nopea palojen tarkastus laadunvalvonnassa käyttäen stereonäköä erikoisvalaistuksen kanssa esim. autojen runkojen tai lentokoneiden siipien virheiden huomaamiseen.
- 3D-mallin rakentaminen: täysin automatisoitu 3D-mallin tekeminen käyttäen useita valokuvia samasta kohteesta monesta eri suunnasta.
- Autojen turvallisuus: yllättävien esteiden havaitseminen liikenteessä kuten jalankulkijat ja kaistaviivojen seuraaminen.
- Päivittäistavarakaupoissa: tuotteiden tunnistus itsepalvelukassoilla.
- Optinen kirjainten tunnistus: käsin kirjoitettujen postinumeroiden tunnistus ja auton rekisterinumeroiden tunnistus.
- Valvonta: tunkeutujien havaitseminen, teiden ruuhkien analysointiin ja hukkuneiden uhrien havaitsemiseen altaista.
- Sormenjälkien ja biometrinen tunnistus: automaattiseen käyttöoikeuksien tarkistamiseen ja rikostekniseen tutkimiseen.

Konenäöllä tarkastamista käytetään teollisuudessa virheellisten tuotteiden havaitsemiseen eli laadunvalvontaan koska sen avulla päästään suurempaan valvonta nopeuteen ja tarkkuuteen kuin pelkän ihmisen valvonnassa. Konenäöllä voidaan myös ohjata laitteiden toimintaa ja estää tapaturmia tapahtumasta. Auto- ja lentokonetehtaat käyttävät konenäköä havaitsemaan virheitä autojen rungoista ja lentokoneiden siivistä stereonäöllä erikoisvalaistuksessa.

Autoissa konenäköä pystytään käyttämään jalankulkijan tai eläimen tunnistamiseen hätäjarrutusta varten. Konenäköä käytetään myös kaistaviivojen ja ympäristön tunnistamiseen, jotta auto voidaan pitää automaattisesti omalla kaistallaan.

Kamerat ja valokuvaussovellukset käyttävät konenäköä kasvojen automaattiseen kohdistamiseen. Monissa kuvaus- ja viestintäsovelluksissa on useita suodattimia, jotka käyttävät konenäköä ja koneoppimista kohteensa muokkaamiseen useilla eri tavoilla lopulliseen otokseen.

Optista kirjainten tunnistusta käytetään useilla eri aloilla tunnistamaan nopeasti numeroita erilaisissa tilanteissa. Posteissa käytetään automaattista osoitteen lukijaa pakettien ja kirjeiden lajitteluun. Monet parkkihallit seuraavat autojen parkkimaksujen maksamista rekisterikilpien avulla ja päästävät auton ilman lipukkeen palautusta parkkihallista pois maksun suorittamisen jälkeen.

Turvallisuusjärjestelmät älypuhelimissa ja vartioiduissa kohteissa käyttävät konenäköä tunnistamaan käyttäjän oikeudet sormenjäljistä, kasvoista ja jopa silmän iiriksestä. Teknologian kehittyminen on helpottanut ja nopeuttanut esimerkiksi älypuhelimen lukituksen avaamista turvallisuudesta tinkimättä.

Peleissä konenäköä voidaan käyttää mahdollistamaan käden tai jopa koko kehon käyttämisen ohjaimena pelatessa ilman lisäohjainta. Lisätyn todellisuuden peleissä pystytään asettamaan kohteita realistisiin paikkoihin oikeassa maailmassa.

2.3 **Konenäkö järjestelmän osat**

2.3.1 Kamera

Konenäkö tarvitsee kameran saadakseen digitaalisia kuvia tai videoita käsiteltäväksi. Kamera voi olla minkälainen vain, kunhan se sopii haluttuun tehtävään ja se tuottaa digitaalisia kuvia. Esimerkiksi röntgenkamera, jos halutaan etsiä konenäöllä murtumia luista tai tavallinen valvontakamera, jos lasketaan sen ohi ajavia autoja. Kuvassa kolme näkyy katua kuvaava valvontakamera tolpassa.

Kuva 3. Kadun valvontakamera (Pxhere, n.d)

Teknologian kehityksen takia lähes jokaisella ihmisellä on taskussaan jatkuvasti kamera älypuhelimen muodossa, jota voidaan käyttää konenäköä varten. Joidenkin älypuhelimien kamerat ovat jo verrattavissa järjestelmäkameroihin. Isoin ero näiden välillä on järjestelmäkameroiden mahdollisuus optiikan vaihtoon. Kuvassa neljä Olympus-järjestelmäkamera ja kaksi älypuhelinta.

Kuva 4. Kuvassa HTC 8X, Nokia Lumia 920 and Olympus OM-D (Ponder, 2012)

2.3.2 Valaistus

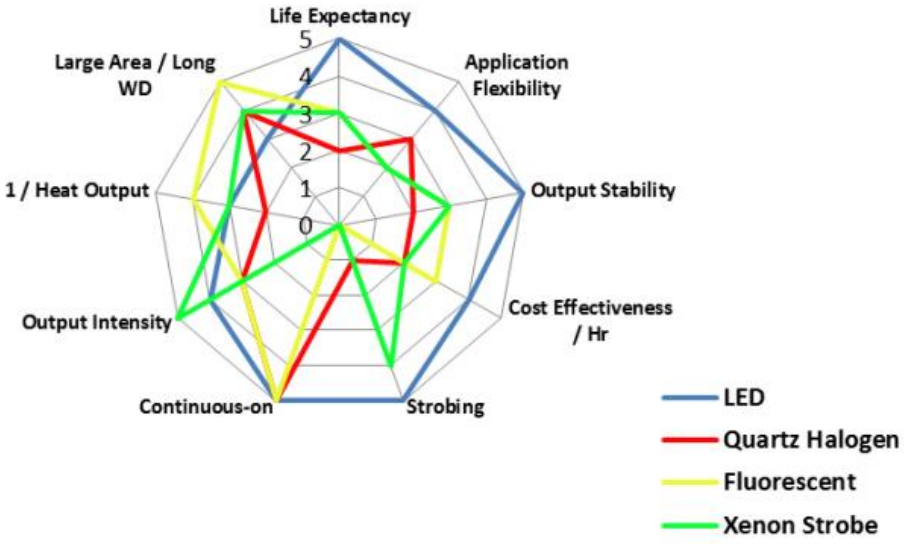
Hyvä valaistus on äärimmäisen tärkeä konenäöllä onnistumista varten. Hyvä valaistus ei tarkoita vain kirkasta valoa koska liian kirkas valo ylivalottaa kohteen ja voi tehdä sen tunnistamisen äärimmäisen vaikeaksi tai lähes mahdottomaksi. Kuvassa viisi on kaksi kuvaa joista toinen on hyvin valaistu ja toinen on ylivalottunut.

Kuva 5. Kuvassa kelmulla päällystetty paketti nuottikortteja. Vasen kuva kelpaa konenäköön. Oikea kuva on ylivalottunut ja ei kelpaa konenäköön. (Martin, 2013, s. 3)

Yleisimmät valon lähteet, jota käytetään konenäössä ja niiden Kelvin arvoja:

- Fluoresenssi – 4500K
- Quartz halogeeni lamppu – 3000K (Lumens, 2020)
- LED – 4000-7000K (Lumens, 2020)
- Monimetallilamppu (elohopea) 4000K (Lumens, 2020)
- Ksenonlamppu – välkkyvä valo(strobovalo) – 5600K
- Luonnonvalo – 10000K+ (Lumens, 2020)

Fluoresenssi, halogeeni ja LED-valot ovat reilusti eniten käytetyimmät valolähteet konenäköä varten, ennen kaikkea pienissä ja keskikokoisissa tarkastuspisteissä. Monimetallilamput ja ksenonlamput ovat tyypillisempiä ison mittakaavan käyttötarkoituksissa tai käyttökohteissa, jossa tarvitaan todella kirkasta valo lähdettä. Monimetallilamppuja käytetään usein mikroskopiassa koska sillä on monta erillistä aallonpituus huippua, jotka täydentävät suodattimien käyttöä fluoresenssi tutkimuksissa. Ksenon valolähde on hyödyllinen käyttötarkoituksissa, jossa tarvitaan erittäin kirkasta ja välkkyvää valoa. Kuvassa kuusi näkyy neljän eri valolähteen eroja toisiinsa. (Martin, 2013, s. 3-4)



Kuva 6. Yleisten valonlähteiden vertailu ja samankaltaisuus. (Martin, 2013, s. 4)

Historiallisesti fluoresenssi ja halogeeni valolähteet ovat olleet käytetyimmät konenäössä, mutta lähivuosina LED-teknologia on kehittynyt vakaudessa, intensiivisyydessä ja kustannustehokkuudessa. Riippuen valaistus tarpeista on usein kannattavampaa käyttää useampaa kuin yhtä valonlähde tyyppiä, koska useat konenäön asiantuntijat ovat yhtä mieltä siitä, että yksi valolähde tyyppi ei pysty toimivasti ratkaista kaikkia valaistus ongelmia. (Martin, 2013, s. 4)

Luonnonvalo on erittäin hyvä pää- tai lisävalaistus sisä- tai ulkotiloissa, jos sitä on mahdollisuus hyödyntää. Päiväsaikaan se saattaa olla ulkona olevien konenäköjärjestelmien ainoa valonlähde ja se riittää siihen erinomaisesti. Sisätiloissakin luonnonvalo voi riittää ainoaksi valonlähteeksi, mutta se rajoittaa käyttöajan päiväsaikaan.

2.3.3 Ohjelmisto

Ohjelmistoa valittaessa tulisi ensimmäiseksi ottaa huomioon kuinka haastavaa konenäköä aiotaan tehdä ja mitä vaatimuksia se tuo ohjelmistolle. Jos tehdään konenäköjärjestelmä todella nopeasti toimivalle sovellukselle, joka vaatii erinomaista kuvanlaatua se todennäköisesti tuottaa paljon kuva dataa. Tämä tarkoittaa, että ohjelmistossa tulisi olla hyvä prosessointi teho ja muistikapasiteetti. (Vision Online Marketing Team, 2017)

Vaikka on tärkeää miettiä mitä sovellus vaatii konenäköohjelmistolta, tulee myös ottaa huomioon tietyt ohjelmiston ominaisuudet huomioon kuten:

- Integraatio ja yhteensopivuus.
- Käytön helppous ja kustannukset.

- Valmistajan tuki ja vakaus. (Vision Online Marketing Team, 2017)

Nykymaailman valmistus ympäristössä mikään ei ole kokonaan eristetty tämä on ennen kaikkea totta konenäköjärjestelmissä. Tarvitseeko konenäköohjelmistosi tallentaa kuvia ja dataa, keskustella kauko-ohjaus käyttöliittymän kanssa tai käyttölaitteen kanssa. Tällöin ohjelmiston kyky integroitua muiden komponenttien kanssa on kriittistä. (Vision Online Marketing Team, 2017)

Monissa tapauksissa ohjelmointikieltä ei tarvita tehtäessä toimivaa konenäköjärjestelmää, mutta toiset sovellukset tarvitsevat kehittynyttä ohjelmointia saadakseen täyden toimivuuden. Tämä lisää ylimääräisen ajan ja rahan käytön järjestelmän alkuperäisen hinnan lisäksi sen käyttämiseen ja ylläpitämiseen. (Vision Online Marketing Team, 2017)

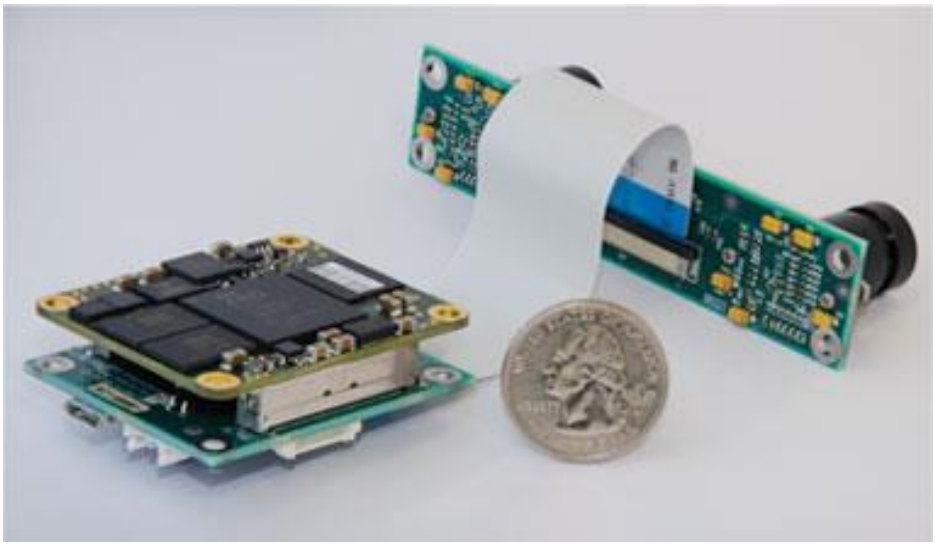
On myös tärkeää ottaa huomioon yhtiö, joka myy konenäköohjelmistoa. Vaikka ohjelmisto olisi helppo käyttöinen on vain ajan kysymys ennen kuin vastaan tulee ongelma, johon tarvitaan apua. Yhtiö missä on tietävä ja reagoiva tekninen tuki pystyy auttamaan välttämään kallista joutoaikaa. (Vision Online Marketing Team, 2017)

Esimerkiksi OpenCV on konenäkemisenkirjasto, joka on tehty pääasiassa reaaliaikaista konenäköä varten, mutta siitä löytyy funktiot ja algoritmit myös pelkkien yksittäisten kuvien kanssa toimimiseen. Se ei myöskään ole liian raskas, joten sen saa toimimana myös pienemmillä laitteilla kuten Raspberry PI tai älypuhelimella. OpenCV:llä on myös kattava dokumentointi ja aktiivinen foorumi, josta saa apua ongelmiin nopeasti.

Tensorflow taas on koneoppimisenohjelmisto, jolla voidaan muun muassa kouluttaa malli(model), joka tunnistaa ja paikantaa erilaisia objekteja kuvasta ja nimeää ne. Tässä tilanteessa on tärkeää valita Tensorflowilta koneoppimis malli, joka on sopiva omaan käyttöön. Mallien eroina ovat pääsääntöisesti nopeus, tarkkuus ja koko. Jos tarkkuus on tärkein eikä nopeudella tai koolla ole väliä kannattaa valita tarkin malli, mutta jos käyttää pientä ei niin tehokasta laitetta ja haluaa painottaa nopeuteen, täytyy miettiä enemmän minkä mallin valitsee käyttöönsä.

2.3.4 Laitteisto

Konenäkösovellukset vaativat haastavuutensa mukaan enemmän laskenta tehoa ja muistia. Yksinkertaiset tai todella spesifioidut konenäkösovellukset eivät tarvitse suuria pöytäkoneita ja kalliita kameroita toimiakseen nopeasti vaan voivat toimia pienellä kameralla varustelulla prosessointi levyllä, jota kutsutaan sulautetuksi konenäköjärjestelmäksi. Kuvassa seitsemän on SRI Internatiolin sulautettu konenäköjärjestelmä. (Vision Online Marketing Team, 2017)



Kuva 7. Älypuhelimen prosessorista tehty sulautettu konenäköjärjestelmä. (SRI International, 2019)

Isoin ero sulautettujen konenäköjärjestelmien ja pöytäkoneisiin perustuvien järjestelmien välillä on se, että sulautetut järjestelmät ovat tehty tiettyjen käyttökohteiden mukaan, kun taas pöytäkone pohjaiset järjestelmät ovat yleensä tehty yleistä kuvankäsittelyä varten. Tämä kasvattaa ensimmäisen integraation vaikeutta, mutta se kompensoituu tehtävään tehdyn sulautetun järjestelmän sopimisen täydellisesti tarkoitukseensa. (Vision Online Marketing Team, 2018)

Sulautettuja konenäköjärjestelmiä käytetään erilaisissa kohteissa kuten itse ajavissa autoissa, automatisoiduissa ajoneuvoissa maanviljelyssä, digitaalisessa dermatoskoopissa ja muissa ainutlaatuisissa viimeisintä teknologiaa käyttävissä sovelluksissa. Jokaisessa näistä käyttökohteista on käytetty juuri sen sovelluksen tehtävälle tehtyä sulautettua järjestelmää. Esimerkiksi sulautettua järjestelmää, joka on tehty dermatoskoopille ei voida käyttää itse ajavassa autossa. (Vision Online Marketing Team, 2018)

3 KONEOPPIMINEN

3.1 Koneoppimisen käyttötavat

Konenäkemisen algoritmit ovat kehittyneet nopeasti lähivuosina. Etenkin konenäkemisen yhdistäminen koneoppimisen kanssa on vaikuttanut notkeiden ja kestävien konenäkemisen algoritmien kehityksessä ja näin edistänyt käytännönläheisten näköjärjestelmien suorituskykyä. (Khan, Rahmani, Shah, Bennamoun, 2018, s. 5)

Koneoppiminen on keinoälyä, joka mahdollistaa tietokoneille datasta oppimisen ilman sen täsmällistä ohjelmointia. Toisin sanoen koneoppimisen tarkoitus on suunnitella menetelmä, joka automaattisesti oppii sille annetusta koulutusdatasta ilman ihmisen tarkkoja määrityksiä säännöistä ja logiikasta. Koneoppimisen voi siis ajatella data näytteillä ohjelmoimisena eli opetella tekemään paremmin tulevaisuudessa käyttäen menneisyyden kokemuksia. (Khan ym., 2018, s. 5)

Nämä koneoppimisen algoritmit voidaan jakaa kolmeen pää lähestymistapaan, jotka ovat valvottu(suprevised), osaksi valvottu(semi-supervised) ja valvomaton(unsuprevised). Suurin osa käytännöllisistä koneoppimisen tavoista ovat valvottuja oppimisen tapoja, koska niiden suorituskyky on suurempi verrattuna muihin oppimistapoihin. (Khan ym., 2018, s. 5)

Valvotuissa oppimistavoissa koulutusdata on muodossa, jossa data ja etiketti(label) muodostavat parin, jonka avulla tehdään ennusteita datasta. Data voi olla vektoreita tai muita monimutkaisia datatyyppejä kuten kuvia, dokumentteja tai graafeja. Tuloste voi olla binaarinen tulos kuten kyllä tai ei. Muita tulosteita voivat olla moniluokka luokittelu, jossa tuloste on etiketti, joka on yksi useista määritellyistä luokista tai useita etikettejä monista eri luokista. (Khan ym., 2018, s. 5)

Valvomattomissa oppimistavoissa on vain syöttödata eikä määriteltyä ulostulo data muuttujaa. Sitä kutsutaan valvomattomaksi oppimiseksi koska siinä ei ole opettajaa eikä selviä ulostuloja. Sen tavoite on mallintaa pinnan alla oleva data runko, jotta löydetään mielenkiintoinen rakenne datasta. (Khan ym., 2018, s. 5)

Osaksi valvotut opettamistavat ovat valvotun ja valvomattoman välissä. Tätä opettamistapaa käytetään, kun on paljon syöttödataa ja vain osa on tunnistettu etiketeillä. Esimerkiksi valokuva arkisto, jossa vain osa kuvista on nimetty/tunnistettu ja suurin osa on merkitsemättömiä. (Khan ym., 2018, s. 6)

3.2 **Syväoppiminen**

Nämä koneoppimisen algoritmit ovat olleet pitkään käytössä, mutta kyky automaattisesti käyttää monimutkaisia matemaattisia laskelmointeja dataan laajassa mittakaavassa on kehittynyt viime aikoina. Tähän kehitykseen on vaikuttanut nykyajan tietokoneiden tehokkuuden paraneminen muistin ja nopeuden kannalta, mikä on auttanut koneoppimisen tekniikoiden kehittymistä mahdollistamalla isojen tietorunkojen käyttämisen koulutusdatana. (Khan ym., 2018, s. 6)

Laskentatehon ja muistin määrän kasvu mahdollistaa neuroverkon tekemisen useilla kerroksilla, jota kutsutaan siinä vaiheessa laajaksi neuroverkoksi. Laajalla neuroverkolla on kolme avain etua, jotka saavutetaan syvä

oppimisella, joita ovat yksinkertaisuus, skaalattavuus ja toimialueen vaihto. (Khan ym., 2018, s. 6)

Yksinkertaisuudella tarkoitetaan, ettei ongelmaan tehdä yksilöityjä muutoksia tai räätälöityjä piirteiden tunnistimia vaan käytetään laajan neuroverkon osia eli verkon kerroksia. Näitä verkon kerroksia toistamalla saadaan tehtyä isoja verkostoja. (Khan ym., 2018, s. 6)

Skaalattavuus tarkoittaa syväoppimisen tuomaa skaalattavuuden helppoutta isoihin tietokokonaisuuksiin. Toisissa kilpailevissa metodeissa kuten käyttöjärjestelmäydinkoneissa tulee isoja laskennallisia ongelmia, jos tietokokonaisuus on liian iso. (Khan ym., 2018, s. 6)

Toimialueen vaihto tarkoittaa tiettyyn tehtävään koulutetun mallin käyttöä toiseen samankaltaiseen tehtävään. Tämän mahdollistaa mallin oppimat piirteet, jotka ovat tarpeeksi yleisiä ja joita voidaan käyttää toiseen tehtävään, josta on vain vähän dataa käytettävissä. (Khan ym., 2018, s. 6)

Syväoppimisen tekniikoita, joita käytetään havaitsemiseen, segmentointiin, luokitteluun ja kappaleiden tunnistukseen kuvista ovat viimeisintä teknologiaa. Näiden teknologioiden onnistumisen myötä niitä jatko kehitetään haastavampiin tehtäviin kuten lääketieteellisiin diagnosointeihin ja automaattiseen kielten kääntämiseen. (Khan ym., 2018, s. 6)

Koneoppiminen ei silti tarkoita, että se ei tekisi virheitä tai ettei sitä pystytäisi huijaamaan. Onnistuneen tunnistuksen voi saada antamaan väärän tunnistuksen muuttamalla kuvan värisävyä tai saturaatioita vain vähän, mikä voi riittää muuttamaan hevosen koiraksi konenäköjärjestelmälle. Lisäämällä pari tarraa stop merkkiin voidaan saada auton konenäköjärjestelmä uskomaan, että merkki onkin nopeusrajoitus. (Heaven, 2019)

Eri hyökkäystekniikoita konenäköä vastaan on useita. Google on kehittänyt vihamielisen tarran joka kuvaan asetettuna saa tunnistimen tunnistamaan kuvasta aina leivänpaahtimen riippumatta mitä kuvassa on. Musta laatikko hyökkäyksessä taas muutetaan mahdollisimman vähän syöttödataa, jotta voidaan muokata saadusta tunnistuksesta haluttu tunnistuksen tieto. Suurimmassa osassa hyökkäystekniikoista täytyy tietää neuroverkon rakenne mutta se ei ole välttämätöntä. Tutkijat ovat todistaneet, että on mahdollista tehdä oletus verkon rakenteesta muokkaamalla toista verkkoa, kunnes se käyttäytyy samankaltaisesti ja käyttää tätä rakennetta hyökkäystä varten. (Cronje, 2018)

4 SOVELLUKSET JA KIRJASTOT KONENÄKÖÖN JA KONEOPPIMISEEN

4.1 **TensorFlow**

Tensorflow on Googlen Brain Teamin tekemä avoimen lähdekoodin koneoppimisen alusta, joka on tehty koneoppimisen ja syväoppimisen tutkimista varten. Siinä on kattava ja joustava ekosysteemi työkaluja, kirjastoja ja yhteisö resursseja, joilla Tensorflowin käyttäjät pystyvät kehittämään ja ottamaan käyttöön helposti koneoppimista käyttäviä applikaatioita. (Khan ym., 2018, s. 160)

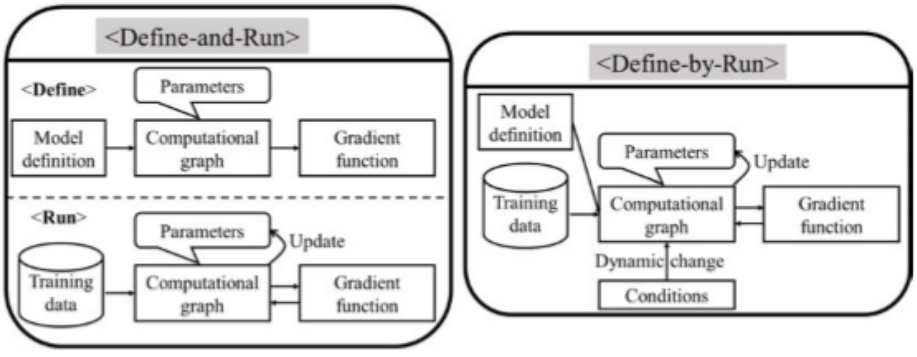
Tensorflow on kirjoitettu Python rajapinnalla C/C++-moottorin päälle numeerista laskentaa varten tietovirta graafeista. Tensorflowilla on monta eri tasoista rajapintaa, jotka on tehty Tensorflow Coren päälle. Näitä rajapintoja on helpompi oppia käyttämään kuin pohjalla toimivaa Tensorflow Corea. Koneoppimista tutkiville ja muille, jotka tarvitsevat täsmällistä hallintaa malleihinsa suositellaan käyttämään suoraan Tensorflow Core rajapintaa. (Khan ym., 2018, s. 160)

Tensorflow tukee montaa eri taustaosa(backend) versiota, joita ovat prosessori (CPU) ja näytönohjain (GPU) versio pöytäkoneelle, serveri versio ja versio mobiili alustoille. Pöytäkoneille saatavien versioiden ero tulee siitä mitä laitteistoa Tensorflow käyttää pääsääntöisesti laskentatehtäviin. Tuettuja käyttöjärjestelmiä ovat Linux, Mac ja Windows. Tensorflowia pystytään käyttämään Pythonilla, C/C++, Java ja Go. (Khan ym., 2018, s. 160)

4.2 **PyTorch**

PyTorch on avoimen lähdekoodin koneoppimisen kirjasto Pythonille. Sen on tehnyt Facebookin koneälyn tutkimusyksikkö. PyTorch on kasvanut nopeasti koneoppimisen tutkijoiden suosikiksi, koska se mahdollistaa tietyntyyppisten haastavien arkkitehtuurien teon helposti. (Khan ym., 2018, s. 165)

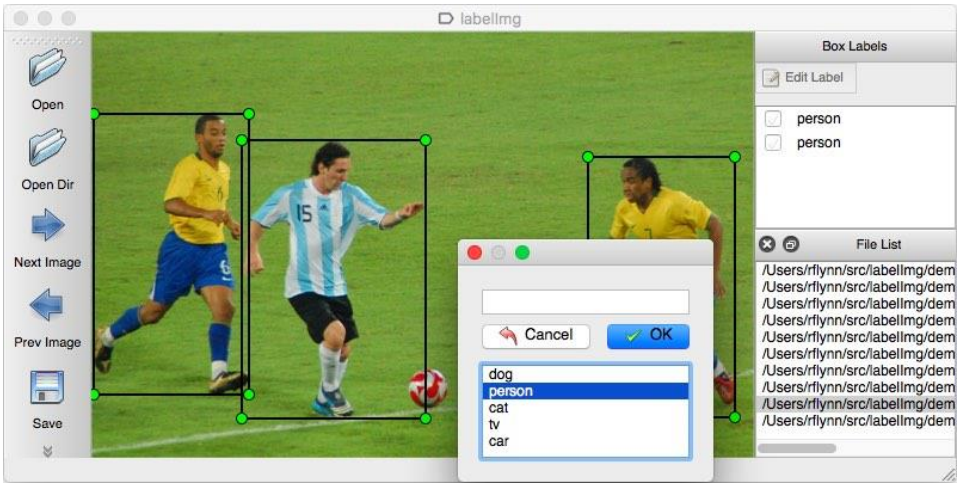
PyTorch eroaa useista muista koneoppimisen kirjastoista, koska se antaa muokata verkkoja ajonaikana. Tätä kutsutaan define-by-run viitekehykseksi. Kuvassa kahdeksan näkyy define-and-run ja define-by-run viitekehyksien eroja. Tuettuja käyttöjärjestelmiä ovat Linux, Mac ja Windows. PyTorchia käytetään vain Python-kielellä. (Khan ym., 2018, s. 165)



Kuva 8. Define-and-run ja define-by-run viitekehyksien toimintakaaviot. (Khan ym., 2018, s. 165)

4.3 Labelimg

Labelimg on ilmainen koneoppimista varten tehty ohjelma, jolla pystytään ottamaan talteen tietoja kuvasta, jota aiotaan käyttää mallin kouluttamisessa. Ohjelmalla merkitään helposti laatikoimalla kohteet kuvasta ja ohjelma ottaa koordinaatit, laatikoiden korkeudet ja leveydet talteen. Tämän lisäksi kohteille tulee antaa etiketit, jotka kertovat mitä laatikot sisältävät. Kuvassa yhdeksän Labelimg toimintaa näyttävä kuva.



Kuva 9. Jalkapalloilijoita merkitään ihmisiksi kuvaan. (Tzutalin, 2020)

Labelimg tallentaa tiedot XML-tiedostona PASCAL VOC-muodossa. Tämä XML-tiedosto tallentuu kuvan nimellä samaan kansioon kuvan kanssa. Ohjelma sopii hyvin kohteiden tietojen talteen ottamiseen kuvista nopeasti ja vaivattomasti. Se myös muistaa viimeisimmän käytetyn etiketin, jotta samojen kohteiden merkitseminen eri kuvista olisi vaivattomampaa. Labelimg toimii Ubuntu Linux, Mac ja Windowsilla. Se on kirjoitettu Pythonilla ja graafinen käyttöliittymä käyttää Qt ohjelmistokehystä, joka on C++ pohjainen.

4.4 Imagemagick

ImageMagick ei itsessään ole konenäkö- tai koneoppimisenohjelmisto vaan komentorivillä toimiva kuvienkäsittelyohjelma. Yleensä kuvien muokkaus tehdään graafisen-käyttöliittymän (GUI) ohjelmilla kuten Photoshopilla tai Gimpillä. Aina kuitenkaan GUI-ohjelmistot eivät ole käytännöllisiä. Jos halutaan muokata isoja määriä kuvia samoilla toiminnoilla tai tietyssä järjestyksessä on komentorivi pohjainen ohjelmisto parempi vaihtoehto.

ImageMagickiä käytetään yleisemmin komentoriviltä mutta siitä on ladattavissa monta ohjelmointirajapintaa (API), jotka mahdollistavat sen käyttämisen useilla käyttöliittymillä ja ohjelmointi kielillä kuten Java, C kielet, Python, Ruby ja tusinalla muilla. (ImageMagick, 2020)

Tämä ohjelmisto sopii konenäkemiseen ja koneoppimiseen liittyen erittäin hyvin isojen kuvamäärien valmistelemiseen nopeasti ja vaivattomasti. Parilla lyhyellä komentorivi käskyllä pystytään muokkaamaan kaikkien kansiossa olevien kuvien kokoa tai muokata ne värikuvista harmaaväriskaalan kuviin. Tämän jälkeen kuvia voidaan käyttää esimerkiksi opettamaan konenäköä tunnistamaan esineitä tai henkilöitä.

4.5 **OpenCV**

OpenCV (Open Source Computer Vision Library) on avoimen lähdekoodin konenäkemisen ja koneoppimisen ohjelmistokirjasto. OpenCV on BSD-lisensoitu, joka mahdollistaa sen lähes rajoittamattoman muokkaamisen ja käyttämisen kaupallisissa tuotteissa ilmaiseksi. OpenCVtä käyttää alan jättiläiset kuten Google, Yahoo, Microsoft, IBM, Sony ja lukuisat startupit. (OpenCV, 2020)

Kirjasto sisältää yli 2500 optimoitua algoritmia, joihin kuuluu kattavasti klassisia sekä viimeisintä tekniikkaa edustavia konenäön ja koneoppimisen algoritmeja. Kirjaston sisältämillä funktioilla pystytään tekemään useita eri konenäön tehtäviä nopeasti ja helposti. OpenCV on suunniteltu laskennalliseen tehokkuuteen ja reaaliaikaiseen konenäköön. Se on kirjoitettu optimoidulla C-kielellä ja pystyy hyödyntämään moniydin prosessoreita tehokkaasti. (Bradski & Kaehler, 2008, s. 1;ks. myös OpenCV, 2020)

OpenCV toimii Windowsilla, Linuxilla ja Mac 0S X käyttöjärjestelmillä. Se sisältää käyttäjäliittymän C++, Python, Java ja MATLAB kielille. OpenCV tukee kappaleiden tunnistukseen erilaisia syväoppimisen ohjelmistokehyksiä kuten Caffe, TensorFlow, Torch, Darknet ja malleja ONNX muodossa. (OpenCV, 2020)

4.6 **SimpleCV**

SimpleCV (Simple Computer Vision) on helppo käyttöinen avoimen lähdekoodin Python-runko, joka yhdistää avoimen lähdekoodin konenäkemisen kirjastoja ja algoritmeja ongelmien ratkaisemiseen. Yksi näistä käytetyistä kirjastoista on OpenCV. (Demaagd, Oliver, Oostendrop & Scott, 2012, s. 2)

SimpleCV tarkoitus on helpottaa ohjelmoijia konenäköjärjestelmien tekemisessä virtaviivaistamalla ja yksinkertaistamalla yleisimmät tehtävät. Sen avulla voidaan työskennellä kuvien ja videoiden kanssa, jotka voivat tulla useista eri lähteistä kuten web-kamera, Kinect, FireWire, IP-kamerat tai puhelimesta. SimpleCV tehtiin toimimaan Pythonilla kielen oppimisen

helppouden takia. Se toimii Macilla, Windowsilla ja Ubuntu linuxilla. (Demaagd ym., 2012, s. 2)

4.7 **CUDA**

CUDA (Compute Unified Device Architecture) on rinnakkaislaskenta alusta ja ohjelmointimalli, jonka on tehnyt NVIDIA. CUDA avulla voidaan käyttää sitä tukevaa näytönohjainta (GPU) dramaattisesti nopeuttamaan laskenta tehtäviä. Näytönohjaimella tehostetuissa applikaatioissa GPU ytimet työskentelevät yhtä aikaa ratkaistakseen vaikeita laskenta tehtäviä jopa sata kertaa nopeammin kuin perinteisillä tekniikoilla. CUDA tukee ohjelmistokieliä kuten C, C++, Fortran, Python ja MATLAB, joissa kaikissa on yhtäläisyyksiä tietyissä avain komennoissa. (Dove, 2006)

Konenäkemisessä ja oppimisessa CUDA nopeuttaa kappaleiden tunnistus mallien (model) opettamisen moninkertaisesti. CUDA antaman näytönohjaimen laskenta tehon avulla parissa tunnissa saadaan koulutettua tunnistusmalli samalle tasolle kuin pelkän prosessorin avulla koulutettu malli, jota on opetettu monia kymmeniä tunteja.

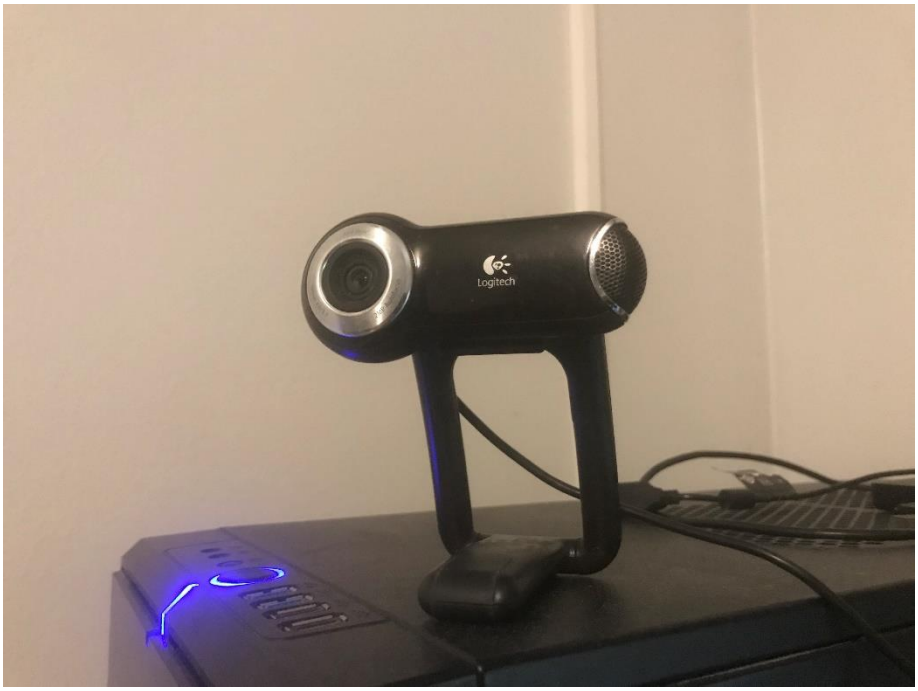
CUDA tukee ainoastaan NVIDIA omia näytönohjaimia, joten tämä tulisi ottaa huomioon, jos aikoo rakentaa tietokoneen konenäkemisen tai koneoppimisen tutkimiseen tai käyttämiseen. CUDA tukee Windows, Mac ja Linux käyttöjärjestelmiä. Linuxilla CUDA Toolkitin saa Fedora, Redhat, SUSE, OPENSUSE ja Ubuntu.

5 **KONENÄKÖ SOVELLUKSEN TEKEMINEN**

5.1 **Ohjelmiston ja laitteiston valinta**

Sovelluksen tekoon käytetään OpenCV-kirjastoa, koska yritys, jossa työskentelen käyttää C++ kieltä projekteissaan ja OpenCV on kirjoitettu C++-kiellä. Päätökseen vaikutti myös OpenCV laaja dokumentointi ja sen käyttäjien suuri määrä. Käden eleiden tunnistamisen käytetään Tensorflow konenoppimisenympäristöä sen dokumentoinnin määrän takia ja CUDA tuen takia. Kuvien valmisteluun koulutusta varten käytetään Imagemagic ja Labelimg-ohjelmia.

Konenäkölaitteistona toimii pöytäkone, jossa on Windows 10 käyttöjärjestelmä ja kuvassa kymmenen näkyvä Logitech carl zeiss tessar 2.0/3.7 webkamera. Tensorflow mallin kouluttamista varten on toinen pöytäkone, jossa on CUDA tukeva NVIDIA-näytönohjain.



Kuva 10. Logitech carl zeiss tessar 2.0/3.7 webkamera

5.2 **OpenCV ja Tensorflow asentaminen**

OpenCV C++-kirjastoa ei saa suorana latauksena vaan se tulee kasata itse. Tämä mahdollistaa siihen kuuluvien kirjaston osien määrän muokkaamisen, jotta käyttäjä pystyy vaikuttamaan tiedoston kokoon ja jättämään turhat kirjaston osat pois. Kirjaston kasaaminen tapahtui ilmaisella CMake-ohjelmalla, jota käytetään paljon vapaan lähdekoodin C++ maailmassa pakettien kasaamiseen. Kun kirjaston osat on valittu ja pakettia aletaan luomaan, valittiin mille Visual Studion-versiolle paketti tehtiin.

Paketin kasaamisen jälkeen Visual Studiolla tuli avata kasatusta paketista OpenCV.sln. Tämän jälkeen asetetiin INSTALL-projekti aktiiviseksi Visual Studiossa ja ajettiin se Debug ja Release-versioina. Tämä asensi paketin ohjelmointialustalle, jonka jälkeen OpenCV oli lähes valmis käyttöä varten. Viimeiseksi lisättiin Windowsin ympäristömuuttujiin reitti OpenCV dynaamisiin linkkikirjastoihin(dll) jotta kyseisen reitin voi lisätä projekteihin nopeasti. Vaihtoehtoisesti dynaamiset linkkikirjastot voi kopioida suoraan Visual Studio-projekteihin, joissa halutaan käyttää OpenCV-kirjastoa.

TensorFlow asentamista ja mallin koulutusta varten seurattiin EdjeElectronicsin Tensorflow mallin koulutus ohjeita. TensorFlow mallin kouluttamista varten tuli ladata ja asentaa Anaconda, CUDA ja cuDNN(CUDA Deep Neural Network). Anacondaa käytettiin sen virtuaaliympäristön takia ja sen voi korvata toisella virtuaaliympäristöllä kuten Dockerilla. Normaalisti CUDA ja cuDNN asentaessa tulisi valita toisiaan tukevat versiot ja siinä täytyy olla tarkkana, koska versioita on paljon ja asennus ei onnistu, jos valitaan sopimattomat versiot. Anaconda helpotti tässä vaiheessa, sillä

kuvassa 11 näkyvällä komennolla voidaan luoda virtuaaliympäristö ja asentaa TensorFlow-näytöohjain version siihen, mikä sisältää myös oikeat CUDA ja cuDNN-versiot.

```
conda create --name tensorflow1 tensorflow-gpu
```

Kuva 11. Komennolla luodaan tensorflow1 niminen virtuaaliympäristö, johon asennetaan TensorFlow-GPU versio ja siihen tarvittavat paketit.

Anacodan virtuaaliympäristöön asennettiin useita paketteja, joita tarvitaan TensorFlow ohjelmistokoodeissa ja mallin testaus ohjelmistokoodeissa. Kuvassa 12 näkyvät komennot, joilla asennettiin tarvittavat paketit.

```
(tensorflow1) C:\>conda install -c anaconda protobuf
(tensorflow1) C:\>pip install pillow
(tensorflow1) C:\>pip install lxml
(tensorflow1) C:\>pip install Cython
(tensorflow1) C:\>pip install contextlib2
(tensorflow1) C:\>pip install matplotlib
(tensorflow1) C:\>pip install pandas
(tensorflow1) C:\>pip install opencv-python
```

Kuva 12. Komennot pakettien asentamiseen Anaconda virtuaaliympäristöön.

Lisäksi tulee ladata TensorFlow Object Detection-ohjelmointirajapinta säiliö GitHubista. Tämä hakemisto sisältää kaiken tarvittavan kappaleen tunnistusmallin kouluttamiseen paitsi itse mallin ja koulutuskuvat. Viimeiseksi ladattiin konenäköä käyttävään laitteeseen sopiva malli TensorFlow detection model zoo:sta. Esimerkiksi mobiililaiteelle valittaisiin jokin ssd_mobilenet-versioista ja pöytäkoneelle faster_rcnn-versioista. Tässä projektissa käytetiin faster_rcnn_inception_v2_pets mallia koska konenäköä käytettiin pöytäkoneella ja tarkka käsieleiden tunnistaminen oli tärkeämpää lopputuloksessa kuin nopeus. Ladattu malli purettiin object_detection-kansioon.

5.3 **Kuvien valmisteleminen mallin koulutusta varten**

Käsieleiden tunnistamiseen käytettävän mallin kouluttamista varten valmisteltiin ensimmäiseksi kuvat, joita käytettiin itse koulutukseen. Kuvien määrä eri käsieleistä vaikuttaa suuresti mallin lopulliseen tarkkuuteen ja mallin luotettavuuteen.

Kuvissa kohteiden tulisi olla erilaisilla taustoilla ja vaihtelevissa valaistuksissa jotta koulutuskuvat olisivat mahdollisimman monipuoliset. Kuvissa tulisi olla myös paljon erilaisia taustakappaleita. Nämä taustakappaleet voivat peittää osan kohteesta, mutta suurimmassa osassa kuvista tulisi kohteen tai kohteiden olla kokonaan näkyvillä. Mallin saa tunnistamaan kohteita alle sadalla kuvalla jokaista tunnistettavaa kohdetta varten, mutta parhainta lopputulosta ja luotettavinta mallia varten tulisi valmistella useita satoja, ellei jopa tuhansia kuvia jokaista kohdetta varten.

Kohteista voi itse ottaa kuvia puhelimella tai järjestelmäkameralla. Vaihtoehtoisesti kuvat voi ladata esimerkiksi Googlen kuvahaun avulla. Kuvat eivät saa olla liian isoja, koska kuvien koko vaikuttaa suuresti koulutuksen kestoon. EdjeElectronicsin (2019) mukaan yksittäisen kuvan tulisi olla alle 200KB kooltaan ja resoluutio ei saisi ylittää 720x1280 pikseliä. Tässä vaiheessa Imagemagic tulee erittäin hyödylliseksi. Kuvassa 13 näkyvä komento pyrkii muokkaamaan kaikki valitun kansion kuvat 700 kertaa 700 pikselin kokoiseksi säilyttäen kuvan mittasuhteet.

```
C:\tensorflow2\models\research\object_detection\images\train> mogrify -resize 700x700 *.jpg
```

Kuva 13. Imagemagic komento Windowsin komentokehotteessa

Käsikuvia otettiin 11 eri ihmiseltä kummastakin kädestä noin kahdeksasta kymmeneen jokaista elettä kohden. Näistä kuvista muodostui noin 850 kuvaa eli jokaista elettä kohden oli vähän yli 200 kuvaa. Kuvat jaettiin kahteen osaan, joista 80 % laitettiin koulutuskuviin ja loput 20 % testauskuviin. Koulutuskuvat ja testauskuvat tulisi jakaa mahdollisimman monipuolisesti kansioihin.

Kun kuvat ovat jaettu kansioihin on aika merkitä kohteet kuvista ilmaisen Labelimg-ohjelman avulla. Jokainen kuva tulee käydä yksi kerrallaan läpi ja jokainen kohde merkitään nelikulmiolla, joka sisältää kohteen kokonaan. Merkitsemisen jälkeen Labelimg kysyy mikä kohde nelikulmion sisällä on etikettiä varten. Etiketissä käytetty sana, merkki tai numero kohteesta tulee muistaa, sillä niitä käytetään myöhemmin koulutuksen valmistelussa. Kohteen merkitsemisen ja etiketin teon jälkeen Labelimg tallentaa kyseiset tiedot XML-tiedostona kuvan nimellä samaan kansioon alkuperäisen kuvan kanssa. Tämä toistettiin jokaisen kuvan kohdalla kummassakin kansiossa. Käsieleet nimettiin etiketteihin lyhennettyinä englanniksi ohand, chand, phand ja lhand.

5.4 **Mallin koulutuksen valmisteleminen**

Kuvien valmistelun jälkeen tuli valmistella tiedostoja koulutusta varten. TensorFlow haluaa kuvista saadut tiedot kahtena CVS-tiedostona, toinen koulutuskuville ja toinen testauskuville. Näissä CVS-tiedostoissa tuli olla jokaisen kuvan nimi, leveys, korkeus, kohteen nimi ja kohteen ympäröivän nelikulmion kahden vastakkaisen kulman tiedot. Kulmien tiedot tulisi

antaa muodossa pienin x-akselin arvo, pienin y-akselin arvo, isoin x-akselin arvo ja isoin y-akselin arvo, näillä arvoilla saadaan laatikon vastakkaiset kulmat tietoon ja TensorFlow tietää missä kohde on kuvassa. Alla kuvassa 14 näkyy osa koulutukseen käytetystä CVS-tiedostosta Excelissä avattuna.

	A	B	C	D	E
1	filename,width,height,class,xmin,ymin,xmax,ymax				
2	img_1.JPG,700,467,phand,136,124,599,405				
3	img_10.JPG,700,467,chand,198,83,544,383				
4	img_100.JPG,700,467,chand,179,65,478,361				
5	img_101.JPG,700,467,chand,201,83,497,406				
6	img_102.JPG,700,467,chand,162,60,490,326				
7	img_103.JPG,700,467,chand,196,62,535,382				
8	img_104.JPG,700,467,chand,217,103,482,386				
9	img_105.JPG,700,467,chand,161,65,466,412				
10	img_106.JPG,700,467,chand,185,127,549,402				
11	img_107.JPG,700,467,chand,182,140,554,420				
12	img_108.JPG,700,467,chand,147,108,501,406				
13	img_109.JPG,700,467,chand,171,69,530,456				
14	img_11.JPG,700,467,phand,137,128,644,388				
15	img_110.JPG,700,467,lhand,81,17,493,426				
16	img_111.JPG,700,467,lhand,49,9,489,450				
17	img_112.JPG,700,467,lhand,57,68,476,444				
18	img_113.JPG,700,467,lhand,84,106,444,403				
19	img_114.JPG,700,467,lhand,187,66,506,378				
20	img_115.JPG,700,467,lhand,214,41,630,399				
21	img_116.JPG,700,467,lhand,169,92,600,393				
22	img_117.JPG,700,467,lhand,136,107,518,405				

Kuva 14. Osa koulutuskuvakansion tiedoista CVS muodossa.

CVS-tiedoston voi tehdä itse käsin, mutta aina tulisi mieluummin käyttää ohjelmakoodia luomaan tiedosto ajan käytön takia. Pienilläkin kuva määrillä aikaa kuluu liian paljon sen luomiseen käsin.

Projektissa käytettiin kahta Dat Tranin ohjelmakoodia hänen pesukarhun havaitsemisen tietopaketista. Kyseiset ohjelmakoodit ovat xml_to_cvs.py ja generate_tfrecord.py, jotka molemmat lisättiin TensorFlow object_detection-kansioon. Ensimmäinen ohjelmakoodi luo CVS-tiedostot automaattisesti koulutus- ja testauskansioiden sisällöistä ja tallentaa CVS-tiedostot samalle tasolle kyseisten kansioiden kanssa. Toinen ohjelmakoodi luo valitun kansion kuvista ja niistä tehdyn CVS-tiedostosta RECORD-tiedoston, jota käytetään koulutuksessa.

Kun XML-tiedostot olivat tehty, voitiin ajaa xml_to_cvs.py, mutta ennen kuin ajettiin generate_tfrecord.py sitä tuli muokata, jotta se sopii tarkoitukseensa. Kuvassa 15 näkyy ainoa kohta mitä tuli muokata kyseisestä

ohjelmakoodista. Siihen tuli kirjoittaa kuvasta näkyvällä tavalla tunnistettavien kappaleiden nimet. Jälkimmäinen ohjelmakoodi ajettiin kummallakin kuvakansiolla ja siitä tehdyllä CVS-tiedostolla erikseen.

```
# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'ohand':
        return 1
    elif row_label == 'chand':
        return 2
    elif row_label == 'phand':
        return 3
    elif row_label == 'lhand':
        return 4
    else:
        None
```

Kuva 15. Tunnistettavat eleet lisättynä ohjelmakoodiin.

Tämän jälkeen luotiin labelmap niminen PBTXT-tiedosto, joka kertoi kouluttajalle mikä numero tarkoittaa mitä nimeä. Tiedostoon kirjoitetiin kuvassa 16 näkyvällä tavalla samat tiedot kuin aikaisemmassa kuvassa.

labelmap.pbtxt

```
item {
  id: 1
  name: 'ohand'
}

item {
  id: 2
  name: 'chand'
}

item {
  id: 3
  name: 'phand'
}

item {
  id: 4
  name: 'lhand'
}
```

Kuva 16. Labelmap.pbtxt avattuna Visual Studiolla.

Viimeiseksi ennen koulutusta konfiguroitiin mallin koulutus. Ensimmäiseksi siirrettiin object_detection\samples\config-kansiosta käytetyn mallin CONFIG-tiedosto object_detection\training-kansioon. Tässä tapauksessa CONFIG-tiedosto oli faster_rcnn_inception_v2_pets.config. Seuraavaksi juuri siirrettyyn tiedostoon tehtiin tarpeellisia muutoksia, jotta se löytäisi valmistellut tiedot. Kuvassa 17 on sinisellä rivinumero, jolle muutos tulee tehdä ja valkoisella mitä kyseisellä rivillä tulisi olla. Lopuksi

siirretään object_detection/legacy-kansiosta ohjelmistokoodi train.py object_detection-kansioon.

```
  9  num_classes: 4
106  fine_tune_checkpoint: "C:/tensorflow2/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"
123  input_path: "C:/tensorflow2/models/research/object_detection/train.record"
124  }
125  label_map_path: "C:/tensorflow2/models/research/object_detection/training/labelmap.pbtxt"
130  num_examples: 103
135  input_path: "C:/tensorflow2/models/research/object_detection/test.record"
136  }
137  label_map_path: "C:/tensorflow2/models/research/object_detection/training/labelmap.pbtxt"
```

Kuva 17. Tarvittavat muutokset mallin CONFIG-tiedostoon.

5.5 **Mallin kouluttaminen ja testaaminen**

Mallin kouluttaminen aloitettiin käyttämällä train.py ohjelmistokoodia komennolla, joka näkyy kuvassa 18. Koulutuksen alkamiseen meni noin 30 sekuntia, jonka jälkeen komentokehotteessa alkoi näkyä missä askeleessa koulutus eteni, kuinka paljon häviötä on ja kuinka paljon aikaa kulunut kyseisessä askeleessa.

```
python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/faster_rcnn_inception_v2_pets.config
```

Kuva 18. Mallin koulutuksen aloitus komento Anacondassa.

Mallin koulutusta tulisi jatkaa, kunnes häviö on tasoittunut ja pysyy alle 0,05 kun käytetään faster_rcnn mallia (EdjeElectronics, 2019). Tähän kului aikaa noin 8 tuntia ja 32000 askelta käyttäessä NVIDIA GTX 1070-näytönohjainta koulutukseen. Koulutuksen edistymistä voitiin seurata Anacondan komentokehotteesta tai TensorBoardin avulla. TensorBoardin avaamiseksi tulee avata uusi Anaconda komentokehote, jolla aktivoidaan virtuaaliympäristö ja käytetään komentoa ’tensorboard –logic=training’. Tämä tekee selaimella avattavan lokaalin nettisivun osoitteeseen tietokoneennimi:6006, mikä näyttää tietoja ja graafeja koulutuksen edistymisestä.

Kun koulutus oli saavuttanut halutun tason, se pysäytettiin painamalla Ctrl+C komentokehotteessa. Koulutuksella on tallennuspiste viiden minuutin välein, mikä tallentaa mallin uusimman version training-kansioon. Koulutuksen voi jatkaa edellisestä tallennuspisteestä, jos esimerkiksi sähköt katkeavat koulutuksen aikana.

Tunnistusta varten tarvittiin jäädytetty päättely graafi, joka saatiin ajamalla export_inference_graph.py ohjelmistokoodi johon annettiin haluttu tallennuspiste mallista, joka on muodossa model.ckpt-xxxx jossa xxxx kuvaa mallin askelmäärää. Inference_graph-kansiosta löytyy tämän jälkeen frozen_inference_graph.pb-tiedosto, joka on kappaleen tunnistin.

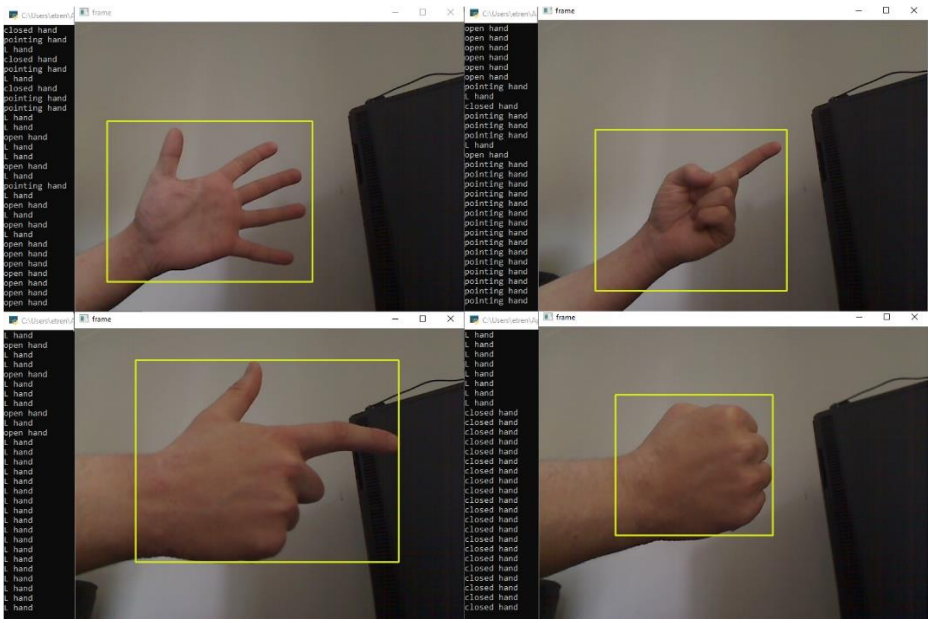
Lopuksi ladattiin malliin sopivan konfiguraatio tiedoston tekemisen ohjelmistokoodi OpenCV TensorFlow Object Detection API sivulta, mikä oli tässä tapauksessa tf_text_graph_faster_rcnn.py. Ohjelmistokoodi tekee mallin ja sen konfiguraatio tiedoston pohjalta graph.pbtxt-tiedoston, joka on teksti esitys muoto jäädytetystä graafista. Ajon jälkeen tulisi olla frozen_inference_graph.pb ja graph.pbtxt joilla voitiin kokeilla tunnistusta kuvassa 19 olevalla Python-koodilla.

PythonApplication2.py

```
import cv2 as cv

cvNet = cv.dnn.readNetFromTensorflow('frozen_inference_graph.pb', 'graph.pbtxt')
cap = cv.VideoCapture(0)
while True:
    hasFrame, frame = cap.read()
    if not hasFrame:
        cv.waitKey()
        break
    rows = frame.shape[0]
    cols = frame.shape[1]

    cvNet.setInput(cv.dnn.blobFromImage(frame, size=(cols, rows), swapRB=True, crop=False))
    cvOut = cvNet.forward()

    for detection in cvOut[0,0,:,:]:
        score = float(detection[2])
        if score > 0.3:
            left = detection[3] * cols
            top = detection[4] * rows
            right = detection[5] * cols
            bottom = detection[6] * rows
            cv.rectangle(frame, (int(left), int(top)), (int(right), int(bottom)), (23, 230, 210), thickness=2)
            if detection[1] == 0:
                print("open hand")
            if detection[1] == 1.0:
                print("closed hand")
            if detection[1] == 2.0:
                print("pointing hand")
            if detection[1] == 3.0:
                print("L hand")

    cv.imshow('frame', frame)
    keyboard = cv.waitKey(1)
    if keyboard == 'q':
        break
```

Kuva 19. Python ohjelmistokoodi, joka tunnistaa käsieleitä käyttäen webkameraa.

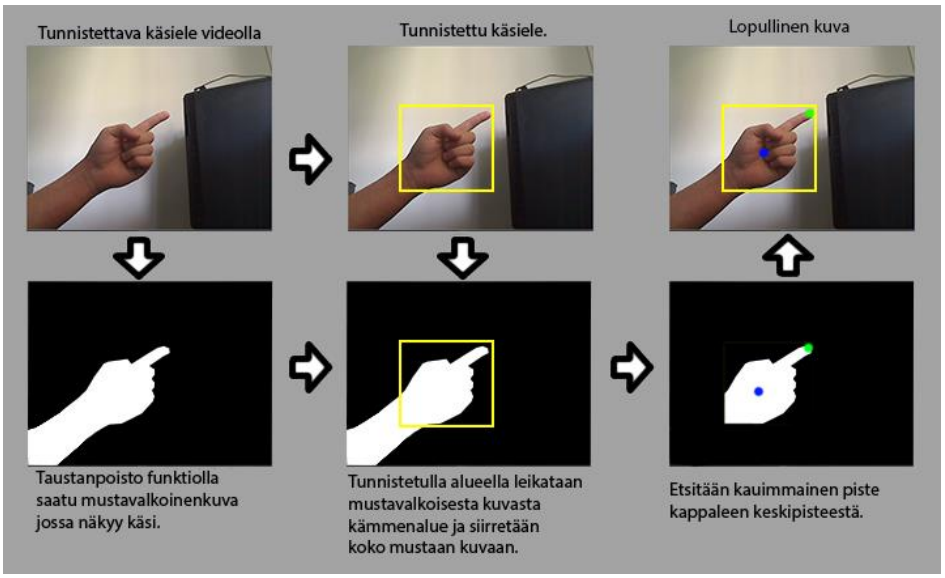
Ohjelmistokoodi avaa webkameran ja tekee keltaisia nelikulmioita ohjelman havaitsemien käsieleiden ympärille. Se myös tulostaa konsoliin mikä ele tunnistettiin. Eleiden tunnistuksen nopeus oli noin viisi ruutua sekunnissa ja läheltä eleet tunnistetiin korkealla onnistumisella. Kuvassa 20 näkyy ohjelmistokoodi toiminnassa ja kuinka se tunnistaa eleitä.



Kuva 20. Kädeneleiden tunnistus toiminnassa Python ohjelmistokoodilla.

5.6 **Käsieleiden tunnistaminen ja sormenpään löytäminen**

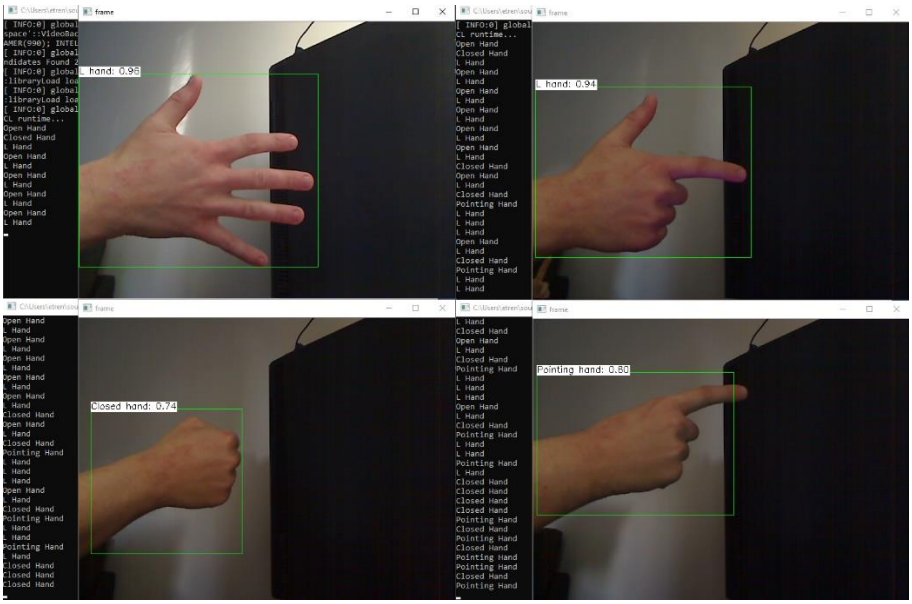
Kirjoitettu C++ ohjelmistokoodi koostuu kolmesta pää funktiosta, joita ovat main, handdetect ja figertip. Funktioiden koodit löytyvät liitteestä yksi. Ohjelmiston toimintaa kuvaa kuva 21 jossa näkyy päätoiminnat ja missä järjestyksessä ohjelma tekee tehtäviä.



Kuva 21. C++ ohjelmiston toimintakaava.

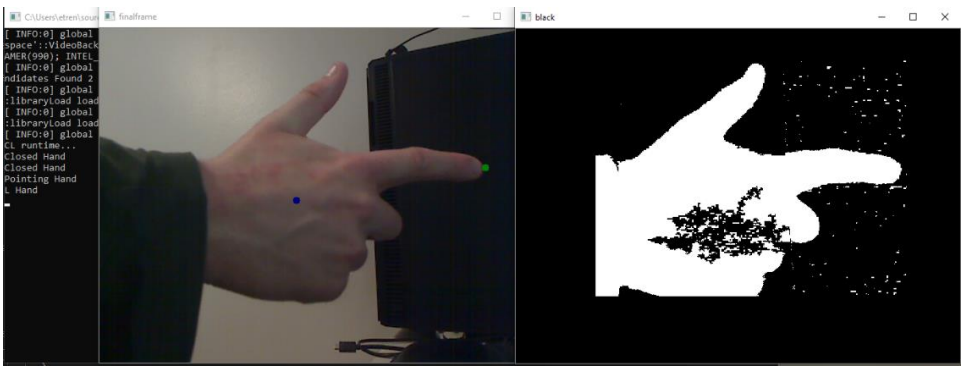
Main-funktio alustaa tunnistus mallin Net-luokkana, kameran käynnistyksen ja ruudun talteenoton tunnistusta varten. Se sisältää myös OpenCV

taustanpoisto funktion, jolla saadaan mustavalkokuva ruudusta, jossa näkyy kaikki taustaan kuulumaton valkoisena mustalla taustalla. Main-funktio käynnistää handdetect-funktion ruudun kappauksen oton jälkeen. Kuvassa 22 handdetect-funktion käden tunnistuksia.



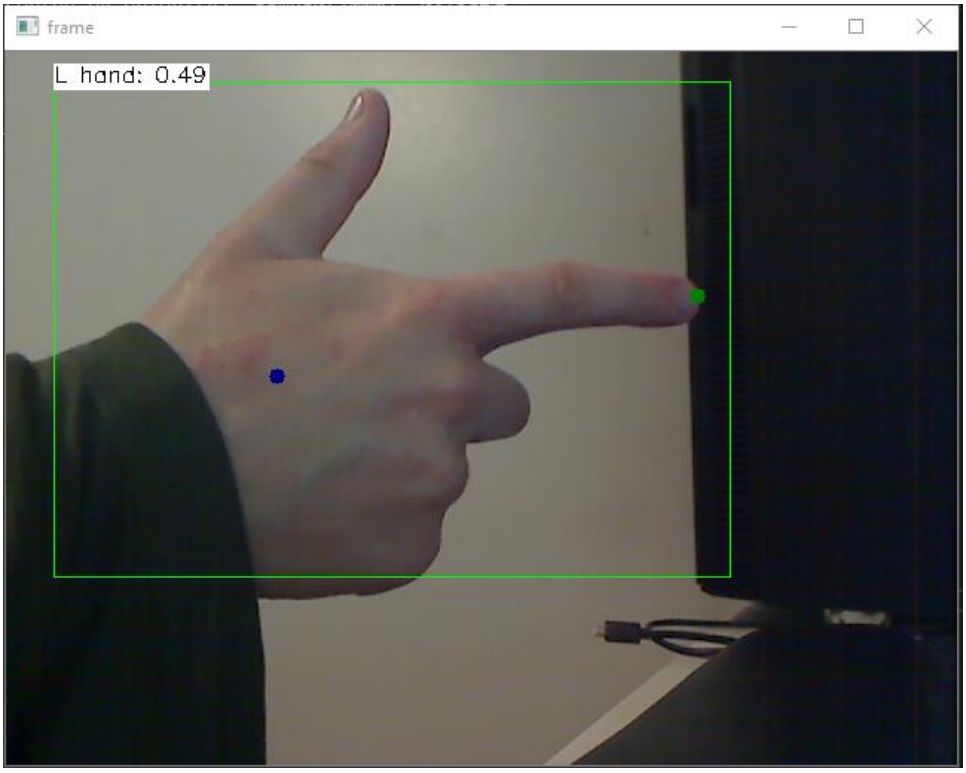
Kuva 22. C++ ohjelmistokoodilla toimiva käsiele tunnistin. Tunnistuksen yläpuolella tunnistetun eleen nimi.

Handdetect-funktio sisältää tunnistuksen ja sen visualisoimiseen vaadittavat koodi rivit ja funktiot. BlobFromImage-funktiolla muutetaan kaapattu ruutu muotoon, mikä voidaan antaa mallin Net-luokkaan tunnistusta varten. Tämän jälkeen se asetetaan Net-luokan setInput-funktioalla tunnistimeen ja Net-luokan forward-funktiolla saadaan lista tunnistuksista. Lista sisältää jokaista tunnistusta kohden tunnistus numeron, luottavaisuuden ja tunnistusalueen reunat. Jos tunnistus luottavaisuus ylittää kynnysarvon laatikko piirretään kohteen ympärille ja sormenpään tunnistus funktio alkaa, jos tunnistettu kohde on osoittava ele tai käsiele, joka muistuttaa L-kirjainta. Fingertip-funktiolle annetaan musta kuva, johon leikataan Main-funktiossa saadusta taustanpoisto kuvasta alue, joka on tunnistettu handdetect-funktiolla. Tällöin fingertip-funktiolle annettu kuva tulisi olla musta, jossa näkyy valkoinen käden muoto. Kuvassa 23 Fingertip-funktion tunnistus.



Kuva 23. Fingertip-funktion löytämä sormenpää merkitty vihreällä pisteellä.

Fingertip-funktiossa etsitään mustavalkokuvan käden ääriviivat OpenCV findContours-funktiolla. Funktio antaa listan koordinaattipiste listoja. Tämä lista listoista käydään läpi for-silmukalla ja contourArea-funktiolla etsitään ääriviivojen alueen koko, jota verrataan isoimpaan löydettyyn, jotta löydetään isoin ääriviiva alue kuvasta, mikä on käsi. Tällä toiminnalla saadaan varmistettua, että sormenpään etsintä tehdään kädelle eikä mahdolliselle taustamelulle. Tässä vaiheessa etsitään käden keskipiste ja käydään for-silmukan avulla käden ääriviivojen pisteet läpi. Silmukan tarkoitus on verrata jokaisen reunapisteen etäisyyttä keskipisteeseen ja tallentaa kauimmainen piste keskipisteestä, mikä on osoittavan sormenpää. Silmukan jälkeen sormenpään piste piirretään kuvaan, jossa on myös tunnistus nelikulmio. Kuvassa 24 tunnistus näkymä lopullisesta ohjelmistokoodista, joka käyttää C++-kieltä.



Kuva 24. Eletunnistin tunnistaa kädeneleen ja löytää sormenpään webkameran kuvasta.

C++ ohjelmistokoodi tunnistaa käsieleitä nopeudella noin 0,33 ruutua sekunnissa. Eleet tunnistetaan tyydyttävällä tarkkuudella läheltä ja kaukaa tunnistin ei tunnista juuri mitään. Paikoittain tunnistin saattaa tunnistaa yksittäisen eleen pitkältä.

6 YHTEENVETO

Työn aihe oli äärimmäisen mielenkiintoinen, sillä konenäön käyttäminen käsien eleiden tunnistamiseen ja paikantamiseen mahdollistaa elepohjaisten järjestelmien käytön esimerkiksi videopeleissä, interaktiivisissa sovelluksissa tai ohjaamaan kodin älylaitteita.

Tässä työssä käytetyssä kuvakannassa oli paljon ongelmia mitä jälkikäteen muuttaisin. Suurin osa kuvista oli otettu läheltä kämmenaluetta paljaalla taustalla. Lähes kaikkiin kuviin muokattiin jotain ylimääräistä koska koulutus ei olisi tullut onnistumaan hyvin, jos malli ei saa tarpeeksi vaihtelevaa dataa. Kuvat tulisi myös ottaa eri etäisyyksiltä eikä pelkästään läheltä. Uskon vahvasti, että tämä on vaikuttanut massiivisesti mallin kykyyn tunnistaa eleitä kaukaa. Odottamattomia ongelmia tuli myös Python- ja C++-versioiden välillä. Vain mallin testaamiseen tarkoitettu Python-version toiminta nopeus oli parikertaa isompi kuin C++-versio, jonka toiminta nopeus jäi tavoitteiden alle.

Yksi jatkokehityskohteista olisi suorituskyvyn optimointi, tämä voitaisiin saavuttaa kouluttamalla nopeampi malli, jossa on pienempi tarkkuus, mutta paremmalla koulutuskuvatietokannalla. Tällä saavutettaisiin nopeampi tunnistaminen ja parempi etäisyys tunnistamiselle. Toinen jatkokehityskohde voisi olla useamman käsieleen tunnistaminen. Tämän saavuttamiseen tarvittaisiin kattavampi kuvatietokanta, jonka koko kasvaisi helposti tuhansiin kuviin. Kolmas voisi olla etsiä käsistä avainpisteitä, jotka voisivat olla vaikka nivelet ja sormenpäät. Tätä voitaisiin käyttää käsien animoimisessa ilman seurantapisteitä tai muuta kalustoa.

LÄHTEET

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV: Computer vision with OpenCV Library* (Ensimmäinen p.). Sebastopol: O'Reilly Media, Inc. Haettu 29.5.2020 osoitteesta https://books.google.fi/books?hl=fi&lr=&id=seA-giOfu2EIC&oi=fnd&pg=PR3&dq=Learning+OpenCV:+Computer+Vision+with+the+OpenCV+Library&ots=hUM0dkgBTh&sig=53xHe3BpJ1emr_Y9wVK-ZUbW317I&redir_esc=y#v=onepage&q&f=false

Cronje, D. (2018). *Neural Networks Easily Fooled*. Haettu 29.5.2020 osoitteesta https://medium.com/deep-learning-cafe/neural-networks-easily-fooled-e19bf575b527

Davies, R. (2012). *Computer and Machine Vision: Theory, Algorithms, Practicalities* (Neljäs p.). Oxford: Academic Press. Haettu 29.5.2020 osoitteesta https://books.google.fi/books?id=lwl7ZjOwq8gC&printsec=frontcover&hl=fi&source=gbs_atb#v=onepage&q&f=false

Demaagd, K., Oliver, A., Oostendrop, N. & Scott, K. (2012). *Practical Computer Vision with SimpleCV: The Simple Way to Make Technology See.* Sebastopol: O'Reilly. Haettu 30.5.2020 osoitteesta https://books.google.fi/books?hl=fi&lr=&id=4st9l6QlkpUC&oi=fnd&pg=PR3&dq=Practical+Computer+Vision+with+SimpleCV:+The+Simple+Way+to+Make+Technology+See&ots=SOdIxZc9Ci&sig=eZ-mWmUx50fvmCj5jw1FMpdQw--4&redir_esc=y#v=onepage&q&f=false

Dove, K. (2006). *NVIDIA Unveils CUDA™-The GPU Computing Revolution Begins*. Haettu 30.5.2020 osoitteesta https://www.nvidia.com/object/IO_37226.html

EdjeElectronics. (2019). *TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10*. Haettu 30.5.2020 osoitteesta https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10#2f-compile-protobufs-and-run-setuppy

Heaven, D. (2019). *Why deep-learning AIs are so easy to fool*. Haettu 29. 5.2020 osoitteesta https://www.nature.com/articles/d41586-019-03013-5

Ihritik. (n.d). *MATLAB | RGB image representation.* GeeksforGeeks. Haettu 29.5.2020 osoitteesta https://www.geeksforgeeks.org/matlab-rgb-image-representation/

ImageMagick. (2020). *Develop*. Haettu 30.5.2020 osoitteesta https://imagemagick.org/script/develop.php

Khan, S., Rahmani, H., Shah, S. & Bennamoun, M. (2018). *A Guide to Convolutional Neutral Networks for Computer Vision.* San Rafeal: Morgan & Claypool. Haettu 29.5.2020 osoitteesta https://scholar.google.fi/scholar?q=A+Guide+to+Convolutional+Neural+Networks+for+Computer+Vision&hl=fi&as_sdt=0&as_vis=1&oi=scholart [PDF] academia.edu linkki hakutuloksen vieressä.

Martin, D. (2013). *A Practical Guide to Machine Vision Lighting.* Advanced illumination. Haettu 29.5.2020 osoitteesta https://www.advancedillumination.com/wp-content/uploads/2018/10/A-Practical-Guide-to-Machine-Vision-Lighting-v.-4-Generic.pdf

OpenCV. (2020). *About*. Haettu 30.5.2020 osoitteesta https://opencv.org/about/

Ponder, G. (2012). *How do flagship Windows Phone 8 cameras compare to a prosumer DSLR? We find out.* Windows Central. Haettu 29.5.2020 osoitteesta https://www.windowscentral.com/how-does-your-windows-phone-camera-measure

Pxhere. (n.d). Pxhere. Haettu 29.5.2020 osoitteesta https://pxhere.com/en/photo/636445

SRI International. (2019). *SRI International*. Haettu 29.5.2020 osoitteesta https://archive.sri.com/engage/products-solutions/embedded-computer-vision

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications.* Berliini: Springer. Haettu 29.5.2020 osoitteesta https://books.google.fi/books?hl=fi&lr=&id=bXzAlkODwa8C&oi=fnd&pg=PR4&dq=computer+vision&ots=g_37amGDD&sig=LasvrhEYAmrBKgKbQoTEGzxdUUA&redir_esc=y#v=onepage&q&f=false

Tzutalin. (2020). *Labelimg*. Haettu 30.5.2020 osoitteesta https://github.com/tzutalin/labelImg

Vision Online Marketing Team. (2017). *Choosing the Right Machine Vision Software for Your Application*. Haettu 29.5.2020 osoitteesta https://www.visiononline.org/blog-article.cfm/Choosing-the-Right-Machine-Vision-Software-for-Your-Application/85

Vision Online Marketing Team. (2018). *What is Embedded Vision?* Haettu 29.5.2020 osoitteesta https://www.visiononline.org/blog-article.cfm/What-is-Embedded-Vision/90

KOODIN PÄÄFUNKTIOT

Main-funktio

```
int main()
{
    classes.push_back("Open hand");
    classes.push_back("Closed hand");
    classes.push_back("Pointing hand");
    classes.push_back("L hand");
    //background subtractor
    Ptr<BackgroundSubtractor> pBackSub;
    pBackSub = createBackgroundSubtractorMOG2();
    //

    String model = "frozen_inference_graph.pb";
    String config = "graph.pbtxt";
    Scalar mean = 0;


    net = readNetFromTensorflow(model, config);
    VideoCapture capture(0);
    if (!capture.isOpened()) {
        //error in opening the video input
        cerr << "Unable to open: webcam" << endl;
        return 0;
    }
    Mat frame, fgMask;

    while (waitKey(1) < 0)
    {
        capture >> frame;
        capture >> ProFrame;
        if (frame.empty())
        {
            waitKey();
            break;
        }
        //update the background model
        pBackSub->apply(frame, fgMask, 0.0001);

        cv::threshold(fgMask, BW, 225, 255, THRESH_BINARY);
        Black = frame.clone();
        Black.setTo(Scalar(0, 0, 0));
        cv::cvtColor(Black, Black, COLOR_RGB2GRAY);

        //
        handDetect(frame, mean);

        int keyboard = waitKey(30);
        if (keyboard == 'q' || keyboard == 27)
            break;
    }
    return 0;
}
```

Liite 1/2

Handdetect-funktio

```
void handDetect(Mat frame, Scalar mean)
{

    Size inpSize(inpWidth > 0 ? inpWidth : frame.cols,
        inpHeight > 0 ? inpHeight : frame.rows);

    rows = frame.rows;
    cols = frame.cols;


    blobFromImage(frame, blob, 1.0, inpSize, mean, true, false);

    net.setInput(blob);
    vector<Mat> outs;
    //resize(frame, frame, inpSize);

    net.forward(outs);

    std::vector<int> classIds;
    std::vector<float> confidences;
    std::vector<Rect> boxes;

    CV_Assert(outs.size() == 1);
    float* data = (float*)outs[0].data;
    for (size_t i = 0; i < outs[0].total(); i += 7)
    {
        float confidence = data[i + 2];
        if (confidence > confThreshold)
        {
            int left = (int)(data[i + 3] * frame.cols);
            int top = (int)(data[i + 4] * frame.rows);
            int right = (int)(data[i + 5] * frame.cols);
            int bottom = (int)(data[i + 6] * frame.rows);
            int width = right - left + 1;
            int height = bottom - top + 1;

            int a = data[i + 1];
            printID(a);

            classIds.push_back((int)(data[i + 1]));
            boxes.push_back(Rect(left, top, width, height));
            confidences.push_back(confidence);
              //Fingertip detection
            if (data[i + 1] == 2 || data[i + 1] == 3)
            {

                Rect rect(left, top, width, height);
                Mat ROI = BW(rect);
                ROI.copyTo(Black(rect));

                fingertip(Black);

            }
        }
    }
    std::vector<int> indices;
    NMSBoxes(boxes, confidences, confThreshold, nmsThreshold, indices);
    for (size_t i = 0; i < indices.size(); ++i)
    {
        int idx = indices[i];
        Rect box = boxes[idx];
        drawPred(classIds[idx], confidences[idx], box.x, box.y,
            box.x + box.width, box.y + box.height, frame);
    }
    imshow("frame", frame);

    //imshow("daw", Black);
}

```

Liite 1/3

Fingertip-funktio

```
void fingertip(Mat BlackAndWhite)
{
    cv::findContours(BlackAndWhite.clone(), handcontours, hierarchy, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE, Point());
    Rect rect;
    MaxArea = 0.0;
    savedcontour = -1;

    //find largers contour from all of the contours in picture
    for (int i = 0; i < handcontours.size(); i++)
    {
        double area = contourArea(handcontours[i]);

        if (area > MaxArea)
        {
            MaxArea = area;
            savedcontour = i;

        }

    }
    if (savedcontour > 0)
    {

        drawContours(Black, handcontours, savedcontour, Scalar(255), cv::FILLED, 8);
        imshow("black", Black);
        Moments m = moments((Black >= 50), true);
        Point2d p(m.m10 / m.m00, m.m01 / m.m00);
        Point2d f(0, 0);
        double distance, maxdistance = 0;

        vector<vector<Point>>hull(handcontours.size());
        cv::convexHull(handcontours[savedcontour], hull[savedcontour], true);

        for (int i = 0; i < handcontours.size(); i++)
        {
            for (int j = 0; j < hull[i].size(); j++)
            {
                float diffY = hull[i][j].y - p.y;
                float diffX = hull[i][j].x - p.x;
                distance = sqrt((diffY * diffY) + (diffX * diffX));
                distance = fabs(distance);

                if (distance > maxdistance)
                {
                    maxdistance = distance;
                    f.x = hull[i][j].x;
                    f.y = hull[i][j].y;
                }
            }
        }
        //f point is fingertip location
        //fingertipX = f.x;
        //fingertipY = f.y;
        circle(ProFrame, p, 5, Scalar(128, 0, 0), -1);
        circle(ProFrame, f, 5, Scalar(0, 128, 0), -1);

    }

    imshow("finalframe", ProFrame);
}

```