Jani Aalto

HAHMONTUNNISTUKSEN TOTEUTUS TEKOÄLYMOBIILI-SOVELLUKSEEN

HAHMONTUNNISTUKSEN TOTEUTUS TEKOÄLYMOBIILI-SOVELLUKSEEN

Jani Aalto
Opinnäytetyö
Syksy 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Jani Aalto
Opinnäytetyön nimi suomeksi: Hahmontunnistuksen toteutus tekoälymobiilisovellukseen
Opinnäytetyön nimi englanniksi: Implementing Object Detection in a Mobile AI Application
Työn ohjaajat: Eino Niemi, Tommi Mänttäri
Työn valmistumislukukausi ja -vuosi: Syksy 2020
Sivumäärä: 27 + 1 liite

Työn tavoitteena oli toteuttaa hahmontunnistus Android-pohjaiseen Alvin-tekoälysovellukseen. Työn tilaajana oli Alvinin kehittänyt tmi Tommi Olavi Mänttäri. Hahmontunnistuksen oli tarkoitus mahdollistaa samanaikaisesti kehitettävän koulujen opetuskäyttöön tarkoitetun robottikäden ohjaus Alvinilla. Hahmontunnistuksen tuli tunnistaa yleisiä, kouluista löytyviä esineitä joita robotti kykenisi nostamaan.

Työ toteutettiin Keras-rajapintaa käyttävällä TensorFlow-koneoppimiskirjastolla, Android Studiolla sekä Google Colab -palvelulla. Se suoritettiin itsenäisesti kotoa käsin. Teorian opiskelun jälkeen tutkittiin TensorFlow-esimerkkiohjelmia, ja valitun ohjelman pohjalta toteutettiin hahmontunnistusnäkymä Alviniin. Koska esimerkkiohjelman käyttämä neuroverkkomalli ei tunnistanut riittävän monta haluttua esineen luokkaa, uudelleenopetettiin toinen malli siirto-opetusta käyttäen tunnistamaan 13 muuta luokkaa. Koska opetus epäonnistui TensorFlow’n komentoriviversiolla, opetus tehtiin uudelleen Google Colabissa. Tuloksena oli toimiva malli, joka oli kuitenkin rakenteeltaan yksinkertaisempi kuin esimerkkiohjelman malli. Mallien yhteensovitus Android-koodissa onnistui kohtuullisen hyvin.

Tuloksena oli toimiva ja tosiaikainen, molempia malleja hyväksikäyttävä hahmontunnistustoiminnallisuus Alvin-sovelluksessa, joskin hahmontunnistuksen luotettavuus oli toivottua huonompi. Toteutettu hahmontunnistus toimii hyvänä perustana sovelluksen jatkokehitykselle.

Asiasanat: hahmontunnistus, koneoppiminen, mobiilisovellukset, TensorFlow, Keras, Android

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author: Jani Aalto
Title of thesis: Implementing Object Detection in a Mobile AI Application
Supervisors: Eino Niemi, Tommi Mänttäri
Term and year when the thesis was submitted: Autumn 2020
Pages: 27 + 1 appendix

The aim of the thesis was to develop an object detection capability for the Android based Alvin AI application. The work was done for the developer of Alvin, Tommi Olavi Mänttäri. The purpose of the object detection functionality was to allow the user to control a robot arm through Alvin. The robot was intended for educational use and was being developed simultaneously. The object detection was supposed to detect objects commonly found in a school and which the robot arm could lift.

Tools used in the thesis were the machine learning library TensorFlow, which uses the Keras API, Android Studio and Google Colab. The work was done independently from home. After studying machine learning theory, several TensorFlow example applications were studied and object detection was implemented in Alvin using the most suitable one as a base. Because the neural net model included in this application did not recognise enough desired classes of objects, a new model was taught using transfer learning to recognise 13 additional ones. Since the training failed using TensorFlow's command line version, the training was repeated using Google Colab. This produced a functioning model, which was however structurally simpler than the model from the example application. The models were combined in the Android code adequately well.

The result was a functioning real-time object detection functionality which utilised both neural net models, although the reliability of the object detection was not as good as desired. The result works as a good basis for further development of the application.

Keywords: object detection, machine learning, mobile applications, TensorFlow, Keras, Android

SISÄLLYS

1 JOHDANTO

Tämän työn tavoitteena oli toteuttaa syväopetettua neuroverkkomallia käyttävä hahmontunnistus Android-pohjaiselle Alvin AI Assistant -tekoälysovellukselle. Alvin on muun muassa opetuskäyttöön suunniteltu symbolinen tekoäly, ja tämä opinnäytetyö liittyy projektiin, jonka tavoitteena on luoda Alvinilla ohjattava robotti koulujen opetustarpeisiin. Lopputuloksena tuli olla arkipäiväisiä esineitä englannin kielellä tunnistava hahmontunnistustoiminnallisuus Alvin-sovellukseen. Työssä käytettiin Android Studiota, TensorFlow-ohjelmistokehystä ja kahta neuroverkkomallia.

Työ koostui teorian opiskelusta, TensorFlow-esimerkkiohjelmien tutkimisesta, sopivan neuroverkkomallin uudelleenopetuksesta sekä hahmontunnistuksen toteuttamisesta Alvin-sovellukseen esiopetetun sekä uudelleenopetetun neuroverkkomallin avulla.

2 KONEOPPIMINEN

Koneoppiminen on tekoälyyn perustuva monipuolinen ja kukoistava teknologia-ala, joka syntyi viime vuosituhannella mutta jonka merkitys kasvaa jatkuvasti laitteiden laskentatehon ja olemassaolevien datamäärien myötä. Se eroaa perinteisestä ohjelmoinnista siten, että ohjelmoija ei luo tiettyjä sääntöjä, joiden avulla tietokoneohjelma voi käsitellä dataa ja tuottaa vastauksia. Sen sijaan koneoppimisessa ohjelmalle annetaan dataa ja sen annetaan itse oppia sääntöjä, joilla syötetystä datasta saadaan tulokseksi haluttuja vastauksia. Näin ohjelma voidaan opettaa tunnistamaan tiettyjä piirteitä datasta ja toteuttamaan tavallisesti vain ihmisten osaamia tehtäviä, kuten puheen ymmärtäminen tai kuvien tunnistaminen. (1, s. 6.)

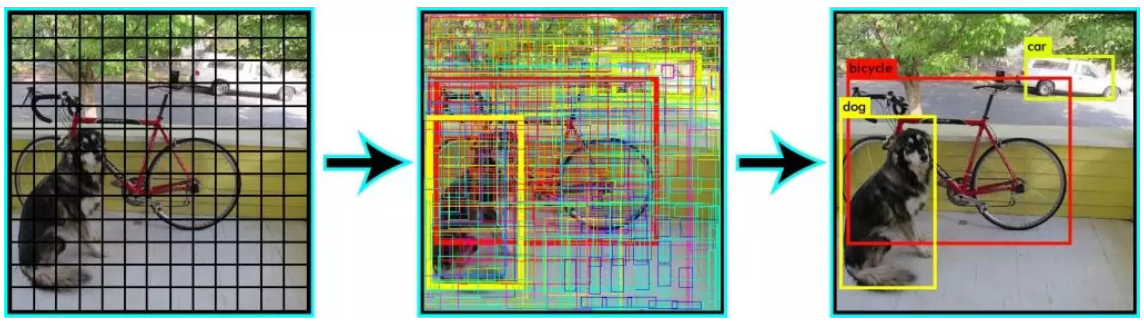
Koneoppiminen jaetaan yleensä kolmeen kategoriaan: ohjattu oppiminen (supervised learning), ohjaamaton oppiminen (unsupervised learning) ja vahvistusoppiminen (reinforcement learning). Ohjatussa oppimisessa ohjelmalle annetaan opetettaessa syötedatan lisäksi esimerkkejä halutuista lopputuloksia, joihin ohjelma pystyy vertaamaan tuottamiaan tuloksia ja siten muuttamaan oppimistaan. Tämä on kaikkein yleisin koneoppimisen muoto. Ohjaamattomassa opetuksessa ohjelma päättelee sopivat lopputulokset datassa olevista säännönmukaisuuksista. Vahvistusoppimisessa ohjelmalle annetaan palautetta sen suorituskyvystä, minkä perusteella se voi parantaa oppimistaan. (1, s. 94–95; 2.)

Koneoppiminen hyväksikäyttää valtavia datamääriä, joiden perinteinen tilastollinen analyysi olisi vaikeaa mutta joiden avulla ohjelmat voivat oppia tunnistamaan kuvioita syötetyssä datassa. Koneoppiminen on siten käytännönläheinen teknologia-ala, joka perustuu tilastollisesta lähestymistavastaan huolimatta vain vähän matemaattiseen teoriaan. (1, s. 6.) Sillä on paljon erilaisia sovellusalueita joissa on tarvetta suurten datamäärien käsittelyyn, kuten hakukoneet, personalisoitu mainonta, automatisoitu kielten kääntäminen ja konenäkö (1, s. 11–12).

2.1 Hahmontunnistus

Hahmontunnistus on koneoppimisen yleinen sovellusalue, joka tarkoittaa objektien, kuten esineiden, ihmisten, kasvojen tai kuvan etu- ja tausta-alan tunnistamista digitaalisista kuvista tai videoista. Kuvien hahmontunnistus koostuu objektien paikannuksesta kuvassa yhdistettynä havaittujen objektien luokitteluun. Sitä käytetään muun muassa kasvojen tunnistukseen, tekstin digitalisointiin ja robottien navigaatioon. Objektien paikannus (object localisation) tarkoittaa rajaavan suorakaiteen (bounding box), piirtämistä esineen ympärille esineen rajoille. Kuvien luokittelu (image classification) tarkoittaa kuvan määrittämistä kuuluvaksi yhteen tai useampaan nimettyyn luokkaan. (3.)

Tyypillisesti hahmontunnistukseen käytetään konvolutiivista neuroverkkoa (convolutional neural network), joka on syväoppimiseen kykenevä neuroverkko. Yleisimmät hahmontunnistuksessa käytettävät algoritmityypit ovat Region-Based Convolutional Neural Network (R-CNN), You Only Look Once (YOLO) ja Single Shot Detector (SSD). R-CNN keksittiin alun perin vuonna 2014, ja sen toimintaperiaatteena on ensin poimia kuvasta alueita, jotka vaikuttavat luokiteltavilta objekteilta, minkä jälkeen jokaiselle alueelle tehdään erikseen kuvanluokittelu. YOLO keksittiin seuraavana vuonna, ja sentyyppiset regressiiviset hahmontunnistusalgoritmit ovat huomattavasti nopeampia mutta yleensä vähemmän tarkkoja. YOLO jakaa käsiteltävän kuvan tiettyyn määrään ruutuja, joista jokaisen kohdalla luodaan ruudun ympärille alustavasti objekteja rajaavia suorakaiteita. Näistä valitaan todennäköisimmin oikeat, joille ennustetaan kaikille kerralla niiden rajaamien objektien luokat. (3.) (Kuva 1.)



KUVA 1. YOLO-hahmontunnistusalgoritmin toiminta (4)

SSD toimii enimmäkseen kuten YOLO mutta se ennustaa objektien rajaavia suorakaiteita useita kertoja neuroverkon kuvan käsittelyn aikana, mikä parantaa eri kokoisten objektien tunnistamista (5).

2.2 Neuroverkot

Neuroverkko koostuu neuroneista koostuvista kerroksista, joiden neuronit ovat kytkeytyneinä toisiinsa sekä seuraavan kerroksen neuroneihin vaihtelevilla tiheyksillä. Neuronit itse ovat solmupisteitä, joissa syötedata-arvolle tehdään matemaattinen operaatio. Yksinkertaisimmissa verkoissa kaikki neuronit ovat täysin yhteenkytkeytyneitä mutta konvolutiivisissa verkoissa neuronit kytkeytyvät vain muutamiin toisiin neuroneihin edellisessä ja seuraavassa kerroksessa, mikä johtaa verkon monimutkaisempaan toimintaan. Verkon alin ja ylin kerros on tarkoitettu syöte- ja tulostedatan käsittelyyn, ja niiden välillä olevat kerrokset tekevät datalle matemaattisia operaatioita. Viimeinen, täysin yhteenkytkeytynyt kerros tuottaa arvoja, jotka (kuvantunnistuksen tapauksessa) ennakoivat kuvan luokkaa eli kuvan esittämää objektia. (6.)

Neuroverkkoihin syötettävä data on tensorimuotoista. Tensorit ovat lukuja sisältäviä taulukkoja, jotka voivat olla minkä tahansa ulotteisia, kuten kaksiulotteiset matriisit tai kolmiulotteiset matriiseista koostuvat taulukot. Kuvadata tallennetaan tyypillisesti kolmiulotteisiin tensoreihin, sillä se koostuu arvoista korkeus x leveys x värikanavat, ja kuvasarja tallennetaan neliulotteiseen tensoriin. (1, s. 35–36.)

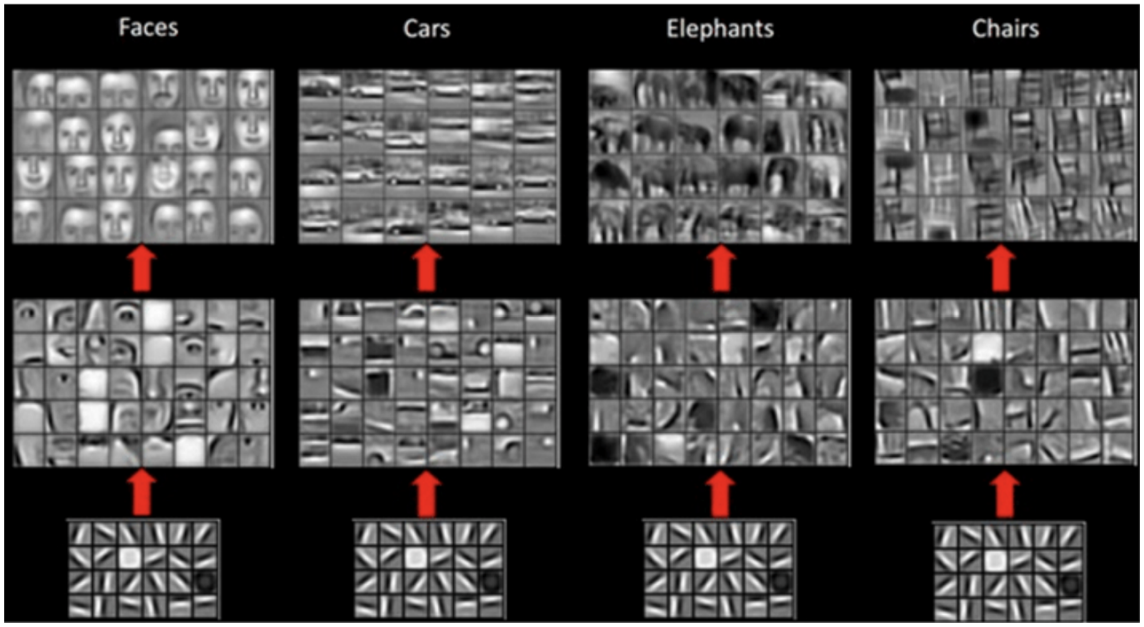
Neuroverkkojen tehokkuus perustuu niiden kykyyn parantaa omaa oppimistehokkuuttaan opetuksen aikana. Kuvantunnistukseen opetettavaa neuroverkkoa ohjatusti opetettaessa syötetyt kuvat on luokiteltu ennakolta, joten verkko voi laskea etäisyysfunktion eli arvon, joka kuvaa verkon tuottaman ennakon virhemäärän suuruuden. Opetuksen alussa painoarvot ovat satunnaisia mutta verkko säätää arvoja jokaisen opetusiteraation jälkeen ja muuttaa kerrosten painoarvoja (määrää jolla neuronit muuttavat niiden läpi kulkevaa numerodataa) virheen vähentämiseksi. Menetelmää, jolla neuroverkko parantaa itseään opetuksen kuluessa kutsutaan nimellä vastavirta-algoritmi (backpropagation). (1, s. 10–11.) Neuroverkon liika opetus samalla

opetusdatalla johtaa ylisovittamiseen (overfitting), jossa verkko oppii vain opetusdataa koskevia piirteitä, jotka eivät päde yleisemmin (1, s. 76). Opetetun neuroverkon staattista representaatiota, joka voidaan ladata haluttuun ohjelmaan käytettäväksi kutsutaan (esiopetetuksi) malliksi.

2.3 Syväoppiminen ja konvolutiiviset neuroverkot

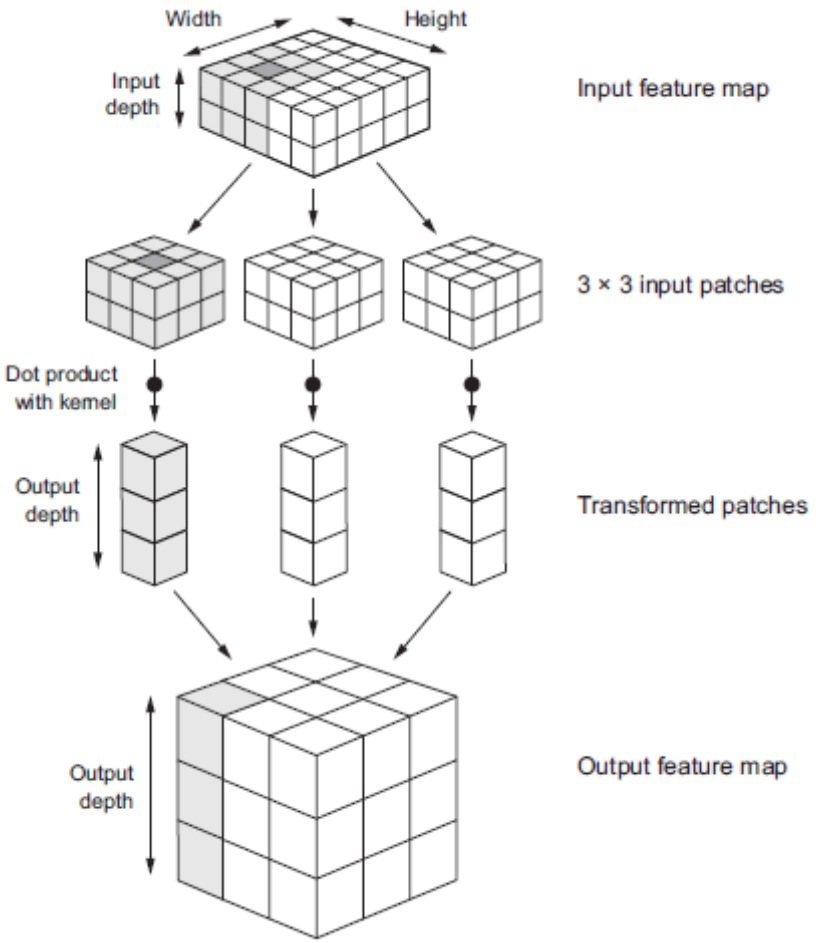
Syväoppiminen on koneoppimisen osa-alue. Syväoppimiselle tyypillistä on syvien, monista kerroksista koostuvien neuroverkkojen sekä suurten datamäärien käyttö haastavien koneoppimistehtävien suorittamiseksi. Lisäksi se vaatii huomattavasti vähemmän feature engineering -alustelua eli syötedatan manuaalista käsittelyä muotoon, jossa neuroverkko oppii haluttuja piirteitä mahdollisimman hyvin. Syväoppimisen tehokkuus perustuu siihen, että peräkkäiset verkon kerrokset kukin tunnistavat datasta tiettyjä yksinkertaisia matemaattisia representaatioita, joiden yhdistelmät voivat kuvata hyvinkin monimutkaisia todellisuudessa esiintyviä asioita. (1, s. 8–9, 18.) Syväoppimiselle tärkeitä käsitteitä ovat konvolutiiviset neuroverkot ja vastavirta-algoritmi, joiden käyttö on tullut yhä yleisemmäksi laskennan tehojen sekä tarjolla olevien datamäärien kasvaessa (1, s. 20).

Konvolutiivinen neuroverkko tarkoittaa syväoppimisessa käytettävää neuroverkkoa, jonka kerrokset oppivat tunnistamaan datassa tiettyjä piirteitä. Alemmat kerrokset oppivat yksinkertaisempiä ja yleisempiä piirteitä, kuten kuvassa esiintyviä reunoja ja tasoja, ja myöhemmät yhä erikoistuneempia piirteitä, jotka ovat yhdistelmiä aiempien tasojen oppimista piirteistä (kuva 2.) Verkko oppii nämä piirteet translaatioinvariantisti eli piirteiden paikasta kuvassa riippumatta ja pystyy oppimaan piirteiden avaruudellisia suhteita. (1, s. 123.)



KUVA 2. Konvolutiivisen neuroverkon oppimia eritasoisia piirteitä (7)

Konvolutiivinen neuroverkko koostuu vaihtelevasti konvolutiivisista ja yhdistävistä (pooling) kerroksista. Nämä kerrokset yhdessä tekevät datalle matemaattisia muunnoksia, jotka vähän kerrallaan vähentävät käsiteltävän datan määrää, esim. kuvan resoluutiota, niin että kaikkein merkittävimmät piirteet säilyvät ja enemminkin tehostuvat. Tämä parantaa verkon laskennallista tehoa vähentämättä olennaisen datan määrää. Käytännössä tämä tapahtuu tuottamalla datasta piirrekarttoja (feature maps), joiden korkeus ja leveys pienenee oppimisprosessin aikana, kun taas niiden syvyys kasvaa (kuva 3). Piirrekarttojen syvyys kuvaa datasta opittujen piirteiden määrää. (1, s. 123–124.)



KUVA 3. Piirrekarttojen tuottaminen (1, s. 125)

2.4 Siirto-oppiminen

Konvolutiivisen neuroverkon pääosa toimii piirteiden poimijana (feature extractor), ja sen viimeinen kerros on luokittelija (classifier). Luokittelijakerroksen poistamalla piirteiden poimijaa voidaan uudelleenkäyttää eri tehtävässä. Lisäksi koska piirteiden poimijan alemmat kerrokset tunnistavat yksinkertaisempia ja paremmin yleistettäviä piirteitä, tällaista mallia on mahdollista hienosäätää uutta tehtävää varten uudelleenopettamalla sen ylempiä kerroksia uuden luokittelijan lisäksi, jolloin näiden kerrosten painoarvot muuttuvat, mikä parantaa mallin tehokkuutta uudessa tehtävässä. Olemassaolevien mallien uudelleenkäyttö eri tarkoituksiin eli siirto-oppiminen (transfer learning) on käytännöllistä, koska täysin uuden neuroverkon opettaminen on laskennallisesti haastavaa ja vaatii suuria datamääriä. Lisäksi

suurellakin tehtävään tarkoitetulla datamäärällä opetettu täysin uusi malli on luultavasti vähemmän tarkka kuin erittäin suurella datamäärällä opetettu olemassa oleva malli, joka uudelleenopetetaan siirto-oppimisella. (1, s. 142–143, 8.)

3 KÄYTETYT TYÖKALUT

Työhön kuului Android-ohjelmointia sekä neuroverkkomallin uudelleenopetusta. Nämä tapahtuivat Windows 10 -pöytäkonetta käyttäen Android Studio -ohjelmointiympäristöllä sekä datatieteen ja koneoppimisen tarkoituksiin luodulla Google Colaboratory -sivustolla. Google Colab mahdollistaa Python-koodin kirjoittamisen internetselaimessa ja sen ajamisen Googlen pilvipalvelun resursseja hyväksikäyttäen. Alvin-sovellusta testattiin Huawei P30 -älypuhelimella.

3.1 Alvin AI Assistant

Alvin on tämän työn teettäjän, tmi Tommi Olavi Mänttärin kehittämä Android-pohjainen symbolinen tekoäly. Alvin on ohjelmoitu avoimella EnglishScript-kielellä ja käyttäjä voi ohjelmoida sitä tosiaikaisesti luonnollisella englannin kielellä. Alvin oppii nimiä, käsitteitä ja logiikkaa, ja sille voi antaa yksinkertaisia käskyjä puhuen tai tekstinä. Sovellus on suunniteltu esimerkiksi henkilökohtaiseksi avustajaksi tai opetuskäyttöön.

Opinnäytetyö liittyy Alvinille kehiteltävän kauko-ohjattavan robottikäden ohjaustoiminnallisuuteen. Robottiprojekti on tarkoitettu koulujen opetustarpeisiin. Projektin tarkoituksena on kehittää yksinkertainen ja halpa robottikäsi, jota on mahdollista ohjata Alvinin avulla ja jonka koululaiset voivat itse rakentaa käsityötunneilla. Hahmontunnistuksen avulla Alvin pystyy tunnistamaan robotin lähellä olevat esineet.

3.2 Keras ja TensorFlow

Keras on Python-kielellä kirjoitettu helppokäyttöinen koneoppimiseen tarkoitettu ohjelmointirajapinta. Kerasta käyttävät TensorFlow-, Theano- ja CNTK-ohjelmistokehykset. Se on suunniteltu olemaan helposti ymmärrettävä ja modulaarinen, ja se toimii rajapintana ensisijaisesti TensorFlow'lle. (9.)

TensorFlow on Googlen kehittämä avoin kirjasto koneoppimiselle. TensorFlow-sovelluksia voi ajaa tietokoneilla, pilvipalveluissa sekä Android- ja iOS-laitteilla. TensorFlow tukee Nvidia-näytönohjaimen käyttöä prosessorin sijasta, mikä

parantaa koneoppimisessa käytettävien algoritmien laskutehoa huomattavasti. Mobiililaitteille optimoitu TensorFlow-versio on TensorFlow Lite, jota käytettiin tässä työssä. TensorFlow Lite käyttää pienempään kokoon pakattuja .tflite-muotoisia malleja tavallisten TensorFlow-formaattien sijasta. (10, 11.)

Keras ja TensorFlow valittiin tähän työhön niiden yleisyyden, helppokäyttöisyyden ja Android-tuen vuoksi.

3.3 Neuroverkkomallit

Työssä käytettiin esiopetettua TensorFlow-esimerkkiohjelmaan kuuluvaa COCO SSD MobileNet v1 -mallia sekä uudelleenopetettua Kerasin kirjastoon kuuluvaa MobileNetV2-mallia. Molemmat ovat mobiililaitteille optimoituja kuvantunnistukseen luotuja konvolutiivisia neuroverkkomalleja. Ensimmäinen malli oli esiopetettu 80 luokkaa käsittävällä COCO-datasetillä ja toinen malli 1000 luokkaa käsittävällä ImageNet-datasetillä. Toinen malli uudelleenopetettiin 13 luokan räätälöidyllä datasetillä.

4 TYÖN TOTEUTUS

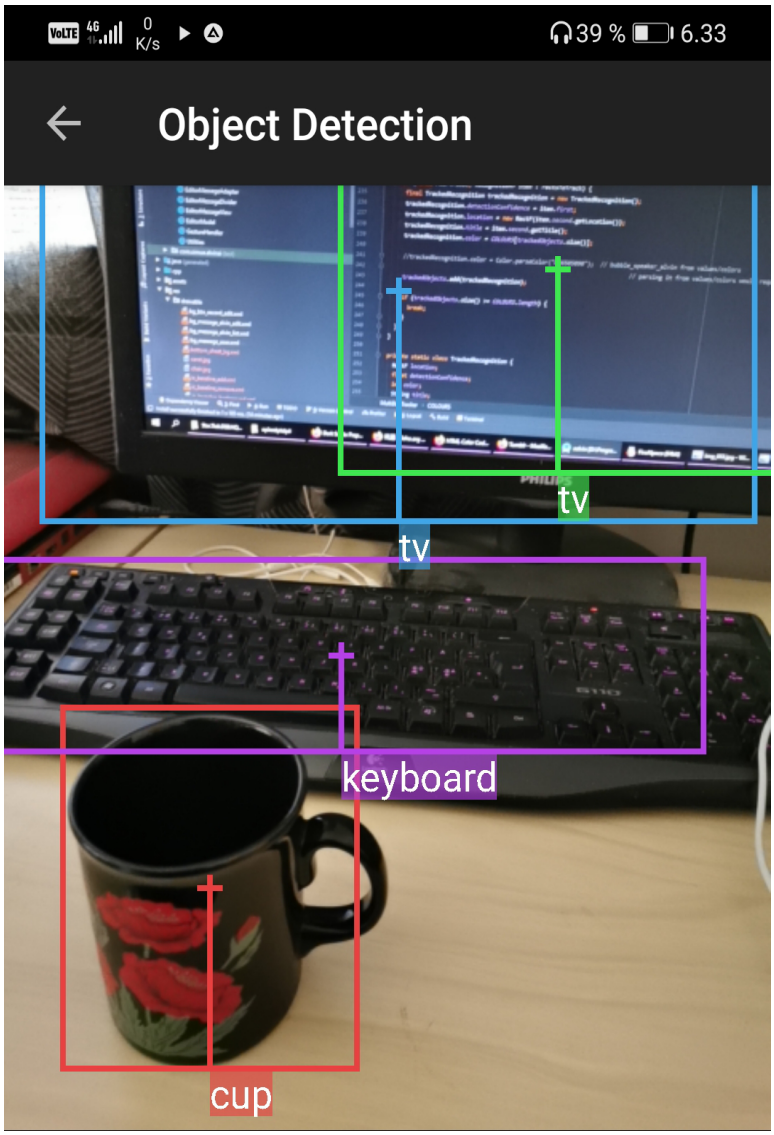
Työn tavoitteena oli lisätä Alvin-sovellukseen uutena aktiviteettina aukeava hahmontunnistusnäkymä, jossa kameran tuottaman videokuvan päällä näkyy tunnistettujen esineiden rajaavat suorakaiteet, esineen keskikohtaa osoittava risti ja esineen nimi englanniksi. Tunnistettavien esineiden tuli olla yleisiä kouluista löytyviä esineitä, kuten pyyhekumi tai matkapuhelin, ja tarpeeksi pieniä, jotta Alvinin ohjaama robotti pystyisi niitä mahdollisesti nostamaan. Työn kuluessa yrityksen työnohjaajaan oltiin yhteyksissä WhatsAppin kautta ja kahdessa kasvokkaisessa palaverissa mutta työ oli lähes kokonaan itsenäistä.

4.1 Työn alku

Työ alkoi teorian opiskeluosuudella, koska tekijällä ei ollut aiempaa kokemusta koneoppimisesta tai neuroverkoista. Tämän lisäksi tutkittiin TensorFlow'n toimintaa sekä olemassaolevia TensorFlow Lite -esimerkkiohjelmia, joiden pohjalta hahmontunnistus voitaisiin toteuttaa. Valittu esimerkkiohjelma käytti COCO SSD MobileNet v1 -mallia, joka tunnistaa 80 luokkaa mukaanlukien eläimiä, ajoneuvoja, hedelmiä ja elektronisia laitteita. Näistä tähän työhön sopivia luokkia oli vain 12.

4.2 Hahmontunnistuksen toteutus

Alvinin lähdekoodi ladattiin työn teettäjän palvelimelta. Koska hahmontunnistusnäkymä tuli muusta Alvinin käyttöliittymästä ja toiminnallisuudesta erilliseksi osaksi, sen toteuttaminen esimerkkiohjelman pohjalta oli varsin suoraviivaista. Tähän kuului esimerkkiohjelmasta käytetyn koodin vanhentuneiden osien päivittäminen ja tarpeettomien osien karsiminen sekä käyttöliittymän muokkaaminen (kuva 4).



KUVA 4. Hahmontunnistusnäkymä Alvinissa

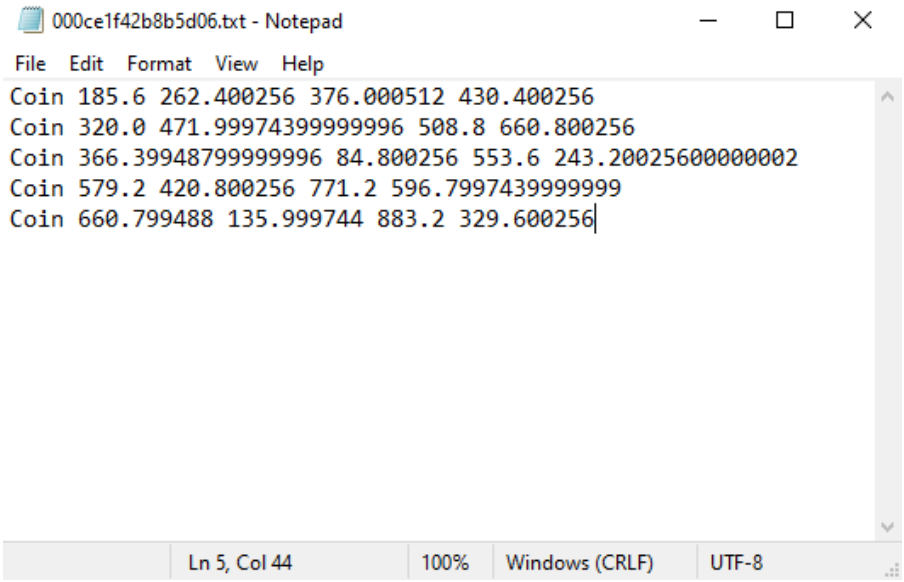
Alviniin toteutettu hahmontunnistusosuus koostui useista Java-luokista jotka avaavat laitteen kameran ja käsittelevät siitä saatuja kuvia sekä piirtävät prosessista saadun datan näytölle. .tflite-muotoinen malli sekä sen tunnistamat luokat listaava tekstitiedosto ovat sovelluksen assets-kansiossa. TensorFlow Lite -kehyksen käyttämä koodi ja pienikokoinen neuroverkkomalli takaavat hahmontunnistuksen korkean päivitystiheyden, keskimäärin noin 30 kertaa sekunnissa. Toteutus mahdollistaa useiden säikeiden käytön, mutta koska tällä ei ollut testatessa huomattavaa merkitystä, monisäikeisyys poistettiin mahdollisten yhteensopivuusongelmien ehkäisemiseksi.

Sovelluksen päänäkymästä aukeavaan pudotusvalikkoon lisättiin painike, joka avaa DetectorActivity-aktiviteetin, jossa on hahmontunnistusnäkymän graafinen osuus. Tämä aktiviteetti myös luo TFLiteObjectDetectionAPIModel-luokan määrittelemän tulkin (Interpreter), joka käyttää sovellukseen liitettyä neuroverkkomallia hahmontunnistuksen toteutukseen kun tulkkiin syötetään kameralta saatu kuva. MultiBoxTracker-luokka piirtää esineiden rajaavat suorakaiteet ja pitää niistä lukua. Muut luokat käsittelevät kameraa ja muita piirto-operaatioita näytölle.

4.3 Toisen mallin uudelleenopetus

Koska valmiin mallin tunnistamat esineet eivät kattaneet tarpeeksi haluttuja esineluokkia, päätettiin kouluttaa uusi malli käyttäen siirto-opetusta. Se päätettiin liittää olemassa olevaan malliin, jolloin uutta mallia ei tarvitsisi opettaa niillä luokilla jotka aiempi malli jo tunnisti. Oli kuitenkin epävarmaa miten mallien yhteenliittäminen onnistuisi.

Uudelleenopetettavaksi malliksi valittiin Kerasin kirjastoon kuuluva MobileNetV2, joka on kevyt mobiilisovelluksiin tarkoitettu malli ja joka on esiopetettu suurella datasetillä, eli se toimii tehokkaana piirteiden irrottajana. Uuden mallin opetusta varten tarvittiin kuvadataa, mieluiten useita satoja kuvia jokaista esineluokkaa kohden. Hahmontunnistuksen tapauksessa jokaiselle kuvalle vaaditaan kuvan sisältämän esineen tai esineiden nimeämisen lisäksi esineiden rajaavien suorakaiteiden koordinaatit kuvassa. Käytännössä siis jokaista kuvaa vastaa tekstitiedosto, joka sisältää tämän annotaatiodatan (kuva 5). Kuvien kommentointi näillä tiedoilla käsin olisi ollut liian työlästä, joten päätettiin ladata sopiva kuvadata Googlen Open Images Datasetistä, johon kuuluu valmiiksi luokiteltuja kuvia 600 esineelle hahmontunnistusopetusta varten. Näistä valittiin sopivat luokat joita valmis malli ei vielä tunnistanut (13 luokkaa). Niitä vastaavat kuvat ladattiin OIDv4 ToolKit -työkalun avulla, joka on nimenomaan Open Images Datasetin kuvien lataukseen luotu komentorivityökalu.



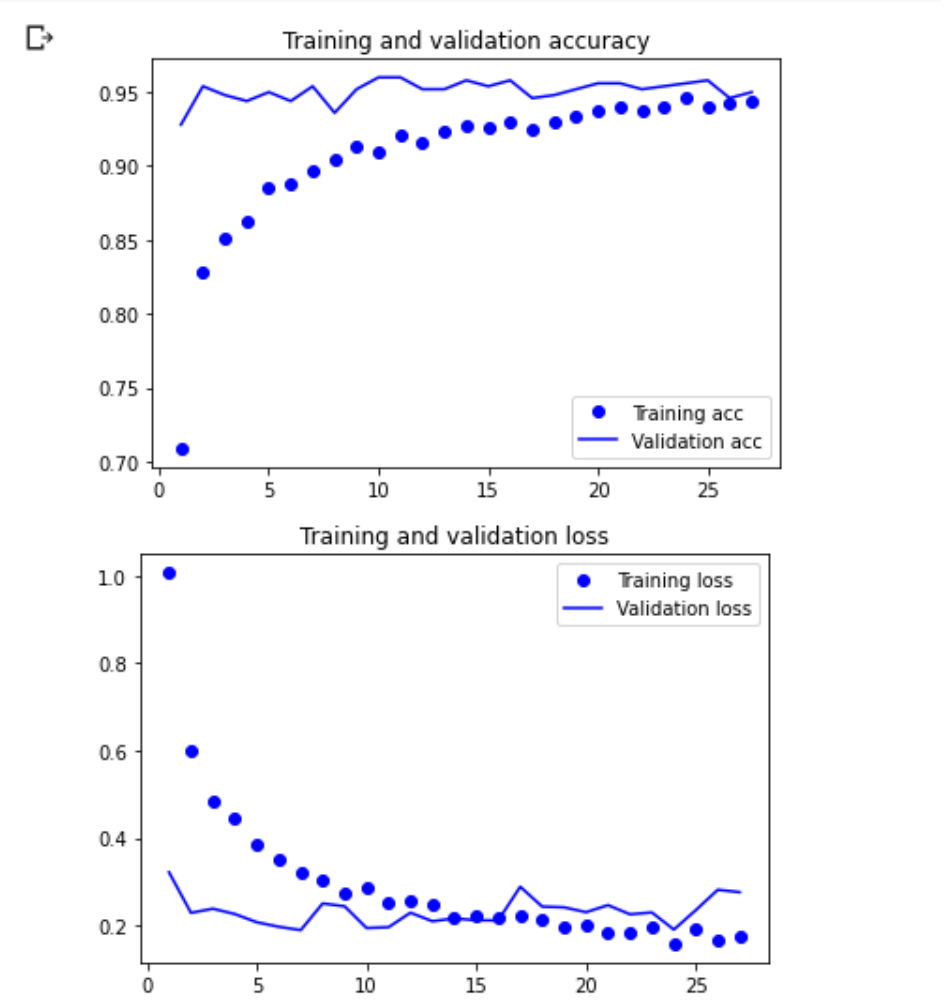
KUVA 5. Erään opetusdatakuvan annotaatiodatan sisältävä tekstitiedosto

Syötedata vaati lisäksi valmistelua ennen opetuksen alkua. Kuvien annotaatiodata oli muutettava TXT-muodosta XML-tiedostoiksi, jotka oli muutettava CSV-tiedostoiksi, joiden pohjalta oli luotava TFRecord-tiedosto. Tähän käytettiin internetistä löydettyjä ja työhön sopivaksi muokattuja Python-skriptejä.

Mallin uudelleenopetusta yritettiin Windowsin komentoriviltä, mikä vaati TensorFlow'n vanhemman version (1.15) asennusta ja oli ongelmallista puuttuvien tai yhteensopimattomien kirjastojen takia. Kelvollisen mallin opetus kesti noin kuusi tuntia koska käytetyssä tietokoneessa ei ollut Nvidia-näytönohjainta. Kun mallia yritettiin integroida Alviniin, se aiheutti yhteensopivuusvirheen Android Studiossa. Näiden ongelmien takia mallin uudelleenopetus päätettiin toteutettaa Google Colab -palvelussa.

Colab-notebookissa kirjoitettiin Python-ohjelma, joka hakee kuvadatan Google Drive -kansiosta ja toteuttaa mallin uudelleenopetuksen Googlen pilviresursseilla (liite 1). Tässä opetustavassa oli virheenä se, että esineiden koordinaatteja kuvissa ei voitu ottaa huomioon, mikä luultavasti heikensi mallin tunnistuskykyä. Malli opetettiin 300 x 300 pikselin kokoisilla kuvilla, mikä vastasi jo käytössä olevaa mallia. Opetukseen käytettiin yhteensä 5353 kuvan sarjaa 13 luokalle, ja testaukseen 520 kuvaa. Opetuksessa käytettiin data-

augmentaatiota, eli ohjelma vääristeli opetuskuvia satunnaisesti esimerkiksi kääntämällä niitä peilikuviksi, venyttämällä tai rajaamalla niitä. Näin syötedatan monipuolisuus kasvoi kuvamäärää kasvattamatta, mikä vähensi ylisovittamisen vaaraa. Opetus tapahtui 27 epookissa 107 askeleella jotka koostuivat 50 kuvan eristä (kuva 6). Epookki tarkoittaa jaksoa jossa koko opetusdata käydään läpi kerran. 27 epookin havaittiin olevan raja, jonka jälkeen verkko alkoi ylisovittamaan opetusdataan. Opetuksen jälkeen malli käännettiin TFLite-muotoon ja lisättiin Alvin-sovelluksen assets-kansioon.



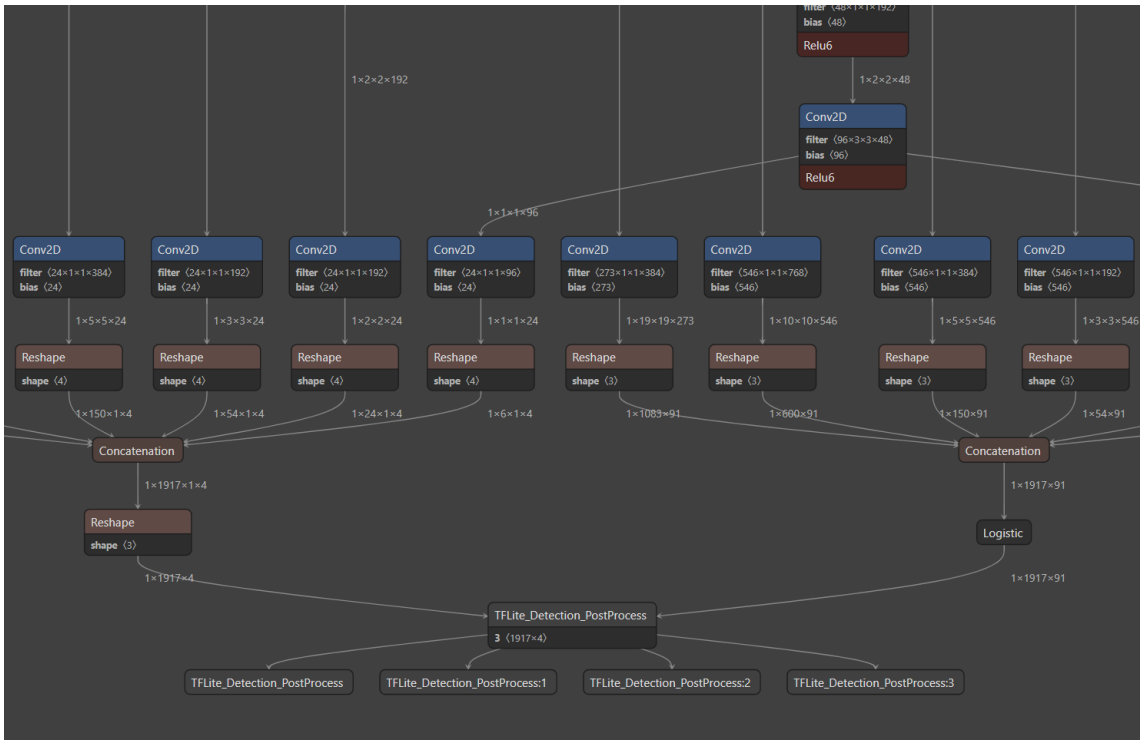
KUVA 6. Google Colabissa tuotettu mallin tarkkuuden muutosta opetuksen aikana kuvaava kaavio

Mallin viimeisen tason aktivaatiofunktioksi määriteltiin ensiksi virheellisesti softmax mutta tämä aiheutti ongelmia Android-sovelluksessa. Aktivaatiofunktio tekee tasosta suodattimen, joka tuottaa tietynlaisia epälineaarisia tulosarvoja.

Aktivaatiofunktiot ovat tärkeitä syvissä neuroverkoissa, sillä ne mahdollistavat datan käsittelyssä suuremman matemaattisen monipuolisuuden (1, s. 72). Viimeiselle tasolle määritelty softmax-funktio tuottaa todennäköisyysarvon jokaiselle luokalle eli todennäköisyysjakauman, jossa korkeinta arvoa korostetaan ja muut arvot laskevat lähelle nollaa, niin että tulostearvojen summaksi tulee yksi. Tästä seurasi että malli tuotti aina noin 99 %:n todennäköisyysarvon yhdelle luokalle, kun taas esiopetettu malli tuotti korkeintaan noin 50–80 %:n todennäköisyyksiä sen tunnistamille esineille. Siten uudelleenopetettu malli yliajoi esiopetetun mallin tulokset riippumatta siitä, mitä esineitä kamera kuvasi. Tämä korjattiin opettamalla malli uudelleen käyttäen sigmoid-aktivaatiofunktiota, joka tuottaa samoin arvoja nollan ja yhden välillä, kärjistäen niitä lähempää ääripäätä kohden. Tämä funktio ei kuitenkaan tuota arvoja niin, että niiden summa olisi yksi, joten korkein arvo ei ole välttämättä kohtuuttoman korkea. Myös relu-funktiota sekä aktivaatiofunktion pois jättämistä kokeiltiin mutta niiden seurauksena mallin tarkkuus laski käyttökelvottomaksi.

4.4 Uudelleenopetetun mallin integrointi

Hahmontunnistus Android-sovelluksessa toimii siten, että alkuperäisen neuroverkkomallin pohjalta luodulle TFLite-tulkille syötetään sopivaan kokoon muokattu kameralta saatu kuva, josta se tunnistaa korkeintaan kymmenen esineen todennäköiset koordinaatit eli rajaavan suorakaiteen sivut, luokat, luokkien todennäköisyydet sekä havaittujen esineiden kokonaislukumäärän (kuva 7). Uudelleenkoulutettu malli oli rakenteeltaan erilainen ja tulosti vain kolmetoista todennäköisyyttä joista jokainen vastaa yhtä luokkaa. Tällä mallilla oli mahdollista ajaa inferenssiä (hahmontunnistusta kuvalle) vain kerran, niin että syötedatana on koko kameran kuva.



KUVA 7. Esiopetetun mallin viimeisten tasojen rakenne

Kun mallit oli integroitu toimimaan yhdessä, sovellusta testattaessa sen todettiin olevan epätarkka useille luokille, eli hahmontunnistus ei tunnistanut luotettavasti joitakin luokkia. Syyksi havaittiin puutteellinen kuvadata mallin opetuksen validaatiovaiheessa, johtuen tavasta, jolla opetusdata oli järjestelty Colabia varten, ja joidenkin luokkien yleensäkin vähäisestä kuvamäärästä. Koska Colabissa käytetty opetusdata ei hyväksikäytä kuvien annotaatiodataa, kuvamäärää lisättiin lataamalla Open Images Datasetistä annotoimattomia kuvia ja mallin uudelleenopetus tehtiin uudelleen 5360 opetuskuvalla ja 839 validaatiokuvalla. Uuden mallin tarkkuus oli jonkin verran parempi.

Alvinin koodissa mallien pohjalta luodaan Interpreter-luokan tulkit (koodissa tfLite ja tfLite2). Alkuperäisen mallin tulkkia ajetaan kerran runForMultipleInputsOutputs()-funktiolla, jolloin se tunnistaa esineet jokaiselle havaitsemalleen esineen mahdollisesti sisältävälle rajaavalle suorakaiteelle. Uudelleenopetettua mallia sen sijaan ajetaan vain kerran run()-funktiolla sen yksinkertaisemman rakenteen vuoksi, jolloin se tunnistaa vain yhden esineen kerrallaan koko kameran kuvasta. Tästä johtuu, että hahmontunnistus voi tunnistaa useita alkuperäisen mallin tuntemia esineitä yhtä aikaa, tai vain yhden

uudelleenopetetun mallin tunteman esineen. Ensimmäisen mallin tuottamia rajaavien suorakaiteiden koordinaatteja ei voi käyttää hyväksi toisen mallin kohdalla syöttämällä koordinaattien määrittelemiä kuvan osia toisen mallin tulkin syötedataksi, koska syötedatan on oltava täsmälleen tietyn kokoinen ByteBuffer (koodissa imgData2). (Kuva 8.)

```
Trace.beginSection( sectionName: "run");
tfLite.runForMultipleInputsOutputs(inputArray, outputMap);
tfLite2.run(imgData2, prediction);
Trace.endSection();

float highestPrediction = 0f;
int classTitle = 0;
String customTitle = "";
// looping through custom model's output scores, selecting highest
for(int x = 0; x < 13; x++) {...}
final ArrayList<Recognition> recognitions = new ArrayList<>(NUM_DETECTIONS);
for (int i = 0; i < NUM_DETECTIONS; ++i) {
  final RectF detection =
          new RectF(
                  left: outputLocations[0][i][1] * inputSize,
                  top: outputLocations[0][i][0] * inputSize,
                  right: outputLocations[0][i][3] * inputSize,
                  bottom: outputLocations[0][i][2] * inputSize);
  String title = labels.get((int) outputClasses[0][i] + 1); // original model's labels
  float confidenceScore = outputScores[0][i];  // score from original model

  if(highestPrediction > confidenceScore && confidenceScore > 0.5f){
    // prediction from custom model overwrites prediction from original
    confidenceScore = highestPrediction;
    title = customTitle;
    recognitions.add(new Recognition( id: "" + i, title, confidenceScore, detection));
    i = NUM_DETECTIONS;
  }
  else {
    recognitions.add(new Recognition( id: "" + i, title, confidenceScore, detection));
  }
```

KUVA 8. Mallien yhteensovittaminen Android-sovelluksen koodissa

4.5 Lopputulos

Lopullisessa toteutuksessa sovellus tunnistaa esineitä kohtuullisen hyvin, mutta mallien erilaisten rakenteiden takia uudelleenopetetun mallin toiminta on puutteellista. Alkuperäinen malli voi tunnistaa useita esineitä yhtäaikaisesti,

mutta uusi malli vain yhden esineen kerrallaan, ja mallit eivät voi tunnistaa esineitä yhtäaikaisesti. Muuten hahmontunnistus toimii ongelmitta hyvällä vasteajalla.

4.6 Jatkokehitys

Alvinin hahmontunnistusta on mahdollista parantaa jatkossa uudelleenopettamalla uusi malli, joka vastaa esimerkkisovelluksen mallin rakennetta, sekä uudelleenopetuksen suorittaminen tavalla, joka ottaa huomioon opetuskuvien annotaatiodatan ja hyväksikäyttää lisäksi neuroverkon hienosäätöä. Vielä parempaa olisi opettaa yksi tällainen malli, joka korvaa molemmat nyt käytössä olevat mallit.

Valmiin työn pohjalta voidaan jatkaa muuhun jatkokehitykseen, kuten kehittämään toiminnallisuus Alvinin ohjaamiseen hahmontunnistusnäkymässä ja laitteen orientaation havaitsemiseen sekä sovelluksen testaamiseen useilla laitteilla. Hahmontunnistuksen toteutus on muusta sovelluksen toiminnallisuudesta erillinen ja helposti muokattavissa.

5 YHTEENVETO

Opinnäytetyön tavoitteena oli kehittää Alvin-sovellukseen hahmontunnistusominaisuus, joka tunnistaa kouluista yleisesti löytyviä esineitä ja jota voi käyttää robottia ohjaavan toiminnallisuuden pohjana. Tämä tavoite saavutettiin, tosin käytetty menetelmä, jossa kaksi neuroverkkomallia yhdistettiin sovelluksen koodissa, ei lopulta toiminut täydellisesti. Mallien yhdistäminen oli periaatteessa hyvä mutta riskialtis idea, sillä sen käytännön toteutuksen mahdollisuudesta ei ollut varmuutta.

Neuroverkkomallin opetus osoittautui odotettua vaikeammaksi siihen liittyvien opetusmateriaalien määrästä huolimatta. Aihe on syvällinen ja täysin tähän työhön sopivan mallin opettaminen vaatisi vielä enemmän siihen perehtymistä. Hahmontunnistus on tavallisesta kuvantunnistuksesta vain hiukan poikkeava osa-alue, mutta hahmontunnistukseen sopivan neuroverkkomallin luominen on vasta-alkajalle huomattavasti vaikeampaa pelkkään kuvantunnistukseen verrattuna. TensorFlow’n käyttö aiheutti myös odotettua enemmän teknisiä ongelmia. Työn aikana opitut asiat tarjosivat hyvän pohjan koneoppimisen laajemmalle opiskelulle.

Työn lopputuloksena saatiin aikaiseksi kohtuullisen hyvin toimiva Alviniin integroitu hahmontunnistus, jonka puutteet voidaan korjata korvaamalla olemassaolevat mallit ja jota on helppo kehittää eteenpäin olemassaolevien suunnitelmien pohjalta.

LÄHTEET

1. Chollet, Francois 2017. Deep Learning with Python. New York: Manning Publications Co.

2. Koneoppiminen. Ite wiki. Saatavissa: https://www.itewiki.fi/opas/koneoppiminen/. Hakupäivä 15.5.2020.

3. Brownlee, Jason 2019. A Gentle Introduction to Object Recognition With Deep Learning. Machine Learning Mastery. Saatavissa: https://machinelearningmastery.com/object-recognition-with-deep-learning/. Hakupäivä 16.5.2020.

4. Sachan, Ankit. Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD. CV-Tricks.com. Saatavissa: https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/. Hakupäivä 18.5.2020.

5. Hui, Jonathan 2018. What do we learn from single shot object detectors (SSD, YOLOv3), FPN & Focal loss (RetinaNet)? Medium. Saatavissa: https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d. Hakupäivä 19.5.2020.

6. Hardesty, Larry 2017. Explained: Neural networks. MIT News. Saatavissa: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. Hakupäivä 19.5.2020.

7. Stewart, Matthew 2019. Simple Introduction to Convolutional Neural Networks. Towards Data Science. Saatavissa: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac. Hakupäivä 19.5.2020.

8. Sarkar, Dipanjan 2018. A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. Towards Data Science. Saatavissa: https://towardsdatascience.com/a-comprehensive-hands-

on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a. Hakupäivä 20.5.2020.

9. Heller, Martin 2019. What is Keras? The deep neural network API explained. InfoWorld. Saatavissa: https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html. Hakupäivä 31.7.2020.

10. Yegulalp, Serdar 2019. What is TensorFlow? The machine learning library explained. InfoWorld. Saatavissa: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html. Hakupäivä 31.7.2020.

11. TensorFlow Lite guide. TensorFlow. Saatavissa: https://www.tensorflow.org/lite/guide. Hakupäivä 31.7.2020.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential

from google.colab import drive
drive.mount('/content/drive')

base_dir = '/content/drive/My Drive/TensorFlow/Dataset/'
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(300,
300),
                                                    batch_size=50,

class_mode='categorical')

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(300,
300),
                                                  batch_size=10,

class_mode='categorical')

mobilenet = tf.keras.applications.MobileNetV2(input_shape=(300, 300,
3),
                                               include_top=False,
                                               weights='imagenet')
mobilenet.trainable = False

inputs = keras.Input(shape=(300, 300, 3))
x = mobilenet(inputs, training=False)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
outputs = keras.layers.Dense(13, activation='sigmoid')(x)
model = keras.Model(inputs, outputs)

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
```

```
              metrics=['acc'])

history = model.fit_generator(train_generator,
                              steps_per_epoch=100,
                              epochs=30,
                              validation_data=test_generator,
                              validation_steps=50)

import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

open("/content/drive/My Drive/TensorFlow/custom_model.tflite",
"wb").write(tflite_model)
```