OULUN AMMATTIKORKEAKOULU

Katja Viinamäki

KADUNNIMIÄ JA -NUMEROITA TUNNISTAVA JA PAIKANTAVA ANDROID-SOVELLUS

KADUNNIMIÄ JA -NUMEROITA TUNNISTAVA JA PAIKANTAVA ANDROID-SOVELLUS

Katja Viinamäki
Opinnäytetyö
Syksy 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Katja Viinamäki
Opinnäytetyön nimi suomeksi: Kadunnimiä ja -numeroita tunnistava ja paikantava Android-sovellus
Opinnäytetyön nimi englanniksi: Android Application for Detecting and Locating Street Names and Numbers
Työn ohjaaja: Eino Niemi
Työn valmistumislukukausi ja -vuosi: syksy 2020
Sivumäärä: 52 + 0 liitettä

Työn tavoitteina oli perehtyä mahdollisuuksiin tunnistaa erilaisia asioita konenäön avulla ja kehittää Android-sovellus, joka tunnistaa ja paikantaa kadunnimiä ja -numeroita. Sovelluksella voidaan ottaa kuva, tunnistaa siinä oleva kadunnimi ja -numero ja paikantaa ne. Sovelluksessa on mahdollisuus kuvan, tunnistus- ja paikannustietojen tallentamiseen puhelimeen, jotta kuvia ja tekstejä voidaan myöhemmin selata. Paikat saadaan näkyviin samalle kartalle.

Työn teoriaosassa selvitettiin konenäön, tekstintunnistuksen ja paikannuksen perusteita sekä vertailtiin Googlen ja Microsoftin pilvipalvelujen valmiita kirjastoja, ominaisuuksia, joilla voidaan tunnistaa erilaisia asioita omissa sovelluksissa. Työn toisessa osassa sovellus toteutettiin käyttämällä Googlen kirjastoja. Tekstintunnistukseen käytettiin Androidille tarkoitettua Firebase ML -kirjastoa pilvessä tapahtuvaan tunnistukseen. Paikannus ja sovelluksen kartat toteutettiin Googlen paikannus- ja karttakirjastoilla. Sovellus tehtiin Android Studio -ohjelmalla ja ohjelmointikielenä oli Java. Sovelluksessa käytettiin erilaisia komponentteja Jetpackin kirjastoista.

Vertailussa Googlen ja Microsoftin konenäön pilvipalvelujen välillä, niiden tarjonnassa ei löytynyt suurta eroa. Molempien ominaisuuksien avulla voidaan tunnistaa pitkälti samoja asioita, esimerkiksi tekstiä, kasvoja ja objekteja eri luokissa. Tehdyllä sovelluksella voidaan tunnistaa kadunnimiä ja -numeroita 97–99 %:n luotettavuudella tavanomaisista mustavalkoisista kylteistä. Fontin vaihtelu, tekstin ympäristö ja muut asiat kuvassa sekä kuvanotto-olosuhteet voivat kuitenkin aiheuttaa häiriötekijöitä tunnistamiseen. Sovelluksen paikannus toimii hyvin, sen tarkkuus on 30 cm:n ja muutaman metrin välillä, mikä on riittävä taso.

Asiasanat: tekstintunnistus, objektin tunnistus, paikannus, konenäkö, koneoppiminen, tekoäly

ABSTRACT

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

---

Author: Katja Viinamäki
Title of thesis: Android Application for Detecting and Locating Street Names and Numbers
Supervisor: Eino Niemi
Term and year when the thesis was submitted: Autumn 2020
Pages: 52 + 0 appendices

The goals of this thesis were to explore what things can be recognized with computer vision and to develop an application for detecting and locating text and numbers of the street sign. With the application you can take a photo of a street sign, detect text from it and get your location. Then you can save information and see photos, detected texts and places later.

In a theory part of the thesis basics of the computer vision, object detection, text detection and location were explored. Also, Google and Microsoft Cloud Vision service's features was compared. In the second part of the thesis the application was implemented with Google Firebase ML library for text detection in the cloud. Maps SDK for Android was used for showing maps and Google Location API from Google Play Services to get location of the phone. The Application was made with Android Studio and programming language was Java. Also, several Jetpack libraries was used.

In comparison between Google and Microsoft Cloud Vision services the features for detecting different things was mostly quite similar. With them you can for example detect texts, faces and objects in different categories. The developed application detects fairly reliably text and number from the ordinary black and white street sign. Reliability of the detections were 97–99 %. However, for example varying font, imaging conditions and other objects in the photo may be a challenge to detect texts and numbers right. The location in the application is working well. The accuracy of the location is between 30 cm and few meters and it can be considered good enough.

Keywords: text detection, object detection, location, computer vision, machine learning, artificial intelligence

SISÄLLYS

1 JOHDANTO

Konenäkö (Computer Vision) pyrkii matkimaan ihmissilmää. Ihminen osaa helposti tunnistaa asioita, värejä ja muotoja, mutta koneelle tämä on vaikeampi tehtävä. Vaikeus johtuu muun muassa objektien vaihtelevasta koosta ja asennosta, kuvanotto-olosuhteista ja kuvassa olevista varjoista tai heijastuksista. (Sankar – Bartoli 2018, 3.) Konenäkö selviää kuitenkin hyvin tehtävistä, jotka on rajattu tekemään tiettyä asiaa, esimerkiksi kasvontunnistusta. Konenäön tutkimuksesta ja sovelluksista hyötyvät useat alat. Sen sovelluksia on käytetty jo pitkään muun muassa valmistusteollisuuden laadunvalvonnassa, terveydenhuollossa lääketieteellisessä kuvantamisessa ja passintarkastuksessa lentokentällä (Nieminen 2018). Sovelluksilla voidaan auttaa myös huononäköisiä tai sokeita ihmisiä (Improve OCR Accuracy With Advanced Image Processing. 2020). Viimeisen kymmenen vuoden aikana konenäöstä on tullut entistä tehokkaampi ja sillä on enemmän sovelluskohteita. Sen kehitystä ovat vauhdittaneet erityisesti koneoppimisen tekniikat (Sankar – Bartoli 2018, 3.)

Tämän opinnäytetyön tutkimuskysymyksenä on, miten kadunnimiä ja -numeroita voidaan tunnistaa ja paikantaa Android-puhelimella. Teoriaosassa luodaan aluksi silmäys tekoälyn, konenäön ja koneoppimisen käsitteisiin. Tarkemmin perehdytään konenäön sovelluksista objektien ja tekstin tunnistukseen. Tavoitteena on myös selvittää, mitä erilaisia asioita voidaan tunnistaa konenäön avulla, ja perehdytään Googlen ja Microsoftin pilvipalvelujen tarjoamiin valmiisiin kirjastoihin tunnistuksen tekemiseen. Luvun lopuksi luodaan silmäys paikannuksen peruskäsitteisiin ja Googlen kartta- ja paikannuskirjastoihin.

Työn toisessa osassa tavoitteena on kehittää Android-sovellus, joka tunnistaa ja paikantaa kadunnimiä ja -numeroita. Osoite tunnistetaan sovelluksen kameralla otettavasta kuvasta ja paikannus tehdään paikantamalla sijainti, jossa kuva otetaan. Kuva, siitä tunnistettu kadunnimi ja -numero näkyvät sovelluksen sivulla ja paikka näkyy merkittynä sovelluksessa olevalla kartalla. Kuvat, tunnistettu teksti ja paikkatiedot voidaan tallentaa puhelimen tietokantaan, jotta kuvia voi myöhemmin selata ja karttaan merkityt paikat saa näkyviin samaan karttaan. Sovellus to-

teutetaan Googlen pilvipalvelun avulla, joka tarjoaa helpon tavan käyttää konenäön ja koneoppimisen tekniikoita omassa sovelluksessa. Tekstintunnistukseen käytetään pilvessä tehtävää tunnistusta ja karttoihin ja paikannuksen tekemiseksi käytetään niihin tarkoitettuja kirjastoja. Lisäksi sovelluksessa käytetään erilaisia Android-ominaisuuksia Jetpackin kirjastoista.

2 KONENÄKÖ JA PAIKANNUS

Tässä luvussa perehdytään konenäön teoriaan, jonka perusteella työn toisessa osassa toteutettavan sovelluksen ominaisuuksia arvioidaan sekä selvitetään paikannuksen peruskäsitteitä. Tavoitteena on myös selvittää, mitä asioita konenäön avulla voidaan tunnistaa, ja vertaillaan Googlen ja Microsoftin konenäön pilvipalvelujen tarjoamia ominaisuuksia. Luvussa esitellään myös sovelluksessa käytettäviä Googlen kirjastoja.

2.1 Tekoäly

Tekoäly voidaan määritellä monella tavalla. Aihetta alettiin tutkia 1950-luvulla IBM:llä opiskelijoiden kesäprojektina ja tuolloin se määriteltiin tarkoitukseksi kehittää koneita, jotka käyttäytyvät, kuten ne olisivat älykkäitä (Korpela 2019, 9; Nilsson 2012, 4). Tekoälyn avulla voidaan koneellisesti tai keinotekoisesti suorittaa samanlaisia tehtäviä kuin ihminen. Tekoäly jaetaan usein vahvaan ja heikkoon tekoälyyn. Vahva tekoäly vastaa ihmisen aivoja ja niiden kykyä käsitellä asioita. (Tiippana 2018, 13.) Heikko tekoäly taas on järjestelmä, jolla voidaan suorittaa tiettyä erikoistunutta tehtävää, kuten shakin pelaaminen tai kaavojen ratkaiseminen, ja se voi suoriutua tehtävästä paremmin kuin ihminen. Vahva tekoäly puolestaan selviäisi lähes kaikista kognitiivisista tehtävistä ihmistä paremmin. (Huimin – Yujie – Min – Hyoungseop – Seiichi 2017, 368.)

Nykyiset tekoälysovellukset ovat heikon tekoälyn sovelluksia (Huimin ym. 2017, 368). Vaikka tekoälyteknologiat ovat kehittyneet viime vuosina muun muassa tietokoneiden laskentatehon kasvaessa ja Big Datan käytön myötä, eivät ne edelleenkään käsittele kuin yhtä erikoistunutta aluetta, kuten kuvien tai äänen tunnistaminen, eivätkä pysty ihmisen aivotoimintoihin (Huimin ym. 2017, 369).

2.2 Konenäkö

Konenäköä voidaan pitää tekoälyn alatieteenä, teknologiana tai sen yhtenä perustutkimuskohteista (Brownlee 2019; Huimin ym. 2017, 372). Sen tutkimus alkoi Standfordin yliopistossa, johon oli perustettu tekoälyä tutkiva ja kehittävä labora-

torio SAIL (Stanford Artificial Intelligence Laboratory) (Nilsson 2012, 8). Konenäkö on monitieteellinen tutkimusalue, jossa käytetään tekniikoita useista insinööritieteistä ja tietotekniikan aloilta. Sillä pyritään matkimaan ihmissilmää ja sen toimintaa. (Brownlee 2019.)

Konenäön sovelluksia on käytetty jo pitkään valmistusteollisuudessa laadunvalvonnassa, terveydenhuollossa ja kulunvalvonnassa (Nieminen 2018). Viimeisen kymmenen vuoden aikana konenäön kehitys on edennyt harppauksin ja sitä voidaan käyttää entistä useammissa sovelluksissa ja useammilla aloilla kuin ennen. Konenäön sekä kaupallinen että akateeminen tutkimus on kasvanut ja siihen on syntynyt uusia teknologioita, prosesseja, malleja ja algoritmeja (Khan – Al-Habsi 2019, 1450). Koneoppimisen yhdistäminen konenäköön on auttanut ymmärtämään monimutkaisia ongelmia (Khan – Al-Habsi 2019, 1446.) ja edistänyt konenäön kehittymistä merkittävästi. Tunnettuja konenäön uusia sovelluksia ovat esimerkiksi kuskiton auto ja virtuaali- ja lisätyn todellisuuden pelit (Azizpour 2016, 6).

2.3 Koneoppiminen

Koneoppiminen tarkoittaa järjestelmien mahdollisuutta oppia ja sopeutua itsenäisesti kehitettyjen algoritmien ja analysoidun tiedon avulla (Tiippana 2018, 13). Koneoppiminen on niin ikään tekoälyn alatiede (Khan – Al-Habsi 2019, 1445), sen yksi teknologia, jota käytetään tekoälyjärjestelmissä.

Koneoppiminen voidaan jakaa ohjattuun, ohjaamattomaan ja vahvistettuun oppimiseen (Korpela 2018, 10). Näistä ohjattua oppimista käytetään muun muassa objektin tunnistuksessa (Zhao 2020, 799). Siinä opetustieto on nimettyä, esimerkiksi kun halutaan tunnistaa tietty objekti, kerätään siitä kuvia, ja liitetään niihin nimet. Sitten valitaan malli ja algoritmi, jolla koulutus tehdään ja toteutetaan se. (Azizpour 2016, 4.) Jotta tunnistus toimisi hyvin, tarvitaan usein paljon opetustietoa (Papadopoulous – Clarke – Keller – Ferrari 2014) Tällöin löydetään algoritmi, joka antaa myös uusilla tapauksilla todennäköisesti lähes oikean vastauksen. (Korpela 2019, 10). Prosessi onkin hidas, kallis ja vaatii osaamista (Khan – Al-Habsi 2019, 1445). Ohjaamattomassa oppimisessa nimiä ei anneta, vaan kone esittää riippuvuuksia, suhteita ja samankaltaisuuksia. Vahvistetussa oppimisessa

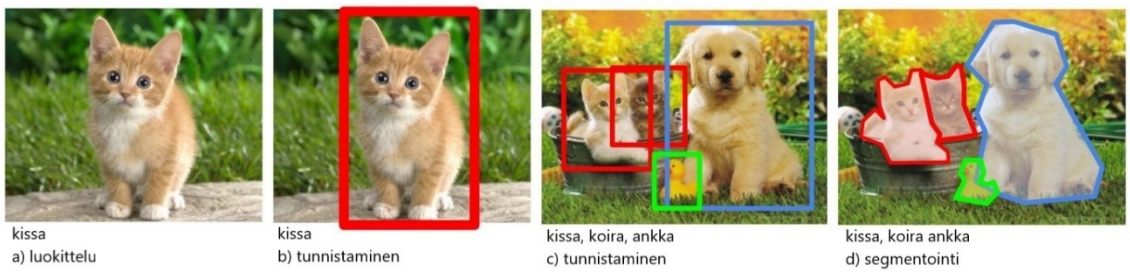
haluttu tulos saadaan antamalla algoritmille palautetta ympäristöstä. (Korpela 2018, 10.)

Konenäön sovelluksissa koneoppimista käytetään useimmin objektien luokitteluun ja tunnistamiseen tietyllä varmuudella prosentteina ilmaistuna. Vaikka edistystä on tapahtunut, on objektien tunnistaminen edelleen haasteellista. (Khan – Al-Habsi 2019, 1446.)

2.4 Objektien tunnistaminen

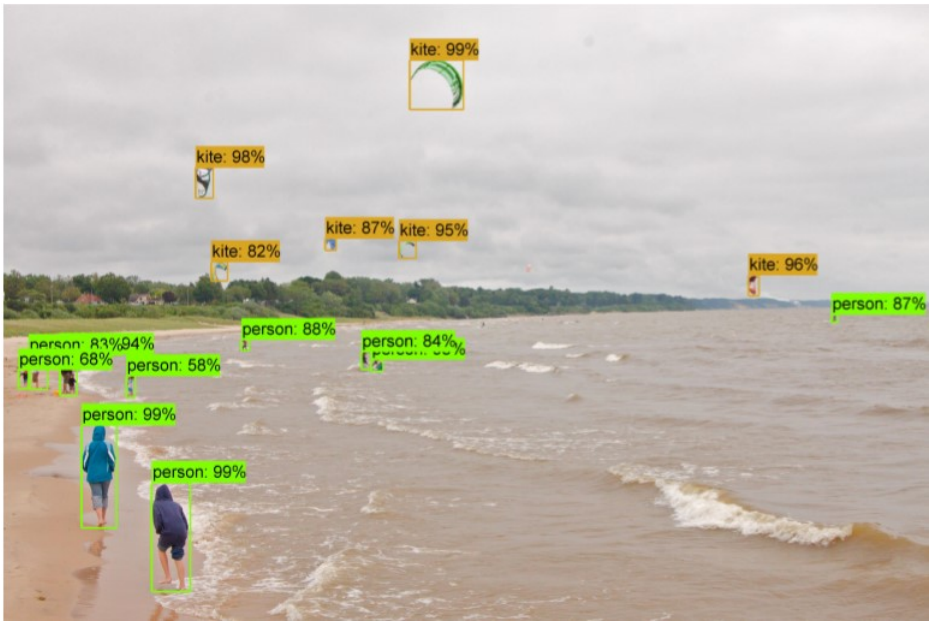
Koneiden kyky havaita visuaalista maailmaa ja erotella ja tunnistaa siitä merkityksellistä tietoa on konenäön keskeisiä tavoitteita. Tunnistuksen tuloksena saadaan kuvaus tekstinä tai numeroina syötteenä annetusta kuvasta tai videosta, kuten mitä objekteja kuvassa on tai millainen tausta niillä on. (Azizpour 2016, 3.)

Tunnistaminen voidaan jakaa objektien luokitteluun, tunnistamiseen ja segmentointiin kuvasta. Nämä tunnistamisen eri tyypit näkyvät kuvassa 1. Luokittelussa kuvasta etsitään, mitä tai mihin luokkaan kuuluvia objekteja kuvasta löytyy, esimerkiksi mikä eläin kuvassa on. Kohdassa a on tunnistettu kissa. Tunnistuksessa pyritään lisäksi paikantamaan objektien sijainti kuvassa. Se merkitään kuvaan objektin ympärille, minimikokoisella nelikulmaisella kehikolla, kuten kohdassa b. Objekteja voidaan tunnistaa useasta luokasta samaan aikaan, kuten kohdassa c. Objektit voidaan myös segmentoida kuvasta siihen kuuluvien pikselien perusteella. Tällöin kuvaan lisätään ääriviivat sen reunoja myöten, kuten näkyy kohdassa d. (Azizpour 2016, 7–8; Khan – Al-Habsi 2019, 1445.)



KUVA 1. Objektien luokittelu, tunnistaminen ja segmentointi (muokattu Ouaknine, 2018).

Tunnistettu objekti ilmaistaan usein myös tekstillä ja todennäköisyydellä tunnistuksen oikeellisuudesta eli luotettavuudesta kehikon yläpuolelle. Kuvassa 2 on tunnistettu esimerkiksi ihmisiä ja leijoja.



KUVA 2. Objektien tunnistaminen ja tunnistuksen luotettavuus Zoph – Vasudevan – Shlens – Le 2018, 14).

2.5 Googlen ja Microsoftin konenäön pilvipalvelut

Suuret teknologiayhtiöt, kuten Google ja Microsoft, ovat alkaneet tarjota pilvipalveluna mahdollisuuden käyttää tekoälyn ja koneoppimisen tekniikoita omissa sovelluksissa helposti ja vähin kustannuksin. Tekniikoita ei tarvitse itse kehittää tai osata. Taulukkoon 1 on koottu Googlen ja Microsoftin Azure pilvipalvelujen tarjoamat ominaisuudet konenäön sovelluksiin. Suurin osa ominaisuuksista on samoja. Tekstintunnistus löytyy molemmilta. Niissä käytetään OCR-teknologiaa ja tunnistuksen voi tehdä dokumenteista tai kuvista. Molemmissa on myös ominaisuus käsin kirjoitetun tekstin tunnistamiseen. (Optical Character Recognition OCR. 2020; Detect text in images. 2020.)

Molemmilla on kirjastot objektien luokitteluun (Applying content tags to images. 2019; Detect Labels. 2020) ja tunnistamiseen kuvista (Detect common objects in images. 2019; Detect multiple objects. 2020). Objekteja tunnistetaan eri luokissa,

jotka ovat samantapaisia, kuten kulkuneuvot, tavarat, kasvit ja eläimet. Molemmissa on niin ikään ominaisuudet kasvojen, nähtävyyksien, kuuluisuuksien, tunnettujen tuotemerkkien logojen, kuvien päävärien ja K-18-materiaalien tunnistamiseen. Lisäksi molemmissa palveluissa on ominaisuus kuvien rajaamiseen kuvassa olevaan pääobjektiin tai objekteihin. (Computer Vision API Documentation, 2020, 4–5; Features list. 2020.)

Molemmissa pilvipalveluissa on myös mahdollista kouluttaa omia malleja objektien tunnistamiseen, jos valmiilla kirjastoilla ei voi tunnistaa haluamiaan asioita (Custom Vision. 2019; Cloud AutoML. 2020).

Palvelujen tarjoamissa ominaisuuksissa on eroa muutamissa tapauksissa. Microsoftilla on ominaisuus, joka kuvailee kuvan sisällön kokonaisilla lauseilla perustuen siinä oleviin objekteihin. Algoritmi luo erilaisia lauseita ja palauttaa ne oikeellisuuden todennäköisyyden mukaan järjestettynä. (What is Computer Vision. 2020.) Tällaista ominaisuutta ei ole Googlella. Lisäksi vain Microsoftilla on kuvien piirrostyylin tunnistava ominaisuus. Se kertoo, onko kuva clip art-kuva vai viivapiirros (Detecting image types with Computer Vision. 2019). Googlella on sen sijaan ominaisuus, vastaavan web-sisällön tunnistaminen, jota ei ole Microsoftilla. Se tunnistaa kuvassa olevien objektien perusteella saman tai vastaavia kuvia webistä ja palauttaa sivujen osoitteet, joilla kuvia on. Ominaisuus toteutetaan Googlen kuvahaun avulla (Detect Web entities and pages. 2020.)

TAULUKKO 1. Googlen Cloud Visionin ja Microsoftin Azure Cloud Visionin tarjoamat ominaisuudet

Ominaisuus	Google	Azure
Tekstintunnistus	x	x
Objektien luokittelu	x	x
Objektien tunnistaminen	x	x
Kasvotunnistus	x	x
Nähtävyyksien tunnistaminen	x	x
Kuuluisuuksien tunnistaminen	x	x
Logojen tunnistaminen	x	x
Värien tunnistaminen	x	x
K-18	x	x
Kuvien rajaus	x	x
Kuvan sisällön kuvailu		x
Vastaavan Web sisällön tunnistaminen	x	
Piirrostyylin tunnistus		x
Omien mallien koulutus	x	x

2.6 Googlen konenäön pilvipalvelu mobiileille alustoille

Googlella on erityisesti myös mobiileille alustoille, Androidille ja iOS:lle, tarkoitettuja palveluja, joilla koneoppimisen tekniikoita voi helposti käyttää omissa konenäön sovelluksissa.

Google on kesäkuussa 2020 uudistanut ja selkiyttänyt näitä palvelujaan. Palvelut on jaettu MLKit ja Firebase ML palveluihin, sen mukaan, tapahtuuko tunnistus laitteessa vai pilvessä. MLKit SDK on uusi palvelu. Se sisältää kaikki työkalut laitteessa tapahtuvaan tunnistamiseen ja Firebase ML SDK sisältää pilvipohjaiset palvelut, tunnistamisen ja itsekoulutettujen datamallien käyttämisen. (Migration Guide. 2020; Firebase Machine Learning. 2020.) Tässä työssä käytetään pilvessä tapahtuvaa tunnistusta.

2.6.1 ML Kit

ML Kitissä on saatavilla seuraavat kirjastot: tekstintunnistus, objektien luokittelu ja tunnistaminen, kasvon- ja kasvonpiirteidentunnistus, viivakoodin lukeminen, kielen tunnistaminen, kielen kääntäminen ja fiksu vastaus. (Firebase Machine Learning. 2020).

2.6.2 Firebase ML

Firebase ML sisältää kirjastot tekstin tunnistamiseen, kuvissa olevien objektien tunnistamiseen sekä nähtävyyksien tunnistamiseen. Pilvessä tapahtuvaan tunnistamiseen tarvitaan nettiyhteys, koska kuva, josta tunnistus tehdään, pitää lähettää pilveen, jotta se voidaan tunnistaa. Pilvessä on kuitenkin käytettävänä enemmän laskentatehoa ja muistia tunnistuksen tekemiseen kuin laitteessa tapahtuvassa tunnistuksessa, ja tuloksen tarkkuuden pitäisi olla paljon tarkempi. (Firebase Machine Learning. 2020.) Lisäksi pilvitunnistusta käyttämällä saa käyttöönsä tunnistuksen luotettavuuden, tunnistetun kielen ja tekstiobjektin sijainnin kertovat ominaisuudet.

2.7 Tekstintunnistus

Tekstintunnistuksella, OCR:llä (Optical Character Recognition), tunnistetaan tekstiä kuvista. Sitä on tutkittu ja kehitetty 1950-luvulta ja se on yksi menestyneimmistä konenäön teknologioista (Trier ym. 1996, 641). Koneoppimisen tekniikat ovat edistäneet myös tekstintunnistuksen kehittymistä (Dervisevic 2006, 2). Nykyisillä OCR-järjestelmillä voidaan saavuttaa lähes 99 %:n tunnistuksen luotettavuus (Improve OCR Accuracy With Advanced Image Processing. 2020). Jotkut pitävät sitä kokonaan ratkaistuna ongelmana ainakin painetun tekstin osalta (Kavelar – Zambanini – Kampel, 1). Tekstintunnistamisen onnistumiseen vaikuttavat kuitenkin muun muassa kuvan laatu ja olosuhteet, jossa kuva otetaan (Johansson 2019, 5; Improve OCR Accuracy With Advanced Image Processing. 2020).

Perinteinen OCR-järjestelmä toimii hyvin tilanteissa, joissa tekstin ja sen taustan välillä on suuri kontrasti ja tausta on yhtenäinen, kuten paperille painetussa tekstissä tai tienviitoissa (Kavelar ym. 2013, 1). On myös käyttötapauksia, joihin perinteinen OCR-järjestelmä ei täysin sovi (Sarshogh – Hines 2019). Tekstintunnistaminen esimerkiksi ulkona, jokapäiväisessä ympäristössä otetuista kuvista on edelleen haasteellista (Veit – Matera – Neumann – Matas – Belongie 2016, 1). Haasteita voivat tuoda muun muassa taustan ja tekstin erotus, väritys, tekstin suunta, tekstin mitan ja fontin vaihtelu (Sarshogh – Hines 2019).

Aluksi tekstintunnistusta käytettiin muun muassa shekkien käsittelyn automatisoinnissa ja se tehtiin siihen tarkoitetuilla laitteilla (Malin 2010, 24). Nykyään tekstintunnistuksella voidaan digitalisoida erilaisia paperidokumentteja ja lomakkeita. Sillä voidaan pyrkiä vähentämään paperilomakkeiden määrää yrityksissä tai sen avulla voidaan esimerkiksi säilyttää vanhoja dokumentteja. Skannereissa tai tulostimissa on yleensä jokin OCR-järjestelmä. (Malin 2010, 2–3; Improve OCR Accuracy With Advanced Image Processing. 2020.) Internetissä on palveluja, joissa voi tehdä tekstien tunnistusta erilaisista asiakirjoista (Malin 2010, 36). Tekstintunnistusohjelmiston kehittäjille on myös saatavilla useita kirjastoja, esi-

merkiksi Tesseract. Se on Windows, Linux ja MacOs X käyttöjärjestelmille tarkoitettu avoimen lähdekoodin ohjelma. (Tesseract OCR. 2020) OCR-tekniikkaa käytetään myös Googlen tekstintunnistuksessa.

Googlen pilvipalvelun tekstintunnistuksella voidaan tunnistaa useita kymmeniä kieliä. Suomen kieli on yksi tuetuista kielistä. Palvelun avulla voidaan tunnistaa eri kieliä myös samasta kuvasta. (OCR Language Support. 2020.) Saatavilla on kirjastot yleiskäyttöiseen tekstintunnistukseen kuvista, ja kirjasto, joka on optimoitu tekstintunnistukseen dokumenteista (Recognize Text in Images with Firebase ML on Android. 2020). Tässä työssä käytetään kuvista tunnistusta.

2.8 Paikannus

Paikannus tarkoittaa jonkin liikkuvan kohteen, kuten ihmisen tai auton, sijainnin selvittämistä. Sijainti voidaan selvittää paikkatiedon avulla, joka on tietoa, johon liittyy maantieteellinen sijainti. (Paikannus. 2020.) Se on tietoa kohteista, joiden paikka Maan suhteen tunnetaan, ja paikannettua kohdetta kuvaava sijaintitiedon ja ominaisuustiedon looginen tietokokonaisuus. Kartoilla voidaan esittää paikkatietoa visuaalisesti. (Paikkatieto. 2019.) Kohteen sijainti voidaan ilmoittaa muun muassa koordinaateilla, asteilla, pituuksilla ja leveyksillä (Paikannus. 2020).

Ensimmäiset paikannuksen menetelmät perustuivat paikallistuntemukseen tai tähtiin ja Aurinkoon. Kompassin ja kellon keksiminen sekä karttojen laatiminen edistivät paikantamista merkittävästi. Nykyään paikannusta tehdään esimerkiksi satelliittipaikannuksella. (Paikannus. 2019.) GPS eli Global Positioning System on Yhdysvaltain puolustusministeriön kehittämä ja rahoittama satelliittipaikannusjärjestelmä, jonka virallinen nimi on Navstar GPS. Sen kehitystyö aloitettiin 1970-luvun puolivälissä ja tarkoituksena oli luoda sekä sotilas- että siviilikäyttöön soveltuva tarkka ja reaaliaikainen yksisuuntainen paikannusmenetelmä. GPS on tällä hetkellä käytetyin järjestelmä maailmanlaajuisista satelliittipaikannusjärjestelmistä. GPS-paikannus tuli autoissa suosituksi 1990-luvun lopussa ja matkapuhelinten paikannusta on tehty 2000-luvun alusta alkaen. (GPS. 2020.)

Paikkatietoa hyödyntävät niin valtio, kunnat, yritykset kuin yksityishenkilötkin. Sen soveltamismahdollisuuksia on paljon: paikkatiedon avulla voidaan esimerkiksi etsiä turvallisin ja nopein laiva- tai lentoreitti, suunnitella rakentamista ja kaavoitusta tai etsiä älypuhelimella lähin bussipysäkki. (Paikkatieto. 2019.)

2.9 Googlen kartta- ja paikannuskirjastot

Google tarjoaa mahdollisuuden käyttää karttoja omissa sovelluksissa. Googlen kartat kattavat 99 % maailmasta (Maps. 2020). Karttatyyppejä on saatavilla erilaisia, esimerkiksi tavallisia ja satelliittikuvakarttoja. Tavallisessa kartassa on merkittynä tiet, joitakin ihmisen tekemiä rakennuksia ja maastokohteet, kuten joet. Teiden nimet on myös merkitty karttaan. Karttoja voidaan räätälöidä, niihin voidaan muun muassa merkitä paikkoja, piirtää reittejä ja niistä voidaan tehdä zoomattavia. (Map Objects. 2020.) Tässä työssä käytetään tavallisia karttoja, joita räätälöidään merkitsemällä niihin paikkoja.

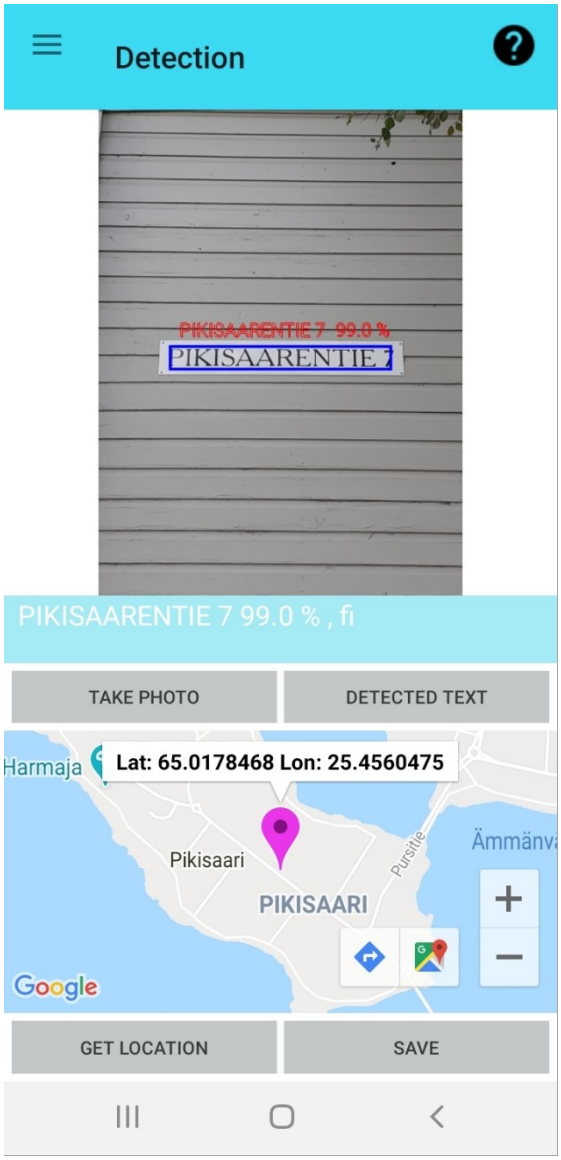
Lisäksi sovelluksiin voidaan tehdä ominaisuuksia, joissa käytetään paikannusta. Paikannuksen voi saada puhelimeen esimerkiksi kerran tai jatkuvasti sovelluksen ollessa päällä. Paikannuksen tarkkuus voi myös olla erilainen eri sovelluksissa riippuen paikannustarpeesta. Paikannus voi olla käyttäjälle näkyvää tai se voi toimia taustalla. Tässä työssä käytetään käyttäjälle näkyvää, mahdollisimman tarkkaa paikannusta ja paikannus pyydetään sovellukseen tietyin väliajoin. Sekä kartta- että paikannuskirjastot toimivat Google Play Servicen kautta, joka tulee asentaa Android Studioon. (Build location-aware apps. 2020; Request location permissions. 2020.)

3 ANDROID-SOVELLUS

Ensimmäinen Android-puhelin julkaistiin vuonna 2008 ja Android on tällä hetkellä yleisin käyttöjärjestelmä matkapuhelimissa (Harju 2013, 11–13). Opinnäytetyön tavoitteena oli tehdä kadunnimiä ja -numeroita tunnistava ja paikantava Android-sovellus. Tässä luvussa kuvaillaan, miten sovellus toimii, ja kerrotaan sen tekemisestä. Sovelluksen tehtiin Android Studion versiolla 3.6.2 ja ohjelmointikielenä oli Java. Testipuhelimena oli Samsung Galaxy A20e, jossa on Android 10. Sovelluksesta tehtiin englanninkielinen.

3.1 Sovelluksen toiminta

Sovellus koostuu viidestä sivusta. Kotisivun lisäksi siihen kuuluu kaikki kuvat (All Images) -, kaikki paikat (All Places) -, poista kaikki (Delete All) - ja ohje (Help) -sivut. Kotisivu eli etusivu (Detection) näkyy kuvassa 3. Sivulta voidaan käynnistää kamera painikkeella, ja ottaa sitten kuva halutusta katukyltistä. Kun kuva on otettu, palataan kotisivulle, johon kuva ilmestyy. Tekstintunnistus voidaan käynnistää niin ikään painikkeesta. Sillä välin, kun tunnistusta tehdään, ruudulla näkyy ProgressBar. Kun tunnistus on valmis, tunnistettu teksti, tunnistuksen luotettavuus sekä tunnistetun kielen tunnus ilmestyy tekstikenttään kuvan alapuolelle. Lisäksi kuvaan tekstin ympärille tulee sen sijainnin rajaava kehikko sekä tunnistettu teksti ja tunnistuksen luotettavuus. Kun halutaan tehdä paikannus, klikataan siihen tarkoitettua painiketta. Paikka ilmestyy hetken kuluttua sivun alaosassa olevalla kartalle purppuran värisellä paikkamerkillä merkittynä. ProgressBar näkyy ruudulla, kunnes paikannus on valmis. Kartta on liikuteltava ja sitä voidaan zoomata kartalla olevilla painikkeilla tai sormilla. Paikkamerkin infoikkunassa näkyvät paikan koordinaatit, jos merkkiä klikkaa. Sivun tiedot voidaan tallentaa tietokantaan, jotta niitä voi myöhemmin selata. Tallennukseen on oma painike sivulla.



KUVA 3. Kuvaruutukaappaus sovelluksen kotisivulta

Sovelluksessa on sivuvalikko, joka avataan klikkaamalla kuvaketta sivun yläreunassa olevan toimintopalkin vasemmassa reunassa tai pyyhkäisemällä oikealle vasemmasta reunasta. Siitä voidaan siirtyä kaikille sovelluksen sivuille. Sivuvalikosta ja toimintopalkista kerrotaan vielä tarkemmin kappaleessa 3.4. Kaikki kuvat -sivulla voidaan selata puhelimeen tallennettuja kuvia ja niistä tunnistettuja tekstejä. Kaikki paikat -sivulla näkyvät tallennetut kuvanottopaikat samalla kartalla paikkamerkeillä merkittynä. Sivuilta voidaan palata kotisivulle klikkaamalla toimintopalkin takaisin-painiketta. Kaikilta sivuilta pääsee myös toimintopalkin oikeasta yläkulmasta, kysymysmerkki kuvaketta klikkaamalla, ohje-sivulle. Sivulla kerrotaan, kuinka sovellusta käytetään. Kaikki tallennetut tiedot voidaan poistaa

puhelimesta poista kaikki -sivulla. Sovelluksessa ei ole lopeta painiketta, vaan siitä poistutaan klikkaamalla puhelimen takaisin painiketta.

3.2 Sovelluksen suunnittelu

Aluksi suunniteltiin sovelluksen tulevien sivujen määrää, sovelluksessa tarvittavia komponentteja, painikkeita ja valikoita sekä sovelluksen ulkoasua ja päätettiin käyttää Jetpackin komponentteja. Jetpack koostuu kirjastoista, joissa käytetään parhaita käytäntöjä Android-sovelluksen tekemisessä. Koodin laatu paranee ja sovellukset toimivat samanlailla eri laitteissa ja versioissa. (Android Jetpack. 2020) Käytettyjä Jetpackin kirjastoja olivat muun muassa Navigation, Room, Camera ja Navigation Drawer. Sovelluksen tekeminen aloitettiin luomalla Android Studiossa uusi projekti, TextDetectionandPlacement. Seuraavana vaiheena oli ottaa käyttöön Navigation-kirjasto.

3.3 Navigaatio

Navigointi tarkoittaa toimintoja, joiden avulla käyttäjä voi siirtyä, navigoida, katsomaan sovelluksen sisältöjä, siirtyä sisällöstä toiseen ja takaisin. Navigation-kirjaston tuoma navigointirakenne noudattaa suunnitteluperiaatteita, joilla pyritään parantamaan käyttäjäkokemusta. Se luo ohjelmalle selkeän arkkitehtuurin ja rakenteen sekä tavan toteuttaa navigointi sivujen välillä. Lisäksi se sisältää ominaisuudet käyttöliittymäkomponenttien, kuten toimintopalkin ja sivuvalikon, toteuttamiseen helposti. Muita sen etuja ovat muun muassa fragmenttien helppo käyttäminen ja aktiviteettipinon toiminnasta huolehtiminen. Tällöin paluu- ja taaksepäin -toimintojen pitäisi mennä aina oikein. (Navigation. 2020.)

Navigation-komponentissa on kolme avainosaa: navigaatiokaavio, NavHost ja NavController. Kaaviossa esitetään kaikki mahdolliset polut, joissa käyttäjä voi sovelluksessa siirtyä. Paikkaa, johon siirrytään, kutsutaan määränpääksi, joka yleensä on fragmentti tai aktiviteetti. (Navigation. 2020.) Tässä työssä käytettiin fragmentteja. Kaaviossa ovat kaikki määränpäät, ja polut niiden välillä esitetään nuolilla, joita kutsutaan toiminnoiksi (Get started with the Navigation component. 2020).

NavHost on tyhjä säiliö, jossa sovelluksen suorituksen aikana näytetään kaikki määränpäät. Se sisältää oletustoteutuksen, NavHostFragmentin, johon, kun käyttäjä navigoi esimerkiksi uudelle sivulle, vaihdetaan vanhan sivun tilalle uusi sivu. NavController on puolestaan objekti, joka hoitaa navigoinnin ja sisällön vaihtamisen NavHostissa. (Navigation. 2020.) Rakenne on suunniteltu toimimaan fragmenttien ja yhden activityn, MainActivityn, avulla (Get started with the Navigation component. 2020). Yleensä Android-sovelluksessa on useita aktiviteetteja, jotka luovat ja asettavat näkyviin halutun näkymän .xml-tiedostosta. Aktiviteetit sisältävät myös sivujen toimintalogiikan ja funktiot, mutta ne voidaan toteuttaa myös fragmenteilla. NavControllerin avulla voidaan siirtyä sivulta toiselle intent-olioiden sijaan, joilla tavallisesti käynnistetään aktiviteetit.

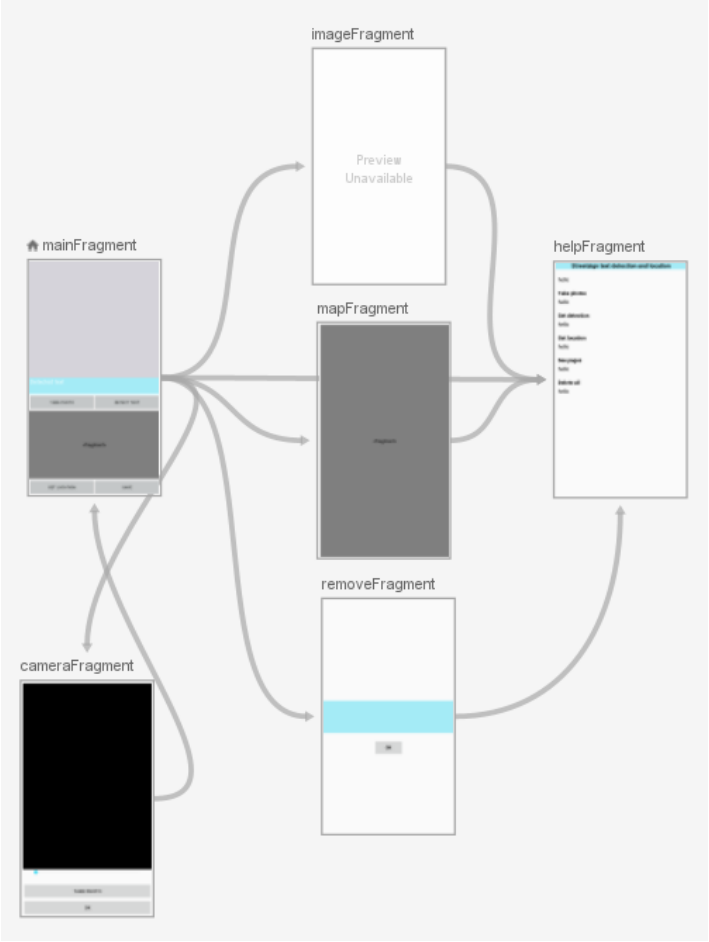
Navigation-kirjastoa varten lisättiin aluksi tarvittavat riippuvuudet sovelluksen build.gradle-tiedostoon. Tiedostossa määritellään sovelluksen konfiguraatio, ja siihen voi lisätä sovelluksessa käytettäviä kirjastoja (Configure your build.2020).

```
def nav_version = "2.3.0-rc01"
implementation "androidx.navigation:navigation-fragment:$nav_version"
implementation "androidx.navigation:navigation-ui:$nav_version"
implementation "androidx.navigation:navigation-dynamic-features-frag-
ment:$nav_version"
```

Navigaatiokaavio on .xml-tiedosto, joka luotiin lisäämällä projektiin uusi resurssihakemisto ja valitsemalla sen tyypiksi Navigation. Kaavion luomisen jälkeen sitä voitiin käyttää Android Studion -editorissa kaavion suunnittelunäkymässä. Siellä lisättiin tarvittavat fragmentit ja niiden välille toiminnot hiirellä klikkaamalla ja vetämällä. Fragmenttien ulkoasu toteutettiin myöhemmin omissa layout-tiedostoissa ja niiden toiminnallisuus .java-tiedostoissa.

Kuvassa 4 näkyy sovelluksen navigaatiokaavio. Siinä on kuusi määränpäätä, jotka ovat mainFragment, mapFragment, imageFragment, cameraFragment, removeFragment ja helpFragment. Kun sovellus käynnistetään, sen aloitussivuna ja määränpäänä on mainFragment. Siitä käyttäjä voi navigoida joko mapFragment-, imageFragment- tai removeFragment-sivulle ja palata sieltä takaisin mainFragmentiin. Kun kamera käynnistetään, siirrytään cameraFragmentiin, ja

kun kuva on otettu, palataan takaisin mainFragmentiin. HelpFragmentiin ja sieltä takaisin pääsee kaikista muista fragmenteista paitsi cameraFragmentista.



KUVA 4. Sovelluksen navigaatiokaavio

Ohjelma tekee automaattisesti navigaatiokaaviosta .xml-tiedoston, josta cameraFragmentin osa näkyy seuraavassa koodissa. Siinä ovat tiedot fragmentista ja toiminto (action). Toimintoa ja navControlleria käyttämällä toteutettiin siirtyminen kamerasta etusivulle kuvanottamisen jälkeen.

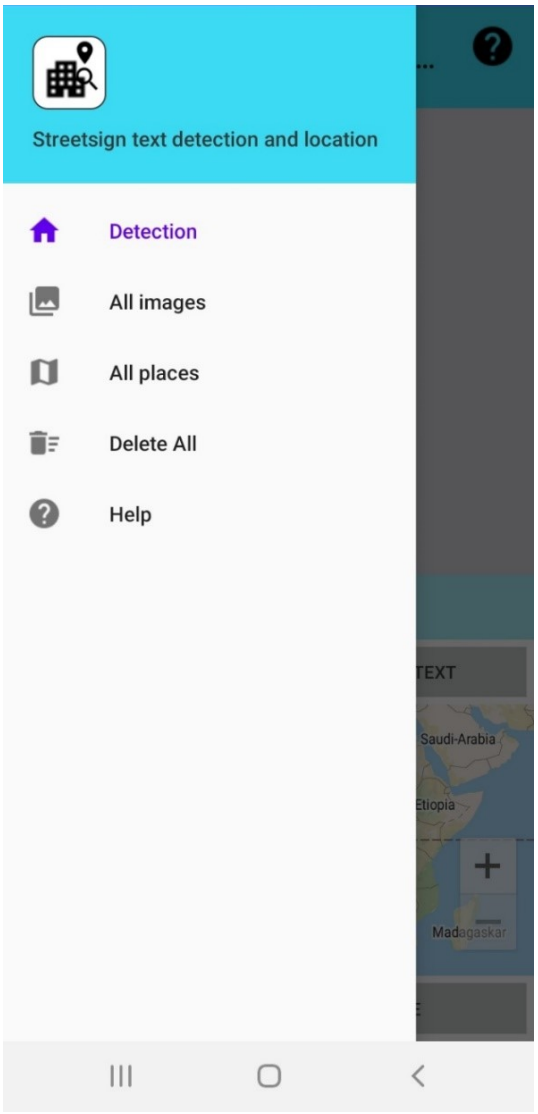
```
<fragment
    android:id="@+id/cameraFragment"
    android:name="fi.example.textrecognitionandlocation.CameraFragment"
    android:label="fragment_camera"
    tools:layout="@layout/fragment_camera" >
    <action
        android:id="@+id/action_cameraFragment_to_mainFragment"
        app:destination="@id/mainFragment" />
</fragment>
```

Navigaatiokaavio toimii sovelluksen dokumentaationa ja sen avulla sovelluksen ylläpito ja muutosten tekeminen ovat helppoja. Kun kaavioon tehdään muutoksia, esimerkiksi lisätään fragmentteja, päivittyvät ne automaattisesti .xml-tiedostoon ja jos fragmentteihin tehdään muutoksia, näkyvät ne kaaviossa.

Navigation-kirjasto sisältää ominaisuudet käyttöliittymäkomponenttien, kuten toimintopalkin ja sivuvalikon, toteuttamiseen helposti. Näiden lisäämisestä sovellukseen kerrotaan tarkemmin seuraavassa kappaleessa.

3.4 Sivuvalikko ja toimintopalkki

Sovellukseen tehtiin navigaatiovalikko eli sivuvalikko, joka näkyy kuvassa 5. Valikko avataan vasemman yläkulman kuvakkeesta tai reunasta pyyhkäisemällä. Se sisältää sovelluksen päävalikon, josta voidaan siirtyä eri sivuille. Valikko tehtiin, jotta etusivulla ei olisi niin paljon nappuloita. Valikko näytetään etusivulla. Se voidaan lisätä sovellukseen, kun on tehnyt navigaatiokaavion. Sitä varten tehtiin menu-resurssikansioon layout-tiedosto, joka sisältää valikon komponentit. Valikko yhdistettiin asetuksissa navigaatiokaavion kanssa, jolloin siirtymiset valikosta tehdään automaattisesti määränpääsivuille ja sieltä takaisin navControllerin avulla. (Update UI components with NavigationUI. 2020.)



KUVA 5. Sovelluksen sivuvalikko

Sovellukseen lisättiin myös toimintopalkki, joka on näkyvissä kaikilla sivuilla kameraa lukuun ottamatta. Siinä esitetään sivujen nimet, etusivulla sivuvalikon kuvake ja sisäsivuilla takaisin-painike vasemmassa reunassa. Oikeassa reunassa on kuvake, jota klikkaamalla pääsee ohje-sivulle. Navigation-kirjastoon kuuluu NavigationUI-luokka, jossa on funktioita, joilla voi hallita navigointia käyttöliittymäkomponenteissa, toimintopalkissa ja sivuvalikossa. Siinä on esimerkiksi objekti sivuvalikon kuvakkeen hallintaan eri sivuilla. (Update UI components with NavigationUI. 2020.)

Sovelluksen sivuille yhteiset käyttöliittymäkomponentit lisättiin main_activity.xml-tiedostoon. Se sisältää myös aiemmin mainitun NavHostFragment-komponentin,

johon eri sivujen sisältö vaihdetaan, kun sovellusta käytetään. Asettelumalliksi tarvittiin drawer-layout sivuvalikon käyttämiseksi (Update UI components with NavigationUI. 2020). Asetukset komponenttien, navigaatiokaavion ja navControllerin välillä tehtiin MainActivityssä.

3.5 Sovelluksen ulkoasu ja resurssitiedostot

Sivujen näkymien asettelu määritellään .xml-tiedostoissa. Tiedostot sijoitetaan resurssitiedostoina layout-resurssihakemistoon. Sivujen elementtien asetteluun on olemassa erilaisia asettelumalleja eli layouteja. Sovelluksessa käytettiin useita asettelumalleja ja kullekin sivulle valittiin tarkoitukseen sopiva layout, kuten sivuvalikolle drawer-layout, jotta se toimisi oikealla tavalla. Sovellukselle tehtiin käynnistyskuvake, joka näkyy myös sivuvalikossa. Se muokattiin valmiista kuvakkeista. Ne ja sivuvalikossa sekä toimintopalkissa näkyvät kuvakkeet saatiin Googlen Material Design -sivustolta. Kuvakkeet lisättiin drawable-resurssihakemistoon.

Sovelluksessa käytettävät kuvat, merkkijonot, värit ja muut suorituksen aikana samana pysyvät tiedot suositellaan sijoitettavaksi erillisiin resurssitiedostoihin. Se tekee sovelluksen ylläpidosta helpompaa, ja esimerkiksi strings-resurssitiedoston avulla, voidaan helposti toteuttaa sovelluksesta erikielisiä versioita. (Harju 2013, 119.) Sovelluksessa lisättiin values-resurssihakemistoon strings-, colors-, dimens- ja styles-tiedostot. Strings-tiedostossa määritellään kaikki sovelluksen tekstit merkkijonoina, colors-tiedostossa sovelluksen värit, dimens-tiedosto sisältää sovelluksessa käytetyt mitat ja styles-tiedostossa määriteltiin painikkeiden, sivuvalikon tekstin ja ohje-sivun tekstien tyyli. Kun tiedostoja käytetään, esimerkiksi muutetaan väriä colors-tiedostossa, näkyy muutos heti koko sovelluksessa kohdissa, joissa kyseistä väriä käytetään. Muita sovelluksen resurssitiedostoja ovat jo aiemmin esitelty navigation, ja lisäksi menu-resurssihakemisto, joka sisältää sivuvalikon ja toimintopalkin valikoiden layout-tiedostot.

3.6 Tekstintunnistuskirjaston lisääminen ja käyttö sovelluksessa

Googlen kirjastot lisättiin sovelluksen käyttöön Googlen pilvipalvelun ja Firebasen konsolissa. Aluksi Googlen konsolissa luotiin uusi projekti, johon voitiin liittää halutut kirjastot.

Firebase ML:n kirjasto tekstin tunnistamiseen lisättiin projektiin Firebasen konsolin kautta. Firebase lisättiin aiemmin luotuun projektiin. Olemassa oleva sovellus rekisteröitiin sen pakkauksen nimellä, jonka jälkeen pilvipalvelu generoi Googlen palvelun google.service.json -tiedoston, joka talletettiin Android Studiossa projektin juurihakemiston App-kansioon. Se sisältää projektin ID-tiedot ja API-avaimet. Projektin build.gradle-tiedostoon lisättiin myös riippuvuus, jotta palvelua käytetään .json-tiedoston lataamiseen.

```
classpath 'com.google.gms:google-services:4.3.3'
```

Lisäksi sovelluksen build.gradle-tiedostoon lisättiin riippuvuudet tarvittaviin kirjastoihin:

```
implementation 'com.google.firebase:firebase-analytics:17.4.4'
implementation 'com.google.firebase:firebase-ml-vision:24.0.3'
```

Koska sovelluksessa käytetään pilvessä tapahtuvaa tunnistusta, piti projekti päivittää Blaze Plan -projektiksi, koska vain sellaiset projektit voivat käyttää pilvipohjaisia kirjastoja. Lisäksi pilvessä tapahtuvaa tunnistusta varten piti Googlen pilvipalvelussa täyttää itsestä laskutusprofiili. Käytössä on kuitenkin 1000 ilmaista tunnistusta kuukaudessa, jonka jälkeen laskutus alkaa. Lisäksi on mahdollista saada käyttöön 300 $ ilmainen kokeilujakso kaikkiin Googlen pilvipalveluihin. Käytön määrän voi halutessaan nähdä ja seurata konsolissa. Lopuksi palattiin vielä Firebasen konsoliin, jossa projektiin liitettiin tarvittava kirjasto.

Tekstintunnistamisen käyttäminen

Tekstintunnistusta varten otetaan ensin kuva sovelluksessa olevalla kameralla. Kun kuva on otettu, luodaan kuvan bitmapista FirebaseVision image. Tämä nä-

kyy alla olevassa koodissa ensimmäisellä rivillä. Sen jälkeen luodaan tunnistaja-olio, textDetector, jolle annettiin asetuksia options-muuttujalla. Kielivihjeen asettaminen suomeksi(”fi”) auttaa tunnistamisen tekemistä, ja mallityyppi on 2, koska tällöin tunnistuksen yhteydessä saadaan tietää myös tunnistuksen luotettavuus (Recognize Text in Images with Firebase ML on Android. 2020; FirebaseVisionText.TextBlock. 2020.)

```
FirebaseVisionImage visionImage =FirebaseVisionImage.fromBitmap(imageBitmap);
FirebaseVisionCloudTextRecognizerOptions options =
        new FirebaseVisionCloudTextRecognizerOptions.Builder()
        .setLanguageHints(Arrays.asList("fi"))
        .setModelType(2)
        .build();

FirebaseVisionTextRecognizer textDetector =
        FirebaseVision.getInstance().getCloudTextRecognizer(options);
```

Seuraavassa koodissa kuva välitetään tunnistajan processImage-funktiolle, jolloin se lähetetään pilvipalveluun tunnistettavaksi. Tämän jälkeen ylikirjoitetaan onSuccess- ja onFailure-funktiot onnistuneeseen ja epäonnistuneeseen tunnistamiseen. Onnistumiseen asetetaan kuuntelijat. Jos tunnistus epäonnistuu, viestitään siitä käyttäjälle Toastilla, ja jos se onnistuu, saadaan tunnistettu teksti kuuntelijan FirebaseVisionText texts-muuttujaan.

```
textDetector.processImage(visionImage).addOnSuccessListener(new OnSuccessLis-
tener<FirebaseVisionText>() {

   @Override
   public void onSuccess(FirebaseVisionText texts) {
       displayTextFromImage(texts);
   }

}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Toast.makeText(getContext(), "Error" +e.getMessage(),
Toast.LENGTH_SHORT);
    }
});
```

Tunnistettu teksti on texts-muuttujassa, joka sisältää 0 tai useamman TextBlock-objektin eli lohkon. Se on nelikulmainen objekti, joka sisältää 0, yhden tai useita Line-objekteja eli rivejä tekstiä. Rivi taas sisältää Element-objekteja, jotka voivat olla sanoja tai sentapaisia elementtejä, kuten numeroita. Objekteja voidaan käyttää sovelluksessa tarpeen mukaan. Niistä saa halutessaan tunnistetun tekstin, tunnistuksen luotettavuuden, listan, jossa tekstistä tunnistetut kielet ovat luotettavuuden mukaan järjestettynä, tekstin kulmien sijaintien pisteet ja valmiin kehikon, joka kertoo objektin sijainnin kuvassa. (Recognize Text in Images with Firebase ML on Android. 2020; FirebaseVision-Text.TextBlock. 2020.) Alla olevassa koodissa näkyy esimerkiksi TextBlock-objektin käsittely for-silmukassa ja siitä saatavat ominaisuudet.

```
for (FirebaseVisionText.TextBlock block : texts.getTextBlocks()) {
    String blockText = block.getText();
    Float blockConfidence = block.getConfidence();
    List<RecognizedLanguage> blockLanguages=
            block.getRecognizedLanguages();
    Point[] blockCornerPoints = block.getCornerPoints();
    Rect blockFrame = block.getBoundingBox();
}
```

Sovelluksessa kuvaan piirretään tekstin sijainnin kertova kehikko, sen yläpuolelle tunnistettu teksti ja tunnistuksen luotettavuus prosentteina, kuten näkyi kuvassa 3. Lisäksi teksti ja luotettavuus näkyvät erillisessä tekstikentässä kuvan alapuolella. Siihen lisätään myös todennäköisimmän kielen tunnus. Todennäköisyytenä käytetään lohkon todennäköisyyttä ja kehikko piirretään niin ikään lohkon tiedoilla. Tekstikenttään luetaan teksti riveittäin samaan muuttujaan Line-objektista, ja siinä käytetään lohkon luotettavuutta.

Tunnistuksen tulos saadaan lohkoina. Jos kadunnimi ja numero ovat kuvassa eri riveillä, tunnistaa sovellus ne 2:ksi eri lohkoksi, jolloin kummallekin riville tulee kuvaan oma kehikko ja todennäköisyys. Samalla rivillä oleva teksti ja numero ovat samaa lohkoa ja tällöin ilmestyy kuvaan vain yksi kehikko ja todennäköisyys.

3.7 Kartta- ja paikannuskirjastojen lisääminen ja käyttö sovelluksessa

Jotta sovelluksessa voi käyttää karttoja ja tehdä paikannuksia, liitettiin siihen tarvittavat kirjastot. Karttakirjasto Androidille, Maps SDK for Android, liitettiin pro-

jektiin Googlen pilvipalvelun konsolissa. Googlen Maps Platform APIt ovat käytettävissä 200 $:n arvosta ilmaiseksi joka kuukausi. (Pricing that scales to fit your needs. 2020.)

Lisäksi projektille luotiin Google API -avain, jolla palvelimen karttakirjastoja voi käyttää sovelluksessa (Get an API Key. 2020). Avain liitettiin sovellukseen Android Studiossa lisäämällä sovelluksen Manifest-tiedostoon seuraavat arvot:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="…" />
```

Manifest-tiedostossa määritellään sovelluksen toimintalogiikan määräävät komponentit ja muun muassa sovelluksen tarvitsemat käyttöluvat (Harju 2013, 38). Kartat ja paikannus toimivat sovelluksessa Google Play Servicen kautta. Se asennettiin Android Studiossa Android SDK Managerin avulla ja Manifest-tiedostoon lisättiin sen versiotiedot, jotka näkyvä alla. (Set Up Google Play Services. 2020.)

```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

Kirjastot lisättiin build.gradle-tiedostossa sovellukseen seuraavilla riippuvuuksilla:

```
implementation 'com.google.android.gms:play-services-maps:17.0.0'
implementation 'com.google.android.gms:play-services-location:17.0.0'
```

Manifest-tiedostoon lisättiin tiedot tarvittavista käyttöluvista, jotta paikannus voidaan tehdä.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="com.google.android.providers.gsf.permis-
sion.READ_GSERVICES" />
```

Karttojen lisääminen sovellukseen

Kartat lisättiin sovelluksen sivuille käyttämällä niiden layout-tiedostoissa fragmenttia. Kartta saadaan Googlen palvelimelta callbak-funktiolla, jonka jälkeen se asetetaan näkyviin ja siihen voi lisätä esimerkiksi paikkamerkkejä. (OnMapReadyCallback. 2020.) Sovelluksen etusivulla on kartta näkyvillä heti, kun sovelluksen avaa. Kun halutaan tehdä paikannus, painetaan siihen tarkoitettua painiketta. Paikannuksen tekemiseen tarvitaan käyttäjän lupa hänen yksityisyytensä suojelemiseksi. Aiemmin lisättiin Manifest-tiedostoon tieto siitä, että tarvitaan käyttäjän lupa ACCESS_FINE LOCATION-paikannukseen. Se on tarkka paikannusmenetelmä, jonka tulos pitäisi olla tarkempi kuin yhden korttelin tarkkuus. (Request location permissions. 2020.) Lupa kysytään ensimmäisellä kerralla ennen paikannuksen tekemistä. Tämän jälkeen se oletetaan sallituksi. Lisäksi, jotta paikannus toimii, pitää laitteen paikannuksen olla myös päällä.

Paikannuksen tekeminen

Paikannus tehdään paikantamalla puhelimen sijainti. Siihen käytetiin Fusion Location Provider -APIa, joka on eräs paikannusmenetelmä Google Play Palvelussa (Get the last known location. 2020). Sijaintitiedot pyydetään tietyin väliajoin sovellukseen. Aluksi luodaan LocationRequest mLocationRequest-objekti, johon liitetään parametreja. Tämä näkyy seuraavassa koodissa. Päivitysten väliajaksi asetettiin 5 sekunnin jakso ja prioriteetiksi korkea tarkkuus, PRIORITY_HIGH_ACCURACY.

```
LocationRequest mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(5000);
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

Paikannuspyynnölle voidaan asettaa prioriteetti setPriority-funktiolla. Sen avulla paikannuspalvelu tekee päätöksen, millä menetelmällä paikannus tehdään. Esimerkiksi PRIORITY_BALANCED_POWER_ACCURACY-prioriteetilla tarkkuus on noin 100 m ja sen käyttäminen kuluttaa vähemmän virtaa. Paikannus tehdään todennäköisesti käyttämällä Wi-Fiä ja matkapuhelimen tukiasemia. Sovelluksessa käytettävällä PRIORITY_HIGH_ACCURACY-prioriteetilla saadaan tarkin

mahdollinen tulos ja paikannus tehdään todennäköisemmin GPS-paikannuksella. Prioriteetin ja 5 sekunnin päivitysvälin yhdessä ACCESS_FINE_LOCATION-paikannuksen kanssa pitäisi tuottaa yhden jalan eli noin 30 cm:n sijainnin tarkkuus. (Change location settings, 2020; Jalka (mittayksikkö). 2018.) Tällainen tarkkuus valittiin, koska sovelluksella halutaan paikantaa tietty paikka mahdollisimman tarkasti.

Fused Location Provider Client -oliolla pyydetään sijainnin päivityksiä requestLocationUpdates() -funktiolla, kuten alla olevassa koodissa näkyy. Olio kutsuu LocationCallback.onLocationResult() callback-funktiota ja mLocationRequest-objekti välitetään parametrina. Tuloksena saadaan lista, locationResult, jonka location objektista saadaan paikan pituus- ja leveyspiirit. (Request location updates. 2020.) Niiden avulla voidaan paikkamerkki sijoittaa oikeaan kohtaan kartalla.

```
LocationServices.getFusedLocationProviderClient(getActivity())
      .requestLocationUpdates(mLocationRequest, new LocationCallback() {
           @Override
           public void onLocationResult(LocationResult locationResult) {

             List<Location> locationList = locationResult.getLocations();
             if (locationList.size() > 0) {
                Location location = locationList.get(locationList.size()-1);

                latitude = location.getLatitude();
                longitude = location.getLongitude();
             }

           }

      }, Looper.myLooper());
```

3.8 Sovelluksen kamera

Sovelluksen kuvat otetaan kameralla, joka tehtiin Jetpack Camera -kirjastolla. Sen pitäisi tuottaa parempilaatuisia kuvia kuin vanhemmalla Camera-kirjastolla ja olla helppokäyttöisempi kuin sitä edeltävä kamerakirjasto. Kirjasto on vielä kehitysvaiheessa, joten siitä oli aika vähän ohjeita ja varsinkin Java-kielisiä ohjeita saatavilla. Kamerassa onkin vain perustoiminnot, sillä voi ottaa kuvia ja lisäksi siinä on säädin, jolla kohdetta voi zoomata. Kamera tehtiin omaan fragmenttiin, johon siirrytään kotisivulta, kun halutaan ottaa kuva. Kuvan ottamisen jälkeen palataan kotisivulle. Kameran käyttämiseen kysytään käyttäjän lupa ensimmäisellä

käyttökerralla, kuten paikannuksen tekemiseen. Manifest-tiedostoon lisättiin tieto tarvittavasta luvasta:

```
<uses-feature android:name="android.hardware.camera.any" />
<uses-permission android:name="android.permission.CAMERA" />
```

Kamerakirjastot lisättiin sovellukseen seuraavilla riippuvuuksilla:

```
def camerax_version = "1.0.0-beta05"
implementation "androidx.camera:camera-camera2:${camerax_version}"
implementation "androidx.camera:camera-lifecycle:${camerax_version}"
implementation "androidx.camera:camera-view:1.0.0-alpha12"
implementation "androidx.camera:camera-extensions:1.0.0-alpha12"
```

3.9 Sovelluksen tiedon tallennus ja tietokanta

Sovelluksessa voidaan tallentaa pysyvästi kuva, siitä tunnistettu teksti, tunnistuksen luotettavuus, tunnistetun kielen tunnus ja paikannustiedot. Näin kuvia ja tunnistettuja tekstejä tai paikkoja voi myöhemmin selata. Kaikki tiedot tallennetaan kerralla, joten oletuksena on, että käyttäjä tekee myös paikannuksen kuvan ottamisen ja tekstintunnistamisen jälkeen. Tallennus tehdään puhelimessa olevaan tietokantaan, SQL Liteen. Tietokantaa käytetään Room-kirjaston avulla. Se muodostaa abstraktin kerroksen SQL Liten päälle. Google suosittelee sen käyttöä, koska se luo vakaan, helpomman ja nopeammin toteutettavan tavan käsitellä tietokantaa, mutta sillä on silti samat ominaisuudet kuin SQL Litessä. (Save data in a local database using Room. 2020.)

Roomin käyttämiseksi lisättiin seuraavat riippuvuudet:

```
def room_version = "2.2.5"

implementation "androidx.room:room-runtime:$room_version"
annotationProcessor "androidx.room:room-compiler:$room_version"
testImplementation "androidx.room:room-testing:$room_version"
```

Roomissa on kolme pääkomponenttia, tietokanta, entiteetti ja Dao (data access object), jotka lisättiin sovellukseen. Dao on objekti, joka muodostaa rajapinnan tietokannan käsittelyyn. (Accessing data using Room DAOs. 2020; Save data in a local database using Room. 2020.)

3.9.1 Entiteetti

Entiteetti kuvaa tietokannan yhtä taulua. Sovellukseen tehtiin yksi taulu, Detections, seuraavasti:

```
@Entity(tableName = "detections")
public class Detection {
        @PrimaryKey(autoGenerate = true)
        public int id;
        @ColumnInfo(name = "image_address")
        public String imageAddress;
        @ColumnInfo(name = "detected_text")
        public String detectedText;
        @ColumnInfo(name = "longitude")
        public Double lon;
        @ColumnInfo(name = "latitude")
        public Double lat;
```

Tauluun tallennetaan kuvan osoite (image_address) puhelimen hakemistossa, kuvasta tunnistettu teksti (detected_text) sekä sijainnin pituuspiiri (longitude) ja leveyspiiri (latitude). Avain (primarykey) luodaan automaattisesti.

Kuvasta tallennetaan tietokantaan vain osoite ja kuva tallennetaan puhelimessa olevaan sovelluksen käyttöön tarkoitettuun tallennustilaan. Kuvat haetaan sieltä tietokannasta saadun osoitteen avulla, kun kuvia halutaan selata. Näin kuvien haku toimii nopeammin. Kuvat tallennetaan .png-muodossa ja niiden koko on noin 1,8 Mt tai suurempi. Kuvat, joiden koko on yli 1 Mt, kannattaa tallentaa mieluummin hakemistoon kuin tietokantaan (Grey 2006.) Kuvien tallennuksesta kerrotaan tarkemmin kappaleessa 3.11.

Koska kaikkien tietokannan taulun sarakkeiden tietoja ei tarvita sovelluksessa kerralla, kyselyitä varten vain tarvittavista sarakkeista luotiin lisäksi 2 sisäistä luokkaa, yksi kuvien ja tunnistettujen tekstien kyselyä varten sekä toinen koordinaattien ja tunnistettujen tekstien kyselyjen tekemiseen.

3.9.2 Dao

Dao voi olla rajapinta tai abstrakti luokka. Tässä työssä se on rajapinta (interface), kuten alla olevassa koodissa näkyy. Se sisältää kaikki funktiot, joilla voidaan käsitellä tietokantaa. Sovellukseen tehdyt kyselyt ja funktiot näkyvät myös koodissa.

Daon ja määrittelyjen kyselyjen avulla Room luo funktioiden toteutukset ohjelman suorituksen aikana. (Accessing data using Room DAOs. 2020). Funktiot palauttavat tiedot listassa.

```
@Dao
public interface MyDao {
    @Query("SELECT image_address, detected_text FROM detections")
    public List<ImageAndText> findImageAndDetectedText();
    @Query("SELECT detected_text, longitude, latitude FROM detections")
    public List<LatLonText> findCoordinates();
    @Insert
    void insert(Detection detection);
    @Query("DELETE FROM detections")
    void deleteAll();

}
```

3.9.3 Tietokanta

Tietokanta luodaan omassa tiedostossaan periyttämällä se Room tietokannasta seuraavasti:

```
@Database(entities = Detection.class, exportSchema = false, version = 1)
public abstract class DetectionDatabase extends RoomDatabase {
    public abstract Detection.MyDao myDao();
    private static final String DB_NAME="detection_db";
    private static volatile DetectionDatabase INSTANCE;
    private static final int NUMBER_OF_THREADS = 4;
    private ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.new
    FixedThreadPool(NUMBER_OF_THREADS);

    public static synchronized DetectionDatabase getInstance(Context context)
    {
    if (INSTANCE == null) {
    synchronized (DetectionDatabase.class) {
    if (INSTANCE == null) {
        INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                            DetectionDatabase.class, DB_NAME)
                            .build();
                }
            }
        }
      return INSTANCE;
    }

    public ThreadPoolExecutor getExecutor() {
        return executor;
    }
}
```

Luokka on abstrakti, luonnin yhteydessä listataan tietokannan entiteetti (entities), joka vastaa tietokannan taulua, exportSchema ja versionumero. Luokka sisältää abstraktin myDao funktiomäärittelyn Daolle, jonka avulla sitä käytetään sovelluksessa. Siinä määritellään myös tietokannan instanssin (instance) luonti niin sanottuna sigle-instanssina, jotta tietokannasta ei avata useita instansseja samaan aikaan. (Android Room with a View – Java. 2020)

Tietokantaa ei voi Roomin avulla käsitellä sovelluksen pääsäikeestä, ellei sitä erikseen sallita (Accessing data using Room DAOs. 2020). Sovelluksessa kyselyt ja tietojen lisääminen ja poistaminen toteutettiin omissa asynkronisissa säikeissään. Ilman erillisiä säikeitä sovellus voi hidastua tai pysähtyä kokonaan, kun se odottaa pitkäkestoisten tehtävien valmistumista. Säikeet toteutettiin käyttämällä Javan CompletableFuture APIa, jolla voidaan tehdä asynkronisia tehtäviä sovelluksissa. Kyselyt tehtiin CompletableFuturen supplyAsync-funktiolla. Se suorittaa kyselyn taustasäikeessä, kunnes se on valmis, jonka jälkeen Futuren get-fuktiolla saadaan kyselyn tulos muuttujaan. Tietojen lisääminen ja poistaminen toteutettiin CompletableFuturen runAsync-funktiolla, joka ei palauta mitään, vaan suorittaa vain jonkin tehtävän. (Singh 2017.)

Säikeitä varten DetectionDatabase-luokassa luotiin myös sovelluksen yhteinen FixedThreadPool neljällä säikeellä sekä getExecutor-funktio säikeiden toteuttajan palauttamista varten. Säikeitä käytetään uudelleen tehtävien suorittamisessa. Jos kaikki säikeet ovat varattuja, odottavat tehtävät jonossa säikeiden vapautumista (Java Thread Pool – ThreadPoolExecutor Example. 2019).

3.10 Tietojen poistaminen ja ohje-sivu

Poista kaikki -sivulla voidaan poistaa tietokannasta kaikki tiedot pysyvästi painikkeen avulla. Muuten tallennetut tiedot poistetaan vain silloin, kun sovellus poistetaan puhelimesta. Sivulle pääsee sivuvalikon kautta. Kaikilta sivuilta, lukuun ottamatta kameraa, pääse toimintopalkin kuvakkeen kautta ohje-sivulle. Ohjeeseen voidaan siirtyä myös sivuvalikosta. Siellä kerrotaan, kuinka sovellusta käytetään.

3.11 Kuvien tallennus

Kuvat tallennetaan puhelimessa olevaan sovelluskohtaiseen tallennustilaan, ulkoiseen tallennushakemistoon. Muut sovellukset voivat päästä hakemistoon, jos niillä on siihen lupa, mutta hakemisto on tarkoitettu vain sovelluksen käyttöön. Jos sovellus poistetaan puhelimesta, myös tallennetut kuvat poistetaan. (Access app-specific files. 2020.) Kuva tallennetaan hakemistoon heti kuvan ottamisen yhteydessä. Ennen tallennusta kuvalle luodaan tiedosto ja tiedostopolku, johon se tallennetaan. Tieto polusta, johon tallennus tehdään, välitetään etusivulle. Navigation-rakenteessa tietoja lähtösivulle voi välittää Bundle-muuttujan avulla, johon tieto tallennetaan kameran .java-tiedostossa. Etusivulla tieto saadaan siihen tarkoitetulla funktiolla, jonka jälkeen haetaan kuva polusta ja laitetaan näkyviin sivulle. Tiedot parametristä laitetaan navigaatiokaavioon.

Tallennustapa valittiin, koska sen avulla saadaan parempilaatuisia kuvia verrattuna kuvien tallentamiseen vasta etusivulla. Tällöin kuvaan olisi ensin lisätty tekstin ympärille kehikko ja sen yläpuolelle tunnistettu teksti tunnistuksen jälkeen. Tällöin kuva olisi ollut bitmap-muodossa. Sitten bitmap olisi pitänyt pakata ennen tallennusta, koska se vie muuten paljon tallennustilaa (Handling bitmaps. 2020). Pakkaus huononsi kuvan laatua kuitenkin huomattavasti, joten tallennus päätettiin tehdä kuvan ottamisen jälkeen. Nyt kuvia kertyy sovelluksen tallennustilaan ajan myötä, ellei sovellusta poisteta puhelimesta, mutta asiaa ei käsitellä tässä työssä tarkemmin.

3.12 Kuvien selailu

Sovelluksessa voidaan selata tallennettuja kuvia ja tunnistettuja tekstejä Kaikki kuvat (All Images) -sivulla. Kuvassa 6 on kuvaruutukaappaus sivulta. Se toteutettiin RecyclerView-komponentin avulla, jolla voidaan tehdä rullattava lista tai ruudukko. Sovelluksessa käytettiin listaa, jossa kuvat ovat allekkain ja niitä rullataan pystysuunnassa. Kun uusia kuvia ja tekstejä lisätään tietokantaan, tulevat ne näkyviin listaan.

KUVA 6. Kuvaruutukaappaus Kaikki kuvat (All Images) -sivulta

Sivun asettelutiedostoon lisättiin RecyclerView-komponentti, joka on säiliö kuville ja tekstille. Se näkyy seuraavassa koodissa. Kuvien ja tekstien esittämistä varten tehtiin oma .xml-tiedosto.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center_horizontal"
    android:scrollbars="vertical">
</androidx.recyclerview.widget.RecyclerView>
```

Tarvittiin myös uusi .java-tiedosto, RecyclerAdapter, jossa luodaan adapteri, joka hallinnoi View Holder -objekteja. Ne esittävät listassa olevat objektit, kuvat ja tekstit. Adapteri luo tarvittaessa uusia View Holder -objekteja ja sitoo esitettävät kuvat ja tekstit niihin. (Create a List with RecyclerView. 2020.) Recycler View asetetaan näkyviin ja kuvat ja tekstit listaan haetaan tietokannasta sivun .java-tiedostossa. (Create a List with RecyclerView. 2020.)

3.13 Paikkojen selailu

Kaikki paikat (All Places) -sivulla voi nähdä kaikki tallennetut paikat samalla kartalla. Tiedot haetaan tietokannasta ja lisätään näkyviin kartalle karttamerkeillä. Karttaa voi zoomata kartalla olevilla painikkeilla tai sormilla. Paikkamerkkien infoikkunassa näkyy paikalla tunnistettu teksti, jos merkkiä klikkaa. Kuvaruutukaappaus sivulta on kuvassa 7.

KUVA 7. Ruutukaappaus Kaikki paikat (All places) -sivulta

4 SOVELLUKSEN TESTAUS

Valmista sovellusta testattiin 15 tapauksella. Sellaisia kadunnimiä, joissa olisi ollut myös numero, oli aika hankala löytää, joten tapauksia ei ole kovin paljon. Kylteissä nimi ja numero oli mustalla tekstillä valkoisella pohjalla. Testaus tehtiin päivällä, aurinkoisella ilmalla. Testauksen tulokset näkyvät taulukossa 2.

TAULUKKO 2. Sovellukset tekstintunnistuksen testauksen tulokset

Luotettavuus %	Kadunnimi/kpl	Numero/kpl
99	10	9
97–98	4	4
0–97	1	2

Kadunnimien tunnistaminen toimi hyvin. Kymmenen kadunnimen tunnistamisen luotettavuus oli 99 % ja neljässä se oli lähes yhtä korkea 97–98 %. Kuvassa samalla rivillä oleva teksti ja numero tunnistettiin samaksi lohkoksi ja tällöin niiden luotettavuus oli sama. Jos kadunnimi ja -numero oli kuvissa eri rivillä, tunnisti sovellus ne 2:ksi eri lohkoksi, jolloin niiden tunnistamisen luotettavuudessa oli eroa kahdessa tapauksessa. Yhdessä tapauksissa numeroa ei tunnistettu ollenkaan ja toisessa tunnistettiin kahdeksi kirjaimeksi. Numeroiden kirjasin oli tällöin eri kuin kadunnimessä ja hieman erikoinen verrattuna tavallisimpiin kirjasimiin.

Tunnistuksissa, joissa luotettavuus oli 97–99 % kuvassa ei näkynyt juuri muuta kyltin lisäksi kuin seinää, kuten kuvan 3 ruutukaappauksessa. Joitakin kuvia otettiin myös kauempaa, jolloin niissä näkyi enemmän ympäristöä. Kahdessa kuvassa ympäristössä olevat tekijät tuottivat, oikein tunnistetun kadunnimen lisäksi, muuta tekstiä tulokseen. Yhdessä kuvassa kadunnimen ympärillä olevat kiiltävät laatat heijastivat vastapäätä olevien yritysten nimiä kuvaan ja toisessa kuvassa oleva valkoinen aita tunnistettiin tekstiksi. Kuvan voi nähdä kuvassa 7. Muissa

kauempaa otetuissa kuvissa sekä kadunnimi että -teksti tunnistettiin kuitenkin 97–99 %:n luotettavuudella.

Kuvia, joissa todennäköisin tunnistettu kieli oli suomi, oli 13. Kahdessa tapauksessa kieli oli jokin muu. Paikannus toimi niin ikään hyvin. Kartasta arvioituna sen tarkkuus oli noin 30 cm suurimmassa osassa paikannuksia. Joissakin tapauksissa paikkamerkki oli kuitenkin väärällä puolen katua, mutta tarkkuus on silloinkin muutaman metrin luokkaa ja se on riittävä, jos haluaa esimerkiksi osata kartan perusteella samalle paikalle. Tarkkuus vastaa myös asetuksia, jotka tehtiin paikannuspyynnölle sovelluksessa.

5 YHTEENVETO

Työn tutkimusongelmana oli, miten kadunnimiä -ja numeroita voidaan tunnistaa ja paikantaa Android-puhelimella. Vastauksena tutkimusongelmaan työn teoria-osuudessa perehdyttiin konenäön teoriaan ja kehitykseen, jonka jälkeen selvitettiin objektien tunnistuksen peruskäsitteitä ja perehdyttiin Googlen ja Microsoftin konenäön pilvipalveluihin. Tavoitteena oli selvittää, mitä asioita konenäön avulla voidaan tunnistaa. Molemmat yritykset tarjoavat koneoppimista hyödyntäviä ratkaisuja käytettäväksi helposti ja edullisesti. Palvelujen tarjonnassa ei ollut suurta eroa, sillä molempien ominaisuuksien avulla voidaan tunnistaa pitkälti samoja asioita. Niillä voidaan tunnistaa objekteja eri luokista, kuten eläimet, kasvit tai laitteet. Molemmilla oli myös kirjastot muun muassa tekstin, kasvojen ja nähtävyyksien tunnistamiseen. Palvelujen väliltä löytyisi kuitenkin varmasti enemmän eroja, jos niihin perehtyisi yksityiskohtaisesti, esimerkiksi tekemällä sovelluksia ja vertailemalla niitä. Teoriaosuuden lopuksi luotiin silmäys sovelluksessa käytettävään konenäön teknologiaan, tekstintunnistukseen, sekä selvitettiin, niin ikään sovelluksessa käytettävän, paikannuksen peruskäsitteitä.

Työn toisessa osassa toteutettiin tavoitteena ollut Android-sovellus. Se toimii asetettujen tavoitteiden mukaisesti: sillä voidaan ottaa kuvia, tunnistaa kuvassa oleva kadunnimi ja -numero ja paikantaa ne paikantamalla kuvanottopaikka. Kuvat, tunnistetut tekstit ja paikkatiedot voidaan tallentaa puhelimeen. Kuvia ja tekstejä voidaan myöhemmin katsella selaussivulla ja paikat saadaan näkyviin samalle kartalle. Tekstintunnistukseen käytettiin Android-alustalle tarkoitettua Firebase ML -kirjastoa pilvessä tapahtuvaan tunnistukseen ja paikannuksen tekemiseen ja karttojen esittämiseen Googlen paikannus- ja karttakirjastoja. Sovellus tehtiin Android Studiolla Javalla ja siinä käytettiin erilaisia komponentteja Jetpackin kirjastoista. Sovelluksen toteutuksessa ei ollut suuria ongelmia. Oppaina käytettiin Googlen Android Developers- ja Stackoverflow-sivustoja sekä joitakin muita nettilähteitä. Android-ohjelmointi ja Jetpackin ominaisuudet tulivat tutummaksi ja useita ominaisuuksia voisi käyttää jatkossa myös muissa sovelluksissa.

Sovelluksen testauksessa kadunnimen tunnistamisen luotettavuus oli suurimmaksi osaksi 97–99 %. Myös numerot tunnistettiin suurimmaksi osaksi yhtä tarkasti, mutta joissakin tapauksissa sitä ei tunnistettu ollenkaan tai se tunnistettiin kirjaimeksi. Tulokset tukevat teoriaa siitä, että tekstintunnistus toimii nykyään jo lähes 100 %:n luotettavuudella painetun tekstin tai vastaavan, kuten mustavalkoisten katukylttien, kohdalla. Tekstin ympäristö ulkona otetuissa kuvissa voi kuitenkin aiheuttaa häiriötekijöitä tunnistamiseen, kuten testin kahdessa tapauksessa tapahtuikin. Paikannus toimi hyvin, noin 30 cm tarkkuudella tai ainakin korttelin tarkkuudella, mikä on riittävä tarkkuus sovelluksessa.

Tällaista sovellusta voisi käyttää esimerkiksi matkoilla, jolloin kuvatusta kadunnimestä ja -numerosta jäisi kuvan lisäksi muistoksi myös paikannustieto kartalle. Karttaa voisi käyttää myös opaskarttana, jos haluaisi palata samalle paikalle myöhemmin. Sovellusta voisi tietysti käyttää minkä tahansa kohteen, jossa on tekstiä, kuvaamiseen ja paikantamiseen. Sovellusta voisi myös vielä kehittää ja lisätä siihen ominaisuuksia. Jotta kuvia ei kertyisi turhaan puhelimen muistiin, pitäisi siihen lisätä kuvien poistaminen sovelluskohtaisesta tallennustilasta, ellei niitä tallenneta tietokantaan. Uusia ominaisuuksia voisi olla yhden kuvan ja siihen liittyvien tietojen poistaminen, mahdollisuus siirtyä isommalle kartalle kotisivun paikannuksesta sekä suomenkielisen version tekeminen.

LÄHTEET

Access app-specific files. 2020. Google. Saatavissa: https://developer.android.com/training/data-storage/app-specific. Hakupäivä 3.7.2020.

Accessing data using Room DAOs. 2020. Google. Saatavissa: https://developer.android.com/training/data-storage/room/accessing-data. Hakupäivä 3.6.2020.

Android Jetpack. 2020. Google. Saatavissa: https://developer.android.com/jetpack. Hakupäivä 3.7.2020.

Android Room with a View – Java. 2020. Google. Saatavissa: https://codelabs.developers.google.com/codelabs/android-room-with-a-view/index.html?index=..%2F..index#7. Hakupäivä 3.6.2020.

Applying content tags to images. 2019. Microsoft. Saatavissa: https://docs.microsoft.com/en-gb/azure/cognitive-services/computer-vision/concept-tagging-images. Hakupäivä 3.6.2020.

Azizpour, Hossein 2016. Visual Representations and Models: From Latent SVM to Deep Learning. KTH School of Computer Science and Communication. Saatavissa: http://www.diva-portal.org/smash/get/diva2:967455/FULLTEXT01.pdf. Hakupäivä 3.6.2020.

Build location-aware apps. 2020. Google. Saatavissa: https://developer.android.com/training/location/. Hakupäivä 3.6.2020.

Brownlee, Jason 2019. A Gentle Introduction to Computer Vision. Saatavissa: https://machinelearningmastery.com/what-is-computer-vision/. Hakupäivä 3.6.2020.

Change location settings. 2020. Google. Saatavissa: https://developer.android.com/training/location/change-location-settings. Hakupäivä 3.6.2020.

Cloud AutoML. 2020. Google. Saatavissa: https://cloud.google.com/automl. Hakupäivä 3.6.2020.

Configure your build. 2020. Google. Saatavissa: https://developer.android.com/studio/build. Hakupäivä 3.6.2020.

Create a List with RecyclerView. 2020. Google. Saatavissa: https://developer.android.com/guide/topics/ui/layout/recyclerview. Hakupäivä 3.6.2020.

Custom Vision. 2019. Microsoft. Saatavissa: https://azure.microsoft.com/en-gb/services/cognitive-services/custom-vision-service/. Hakupäivä 3.6.2020.

Dervisevic, Ivan 2006. Machine Learning Methods for Optical Character Recognition. Saatavissa: https://perun.pmf.uns.ac.rs/radovanovic/dmsem/completed/2006/OCR.pdf. Hakupäivä 3.6.2020.

Detect common objects in images. 2019. Microsoft. Saatavissa: https://docs.microsoft.com/en-gb/azure/cognitive-services/computer-vision/concept-object-detection. Hakupäivä 3.6.2020.

Detecting image types with Computer Vision. 2019. Microsoft. Saatavissa: https://docs.microsoft.com/en-gb/azure/cognitive-services/computer-vision/concept-detecting-image-types. Hakupäivä 3.6.2020.

Detect Labels. 2020. Google. Saatavissa: https://cloud.google.com/vision/docs/labels. Hakupäivä 3.6.2020.

Detect multiple objects. 2020. Google. Saatavissa: https://cloud.google.com/vision/docs/object-localizer. Hakupäivä 3.6.2020.

Detect text in images. 2020. Google. Saatavissa: https://cloud.google.com/vision/docs/how-to. Hakupäivä 3.6.2020.

Detect Web entities and pages. 2020. Google. Saatavissa: https://cloud.google.com/vision/docs/detecting-web. Hakupäivä 3.6.2020.

Features list. 2020. Google. Saatavissa: https://cloud.google.com/vision/docs/features-list. Hakupäivä 3.6.2020.

Firebase Machine Learning. 2020. Google. Saatavissa: https://firebase.google.com/docs/ml. Hakupäivä 3.6.2020.

FirebaseVisionText.TextBlock. 2020. Google. Saatavissa: https://firebase.google.com/docs/reference/android/com/google/firebase/ml/vision/text/FirebaseVisionText.TextBlock. Hakupäivä 3.6.2020.

Get an API Key. 2020. Google. Saatavissa: https://developers.google.com/maps/documentation/android-sdk/get-api-key. Hakupäivä 3.6.2020.

Get the last known location. 2020. Google. Saatavissa: https://developer.android.com/training/location/retrieve-current. Hakupäivä 3.6.2020.

Get started with the Navigation component. 2020. Google. Saatavissa: https://developer.android.google.cn/guide/navigation/navigation-getting-started?hl=en. Hakupäivä 3.6.2020.

GPS. 2020. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/GPS. Hakupäivä 3.6.2020.

Grey, Jim 2006. To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem. Saatavissa: https://www.microsoft.com/en-us/research/publication/to-blob-or-not-to-blob-large-object-storage-in-a-database-or-a-filesystem/. Hakupäivä 3.6.2020.

Handling bitmaps. 2020. Google. Saatavissa: Saatavissa: https://developer.android.com/topic/performance/graphics. Hakupäivä 3.6.2020.

Harju, Jukka 2013. Android-ohjelmoinnin perusteet. Helsinki: Books on Demand GmbH.

Huimin, Lu – Yujie, Li – Min, Chen – Hyoungseop, Kim – Seiichi, Serikawa 2017. Brain Intelligence: Go beyond Artificial Intelligence. Mobile Networks and Applications (2018) vol. 23, nro 2. S. 368–375. Saatavilla: https://link.springer.com/article/10.1007/s11036-017-0932-8. Hakupäivä 3.6.2020.

Improve OCR Accuracy with Advanced Image Processing. DocParser. 2020. Saatavissa: https://docparser.com/blog/improve-ocr-accuracy/. Hakupäivä 3.6.2020.

Jalka (mittayksikkö). 2018. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Jalka_(mittayksikkö). Hakupäivä 3.6.2020.

Johansson, Elias 2019. Separation and extraction of valuable information from digital receipts using Google Cloud Vision OCR. Engineering's Degree Project. Linnaeus University. Faculty of Technology. Saatavissa: http://www.diva-portal.org/smash/get/diva2:1349283/FULLTEXT01.pdf. Hakupäivä 3.6.2020.

Kavelar, Albert – Zambanini, Sebastian – Kampel, Martin 2013. Reading Ancient Coin Legends: Object Recognition vs. OCR. The 37th Annual Workshop of the Austrian Association for Pattern Recognition. Cornellin yliopisto. Saatavissa: https://arxiv.org/format/1304.7184. Hakupäivä 3.6.2020.

Khan, Asharul Islam – Al-Habsi, Salim 2019. Machine Learning in Computer Vision. Procedia Computer Science (2020). nro 167. S. 1444–1451. Saatavissa: https://www.sciencedirect.com/science/article/pii/S1877050920308218. Hakupäivä 3.6.2020.

Korpela, Jari 2019. Koneoppimisen hyödyntäminen kaupallisen lentoyhtiön toiminnoissa. Kandidaatintutkielma. Jyväskylän yliopisto. Tietojärjestelmätiede. Saatavissa: https://jyx.jyu.fi/bitstream/handle/123456789/63593/URN%3aNBN%3afi%3ajyu-201904242259.pdf?sequence=1&isAllowed=y. Hakupäivä 3.6.2020.

Lokesh, Gupta 2019. Java Thread Pool – ThreadPoolExecutor Example. Saatavissa: https://howtodoinjava.com/java/multi-threading/java-thread-pool-executor-example/. Hakupäivä 3.6.2020.

Malin, Kalle 2010. Paperilomakkeesta tietomalliin. Pro gradu -tutkielma. Tampereen yliopisto. Tietojenkäsittelytieteiden laitos. Saatavissa: https://docplayer.fi/19129166-Paperilomakkeesta-tietomalliin-kalle-malin.html. Hakupäivä 3.6.2020.

Map Objects. 2020. Google. Saatavissa: https://developers.google.com/maps/documentation/android-sdk/map. Hakupäivä 3.6.2020.

Maps. 2020. Google. Saatavissa: https://cloud.google.com/maps-platform/maps. Hakupäivä 3.6.2020.

Migration Guide. 2020. Google. Saatavissa: https://developers.google.com/ml-kit/migration. Hakupäivä 3.6.2020.

Navigation. 2020.Google. Saatavissa: https://developer.android.com/guide/navigation/. Hakupäivä 3.6.2020.

Nieminen, Elina 2018. Tietokonenäkö on kehittynyt huimasti, alan osaajat viedään käsistä – Professori: "Opiskelijoita ei uskalla lähettää konferensseihin, kun kaikki rekrytoidaan". Saatavissa: https://yle.fi/uutiset/3-10418805. Hakupäivä 3.6.2020.

Nilsson, Nils. J. 2012. A Biographical Memoir by. National Academy of Sciences. Saatavissa: http://www.nasonline.org/publications/biographical-memoirs/memoir-pdfs/mccarthy-john.pdf. Hakupäivä 3.6.2020.

OCR Language Support. Google. 2020. Saatavissa: https://cloud.google.com/vision/docs/languages. Hakupäivä 9.6.2020.

OnMapReadyCallback. 2020. Google. Saatavissa: https://developers.google.com/android/reference/com/google/android/gms/maps/OnMapReadyCallback. Hakupäivä 3.6.2020.

Optical Character Recognition OCR. 2020. Microsoft. Saatavissa: https://docs.microsoft.com/en-gb/azure/cognitive-services/computer-vision/concept-recognizing-text. Hakupäivä 3.6.2020.

Ouaknine, Arthur 2018. Review of Deep Learning Algorithms for Object Detection. Saatavissa: https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852. Hakupäivä 3.6.2020.

Paikannus. 2020. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Paikannus. Hakupäivä 3.6.2020.

Paikkatieto. 2020. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Paikkatieto. Hakupäivä 3.6.2020.

Papadopoulous, D.P – Clarke, A.D.F – Keller, F. – Ferrari, V. 2014. Learning to Detect Objects from Eye-Tracking Data. i-Perception vol 5, nro 5. S. 488–488. Saatavissa: https://journals.sagepub.com/doi/abs/10.1068/ii57. Hakupäivä 3.6.2020.

Pricing that scales to fit your needs. 2020. Google. Saatavissa: https://cloud.google.com/maps-platform/pricing/. Hakupäivä 3.6.2020.

Recognize Text in Images with Firebase ML on Android. 2020. Google. Saatavissa: https://firebase.google.com/docs/ml/android/recognize-text. Hakupäivä 3.6.2020.

Request location permissions. 2020. Google. Saatavissa: https://developer.android.com/training/location/permissions. Hakupäivä 3.6.2020.

Request location updates. 2020. Google. Saatavissa: https://developer.android.com/training/location/request-updates. Hakupäivä 3.6.2020.

Sankar, Shrinivasan – Bartoli, Adrien 2018. Model-based active learning to detect an isometric deformable object in the wild with a deep architecture. Saatavissa: https://arxiv.org/pdf/1806.02850v1.pdf. Hakupäivä 3.6.2020.

Sarshogh, Reza – Hines Keegan 2019. Learning to Read: Computer Vision Methods for Extracting Text from Images. Capital One. Saatavissa: https://medium.com/capital-one-tech/learning-to-read-computer-vision-methods-for-extracting-text-from-images-2ffcdae11594. Hakupäivä 3.6.2020.

Save data in a local database using Room. 2020. Google. Saatavissa: https://developer.android.com/training/data-storage/room. Hakupäivä 3.6.2020.

Set Up Google Play Services. 2020. Google. Saatavissa: https://developers.google.com/android/guides/setup. Hakupäivä 3.6.2020.

Singh, Rajeev 2017. Java CompletableFuture Tutorial with Examples. Saatavissa: https://www.callicoder.com/java-8-completablefuture-tutorial/. Hakupäivä 3.6.2020.

Tesseract OCR. 2020. Saatavissa: https://github.com/tesseract-ocr/tesseract. Hakupäivä 3.6.2020.

Tiippana, Jani 2018. Tekoälyn hyödyntäminen palvelukehityksessä. Kandidaatintutkielma. Jyväskylän yliopisto. Tietojärjestelmätiede. Saatavissa: https://jyx.jyu.fi/bitstream/handle/123456789/59548/URN%3aNBN%3afi%3ajyu-201809184149.pdf?sequence=1&isAllowed=y. Hakupäivä 3.6.2020.

Trier, Øivind Due – Jain, Anil K. – Taxt, Torfinn 1996. Feature extraction methods for character recognition-A survey. Pattern Recognition vol. 29, nro 4. S. 641–662. Saatavissa: https://www.sciencedirect.com/science/article/pii/0031320395001182. Hakupäivä: 3.6.2020.

Update UI components with NavigationUI. 2020. Google. Saatavissa: https://developer.android.com/guide/navigation/navigation-ui. Hakupäivä 3.6.2020.

Veit, Andreas – Matera, Tomas – Neumann, Lukas – Matas, Jiri – Belongie, Serge 2016. COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images. Saatavissa: https://vision.cornell.edu/se3/wp-content/uploads/2016/01/1601.07140v1.pdf. Hakupäivä 3.6.2020.

What is Computer Vision?. 2020. Microsoft. Saatavissa: https://docs.microsoft.com/en-gb/azure/cognitive-services/computer-vision/home#use-containers. Hakupäivä: 3.6.2020.

Vision API Documentation. 2020. Microsoft. Saatavissa: https://opdhsblobprod01.blob.core.windows.net/contents/4a6d75bb3af747de838e6ccc97c5d978/dea64b1832bbe603f7ddf70e41bc9e2c?sv=2018-03-28&sr=b&si=ReadPolicy&sig=QKpVqP5bpOJugPJLyh%2FxuS39RgkVVBaPT-BexdhbNcac%3D&st=2020-08-19T08%3A07%3A23Z&se=2020-08-20T08%3A17%3A23Z. Hakupäivä 3.6.2020.

Zhao, Wencang 2020. Blindly Selecting Method of Training Samples Baded Data's Intrinsic Character for Machine Learning. Saatavissa: https://ieeexplore.ieee.org/document/4216510. Hakupäivä 3.6.2020.

Zoph, Barret – Vasudevan, Vijay – Shlens, Jonathon – Le, V. Quoq 2018. LearningTransferableArchitecturesforScalableImageRecognition. Google. Saatavissa: https://arxiv.org/pdf/1707.07012.pdf. Hakupäivä 3.6.2020.