LAB University of Applied Sciences

Tekstiilin tunnistaminen koneoppimismallin avulla

LAB-ammattikorkeakoulu
Insinööri (YAMK), IoT:stä tekoälyyn
2023
Tomi Saari

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Tomi Saari	Opinnäytetyö, YAMK	2023
	Sivumäärä	
	54	

Työn nimi

Tekstiilin tunnistaminen koneoppimismallin avulla

Tutkinto ja koulutusala

Insinööri (YAMK), IoT:stä tekoälyyn

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

LAB-ammattikorkeakoulu

Tiivistelmä

Tämän opinnäytetyön tavoitteena oli toteuttaa mobiilisovellus sekä koneoppimismalli kankaan tunnistamiseen. Näiden avulla toteutui kokonaisuus, joka hyödyntää koneoppimismallia tunnistamaan onko otetussa kuvassa kude vai neule. Tavoitteena oli tutkia, onko tällainen tunnistaminen yleisesti ottaen mahdollista.

Mobiilisovelluksen teknologiaksi valikoitui Xamarin, koska se tukee kehittämistä Android ja iOS alustoille samalla koodipohjalla. iOS sovellus ei toteutunut tämän opinnäytetyön aikana, koska se olisi vaatinut Mac tietokoneen, eikä sellaista ollut saatavilla opinnäytetyöprosessin aikana. Android sovellus toimi koneoppimismallin käytön mahdollistajana.

Koneoppimisteknologiaksi valikoitui TensorFlow Lite. TensorFlow Litellä pystytään toteuttamaan Keraksen avustuksella nopeasti koneoppimismalli. Koneoppimismallin opettamiseen käytettiin LAB:n toimittamia valmiita kuvia, jotka oli luokiteltu valmiiksi.

Lopputuloksena syntyi Android sovellus, joka hyödyntää muodostettua TensorFlow Lite koneoppimismallia. Koneoppimismallin tarkkuus ei ollut riittävän hyvällä tasolla, että sitä voisi käyttää tuotantokäytössä. Loppupäätelmänä on, että tähän pitäisi pystyä käyttämään enemmän aikaa, kuin tämän opinnäytetyön kirjoittajalla oli mahdollista allokoida tämän kehittämiseen päivätyön ohessa, jotta tunnistamisen tarkkuus olisi riittävällä tasolla.

Asiasanat

koneoppiminen, kuvantunnistus, neule, kude, mobiilisovellus

Abstract

sAuthor(s)	Type of Publication	Published
Tomi Saari	Thesis, UAS	2023
	Number of Pages	
	54	

Title of Publication

Fabric identification with machine learning model

Degree, Field of Study

Master of Engineering (UAS), From IoT to AI

Organization of the client (if the thesis work is commissioned by another party)

LAB University of Applied Sciences

Abstract

Goal of this thesis was to create mobile application and machine learning model that can identify fabrics. These form a solution that uses machine learning model to identify if picture contains weft knit or warp knit. Goal was to research if this kind of identification is possible at all.

Xamarin was technology that was selected to develop the mobile app because it allows to develop Android and iOS applications with the same codebase. iOS app was left out during development because it requires Mac computer, and it wasn't available during the thesis process. Android app was enabler for using the machine learning model.

TensorFlow Lite was selected as a machine learning technology. TensorFlow Lite with Keras allows creating machine learning model quickly. Photos of weft knit, and warp knit was provided by LAB and those were used for training the machine learning model.

End result was Android app that uses trained TensorFlow Lite machine learning model. Accuracy of the machine learning wasn't so good that it would be ready for production use. Conclusion is that it would require more time than writer of this thesis could use while being on a full-time job to get good identification success rate.

Keywords

machine learning, image recognition, weft, knit, mobile application

Sisällys

1 Johdanto

Tämän opinnäytetyön tavoitteena oli kehittää mobiilisovellus sekä koneoppimismalli kankaiden ja kuteiden lajittelua varten. Mobiilisovelluksen tarkoitus on toimia esimerkkinä siitä, miten koneoppimista voidaan hyödyntää lajittelussa ja herättää mielenkiinto automaattisempaa kankaidenlajittelua varten ja sitä kautta mahdollisen hankerahoituksen hakemiseen.

1.1 Työn tavoitteet ja rajaus

Tavoitteena on kehittää mobiilisovellus, joka pystyy tunnistamaan, onko kuvassa kude vai neule. Tavoitteena on sellainen toteutus, jolla ei ole ylläpitokuluja. Tämän vuoksi toteutus tehdään sellaisena, että kuvat käsitellään paikallisesti käyttäjän laitteella, eikä lähetetä pilveen. Tavoitteena on toteuttaa mobiilisovellus Android ja iOS puhelimille.

Tunnistaminen on rajattu vain kuteiden ja neuleiden kategorioihin, eikä ole tarkoitus pystyä tunnistamaan laajempaa skaalaa kankaita. Ei myöskään ole tarkoitus pystyä tunnistamaan onko kuvassa t-paita, housut tai sukka. Tässä vaiheessa tunnistamisen reaaliaikaisuus ei ollut mukana suunnitelmassa.

1.2 Tutkimuskysymykset

Tämän opinnäytetyön tutkimuskysymyksinä olivat seuraavat:

- Millä teknologialla kankaita ja kuteita valokuvasta tunnistava mobiilisovellus on mahdollista tehdä?
- Mikä koneoppimismalli sopii parhaiten kankaiden ja kuteiden mobiiliin tunnistustehtävään?

Opinnäytetyön ensimmäisessä tutkimuskysymyksessä tarkastellaan avoimen lähdekoodin teknologioita, joilla voidaan toteuttaa monialustainen mobiilisovellus kuvantunnistamiseen valokuvasta. Toisessa tutkimuskysymyksessä puolestaan tarkastellaan teknologioita, joilla on mahdollista muodostaa koneoppimismalli niin, että mallia voidaan käyttää muun muassa mobiilisovelluksessa. Kuten mobiilisovelluksen osalta, myös koneoppimismallin teknologioissa pääpainoarvo on avoimessa lähdekoodissa. Vastaus näihin kysymyksiin etsitään soveltuvista lähteistä. Näiden vastausten pohjalta valitaan teknologiat mobiilisovelluksen sekä koneoppimismallin toteuttamiseen.

2 Tekstiilit

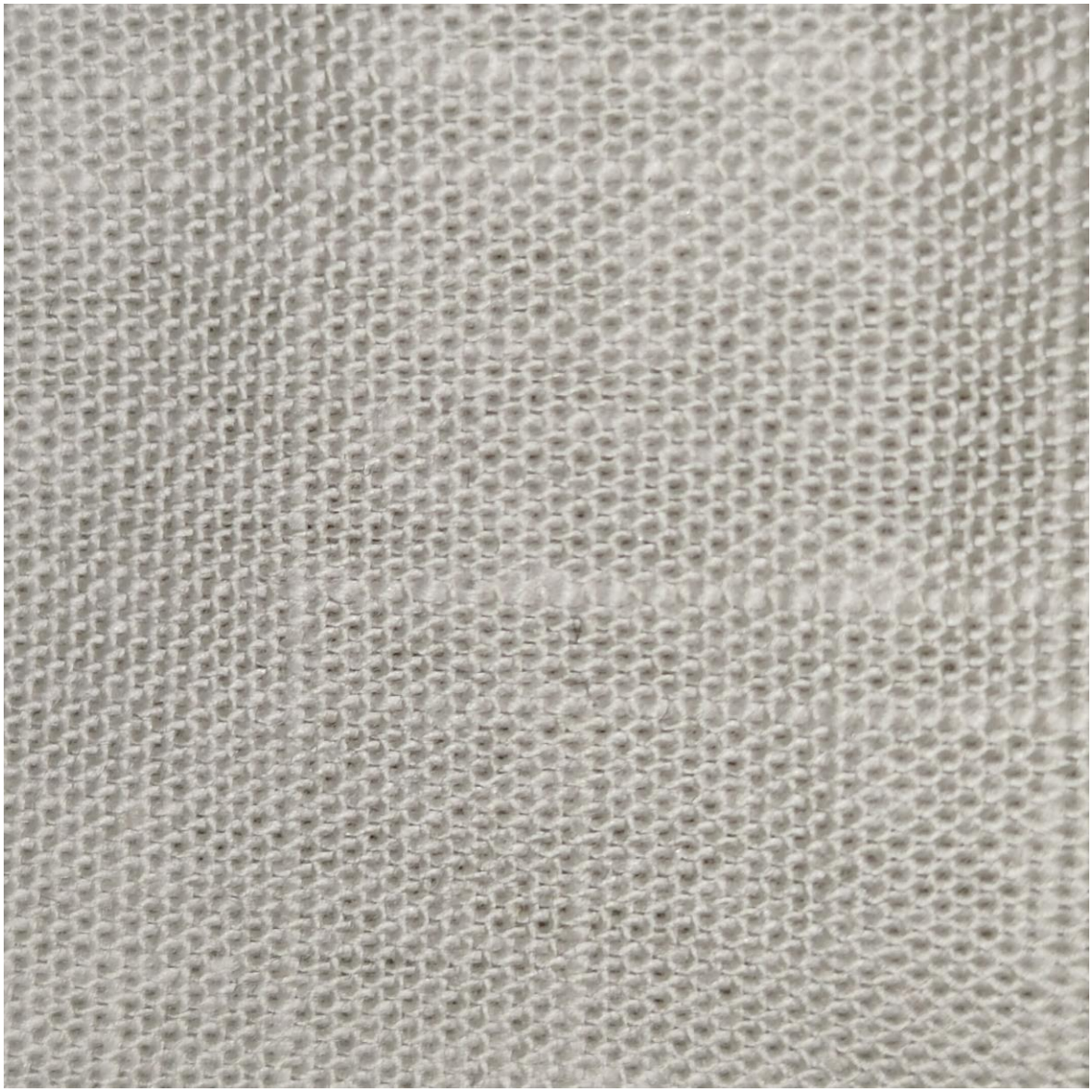
2.1 Kudos

Kuva 1. Lähikuva kudekankaasta

Kudekoneessa langansyöttö ja silmukanmuodostus tapahtuu jokaisella neulalla peräkkäin koko neulapedissä saman kudontakierroksen aikana, olivatpa neulat kiinteitä tai ei. Kaikille, tai osalle neuloja annetaan vuorotellen sama kudoslanka saman kudontasyklin aikana. Tämän vuoksi kudoslangan reitti kulkee koko kankaan läpi kulkien jokaisen neulalla muodostetun silmukan läpi. (Spencer 2011, 48.) Kuva 1 on puhelimella otettu lähikuva kudekankaasta.

Kudonta on monipuolisempi, laajemmalle levinnyt, verrattuna neulontaan. Kudontakoneet ovat suosittuja varsinkin pienien valmistajien keskuudessa, niiden monipuolisuuden, suhteellisen pienen alkupääoman ja pienen kokonsa vuoksi. Suuri osa kudontateollisuudesta on suoraan tekemisissä asusteiden koostamisen kanssa, käyttäen tekniikoita kuten overlock-ommel, saumaaminen ja linkitys, jotka ovat kehitetty nimenomaan tuottamaan samanlaisen lopputuloksen neulokseen verrattuna. (Spencer 2011, 49.)

2.2 Neulos



Kuva 2. Lähikuva neuloksesta

Neuloskoneessa on samanaikainen langansyöttö ja silmukan muodostus jokaisessa neulassa neulatangossa saman neulontasyklin aikana. Kaikki neulatangon neulat ovat

samanaikaisesti läpäisty erillisillä loimiohjaimilla. (Spencer 2011, 48.) Neulotut kankaat ovat neulottu yhtenäisellä leveydellä, vaikkakin on mahdollista neuloa useita kapeampia kankaita yhdellä neulapedillä ja kankaat erotellaan toisistaan viimeistelyn jälkeen. Kankaiden ominaisuuksia voidaan muuttaa viimeistelyprosessissa tai neulonnan aikana. (Spencer 2011, 50.) Kuva 2:ssä on puhelimella otettu kuva neuloksesta.

3 Mobiilisovelluksen teknologiavalinta

Projektin kehitys alkoi teknologiapinon valinnasta. Alkuperäinen toive oli, että projekti toteutettaisiin Android ja iOS yhteensopivaksi, joka asetti omat rajoitteensa mobiilisovelluksen teknologiavalinnalle. Monialustatuki on parempi toteuttaa teknologialla, jolla saadaan samalla koodipohjalla alustariippumattomia toimintoja.

Monialustatuki rajasi pois esimerkiksi puhtaan Java toteutuksen, koska sillä ei ollut kunnon edellytyksiä monialustaversion tekemiseen. Java on muutoinkin väistymässä Kotlin ohjelmointikielen alta Android kehitykseen suositeltuna ohjelmointikielenä. Kehittäjällä ei ollut aiempaa Kotlin kokemusta, joten sekin olisi ollut mielenkiintoinen lähestymiskulma.

iOS laitteille sopiva ohjelmointikieli olisi ollut Applen kehittämä Swift. Swift ohjelmointikieli on iOS natiivi ohjelmointikieli, joka rajattiin heti alussa pois monialustatuen puutteen vuoksi. Swift olisi myös vaatinut Mac -tietokoneen, jota ei ollut saatavilla tämän opinnäytetyön kirjoittamisen aikana.

React Native ja Ionic Angular viitekehykset olivat mielenkiintoisia vaihtoehtoja web-kehitys tyyppisen kehittämisen vuoksi. Nämä olisivat olleet myös hyviä vaihtoehtoja jatkokehityksen kannalta, koska niille löytyy osaajia paljon ympäri maailmaa. Angular on suosittu kieli verkkosivustojen kehittämiseen.

Tämän opinnäytetyön kirjoittajalla on pitkä työura takana Microsoft teknologioiden parissa, jonka vuoksi Microsoftin kehittämä Xamarin oli luonnollinen vaihtoehto ottaa mukaan mobiilisovelluksen teknologiavertailuun. Opinnäytetyön kirjoittajalla on aiempaa kokemusta C# -ohjelmointikielestä muun muassa ohjelmointirajapinnan kehittämisen kautta. Kirjoittajalla on myös kokemusta Azuresta, mutta kustannusten vuoksi pilviratkaisut jäivät suunnittelun asteelle.

Määritysten tarkennuttua, myös pelienkehittämiseen tarkoitettu Unity olisi ollut hyvä vaihtoehto sovelluksen kehittämiseen. Unity voisi toteuttaa lisätyn todellisuuden version, jolla tunnistetaan videosyötteestä esine (tässä yhteydessä liukuhihnalla oleva kangas) ja tunnistaa reaaliajassa useita eri kankaita. Jos sovellusta halutaan jatkokehittää, olisi hyvä tutkia Unityn mahdollistamaa reaaliaikaista esineiden tunnistamista ja sen soveltuvuutta tähän käyttötarkoitukseen. Unityllä voi päästä lähelle sitä ratkaisua, joka vastaisi lopullista käyttötarkoitusta.

3.1 Xamarin

Vuoden 2000 kesäkuussa Ximian niminen yritys aloitti avoimenlähdekoodin projektin, nimeltään Mono, joka on vaihtoehtoinen toteutus C# -kääntäjälle ja .NET viitekehykselle, joka

toimii Linuxilla. Vuosikymmen myöhemmin, vuonna 2011 Ximianin perustajat perustivat Xamarinin, joka edelleen jatkokehittää Monoa, mutta käyttää Monoa perustana mobiilipuolen monialustatuelle. Vuonna 2014 C# ja .NET viitekehys kokivat muutoksia, jotka lupasivat hyvää Xamarinin tulevaisuudelle. C#:sta julkaistiin avoimen lähdekoodin versio, nimeltään .NET Compiler Platform. Samalla julkaistiin .NET säätiö avoimen lähdekoodin .NET teknologioille, joissa Xamarinilla oli iso rooli. Microsoft osti Xamarinin vuoden 2016 maaliskuussa, tavoitteenaan tuoda mobiilipuolen monialustatuki laajemmalle Microsoft kehittäjä yhteisölle. Xamarin.Forms on nykyään Visual Studio käyttäjien vapaasti käytettävissä. (Petzold 2016, 5.)

Kolmena ensimmäisenä vuotena Xamarin keskittyi pääasiallisesti kääntäjäteknologiaan, sekä kolmeen .NET -kirjastoon. Yksi näistä oli Xamarin.Mac, joka kehittyi MonoMac -projektista. Toinen oli Xamarin.iOS, joka kehittyi MonoTouchista. Kolmas oli Xamarin.Android, joka kehittyi MonoDroidin pohjalta. Nämä kirjastot tunnetaan Xamarin alustana. Kirjastot koostuvat natiivi Mac, iOS ja Android ohjelmistorajapintojen .NET versioista. Ohjelmoijat voivat kehittää ohjelmia C# -ohjelmointikielellä käyttäen näitä alustoja, lisähyötynä on se, että voidaan käyttää lisäksi .NET viitekehyksen luokkakirjastoja. Ohjelmistokehittäjät voivat käyttää Visual Studiota Xamarin ohjelmien kehittämiseen iOS, Android ja eri Windows alustoille. Jotta iPhone ja iPad kehitys on mahdollista, tarvitsee samasta lähiverkosta löytyä Mac -tietokone. (Petzold 2016, 5-6.)

Mac -tietokonevaatimus oli syy, miksi iOS -versio jäi tässä vaiheessa kehittämättä / testaamatta. Mac -tietokoneet ovat arvokkaita, eikä sellaista ollut käytössä tämän kehitystyön aikana. Tämä on harmillista, koska iOS -versio olisi ollut kuitenkin hyvä kehittää samalla, laajemman käyttäjäkunnan vuoksi.

3.2 Xamarinin hyödyt

Xamarin valikoitui tämän työn teknologiaksi, niin hyötyjä tarkastellaan sen näkökulmasta. Xamarin mahdollistaa iOS ja Android sovelluksen kehittämisen yhdellä ohjelmointikielellä. Joitakin samoja hyötyjä olisi ollut muillakin teknologoilla.

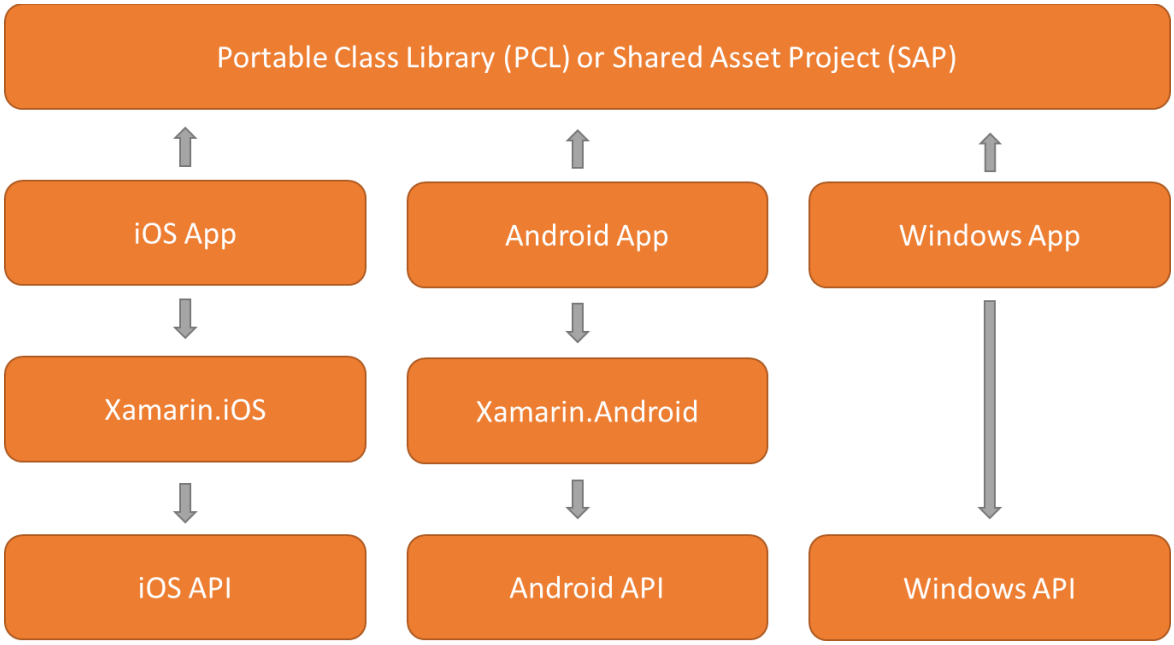
Monelle alustalle suunnatussa kehittämisessä on etuna se, että samaa koodia voidaan hyödyntää alustariippumattomasti, joka tuo modulaarisuutta ja ylläpidettävyyttä sovellusprojektiin. Ennen kuin koodia voidaan jakaa, on sovellus strukturoitava tähän tarkoitukseen. Erityisesti, koska kyseessä on graafiset käyttöliittymät, on ohjelmoitavan ymmärrettävä ohjelmakoodi jakaminen eri kerroksiin. Ehkäpä hyödyllisin lajittelu on erotella käyttöliittymäkoodi tietomalleista ja algoritmeista. Suosittu Model-View-Controller (MVC) sovellusarkkitehtuuri

lajittelee koodin kolmeen osaan. Model -kerros on taustalla oleva data, View -kerros on datan visuaalinen esitys ja Controller -kerros hoitaa käyttäjän syötteen. (Petzold 2016, 6.)

Model-View-Controller sai alkunsa 1980-luvulla. Myöhemmin Model-View-ViewModel (MVVM) malli on modernisoinut MVC-mallia. MVVM erottelee koodin myös kolmeen osaan. Model -kerros on taustalla oleva data, View -kerros on käyttöliittymä, sisältäen visuaalisuuden ja käyttäjän syötteet ja ViewModel -kerros hallitsee tiedonvälityksen Model ja View -kerrosten välillä. Kun kehitetään sovellusta, joka on suunnattu monelle alustalle, MVVM arkkitehtuuri helpottaa kehittäjää jakamaan koodin alustakohtaiin näkymiin ja koodeihin, jotka eivät ole alustariippuvaisia. Monesti koodin, joka ei ole alustariippuvaista, tarvitsee esimerkiksi pääsyn tiedostoihin, verkkoon tai säikeisiin. Normaalisti nämä mielletään osaksi käyttöjärjestelmän ohjelmointirajapintaa, mutta ne ovat toimintoja, joita voidaan käyttää .NET viitekehys luokkakirjaston kautta ja jos .NET on saatavilla jokaiselle alustalle, niin koodi on silloin alustariippumatonta. (Petzold 2016, 6.)

Alustariippuvaiset sovelluskoodit voidaan eriyttää, eli Visual Studion tapauksessa laittaa eri projekteihin. Tämä voidaan tehdä Shared Asset Projektilla (SAP), joka sisältää koodin ja muut tiedostot, joita tarvitaan eri projekteissa tai Portable Class Libraryllä (PCL), joka sisältää yhteiset koodit jaetun kirjaston kautta, jota voidaan käyttää muissa projekteissa. Riippumatta toteutustavasta, yhteisellä koodilla pitää olla pääsy .NET viitekehyksen luokkakirjastoon, jotta se voi hoitaa tiedostojen lukemisen/kirjoittamisen, verkkopalveluliikenteen ynnä muut tarpeelliset toiminnot. Tämä tarkoittaa, että voidaan luoda yksi Visual Studio projektikokonaisuus, jossa on kolme C# -projektia, yksi iOS -alustalle ja yksi Android -alustalle, sekä yhteinen PCL tai SAP-projekti. (Petzold 2016, 6.)

Kuva 3 havainnollistaa Visual Studio projektien, Xamarin kirjastojen ja alustan ohjelmointirajapintojen keskinäiset suhteet. Kolmas sarake kuvastaa minkä tahansa Windows alustan laitetta. Toisen rivin laatikot ovat alustakohtaiset sovellukset. Nämä sovellukset tekevät kutsuja yhteiseen projektiin sekä Xamarin kirjastoihin, jotka toteuttavat natiivin alustan ohjelmointirajapinnat. (Petzold 2016, 7.)



Kuva 3. Visual Studio projektien, Xamarin kirjastojen ja alustan ohjelmointirajapintojen keskinäiset suhteet (mukailtu Petzold 2016, 7)

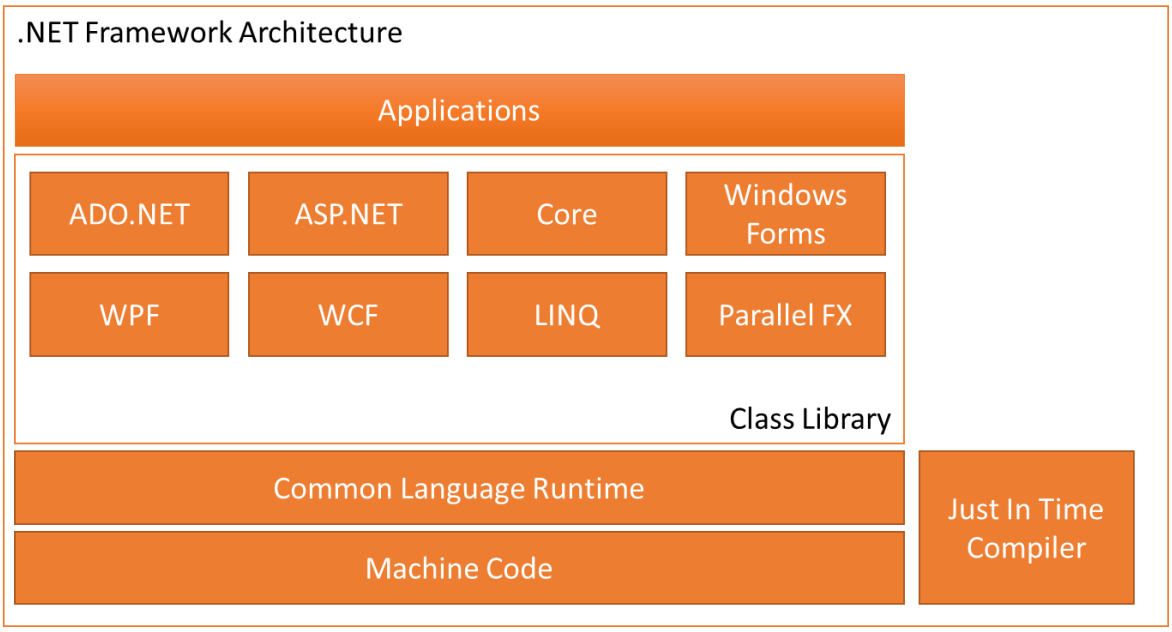
Kuva ei kuitenkaan ole täydellinen, se ei näytä SAP tai PCL kutsuja .NET viitekehyksen luokkakirjastoon. Se, mikä on vaadittu -NET versio, riippuu yhteisen koodista, PCL:lla on pääsy omaan .NET versioon, kun taas SAP käyttää alustaan sisällytettyä .NET versiota. Kuva 3 Xamarin.iOS ja Xamarin.Android kirjastot näyttävät olennaisilta ja vaikka ne ovatkin tärkeitä, ne ovat pääasiassa vain kieleen sidottuja ohjelmointirajapintakutsuja. (Petzold 2016, 7.)

Kun iOS ohjelma käännetään, Xamarin C# -kääntäjä kääntää koodin C# Intermadiate Language (IL) kuten normaalistikin, mutta käyttää Apple -kääntäjää Mac tietokoneissa, kääntäessään koodin natiiviksi iOS konekoodiksi, kuten Objective-C kääntäjä. Sovelluksen kutsut iOS ohjelmointirajapintaan ovat samoja, kuin Objective-C-ohjelmointikielellä kehitetyt ohjelmat. Android ohjelmille Xamarin C# -kääntäjä generoi IL-kielelle, joka ajetaan Monolla Java moottorin rinnalla, mutta ohjelmointirajapintakutsut ovat käytännössä samoja, kuin Javalla kehitetyillä sovelluksilla. Mobiilisovelluksille, jotka tarvitsevat alustakohtaista koodia sekä jaettua koodia, Xamarin.iOS ja Xamarin.Android tarjoaa loistavan ratkaisun. Niiden ansiosta kehittäjällä on pääsy alustan ohjelmointirajapintoihin ja kaikkeen mitä se mahdollistaa. (Petzold 2016, 7-8.)

3.2.1 .NET

Vuonna 2002 julkaistu .NET on ohjelmistokehityksen viitekehys, jota voidaan ohjelmoida monilla eri kielillä, kuten C#, ASP.NET, C++, Python, Visual Basic ja F#. .NET viitekehys

tarjoaa yhteen toimivuuden monille eri ohjelmointikielille. Ohjelmat, jotka ovat kehitetty .NET viitekehyksellä suoritetaan joko ympäristössä, tai virtuaalikoneella Common Language Runtime komponentilla. (Simaranjit & SrinivasMadhav 2019, 15.)



Kuva 4. .NET viitekehyksen arkkitehtuuri (mukailtu Simaranjit & SrinivasMadhav 2019, 15)

Kuva 4 esitetyn .NET viitekehyksen arkkitehtuurissa hierarkian ylimmällä tasolla on sovellus, joka on kehitetty .NET:llä. Sovellus voi olla yksinkertainen konsoliohjelma, joka kirjoittaa näytölle 'Hei maailma!', tai monimutkaisempi ohjelmistokokonaisuus. Sovellukset pohjautuvat joukkoon luokkia tai suunnittelumalleja. Common Language Runtime käyttää Just In Time kääntäjää muuntaakseen sovelluskoodin konekielelle. Konekieli on spesifi ajoympäristölle ja eroaa Linuxin ja Windowsin välillä. (Simaranjit & SrinivasMadhav 2019, 15.)

3.2.2 C#

Microsoft julkaisi C# ohjelmointikielen vuonna 2000. C# on suhteellisen uusi ohjelmointikieli, varsinkin verrattuna Objective-C:n tai Javaan. Alkuun C# vaikutti olevan suoraviivainen, tiukasti tyypitetty, olio-ohjelmointikieli, joka on saanut vaikutteita C++ ja Javasta, mutta puhtaammalla syntaksilla kuin C++ ja ilman historian painolastia. C# on kasvanut ja parantunut vuosien mittaan. Geneeristen lambda funktioiden, LINQ, ja asynkronisten operaatioiden tuki ovat muuntaneet C# niin, että se luokitellaan nykyään moniparadigmaiseksi ohjelmointikieleksi. (Petzold 2016, 4.)

Julkaisustaan lähtien, C# on läheisesti assosioitu Microsoft .NET viitekehyksen kanssa. Alimmalla tasolla .NET tarjoaa C# perustietotyypit (esimerkiksi int, double ja string). Laaja .NET viitekehys luokkakirjasto tarjoaa tuen monille eri toiminnoille, joita ohjelmoinnissa

tulee vastaan, kuten muun muassa matematiikka-, virheenselvitys-, kokoelma-, tiedostonkäsittely- ja verkko-operaatioluokat. (Petzold 2016, 4.)

3.3 Vaihtoehtoiset teknologiat

Ennen kuin mobiilisovelluksen teknologia valikoitui, tutkittiin muita vaihtoehtoja toteutukseen. Enemmän web-kehitystyyliset ohjelmointikielet, kuten React ja Angular, ovat suuressa suosiossa nykypäivänä. Ottaen huomioon sen, että React ja Angular osaajia on maailmassa paljon, olisi hyvä tutkia näiden tekniikoiden soveltuvuus esimerkiksi pienimuotoisen soveltuvuusselvitys projektin kautta.

3.3.1 Kotlin

Kotlin oli yksi mielenkiintoinen vaihtoehto monialustaisen mobiilisovelluksen kehittämiseen. Syitä tähän on muun muassa se, että tämän opinnäytetyön kirjoittaja oli aiemmin tehnyt kevyitä mobiilisovelluksia Androidille Javalla. Kotlin on suositeltu ohjelmointikieli Androidille ja siinä on samanlaisia piirteitä Javan kanssa.

Android tiimi ilmoitti, että Kotlin on nyt virallinen Android sovellusten virallinen ohjelmointikieli, Google I/O 2017 tapahtumassa. Tämä tarkoittaa sitä, että vaikka Android sovelluksia voi kehittää jatkossakin Javalla, Kotlin on täysin tuettuna ja Google julkaisee kaikki uudet Android ominaisuudet, kehitysympäristön ynnä muut täydellä Kotlin tuella. (Leiva 2017, 5.)

Kotin on JetBrains yrityksen kehittämä Java Virtual Machine pohjainen ohjelmointikieli. JetBrains on tunnettu mm. Javalle suunnatun IntelliJ IDEA ohjelmointiympäristön kehittäjänä. IntelliJ on käytetty Androidin virallisen ohjelmointikehitysympäristön Android Studion pohjana. (Leiva 2017, 5.)

Kotlin kehitettiin Java kehittäjiä silmällä pitäen. Kotlin on intuitiivinen ja helppo oppia aiemmin Java ohjelmointikokemuksen pohjalta. Kotlin toimii ilmaisella Android Studiolla, joten hinta ei nouse esteeksi. (Leiva 2017, 5.)

Syy miksi Kotlin ei valikoitunut tässä vaiheessa teknologiaksi, oli se, että Kotlinin monialustatuki oli vielä kirjoitushetkellä Alpha vaiheessa. Tämä tarkoittaa sitä, että monialustatuen kehitys on vielä niin varhaisessa vaiheessa, että ominaisuudet voivat muuttua ja kaikki toiminnallisuudet eivät välttämättä toimi täysin oikein. Jos projektia halutaan myöhemmin jatkokehittää ja Kotlinin monialustatuki on kypsemmässä vaiheessa, voisi se olla hyvä vaihtoehto ohjelmointikieleksi.

3.3.2 React Native

React Native oli mukana mahdollisten teknologioiden osalta sen vuoksi, että sille löytyy paljon osaajia ympäri maailman. Tämä tarkoittaisi sitä, että web-kehittäjän olisi helppoa päästä mukaan kehittämään mobiiliohjelmaa. React osaaminen olisi myös hyödyllinen kieli osata muiden lisäksi.

Täysiverisen mobiilisovelluksen kehittäminen voi olla monimutkaista. Monimutkaisten ympäristöjen, monipuolisten viitekehysten ja pitkien sovelluskäännösaikojen takia puhtaiden mobiilisovellusten kehittäminen ei välttämättä ole helppo työ. Tämän vuoksi markkinoille on tullut useita eri ratkaisuja, jotka yrittävät ratkaista nämä ongelmat. (Dabit 2019, 3.)

Monimutkaisuuden ytimessä on monialustatuen kehittäminen. Eri alustat ovat sisimmiltään erilaisia, eikä sisällä yhteneväisyyksiä kehitysympäristön, ohjelmointirajapinnan tai koodin osalta. Tämän vuoksi yrityksillä voi olla eri tiimit kehittämässä ohjelman Android versiota kuin ohjelman iOS versiota. (Dabit 2019, 3.)

React Nativen avulla pystytään kehittämään monialustaisia mobiilisovelluksia, jotka toimivat lähes yhtä tehokkaasti, kuin alustakohtainen mobiilisovellus. Yksi koodipohja mahdollistaa sen, että yksi tiimi kehittää sovellusta kaikille alustoille. React Native on viitekehys, jonka avulla voidaan kehittää natiiveja mobiilisovelluksia JavaScriptillä, käyttäen JavaScriptin React -kirjastoa. React Native ohjelma kääntyy oikeaksi mobiilisovellukseksi. React on Facebookin kehittämä ja käyttämä avoimen lähdekoodin JavaScript kirjasto. (Dabit 2019, 3-4.)

React Native tarjoaisi hyvän pohjan monialustatuelle. Jos projektia haluttaisi jatkokehittää enemmän mobiilialustoille, sen sijaan, että se toimisi enemmän autonomisesti liukuhihnalla, olisi React Native hyvä vaihtoehto. Opinnäytetyön kirjoittajalla on kollegoiden kautta muodostunut käsitys, että React Native osaajia löytyy Suomestakin paljon, joka helpottaisi kehittäjien löytämistä projektiin.

3.3.3 Ionic Angular

Ionic Angular oli mukana mahdollisten teknologioiden listalla sen vuoksi, että tämän opinnäytetyön kirjoittajalla on työkokemusta Angular kehityksestä web-sivuston kehittämisen myötä. Angular on kirjoittajan oman kokemuksen pohjalta sellainen kieli, millä pääsee nopeasti alkuun. Se on myös hyvin tuettu ja siihen löytyy paljon materiaalia, ongelmatilanteiden ratkaisemiseksi.

Ionic on monipuolinen viitekehys monialustaisten mobiilisovellusten kehittämiseen. Se on julkaistu avoimena lähdekoodina. Ionic komponentit ovat kehitetty mukautettuina

elementteinä omalla Stensil -työkalulla, joka on myös avoimen lähdekoodin projekti. Ionic on suosittu viitekehys mobiilikehitykseen. (Cheng 2018, 5.)

On useita syitä miksi Ionic on suosittu viitekehys. Se pohjautuu web-komponentti standardeihin ja on viitekehys riippumaton, eli sitä voidaan käyttää myös ilman Angularia. Web-komponentit ovat W3C spesifikaation mukaisia komponentteja web-alustalle. Ionicia voidaan käyttää Angularilla, Reactilla ja Vue viitekehyksillä. Ionic käyttää Apache Cordovaa mobiilisovelluksen ajoympäristönä. Ionic Native helpottaa Cordovan laajennusten käytön Ionic ohjelmissa. Ionicin suorituskyky on hyvä mobiililaitteilla. (Cheng 2018, 5.)

Angular on Googlen Angular -tiimin Microsoft TypeScriptillä kehittämä sovellusarkkitehtuuri, joka pohjautuu JavaScriptiin. Angularilla kehitetyt ohjelmat perustuvat arkkitehtuuri suunnitteluun, joka koostuu komponenteista, jotka keskustelevat keskenään I/O-rajapintojen välityksellä. (Bampakos & Deeleman 2020, 6.)

Yksi syy miksi Ionic Angular yhdistelmä ei valikoitunut teknologiaksi, oli se, että kehittäjän omakohtaisten kokemuksien perusteella virhetilanteiden selvittäminen voi olla haastavampaa Angularilla kuin muilla ratkaisuilla. Tällä tarkoitetaan sitä, että virheviestit eivät ole aina kovin loogisia ja virheen paikallistaminen voi olla työlästä. Toki virheiden paikallistaminen nopeutuu, mitä enemmän on kokemusta Angularista ja web-kehittämisestä.

4 Koneoppiminen

4.1 Syväoppiminen

Syväoppiminen on tekoälyn alalaji, joka keskittyy luomaan suuria neuroverkkomalleja, jotka ovat kyvykkäitä tekemään tarkkoja päätöksiä datan pohjalta. Syväoppiminen soveltuu erityisen hyvin sellaisiin tapauksiin, joissa data on monimutkaista ja tapauksiin, joista on saatavilla paljon dataa. Nykypäivänä monet verkkoyritykset ja huippuluokan kuluttajaelektroniikkavalmistajat käyttävät syväoppimista. Muiden asioiden lisäksi Facebook käyttää syväoppimista tekstin analysoimiseen verkkokeskusteluista. Google, Baidu ja Microsoft käyttävät syväoppimista kuvahaussa, sekä kielikäännöksissä. Kaikissa moderneissa älypuhelimissa on syväoppimisjärjestelmä, esimerkiksi syväoppiminen on nykypäivän standarditeknologia puheentunnistamiseen ja sitä käytetään myös kameroissa kasvojen tunnistamiseen. Terveydenhuoltosektorilla syväoppimista käytetään muun muassa. röntgenkuvien prosessointiin ja terveysongelmien diagnosointiin. Syväoppiminen on myös itseajavien autojen keskeisessä osassa, jossa sitä käytetään lokalisaatioon, kartoittamiseen, liikkeiden suunnitteluun ja ohjaamiseen sekä ympäristön havainnointiin ja kuljettajan valmiuden seuraamiseen. (Keller 2019, 9.)

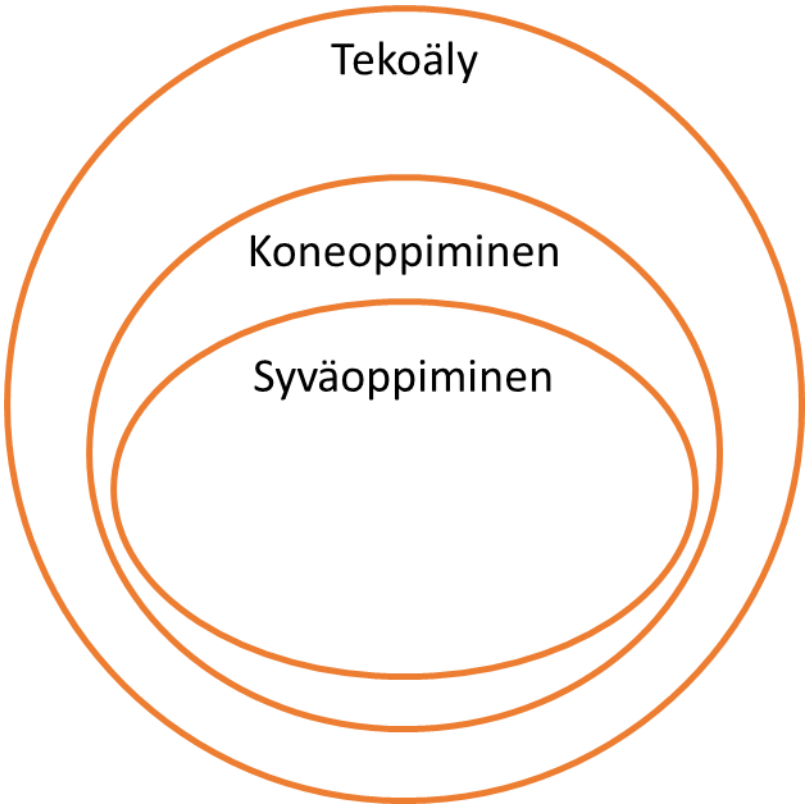
Ehkä tunnetuin esimerkki syväoppimisesta on DeepMindin AlphaGo. Go on shakin kaltainen peli. AlphaGo oli ensimmäinen tietokoneohjelma, joka voitti ammattilaisen Go pelaajan. Vuoden 2016 maaliskuussa se voitti korealaisen ammattilaisen, Lee Sedolin, kilpailussa, jota katsoi yli 200 miljoona ihmistä. Seuraavana vuonna AlphaGo voitti maailman parhaan pelaajan,kiinalaisen Ke Jienin. (Keller 2019, 9.)

AlphaGo käyttää syväoppimista arvioidakseen laudan konfiguraatiot ja päättääkseen minkä siirron se tekee seuraavaksi. Fakta, että AlphaGo käyttää syväoppimista seuraavan päätöksen tekoon on vinkki siitä, miksi syväoppiminen on hyödyllinen monilla eri aloilla ja käyttökohteilla. Päätöksenteko on tärkeä osa elämää. Yksi tapa tehdä päätöksiä on intuitio. Kuitenkin monet ihmiset pitävät parhaana päätöksentekomallina sitä, että päätös pohjautuu relevanttiin tietoon. Syväoppiminen mahdollistaa päätöksenteon tiedon pohjalta tunnistamalla ja poimimalla kaavoja isoista tietojoukoista, jotka määrittelevät hyvän päätöksen lähtödatan pohjalta. (Keller 2019, 9.)

4.2 Tekoäly, koneoppiminen ja syväoppiminen

Syväoppiminen on saanut alkunsa tekoäly- ja koneoppimistutkimusten pohjalta. Kuva 5 esittää, miten tekoäly, koneoppiminen ja syväoppiminen ovat relaatiossa toisiinsa nähden. Tekoälyala syntyi Dartmouthin yliopiston työpajassa kesällä 1956. Työpajassa tutkittiin

monia eri aiheita, kuten matemaattisen teorioiden todistamista, luonnollisen kielen prosessointia, sekä pelien ja ohjelmien suunnittelua, jotka voisivat hyötyä esimerkeistä ja neuroverkoista. Moderni tekoälyala pohjautuu viimeksi mainittuihin osa-alueisiin. (Keller 2019, 9.)



Kuva 5. Tekoälyn, koneoppimisen ja syväoppimisen relaatio (mukailtu Keller 2019, 10)

Koneoppiminen sisältää sellaisten funktioiden kehittämistä ja arviointia, jotka pystyvät oppimaan tietojoukosta (esimerkeistä). Jotta voimme ymmärtää mitä koneoppiminen tarkoittaa, täytyy ymmärtää kolme termiä: tietojoukko, algoritmi ja funktio. (Keller 2019, 10.)

4.2.1 Tietojoukko

Yksinkertaisimmillaan tietojoukko on taulukko, jossa jokainen rivi sisältää tietoa osa-alueesta ja jokainen sarake sisältää tietoa kyseisestä osa-alueesta. Taulukko 1 on kuvattuna tietojoukko lainanantajan näkökulmasta. Tässä tietojoukossa on neljän lainanhakijan tiedot. Tunniste -kenttää lukuun ottamatta, joka on mukana rivin yksilöimisen vuoksi, jokaisella rivillä on hakijan vuositulot, nykyinen lainanmäärä ja luottokyky. (Keller 2019, 10.)

Tunniste	Vuositulot	Nykyinen lainamäärä	Luottokyky
1	150 €	-100 €	100
2	250 €	-300 €	-50
3	450 €	-250 €	400
4	200 €	-350 €	-300

Taulukko 1. Tietojoukko lainanhakijoista ja heidän luottokyvystään (mukailtu Keller 2019, 26-27)

4.2.2 Algoritmi

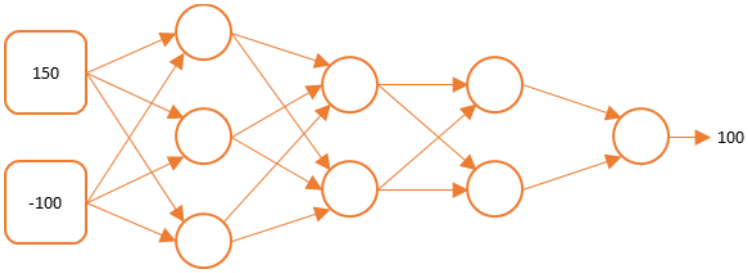
Algoritmi on prosessi, ”resepti” tai ohjelma, jota tietokone noudattaa. Koneoppimisen näkökulmasta algoritmi määrittää prosessin, jolla tietojoukko analysoidaan ja jolla tunnistetaan toistuvat kaavat datan pohjalta. Esimerkiksi algoritmi voi etsiä kaavoja, jotka ovat relaatiossa henkilön vuositulojen, nykyisen lainan määrän ja luottokyvyn välillä. Matematiikassa tällaisia relaatioita kutsutaan funktioiksi. (Keller 2019, 10.)

4.2.3 Funktio

Funktio on annettujen sisääntuloarvojen deterministinen kartoitus yhdeksi tai useammaksi ulostuloksi. Deterministinen kartoitus tarkoittaa sitä, että samoille sisääntuloarvoille palautuu aina samat ulostuloarvot. Esimerkiksi summaus on deterministinen kartoitus ja 2+2 on aina 4. Voimme luoda funktioita aloille, jotka ovat monimutkaisempia kuin perinteinen aritmetiikka. Voidaan esimerkiksi määritellä funktio, joka ottaa henkilön vuositulon ja lainanmäärän sisääntulona ja palauttaa luottokyvyn ulostuloarvona. Funktio konseptina on todella tärkeä syväoppimisessa, joten on hyödyllistä kerrata määritelmä: funktio on yksinkertaistetusti sisääntulojen kartoitus ulostuloarvoiksi. Itseasiassa, koneoppimisen määränpää on opettaa funktiot datan pohjalta. Funktio voidaan esittää monella eri tavalla: se voi olla yksinkertainen aritmeettinen operaatio, sarja mitä-jos-sitten sääntöjä, tai se voi olla paljon monimutkaisempi representaatio. (Keller 2019, 10.)

Yksi tapa esittää funktio on käyttää neuroverkkoa. Syväoppiminen on koneoppimisen alalaji, joka keskittyy syviin neuroverkkomalleihin. Itse asiassa kaavat, joita syväoppimisalgoritmit tuottavat tietojoukosta, ovat funktioita, jotka ovat esitettynä neuroverkkona. Kuva 6 esittää neuroverkon rakenteen. Vasemmassa reunassa olevat laatikot esittävät muistipaikkaa, jossa sisääntulot ovat esitettynä verkolle. Kuvan jokainen ympyrä on nimeltään neuroni ja jokainen neuroni toteuttaa funktion: se ottaa sisään numeraalisen arvon ja kartoittaa ne

ulostuloarvoiksi. Nuoli esittää sitä, miten jokaisen neuronin ulostuloarvo välitetään toisen neuronin sisääntuloon. Neuroverkossa informaatio liikkuu vasemmalta oikealle. Esimerkiksi jos neuroverkko on opetettu ennustamaan henkilön luottokyky vuositulojen ja lainan perusteella, se saisi sisääntulossa tulot ja lainan neuroverkon vasemmassa laidassa ja oikeasta reunasta olevasta neuronista tulee ulos luottokyky. Neuroverkko käyttää neuroverkon funktioiden opettamiseen hajota ja hallitse -strategiaa: jokainen neuroverkon neuroni oppii yksinkertaisen funktion ja kokonaisuutenaan monimutkaisempi funktio, joka määritellään tämän verkon pohjalta, luodaan yhdistämällä nämä yksinkertaisemmat funktiot. (Keller 2019, 10-11.)



Kuva 6. Neuroverkon skemaattinen kuva (mukailtu Keller 2019, 11)

4.3 Koneoppiminen

Koneoppimisalgoritmi on prosessi, jolla etsitään paras funktio olemassa olevien funktioiden koosteesta, jolla voidaan parhaiten selittää tietojoukon piirteiden suhteet. Jotta ymmärtäisimme paremmin, funktioiden opettaminen datan pohjalta tarkoittaa, esitetään tämä esimerkkien kautta. Näiden esimerkkien kautta päätetään mikä aritmeettinen operaatio (summaus, erotus, kerto- tai jakolasku) sopii parhaiten sisään- ja ulostuloarvojen selittämiseen:

funktio(sisääntuloarvot) = ulostuloarvo
funktio(5,5) = 25
funktio(2,6) = 12
funktio(4,4) = 16
funktio(2,2) = 04

Monien mielestä kertolaskutoimitus on paras ratkaisu, jolla voidaan määritellä suhteet tai kartoitus sisään- ja ulostulojen kesken:

5 * 5 = 25
2 * 6 = 12
4 * 4 = 16
2 * 2 = 04

Tässä esimerkissä parhaan funktion löytäminen on suhteellisen suoraviivaista ja ihminen pystyy tekemään tämän ilman tietokoneen avustusta. Kun sisääntuloarvojen määrä ennalta tuntemattomaan funktioon lisääntyy (ehkä jopa satoihin tai tuhansiin sisääntuloarvoihin) ja

potentiaalisten funktioiden määrä kasvaa, tulee tehtävästä liian vaikea ihmiselle. Näissä tapauksissa koneoppimisen potentiaalin valjastaminen parhaan funktion etsimiseen tietojoukon kaavojen yhdistämiseen tulee pakollista. (Keller 2019, 12.)

Koneoppiminen koostuu kahdesta askeleesta, opettamisesta sekä varmistamisesta. Opettamisessa koneoppimisalgoritmi prosessoi tietojoukon ja valitsee funktion, joka parhaiten sopii datan kaavoihin. Tuotettu funktio ohjelmoidaan ohjelmaan tietyssä muodossa (kuten jos-sitten-muuten tai tietyn kaavan parametreihin). Ohjelmoitu funktio tunnetaan nimellä malli ja tätä prosessia kutsutaan usein mallin opettamiseksi. Mallit ovat käytännössä funktioita tietokoneohjelmina. Koneoppimisessa funktioiden ja mallien konseptit ovat niin lähellä toisiaan, että termejä voidaan jopa käyttää saman asian tarkoittamiseen. (Keller 2019, 12.)

Syväoppimisen kontekstissa funktioiden ja mallien välinen relaatio on se, että tietojoukosta opetusvaiheessa toteutetut funktiot esitetään neuroverkkomallina ja päinvastoin neuroverkkomalli toteuttaa funktion tietokoneohjelmana. Tavanomainen neuroverkon opettaminen tapahtuu aloittamalla opettaminen neuroverkolla, jossa parametrit ovat alustettu satunnaisesti. Tämä satunnaisesti alustettu verkko on todella epätarkka, kun puhutaan sen kyvystä yhdistää suhteita eri sisääntuloarvojen ja ulostuloarvojen välillä esimerkiksi tietojoukosta. Tämän jälkeen opettaminen tapahtuu iteroimalla eri esimerkkitietojoukkoja ja jokaisessa esimerkissä annetaan sisääntuloarvot neuroverkolle ja pyritään saamaan tietojoukosta löytyvän halutun tuloksen päivittämällä neuroverkon parametreja, jotta se olisi lähempänä oikeaa tulosta. Kun koneoppimisalgoritmi on löytänyt funktion, joka on riittävän tarkka (sisääntuloarvot johtavat oikeaan ulostuloarvoon) ongelmaan, jota koneoppimismallilla halutaan ratkaista, opetusprosessi on valmis ja algoritmi palauttaa lopullisen mallin. Tässä vaiheessa koneoppimismallin opetus loppuu. (Keller 2019, 12.)

Kun opetus on valmis, malli on kiinteä. Koneoppimisen toinen vaihe on varmistaminen. Tällä tarkoitettaan sitä, että kun koneoppimismalli on valmis ja sitä käytetään sellaisen datan kanssa, jota malli ei ole aiemmin nähnyt ja haetaan sille mallin avulla oikea ulostuloarvo. Suurin osa koneoppimiseen käytetystä ajasta menee siihen, että opetetaan malli ja miten siitä saadaan tarkka (miten saadaan oikeat funktiot datasta). Tämä sen takia, koska taidot ja tavat koneoppimismallin tuotantoon julkaisussa ja jota käyttö laajalla skaalalla vaatii, ovat sellaisia, mitä ei löydy tyypilliseltä datatieteilijältä. Alalla on tunnistettu taidot, joita koneoppimismallin tuotantoon vienti laajalla skaalalla vaatii ja se näkyy mielenkiintona DevOps suuntautumiseen. Termillä kuvataan kehitys- ja operatiivisen -tiimin yhteistyötä (operatiivinen tiimi vie koneoppimismallin tuotantoon ja varmistaa, että kaikki toimii luotettavasti ja on skaalattavissa). Termit, MLOps koneoppimisoperaatioille ja AIOps tekoälyoperaatiolle kuvaavat myös haasteita opetetun mallin julkaisemiselle. (Keller 2019, 13.)

Miksi funktioiden tuottaminen on hyödyllistä? Syy on siinä, että kun funktiot ovat tuotettu tietojoukosta, sitä voidaan käyttää uudelle datalle, jota malli ei ole koskaan nähnyt ja funktioiden palauttamat vastaukset voivat tuottaa näkemyksiä siitä, millaisia päätöksiä datan pohjalta kannattaa tehdä. Funktio on yksinkertaisesti deterministinen sisääntulojen kartoitus ulostuloihin. Yksinkertaistaminen tosin piilottaa funktiojoukon monipuolisuuden. Alla muutamia esimerkkejä:

- Roskapostin suodatus on funktio, joka ottaa sähköpostin sisääntulona ja palauttaa ulostulona vastauksen, onko sähköposti roskapostia vai ei.
- Kasvojentunnistusfunktio ottaa kuvan sisääntulona ja palauttaa pikselit, jotka rajaavat kuvasta naaman.
- Puheentunnistusfunktio ottaa ääninäytteen puheesta sisääntulona ja palauttaa siitä transkription tekstinä.
- Konekääntämisen funktio ottaa lauseen sisään jollain kielellä ja palauttaa ulostulona saman tekstin toisella kielellä.

Näiden ratkaisujen ansiosta monet ongelmat monilla eri toimialoilla voidaan pukea funktioksi, että koneoppiminen on tullut tärkeäksi viimevuosina. (Keller 2019, 13.)

4.4 Neuroverkko

Termi syväoppiminen kuvaa joukkoa neuroverkkoverkkomalleja, joilla on useita kerroksia yksinkertaisen tiedon prosessointiohjelmia, nimeltään neuroni verkossa. Neuroverkko on laskennallinen malli, joka on saanut vaikutteita ihmisaivojen rakenteesta. Ihmisen aivot koostuvat massiivisesta määrästä hermosoluja, joita kutsutaan neuroneiksi. Joidenkin arvioiden mukaan, ihmisen aivoissa on sata miljardia neuronia. Neuroneilla on yksinkertainen kolmiosainen rakenne, joka koostuu solun rungosta, joukosta kuituja, joita kutsutaan tuojahaarake (dendriitti), sekä yhdestä pitkästä kuidusta nimeltään viejähaarake (aksoni). Kuva 7 havainnollistaa neuronin rakenteen ja miten se on yhteydessä toisiin neuroneihin. Tuojahaarakkeet ja viejähaarakkeet lähtevät solun rungosta ja tuojahaarakkeet ovat yhteydessä toisen neuronin viejähaarakkeisiin. Tuojahaarakkeet toimivat sisääntulokanavana neuronille ja saavat toisen neuronin lähettämän signaalin viejähaarakkeen kautta. Viejähaarakkeet toimivat neuronin ulostulokanavana ja muut neuronit, jotka ovat yhteydessä tuojahaarakkeiden kautta kyseiseen neuroniin, saavat signaalin sisääntulona viejähaarakkeesta. (Keller 2019, 34.)

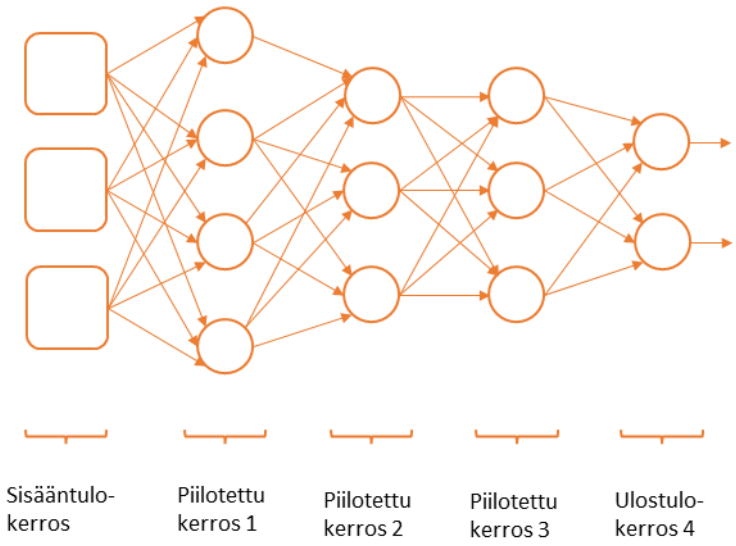
Kuva 7. Aivojen hermosolun, eli neuronin rakenne (mukailtu Keller 2019, 34)

Neuronit toimivat yksinkertaisella tavalla. Jos sisään tuleva stimulantti on riittävän voimakas, neuroni välittää sähköpulssin, nimeltään toimintapotentiaali, sen viejähaarakkeen kautta muille neuroneille, jotka ovat siihen kytköksissä. Neuroni toimii kaikki tai ei mitään kytkimenä, joka ottaa sisääntulon ja joko välittää sen eteenpäin, tai sitten ei. (Keller 2019, 34.)

Tämä havainnollistaminen on todella yksinkertaistettu malli aivojen toiminnasta, mutta se esittää pääasiat, jotta ymmärretään analogia aivojen rakenteen ja tietokonemallinnettujen neuroverkkojen välillä. Nämä analogian ideat ovat: aivot koostuvat monista toisiinsa yhdistyneistä ja yksinkertaisista yksiköistä nimeltään neuronit, aivojen toiminta voidaan ymmärtää tiedonkäsittelynä korkeiden ja matalien sähkösignaalien tai aktivaatiopotentiaaleina, jotka leviävät neuroniverkossa sekä jokainen neuroni saa ärsykkeen viereisiltä neuroneilta ja kartoittaa nämä sisääntulot joko korkeaksi tai matalaksi ulostulo arvoksi. Kaikki laskennalliset tietotekniikan neuroverkkomallit sisältävät nämä ominaisuudet. (Keller 2019, 34.)

4.5 Keinotekoinen neuroverkko

Keinotekoinen neuroverkko koostuu verkosta yksinkertaisia informaationkäsittely yksiköitä, nimeltään neuronit. Neuroverkon todellinen kyky luoda malli monimutkaisia relaatioita, ei ole sen kyky luoda monimutkaisia matemaattisia malleja, vaan se tulee sen kyvystä luoda interaktioita monen yksinkertaisen neuronin joukosta. (Keller 2019, 34.) Kuva 8 havainnollistaa neuroverkon rakenteen.



Kuva 8. Yksinkertaisen neuroverkon topologinen kuva (mukailtu Keller 2019, 35)

On standardin mukaista ajatella, että neuronit neuroverkossa on organisoitu kerroksin. Kuvattu verkko sisältää viisi kerrosta: yksi sisääntulokerros, kolme piilotettua kerrosta ja ulostulokerros. Piilotettu kerros on taso, joka ei ole sisääntulo- tai ulostulokerros. Syväoppimisen verkot ovat neuroverkkoja, joilla on monia neuronien muodostamia piilokerroksia. Verkossa on oltava vähintään kaksi piilotettua kerrosta, jotta sitä voidaan kutsua syväksi. Monissa syväoppimisen verkoissa on kuitenkin enemmän tasoja kuin kaksi. Neuroverkon syvyys lasketaan siitä, monta piilotettua kerrosta siinä on plus ulostulokerros. (Keller 2019, 34.)

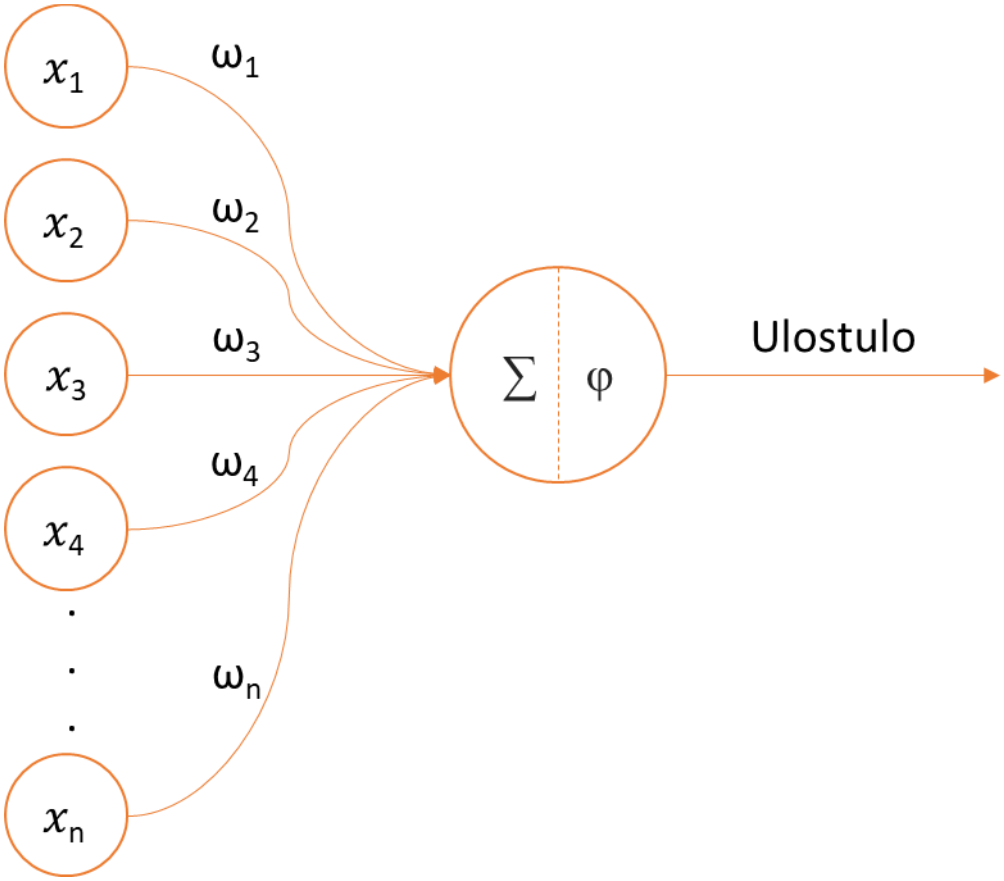
Kuva 8 neliöt kuvastavat sisääntulokerroksen muistipaikkoja, jotka toimivat sisääntuloarvoina verkolle. Näitä muistipaikkoja voidaan ajatella ikään kuin tunnistavina neuroneina. Näissä tunnistavissa neuroneissa ei ole mitään informaationkäsittely logiikkaa, vaan ne ovat ainoastaan ulostuloarvoja tiedolle kyseisessä muistipaikassa. Ympyrät kuvastavat verkon informaationkäsittely neuroneja. Jokainen neuroni ottaa sisäänsä joukon numeerisia arvoja ja kartoittaa ne yhdeksi ulostuloarvoksi. Neuronin jokainen sisääntuloarvo on joko tunnistelijaneuronin, tai tiedonkäsittely neuronin ulostuloarvo. (Keller 2019, 34.)

Kuva 8 nuolet havainnollistavat, miten informaatio liikkuu verkon neuronin ulostulosta toisen neuronin sisääntuloon. Jokainen yhteys yhdistää kaksi neuronia toisiinsa ja jokainen yhteys on yhdensuuntainen, eli tieto kulkee yhteydessä vain toiseen suuntaan. Jokaiselle yhteydelle on määriteltynä painoarvo. Painoarvot ovat vain numeerinen arvo, mutta ne ovat tärkeitä. Yhteyden painoarvo vaikuttaa siihen, miten neuroni käsittelee vastaanottamansa tiedon ja keinotekoisen neuroverkon opettaminen on käytännössä optimaalisten painoarvojen hakemista. (Keller 2019, 34.)

4.6 Aktivaatiofunktiot

Neuronin sisäinen informaationkäsittely on hyvin samankaltaista kuin aiemmin mainittu lainamalli. Lainapäätöksen mallilla lasketaan ensin painotettu summa sisääntulotietojen (vuositulot ja lainanmäärä). Summan painoarvoa muutetaan käyttäen tietojoukkoa, jotta painotettu summalaskenta lainan ja tulojen vastaa tarkasti lainanhakijan lainakykyä. Toinen vaihe on prosessoida painotettu summaus (arvioitu lainakyky) päätöksentekosäännön kautta. Tämä sääntö on funktio, joka on kartoitettu luottokykyarvoon ja sen pohjalta päätökseen, annetaanko henkilölle lainaa vai ei. (Keller 2019, 35.)

Neuroni toteuttaa myös kaksitasoisen prosessin, joka yhdistää sisääntulot ulostuloihin. Ensimmäinen vaihe on painotetun summan laskeminen neuronin kahdelle sisääntulolle. Sen jälkeen painotettu summa välitetään toiselle funktiolle, joka yhdistää painotettujen summien arvon neuronin viimeiselle ulostulolle. Kun neuronia suunnitellaan, voimme käyttää monen tyyppisiä funktioita tähän toiseen vaiheeseen, tai prosessointiin. Se voi olla yksinkertainen päätössääntö, jota käytettiin lainapäätösmallissa, tai se voi olla monimutkaisempi. Tyypillisesti neuronin ulostuloarvoa kutsutaan aktivaatioarvoksi, joten tämä toinen funktio, joka yhdistyy neuronin painotetun summan aktivaatio arvoon, on nimeltään aktivaatiofunktio. (Keller 2019, 35.)



Kuva 9. Keinotekoisen neuronin rakenne (mukailtu Keller 2019, 35)

Kuva 9 havainnollistaa kuinka nämä prosessin vaiheet näkyvät keinotekoisessa neuronissa. Kuvan ∑ -symboli esittää painotetun summan laskemista ja φ -symboli esittää painotetun summan prosessointia aktivaatiofunktiolla neuronin ulostulon muodostamiseksi. Kuva 9 neuroni saa sisäänsä n-kpl sisääntuloja [x_1,…, x_n] n-kpl eri sisääntuloyhteyksistä ja jokaiselle yhteydelle on assosioitu painoarvo [ω_1,…, ω_n]. Painotetun summan laskenta tapahtuu kertomalla sisääntuloarvo painoarvolla ja summaamalla näiden tulokset. Matemaattisesti kaava kirjoitetaan:

$$z = (x_1 * x_1) + (x_2 * x_2) + (x_n * x_n)$$

Tämä kaava voidaan kirjoittaa lyhyemmin:

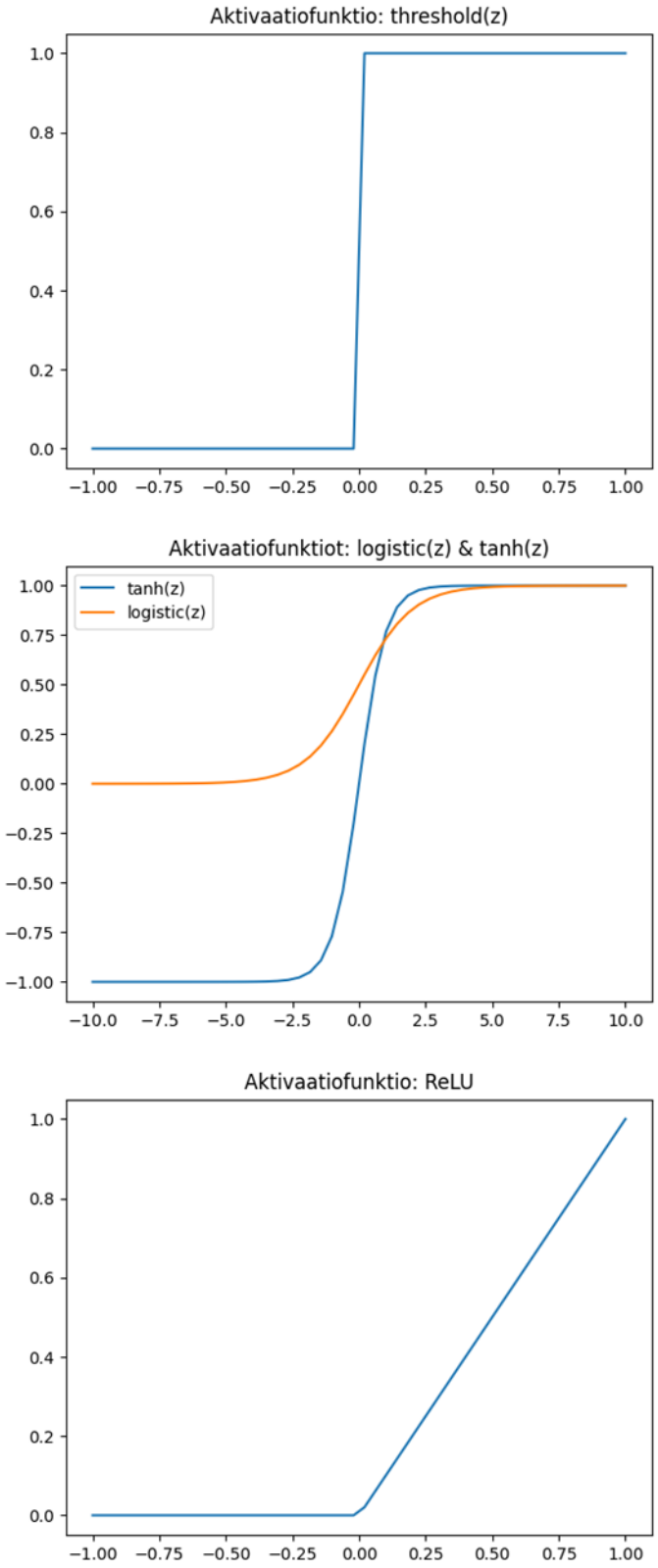
$$z = \sum_{i=1}^{n} x_i * w_i$$

Esimerkiksi, jos neuroni saa sisääntuloarvoiksi [x_1= 3, x_2 = 9] ja vastaavat painoarvot ovat [ω_1 = -3, ω_2 = 1], niin painotettu summa lasketaan seuraavasti:

$$z = (3 * -3) + (9 * 1)$$
$$= 0$$

(Keller 2019, 35-36.)

Toinen prosessoinnin vaihe on välittää neuronin painotetun summan arvo aktivaatiofunktiolle. Kuva 10 esittää mahdolliset aktivointifunktiot ja näiden funktioiden sisääntuloarvot ovat joko väliltä [-1, …, +1] tai [-10, …, +10] riippuen, kumpi intervalli kuvaa paremmin funktion muotoa. Luottokyvyn arvioinnissa oli käytössä threshold funktio ja rajana oli, onko lainaluokitus yli 200. Threshold aktivaatiofunktiot olivat yleisiä neuroverkkojen tutkimuksen alkuajoilla. Kuva 10 keskimmäinen graafi kuvaa logistic ja tanh aktivaatiofunktiot. Nämä olivat viimepäiviin asti melko suositut funktiot. Alin kuva esittää ReLU aktivaatiofunktiota. Tämä on nykypäivänä melko suosittu aktivaatiofunktio syväoppimisen verkoissa. (Keller 2019, 36.)



Kuva 10. Ylin: threshold funktio, keskimmäinen: logistic ja tanh -funktiot, alin: rectified linear funktio (ReLU) (mukailtu Keller 2019, 36)

Kuva 10 keskimmäisen kuvaajan oranssi viiva esittää logistic funktion. Olettaen, että neuroni käyttää logistic aktivaatiofunktiota, tämä kuvaaja näyttää miten summauksen tulos antaa ulostuloarvoksi logistic(0) = 0.5. Neuronin ulostuloarvon aktivaatio voidaan esittää seuraavasti:

Ulostulo = aktivaatiofunktio $(z = \sum_{i=1}^{n} x_i * w_i)$
= logistic $(z = (3 * -3) + (9 * 1))$
= logistic $(z = 0)$
= 0.5

(Keller 2019, 36-37.)

Tämän neuronin arvo on lähes identtinen lainapäätös esimerkin kanssa. Suurin ero on se, että tässä on muutettu painotetun summan arvo 0 ja 1 välille. Riippuen neuronin sijainnista neuroverkossa, neuronin ulostulon aktivaatio, tässä tapauksessa y = 0.5, välitetään joko seuraavan neuronin sisääntuloarvoksi, tai se on osa neuroverkon lopputulosta. Jos neuroni on ulostulokerroksessa, ulostuloarvon tulkinta riippuu siitä, mitä neuroni on suunniteltu toteuttavan. Jos neuroni on jossain piilokerroksessa, ei välttämättä ole mahdollista saada tietoon, mitä arvo oikeasti tarkoittaa, koska se voi olla jonkinlainen johdettu luku, minkä neuroverkko on todennut hyödylliseksi oikean lopputuloksen saavuttamiseksi. (Keller 2019, 36-37.)

4.7 Konvoluutioneuroverkko

Konvoluutioneuroverkot suunniteltiin kuvantunnistustehtäviin ja sitä käytettiin alun perin käsin kirjoitettujen numeroiden tunnistamiseen. Konvoluutioneuroverkon suunnitteluperiaatteena oli luoda verkko, jossa aikaisen tason neuronit poimivat paikallisia visuaalisia piirteitä ja myöhemmän tason neuronit yhdistävät nämä piirteet ja muodostavat korkeamman tason ominaisuudet. Lokaali visuaalinen piirre on ominaisuus, joka rajoittuu kuvan viereisten pikselien pieneen otokseen. Esimerkiksi kasvojentunnistuksessa aikaisen tason neuroverkon konvoluutioverkon neuronit oppivat aktivoitumaan yksinkertaisista piirteistä, kuten linjoista tietyssä kulmassa, tai kurvien ryhmästä ja syvemmän tason neuronit yhdistävät nämä alemman tason ominaisuudet kuvaamaan kasvojenpiirteitä, kuten silmiä tai nenää ja verkon viimeisen kerroksen neuronit yhdistävät kasvojenpiirteiden aktivaatiot, jotta se pystyy tunnistamaan kuvasta kokonaiset kasvot. (Keller 2019, 71.)

Käyttäen tätä lähestymistapaa, kivantunnistamisen perimmäinen tehtävä on opettaa piireteiden tunnistamisen funktiot, jotka pystyvät varmasti tunnistamaan visuaalisten piirteiden olemassaolon tai niiden puuttumisen annetusta kuvasta. Funktioiden opetus on neuroverkkojen ydin ja tämä saavutetaan opettamalla sopivat painoarvot neuroverkon yhteyksille. Konvoluutioneuroverkot oppivat tunnistamaan nämä piirteet piirteidentunnistusfunktioilla

kuvan ominaisuuksista. Neuroverkkoarkkitehtuurin suunnitteluun kuitenkin liittyy haaste, että paikalliset visuaaliset piirteet voidaan tunnistaa riippumatta siitä, missä kohtaa kuvaa ne esiintyvät. Ominaisuuksien tunnistamisen on siis toimittava erilaisilla kuvilla. Esimerkiksi, kasvojentunnistuksen pitää tunnistaa silmän muoto kuvasta, riippumatta siitä onko silmä keskellä kuvaa, vai oikeassa ylänurkassa. Tämä on ollut yksi konvoluutioneuroverkkojen suunnitteluprinsiipeistä kuvankäsittelyssä. (Keller 2019, 71.)

Konvoluutioneuroverkko kiertää tämän ongelman käyttäen neuronien välillä jaettuja painoarvoja. Kuvantunnistamisessa neuronin implementoimassa funktiossa tämä voidaan käsittää visuaalisen piirteen tunnistajana. Esimerkiksi neuroverkon ensimmäisessä piilotetussa kerroksessa neuronit saavat pikselijoukon sisääntuloarvona ja ne antavat ulostuloksi korkean aktivoinnin, jos tietty kuvio (lokaali visuaalinen piirre) löytyy tästä pikselijoukosta. Se, että neuronin funktio määritellään sen painotettujen arvojen mukaan tarkoittaa sitä, että jos kahdella neuronilla on samat painotetut sisääntulot, ne käyttävät samaa funktiota ja toteuttavat samaa tehtävää. (Keller 2019, 71.)

4.8 Epoch

Epoch tarkoittaa, monta kierrosta mallia opetetaan. Yksi epoch on, kun koko tietojoukko käy neuroverkon edestakaisin kertaalleen. Epoch on helppo tapa pitää kirjaa opetuskierroksista. (Ranjan & Dr. Senthamilarasu 2020, 49.)

4.9 Hyperparametrit

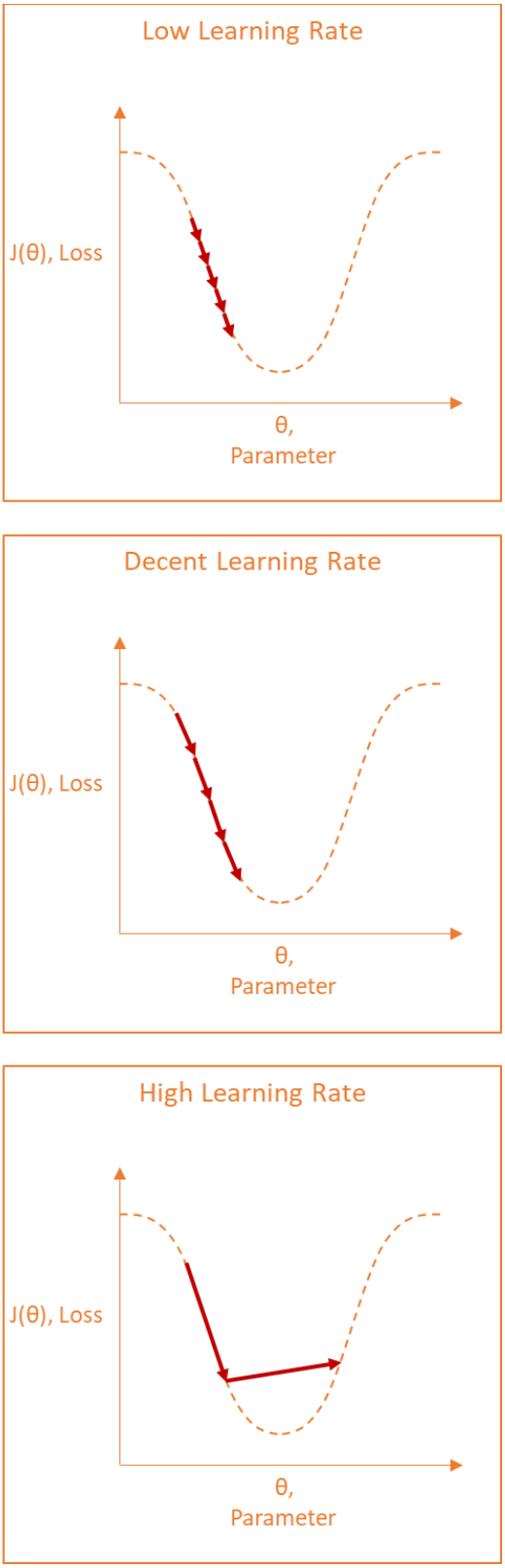
Hyperparametreja voisi verrata kitaran virityskoneistoon, joita käytetään parhaan soinnin saavuttamiseen. Hyperparametrit ovat asetuksia, joilla voidaan säätää koneoppimisalgoritmin käyttäytymistä. Hyperparametrien valinta on elintärkeä osa syväoppimismallin ratkaisuissa. (Ranjan & Dr. Senthamilarasu 2020, 46.)

Monilla syväoppimismalleilla on tietyt hyperparametrit, jotka säätävät mallin eri aspekteja, kuten muistinkäyttö ja suorituksen vaativuus. On myös mahdollista määritellä lisähyperparametreja auttamaan algoritmia mukautumaan skenaarioon tai ongelmaan. Parhaan suorituskyvyn saavuttamiseksi, datatieteilijät tyypillisesti käyttävät paljon aikaa hyperparametrien viilaamiseen, sillä ne ovat niin isossa roolissa syväoppimismallin kehittämisessä. Hyperparametrit voidaan jaotella kahteen kategoriaan, mallin opettamiseen liittyvät hyperparametrit sekä verkkoarkkitehtuuriin liittyvät hyperparametrit. (Ranjan & Dr. Senthamilarasu 2020, 46.)

4.9.1 Mallin opettamiseen liittyvät hyperparametrit

Mallin opettamiseen liittyvät hyperparametrit ovat tärkeässä roolissa mallin opettamisessa. Nämä hyperparametrit ovat mallin ulkopuolella, mutta niillä on suora vaikutus siihen. Hyperparametreja ovat mm. oppimisnopeus, joukon koko ja epoch, eli opetuskierrosten lukumäärä. (Ranjan & Dr. Senthamilarasu 2020, 47.)

Oppimisnopeus on yksi tärkeimmistä hyperparametreista ja se määrittää mallin oppimisen edistymisen ja tavallaan sitä voidaan käyttää optimoimaan sen kapasiteetti. Liian alhainen oppimisnopeus lisää mallin opettamisessa kestävää aikaa, koska sillä kestää kauemmin aikaa muuttaa inkrementaalisesti painoarvoja päästäkseen optimaaliseen lopputulokseen. Toisaalta liian suuri oppimisnopeus auttaa opettamaan mallia muuntautumaan nopeammin, mutta se taas aiheuttaa harhaan menoa minimiarvojen kanssa. Kuva 11 ylimmäisenä nähdään, että matala oppimisnopeus vaatii monta päivitystä, ennen kuin minipiste saavutetaan. Kuva 11 keskimmäisenä on kohtalainen oppimisnopeus, joka saavuttaa minimipisteen nopeasti. Se vaatii vähemmän päivityksiä minimiin pääsyyn. Suuri oppimisnopeus johtaa poikkeavaan käytökseen, kuten Kuva 11 alimmaisesta laatikosta huomataan. (Ranjan & Dr. Senthamilarasu 2020, 47-48.)



Kuva 11 Oppimisnopeus (mukailtu Ranjan & Dr. Senthamilarasu 2020, 47-48)

Joukon koko on myös tärkeä hyperparametri, jolla on suuri merkitys mallin tarkkuuteen, aikaan ja resurssivaatimuksiin. Joukon koko määrittää monta datapistettä lähetetään kone-oppimisalgoritmille yhdessä iteraatiossa opettamisen aikana. Vaikkakin suurella joukon

koolla on isoja hyötyjä laskennan nopeuden kannalta, mutta sillä on huomattu olevan negatiivisia vaikutuksia mallin laadun kanssa, mitattuna sen yleistämiskykynä. Joukon koko vaikuttaa myös mallin opettamisen muistivaatimuksiin nostattavasti. (Ranjan & Dr. Senthamilarasu 2020, 49.)

Vaikka pienempi joukon koko kasvattaa opetusaikaa, se tuottaa lähes aina paremman mallin, kuin suurempi joukon koko. Tämä johtunee siitä, että pienemmät joukon koot tuottavat enemmän kohinaa gradientti arvoille, joka auttaa niitä lähestymään paremmin tasaista minimiä. Huonona puolena tässä on se, että pienempi joukon koko kasvattaa opetusaikaa. (Ranjan & Dr. Senthamilarasu 2020, 49.)

Epoch on mallin opetuskierrosten lukumäärä. Koko tietojoukon välitys edestakaisin neuroverkon läpi on yksi opetuskierros. Voidaan sanoa, että opetuskierros on helppo tapa pitää kirjaa kierroista, sillä välin kun opetus- tai validointivirheet kasvavat. Koska yksi epoch on liian suuri kerralla käsiteltäväksi, niitä jaetaan pienempiin osiin. Yksi hyödyllinen tekniikka on Keras takaisinkutsu, joka lopettaa mallin opettamisen, jos opetus- / validointivirheet eivät ole parantuneet viimeisein 10–20 opetuskierroksen aikana. (Ranjan & Dr. Senthamilarasu 2020, 49.)

4.9.2 Verkkoarkkitehtuuriin liittyvät hyperparametrit

Hyperparametreja, jotka liittyvät suoraan syväoppimismallin arkkitehtuuriin, kutsutaan verkkoarkkitehtuuri spesifeiksi hyperparametreiksi. Niitä on kolmea erilaista. Ne ovat piilotettujen kerrosten määrä, regularisointi sekä aktivaatiofunktiot hyperparametreina. (Ranjan & Dr. Senthamilarasu 2020, 50.)

Mallin on helppo oppia simppeleitä ominaisuuksia pienestä määrästä piilotettuja kerroksia. Kun piirteet monimutkaistuvat tai epälineaarisuus kasvaa, se tarvitsee lisää ja lisää kerroksia ja yksiköitä. Jos on liian pieni verkko monimutkaiseen tehtävään, lopputulos on se, että malli ei toimi hyvin, koska sillä ei ole riittävää oppimiskapasiteettia. Jos yksiköitä on vähän liikaa, se ei haittaa paljoa, mutta jos niitä on todella paljon liikaa, malli rupeaa ylisovittumaan. Ylisovittaminen tarkoittaa, että malli yrittää muistella tietojoukkoa ja toimii hyvin opetusdatan kanssa, mutta ei toimi kunnolla testidatan kanssa. Piilotetuilla kerroksilla voidaan hakea optimaalinen tarkkuus verkolle. (Ranjan & Dr. Senthamilarasu 2020, 50.)

Regularisointi mahdollistaa pienten muutosten tekemisen oppimisalgoritmiin, jotta mallista tulee yleistetympi. Se myös parantaa mallin oikeellisuutta datalla, jota ei ole käytetty opetukseen. Koneoppimisessa regularisointi vaikuttaa kertoimiin heikentävästi. Syväoppimisessa regularisointi vaikuttaa matriisien solmujen painoarvoihin heikentävästi. (Ranjan & Dr. Senthamilarasu 2020, 50.)

5 Koneoppimisalustan valinta

Kankaiden lajittelua tehdään monessa eri paikassa, eikä joka paikassa ole välttämättä tarjolla vakaita ja/tai edullisia internetyhteyksiä. Tämä oli yksi syy, miksi oli järkevää toteuttaa laitteen sisällä toimiva kuvantunnistaminen. Tällöin ohjelma toimii, vaikka laitteessa olisi epävakaa internetyhteys, tai vaikka yhteyttä verkkoon ei olisi ollenkaan.

Kustannukset olivat toinen syy, miksi kuvantunnistamiseen ei käytetä Azuren, Google Cloudin tai Amazon Web Servicesin tarjoamia rajapintoja. Pilvipalveluista tulee kiinteitä kuluja ja mahdollisesti minuutti kustanteisia kuluja koneoppimiseen käytettyjen resurssien takia. Pilven käytössä olisi ollut monia isoja hyötyjä, kuten alustariippumattomuus ja se, että malli sijaitsisi pilvessä, eikä vaadi sovelluksen / sovellusten päivittämistä mallin muuttuessa.

Edellä mainittujen seikkojen takia, oli järkevää toteuttaa mobiililaite yhteensopiva koneoppimismalli, joka voitiin tallentaa ohjelmaan sisäänrakennetusti. Koneoppimismalli tarvitsee jatko-opettamista, jolloin olisi myös hyödyllistä toteuttaa ohjelmaan koneoppimismallin päivityslogiikka, joka hakee pyydettäessä uudemman version FTP-palvelimelta, mutta ei ole siitä riippuvainen. FTP-palvelimella muutamien megatavujen kokoiset koneoppimismallit ovat edullisempia ylläpitää, kuin kokonainen pilvipohjainen ratkaisu. FTP-jaon voi toteuttaa pilviratkaisuna, joka poistaa fyysisen palvelimen ylläpidon. Mallin säilömiseen voidaan myös käyttää valmista FTP-palvelinta/tai -palvelua, jos sellainen on edullisesti saatavilla.

5.1 Azure Machine Learning

Azure Machine Learning olisi ollut hyvä vaihtoehto koneoppimisalustaksi, mutta kustannusten vuoksi se jäi pois. Se tarjoaisi mahdollisuudet luoda malleja, tai lähettää kuva analysoitavaksi pilveen, jolloin laitteella tehtävää prosessointia ei tarvittaisiin. Tämä olisi alustariippumaton ratkaisu, mutta vaatisi hyvät vakaat internetyhteydet.

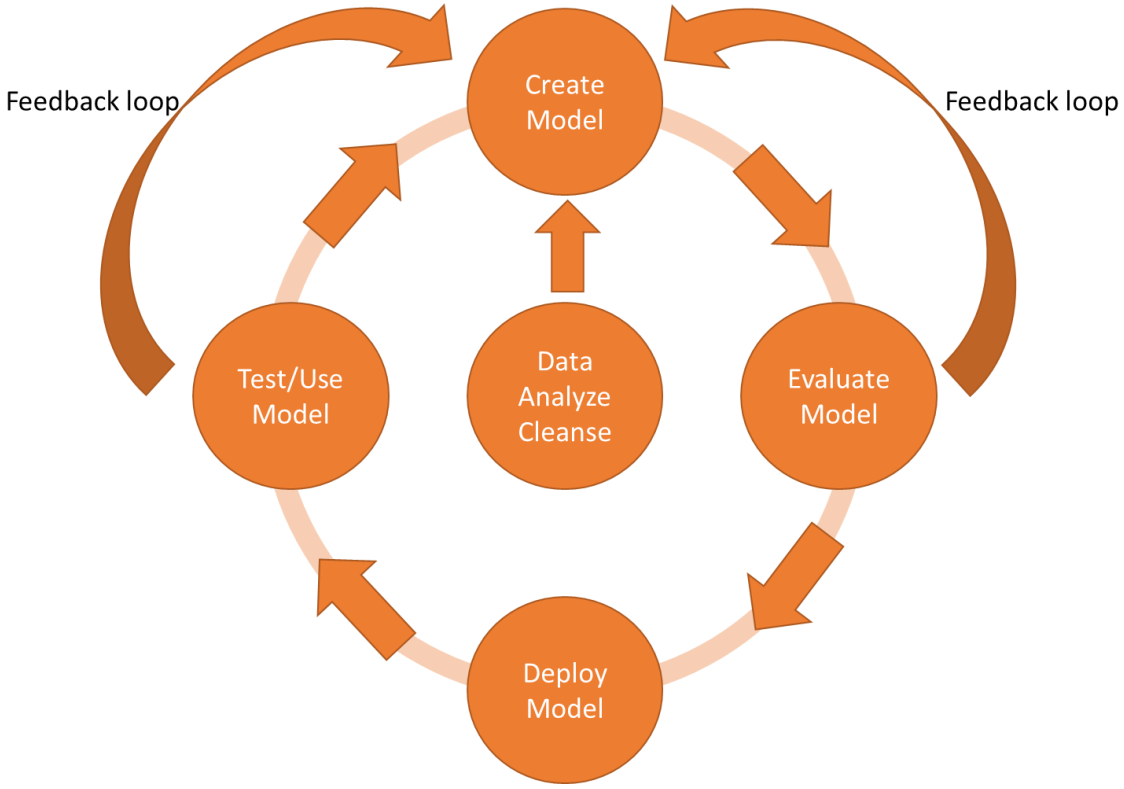
Azure Machine Learningin keskeisiä teemoja on mahdollisuus luoda nopeita kokeiluja, arvioida niiden luotettavuutta ja poissulkemaan mallit, jotka eivät ole käyttökelpoisia. Koneoppimismallin tarkoitus on kuitenkin olla luotettavampi, kuin pelkkä arvaus. Monet menestyvät yrittäjät ovat aina halukkaita pääsemään kilpailijoiden edelle parantamalla omia liiketoiminnan päätöksiään. Ennakoiva analytiikka ja Azure Machine Learning auttaa tällä osa-alueella. Yritystoiminnassa, kuten muillakin elämän osa-alueilla, oikean lopputuloksen päätteleminen ja sen tarkkuuden parantaminen antaa kilpailuedun muihin nähden. (Barnes 2015, 25.)

Yksi esimerkki ennakoivan analysoinnin käyttämisestä, on markkinointikampanjan onnistumisen palaute. Kun otetaan huomioon asiakkaiden suhtautuminen tarjouksiin, segmentointi

asiakasryhmien kautta, hinnoittelun, alennuksen ja vuodenaikojen vaikutukset, alkaa hahmottua kaavoja. Nämä kaavat tarjoavat vihjeitä, syistä ja seurauksista, jotka auttavat lopulta tekemään valistuneempia markkinointipäätöksiä. Tämä on tavallinen lähtökohta monille kohdistetuille markkinointikampanjoille. (Barnes 2015, 25.)

Ihmiset, jotka ovat myös markkinoinnin asiakaskuntaa, ovat niin sanotusti ”tapojensa orjia”. Kun puhutaan ihmisten käyttäytymisestä, aiempi käyttäytyminen on vahva indikaattori tulevasta käyttäytymisestä. Ennakoiva analytiikka ja koneoppiminen voi auttaa tienaamaan näillä pääprinsiipeillä. Tämä tapahtuu tutkimalla aiempaa käyttäytymistä, jotta tulevat markkinointiyritykset tuottavat suuremmalla lopputuloksella tulosta. (Barnes 2015, 25-26.)

Azure Machine Learning ratkaisut koostuvat toistettavista kaavoista. Työnkulku koostuu vaiheista, jotka ovat suunniteltu auttamaan ennakoivan analytiikan ratkaisun nopeasti. Kuva 12 vaiheet ovat esitettynä.



Kuva 12. Azure Machine Learning työnkulku (mukailtu Barnes 2015, 26)

Data on kaikki kaikessa. Tässä työvaiheessa käännetään, analysoidaan testi ja opetus tietojoukot, joita käytetään Azure Machine Learning ennakoivan muodostamiseen. Kuvan create model tarkoittaa monen eri koneoppimisalgoritmin käyttömahdollisuutta uusien koneoppimismallien muodostamiseen, jotka voivat tehdä ennustuksia datan pohjalta. Kuvan

evaluate model tarkoittaa mallin tarkkuuden arviointia sellaisella datalla, jonka lopputulos on etukäteen tiedossa. Vertailua tehdään yhdistelemällä eri ennakoivien mallien käyttämistä parhaan kombinaation löytämiseksi. Deploy model tarkoittaa koneoppimismallin julkaisemista ja mallin käyttämistä sellaisella tiedolla, jota ei ole käytetty mallin opettamiseen. Test/Use Model tarkoittaa mallin käyttöä testi- tai tuotantoympäristössä. Siihen voidaan implementoida manuaalinen tai automaattinen palauteen käsittely, jolla tarkoitetaan mallin jatko-opettamista, kun oikea tai väärä päätelmä on tehty. Mahdollistamalla mallin itseoppiminen, malli oppii jatkuvasti virheellisistä ennustuksista, eikä toista sama virhettä uudelleen, toisin kuin ihminen voi tehdä. (Barnes 2015, 25-26.)

5.2 TensorFlow

TensorFlown tarina alkoi Google Brain -tiimin sisäisenä projektina, alkuperäiseltä nimeltään DistBelief. Projekti on alusta lähtien kehitetty nopeaan datan käsittelyyn avoimen lähdekoodin viitekehyksenä. Projekti julkaistiin 2015 vuoden marraskuussa, nimellä TensorFlow. Nimi tulee tensoreista, jotka ovat yleistettynä skalaareja, vektoreita, matriiseja ja moniulotteisia matriiseja. Vuonna 2019 Google julkaisu TensorFlow 2.0 -version, jota käytetään tässä opinnäytetyössä, vaikka versionumeroa ei erikseen mainitakaan. (Audevart ym. 2021, 2.)

TensorfFlow on viitekehys kaikenlaiseen laskentaan, joka vaatii hyvää suorituskykyä ja helppoa jakelua, koska se on suunniteltu tuotantokäyttöön ja se osaa hyödyntää erilaisia laskenta-arkkitehtuureja, kuten prosessoria, näytönohjainta tai tensorisuoritinta. Se on hyvä syväoppimisessa, joka mahdollistaa kaiken matalien verkkojen (neuroverkko, joka koostuu muutamasta kerroksesta) teosta aina kuvantunnistamiseen ja luonnollisen kielen käsittelyyn. (Audevart ym. 2021, 2.)

5.3 Python

Python toimii ohjelmointikielenä TensorFlow Lite mallin muodostamiseen. Se on suosittu kieli ja paljon käytetty koneoppimisessa. Python on todella monipuolinen kieli ja se soveltuu todella moniin eri käyttötarkoituksiin.

Python on yksi suosituimmista kielistä datatieteisiin. Koska Python on saavuttanut suuren suosion, sille löytyy paljon hyödyllisiä kirjastoja tieteelliseen laskentaan ja koneoppimiseen. Vaikka tulkittavat ohjelmointikielet, kuten Python, ovat suorituskyvyltään heikompia, kuin alemman tason ohjelmointikielet, niin kirjastot kuten NumPy ja SciPy ovat kehitetty alemman kerroksen kielillä, kuten Fortran ja C, ne nopeuttavat vektorisoituja operaatioita monidimensionaalisilla taulukoilla. (Mirjalili & Raschka 2019, 14.)

Koneoppimiseen käytetään monesti scikit-learn kirjastoa, joka on yksi suosituimmista ja matalan oppimiskynnyksen avoimen lähdekoodin koneoppimiskirjasto. TensorFlow kirjastoa käytetään syvien neuroverkkojen opettamiseen. TensorFlow osaa hyödyntää näytönohjainta, joka nopeuttaa mallien opettamista. (Mirjalili & Raschka 2019, 451.)

5.4 Keras

Keras on korkean tason neuroverkko ohjelmointirajapinta ja se on alun perin kehitetty toimimaan muiden koneoppimiskirjastojen, kuten TensorFlown ja Theanon päällä. Keras tarjoaa käyttäjäystävällisen modulaarisen ohjelmointirajapinnan, joka mahdollistaa nopeiden prototyyppien teon ja monimutkaisten mallien rakentamisen muutamalla rivillä koodia. Keras voidaan asentaa käyttäen Pythonin PyPI paketinhallintajärjestelmää, jonka jälkeen voidaan konfiguroida Keras käyttämään TensorFlowta taustamoottorina. Keras on tiiviisti integroitu TensorFlown ja sen moduuleita pääsee käyttämään tf.keras -luokan kautta. (Mirjalili & Raschka 2019, 451.)

Keras tekee neuroverkkojen rakentamisesta helppoa. Yleinen tapa rakentaa neuroverkko TensorFlowssa Keraksen avustuksella, on käyttää tf.keras.Sequential() -funktiota, joka mahdollistaa tasojen kasaamisen verkon rakentamiseen. Kasattu lista voidaan antaa Python listaan mallina, joka on määritelty tf.keras.Sequential() -funktiolla. (Mirjalili & Raschka 2019, 451.)

6 TensorFlow Lite soveltuvuusselvitys

Tämän opinnäytetyön kirjoittajalla oli ennestään vähän kokemusta TensorFlowsta ja mobiilisovelluksen kehittämisestä, mutta ei kuitenkaan kokemusta sellaisesta projektista, jossa nämä kaksi yhdistyvät. Tämän vuoksi paras tapa todentaa, että teknologiat toimivat hyvin toistensa kanssa, oli luoda yksinkertainen sovellus, jotta TensorFlow Lite mallin käyttäminen tulisi tutuksi.

Jotta TensorFlow Lite alustan soveltuvuus koneoppimismallin tekoon oli mahdollista todeta, toteutettiin ensimmäinen ohjelman testiversio Java ohjelmointikielellä Android puhelimille. Android valikoitui soveltuvuusselvityksen kohteeksi osittain siitä syystä, että iOS olisi vaatinut Apple Mac tuoteperheen tietokoneen, jotta iOS applikaation olisi voinut toteuttaa. Java valikoitui teknologiaksi myös sen vuoksi, että monialustateknologiaa ei ollut vielä siinä vaiheessa päätetty ja se oli kirjoittajalle ennestään tuttua.

Jotta oli mahdollista todeta, että TensorFlow Lite toimii Javan kanssa, toteutettiin aluksi hyvin yksinkertainen koneoppimismalli, joka sisälsi vain muutamia internetistä otettuja kuvia. Mallissa oli niin vähän kuvia, että lopputulos olisi ollut yhtä tarkka, kuin että olisi arvottu vastaus luokista, jota mallissa oli mukana. Myös koneoppimismalli luotiin oletusasetuksilla, joka ei anna parasta lopputulosta. Koneoppimismallin tarkkuudella ei ollut vielä tässä vaiheessa suurta merkitystä, vaan pääideana oli testata teknologiaa.

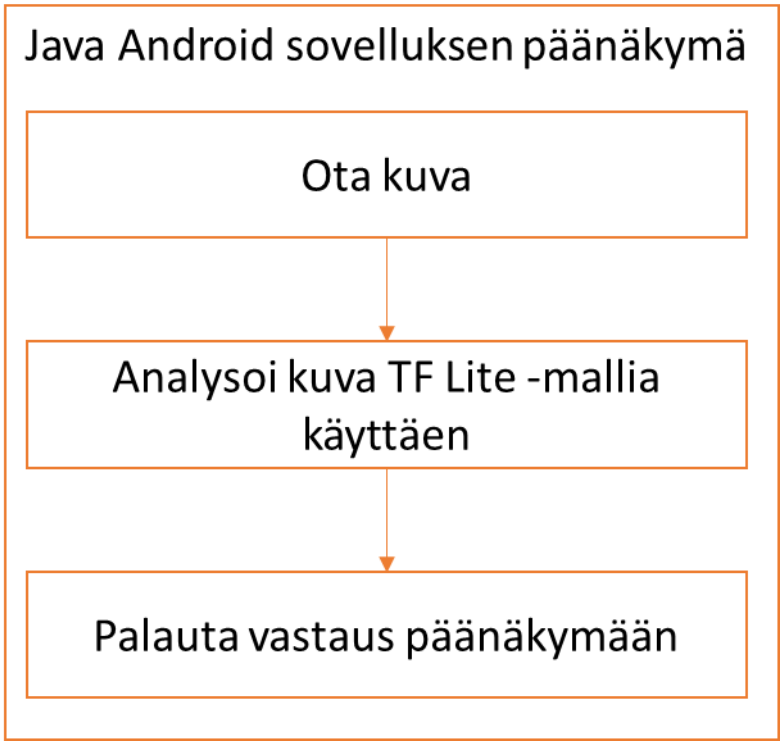
6.1 TensorFlow Lite testimallin luonti

Jotta TensorFlow Lite mallin muodostusta ja käyttämistä pystyi testaamaan, täytyi koostaa jonkinlainen kuvagalleria, jossa on kansioihin lajiteltuna muutamia kankaita. Kirjoittaja haki internetistä muutamia kuvia, jotka toimivat koneoppimismallin pohjana. Tässä vaiheessa tunnistuksen tarkkuus ei ollut pääasia, vaan se, että malli toimii mobiilisovelluksessa. Kuva 13 on kuvattuna Python koodin vaiheet.

Kuva 13. TensorFlow Lite mallinmuodostamisen vaiheet

6.2 Mobiilisovellus

Testiä varten mobiilisovellus tehtiin Java ohjelmointikielellä, koska se oli tämän opinnäytetyön kirjoittajalle ennestään tuttua. Sovellukseen toteutettiin kameratoiminnallisuus, jotta sillä voi ottaa kuvan tunnistettavasta kankaasta. Tämän jälkeen ohjelmassa hyödynnettiin TensorFlow Lite -kirjaston funktioita kuvantunnistamiseen aiemmin luodun koneoppimismallin pohjalta.



Kuva 14. Android sovelluksen toimintamalli

Soveltuvuusselvitystä varten sovelluksesta tehtiin hyvinkin yksinkertainen, jotta oli perus-runko valmiina TensorFlow Lite mallin testaamista varten. Kuva 14 on kuvattu ohjelman runko. Sovelluksella otetaan kuva kankaasta ja se esitetään päänäkymässä, joka on esitetty Kuva 15.

Kuva 15. Soveltuvuusselvityksen mobiilisovellus

6.3 Soveltuvuusselvityksen lopputulema

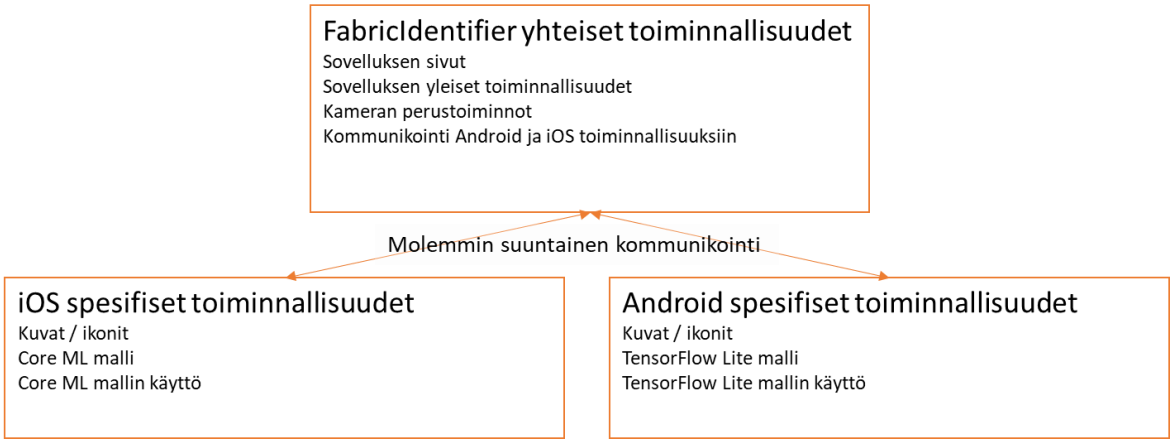
Soveltuvuusselvityksen lopputulos oli se, että TensorFlown avulla saa tehtyä helposti mallin, joka on käytettävissä muun muassa. mobiilisovelluksessa. Tiedoston koko oli pieni ja sovelluksen nopeus oli riittävällä tasolla. Tämän vuoksi TensorFlow Lite valikoitui lopulliseksi teknologiaksi.

7 Mobiilisovellus

Mobiilisovellus toteutettiin käyttäen Microsoft Xamarinia, koska se mahdollistaa saman koodipohjan käyttämisen tietyiltä osin eri alustoille. Aikataulu ja laitteistovaatimusten takia iOS -versio jäi kuitenkin pois tämän opinnäytetyön piiristä. iOS -versio olisi vaatinut Mac-tietokoneen, jota tämän opinnäytetyön tekijällä ei ennestään ollut, eikä sellaisen hankinta tätä projektia varten ollut vaihtoehto. Projektin työnimi on FabricIdentifier, jota käytetään jatkossa kuvissa.

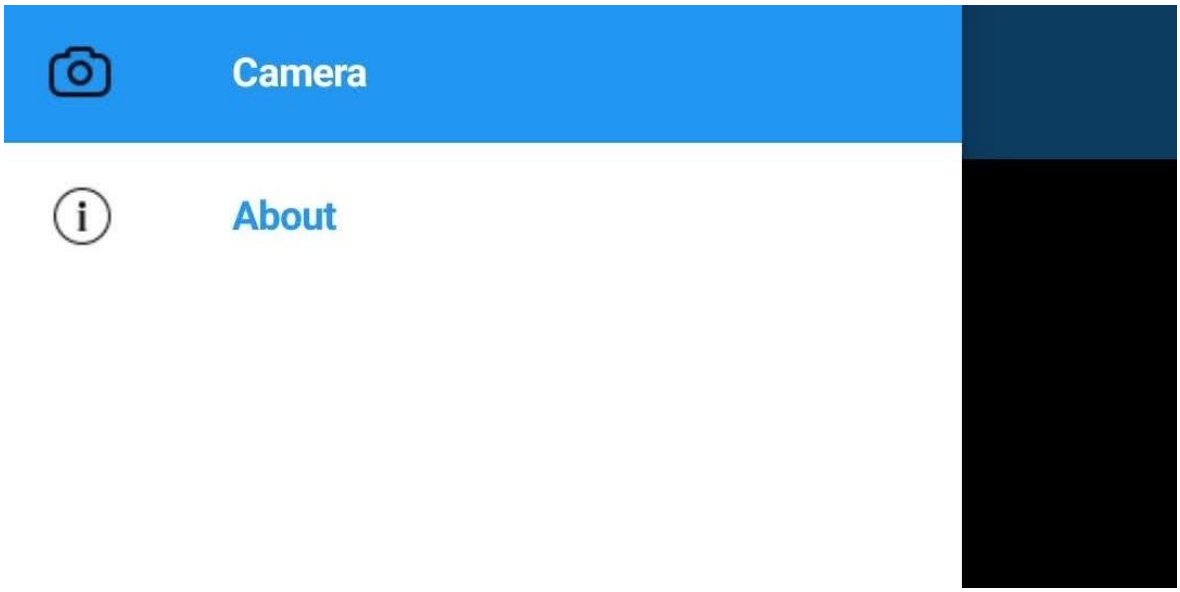
7.1 Ylätason kuvaus

Mobiilisovelluksen Xamarin projektikokonaisuus koostuu kolmesta eri projektista. Yksi projekti sisältää kaikille alustoille yhteiset toiminnallisuudet, toinen sisältää Android version toiminnallisuudet ja kolmas iOS version toiminallisuudet. Sovelluksen sivut ovat osa yhteisiä toiminnallisuuksia. Kuva 16 on esitetty Xamarinilla tehdyn mobiilisovelluksen projektienväliset yhteydet.



Kuva 16. Xamarin ohjelman projektienväliset yhteydet

7.2 Valikko



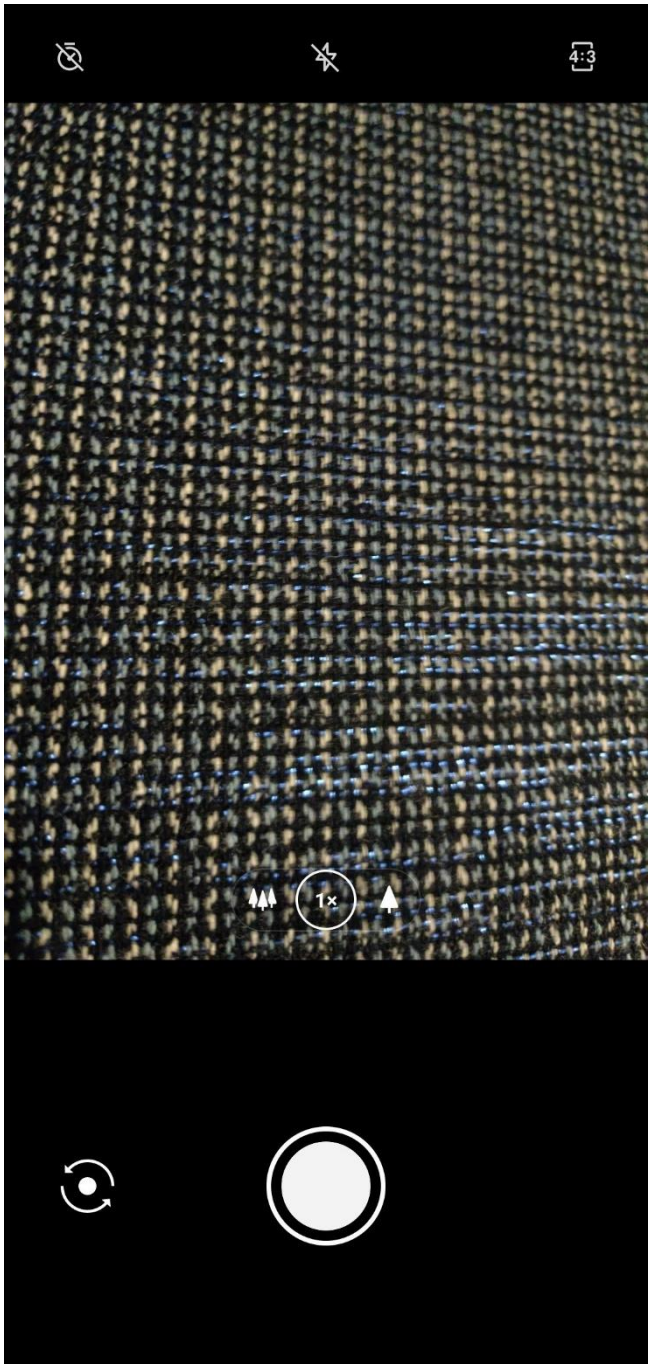
Kuva 17. Mobiilisovelluksen valikko

Kuva 17 on sovelluksen valikko, eli sivut, joita ohjelmassa on. Ohjelma pidettiin yksinkertaisena, eikä mukaan lisätty muita, kuin kamera- ja infosivu. Ohjelmaa voisi jatkokehityksessä laajentaa ja tuoda mukaan uusia toiminnallisuuksia.

7.3 Kamera näkymä

Kuva 18. Kameranäkymä

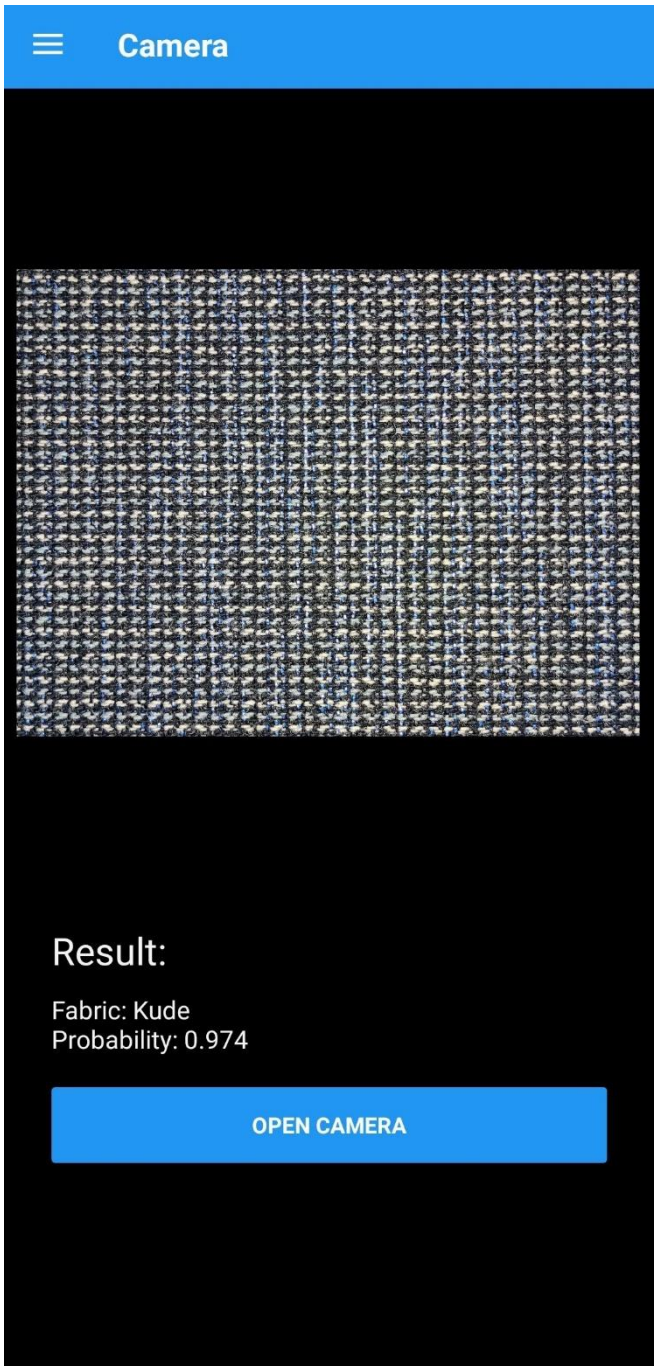
Kuva 18 kamera näkymä käyttää yhteistä koodipohjaa Android ja iOS -versioiden kameran ohjaamiseen, kuten muutkin näkymät. OPEN CAMERA -napin painallus käynnistää pääsäikeen, joka suorittaa TakePhotoAsync -nimisen asynkronisen funktion. TakePhotoAsync funktio käyttää asynkronista MediaPicker.CapturePhotoAsync -funktiota kuvan ottamiseen. Funktio palauttaa tiedoston sijainnin muuttujaan.



Kuva 19. Kuvan ottamisen dialogi

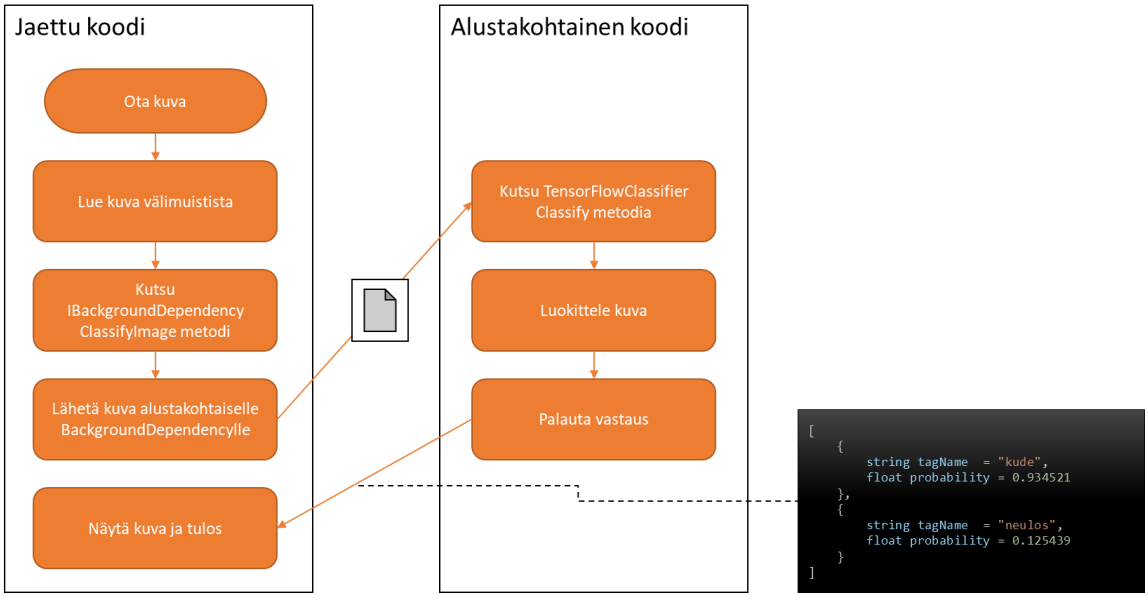
Tämän jälkeen, palautettua tiedoston nimeä käytetään parametrina LoadPhotoAsync -funktiolle. Otettu kuva tallennetaan laitteen väliaikaiseen muistiin, jatkokäsittelyä varten. Kuva luetaan väliaikaisesta muistista ja se annetaan parametrina taustariippuvuus -luokan kautta kyseisen laitteen projektissa olevalle vastaavan nimiselle funktiolle.

7.4 Tulos näkymä



Kuva 20. Tulos näkymä

Kuva 20 esitetty tulosnäkymä on sama näkymä kuin kuvan ottamiseen käytetty näkymä, mutta sivulla näytetään edellinen sovelluksella otettu kuva. Kuvan alapuolella esitetään tunnistuksen lopputulos ja tunnistuksen varmuus. Lisäkehityksenä tähän voisi lisätä mahdollisuuden merkitä tunnistus vääräksi ja mahdollisuus kuvan lähettämiseen FTP-palvelimelle. Tämä toiminnallisuus ei ole ehkä kovin suotavaa, jos sovellus on jaossa julkisesti, mahdollisten väärinkäytösten vuoksi.

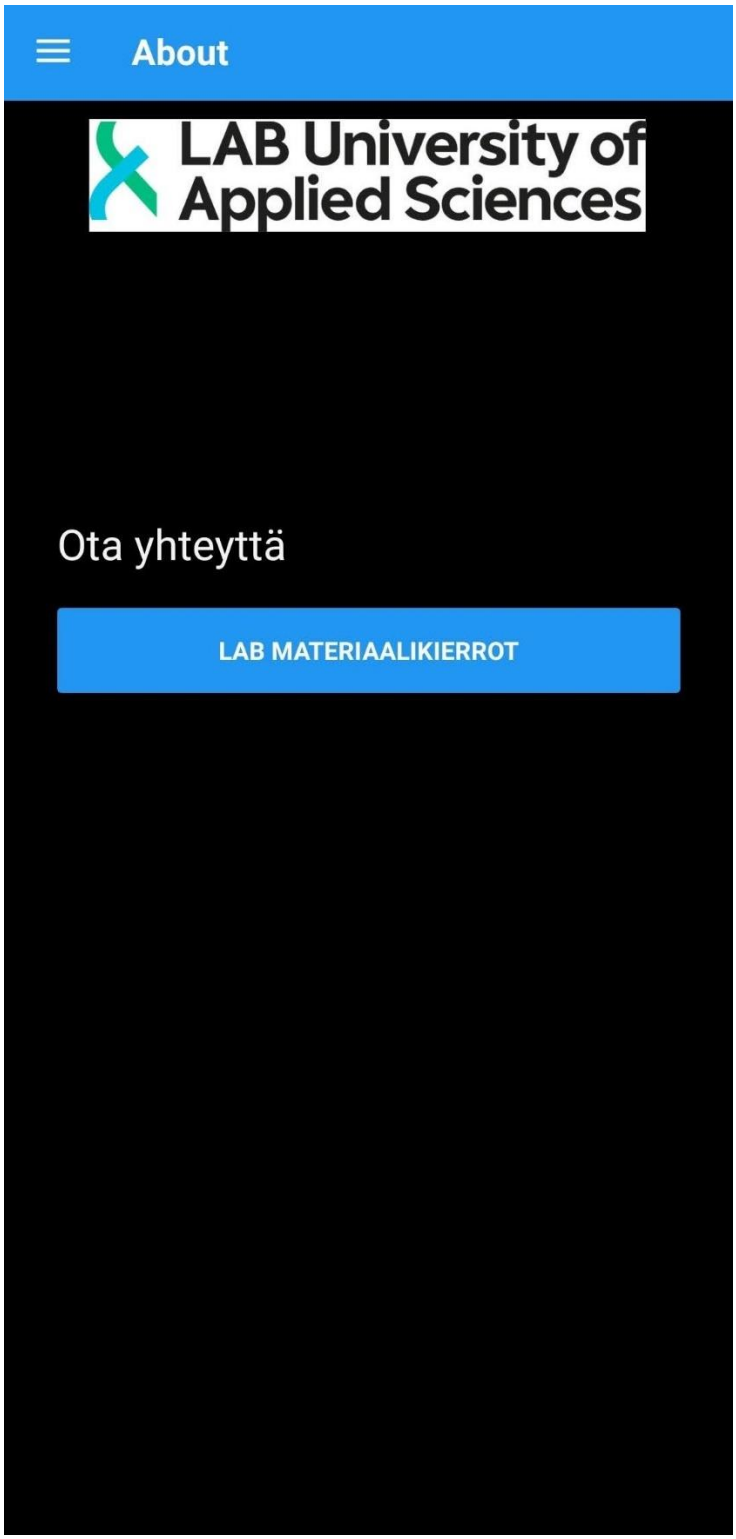


Kuva 21. Tunnistamisen vaiheet mobiilisovelluksessa

Kuva 21 on esitetty vaiheet, jota mobiilisovelluksessa tapahtuu kuvan ottamisen jälkeen siihen pisteeseen, että tulos ja kuva esitetään käyttäjälle. Kun kuva on otettu, se luetaan välimuistista, johon kamerasovellus tallentaa sen kuvan ottamisen jälkeen. Tämän jälkeen kutsutaan IBackgroundDependency luokan ClassifyImage metodia, joka ottaa kuvan input parametrina vastaan. Se suorittaa vastaavan kutsun laitteen puolella, tässä tapauksessa Androidin, IBackgroundDependency_Android luokassa välittäen kuvan sinne.

Android puolella kutsutaan TensorFlowClassifier luokan Classify metodia. Classify metodi luokittelee kuvan ja palauttaa vastauksen listana, jossa ensimmäisenä on suurimman todennäköisyyden saanut luokitus. Lista palautetaan yhteiselle alueelle, jossa näytetään kuva ja luokittelun lopputulos.

7.5 Info näkymä



Kuva 22. Info näkymä

Kuva 22 esitetty infosivu on hyvinkin pelkistetty. Mukana on LAB-ammattikorkeakoulun logo sekä nappi, joka ohjaa LAB materiaalikierrot -sivulle. Sivulta löytyy ajantasaiset yhteystiedot, eikä sovellusta tarvitse päivittää, jos yhteyshenkilö muuttuu.

7.6 Android spesifit asiat

Kaikkia toiminnallisuuksia ei voi toteuttaa pääprojektiin yhteisellä koodipohjalla, vaan osa toiminnoista on toteutettava erikseen Androidille ja iOS:lle. Yksi sellainen toiminnallisuus tässä projektissa on TensorFlow luokittelija. Se toimii vain Android puolella ja iOS puolelle tarvitsee toteuttaa vastaava Core ML versiona.

7.6.1 TensorFlow luokittelija

TensorFlow Lite toimii ainoastaan Android puolella, jonka vuoksi tämä toiminnallisuus ei ole käytössä iOS puolella. iOS projektiin pitää toteuttaa Applen oma Core ML versio.

TensorFlow luokittelijan toteutuksessa käytettiin hyödyksi Jayme Singletonin julkaisemaa blogikirjoitusta, TensorFlow Lite -mallin hyödyntämiseen Xamarin -pohjaisissa toteutuksissa (Hindrikes 2020). Blogijulkaisun kirjoittaja Daniel Hindrikes on Microsoft MVP, joka toimii myös ohjelmistokehittäjänä, sekä -arkkitehtina. Blogikirjoituksessa TensorFlow Lite -mallin tuottamiseen käytettiin Azure Custom Vision Serviceä, jota tässä toteutuksessa ei käytetty, vaan tässä käytettiin Pythonilla tuotettua TensorFlow Lite -mallia.

Pääprojektin ja Android projektin välissä rajapinta, jonka avulla pääprojektista voidaan kutsua Android projektissa olevan TensorflowClassifierin Classify -funktiota.

TensorflowClassifier -luokka koostuu kolmesta funktiosta: Classify, GetModelAsMappedByteBuffer ja GetPhotoAsByteBuffer.

7.6.2 Classify -funktio

Classify funktio on ensimmäinen funktio, jota kutsutaan TensorflowClassifier -luokasta. Se ottaa sisäänsä kuvan tavulistana. Ensin TensorFlow Lite -malli konvertoidaan Java.Nio.MappedByteBuffer -muotoon, joka on Xamarin.TensorFlow.Lite.Interpreter -tulkin vaatima muoto käyttämällä GetModelAsMappedByteBuffer -funktiota. Tätä hyödyntäen luodaan on Xamarin.TensorFlow.Lite.Interpreter -tulkki. Tulkista haetaan mallin ensimmäinen tensori, jota käytetään päättelemään minkä kokoinen (leveys ja korkeus pikseleinä) kuvan pitää olla.

Tämän jälkeen kutsutaan GetPhotoAsByteBuffer -funktiota, jolle annetaan parametreina kuva, kuvan leveys ja kuvan korkeus. Tämä palauttaa kuvan pienennettynä.

Kun kuva on pienennetty, luetaan koneoppimismallin luokkien nimet (Kude, Neule) tekstimuotoiseen listaan labels.txt -tiedostosta. Tämä muunnetaan Java.Lang.Object -muotoon, koska se on Xamarin.TensorFlow.List.Interpreter vaatima muoto.

Tämän jälkeen ajetaan Xamarin.TensorFlow.Lite.Interpreter -tulkki, joka on muodostettu käyttäen TensorFlow Lite -mallia. Run -funktiolle annetaan parametreina pienennetty kuva sisääntuloparametrina ja ulostuloparametriksi Java.Lang.Object olio, joka sisältää luokkien (kankaiden) nimet. Tulos luetaan uuteen listaan.

Tämän jälkeen tulos käydään läpi ja luodaan lista, jossa on luokkien nimet ja annetaan niille todennäköisyydet, millä todennäköisyydellä kyseessä on kyseinen kangas.

Lopputuloksena on tulosjoukko, jossa on kankaiden nimet ja nimeä vastaava todennäköisyys. Tämä lista palautuu lopputuloksena takaisin pääprojektille.

8 Koneoppimismalli

Jotta kankaiden tunnistaminen on mahdollista, on koneoppimismallin oltava hyvä. Koneoppimismallin opettaminen olikin eniten iteraatioita vaatinut työvaihe. Jotta mallin tekeminen helpottuu, käytettiin hyväksi TensorFlow Liteä ja Keras kirjastoa.

Jotta mallin toteuttamisessa päästiin nopeasti vauhtiin, käytettiin runkona Googlen TensorFlow -ohjetta mallin luomiseen (Google LLC. 2022.). Tämä ei kuitenkaan toiminut sellaisenaan kovin hyvin, vaan vaati muutoksia hyperparametreihin, ynnä muihin. konfiguraatioihin. TensorFlow ohje antoi silti hyvän pohjan koneoppimismallin luontiin, eikä mennyt liikaa aikaa alkuunpääsyssä.

8.1 Opetuskoodin rakenne

Koodi on toteutettu Jupyter Notebookiin. Sillä saadaan jaoteltua koodi järkeviin palasiin ja saa laitettua väliin Markdown tekstiä, niin halutessaan. Muutoin koodi voisi olla perinteinen Python tiedosto.

Koodissa on määritelty muuttujat kuvan käsiteltävän kuvajoukon koolle, kuvan pysty ja vaakaresoluutiolle, pienennyskerroin, validointijoukon koko, opetuskierrosten määrä, sekä käytetäänkö ylisovittamisen estoon tarkoitettuja funktioita. Kuvien koolle on tehty omat muuttujat, jotta vähennetään samojen arvojen toistoa ja virheen mahdollisuutta. Kuvia varten on myös luotu pienennyskerroin muuttuja. Kuvat ovat kooltaan 2560 pikseliä korkeita ja 1440 pikseliä leveitä, niin pienennyskertoimella neljä, saadaan kuvista 640 pikseliä korkeita ja 360 pikseliä korkeita. Pienentäminen on osittain sen takia, että pienempien kuvien käsittely on nopeampaa ja liian tarkoissa kuvissa voi näkyä liikaa tunnistamisen kannalta epäoleellista, kuten pöly. Pienennyskerroin on sen vuoksi, että kuvasuhde pysyy oikeana, eikä tarvitse laskea sitä erikseen. Validointijoukon kokoa käytetään myös useammassa kohdassa koodia, niin se on lisätty omaan muuttujaan.

Muuttujien jälkeen luodaan opetustietojoukko. Tietojoukossa käytetään 20 % validointijakaumaa. Opetustietojoukossa käytetään Keras kirjaston image_dataset_from_directory -funktiota. Kun opetustietojoukko on tehty, tehdään validointitietojoukko samoilla parametreilla.

Luokkien nimet luetaan testidatasetistä. Visualisointitarkoituksessa näytetään yhdeksän kuvan kooste, joka on luotu opetustietojoukosta. Kuvat visualisoidaan Matplotlib kirjastolla. Kuva 23 on esimerkki tästä visualisoinnista.

Kuva 23. Opetuskuvien visualisointi Matplotlib -kirjastolla

Tämän jälkeen määritellään käyttöön tietojoukon välimuisti, joka nopeuttaa opetusta, koska kuvat luetaan keskusmuistiin sen sijaan, että ne luettaisiin joka kerralla tallennusmedialta. Vaikka nykypäivän PCIe versio 4.0 NVMe SSD -levyt ovat todella nopeita jopa 7000 megatavun sekuntivauhdin lukunopeuksissa, ne eivät vedä vertoja nykypäiväisille keskusmuisteille. Jos tietokoneessa ei ole riittävästi keskusmuistia kaikkien kuvien tallentamiseen, välimuisti tehdään levylle, joka on silti nopeampaa, kuin lukea kuvat suoraan levyltä.

Mallin muodostamiseen käytetään kaksiulotteisia konvoluutiokerroksia. Kerroksia on yhteensä kolme, joista kaikissa käytetään aktivaatiofunktiona Leaky ReLU (Rectified Linear Unit), jolla saatiin parempi lopputulos kuin tavallisella ReLU:lla. Erona ReLU:n ja Leaky ReLU:n välillä on se, että Leaky ReLU ei nollaa kaikkia negatiivisia arvoja, vaan ne saavat pienen arvon.

Seuraavaksi annetaan mallin käännösparametrit. Optimointina käytetään Adam (Adaptive moment estimation) optimointia. Hävikkifunktiona käytetään Sparse Categorical

Crossentropy funktiota. Tämän jälkeen tehdään mallin opetus. Opetukselle annetaan parametreiksi opetustietojoukko, validointitietojoukko sekä opetuskierrosten määrä.

Kun mallinmuodostus on valmis, se tallennetaan tflite -tiedostoksi. Tflite -tiedosto on itse malli, jota voidaan käyttää esimerkiksi puhelinsovelluksessa. Mallia voitaisiin käyttää myös muualla.

8.2 Kuvamäärä ja sen tuomat haasteet

Kuvia oli yhteensä yli 7000 kpl ja kokoa oli yhteensä yli 25 gigatavua. Pelkällä suorittimella kuvien pohjalta tehtävä opetus oli todella hidasta ja opetuskierrosten määrän lisääminen olisi kasvattanut opetusaikaa aivan liikaa. Tämän vuoksi oli kokeiltava muita vaihtoehtoja pelkän suorittimen tilalle.

Alun perin koneoppimismalli luotiin käyttäen Lenovo ThinkCentre M720q Tiny minitietokonetta, jossa on 6-ytiminen Intel Core i5-8400T suoritin ja 16 gigatavua keskusmuistia, mutta ei ollenkaan erillistä näytönohjainta. Suuren kuvamäärän takia opetuskierrokset olivat sen verran hitaita, että opetuskierroksia lisäämällä, olisi mennyt tunteja mallin opettamiseen. Muutoin pitkä kesto ei olisi ongelma, mutta alkuvaiheessa hyperparametrien, aktivaatiofunktioiden ja konvoluutioverkkojen hienosäätämisen vuoksi mallia joutuu muodostamaan useita kertoja, jolloin pitkä kesto tietenkin kertaantuu.

Tämän ongelman takia päädyttiin siirtämään mallinmuodostus tehokkaammalle pöytäkoneelle. Pöytäkoneessa on 6-ytiminen, 12-säikeinen Intel Core i7-8700K suoritin, 32 gigatavua keskusmuistia sekä erillinen 8 gigatavun muistilla oleva NVIDIA GeForce RTX 2080 Super -näytönohjain.

Arkkitehtuuri	**Minimi (s)**	**Maksimi (s)**	**Keskiarvo (s)**	**Yhteensä (s)**
Intel Core i5-8400T -suoritin	58	93	62,4	624
NVIDIA GeForce RTX 2080 Super -näytönohjain	3	23	5	50
Ero	**55**	**70**	**57,4**	**574**

Taulukko 2. Näytönohjaimen ja suorittimen nopeusero opetuksessa

Mielenkiinnon vuoksi ajettiin sama opetuskoodi molemmilla ja vertailtiin tuloksia, jotka näkyvät Taulukko 2:ssa. Tästä voidaan huomata, että ero on huomattava. Näytönohjaimen oli yli 12x nopeampi, kuin suoritin. Maksimikestot tulee ensimmäisestä opetuskierroksesta, jolloin lähtökuvat eivät ole vielä muistissa, vaan kuvat joudutaan lukemaan suoraan levyltä. Tässä tosin tulee sellainen pieni epäkohta vertailtavien tietokoneiden välillä, että näytönohjaimellisessa tietokoneessa on 32 gigatavua keskusmuistia, johon kuvat mahtuvat, mutta

toisessa koneessa keskusmuistia on vain 16 gigatavua, johon kuvat eivät mahdu. Sen tarkempaa analyysiä ei tehty, mikä vaikutus keskusmuistin määrällä on nopeuteen.

Jotta muistin vaikutuksen tuloksiin saatiin rajattua ulkopuolelle, ajettiin toinen testi 10 opetuskierroksella ja 4x pienemmällä kuvakoolla, kuin alkuperäiset (1440x2560) -> (360x640). Kuvina käytettiin pelkästään lähikuvia, joita oli 448 kpl ja koko yhteensä 503 megatavua. Kuten Taulukko 3 voidaan huomata, oli näytönohjain silti aivan ylivertainen suorittimeen nähden yli 20x nopeudella.

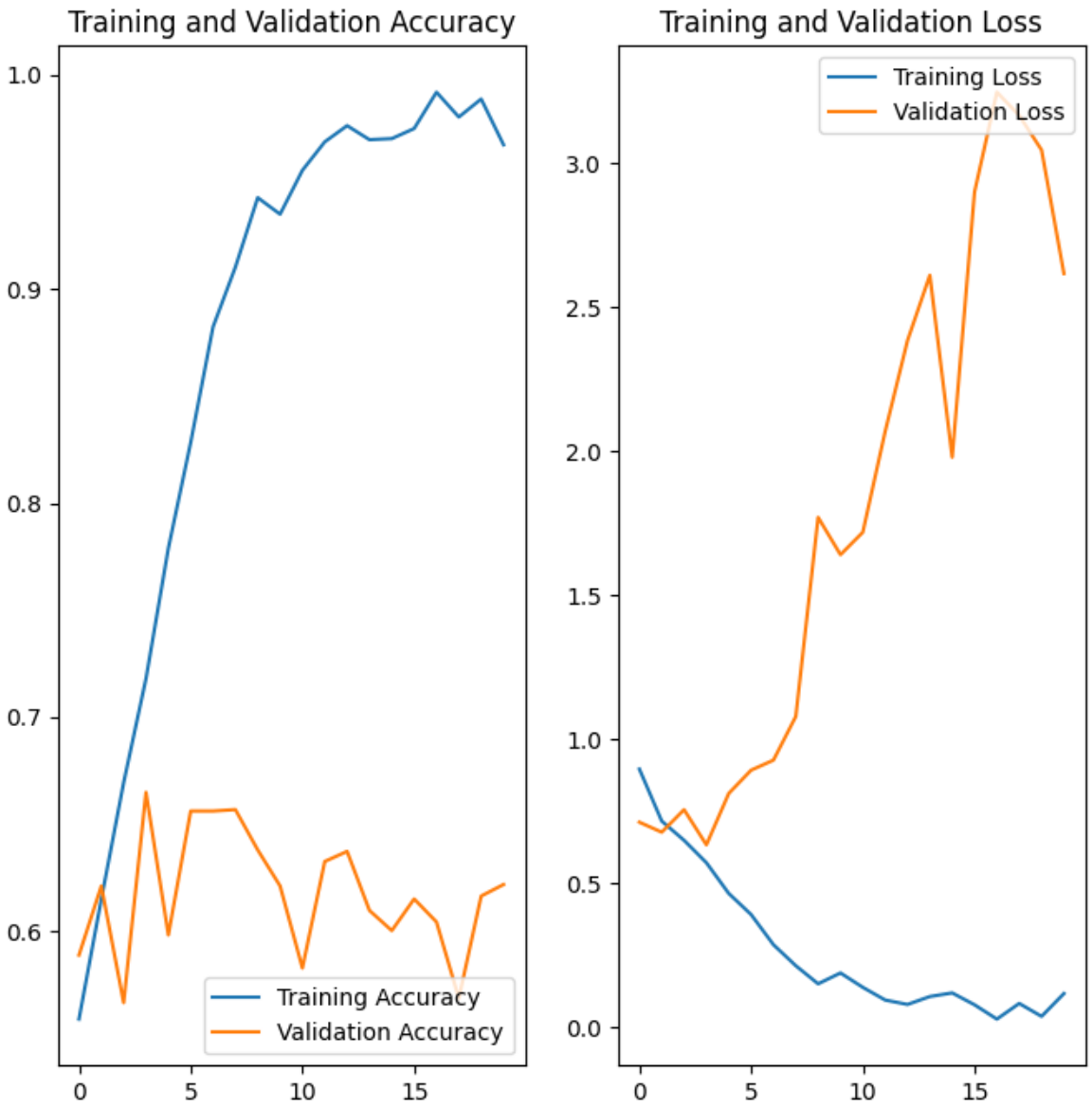
Arkkitehtuuri	**Minimi (s)**	**Maksimi (s)**	**Keskiarvo (s)**	**Yhteensä (s)**
Intel Core i5-8400T -suoritin	26	28	26,2	262
NVIDIA GeForce RTX 2080 Super -näytönohjain	1	4	1,3	13
Ero	**25**	**24**	**24,9**	**249**

Taulukko 3. Näytönohjaimen ja suorittimen nopeusero muistimäärärajoite huomioiden

8.3 Mallin testaaminen

Kun malli oli opetettu, oli sitä syytä testata tunnistustarkkuuden varmentamiseksi. Testaamista varten kehittäjä otti yhteensä 23 lähikuvaa kankaista, joista 12 oli kuteita ja 11 neuloksia. Kuvat oli nimetty kankaan nimellä ja juoksevalla numerolla. Kuvat käytiin läpi Python koodissa ja annettiin mallille tunnistettavaksi. Kuvan nimeä käytettiin vertaamaan mallin antamaa vastausta tiedoston nimeen, jolla saatiin lopullinen tunnistusten määrä suhteessa kuviin.

Mallia opetettiin monilla eri hyperparametreilla, mutta parhaimmillaankin tunnistamisessa päästiin alle 70 % tunnistustarkkuuteen. Joissain onnistuneissa tunnistuksissa varmuus oli vain hiukan yli 50 %, joten arvaus olisi melkein yhtä hyvä. Tämä ei ole millään tasolla riittävän tarkka, että sen kanssa pystyisi tekemään luotettavia päätöksiä. Kuva 24 näkyy, että opetuksen tarkkuus ja hävikki olivat hyvällä tasolla, mutta validointi oli huono.



Kuva 24. Opetuksen ja validoinnin tarkkuus, sekä opetuksen ja validoinnin hävikki

9 Yhteenveto ja pohdinta

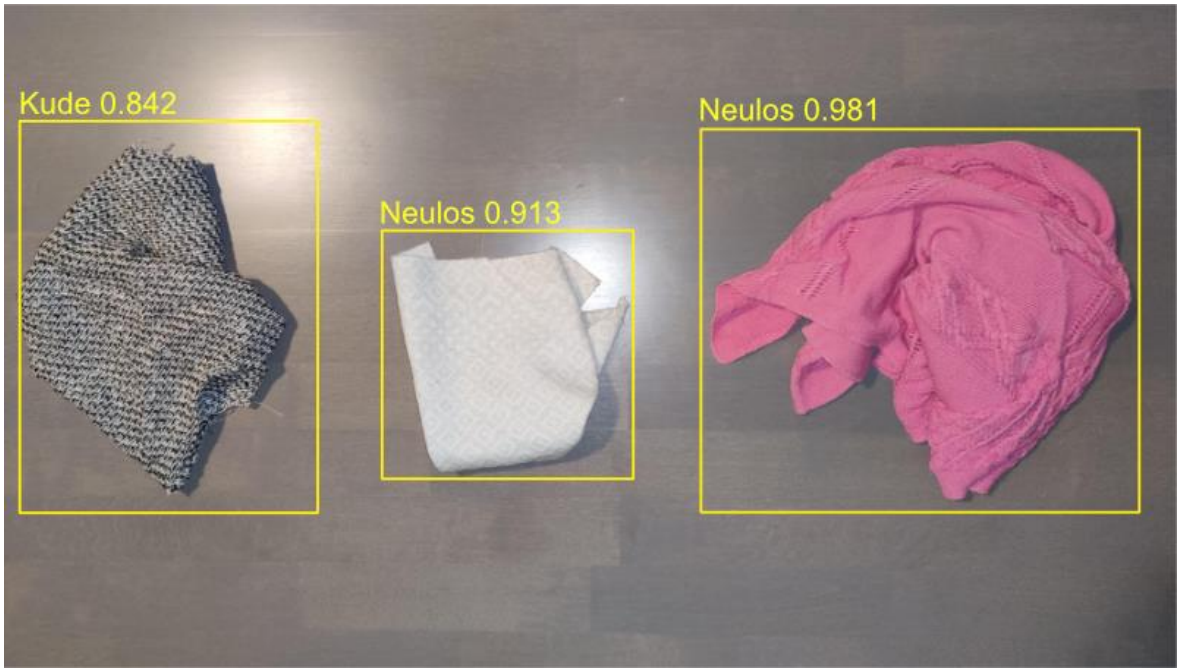
Vastaus tutkimuskysymyksenä olleeseen kysymykseen, että millä teknologialla kankaita ja kuteita valokuvasta tunnistava mobiilisovellus on mahdollista tehdä, ei rajaudu pelkästään yhteen teknologiaan. Näistä kuitenkin Xamarin valikoitui tämän opinnäytetyön teknologiaksi, koska se täytti kaikki vaatimukset, joita teknologialta odotettiin. Se onko Xamarin helpoiten omaksuttava teknologia tämän toteuttamiseen, riippunee paljon kehittäjän omasta taustastaan ja mieltymyksistään. Tämän opinnäytetyön kirjoittaja on tehnyt töitä vuosia pääasiassa Microsoftin teknologioilla, niin se oli sen vuoksi tähän luontainen valinta. Kehittäjä, jonka osaaminen on enemmän front-end kehittämisen puolelta, olisi ehkä valinnut teknologiaksi Ionic Angularin tai React Nativen. Pilvipalvelut mahdollistaisivat muun muassa koneoppimismallin parantamisen käyttäjien ottamia kuvia hyödyntämällä. Koneoppimismalli olisi myös mahdollista päivittää suoraan käytetyssä pilvipalvelussa, eikä vaadi ohjelman uuden version julkaisua markkinapaikoille.

Toinen tutkimuskysymys oli, että mikä koneoppimismalli sopii parhaiten kankaiden ja kuteiden mobiiliin tunnistustehtävään. Kun otetaan huomioon juoksevien kustannusten minimointi, sekä mallin opettamisen helppous, teknologioiksi valikoitui TensorFlow ja Keras kombinaatio. Keraksen syväoppimismalleille tarkoitettu Sequential model soveltui tähän tarkoitukseen hyvin, koska mallin sai TensorFlow Lite mobiilikirjaston avulla sellaiseen muotoon, että sen saa käyttöön mobiilisovelluksella. Vaihtoehtoisia teknologioita on useampia ja teknologiaksi kannattaakin valita sellainen, joka on kehittäjille jo ennestään tuttua. Kuitenkin, jos lopputulos tulee toimimaan liukuhihnalla, eikä ole tarvetta mobiilisovellukselle, Unity tarjoaa kirjastoja, joilla on mahdollista tunnistaa videokuvasta asioita. Kuva 25 on havainnollistettu, millainen kankaidenlajitteluliukuhihnalla toimiva tunnistus voisi olla tulevaisuudessa. Tämä vaatisi todella tarkan kameran ja laitteiston, joka kykenee käsittelemään kameran tuottamaa videokuvaa. Laitteiston pitäisi kyetä tunnistamaan kankaat kuvasta, sekä analysoimaan mikä kangas on kyseessä, kaikki reaaliaikaisesti. Monialustatukea ajatellen on hyvä tiedostaa, että iOS version kehittäminen vaatii Apple tietokoneen.

Eri kankaiden tunnistaminen toisistaan kuvan perusteella on hankalaa. Kaksiulotteisesta kuvasta ei pysty erottamaan, miten lanka kulkee kankaassa. Esimerkiksi, on huomattavasti helpompaa tunnistaa kuvasta, onko siinä pöytä, kuin tunnistaa onko kuvassa pöytä, jossa on naarmu. Tämän opinnäytetyön tuloksena paras malli tunnisti kankaan oikein 70 % testikuvista ja joidenkin kuvien osalta varmuus oli todella heikko, vaikka lopullinen tunnistus olikin oikein. Testikuvat oli otettu salamalla, eivätkä ne olleet epäselviä, joten niiden puolesta mallilla olisi pitänyt olla hyvä mahdollisuus tunnistaa oikein. Tämän opinnäytetyön kehittämisen tarkoituksena oli luoda toimiva koneoppimismalli, joka lopputuloksena syntyikin.

Jatkokehityshankkeena voisi olla mallin tunnistustarkkuuden parantaminen, mallin muodostamisen hyperparametreja muokkaamalla.

Luotettavan lopputuloksen aikaansaamiseksi, pitäisi olla käytössä enemmän resursseja, tarkoittaen aikaa ja rahaa, kuin yksittäisellä tekijällä on mahdollista panostaa. 3D-kuvantaminen voisi auttaa analysoimaan kankaan rakennetta paremmin, jonka pohjalle koneoppimismallin voisi toteuttaa. Hyödyllisintä olisi, jos kokonaisuuden tekemiseen olisi tiimi, joka koostuu erikseen käyttöliittymäkehittäjistä, koneoppimismallin kehittäjistä, sekä materiaalitekniikkoja. Näin olisi valmiudet saada lopputuote, joka on valmis markkinoille.



Kuva 25. Esimerkki tunnistamisen lopullisesta muodosta

Lähteet

Audevart, A., Banachewicz, K. & Massaron, L. 2021. Machine Learning Using TensorFlow Cookbook. Birmingham: Packt Publishing Ltd.

Bampakos, A. & Deeleman, P. 2020. Learning Angular - Third Edition. Birmingham: Packt Publishing Ltd.

Barnes, J. 2015. Azure Machine Learning. Redmond, Washington: Microsoft Press.

Cheng, F. 2018. Build Mobile Apps with Ionic 4 and Firebase. Californi: Apress Media, LLC.

Dabit, N. 2019. React Native in Action. Shelter Island: Manning Publications Co.

Google LLC. Tensorflow - Image classification. Viitattu 25.11.2022. Saatavissa https://www.tensorflow.org/tutorials/images/classification

Hindrikes, D. 2020. Android Image Classification with TensorFlow Lite & Azure Custom Vision Service. Singleton, J. Viitattu 23.11.2022. Saatavissa https://devblogs.microsoft.com/xamarin/image-classification-xamarin-android

Keller, J. Deep Learning. 2019. Cambridge, Massachusetts: The MIT Press.

Leiva, A. 2017. Kotlin for Android Developers. Victoria: Ruboss/Leanpub.

Maddalone, A., Moocarme, M. & So A. 2021. The TensorFlow Workshop. Birmingham: Packt Publishing Ltd.

McMahon, P. 2021. Machine Learning Engineering With Python. Birmingham: Packt Publishing Ltd.

Mirjalili V., Raschka S. 2019. Python Machine Learning - Third Edition. Birmingham: Packt Publishing Ltd.

Petzold, C. 2016. Creating Mobile Apps with Xamarin.Forms. Redmond, Washington: Microsoft Press.

Ranjan, S. & Dr. Senthamilarasu, S. 2020. Applied Deep Learning and Computer Vision for Self-Driving Cars. Birmingham: Packt Publishing Ltd.

Simaranjit, B. & SrinivasMadhav, G. 2019. Programming in C# Exam 70-483 (MCSD) Guide. Birmingham: Packt Publishing Ltd.

Spenser, D. 2011. Knitting technology. Cambridge: Woodhead Publishing Limited.