Tampereen ammattikorkeakoulu

Syvän neuroverkon päättelynopeus mobiililaitteella

Roope Uitto

OPINNÄYTETYÖ
Lokakuu 2022

Tietojenkäsittely
Ohjelmistokehitys

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

UITTO, ROOPE:
Syvän neuroverkon päättelynopeus mobiililaitteella

Opinnäytetyö 44 sivua, joista liitteitä 0 sivua
Lokakuu 2022

Tarve ja halu hyödyntää tekoälyä resurssirajoitteisissa laitteissa, kuten älypuhelimissa on lisääntynyt. Erityisesti syvien neuroverkkojen suorituskyky halutaan tuoda käytettäväksi myös mobiilisovelluksissa. Neuroverkkojen ajamisen laskennallinen vaativuus on ongelma rajoitetun prosessointitehon omaavissa älypuhelimissa. Ongelmaa on kierretty ulkoistamalla neuroverkkojen päättely palvelimille, mutta se on tuonut mukanaan ongelmia ja rajoitteita: datan siirtoon ei ole käytössä verkkoyhteyttä tai yhteys on hidas, palvelimien hankinta ja ylläpito tuottaa kustannuksia, ja datan säilyttämisessä ja käsittelyssä tulee huolehtia EU:n tietosuoja-asetuksen noudattamisesta.

Neuroverkkojen ajaminen paikallisesti laitteessa voi pienentää edellä mainittuja ongelmia tai poistaa ne jopa kokonaan. Tämän opinnäytetyön tarkoituksena oli selvittää, kuinka nopeasti neuroverkko päättelee nykyaikaisilla älypuhelimilla, kun käytetään resurssirajoitteisille laitteille tarkoitettua neuroverkkomallia. Tähän tarkoitukseen koulutettiin syvä neuroverkko kohteen tunnistus -tehtävään, jossa tunnistettiin levytangon pää. Toisena tarkoituksena oli selvittää, kuinka reaaliaikaisesti levytangon päätä voidaan seurata, kun seuranta toteutetaan peräkkäisten kuvien tunnistuksena videodatasta.

Levytangon pään tunnistukseen mukautettu neuroverkkomalli lisättiin React Native -sovelluskehyksellä toteutettuun mobiilisovellukseen, joka asennettiin testattaviin älypuhelimiin. Päättelynopeus testatuilla älypuhelimilla oli noin 330–550 ms, joka vastaa kuvataajuutena noin 2–3 FPS. Saavutettu päättelynopeus on kohtuullinen ja osoittaa, että neuroverkon päättelynopeus on riittävä, jos ei vaadita reaaliaikaisuutta tai nopeaa vasteaikaa. Jos halutaan seurata nopeasti liikkuvaa kohdetta, kuten levytangon päätä, neuroverkon tulisi kyetä vähintään 40–50 ms päättelynopeuteen. Tämän opinnäytetyön perusteella perustason älypuhelimilla ei pystytä reaaliaikaiseen kohteen seurantaan.

Asiasanat: tekoäly, neuroverkko, mobiilisovellus

ABSTRACT

Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Software Production

UITTO, ROOPE:
The inference speed of deep neural network on mobile device

Bachelor's thesis 44 pages, appendices 0 pages
October 2022

The desire and need to use artificial intelligence on devices with low processing power, like mobile phones, has increased. Powerful deep neural networks are in special demand, but their high computing needs are a problem for mobile phones. Previously the problem has been solved by using external processing on servers, but this doesn’t come without problems. The network connection might be slow or nonexistent, server acquisition and maintenance have costs, and data handling most obey legislation.

By having the neural network infer on device, these problems shrink in size or might disappear altogether. The purpose of this thesis was to find out the inference speed of modern mobile phones, when using neural network model that is designed to be used on devices with lower processing power. For this purpose, a network was trained to detect barbell end from images. A secondary aim for this thesis was to assess the real-time aspect of object tracking by finding out how well the barbell end is tracked through consecutive detections from video.

A neural network trained with custom data was added to the source code of mobile applications develop with React Native framework. The application was installed on tested mobile devices, which reached inference speeds of approximately 330–550 ms that corresponds to about 2–3 FPS. The reached inference speeds are reasonable for applications that doesn’t require low latency or real-time tracking. For real-time tracking the inference speed should be at least 40–50 ms. According to this thesis, basic smart phones aren’t capable of real-time object tracking through object detection with neural network.

Key words: artificial intelligence, neural network, mobile application

SISÄLLYS

LYHENTEET JA TERMIT

COCO	Common Objects in Context (kuvien datajoukko)
harjoitusdata	Joukko syöte–tulos -pareja, joiden perusteella koneoppimismalli oppii ja testaa osaamistaan.
kohteen seuranta	Kohteen tunnistaminen ja seuranta peräkkäisissä kuvissa.
kohteen tunnistus	Kohteen luokan ja sijainnin tunnistaminen kuvasta.
koulutusdata	Osa harjoitusdataa, jonka avulla koneoppimismallin oppiminen toteutetaan.
neuroni	Toiminnallinen yksikkö, joka prosessoi sille annetuista syötteistä yhden tuloksen.
neuroverkko	Kerroksittainen rakenne, joka muodostuu neuroneista ja niiden välisistä yhteyksistä.
syöte	Data, joka annetaan koneoppimismallille analysoitavaksi.
TensorFlow	Ohjelmakirjasto koneoppimisen ja erityisesti neuroverkkojen käyttöön
tensori	n-ulotteinen vektori
testidata	Osa harjoitusdataa, jonka avulla koneoppimismallin oppimista testataan, kun se on koulutettu koulutus- ja validointidatalla.
tulos	Data, jonka koneoppimismalli tuottaa syötteestä.
validointidata	Osa harjoitusdataa, jonka avulla koneoppimismallin oppimista testataan koulutuksen aikana. Validointidataa käytetään myös lopulliseen koulutukseen.
YOLO	You Only Look Once (neuroverkkomalli)

1 JOHDANTO

Tekoälyn hyödyntäminen liiketoiminnassa on trendikästä ja sillä on suuri potentiaali kehittää olemassa olevia palveluita tai prosesseja, sekä luoda uutta liiketoimintaa. Mobiililaitteita käytetään lähes joka puolella maailmaa, joten mobiilisovellusten kehittäminen on suuri liiketoiminta-alue, jossa tekoälyä halutaan myös hyödyntää. Viime vuosikymmenen aikana varsinkin neuroverkot ovat kehittyneet edistyksellisiksi ja niitä käytetään erityisen paljon konenäköön liittyvissä sovelluksissa, mutta myös esimerkiksi luonnollisen kielen prosessoinnissa.

Syvien neuroverkkojen ajaminen on kuitenkin laskennallisesti hyvin vaativaa, varsinkin kun neuroverkkojen koot suurenevat ja niiden arkkitehtuurit monimutkaistuvat parempien tulosten toivossa. Mobiililaitteiden prosessointiteho on ollut vielä yleisesti riittämätöntä, jotta syviä neuroverkkoja voitaisiin ajaa paikallisesti laitteessa kohtuullisen lyhyellä viiveellä. Neuroverkoista on tehty viime aikoina kevyempiä, kun on ymmärretty niiden potentiaali ja tarve myös resurssirajoitteisissa laitteissa, kuten mobiili- ja IOT-laitteissa.

Jos on vaadittu nopeaa vasteaikaa tai reaaliaikaisuutta mobiililaitteessa, niin neuroverkkojen prosessointi on ulkoistettu erillisille palvelimille. Tästä syntyy rajoituksia ja ongelmia: datan siirtäminen verkon yli voi olla hidasta tai verkkoa ei ole välttämättä saatavilla, oman tai kolmannen osapuolen palvelimen hankkiminen ja ylläpito tuottavat kustannuksia, ja käyttäjän dataa käsiteltäessä on huolehdittava sen oikeanlaisesta säilyttämisestä ja käsittelystä EU:n tietosuoja-asetuksen mukaisesti. Datan prosessointi paikallisesti käyttäjän laitteessa tekisi edellä mainituista ongelmista pienempiä tai saattaisi poistaa ne kokonaan.

Tämän työn tarkoituksena on selvittää kuinka nopeasti nykyaikaiset älypuhelimet kykenevät ajamaan syvää neuroverkkoa, kun käytetään resurssirajoitteisille laitteille tarkoitettua neuroverkkomallia. Selvitystä varten koulutetaan syvä neuroverkko kohteen tunnistus -tehtävään, jossa tunnistetaan levytangon pää. Kun tunnistusta suoritetaan peräkkäin puhelimen videokameran datasta, niin tehtävä muuttuu kohteen seurannaksi tunnistuksen kautta. Toisena tarkoituksena onkin selvittää, kuinka reaaliaikaiseen kohteen seurantaan pystytään.

2 TEKOÄLY JA KONEOPPIMINEN

2.1 Tekoäly

Tekoäly tieteenalana on yksi uusimmista ja siten se myös kehittyy vielä nopeaan tahtiin. Se on myös yksi suosituimmista vastauksista, kun kysytään muilla tieteenaloilla toimivilta, minkä parissa he haluaisivat työskennellä. Siihen liittyvää työtä ja projekteja alettiin toteuttaa pian toisen maailmansodan jälkeen, ja termi itsessään keksittiin vuonna 1956. (Russell & Norvig 2016, 1.)

Ihmiselle on luontaista ajatella ja sen kautta tarkastella, ymmärtää, ennustaa ja muokata ympäröivää maailma. Ajattelu on ilmentymä ihmisen älykkyydestä, mutta emme kuitenkaan täysin ymmärrä miten me ajattelemme. Tekoälytieteenalan tarkoituksena ei ole vain yrittää ymmärtää ihmisen ajattelua, mutta myös luoda älykkäitä itsenäisiä kokonaisuuksia. (Russell & Norvig 2016, 1.) Tekoälytermi onkin tässä suhteessa melko helppo ymmärtää käsittämään jotain keinotekoisesti luotua ja ihmisen älykkyyttä matkivaa asiaa. Toisaalta termi voi johtaa myös harhaan, koska sen perusteella voi käsittää tekoälyyn liittyvän älykkyyden olevan ns. yleistä älykkyyttä, joka sisältää ihmismäisen tietoisuuden. Yleensä tekoälyt ovat kuitenkin kykeneviä tekemään vain muutamia asioita älykkäästi.

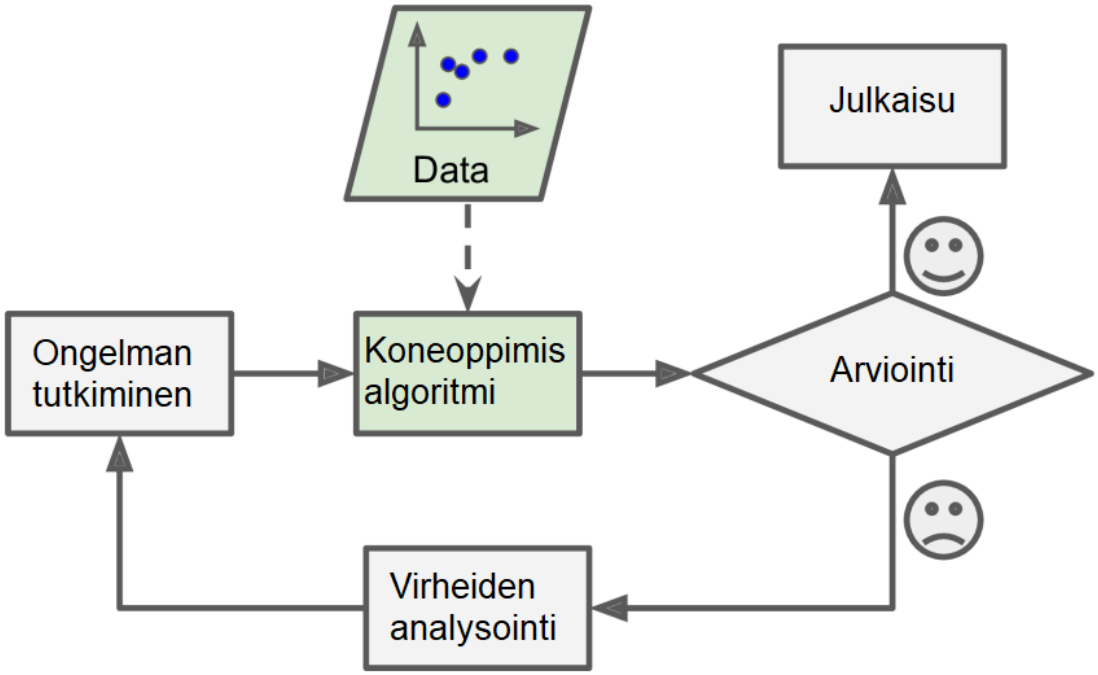
Russell & Norvig (2016, 2) ovat keränneet tekoälyn määritelmiä kirjallisuudesta (taulukko 1), mitkä on jaettu neljään eri kategoriaan: ihmismäiseen ajatteluun ja toimintaan, sekä järkevään ajatteluun ja toimintaan. Tässä voidaan huomata tekoälyn määrittelyn ja luomisen ongelmat, kun sen pitäisi toimia ihmismäisesti, mutta samalla järkevästi.

TAULUKKO 1. Tekoälyn määritelmiä neljässä eri kategoriassa (Russell & Norvig 2016, 2). Määritelmät on kerätty kirjallisuudesta kirjoittajien toimesta ja vapaasti suomennettu opinnäytteen kirjoittajan toimesta.

Ihmismäinen ajattelu	**Järkevä ajattelu**
”Jännittävä uusi kokeilu saada koneet ajattelemaan … koneita, joilla on mieli, kirjaimellisesti.” ”Toimintojen automatisointi, jotka yhdistetään ihmismäiseen ajatteluun kuten päätösten teko, ongelmanratkaisu, oppiminen …”	”Päänsisäisten kykyjen laskennallinen mallintaminen.” ”Laskenta, joka mahdollistaa havainnoimisen, järkeilyn ja toimimisen.”
Ihmismäinen toiminta	**Järkevä toiminta**
”Taito luoda koneita, jotka suorittavat toimintoja, joiden suorittaminen vaatii älykkyyttä ihmiseltä.” ”Ala, jossa tutkitaan, kuinka koneet saadaan tekemään asioita, joissa ihmiset ovat tällä hetkellä parempia.”	”Laskennallinen älykkyys on ala, jossa suunnitellaan älykkäitä agentteja.” ”Tekoäly liittyy luomusten älykkääseen käyttäytymiseen”

2.2 Koneoppiminen

Koneoppiminen on yksi tekoälyn osa-alue, mikä on syntynyt tarpeesta automatisoida datan analysointia nykyisen massadatan (big data) aikakaudella. Koneoppiminen voidaan lyhyesti määritellä joukoksi menetelmiä, joilla voidaan ennustaa jonkin datan tuleva tila olemassa olevasta datasta tunnistettujen kaavojen perusteella. Koneoppiminen on siten vahvasti kytköksissä tilastotieteisiin ja tiedonlouhintaan. (Murphy 2012, 1.) Toisin sanottuna koneoppimisessa tietokone kykenee oppimaan jonkin asian ilman, että sitä ohjelmoidaan eksplisiittisesti tekemään kyseistä asiaa (Géron 2019, 4). Kuvassa 1 on yksinkertaistettu prosessi koneoppimiseen perustuvan ohjelmiston julkaisemista (Géron 2019, 6).



KUVA 1. Prosessi koneoppimiseen perustuvan ohjelmiston julkaisemisesta (Géron 2019, 6). Tekstit suomennettu.

Kuten voidaan havaita, niin prosessi on iteratiivinen. Jos koneoppimismalli ei toimi riittävän tarkasti tai luokittelee väärin, niin ongelmaa voidaan tarkastella uudestaan ja määritellä data tai malli eri tavalla uuden tiedon valossa. Esimerkiksi roskapostisuodattimen ohjelmoinnissa perinteisellä tavalla kaikki mahdolliset roskapostiin viittaavat merkkijonot ja niiden yhdistelmät tulisi määritellä ohjelmakoodissa. Tällaisen järjestelmän ylläpito ja päivitys olisi erittäin työlästä. Sopivan koneoppimismallin avulla datasta voidaan löytää kaavat (merkkijonot, yhdistelmät yms.) automaattisesti, jonka myötä järjestelmän päivitys ja ylläpito vaatii vain koneoppimismallin uudelleenkouluttamista ajantasaisella datalla.

2.2.1 Koneoppimisen tyypit

Koneoppimisen metodit voidaan korkeimmalla tasolla jakaa kahteen päätyyppiin: **ohjattuun** ja **ohjaamattomaan** oppimiseen. Lisäksi on olemassa **osittain ohjattua** oppimista ja **vahvistusoppimista**, jotka ovat vähemmän käytettyjä metodeja. (Murphy 2012, 2; Géron 2019, 8.)

Ohjatussa oppimisessa järjestelmän tavoitteena on oppia millainen **tulos** (output) saadaan tietyillä **syötteillä** (input). Syötteitä kutsutaan myös **ominaisuuksiksi**. Oppimisen ohjattu osuus tapahtuu siinä, että järjestelmälle annetaan **harjoitusdataa** syöte-tulos -pareina, jotka ovat todellisia havaintoja maailmasta. Yksinkertaisena esimerkkinä havainnon ominaisuutena voisi olla henkilön pituus ja tuloksena henkilön paino. Havainnon ominaisuus on yleensä kuitenkin monimutkaisempi kokonaisuus, kuten sähköpostiviesti, kuva tai aikasarja. Tulos on yleensä joko luokittelutyyppinen (esim. mies tai nainen), jolloin puhutaan **luokittelusta**, tai numeerinen (esim. tulotaso), jolloin puhutaan **regressiosta**. (Murphy 2012, 2.)

Ohjaamattomassa oppimisessa järjestelmän tulee itse löytää datasta merkityksellisiä kaavoja. Harjoitusdatana tällaiselle järjestelmälle annetaan siis vain syötteitä ilman toivottuja tuloksia. Ohjatussa oppimisessa järjestelmän suorituskykyä voidaan arvioida harjoitusdatan tuloksen ja järjestelmän ennustaman tuloksen eroavaisuuden avulla, mutta ohjaamattomassa oppimisessa näin ei voida tehdä. (Murphy 2012, 2.) Ohjaamattomassa oppimisessa tyyppejä ovat mm. **ryhmittely** (clustering), jossa data jaetaan ryhmiin samankaltaisten ominaisuuksien perusteella, ja **ulottuvuuksien vähentäminen** (dimensional reduction), jossa moniulotteisesta ominaisuudesta tehdään yksinkertaisempi esimerkiksi yhdistämällä toistensa kanssa korreloivia ominaisuuden arvoja. (Géron 2019, 11-12.)

Koneoppimismalleja voidaan erotella myös niiden oppimistavan perusteella. Sopeutuvia malleja voidaan kouluttaa ”lennosta” ajamalla uutta dataa mallin läpi samalla kun järjestelmä on käytössä. Vastakohtana järjestelmä voi vaatia koko datan ajamista mallin läpi pelkästään uuden datan lisäksi. Tällöin uudelleenkouluttaminen vaatii selvästi enemmän aikaa ja resursseja.

2.2.2 Koneoppimisen haasteet

Koneoppimisen ideana on yleistää jokin ilmiö matemaattiseksi malliksi havaintojen perusteella (Bishop 2006, 2), mikä vaatii usein paljon harjoitusdataa. Tarvittavan datan määrä riippuu ongelmasta ja sen monimutkaisuudesta, mutta yksittäisten datapisteiden määrä harjoitusdatassa voi liikkua jopa miljoonissa (Géron

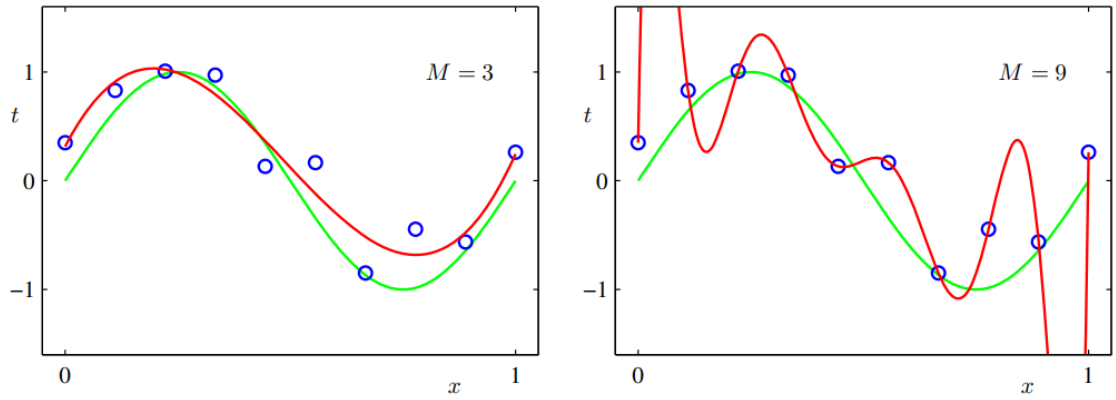
2019, 24). Yhtenä koneoppimisen haasteena onkin saada kerättyä tarpeeksi dataa luotettavan mallin kehittämiseksi.

Ilmiön yleistämisessä on tärkeää myös se, että harjoitusdata edustaa hyvin yleistettävää ilmiötä. Géron (2019, 26) kuvaa tämän olevan haastavampaa kuin miltä se kuulostaa. Esimerkkinä hän kirjoittaa erään kyselyn otantaharhasta Yhdysvaltain presidentinvaalien alla vuonna 1936 (Landon v. Roosevelt). Kyselyn vastaajia rekrytoitiin mm. puhelinluetteloista, aikakauslehtien tilaajista ja kerhojen jäsenistä. Kyselyyn vastaajia oli jopa 2,4 miljoonaa ja sen perusteella Landon saisi 57 % äänistä. Lopulta Landon hävisi Rooseveltille saaden vain 38 % äänistä. Yhtenä syynä kyselyn epäonnistumiseen oli se, että vastaajia rekrytoitiin lähteistä, jotka suosivat varakkaampaa kansaa. Vaikka vastaajia oli määrällisesti paljon, niin he eivät edustaneet sitä otantaa, joka lopulta äänestäisi vaaleissa.

Datan tulee olla myös laadukasta ja oleellista. Datan laatu on heikko, jos datapisteet eivät ole yhteneväisiä. Esimerkiksi, jos osasta dataa puuttuu täysin yksi ominaisuus, koska kyselyyn vastaajat eivät halunneet kertoa ikäänsä. Vaihtoehtoina laadun parantamiseksi on jättää ikä kokonaan pois ominaisuuksista tai karsia ne datapisteet pois, joissa ikä on tyhjä. Oleellista data on silloin, kun siinä on sopiva määrä hyödyllisiä ominaisuuksia. Oleellisuus kasvaa, kun mallista karsitaan pois ne ominaisuudet, jotka eivät suurella todennäköisyydellä kuvaa ilmiötä, tai kun yhdistetään keskenään korreloivia ominaisuuksia uusiksi ominaisuuksiksi. (Géron 2019, 27-28.)

Edellä mainitut haasteet liittyvät dataan, joka onkin todella tärkeä komponentti koneoppimismalleissa. Datan käsittely ja analysointi tuo mukanaan myös omia haasteita. Ensinnäkin oikeiden mallien, metodien ja parametrien valinta juuri käsillä olevaan ongelmaan vaatii syvällistä osaamista matematiikasta ja koneoppimisesta. Osaamisen ja ymmärryksen puute voi johtaa koneoppimisessa tunnettuun ongelmaan nimeltä **ylisovittaminen** (overfitting) (Géron 2019, 28-30). Kuvassa 2 on visualisoitu mitä ylisovittaminen tarkoittaa. Kuvasta on helppo ymmärtää, että vaikka ylisovittava malli sopisi täydellisesti harjoitusdataan, niin se saattaa tuottaa merkittävän virheellisiä arvoja uusilla datapisteillä. Géronin (2019, 29)

mukaan ylisovittamista voi vähentää valitsemalla yksinkertaisemman tai rajoitetumman mallin, vähentämällä harjoitusdatan ominaisuuksien määrää, keräämällä lisää harjoitusdataa ja parantamalla harjoitusdatan laatua.



KUVA 2. Punaiset käyrät edustavat vasemmalla kolmannen asteen yhtälöä ja oikealla yhdeksännen asteen yhtälöä. Vasemmalla yhtälö yleistää ilmiön melko hyvin, mutta oikealla yhtälö ylisovittaa dataan. (Bishop 2006, 7.)

Koneoppimisessa etsitään johonkin ongelmaan sopivinta mallia, mihin liittyen Wolpert (1996) esitti ns. ”No Free Lunch” -teoreeman. Sen mukaan ei voida sanoa varmasti minkään tietyn algoritmin toimivan paremmin kuin toinen. Käytännössä voidaan tehdä valistuneita olettamuksia datasta, minkä mukaan voidaan rajata testattavia malleja, koska kaikkien mallien testaaminen parhaan löytämiseksi on mahdotonta. (Murphy 2012, 24-25; Géron 2019, 33-34.)

2.2.3 Mallin valinta

Lähes aina tavoitteena on valita malli, joka suoriutuu parhaiten käsillä olevan ongelman ratkaisemisesta. Malleja täytyy jotenkin pystyä vertailemaan, jotta on mahdollista määritellä paras vaihtoehto. Looginen ratkaisu voisi olla mallin testaaminen siinä, kuinka tarkasti se ennustaa tai kuinka vähän virheitä se tekee harjoitusdatalla. Ylisovittamisen takia tämä voi johtaa usein yllättäviin tuloksiin, kun malli ei toimikaan yhtä hyvin sen ollessa tuotantokäytössä. Mallia tulisikin testata sellaisella datalla, jota ei ole käytetty mallin kouluttamiseen.

Tyypillistä on jakaa harjoitusdata kahteen osaan: **koulutusdataan** ja **testidataan**. Esimerkiksi niin, että 80 % harjoitusdatasta käytetään koulutusdatana

mallin kouluttamiseen ja loput sen testaamiseen testidatana. Jos harjoitusdataa on paljon, niin pienempikin suhteellinen osuus riittää testidataksi. Ehkä hieman yllättäen ongelmaksi voi kuitenkin muodostua ylisovittaminen myös testidataan. Kun mallia muokataan ja säädetään iteratiivisesti riittävän tarkaksi testidataa vasten, niin malli muovautuu sopivaksi juuri siihen yhteen datajoukkoon. (Bishop 2006, 32.)

Ratkaisuna tähän voidaan käyttää kolmatta datajoukkoa, joka on **validointidata**. Se erotetaan koulutusdatasta ja sitä käytetään alustavana datajoukkona testaamiseen ennen varsinaista testaamista testidatalla. Esimerkki: koulutetaan kolme erilaista mallia koulutusdatalla, josta on erotettu pois validointidata. Nämä kolme mallia testataan validointidatalla. Näistä valitaan parhaiten suoriutuva malli ja se koulutetaan uudestaan koko koulutusdatalla, johon on nyt lisätty siitä erotettu validointidata. Lopuksi malli testataan testidatalla, jotta voidaan mitata mallin kyky yleistää ilmiö. (Géron 2019, 32.)

Datan jakaminen kolmeen osaan saattaa aiheuttaa ongelmia, kun datan määrä joukoissa pienenee. Sattuman takia saatetaan valita huono malli, jos validointidata on pieni ja siihen valikoituu sattumalta joukko ”sopivia” datapisteitä. Tähän ratkaisuna on käytetty **ristiinvalidointia**, jossa koulutusdata jaetaan useampaan pieneen joukkoon. Jokainen joukoista on vuorotellen validointidatana ja muut koulutusdatana, jolloin lopullinen tulos on keskiarvo kaikista testeistä. Huonona puolena tämä vie moninkertaisesti aikaa ja resursseja, kun mallit koulutetaan uudestaan jokaista pienempää ristiinvalidointijoukkoa kohden. (Murphy 2012, 24.)

2.2.4 Mallien tarkkuuden mittaaminen

Varsinainen mallien tarkkuuden mittaamistapa riippuu mallin tyypistä. Yhden luokan luokittelumalleissa on helppo määrittää, oliko malli oikeassa vai väärässä, kun se luokittelee yhden datapisteen. Silloin koko datajoukosta voidaan laskea esimerkiksi suhteellinen osuus väärin luokitelluista datapisteistä. Luokittelumallien tarkkuuden ja suoriutumisen analysoinnissa käytetään usein sekaannusmatriisia ja ROC-käyrää (Géron 2019, 92-102). Regressiomalleissa ei voida suoraan määritellä ennustiko malli oikein vai väärin, koska ennusteet ja harjoitusdatan to-

siarvot ovat yleensä todellisia lukuja jatkuvalla mitta-asteikolla. Toisin kuin luokittelumalleissa, regressiomalleille on kuitenkin mahdollista määritellä numeerisia virhemuuttujia.

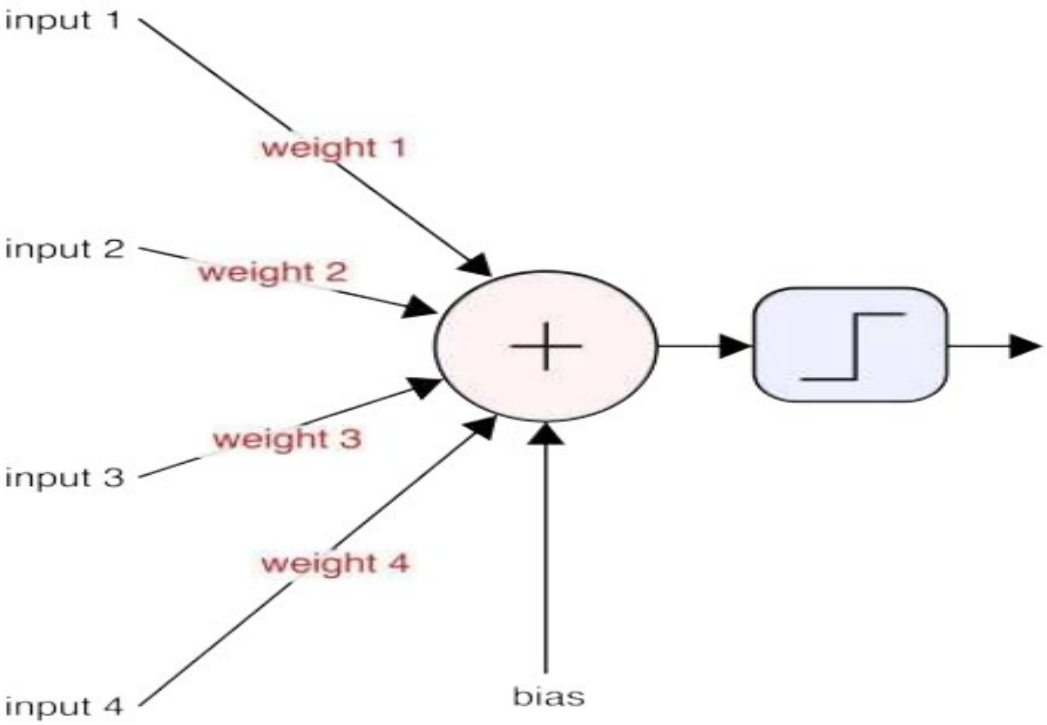
Regressiomalleissa sen suorituskykyä voidaan mitata laskemalla erilaisia virhemuuttujia havaintojen perusteella. Yksi yleisimmistä on keskineliövirheen neliöjuuri (RMSE). Ensin siinä lasketaan jokaisen datapisteen ennusteen ja todellisen arvon erotuksen neliö, sekä lasketaan niiden kaikkien summa. Tämä summa jaetaan havaintojen määrällä ja saadaan siis keskiarvo neliöidyistä erotuksista. Lopuksi tästä keskiarvosta otetaan neliöjuuri ja päädytään lopulliseen RMSE-arvoon. Käytännössä tämä arvo kertoo kuinka kaukana mallin ennuste, ja havainnon todellinen arvo keskimäärin ovat toisistaan. (Géron 2019, 42–44.)

3 SYVÄT NEUROVERKOT

Syväoppiminen on koneoppimisen erillinen osa-alue, jossa algoritmi voi oppia monimutkaisia käsitteitä ilman, että käyttäjä määrittelee tarkalleen mitä tehdä (Goodfellow ym. 2016, 1). Matemaattisten funktioiden ja asianmukaisten opetusnäytteiden avulla syväoppimisalgoritmit voivat oppia havaitsemaan datassa olevia kuvioita ja suorittamaan tiettyjä tehtäviä tarkasti (esim. kuvassa esiintyvien esineiden luokittelu tai käsin kirjoitettujen numeroiden tunnistaminen). Syväoppiminen on mahdollista ns. neuroverkkojen avulla. (Nielsen 2015.) Syvät neuroverkot ovat ns. ”end-to-end” -ratkaisuja, joiden avulla raa’asta pikselidatasta päädytään lopputulokseen ilman muuta prosessointia. Niiden kehittämisen aloittaminen sai inspiraationsa aivojen hermosolujen muodostamista yhteyksistä, mutta tutkijayhteisö on nykyään suurelta osin erkaantunut biologisesti mahdollisten mallien kehittämisestä. (Szelisky 2022, 268-269.)

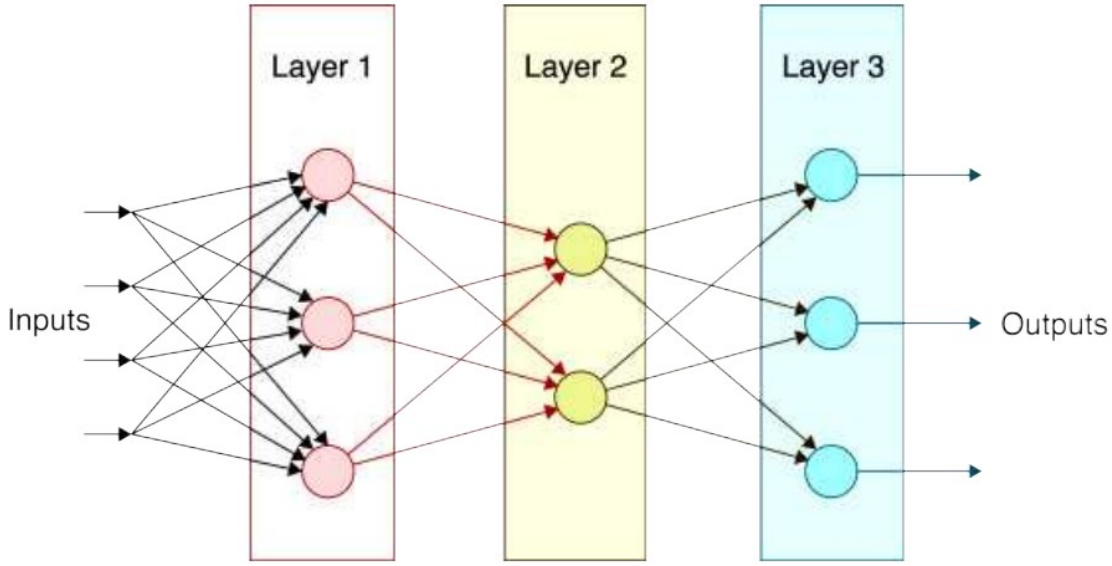
3.1 Rakenne ja toiminta

Neuroverkko koostuu keinotekoisista toiminnallisista yksiköistä, neuroneista, ja niiden välisistä yhteyksistä (kuva 3). Yleensä neuroni vastaanottaa useita numeerisia syötteitä, käsittelee ne ja tuottaa yhden numeerisen arvon. (Nielsen 2015.) Tarkemmin määriteltynä neuroneilla on jokaista syötettä kohden yksi painokerroin, mutta vain yksi vinouma-arvo. Neuroniin tullessaan syöte kerrotaan ensin sen omalla painokertoimella ja tämä arvo lisätään kasautuvaan summaan. Lopuksi summaan lisätään neuronin oma vinouma-arvo. Neuroni tuottaa ns. aktivaation, joka on yleensä myös syöte seuraavalle kerrokselle. Neuronin aktivointi lasketaan aktivointifunktiolla, joka saa parametrina aikaisemmin kuvaillun neuronin syötteiden painotetusta summasta ja vinouma-arvosta syntyneen summan. (Bishop 2006, 227). Tuotoskerroksen neuronien aktivaatiot muodostavat neuroverkon synnyttämän tuloksen. Aktivaatiofunktioita on erilaisia eri tarkoituksiin, mutta tyypillisiä neuroverkoissa käytettyjä funktioita ovat mm. ReLU ja logistinen funktio (Sharma ym. 2020).



KUVA 3. Yksittäinen neuroni, jolla on neljä syötettä (input) painoarvoineen (weight), yksi vinouma (bias) ja tulos (Szelisky 2022, 270).

Neuronit on järjestetty kerroksiin ja ns. eteenpäin kytketyissä neuroverkoissa yhden kerroksen arvoja käytetään syötteenä seuraavalle kerrokselle. Tasot on järjestetty hierarkkisesti verkossa (kuva 4). Neuroverkon ensimmäinen kerros on syötekerros ja käytännössä se on neuroverkolle syötetty data ulkopuolelta. Esimerkiksi kohteen tunnistuksessa syötedata voi olla kuvan yksittäisten pikselien RGB-arvoja. Verkon viimeinen kerros on tuotoskerros. Neuronien aktivaatio tuotoskerroksessa määrittää neuroverkon tuloksen tietyillä syötteillä. Neuronien aktivaatiot tuotoskerroksessa voisivat edustaa todennäköisyyttä sille, että kuva kuuluu tiettyyn luokkaan, kuten koira tai kissa, tai todennäköisyyttä sille, että pikselissä on tietty kohde (esim. silmä). Syöte- ja tuotoskerrosten välisiä kerroksia kutsutaan piilokerroksiksi. Piilokerrosten ja niissä olevien neuronien määrä määrittelee syötteen ja tuotoksen välisten suhteiden monimutkaisuuden. (Nielsen 2015.)



KUVA 4. Syvä neuroverkko, jossa on kolme kerrosta: syöte- (1), tuotos- (3), ja yksi piilokerros (2) (Szelisky 2022, 272).

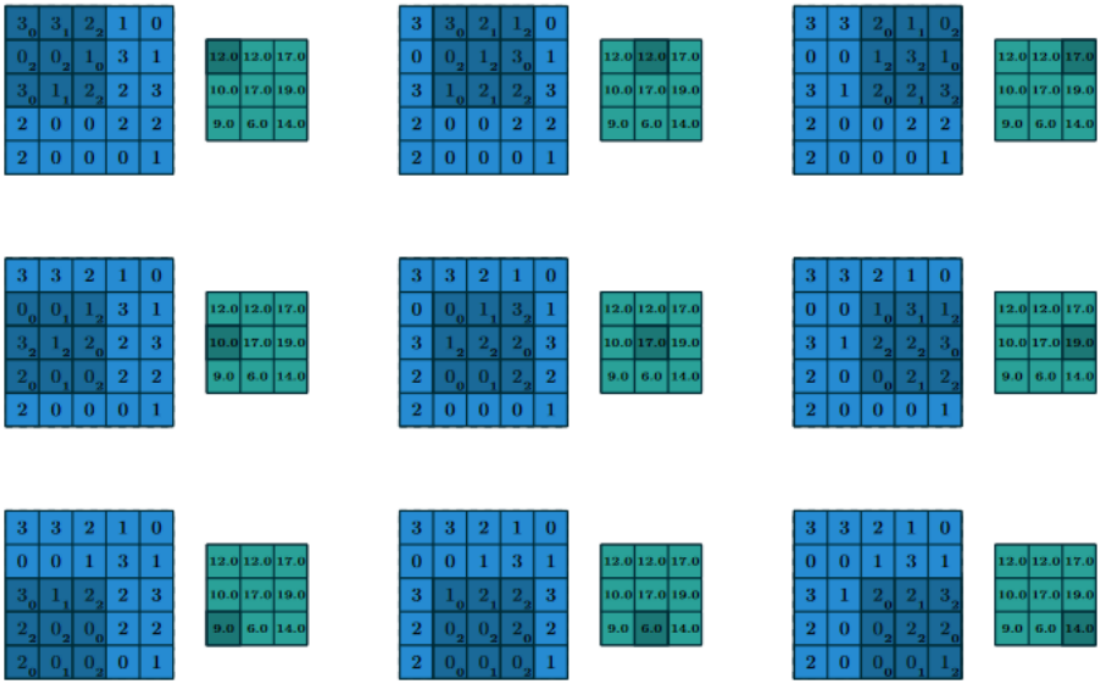
Koulutusdata määrittelee, mitä neuroverkon tulisi tuottaa tietyillä syötteillä, mutta ei sitä mitä piilokerrosten tulee tehdä. Oppimisalgoritmin on määriteltävä piilotettujen kerrosten käyttäytyminen oikean tulosteen tuottamiseksi. Neuroverkon kerrosten ja niiden neuronien voi kuvitella oppivan kuvasta tietynlaisia piirteitä, samoin kuin perinteisessä kohteen tunnistuksessa (Szelisky 2022, 308-311).

Neuroverkko voidaan kuvitella yhtenä isona funktiona, jossa on annetut syötteet, halutut tulokset, ja suuri määrä muokattavia parametreja eli painokertoimia ja vinoumia. Säätämällä näitä parametreja, neuroverkon syötteistä voidaan muokata erilaisia tuloksia. (Bishop 2006, 228.) Neuroverkot noudattavat ohjatun oppimisen periaatetta. Periaatteessa neuroverkko "oppii" siten, että annetaan sille ensiksi koulutusdataa, joiden mukana tulee oikeat vastaukset (esim. tässä kuvassa on koira). Sen jälkeen neuroverkolle annetaan uusia syötteitä ilman vastauksia ja annetaan palautetta sen tuottamista tuloksista. Neuroverkko muuttaa käyttäytymistään palautteen perusteella ja koulutusprosessi voidaan toistaa. Neuroverkko voi muuttaa rakenteessaan olevien osasten yhteyksiä ja muuttaa siten synnyttämiään tuloksia. Eri osien yhteyksiä muokataan algoritmeilla, joiden tarkoitus on yleensä minimoida jokin hukkafunktio. Neuroverkkoja opetetaan yleensä hyvin suurilla datamäärillä. (Goodfellow ym. 2016, 80, 164 – 165; Nielsen 2015.)

3.2 Konvoluutioneuroverkot

Konvoluutioneuroverkot ovat erityisiä neuroverkkoja, joiden jokainen kerros ei ole täysin kytköksissä sitä edelliseen ja seuraavaan kerrokseen. Aiemmin esitellyt eteenpäin kytketyt neuroverkot ovat täysin kytkettyjä, eli yksittäinen neuroni saa kaikilta edellisen kerroksen neuroneilta syötteen, ja sen aktivaatio annetaan syötteenä kaikille seuraavan kerroksen neuroneille. (Bishop 2006, 361; Szelisky 2022, 291.) Täysin kytketyissä neuroverkoissa tämä tarkoittaa sitä, että kaukana toisistaan olevat pikselit vaikuttavat toisiinsa yhtä paljon kuin vierekkäin olevat pikselit. Kuvadatassa tärkeä informaatio on yleensä keskittyneenä jollekin alueelle, joten on loogista, että kuvan pikseleitä prosessoidaan osissa niin kuin konvoluutioneuroverkoissa.

Konvoluutioneuroverkon kerrokset muodostuvat **piirrekartoista** (feature map), jotka syntyvät, kun edelliseltä kerrokselta tullut syötedata syötetään konvoluutiomaskien läpi. Piirrekartan voi ajatella kuvaavan kuvasta löytyneitä piirteitä (reunoja, kaaria yms.). Kuvassa 5 on havainnollistettu sitä, kuinka edellisen kerroksen yksi piirrekartta viedään yhden konvoluutiomaskin läpi ja siitä syntyy seuraavan kerroksen yksi piirrekartta. Konvoluutiomaski on käytännössä kaksiulotteinen matriisi, jonka alkiot ovat painokertoimia ja jolla on yksi vinouma-arvo. Konvoluutiomaskit ovat ikään kuin suodattimia, joilla kuvasta voidaan tunnistaa erilaisia piirteitä. Konvoluutioneuroverkon jokaista piirrekarttaa kohden on oma konvoluutiomaski. Konvoluutioneuroverkkoihin pätee pitkälti samat oppimisprosessit ja säännöt kuin aiemmin esiteltyihin eteenpäin kytkettyihin neuroverkkoihin. Niissä on usein myös muutamia täysin kytkettyjä kerroksia esim. viimeisinä kerroksina, joihin lukeutuu tuotoskerros. (Bishop 2006, 267–269.)



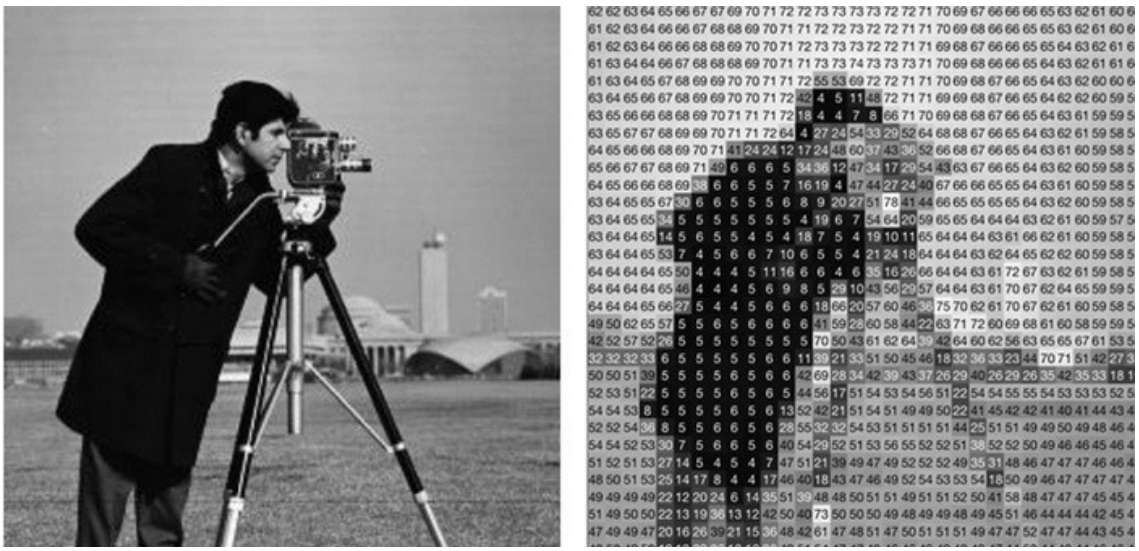
KUVA 5. Piirremaskin (vihreä) syntyminen, kun edellisen kerroksen piirrekartta tai raaka pikselidata (sininen) ajetaan konvoluutiomaskin (tumman sininen) läpi. Tämän konvoluutiomaskin painokertoimet näkyvät alaindeksissä ja vinouma-arvoksi on asetettu 0. (Dumoulin & Visin 2018.)

4 KONENÄKÖ, KOHTEEN TUNNISTUS JA SEURANTA

Konenäössä tietokone pyrkii kuvailemaan ympäröivää maailmaa sille syötetyn datan (kuvien) avulla, ja uudelleenrakentaa datan perusteella ympäröivän maailman ominaisuuksia, kuten muotoja, valaistusta ja värejä. Konenäön sovellutuksia on monia: optinen laadunvarmistus, itseajavat autot, 3D-mallinnus, kohteen tunnistus ja seuranta, varastojen ja tuotantojen automatisointi, ja kasvojen tunnistus. (Szeliski 2022, 5). Maailma on kolmiulotteinen ja kuvat kaksiulotteisia, joten yhteen kuvaan voidaan tallentaa vain hyvin rajattu versio totuudesta. Tästä syystä monet konenäön ongelmista ovat haastavia inversio-ongelmia, joissa 2D-kuvien perusteella yritetään palauttaa jokin osa tästä rajautuneesta totuudesta. Tähän käytetään fysiikkaan ja tilastotieteeseen perustuvia malleja, sekä koneoppimista ja suurien datamäärien avulla. (Szeliski 2022, 4).

Vaikka konenäkösovellutuksia alettiin kehittää jo 1970-luvulla, niin koneoppimista alettiin hyödyntää alalla vasta 2000-luvulla ja syviä neuroverkkoja 2010-luvulla. Tarjolle on tullut myös massiivisia ja korkealaatuisia datajoukkoja valmiiksi luokitelluista kuvista, jotka yhdessä koneoppimisen kanssa ovat vieneet konenäköalaa suurin harppauksin eteenpäin viimeisen kymmenen vuoden aikana. Myös näytönohjainten prosessointitehon kasvaminen, ilmaisten syväoppimisohjelmistokirjastojen kehitys ja vapaan lähdekoodin neuroverkkomallit ovat mahdollistaneet konenäköön liittyvän tutkimuksen tekemisen ja sovellutusten kehittämisen pienemmillekin toimijoille. (Szelisky 2022, 11-22.)

Näkeminen ja sen prosessointi on niin helppoa ja automatisoitua ihmiselle, että konenäön haastavuuden ymmärtämiseksi olisi hyvä kerrata miten tietokone prosessoi kuvia. Tietokone ”näkee” vain numeroita ja esimerkiksi harmaasävykuva on käytännössä vain matriisi numeroita, jotka kuvaavat kuvan pikselien intensiteettejä. Kuvassa 6 on havainnollistettu miten pikselimatriisi vastaa harmaasävykuvaa.



KUVA 6. Harmaasävykuva, jonka päälle on liitetty matriisi kuvassa olevien pikselien intensiteettien arvoista (Agammed 2020).

4.1 Kohteen tunnistus

Kohteen havaitseminen ja **tunnistus** ovat konenäköjärjestelmille eräitä oleellisimpia kykyjä, jotta tiedetään mitä kohdetta tarkastellaan, missä se on ja millaisia ominaisuuksia sillä on. Jos järjestelmä löytää kuvasta esimerkiksi auton ja sen sijainnin, niin puhutaan vain kohteen havaitsemisesta. Kun järjestelmä pystyy päättelemään myös auton ominaisuuksia, kuten värin, merkin tai tyypin, niin puhutaan kohteen tunnistuksesta. (Gollapudi 2019, 97-98.)

Ennen syvien neuroverkkojen käytön yleistymistä kohteen havaitsemisessa ja tunnistuksessa käytettiin mm. **piirteiden havaitsemista** (feature detection) ja **sovittamista** (matching). Syvät neuroverkot ovat kuitenkin yleensä paljon piirreperusteista lähestymistä tehokkaampia kohteen tunnistuksessa ja piirteiden käyttö on vähentynyt. (Szeliski 2022, 345). Kuvan piirteiden havaitsemisessa pyritään löytämään kohteesta piirteitä (reunoja, kulmia, pisteitä yms.), jotka kuvaisivat sitä parhaiten. Piirteet erottuvat yleensä parhaiten, jos niillä on suuri kontrasti taustaan. (Szeliski 2022, 422). Sovittamisessa kuvasta erotettuja piirteitä verrataan tietokantaan, jossa on muista kuvista erotettuja piirteitä ja etsitään niistä paras vastaavuus (Gollapudi 2019, 104-105). Szeliskin (2022, 21) mukaan nykyisin syvät konvoluutioneuroverkot ovat usein ainoita tekniikoita, joita tunnistusongelmiin harkitaan käytettävän niiden tehokkuuden vuoksi.

4.2 Kohteen seuranta

Kohteen seuranta on käytännössä saman kohteen havaitsemista peräkkäisissä videon kuvissa. Kohteen seurantaongelmia on erilaisia, kuten yhden kohteen seuranta, monen kohteen samanaikainen seuranta, 3D seuranta ja videon kohteiden jakaminen osiin (Zhang ym. 2021). Kohteen seurantaa sovelletaan mm. liikenteen ja autojen seurannassa, teollisuuden robotiikassa, videopeleissä, lääketieteen diagnostiikkajärjestelmissä ja turvallisuusjärjestelmissä (Ali ym. 2016).

Kohteen seurannalle on useita eri lähestymistapoja, jotka kuitenkin aina sisältävät kohteen havaitsemisen vähintään kerran. Kohde voidaan periaatteessa havaita kerran videon ensimmäisessä kuvassa ja käyttää erilaisia algoritmeja sen seuraamiseen. Nämä algoritmit ovat myös tavallaan kohteen havaitsemista, mutta ne vaativat kuitenkin esitietona kohteen nykyisen sijainnin. Kohde voidaan myös havaita joka kuvassa uudestaan jollakin perinteisellä kohteen havaitsemismetodilla. Haasteet seurannassa riippuvat käytetyistä lähestymistavoista ja ongelmasta, mutta niitä ovat mm. kohteen peittyminen toisen taakse, suuri liikkumisnopeus, sumentuminen, muodon ja värityksen muuttuminen, sekä virheelliset havainnot, kun kuvassa on useita samanlaisia kohteita. (Gollapudi 2019, 119.) Seuraavassa on käyty läpi muutamia kohteen seuraamisessa käytettyjä algoritmeja ja metodeja.

Yksi klassisista kohteen seuranta -algoritmeista on **Mean Shift (MS)** -algoritmi. MS-algoritmissa kohde muunnetaan histogrammiksi esim. pikselien värien intensiteeteistä. Kohdetta etsitään seuraavasta kuvasta etsintäalueelta, joka sijoitetaan edellisestä kuvasta paikannetun kohteen ympärille. Histogrammia vastaava alue paikannetaan algoritmissa liikkumalla iteratiivisesti kohti aluetta, jossa histogrammin vastaavuus on vahvempaa. Uudesta sijainnista luodaan uusi histogrammi, jota etsitään taas seuraavasta kuvasta. (Zhang ym. 2021.) MS-algoritmi vaatii toimiakseen siis ennakkotietoa kohteen sijainnista, koska algoritmille tulee määrittää sen aloituspiste. Lisäksi jos seurattava kohde liikkuu nopeasti ja se liikkuu etsintäalueen ulkopuolelle, niin MS-algoritmi suppenee väärään pisteeseen

ja algoritmi hukkaa kohteen. (Shivhare & Choudhary 2015.) Algoritmi ei toimi hyvin myöskään silloin, kun kohde peittyy edes osittain toisen kohteen taakse (Ali ym. 2016).

Kalman-suodin on ennusteperusteinen kohteen seurantametodi. Siinä kohteen liikkumisesta luodaan dynaaminen malli edeltävien kuvien perusteella ja mallilla voidaan ennustaa kohteen sijainti seuraavassa kuvassa. Kohteen tilaan ja sen muutokseen liittyy kuitenkin kohinaa, jota ei voida ennustaa mallilla. Tämä otetaan huomioon Kalman-suotimessa korjaamalla ennustetta todellisen havainnon perusteella, joka voi tulla esim. kohteen havaitsija-algoritmilta. (Soleimanitaleb & Keyvanrad 2022.) Kalman-suodinta käytetään laajasti yhdessä muiden seuranta-algoritmien kanssa. Kalman-suotimen ennustetta voidaan käyttää apuna, kun määritellään seuraavan kuvan etsintäalueen sijaintia ja laajuutta. Lisäksi Kalman-suodin toimii yhtenä ratkaisuna kohteen peittymisongelmalle. Seurantaa voidaan jatkaa jonkin aikaa pelkkien suotimen ennusteiden perusteella. (Ali ym. 2016.)

Seuranta havaitsemisen kautta -metodit ovat joukko algoritmeja, jotka havaitsevat kohteen jokaisessa kuvassa uudestaan. Prosessina tämä metodi on yksinkertaisempi kuin monta erillistä metodia yhdistävä algoritmi kohteen seuraamiseen. Nykyään havaitsemiseen käytettävät algoritmit ovat tehokkaita ja niiden kouluttamiseen on paljon dataa ja prosessointitehoa, joten kohteen uudelleenhavaitseminen on edullista. (Ali ym. 2016.) Neuroverkot voivat olla todella tarkkoja kohteen seurannassa havaitsemisen kautta, mutta niiden nopeus saattaa olla heikko, jos neuroverkkojen arkkitehtuuri ja koko kasvavat suuriksi (Zhang ym. 2021). Martinez-Alpiste ym. (2022) arvioivat tutkimuksessaan kohteen tunnistukseen tarjolla olevia koneoppimisalustoja ja -kirjastoja, jotka mahdollistavat syvien konvoluutioneuroverkkojen hyödyntämisen laitteissa, joissa on rajattu prosessointiteho (esim. älypuhelimet). Tällaisia alustoja ovat mm. TensorFlow Mobile, TensorFlow Lite, OpenCV ja Snapdragon Neural Processing Engine SDK, joita voi käyttää Android-käyttöjärjestelmän laitteissa.

4.3 Kevyet menetelmät resurssirajoitteisissa laitteissa

Tyypillisesti raskaita koneoppimismalleja ajetaan tehokkailla palvelimilla, jolloin mallin arkkitehtuurista ei tarvitse tehdä erityisen kevyttä. On yhä suurempi tarve

ajaa koneoppimismalleja, kuten konvoluutioneuroverkkoja, resurssirajoitteisissa laitteissa, kuten älypuhelimissa ja sulautetuissa järjestelmissä. Laitteet eivät välttämättä kykene olemaan yhteydessä palvelimiin välitöntä mallin ajamista varten ja tulokset saatetaan tarvita heti. (Martinez-Alpiste ym. 2022.)

4.3.1 Yhden vaiheen konvoluutioneuroverkot

Kohteen tunnistukseen käytetyt konvoluutioneuroverkot tekevät tunnistuksen yleensä yhdessä tai kahdessa vaiheessa. Kahden vaiheen neuroverkot etsivät ensimmäisessä vaiheessa kuvasta kohteet ja toisessa vaiheessa tunnistaa mitä kohteet ovat. Yhden vaiheen neuroverkot tekevät molemmat tehtävät yhdessä vaiheessa ja ovatkin siis paljon nopeampia ja parhaimpia vaihtoehtoja reaaliaikaisuuden saavuttamiseen. (Szeliski 2022, 382–385.) Reaaliaikaisuus vaatii ruudunpäivitysnopeudeksi (**FPS**) vähintään noin 20 – 25 ruutua sekunnissa, jolloin päättelynopeudeksi tulee 40 – 50 ms.

Tällä hetkellä parhaimpia yhden vaiheen konvoluutioneuroverkkoja reaaliaikaiseen kohteen tunnistukseen ovat **YOLO**-perheen (You Only Look Once) mallit (Papers With Code n.d.). Ensimmäinen YOLO-malli esiteltiin Redmonin ym. (2016) toimesta ja se oli arkkitehtuurinsa ansiosta merkittävästi nopeampi ja tarkempi kuin aiemmat yhden vaiheen mallit (Zaidi ym. 2021). Ensimmäisessä YOLO-mallissa kuva jaetaan ruudukoksi ja kohde havaitaan siinä ruudussa, jossa sen keskipiste sijaitsee. Jokaisessa mallin ennusteessa on kohteen keskipisteen koordinaatit, kohteen ympäröivän suorakaiteen leveys ja korkeus, sekä ennusteen varmuusarvo. (Redmon ym. 2016.) Ensimmäisen YOLO-mallin jälkeen julkaistu **SSD**-malli (Liu & Angelov 2016) oli ensimmäinen yhden vaiheen konvoluutioneuroverkko, joka oli yhtä tarkka kuin kahden vaiheen neuroverkot, mutta pystyi silti reaaliaikaiseen päättelyyn (Zaidi ym. 2021).

YOLO-malleja on paranneltu ensimmäisestä versiosta ja niitä oli Zaidin ym. (2021) katsauksen aikana YOLOv2, YOLO9000, YOLOv3 ja YOLOv4, joista viimeisin YOLOv4 (Bochkovskiy ym. 2020) saavutti ”state-of-the-art” -statuksen reaaliaikaisissa yhden vaiheen kohteen tunnistajissa. Myöhemmin YOLOv4 pohjalta kehitettiin Scaled-YOLOv4 (Wang ym. 2021a), joka oli tarkempi ja silti yhtä nopea kuin YOLOv4. Lisäksi viimeisimpinä lisäyksinä YOLO-perheeseen ovat

tulleet YOLOR (Wang ym. 2021b) ja PP-YOLOE (Xu ym. 2022). YOLOR on jopa 88 % nopeampi kuin Scaled-YOLOv4 ja saavuttaa silti saman tason tarkkuuden (Wang ym. 2021a). YOLOR on tällä hetkellä ”state-of-the-art” reaaliaikaisessa kohteen tunnistuksessa (Papers With Code n.d.). EfficientDet-mallit (Tan ym. 2020) olivat ennen YOLOv4-mallia parhaita, mutta sen jälkeen YOLO-mallit ovat pitäneet kärkipaikkaa reaaliaikaisessa kohteentunnistuksessa.

Mobiili- ja IOT-laitteet tuovat erityisen rajoitteen laitteiston suorituskykyyn niiden pienen koon takia. Jotta tällaisilla laitteilla voitaisiin suorittaa reaaliaikaista kohteen tunnistusta, on käytettyjen neuroverkkojen oltava erityisen kevyitä tai niiden prosessoinnissa on käytettävä erityisen taloudellisia metodeja prosessointiresurssien käytön suhteen. **MobileNet**-perheen mallit on suunniteltu resurssirajoitteisille laitteille ja niistä uusimmat perustuvat SSD-malliin. (Zaidi ym. 2021.)

4.3.2 Deep SORT

Reaaliaikainen kohteen seuranta on tärkeää järjestelmissä, jotka vaativat nopeaa reagointia havaintoihin tai joissa pitkä viive haittaa sen toimintaa tai käyttämistä. **SORT** (Bewley ym. 2016) ja **Deep SORT** (Wojke ym. 2017) ovat samaan ideaan perustuvia yksinkertaisia, mutta nopeita algoritmeja usean kohteen seurantaan. Ne perustuvat seurantaan havaitsemisen kautta ja hyödyntävät Kalman-suodinta sekä ns. Unkarilaista algoritmia päätellessään mikä kohde nykyinen havainto on. Alkuperäinen SORT kehitettiin tietoisesti olemaan nopea, mutta ei käsittelemään ongelmatilanteista. Se kärsii erityisesti kohteen peittymisen aiheuttamasta kohteen virheellisestä uudelleentunnistuksesta (Bewley ym. 2016). Wojke ym. (2017) kehittivät alkuperäistä SORT-algoritmia, jotta kohteen peittyminen ei aiheuttaisi kohteen identifioimista eri kohteeksi, kun se peittyy edes yhden kuvan ajaksi videossa. Deep SORT nimi tulee kohteen uudelleentunnistuksessa käytetystä syvästä konvoluutioneuroverkosta, jonka avulla virheellisiä uudelleentunnistuksia tapahtuu vähemmän. Kohteen tunnistukseen näissä algoritmeissa voidaan käyttää mitä tahansa syvää neuroverkkoa, joten algoritmia voidaan soveltaa mm. resurssirajoitteisissa laitteissa kevyemmillä neuroverkoilla, ja se ei ”vanhene” teknologisesti, kun neuroverkkojen arkkitehtuurit kehittyvät (Bewley ym. 2016).

Parico & Ahamed (2021) osoittivat, että Deep SORT yhdessä YOLOv4-mallien avulla kykenee reaaliaikaiseen päärynöiden seuraamiseen ennalta kuvatussa videossa. Päättelynopeuden testaaminen suoritettiin pöytäkoneella, jossa oli näytönohjaimena NVIDIA GeForce GTX 1060. Keveimmällä YOLOv4-tiny -mallilla FPS oli yli 50 ja jonkin verran tarkemmalla YOLOv4-mallilla FPS oli vähintään 24. Parico & Ahamed (2021) suosittelivat YOLOv4-tiny -mallin käyttämistä, jos on tarpeen suorittaa päättely mobiililaitteessa. He mainitsevat myös, että kyseinen kevyinkin malli tarvitsee vähintään 2 GB näytönohjaimen muistia.

5 TEKOÄLYMALLI LEVYTANGON PÄÄN SEURAAMISEEN

Levytangon pään seuraamiseksi koulutettiin syvä neuroverkko TensorFlow-kirjaston ja itse kuvatun datan avulla. Syvän neuroverkon koulutukseen käytettiin Google Colab -palvelua, joka tarjoaa ilmaiseksi suoritusympäristön Python-koodille internetin välityksellä (Google n.d.).

5.1 TensorFlow

TensorFlow on ilmainen ja vapaan lähdekoodin ohjelmistokirjasto, jota käytetään laajasti koneoppimisessa ja tekoälysovelluksissa. Se on erityisen suosittu syvien neuroverkkojen koulutuksessa ja käytössä. TensorFlow'n ensimmäinen versio julkaistiin vuonna 2015 Googlen kehitystiimin toimesta (Abadi ym. 2015). TensorFlow 2.0 julkaistiin neljä vuotta myöhemmin vuonna 2019 (Medium 2019). TensorFlow'ta voi käyttää useammalla kielellä, mutta se tarjoaa parhaimman tuen Python-kielelle, joka on yksi suosituimmista ohjelmointikielistä datatieteiden parissa (TensorFlow documentation n.d.).

Vuonna 2018 TensorFlow'sta julkaistiin myös JavaScriptia tukeva versio TensorFlow.js (Gordon & Robinson 2018). TensorFlow.js mahdollistaa TensorFlow'n rajapinnan hyödyntämisen myös JavaScript-sovelluksissa eli mm. selaimessa ja Node.js ajoalustalla. JavaScript tuen avulla koneoppimista ja syviä neuroverkkoja voidaan siis hyödyntää suoraan esimerkiksi käyttäjän selaimessa. Vuonna 2020 TensorFlow.js sai virallisen tuen ja adapterin myös React Native -mobiilisovelluskehykselle (TensorFlow blog 2020). TensorFlow'ssa on valmiiksi toteutettuja ja esikoulutettuja neuroverkkomalleja kohteen tunnistusta varten (GitHub n.d. a). Monet näistä malleista on esikoulutettu COCO-datajoukolla (Lin ym. 2015) tunnistamaan 80 erilaista kohteita.

5.2 Neuroverkkomalli

Neuroverkoksi sovellukseen valittiin **MobileNetV2** (Sandler ym. 2019). Kyseinen neuroverkko on toteutettu TensorFlow'lla ja siitä löytyy COCO-datajoukolla esikoulutettu versio "ssd_mobilenet_v2_coco" (GitHub n.d. a). MobileNetV2 valikoitui siksi, että se oli TensorFlow'n suorittamien testien mukaan (GitHub n.d. a) yksi

nopeimmista neuroverkoista säilyttäen kuitenkin kohtuullisen tarkkuuden COCO-datajoukolla testatessa.

Tämä TensorFlow'lla toteutettu versio MobileNetV2-mallista ottaa vastaan syötteenä 300x300 resoluution värikuvia. Koska kyseessä on värikuva, niin sillä on kolme kanavaa (punainen, sininen ja vihreä) ja syötteessä on siis yhteensä 3x300x300 lukua. Malli päättelee syötteestä mitä kohteita siinä on, missä ne sijaitsevat syötteen kuvassa, ja kuinka varma malli on siitä, että tietyssä sijainnissa on tietty kohde. Malli tuottaa käytännössä päättelyn tuloksena erikokoisia tensoreita, jotka sisältävät jokaisesta ennusteesta kohteen luokan numeron kokonaislukuna, varmuusarvon desimaalilukuna (0-1), sekä sen sijainnin. Sijainti ilmoitetaan kohdetta ympäröivän suorakulmion vasemman yläkulman ja oikean alakulman koordinaatteina desimaalilukuina.

5.3 Harjoitusdata

Syvän neuroverkon koulutukseen tarvittava data kuvattiin OnePlus 6 -älypuhelimella paikallisella kuntosalilla Tampereella. Tarkemmin harjoitusdataksi kuvattiin videoita levytangolla tehtävistä voimaharjoitteluliikkeistä (esim. maastaveto ja kyykky). Videot kuvattiin noin 3 metrin päästä sivusta, jolloin levytangon pää on kohtisuorassa kameraan. Kuvattujen videoiden resoluutio oli 1080x1920. Videoita kuvattiin seitsemän, joista erotettiin kuvakaappauksia VLC-mediasoittimella (VideoLAN n.d.). Kuvakaappausten erottaminen toteutettiin automaattisesti siten, että joka 250. kuvaruutu tallennettiin levylle. Kuvakaappaukset rajattiin tämän jälkeen resoluutioon 600x600 siten, että rajattu alue otettiin täsmälleen kuvakaappauksen keskeltä. Rajaus toteutettiin automaattisesti Python-ohjelmalla. Rajauksen jälkeen kuvakaappauksista valittiin harjoitusdataan ne, joissa levytangon pää oli kokonaan näkyvissä. Rajattuja kuvakaappauksia jäi lopulta yhteensä 79 kappaletta harjoitusdataksi.

Harjoitusdatan merkkaaminen (**labeling** tai **annotating**) suoritettiin vapaan lähdekoodin LabelImg-ohjelmistolla (Lin n.d.). Levytangon pää merkattiin videoista erotettuihin kuviin ympäröimällä se suorakaiteen muotoisella alueella (kuva 7). Päättelyyn käytetty MobileNetV2-malli tuottaa syötteestä ennusteita, joista jokai-

seen liittyy vastaavanlainen suorakaide. Ohjelmisto tallensi merkinnät XML-muodossa siten, että jokaisen kuvan suorakaiteesta tallennettiin vasemman yläkulman ja oikean alakulman koordinaatit yhteen tiedostoon.

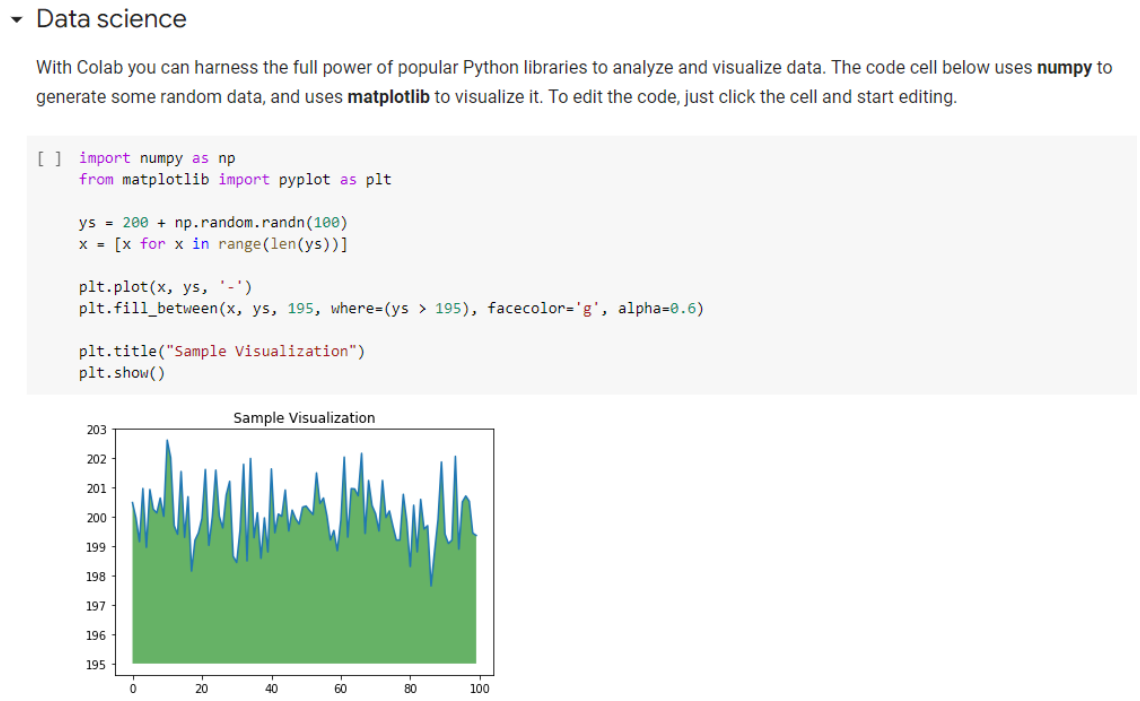
KUVA 7. Kuvakaappaus LabelImg-ohjelmistosta yhdestä harjoitusdatana käytetystä kuvasta, johon on merkattu levytangon pään sisältävä suorakulmio.

Harjoitusdata jaettiin koulutus- ja testidataan siten, että 59 kuvaa toimi koulutusdatana ja 20 kuvaa testidatana. Suhteellisesti koulutusdataa oli siis noin 74,7 % ja testidataa 25,3 % harjoitusdatasta. Neuroverkon koulutukseen käytetty Python-ohjelma vaati datan tietyssä muodossa, joten data vaati vielä muuntamista. Kun kaikki kuvat oli merkattu, niin koulutus- ja testidatajoukkojen XML-tiedostoista luotiin Python-ohjelmalla yhdet CSV-tiedostot, jotka sisälsivät datajoukon

kaikkien kuvien suorakulmioiden koordinaatit. Lisäksi kuvadata ja merkinnät yhdistettiin Python-ohjelmalla TFRecord-muotoon. Data on tallennettu TFRecord-tiedostoon binäärisarjoina, joten se vie vähemmän tilaa, data voidaan lukea nopeammin ja kaikki data sijaitsee samassa tiedostossa.

5.4 Mallin koulutus

Neuroverkkomallin koulutus suoritettiin Google Colab -palvelussa, jossa koulutukseen pystyttiin käyttämään suorituskykyistä näytönohjainta ilmaiseksi. Colab-palvelu toimii ns. notebookien avulla, jotka hostataan Googlen palvelimilla, joihin on asennettu Python-kehitysympäristö. Colabissa olevat notebookit ovat käytännössä samoja kuin Jupyter notebookit (Jupyter n.d.). Notebookeissa voi ajaa koodia solu kerrallaan ja niihin voi upottaa tulosteita sekä lisätä esim. tekstiosuuksia (kuva 8). Colabin kehitysympäristö sisältää valmiiksi kaiken tarvittavan perinteiseen datatieteilyyn, kuten TensorFlow'n, joten koodin testaaminen ja ajaminen on nopeaa. Palvelun tarjoama palvelin sisältää suorituskykyisen näytönohjaimen, joka mahdollistaa syvien neuroverkkojen nopean koulutuksen ilman omaa näytönohjainta.



KUVA 8. Esimerkki notebookin osasta, jossa on tekstiosio, koodisolu ja solun ajamisesta syntynyt upotettu tuloste (Google n.d.).

Valittu neuroverkkomalli oli TensorFlow 1 -version kanssa yhteensopiva, joten Colabissa käytettiin TensorFlow'sta versiota 1.15.2. Colabin palvelimella oli NVIDIA Tesla T4 16GB näytönohjain, joka on erityisesti kehitetty mm. syvien neuroverkkojen koulutusta varten.

TensorFlow'lla on erillinen ohjelmakirjasto "Model Garden" (GitHub n.d. c), joka mahdollistaa mm. omien mallien yksinkertaisemman käyttöönoton ja olemassa olevien mallien uudellenkoulutuksen. Model Gardenia hyödynnettiin neuroverkkomallin koulutuksessa Colab-ympäristön lisäksi. Kyseinen ohjelmistokirjasto ladattiin ensin omaan Drive-palveluun, jonka jälkeen kirjasto kiinnitettiin ja asennettiin Colab-palvelun kautta (kuva 9).

KUVA 9. TensorFlow'n Model Gardenin kiinnitys Google Drivestä Colabiin ja sen asennus.

Varsinaista mallin koulutusta varten Model Gardenin ohjelmakirjastoon tuotiin aiemmin luotu harjoitusdata CSV- ja TFRecord-muodossa (kohta 5.3), sekä alkuperäiset kuvat. Lisäksi luotiin tekstitiedosto, jossa määriteltiin kaikki luokat, joihin malli voi luokitella kohteita. Käytännössä luokkia oli vain yksi, joka vastaa levytangon päätä. Kuvassa 10 on kuvattu koodisolu Colabin notebookista, jolla käynnistettiin mallin koulutus ja osoitettiin tiedosto (ssd_mobilenet_v2_coco.config), jossa määritellään mm. neuroverkon ja koulutuksen parametreja sekä harjoitusdatan sijainti.

```
[ ]  !pip install tf_slim
     %cd /content/gdrive/My Drive/object_detection/models/research/object_detection
     os.environ['PYTHONPATH'] += ':/content/gdrive/My Drive/object_detection/models/research/:/content/gdrive/My Drive/Tensorflow/models/research/slim'

     !python train.py --train_dir=training/ --pipeline_config_path=training/ssd_mobilenet_v2_coco.config --logtostderr
```

KUVA 10. Mallin koulutuksen käynnistäminen.

Koulutusta suoritettiin noin 30 minuuttia. Lyhyt koulutusaika ja pieni harjoitusdatan määrä (79 kuvaa) on mahdollista, koska malli oli esikoulutettu COCO-datajoukolla. Malli koulutettiin uudelleen tunnistamaan vain yhdenlaisia kohteita, joten koulutuksen jälkeen se ei kykene enää tunnistamaan COCO-datajoukon luokkia. Tätä kutsutaan siirto-oppimiseksi (transfer learning). Tämän jälkeen malli tallennettiin muotoon, jolla voidaan suorittaa päättelyä Python-ympäristössä.

6 MOBIILISOVELLUS LEVYTANGON PÄÄN SEURAAMISEEN

Mobiilisovellus toteutettiin JavaScript-pohjaisella React Native sovelluskehyksellä. Koulutettu neuroverkkomalli muunnettiin TensorFlow.js-muotoon ja paketoitiin ohjelmiston lähdekoodiin.

6.1 TensorFlow.js yhteensopiva malli

Neuroverkkomallin koulutuksen tuloksena syntynyt malli (kohta 5.4) oli yhteensopiva Python-ympäristössä ajettavaksi, joten malli täytyi muuntaa JavaScript-ympäristössä ajettavaan muotoon. Kun malli halutaan paketoida lähdekoodin sisään, niin TensorFlow.js-kirjastossa tällaiset mukautetut mallit vaativat kaksi tiedostoa. Toinen on **json**-päätteinen tiedosto, joka sisältää mm. mallin rakenteen ja syötteen prosessointiin liittyviä määrityksiä. Toinen on **bin**-päätteinen tiedosto, joka sisältää mallin painokertoimet ja vinoumat.

TensorFlow on luonut muunto-ohjelman (GitHub n.d. a), jolla Python-yhteensopivista malleista voidaan luoda ns. TensorFlow.js malleja, joita voidaan ajaa JavaScript-ympäristössä. Käyttämällä kyseistä ohjelmaa aiemmin luodusta mallista luotiin TensorFlow.js malli, eli yhdet json- ja bin-tiedostot.

6.2 React Native -sovellus

React Native -mobiilisovellus luotiin Expo-alustalla ja sen tarkoituksena oli olla prototyyppi, joka päättelee videokuvasta levytangon pään sijainnin. Sovellukseen asennettiin tarvittavat riippuvuudet, jotka löytyvät kuvasta 11. Oleellisimpina näistä ovat TensorFlow.js riippuvuudet: *@tensorflow/tfjs* ja *@tensorflow/tfjs-react-native*. TensorFlow.js malli tallennettiin sovelluksen lähdekoodiin assets/model -kansioon.

```
"dependencies": {
  "@react-native-async-storage/async-storage": "~1.17.3",
  "@tensorflow/tfjs": "^3.18.0",
  "@tensorflow/tfjs-react-native": "^0.8.0",
  "@types/react-native-canvas": "^0.1.8",
  "expo": "~45.0.0",
  "expo-camera": "~12.2.0",
  "expo-gl": "~11.3.0",
  "expo-gl-cpp": "~11.3.0",
  "expo-status-bar": "~1.3.0",
  "expo-updates": "~0.13.1",
  "react": "17.0.2",
  "react-dom": "17.0.2",
  "react-native": "0.68.2",
  "react-native-canvas": "^0.1.38",
  "react-native-fs": "^2.20.0",
  "react-native-web": "0.17.7",
  "react-native-webview": "^11.18.2"
},
"devDependencies": {
  "@babel/core": "^7.12.9",
  "@types/react": "~17.0.21",
  "@types/react-native": "~0.66.13",
  "typescript": "~4.3.5"
},
```

KUVA 11. React Native -mobiilisovelluksen asennetut riippuvuudet.

Kun sovellus oli käynnistynyt ja TensorFlow.js -riippuvuudet latautuneet, neuroverkkomallin osat ladattiin lähdekoodista, ne paketoitiin käytettävään muotoon ja malli asetettiin sovelluksen tilamuuttujaan käytettäväksi (kuva 12).

```
if (isTfReady) {
  const modelJson = require('./assets/model/ssd_mobilenet_v2_model.json')
  const modelWeight = require('./assets/model/ssd_mobilenet_v2_shard.bin')
  const model = await tf.loadGraphModel(
    bundleResourceIO(modelJson, modelWeight)
  )
  console.log('Setting the model')
  setModel(model)
}
```

KUVA 12. Mallin osien lataaminen lähdekoodista ja käytettävän mallin luominen niistä.

Malli vaatii syötteen tensorina, jonka muoto on (-1, 300, 300, 3), joten kamerasta tulevan videon data tuli muuntaa sopivaan muotoon. Ensimmäinen luku -1 tarkoittaa, että ensimmäisen ulottuvuuden koko voi olla mikä tahansa. Käytännössä tämä luku kuvaa sitä, kuinka monen kuvan data syötetään kerralla. Tämän sovelluksen kohdalla se oli 1.

TensorFlow.js -kirjaston avulla voidaan luoda komponentti, johon tyypillinen kamerakomponentti voidaan paketoida ja jolla kamerasta tuleva syöte voidaan muuntaa suoraan tensorimuotoon (kuva 13). Komponentista saatavalla tensorilla on kolme ulottuvuutta, jotka voidaan määritellä resize-alkuisilla parametreilla.

```
<TensorCamera
  style={[
    styles.camera,
    { width: previewDims.width, height: previewDims.height }
  ]}
  cameraTextureHeight={0}
  cameraTextureWidth={0}
  resizeHeight={300}
  resizeWidth={300}
  resizeDepth={3}
  onReady={handleCameraStream}
  autorender={true}
  useCustomShadersToResize={false}
/>
```

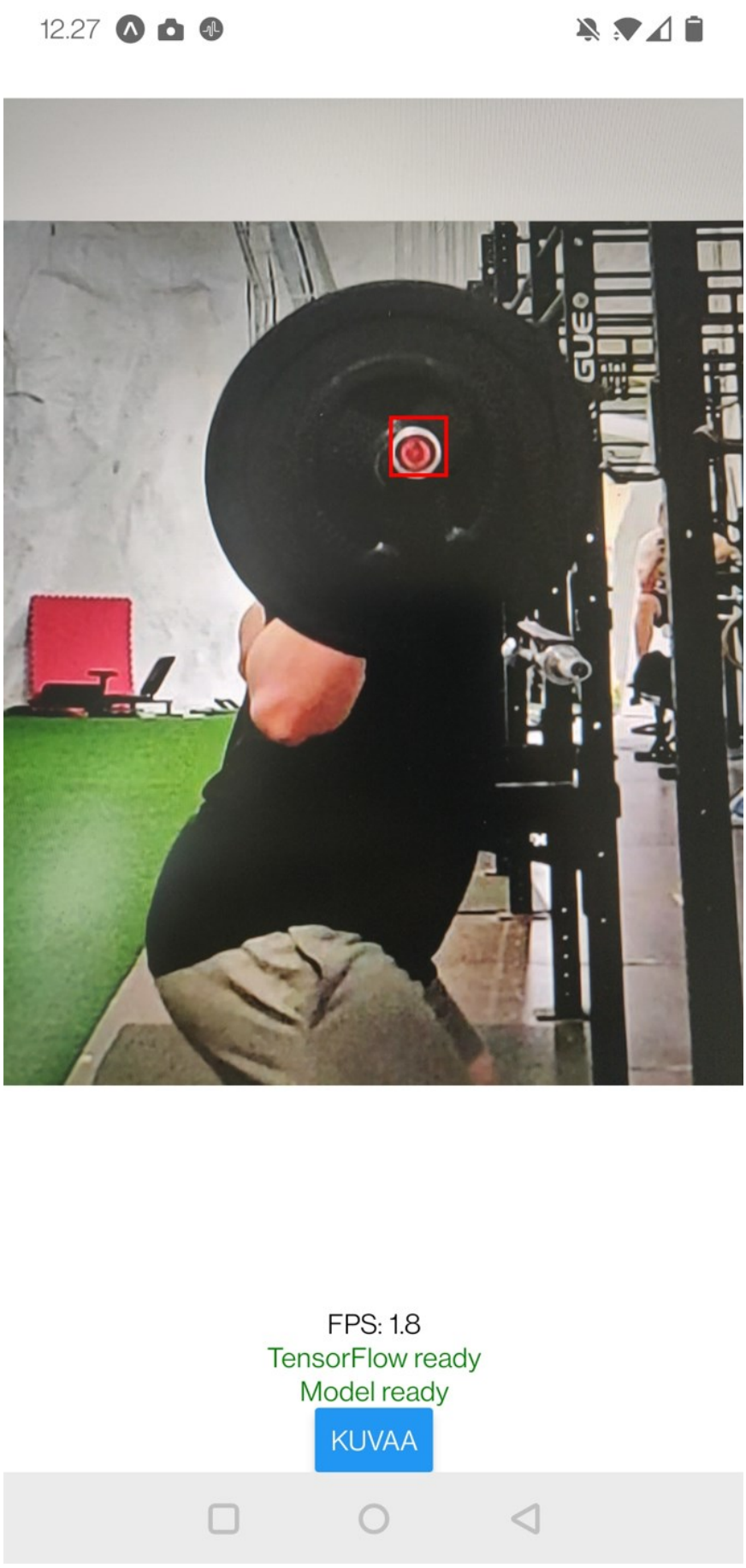
KUVA 13. Komponentti, joka paketoi ns. tyypillisen kamerakomponentin ja palauttaa kuvan tensorimuodossa. Komponentin luontia ei näytetä tässä kuvassa.

Kuvassa 14 on funktio, jossa malli päättelee ennusteet ja kutsuu levytangon pään sijainnin piirtävää funktiota, kun mallin varmuusarvo oli yli 0,1. Tensoriin lisätään yksi ulottuvuus ennen kuin se syötetään mallille, koska kamerakomponentista tulevan tensorin muoto on (300, 300, 3). Tämä tarkoittaa vain koko tensorin paketoimista yksiin hakasulkeisiin, jolloin muodoksi tulee (1, 300, 300, 3). Päättelyyn kuluva aika mitataan ja tallennetaan sovelluksen tilamuuttujaan. Päättelyn tuloksena ennusteet ovat useammassa tensorissa pred-muuttujassa. Indeksin 3 tensorissa ovat kohteiden luokkien varmuusarvot ja tässä luetaan levytangon pään luokan varmuusarvo score-muuttujaan. Indeksin 4 tensorissa ovat ennusteiden sijaintien koordinaatit. Jos varmuusarvo oli yli 0,1, niin sitä vastaava sijainti piirretään videon päälle.

```
const handleCameraStream = (images: any) => {
  const loop = async () => {
    if (typeof model !== 'undefined') {
      const nextImageTensor = images.next().value

      if (!nextImageTensor) return

      const t0 = performance.now()
      const pred: any = await model.executeAsync(nextImageTensor.expandDims(0))
      const t1 = performance.now()

      setInferTime(t1 - t0)

      const score = pred[3].arraySync()[0][0]

      if (score > 0.1) {
        drawBox(pred[4].arraySync()[0][0])
      }
    }

    requestAnimationFrame(loop)
  }
  loop()
}
```

KUVA 14. Funktio, jossa malli päättelee ennusteet, mitataan mallin päättelyyn käyttämä aika ja kutsutaan piirtofunktiota, jos varmuusarvo on yli 0,1.

Valmiista sovelluksesta näkee videokuvan, sen päälle piirretyn suorakaiteen ja kuvataajuuden (kuva 15). Suorakaide piirretään punaisella värillä ja vain silloin, kun ennusteen varmuusarvo on yli 0,1.



KUVA 15. Kuvakaappaus sovelluksesta. Mallin tekemä ennuste on piirretty suorakaiteena videon päälle.

Neuroverkon päättelynopeus OnePlus 6 -älypuhelimella oli noin 500–550 ms. OnePlus 6 -älypuhelimessa on 8-ytiminen Snapdragon 845 siru, joka sisältää Adreno 630 näytönohjain piirin. Uudemmalla OnePlus 8 PRO -älypuhelimella päättelynopeus parani noin 330–360 ms tasolle. OnePlus 8 PRO -älypuhelimessa on 8-ytiminen Snapdragon 865 siru, joka sisältää Adreno 650 näytönohjain piirin. Vaikka päättelynopeus on selkeästi lyhyempi OnePlus 8 PRO -älypuhelimella, niin kuvataajuudet näillä puhelimilla olivat silti vain noin 2–3 FPS.

7 POHDINTA

Tämän opinnäytetyön tarkoituksena oli selvittää mikä on nykyaikaisten älypuhelimien suorituskyky syvien neuroverkkojen ajamisessa paikallisesti laitteessa. Tähän tarkoitukseen kehitettiin prototyyppi mobiilisovelluksesta React Native -sovelluskehyksellä, millä voidaan seurata levytangon päätä videokuvasta. Levytangon pään seuraamista varten koulutettiin mukautettu syvä neuroverkko omalla datalla, joka sisälsi ruutuja videoista, joissa oli levytangon pää suoraan sivusta kuvattuna. Päättelynopeus testatuilla älypuhelimilla vaihteli noin 330–550 ms välillä ja kuvataajuus oli siten 2–3 FPS. Tämän opinnäytteen perusteella React Native -mobiilisovelluksen kautta ajetulla syvällä neuroverkolla voidaan kohtuullisen nopeasti ajaa yhden luokan kohteen tunnistusta. Reaaliaikaiseen kohteen seurantaan on kuitenkin vielä matkaa nykyaikaisillakin älypuhelimilla, kun seuranta toteutetaan kohteen tunnistuksen avulla.

Päättelynopeutta ei kuitenkaan testattu tämän hetken parhaimmilla ja tehokkaimmilla älypuhelimilla, joilla päättelynopeus saattaisi olla selkeästi nopeampaa. Uusimmissa älypuhelimissa saattaa myös olla erityisesti syvien neuroverkkojen prosessointiin erikoistuneita piirejä. Suorituskyky tällaisen piirin sisältävässä älypuhelimessa voi olla selkeästi parempi kuin tässä opinnäytteessä testattujen älypuhelimien suorituskyky.

JavaScript, jota käytetään React Nativen kanssa luo yhden ylimääräisen kerroksen mobiililaitteen varsinaisen laitteistorajapinnan ja mallin väliin. Käyttämällä natiivia Androidin kehitysympäristöä ja luomalla natiivi sovellus olisi saatettu saada nopeampia tuloksia. Tämä työ antaa kuitenkin kuvan siitä, kuinka nopeaa neuroverkon päättelynopeus on JavaScriptin kautta.

Tässä opinnäytteessä testattiin vain yhtä neuroverkkomallia, joten jollain toisella mallilla olisi voitu saavuttaa parempia tuloksia päättelynopeudessa. Valittu malli oli silti esiselvityksen perusteella yksi nopeimmista säilyttäen kohtuullisen tarkkuuden COCO-datajoukolla, kun luotetaan TensorFlow’n tekemiin omiin testeihin. Valittu malli oli MobileNetV2, mutta MobileNet-perheessä on myös uudempi V3-malli. Tämä voisi oletetusti olla hieman nopeampi kuin edeltäjänsä. YOLO-

perheen mallit ovat tämän hetken parhaimpia kohteen tunnistuksessa, joten niillä päättelynopeudessa olisi voitu nähdä eroja. YOLO-mallit ovat kuitenkin TensorFlow'n virallisen tuen ulkopuolella ja niiden toteuttaminen kohteen tunnistusta varten vaatisi syvällisempää osaamista neuroverkoista ja TensorFlow-kirjaston käyttämisestä. Tässä opinnäytteessä saadut tulokset antavat kuitenkin suurpiirteisen suunnan sille, mihin moderneilla neuroverkoilla ja melko uusilla älypuhelimilla pystytään. Vaikka neuroverkkomallina olisikin käytetty YOLO-perheen malleja, niin kuvataajuus olisi tuskin kasvanut moninkertaiseksi, jota reaaliaikaisuus vaatisi.

Suorituskyky resurssirajoitteisissa laitteissa kuten älypuhelimissa ja pienissä IOT-laitteissa on ollut vielä riittämätön ajamaan syviä neuroverkkoja reaaliaikaisesti. Jos neuroverkkoja on käytetty mobiilisovelluksissa, niin usein ratkaisuna on käytetty ulkoista prosessointitehoa. Tällainen voi olla esimerkiksi kolmannen osapuolen ratkaisu tai oma palvelin, jossa päättely suoritetaan omalla mukautetulla neuroverkolla.

Ongelmaksi tulee ensimmäisenä se, että data tulee lähettää pois laitteesta analysoitavaksi ja vastaanotettava takaisin laitteeseen, kun analysointi on valmis. Tämä vaatii jatkuvan yhteyden verkkoon ja se voi olla joissain tapauksissa vaikeaa, jos prosessoitava data tulee verkkoyhteyksistä erillään olevasta paikasta. Vaikka verkkoyhteys olisikin olemassa, niin sen tulee olla myös riittävän nopea erityisesti silloin, kun halutaan reaaliaikaista vastetta.

Nykyaikana yleistyvien 5G-verkkojen myötä yhteyksien nopeudet eivät enää tule rajoittamaan ulkoisen prosessointitehon käyttöä yhtä paljon. Nykyisillä 4G-verkoilla voidaan parhaimmillaan saavuttaa noin 15 – 30 ms viive, joka kuluu pelkästään datan siirtymiseen (ETN, 2019). Neuroverkon päättelyyn ei siis jää kovinkaan paljon aikaa, jos halutaan saavuttaa reaaliaikaisuuden vaatima 40 – 50 ms taso. Elisan mukaan pitkän ajan tavoite on päästä 5G-verkoilla alle 5 ms viiveeseen (ETN, 2019), jolloin neuroverkon päättelyyn jäisi enemmän aikaa. Ennen kuin muutaman millisekunnin taso saavutetaan, verkkojen arkkitehtuuria tulee kehittää, koska nykyisellä tekniikalla ei voida saavuttaa edes teoriassa alle 5 ms tasoa (ETN, 2019). Suomessa 5G-yhteyksillä ei päästy vielä vuonna 2019 merkittävästi pienempiin viiveisiin kuin 4G-yhteyksillä (ETN, 2019). 5G-yhteyksiin

erikoistuneen verkkosivun tekemän raportin mukaan vuosina 2019 – 2020 Yhdistyneissä Kuningaskunnissa 5G-yhteyksien todelliset viiveet olivat 17 – 26 ms, kun taas 4G-yhteyksien osalta 36 – 48 ms (5G.co.uk, 2022).

Ulkoisen prosessointitehon hankkiminen tai ylläpitäminen vaatii myös rahallista panostusta. Paikallisessa prosessoinnissa säästyttäisiin tällaisen infrastruktuurin hankkimisen ja ylläpitämisen kuluilta, tehtäisiin se sitten itse tai ulkoisen palvelun kautta.

Seuraavina ongelmina ovat datan omistajuus ja tietosuoja-asetus. Jos dataa siirretään käyttäjän laitteesta jonkin palveluntarjoajan järjestelmiin, niin datan käsittely ja säilyttäminen tulee ainakin EU:n alueella noudattaa EU:n yleistä tietosuoja-asetusta. Tämä voi joskus muodostua esteeksi käyttäjälle, jos hän ei halua jakaa omaa dataansa. Palveluntarjoaja joutuu myös määrittelemään, miten dataa käsitellään ja säilytetään, sekä käyttämään jonkin verran resursseja myös esim. datan lähetysvaatimuksiin käyttäjiltä.

Paikallinen prosessointi laitteessa voisi poistaa kaikki edellä mainitut ongelmat, mutta sen myötä voi tulla toisenlaisia rajoitteita tai ongelmia. Ainakin toistaiseksi prosessointitehoa ei ole ollut riittävästi, joten laitteisiin tulisi asentaa tehokkaammat prosessointiyksiköt. Tämä vie enemmän resursseja ja saattaa tehdä laitteesta isomman, jos esim. laitteen jäähdytykseen vaaditaan isompia komponentteja. Datan kerääminen voi myös olla tärkeä osa palveluntarjoajan liiketoimintaa tai sen kehittämistä. Jos data jää käyttäjän laitteeseen, niin sitä ei voida hyödyntää.

Paikallinen prosessointi laitteessa on kuitenkin suurella todennäköisyydellä arkipäivää tulevaisuudessa. Laitteiden prosessointiteho tulee jatkossakin kasvamaan, kun piirien arkkitehtuurista tehdään parempia. Piirikomponenttien koko ei nykytekniikan avulla tule enää pienenemään samaan tahtiin kuin Mooren laki ennustaa, koska komponentit ovat jo niin pieniä, että komponenttien tilan määrittämisestä tulee ongelmallista. Vaikka kehitys olisikin hitaampaa, niin se ei tule täysin päättymään, koska kyseessä on massiivinen liiketoiminnan alue. Lisäksi neuroverkkojen arkkitehtuurit ja tekoälyalgoritmit kehittynevät jatkossakin yhä tehokkaammiksi ja taloudellisemmiksi.

LÄHTEET

5g.co.uk. 2022. 5G vs 4G: No Contest. https://5g.co.uk/guides/4g-versus-5g-what-will-the-next-generation-bring/. Viitattu 25.9.2022.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. & Zheng, X. 2016. TensorFlow: A System for Large-Scale Machine Learning. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). arXiv:1605.08695.

Agammed, E. 2020. Deep Learning and Computer Vision Basics I. Medium. Viitattu 13.4.2022. https://medium.com/@elvinaqa/deep-learning-and-computer-vision-basics-i-86acab09e3b7

Ali, A., Jalil, A., Niu, J., Zhao, X., Rathore, S., Ahmed, J. & Aksam Iftikhar, M. 2016. Visual object tracking – classical and contemporary approaches. Frontiers in Computer Science 10 (1), 167–188.

Bewley, A., Ge, Z., Ott, L., Ramos, F. & Upcroft, B. 2016. Simple Online and Realtime Tracking. 2016 IEEE International Conference on Image Processing 3464–3468.

Bishop, C. M. 2006. Pattern recognition and machine learning. New York: Springer.

Bochkovskiy, A., Wang, C-Y. & Liao, H-Y. M. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934v1.

Dumoulin, V. & Visin, F. 2018. A guide to convolution arithmetic for deep learning. arXiv:1603.07285v2.

ETN. 2019. Operaattorien 5G-lupaus toteutuu vasta ensi vuonna. https://etn.fi/index.php/about/13-news/9855-operaattorien-5g-lupaus-toteutuu-vasta-ensi-vuonna. Viitattu 25.9.2022.

Géron, A. 2019. Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow, 2nd edition. Sebastobol, CA: O'Reilly Media.

GitHub. n.d. a. TensorFlow 1 Detection Model Zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md. Viitattu 1.6.2022.

GitHub. n.d. b. TensorFlow.js converter. https://github.com/tensorflow/tfjs/tree/master/tfjs-converter. Viitattu 1.6.2022.

GitHub. n.d. c. TensorFlow Model Garden. https://github.com/tensorflow/models. Viitattu 1.6.2022.

Goodfellow, I., Bengio, Y. & Courville, A. 2016. Deep learning. Cambridge, Massachusetts: The MIT Press.

Google. n.d. Welcome to Colab. https://research.google.com/colaboratory. Viitattu 29.5.2022.

Gordon, J. & Robinson, S. 2018. Introducing TensorFlow.js: Machine Learning in Javascript. Medium. Viitattu 29.5.2022. https://medium.com/tensorflow/introducing-tensorflow-js-machine-learning-in-javascript-bf3eab376db.

Jupyter. n.d. Jupyter. https://jupyter.org/. Viitattu 1.6.2022.

Lin, T. n.d. LabelImg. GitHub. Viitattu 29.5.2022. https://github.com/tzutalin/labelImg.

Lin, T-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L. & Dollár, P. 2015. Microsoft COCO: Common Objects in Context. arXiv:1405.0312v3.

Liu, W., Anguelov, D., Erhan, D., Szedegy, C., Reed, S., Fu, C-Y. & Berg, A. C. 2016. SSD: Single Shot MultiBox Detector. arXiv:1512.02325v5.

Martinex-Alpiste, I., Golcarenarenji, G., Wang, Q. & Alcaraz-Calero, J. M. 2022. Smartphone-based real-time object recognition architecture for portable and constrained systems. Journal of Real-Time Image Processing 19, 103–115.

Medium. 2019. TensorFlow 2.0 is now available. https://medium.com/tensorflow/tensorflow-2-0-is-now-available-57d706c2a9ab. Viitattu 29.5.2022.

Murphy, K. P. 2012. Machine learning: A probabilistic perspective. Cambridge, MA: MIT Press

Nielsen, M. A. 2015. Neural Networks and Deep Learning. Determination Press. http://neuralnetworksanddeeplearning.com/

Papers With Code. n.d. Real-Time Object Detection on COCO. Viitattu 19.5.2022. https://paperswithcode.com/sota/real-time-object-detection-on-coco

Parico, A. I. B. & Ahamed, T. 2021. Real Time Pear Fruit Detection and Counting Using YOLOv4 Models and Deep SORT. Sensors 21 (14), 4803–4834.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. 2016. You Only Look Once: Unified, real-time object detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779–788.

Russell, S. & Norvig, P. 2016. Artificial Intelligence: A Modern Approach, 3rd edition. NOIDA: Pearson Education Limited.

Sandler, M., & Howard, A., Zhu, M., Zhmoginov, A. & Chen, L-C. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381v4.

Sharma, Sid., Sharma, Sim. & Athaiya, A. 2020. Activation functions in neural networks. International Journal of Engineering Applied Sciences and Technology 4 (12), 310–316.

Shivhare, A. & Choudhary, V. 2015. Object Tracking in Video Using Mean Shift Algorithm: A Review. International Journal of Computer Science and Information Technologies 6 (4), 3774–3777.

Soleimanitaleb, Z. & Keyvanrad, M. A. 2022. Single Object Tracking: A Survey of Methods, Datasets, and Evaluation Metrics. arXiv:2201.13066.

Szeliski, R. 2022. Computer Vision: Algorithms and Applications, 2nd edition. Springer International Publishing.

Tan, M., Pang, R. & Le, Q. V. 2020. EfficientDet: Scalable and Efficient Object Detection. arXiv:1911.09070v7.

TensorFlow blog. 2020. TensorFlow.js for React Native is here. https://blog.tensorflow.org/2020/02/tensorflowjs-for-react-native-is-here.html. Viitattu 29.5.2022.

TensorFlow documentation. n.d. API Documentation. https://www.tensorflow.org/api_docs/. Viitattu 29.5.2022.

VideoLAN. n.d. VLC media player. https://www.videolan.org/. Viitattu 1.6.2022.

Wang, C-Y., Bochkovskiy, A. & Liao, H-Y. M. 2021a. Scaled-YOLOv4: Scaling Cross Stage Partial Network. arXiv:2011.08036v2.

Wang, C-Y., Yeh, I-H. & Liao, H-Y. M. 2021b. You Only Learn One Representation: Unified Network for Multiple Tasks. arXiv:2105.04206v1.

Wojke, N., Bewley, A. & Paulus, D. 2017. Simple Online and Realtime Tracking with deep association metric. 2017 IEEE International Conference on Image Processing 3645–3649.

Wolpert, D. 1996. The Lack of A Priori Distinctions Between Learning Algorithms. Neural Computation 8 (7), 1341–1390.

Xu, S., Wang, X., Lv, W., Chang, Q., Cui, C., Deng, K, Wang, G., Dang, Q., Wei, S., Du, Y. & Lai, B. PP-YOLOE: An evolved version of YOLO. arXiv:2203.16250v2

Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M. & Lee, B. 2021. A Survey of Modern Deep Learning based Object Detection Models. arXiv:2104.11892v2.

Zhang, Y., Wang, T., Liu, Kexin., Zhang, B. & Chen, L. 2021. Recent advances of single-object tracking methods: A brief survey. Neurocomputing 455, 1–11.