Tampereen yliopisto

Ari Kukkaro

HÖYRYNTUNNISTUS PROSESSIAUTOMAATIOYMPÄRISTÖSSÄ

Koneoppimispohjainen kuvantunnistus

Tekniikan ja luonnontieteiden tiedekunta
Diplomityö
Tarkastajat:
Professori Matti Vilkko ja
Professori David Hästbacka
Lokakuu 2023

TIIVISTELMÄ

Ari Kukkaro: Höyryntunnistus prosessiautomaatioympäristössä
Diplomityö
Tampereen yliopisto
Automaatiotekniikan DI-ohjelma
Lokakuu 2023

Prosessiautomaatiossa valvontajärjestelmältä vaaditaan pehmeää reaaliaikaisuutta, jossa viive on muutamasta muutamaan sataan millisekuntia ja satunnaisesti tämän aikavälin yli, mistä poikkeaminen voi vaarantaa turvallisuuden. Tästä lähtökohdasta tutkimuksen tavoitteena on rakentaa koneoppimispohjainen konstruktio, jolla voitaisiin toteuttaa valvontaa automaatioympäristössä reunalaskentana lähellä valvottavaa prosessia tai sumulaskentana kauempana prosessista tehokkaammalla laskentalaitteella. Konstruktio koostuu karkeasti ottaen kahdesta osasta. Ensimmäinen osa on pehmeän reaaliaikaisuuden huomioiva höyryntunnistussovellus, joka rajaa kuvasta höyryn. Toisessa konstruktion osassa opetetaan ilman reaaliaikavaateita jatkuvan oppimisen avulla koneoppimismalli tunnistamaan höyryä. Konstruktion osien avulla pyrittiin vastaamaan tutkimuskysymyksiin, mitä palvelunlaadullisia eroja ajantasaisen höyryntunnistuksen suorittamisessa reuna- tai sumulaskentalaitteella. Molemmissa tapauksissa huomioidaan pehmeä reaaliaikaisuus. Sitten toisella konstruktion osalla pyrittiin saamaan vastaus tutkimuskysymykseen mitä hyötyjä ja haasteita on jatkuvan oppimisen hyödyntämisessä koneoppimisessa.

Data mallin harjoittamiseen kerättiin Tampereen Yliopiston höyryä tuottavista osaprosesseista, eikä data ole yleisesti jaossa. Rakennettu konstruktio koostuu viidestä laiteesta. Kuvia keräävästä IP-kamerasta, Intel NUC 10-minititetokoneesta, jossa suoritetaan kuvapalvelinsovellusta. IR1101-teollisuusreitittimestä, jossa reunalaskentana suoritetaan höyryntunnistussovellusta. Neljäs laite on Mac Studio -tietokoneklusteri, jossa sekä suoritetaan aktiivista höyryntunnistusta että harjoitetaan koneoppimisen mallia tunnistamaan höyryä. Viides laite on kannettava tietokone, johon joko IR1101 tai Mac Studio -klusterilta siirretään höyryntunnistuksen tulos. Kuvapalvelin toteutettiin FTP:llä, Höyryntunnistussovellus toteutettiin REST-tyyppisenä Uvicorn-palvelimena. Höyryntunnistuksessa käytettiin YOLOv8n-mallia, jota opetettiin Ultralytics-kirjaston avulla ja MLOps-kanavan mallin harjoitus-, testaus ja käyttöönottovaiheet tehtiin DVC:n avulla.

Tässä työssä käytettiin menetelmänä konstruktiivista tutkimusta, jossa rakennetaan konstruktio, jonka avulla pyrittiin vastaamaan tutkimuskysymyksiin. Konstruktion avulla mitattiin tunnistuksen ja sen tuloksen tiedonsiirron viivettä laskentalaitteelta kannettavalle tietokoneelle. Mittaukset toistettiin kymmenen kertaa sekä reuna- että sumulaskennassa. Kymmenelle mittaukselle laskettiin keskiarvo ja keskihajonta, jotta saatiin keskimääräinen laskennan ja tiedonsiirron viive sekä näiden todennäköinen vaihteluväli. Toisessa osassa mitattiin tunnistustarkkuutta sekä harjoitus- että testausvaiheessa. Mallin jatkuvaa opetusta kolmella peräkkäisellä opetuskierroksella verrattiin mallin harjoittamiseen kaikilla kuvilla kerralla. Tutkimuksen tulokset reunalaskennan ja sumulaskennan eroista olivat, että reunalaskennassa tiedonsiirron viive on hyvin pieni ja laskenta-aika on suuri. Sumulaskennassa laskenta-aika oli 458 kertaa pienempi kuin reunalaskennassa, mutta tiedonsiirron viive oli 322 millisekuntia. Höyryntunnistuksessa harjoitusvaiheessa kolmen opetuskierroksen aikana mallin höyryntunnistustarkkuus laski, vaikka testivaiheessa se oli kolmannella kierroksella hyvä. Kertaopetuksena malli tunnistustarkkuus oli molemmissa vaiheissa hyvä ja yhdenmukainen.

Tällä reunalaskennan laitteella pehmeää reaaliaikaisuutta noudattava höyryntunnistaminen ei ole mahdollista, kun taas sumulaskennan laitteella voidaan tietyissä rajoissa sitä noudattaa. Jatkossa reunalaskentaa voisi tutkia laskentatehoisemmalla laitteella tai sitten tutkia lisää sumulaskennan mahdollisuuksia. Toiseksi jatkuva oppiminen parantaa kohteen tunnistustarkkuutta, mutta 1302 kuvalla ja kolmella opetuskierroksella Ultralytics-kirjastolla kuvien opettaminen kerralla mahdollistaa paremman tunnistustarkkuuden. Näin ollen jatkuvaa oppimista suositellaan toteutettavan uusilla ja vanhoilla kuvilla kerralla. Jatkotutkimuksissa voitaisiin tutkia menetelmiä, joilla jatkuvaa oppimista voitaisiin suorittaa pienemmissä erissä tarkemmin.

Avainsanat: Pehmeä reaaliaikaisuus, Sumulaskenta, Reunalaskenta, MLOps, Höyryntunnistus

ABSTRACT

Ari Kukkaro: Steam detection in process automation environment
Master's thesis
Tampere University
Master's degree programme in automation
October 2023

In process automation monitoring of system requires soft real time capability in which allowed delay range is from couple to couple hundred milliseconds. Also, delay is allowed to exceed limit sometimes, but major deviation can cause safety issues. Considering this goal of this research is design and build a system based on machine learning to compare steam monitoring as edge and fog computing tasks and train steam detection model in fog device. System can be divided into two major components first of which is steam detection application that draws bounding box into image. Moreover, this component of system must consider soft real time requirements. Second component is MLOps pipeline that trains, tests and deploys machine learning model that steam detection application uses. With first component and measurements derived from it, idea is to answer research question what differences in quality of service are there when implementing steam detect on edge or fog device and do they satisfy soft real time requirements. With second component we want to get answer to question which are advantages and challenges in using continuous train in machine learning.

Data used in training ML model was collected from Tampere University processes and data in not publicly available. Build steam detection system consists of five different devices. IP camera captures images. On Intel NUC 10 minicomputer runs image server. On IR1101 industrial router runs steam detection application as an edge computing service. Fourth device is Mac Studio computer cluster which runs both steam detection application and steam detection ML model development pipeline. Lastly laptop executes a client for getting detection results from computing devices. On Intel NUC 10 image collecting server was implemented as FTP server and on IR1101 and Mac Studio inference application was implemented as Uvicorn server that utilizes RESTful resources. As a base of ML model, we used pretrained YOLOv8n model together with Ultralytics Python library. ML pipeline with its three stages was realized using DVC. Notion that all servers were created using Python programming language as it is main language in machine learning.

Research method in this thesis is constructive research where system or construction is designed and built which solves real world problem and gives answers to research questions. For first research case different experiments of edge and fog computing variants were made and in total ten measurements for each were captured. Measurements included inference time and data transmission delays and based on these measurements mean delays and standard deviations were calculated. Second research case in training and test stages steam detection model prediction accuracies and box loss were captured. Training was implemented in three steps so that only group of all images was trained in each step separately and another case was that all the images were trained at the same time.

Results in research case one was that in edge computing data transmission delay was small and inference delay was high. Compared to fog computing where inference delay was 458 times smaller than in edge computing, but data transmission delay was as high as 322 milliseconds when in edge computing it was under a millisecond. Conclusion was that edge computing with this low computing resources is not sufficient to be used in soft real time steam monitoring. But fog computing was sufficient, but not necessarily applicable to all soft real time environments. More research is needed to test more capable computing devices on edge. In ML training after three rounds of continuous learning training phase accuracies were dropping even though in test stage accuracies on last round were good. When training all images and once showed good results both in training and test stage and accuracies were over 90 %. As training each image separately yield odd results recommendation is to apply continuous training so that each time new model is trained former and new images should be bundled together and model train from a scratch. More research is needed to solve problem with CT in smaller batches.

Keywords: Soft real-time, Fog computing, Edge computing, MLOps, steam detection

ALKUSANAT

Kiitokset Tampereen Yliopistolle, että sain olla mukana mielenkiintoisessa PRODI-hankkeessa tutkimusapulaisena sekä diplomityöaiheesta. Tutkimukseen kuuluu tietty epävarmuus ja sitä lisäsi varmasti osaltaan se, etten tiennyt aiheesta kovinkaan paljoa entuudestaan. Kuitenkin sinnikkään työn ja yhteisön tuen avulla kävi ilmi, että uuden oppiminen on mahdollista ja pientä epävarmuutta voi sietää.

Kiitokset ohjaajalleni Professori Matti Vilkolle hyvistä näkemyksistä työn rakenteeseen liittyen sekä ideoista, jotka tarkensivat ajattelua eteenkin automaatioon liittyvien käsitteiden osalta. Kiitokset ohjaajalleni Apulaisprofessori David Hästbackalle termien ja käsitteiden tarkennuksista liittyen IT-alaan ja sen laitteisiin sekä yleisesti tuesta, kun koin epävarmuutta. Kiitokset tuoreelle Tohtori Sergio Moreschinille avusta kuvantunnistukseen liittyvissä kysymyksissä. Kiitokset esihenkilölleni Jari Seppälälle, joka jaksoi opettaa ja ohjeistaa minua automaatiolaitteiden käytössä ja niiden välisessä tiedonsiirrossa sekä tuesta epävarmoina hetkinä. Kiitokset myös muulle työyhteisölle.

Kiitokset läheiselleni myös tuesta ja kärsivällisyydestä.

Sananlaskun sanoin: ”Jos haluat mennä nopeasti, mene yksin. Jos haluat mennä kauas, menkää yhdessä.”

Tampereella, 26.10.2023

Ari Kukkaro

SISÄLLYSLUETTELO

1.JOHDANTO 1
2.HAJAUTETTU LASKENTA PROSESSIAUTOMAATIOSSA 4
2.1 Prosessiautomaation reaaliaikaisuus ja IT 4
2.2 Pilvipalvelut 7
2.3 Reunalaskenta 9
2.4 Prosessiautomaation laskennan ja laskentatehtävien siirtäminen 15
2.4.1 Virtualisointi ja kontit 15
2.4.2 Laskentatehtävien siirtäminen 17
2.5 Koneoppimisen malleihin perustuva laskenta 20
2.5.1 Koneoppimisen malleja ja neuroverkot 20
2.5.2 Koneoppimisprosessi ja MLOps 24
2.6 Kuvantunnistus prosessiympäristössä 27
2.7 Jatkuva oppiminen ja laskennan allokointi 33
2.7.1 Jatkuva oppiminen 33
2.7.2 Laskennan allokointi 35
3.KONSTRUKTIIVINEN TUTKIMUS 38
4.KONSTRUKTION OHJELMISTOT TUTKIMUKSESSA 41
4.1 Roboflow 41
4.2 Docker ja Docker Daemon 42
4.3 YOLOv8-malli 44
4.4 MLOps jatkuva oppiminen Dagshubin avulla 46
4.5 MLflow 48
4.6 Uvicorn-höyryntunnistuspalvelin 50
5.TESTILAITTEISTO JA -SUUNNITELMA 54
5.1 Testilaitteisto 54
5.2 Testisuunnitelma 57
6.HAVAINNOT 61
7.TULOKSET 64
8.YHTEENVETO 70
LÄHTEET 73

KUVALUETTELO

Kuva 1. Reunalaskenta IT/OT-Rajapinnassa……7
Kuva 2. Reunalaskennan rajapinnat ..……..…10
Kuva 3. Yhdistetty reuna- ja pilvilaskenta ..……13
Kuva 4. Sumupilven, pilvipalvelimen ja reunalaskennan sijoittuminen ..……..…14
Kuva 5. Konttipohjainen ja laitteistotason virtualisointi ..……..……16
Kuva 6. Sumulaskennan kolmikerrosmalli ……..……..…18
Kuva 7. Koneoppimisen taksonomia……..…20
Kuva 8. Neuroni ……..……..…22
Kuva 9. Syvä neuroverkko ……..…23
Kuva 10. MLOps:n rakennekuva ……..…25
Kuva 11. Hahmontunnistusprosessi ……..……..…27
Kuva 12. Luokittelu ja kohteentunnistus ……..…29
Kuva 13. Kohteentunnistus YOLO-mallilla ……...……..…31
Kuva 14. YOLO-mallin rakenne ……..……..…32
Kuva 15. Konstruktiivisen tutkimuksen vaiheet ……38
Kuva 16. Docker-ohjelmiston toimintaperiaate ……..…42
Kuva 17. Jatkuvan treenaamisen kanava Dagshubissa ……..…46
Kuva 18. Testilaitteiston pohjapiirustus ……..…56
Kuva 19. Höyryntunnistus reunalaskentana ……..…58
Kuva 20. Höyryntunnistus sumulaskentana ……..…59
Kuva 21. CT-kanavan mittaukset ……..……60
Kuva 22. Höyryntunnistustarkkuus mallin testausvaiheessa……..……..……..62

LYHENTEET JA MERKINNÄT

2D	Kaksiulotteinen
5G	Viidennen sukupolven datayhteys mobiilitekniikassa
(A)NN	Keinotekoinen neuroverkko (eng. (Artificial) Neural networks)
AMD64	AMD:n 64-bittinen suoritinarkkitehtuuri
API	Ohjelmointirajapinta (eng. Application Programming Interface)
ARM	Mikroprosessoriarkkitehtuuri (eng. Advanced RISC Machine)
ARM64	64-bittinen mikroprosessoriarkkitehtuuri
CD	Jatkuva käyttöönotto (eng. Continous Deployment)
CI	Jatkuva integraatio (eng. Continous Integration)
CLI	Komentorivi tai komentokehote (eng. Command Line Interface)
CNN	Konvoluutionaalinen neuroverkko (eng: Convolutional Neural Networks)
CPU	Suoritin (eng. Central Processing Unit)
csv	tiedostoformaatti, jossa arvot erotetaan toisistaan pilkulla (eng. comma separated values)
CT	Jatkuva oppiminen (eng. Continous Training)
CUDA	Alusta ja ohjelmointi rajapinta grafiikkaprosessorilla suoritettavaan laskentaan (eng. Compute Unified Device Architecture)
DAG	Suunnattu syklitön verkko (eng. Directed acyclic graph)
DDR	keskusmuistityyppi (eng. Double Data Rate)
DIN	Saksan standardointi-instituutti (Saks. Deursches Institut für Normung)
DNS	Internetin tai muun verkon nimipalvelujärjestelmä (eng. Domain Name System)
DPM	Muotoiltavien osien koneoppimis malli (eng. Deformable Parts Model)
DVC	Datan versionhallintaohjelmisto (eng. Data Version Control)
FTP	Tiedoston siirtoprotokolla (eng. File Transfer Protocol)
GB	Gigabitti
GPS	maailmanlaajuinen paikannusjärjestelmä (eng. Global Positioning System)
GPU	Grafiikkaprosessori (eng. Graphics Processing Unit)
HTTP	hypertekstin siirtoprotokolla (eng. Hypertext Tranfer Protocol)
IaaS	Infrastruktuuri palveluna (eng. Infrastructure-as-a-Service
IIoT	Teollisuuden asioiden internet (eng. Industrial Internet of Things)
IOS XE	Ciscon Linux-pohjainen käyttöjärjestelmä reitittimelle tai kytkimelle (eng. Internerworking Operating System XE
IoT	Asioiden internet
IOx	Ciscon toteutusympäristö IoT-sovelluksille
IP	Internet protokolla (eng. Internet Protocol)
IPv4	Version neljä internet protokolla
ISA	Kansainvälinen automaatioyhteisö (eng. International Society of Automation)
IT	Tietotekniikka (eng. Information Technology)
JSON	JavaScriptiin perustuva tiedonvälityksen ja tallennuksen tiedostomuotostandardi (eng. JavaScript Object Notation)
LAPI	Paikallinen ohjelmointirajapinta (eng. Local Application Programming Interface)
LTE	Neljännen sukupolven langaton tiedonsiirtotekniikka (eng. Long Term Evolution)
LXC	Virtualisoitujen Linux-konttien suoritusympäristö (eng. Linux Container

MAN	Suurkaupungin laajuinen tietoverkko (eng. Metropolitan Area Network
mAP50	Koneoppimisen keskitarkkuusmetriikka (eng. mean Average Precision 50)
MLOps	Koneoppimisen paradigma, jolla koneoppimisen osia voidaan automatisoida (eng. Machine Learning Operations)
MMDC	Mikromodulaarinen datakeskus (eng. Micro Modular Data Center)
NAS	Verkkoon kytketty tai sen osana toimiva tallennustilapalvelin (eng. Network-attached Storage)
NIST	Yhdysvaltain standardointi- ja teknologiainstituutti (eng. National Institute of Standards and Technology)
NMS-algoritmi	YOLO-mallin käyttämä algoritmi (eng. non-Maximum Supression algorithm)
NUC	Intelin kehittämä kompakti minitietokone (eng. Next Unit of Computing)
ONNX	Koneoppimismallintallennusmuoto (eng. Open Neural Network Exchange)
OT	Teollisuuden laitteisto ja ohjelmistot, joilla järjestelmää ja sen prosesseja ohjataan (eng. Operational Technology)
PaaS	Alusta palveluna (eng. Platform-as-a-Service)
pip	Python-pakettien asennusohjelma (Packet Installer for Python)
PLC	Ohjelmoitava logiikka
QoS	Palvelun laatu (eng. Quality-of-Service)
ReLU	Lineaarinen tasasuunnattu aktivointi funktio (eng. Rectified Linear Unit)
REST	Protokollariippumaton ohjelmointirajapinta-arkkitehtuurityyli (eng. REpresentational State Transfer)
RESTful HTTP	REST-arkkitehtuurityyliä noudattava ohjelmointirajapinta, joka käyttää http-metodeja.
RGB	Punaisen, vihreän ja sinisen värin sävyjä sekoitteleva malli
RTT	Aika, joka kestää lähettää pyyntö edestakaisin asiakkaalta palvelimelle (eng. Round-Trip-Time)
SaaS	Sovellus palveluna (eng. Software-as-a-Service)
SQL	Standardoitu kieli tietokannan luomiselle ja sen käyttämiselle (eng. Structured Query Language)
SVM	Ohjatun oppimisen luokittelu- ja regressionmenetelmä (eng. Support Vector Machine)
TCP	Tiedonsiirron tietoliikenneprotokolla (eng. Transmission Control Protocol)
TF	Tensor Flow
VM	Virtualisoitu tietokone (eng. Virtual Machine)
X86	Intelin kehittämä suoritinarkkitehtuuri
YAML	Tietojen tallennuksen merkintäkieli (YAML Ain't Markup Language)
YOLO	Kuvantunnistukseen erikoistunut koneoppimismalli (eng. You Only Look Once)
a_i	Laskennan i:s osatehtävä a_i
f(a)	Osatehtävistä koostuvan joukon a
i	Indeksiä ilmaiseva muuttuja
k	Tiettyyn laskentatehtävään viittaava muuttuja
w_i	Neuroverkon neuronien i:s painokerroin
x_m	Harjoitusdataan tehdyn sovitteen m:s arvo
x_n	Harjoitusdatan n:s syöte
y_m	Harjoitusdatan tehdyn sovitteen m:nnelle arvolle tehty ennuste
y_n	Harjoitusdatan n:s ulostulo
A.-C.	MLOps mallin osat A:sta C:hen
A_k	Vektori A, jonka koko riippuu laskentatehtävien k määrästä
B_k	Vektori B, jonka koko riippuu laskentatehtävien k määrästä
SxS	Matriisi, jonka leveyden ja korkeuden määrittää vektori S
φ	Neuronin aktivointifunktio

1. JOHDANTO

Prosessiautomaatiossa tulevaisuuden digitalisaation suunta on kytkeä verkkoon yhä tehokkaampia ja edistyneempiä teknologioita, jotka mahdollistavat reuna-, sumu- ja pilvilaskennan, koneoppimisen sekä tuotannon tehostamisen. Pilvilaskennassa suoritetaan laskentatehtäviä usein tuotantolaitoksen ulkopuolella datakeskuksissa, joista vuokrataan laskentatehoa tai tallennustilaa käyttöperusteisesti. Pilvissä laskentaresursseja ja tallennustilaa on paljon saatavilla, esimerkiksi laskentaa voidaan suorittaa kymmenillä tai sadoilla näytönohjaimilla. Reunalaskennassa laskentaa suoritetaan tuotantolaitoksessa lähellä dataa tuottavia antureita tai laitteita. Reunalaskennassa on usein pilveen verrattuna rajoitetumpi laskentakapasiteetti. Sumulaskennassa pilvimäisiä laskentaresursseja tuodaan lähemmän tuotantolaitoksen antureita. Sumulaskenta asemoituu karkeasti ottaen kerrokseksi reuna- ja pilvilaskennan väliin. Tavoitteena teollisuudessa on saavuttaa OT eli tehtaanohjausjärjestelmien ja IT eli tietotekniikan yhdistyminen, jossa tehtaan ohjausjärjestelmistä tietoa voidaan siirtää ja tallentaa tietokantoihin sekä analysoida IT-puolen tarjoamissa datakeskuksissa [45]. OT-järjestelmissä laskenta-, muisti- ja tallennustilaresurssien käyttöä voidaan tehostaa IT:n tarjoamilla teknologioilla kuten virtualisoinnilla ja konttitekniikalla samalla aikaansaaden merkittäviä kustannussäästöjä [5].

Prosessiautomaatioympäristöstä kerättävää dataa voidaan hyödyntää prosessien ohjauksessa, laitteiden ja prosessien kunnonvalvonnassa sekä näiden vikadiagnostiikassa. Prosessihöyryn ulosvirtauksen tunnistaminen kuvasta voidaan toteuttaa koneoppimisen avulla. Koneoppimisen vaiheet datan esikäsittely, mallin luominen ja käyttöönotto tuotannossa voidaan automatisoida erillisiksi kanaviksi, jotka voidaan suorittaa pilvessä, yksityisissä tietokoneklustereissa tai automaatioverkon reunalla reitittimissä tai kytkimissä. Koneoppimisen mallien parempi höyryntunnistustarkkuus vaatii usein paljon opetusdataa ja siten raskaampaa laskentaa, mikä vaikuttaa missä tällaista laskentaa voidaan suorittaa. Tässä tutkimuksessa käyttöönotetun mallin avulla inferenssi tai päätöksenteko eli aktiivinen höyryntunnistus itsessään suoritetaan kahdessa eri paikassa: yksityisessä tietokoneklusterissa ja verkon reunalla. Näitä kahta eri toteutustapaa vertaillaan keskenään.

Haasteena prosessiautomaatiossa ovat reaaliaikavaatimukset, joissa edellytetään tiedonsiirrolta riittävää nopeutta, pientä viivettä ja deterministisyyttä. Tiedonsiirtoon liittyvän viiveen pituus ja vaihtelu tulee tietää, jotta erilaisia automaatiojärjestelmiä voidaan ohjata

onnistuneesti. Lisäksi datan keräämisen, prosessoinnin ja sen perusteella reagoinnin tulee yhdessä tiedonsiirron viiveen kanssa olla reaaliaikaista. Usein ohjausjärjestelmien valvonnassa pehmeä reaaliaikavaatimus on muutamasta muutamaan sataan millisekuntia, mistä systemaattinen poikkeaminen voi aiheuttaa tuotannon katkoja ja pahimmassa tapauksessa vaarantaa turvallisuuden. Toinen haaste on prosessien ohjauksesta kerätyn datan käyttäminen koneoppimisessa. Vaatimus reaaliaikaisesta höyryntunnistuksesta edellyttää pientä viivettä ja sen vaihtelua. Automaatioverkon eritehoiset reuna- ja sumulaskennanlaitteet voivat suorittaa tietyn monimutkaista koneoppimista automaatioverkon reunalla tai kauempana tietokoneklusterissa. Reaaliaikaisen reuna- tai sumulaskennan onnistuminen riippuu käytettävien laitteiden suorituskyvystä. Toisaalta ei-reaaliaikavaateista laskentaa voidaan suorittaa vapaammin eri osissa verkkoa eli automaatioprosessiverkon läheisyydessä, sen ulkopuolella sumulaskentana tai suorittaa laskenta pilvipalvelussa.

Tutkimuskysymyksenä on, mitä palvelunlaadullisia eroja on ajantasaisen höyryntunnistuksen suorittamisessa reuna- tai sumulaskentalaitteella. Palvelunlaatumittareilla tutkitaan, saavutetaanko eri laskentatavoilla pehmeän reaaliaikaisuuden vaatimuksia. Toisena tutkimuskysymyksenä on, mitä hyötyjä ja haasteita on jatkuvan oppimisen hyödyntämisessä koneoppimisessa. Tutkimusmenetelmänä tässä käytetään konstruktiivista tutkimusta, jolla pyritään aikaansaamaan käytännön ongelmaan pohjautuva konstruktio, jonka avulla voidaan saada vastauksia tutkimuskysymyksiin. Tässä työssä rakennettava konstruktio on hajautettuun koneoppimiseen perustuva prosessiautomaation höyryntunnistusjärjestelmä, jonka tunnistustarkkuutta parannetaan jatkuvan oppimisen keinoin. Höyryntunnistusjärjestelmän soveltuvuutta automaatiokontekstiin arvioidaan tutkimuskysymysten kautta. Koneoppimisen ja tietotekniikan teknologiat kehittyvät koko ajan ja voi olla, ettei tässä työssä esitellyt kirjastot ole relevantteja myöhemmin. Kuitenkin eräs työn päämäärä on tarjota tietoa tässä työssä esiteltyjen teknologioiden perustoiminnallisuudesta, joka voi nopeuttaa tutustumista niiden käyttöön.

Tässä työssä luvussa 2 esitellään taustatietoa prosessiautomaation reaaliaikavaatimuksista, reuna- ja pilvilaskennasta, koneoppimisen malleista ja koneoppimisen osien automatisoinnin mahdollistavasta MLOps-paradigmasta. Sitten laskentatehtävien siirtämisestä ja sen mahdollistavista teknologioista sekä lopuksi kuvantunnistamisesta. Luvussa 3 käsitellään konstruktiivisen tutkimusmenetelmän periaate ja sen soveltaminen tämän työn viitekehykseen. Luvussa 4 esitellään konstruktion rakentamiseksi valitut teknologiat, johon sisältyvät ohjelmistot, ohjelmointikieli kirjastoineen, koneoppimisen versionhallinta ja muut työkalut. Lisäksi luvussa 4 esitetään kunkin teknologian perustoiminnal-

lisuudet. Luvussa 5 esitellään tutkimukseen käytettävä testilaitteisto ja määritetään, miten tutkimus aiotaan toteuttaa ja tarkemmin millaisiin kysymyksiin sen tulisi vastata. Luvussa 6 nostetaan esille tutkimuksen aikana esiin nousseet havainnot ja poikkeamat suunnitellusta. Luvussa 7 esitetään työn tulokset, arvioidaan, kuinka kattavasti tulokset antavat vastauksen tutkimuskysymyksiin ja missä kohdin ne ovat puutteelliset. Luvussa 8 on yhteenveto, jossa kerään yhteen työn tärkeimmät lähtökohdat ja niitä seuranneet tutkimustulokset.

2. HAJAUTETTU LASKENTA PROSESSIAUTOMAATIOSSA

Tässä luvussa esitetään taustatietoa tämän työn pohjaksi. Prosessiautomaatio asettaa hajautetulle mallipohjaiselle laskennalle reunaehtoja, joita käydään läpi luvussa 2.1. Luvussa 2.2 pohjustetaan, mitä ovat pilvipalvelut ja miten pilvipalveluja hyödynnetään prosessiautomaatiossa. Luvussa 2.3 käydään läpi pilvi- ja reunalaskennan perusteita ja haasteita ja sitten esitetään reunalaskentaa yleisesti teollisuuden ja tarkemmin prosessiautomaation kontekstissa. Luvussa 2.4 perehdytään kokonaisten laskentatehtävien ja niiden osien siirtämiseen reuna-pilvilaskentajatkumossa. Aliluvussa 2.4.1 esitellään yleisesti laskentatehtävien siirtämisen mahdollistavat teknologiat: virtualisointi ja kontit. Sitten alaluvussa 2.4.2 käydään läpi, miten konttien ja virtualisoinnin avulla voidaan siirtää ja osittaa laskentatehtäviä sumu- ja pilvikerroksen laitteille. Luvussa 2.5 tutustutaan koneoppimisen mallityyppeihin ja esitetään MLOps:n periaatteet.

2.1 Prosessiautomaation reaaliaikaisuus ja IT

Prosessi on määritelty Saksan teollisuusstandardissa DIN 66201 systeemiseksi kokonaisuudeksi toimintoja, jotka vaikuttavat toisiinsa ja joilla materiaalia, energiaa tai tietoa muutetaan, siirretään tai tallennetaan [34, s.4]. ISA (eng. International Society of Automation) määrittelee automaation teknologioiden luomiseksi ja soveltamiseksi valvonta-, tuotannonohjaamis- ja tuotteiden sekä palveluiden tarjoamistarkoituksessa [76]. Prosessiautomaatio koostuu näistä kahdesta elementistä, joissa prosessin tilaa mitataan ja prosessia ohjataan mittauksen perusteella kohti haluttua lopputulosta. Prosessin tuotteena syntyvää materiaalia, energiaa tai tietoa tuotetaan yrityksen tai tuotannon tavoitteiden saavuttamiseksi. Prosessiautomaatiossa on tärkeää huomioida reaaliaikavaatimukset, jotta tuotantoon ei tule katkoja ja ylipäätään itse tuotanto on turvallista.

Reaaliaikaisuus on määritelty Saksan teollisuusstandardissa DIN 44300 tietokoneiden käyttötilaksi, jossa järjestelmän ulkopuolelta saapuvaa dataa prosessoivien ohjelmien tuottamat tulokset ovat saatavilla ennalta määritetyssä ajassa riippumatta siitä, onko datan saapuminen satunnaisesti jakautunut vai ei [34, s. 1]. Prosessiautomaatiossa prosessia ohjataan mittauksen perusteella laskettavien parametrien mukaan tietyin väliajoin. Prosessienohjauksen laskenta voidaan suorittaa prosessin ulkopuolella, jolloin myös tiedonsiirto aiheuttaa viivettä. Tiedonsiirron viiveen pituus voi vaihdella, mikä voi

aiheuttaa prosessin epästabiloitumista, jolla voi olla vaikutusta prosessin turvallisuuteen. Toisena prosessin epästabiloituminen voi aiheuttaa tuotannon katkoja, koska prosessin saattaminen uudelleen toimintakuntoon voi kestää. Tällöin prosessinohjaamisessa edellytetään reaaliaikaisuutta, jossa ensinnäkin tiedonsiirron viiveen vaihtelu tai sen pituus on määritetyssä aikaikkunassa ja toiseksi datan kerääminen, prosessointi ja sen perusteella reagointi tulee yhdessä tiedonsiirron viiveen kanssa olla reaaliaikaista [34].

Reaaliaikaisuuden osa-alueita ovat samanaikaisuus ja oikea-aikaisuus. Automaatiojärjestelmän ohjaamiseksi voidaan tarvita useita syötteitä eri lähteistä samanaikaisesti. Eteenkin prosessoitavat syötteet pitää siirtää sinne, missä prosessin ohjaus parametrit lasketaan yhtaikaa edellyttävät järjestelmältä samanaikaisuutta. Oikea-aikaisuus tarkoittaa datan keräämistä, arviointia ja sen perusteella reagointia tietyssä ajassa. Oikea-aikaisuuden alakäsitteitä ovat ennustettavuus ja luotettavuus. Järjestelmän tulee olla ennustettava huolimatta siitä, että prosessointia tarvitseva data saapuu satunnaisilla ajanhetkillä. Prosessin tulee tuottaa tuloksia suunnitellusti ja ennakoidusti, vaikka syötteet saapuisivat prosessiin satunnaisesti. Ennustettavuus mahdollistaa oikea-aikaisuuden. Luotettavuus tarkoittaa, että järjestelmä kykenee jatkamaan toimintaansa virheistä ja häiriöistä huolimatta. [34]

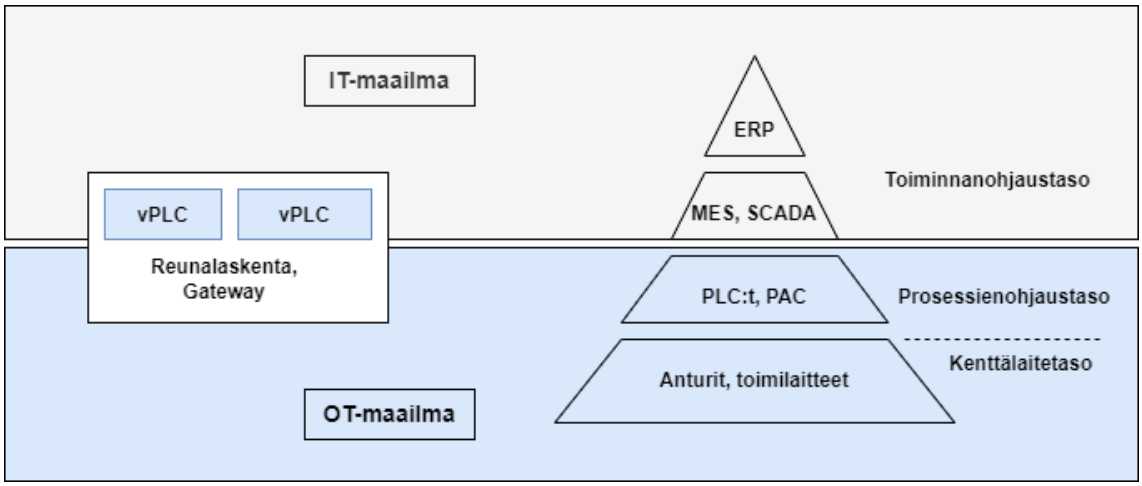
Reaaliaikaisuus voidaan jaotella ”Best effort”, pehmeään, kovaan ja isokrooniseen reaaliaikaisuuteen. **Best effort** reaaliaikaisuudessa siirrettävälle tiedolle ei ole määritetty ajanhetkeä, jolloin tiedon tulee olla perillä. Myöskään tiedon hyödynnettävyys tai käyttökelpoisuus ei vähene ajan kuluessa. Pääasia on, että tieto saapuu perille mahdollisimman pian. **Pehmeässä reaaliaikaisuudessa** tiedon käyttökelpoisuus on optimaalisin johonkin ajan hetkeen asti. Tämän jälkeen tiedon käyttökelpoisuus laskee mitä enemmän aikaa kuluu. Jonkin ajanhetken jälkeen tieto on vanhentunutta eikä näin ollen ole enää hyödynnettävissä. **Kovassa reaaliaikaisuudessa** siirrettävä tieto sen sijaan on käyttökelpoista johonkin määräaikaan asti. Välittömästi määräajan ylityksen jälkeen tieto on käyttökelvotonta. **Isokroonisessa reaaliaikaisuudessa** tieto on käyttökelpoista jollain ennalta määrätyllä aikavälillä. Aikavälin ulkopuolella tieto ei ole käyttökelpoista. Automaatiossa reaaliaikaisuus on usein kovaa ja isokroonista, sillä prosessien säätösyklit vaativat datan perille saapumisen määräajassa, mutta kunnonvalvonnassa tai vikadiagnostiikassa riittää vaatimukseksi myös pehmeä reaaliaikaisuus.

Reaaliaikaisuuden mittaamisessa voidaan soveltaa palvelunlaatumittareita (QoS). Palvelunlaatu määrittää tässä kontekstissa prosessiautomaation ja sen laskentainfrastruktuurin muodostaman verkon vaaditun kokonaissuorituskyvyn [76]. Verkon latenssi kuvaa tiedonsiirron viivettä, joka muodostuu siirtäessä tietoa lähettäjältä vastaanottajalle. La-

tenssilla voidaan arvioida esimerkiksi, kuinka kauan kestää saada prosessin mittaus siirrettyä anturilta ohjausparametrien laskentaa varten muualle verkkoon. Round-Trip-Time, RTT, määrittää ajan, joka kestää siirtää tietoa lähettäjältä vastaanottajalle ja sieltä takaisin lähettäjälle. Prosessiautomaatiossa RTT:llä voidaan mitata prosessista kerätyn datan siirtämisen esimerkiksi pilvipalveluun, pilvessä datan prosessoinnin ja siellä lasketun ohjauksen siirron säätimelle. RTT näin ollen määrittää sekä tiedonsiirtoon että mittausdatan laskentaan kuluvan ajan. Jitterillä voidaan mitata viiveen pituuden tai muiden verkon ominaisuuksien poikkeamia tai vaihteluita. Jitterillä mitataan esimerkiksi prosessien ohjausparametrien tiedonsiirron viiveen vaihteluita, jotta voidaan arvioida prosessinohjauksen oikea-aikaisuutta.

Viime vuosina tehdasautomaatio- ja prosessienohjausjärjestelmät eli OT ja tietotekniikka eli IT ovat lähentyneet. Tietotekniikan käsitteitä sovelletaan automaatiojärjestelmiin yhä enemmän. Päämääränä on saada yhä monimutkaisempien ja vaativampien automaatiojärjestelmien tuottama tieto käytettäväksi ja helpommin hallittavaksi. Kun Industry 4.0 ja IIOT mullistavat tuotannon tehokkuuden, joustavuuden, nopean reagoinnin kysyntään ja energiatehokkuuden, tarvitaan näiden toteuttamiseksi tarkempaa reaaliaikaisuutta, turvallisuutta ja laitteiden liitettävyyttä. Lisäksi tiedon tulee olla saatavilla mistä paikasta tahansa, ei vain OT- ja IT järjestelmistä. Tämän vuoksi on tärkeää poistaa OT: ja IT: rajaaitoja, jotta korkeamman tason automaatioratkaisuja voidaan tuottaa. [54,57]

IT-alalla IP-pohjaista tiedonsiirtoa on käytetty tavallisesti kytkemään koneita yrityksen verkkoon, mutta automaation kenttälaitteissa sitä ei ole käytetty puutteellisen reaaliaikaisuuden vuoksi. Kuitenkin viime aikoina tutkimuksen suuntana on ollut kehittää determinististä IP-pohjaista tiedonsiirtoa, joka tukee prosessiautomaation reaaliaikaista teollisuus-Ethernet-viestintää. Esimerkiksi ohjelmoitavien logiikoiden ajaminen tai yleisesti prosessienhallinnan tehtäviä voidaan siirtää verkon reunalla olevien determinististen yhdyskäytävien, kuten reitittimien tai kytkimien, suoritettavaksi. Lisäksi prosessinhallinta tai PLC:n ajaminen voidaan palvelullistaa pilvi- tai reunalaskennan alustalle tarpeen mukaan saataville. Uusi automaatio- ja tiedonsiirtoteknologia pyrkiikin tarjoamaan reaaliaikaisuuden, mutta myös korjaamaan OT:n puuteet järjestelmän joustavuuteen ja laajennettavuuteen liittyen. [6]



Kuva 1: Reunalaskenta IT/OT-Rajapinnassa [6]

Kuvassa 1 on esitetty virtuaalisten ohjelmoitavien logiikoiden (vPLC) ajaminen reunalaskennan yhdyskäytävässä (eng. Gateway). Kuvassa pyramidin alimpana tasona on kenttälaitetaso, johon kuuluvat anturit ja toimilaitteet. Sen yläpuolella on prosessienohjaustaso, johon kuuluvat ohjelmoitavat logiikat ja säätimet. Hallinnointitasoon kuuluvat tuotannonohjaus ja -valvontajärjestelmät (MES, SCADA) ja ylimpänä on yrityksen toiminnanohjausjärjestelmä (ERP). IT/OT-rajapinnassa hajautetuille reunalaskentalaitteille voidaan siirtää prosessidataa analysoitavaksi tai koneoppimisen avulla laskettavaksi lähellä tuotannon koneita. Reunalaskentalaitteilta vain tarpeellinen data voidaan siirtää eteenpäin pilveen. [6]

2.2 Pilvipalvelut

Datakeskuksissa olevilla laitteilla on resursseina prosessointitehoa, muistia ja tallennustilaa. Erilaiset palvelimet voivat jakaa saman tietokoneen resursseja virtualisoinnin avulla niin, että tietokoneen resurssit tulisivat hyödynnettyä mahdollisimman tehokkaasti. Datakeskuksen laitteistojen resursseja hyödyntävät palvelimet tarjoavat erilaisia verkko-, sovellus- ja tietokantapalvelimia. Pilvet ovat datakeskuksia ja pilvipalvelut ovat yksittäisiä tai kokoelma useampia palveluja, joita datakeskuksien erilaiset palvelimet tarjoavat. Tässä työssä on jaoteltu pilvet yksinkertaisuuden vuoksi kahteen erilaiseen luokkaan, julkiset pilvet ja yksityiset pilvet. Näitä kahta yhdistelemällä tai skaalaamalla voidaan muodostaa ”hybridi- ja yhteisöpilviä” [36,47].

Julkiset pilvipalvelut ovat jonkin palvelun tarjoajan, kuten Amazon Web Sevice, Microsoft Azure tai Google Cloud, tuottamia asiakkaan erilaisiin tarpeisiin sopivia palveluja. Pilvi-

palvelun tuottajat tarjoavat esimerkiksi tietokoneresursseja, valmiita palvelimia, arkkitehtuurin tai laitteiston, jolle asiakas voi rakentaa oman kustomoidun palvelimensa ja valmiita sovelluksia. SaaS (eng. Software-as-a-Service) on palvelumalli, jossa asiakkaalle tarjotaan verkkoselaimella käytettävä sovellus, jonka konfigurointi on rajoitettua. SaaS palvelumallissa kuitenkin palveluntuottaja vastaa palvelun toimivuudesta ja kaikesta arkkitehtuurin hallinnasta ja huolloista, joita palvelun tarjoaminen vaatii. Sen sijaan PaaS (eng. Platform-as-a-Service) tarjoaa asiakkaalle alustan, jolle asiakas voi ohjelmoida oman palvelunsa ja konfiguroida sitä. Palveluntarjoaja huolehtii alustan toteuttamiseen vaadittavasta arkkitehtuurista. IaaS-palvelumallissa (eng. Infrastructure-as-a-Service) asiakkaalle tarjotaan laskentaresurssit ja palvelin, jolle asiakas voi asentaa itse ohjelmistoja ja hallita niitä. IaaS-mallissa palveluntarjoajalla on vähemmän vastuuta palvelun hallinnasta ja asiakkaan vastuu on suurempi. [47]

Pilvipalvelulle NIST-järjestö (eng. National Institute of Standards and Technology) määrittelee viisi ominaispiirrettä, joiden perusteella palvelua tai sovellusta voidaan kutsua pilvipalveluksi: pilveen on pääsy kaikkialta, asiakkaalla on tarvittaessa palvelu saatavilla, resurssien jaettavuus (eng. pooling), nopea elastisuus ja palvelun käytön mittaaminen. Asiakkaalla tulee olla pääsy pilvipalveluun sijainnista rippumatta tavallisen internetyhteyden avulla. Lisäksi asiakkaalla tulee olla mahdollisuus päästä käyttämään palvelua eri laitteilla esimerkiksi kannettavalla, pöytäkoneella, puhelimella ja muilla älylaitteilla, joissa on tavallinen internetyhteys. Toisena asiakkaalle pääsy palveluun ja mahdollisuus pyytää palvelua tulee olla saatavilla riippumatta kellon ajasta. [36,47,53]

Kolmantena pilvipalvelun tulee mahdollistaa tehokas resurssienkäyttö siten, että muut asiakkaat voivat käyttää vapaana olevia resursseja. Resurssien jakaminen mahdollistaa joustavan asiakaspalvelun. Neljäntenä pilvipalveluympäristön tulee voida joustaa kysynnän mukaan. Kun palvelussa on paljon kysyntää, pilvipalvelu skaalautuu horisontaalisesti eli palvelun toiminnan mahdollistamiseksi verkkoon automaattisesti kytkeytyy lisää palvelimia. Uudet palvelimet tasaavat toisin sanoen palveluun kohdistuvaa kuormaa. Pilvipalvelu ”skaalautuu ulos”, kun sen mahdollistamiseksi verkkoon liittyy lisää palvelimia. Toisaalta palvelu ”skaalautuu sisään” kysynnän pienentyessä, jotta resursseja vapautuu muuhun käyttöön. Pilvipalvelun skaalautuvuudella voidaan reagoida kysynnänvaihteluihin. Viidentenä palvelun käyttöaikaa, sekä kaistanleveyden- että datankäyttöä pitää mitata. Käytönmittaus mahdollistaa palvelumallin, jossa asiakkaalta veloitetaan palvelusta käyttöperusteisesti [36,53].

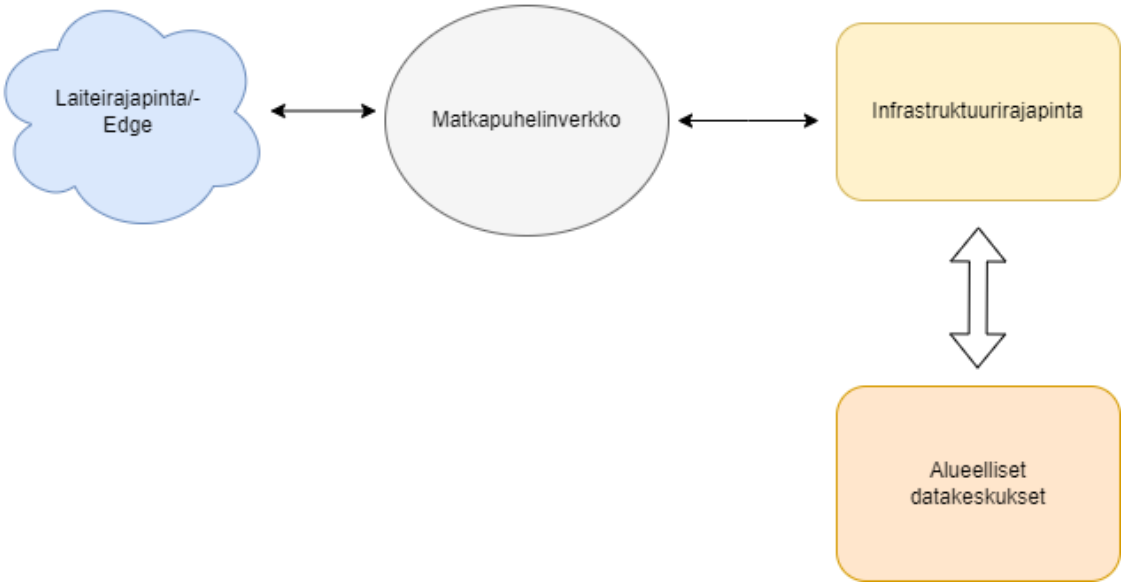
Teollisuuden tuotanto- ja hallintateknologioiden sekä pilvilaskentateknologioiden yhdistäminen muodostaa teollisuuspilvilaskentajärjestelmän, mikä parantaa teollisuuden tie-

tojenhallintaa. Pilven laskentateho, tallennustila ja jaettujen resurssien uudelleenkäyttäminen auttavat yrityksiä seuraamaan Teollisuus 4.0:n ja siihen liittyvän IIOT:n (eng. Industrial Internet-of-Things) kehitysaskeleita. Käytännössä matalan tason automaatiossa eli prosessien ohjauksessa ja kenttälaitetasolla suoritettavia tehtäviä voidaan tarjota pilvipalveluina. SaaS-palveluna voidaan toteuttaa automaation erityistehtäviä ja PaaS-palveluna voidaan toteuttaa automaatioalusta esimerkiksi prosessitiedon lokikirjan tai helposti määritettävien (eng. Plug-n-Play) tunnuslukujen integroimiseen. Lisäksi PaaS-palvelua voidaan hyödyntää pilvilaskennan tuottaman tiedon visualisointiin. Tiedon esitys kuvina ja graafeina mahdollistaa tarpeellisen tiedon suodattamisen ja sekä laskennan oikeellisuuden että mielekkyyden. [19,30]

2.3 Reunalaskenta

Reunalaskennassa reuna (eng. Edge) on käsite, joka kuvaa kahden asian yhtymäkohtaa tai rajapintaa. Reunan käsite voidaan jaotella lähteen [17] mukaan laiterajapintaan, MAN- tai WAN-tason (eng. Metropolitan Area Network ja Wide Area Network) eli suurkaupungin laajuiseen tai maantieteellisesti laajaan puhelinverkkoon, infrastruktuurirajapintaan ja alueellisiin datakeskuksiin. Laiterajapinnassa on jokin kooltaan, kyvykkyydeltään, kuten prosessointiteholta, muistilta tai tallennustilalta, rajoittunut laite ja tällaisen laitteen hankintahinta on myös pieni. Laiterajapinnassa oleviksi ’Edge-laitteiksi’ lukeutuvat muuan muassa älypuhelimet, tietokoneet, teollisuuden robotit, älykkäät anturit ja muut IOT-laitteet. Laiterajapinnan laitteet kommunikoivat epäsuorasti WiFi:n avulla yhdyskäytävään (eng. Gateway), esimerkiksi teollisuusreitittimeen ja sitä kautta tai suoraan LTE-puhelinverkon eli neljännen sukupuolen langattoman tiedonsiirtotekniikan avulla infrastruktuurirajapintaan. [17,51]

Infrastruktuurirajapinnassa sijaitsevat mikromoduulaariset datakeskukset, MMDC:t, jotka ovat alueellisiin datakeskuksiin verrattuna pienempiä ja sijaitsevat lähempänä sekä loppukäyttäjää että puhelinverkon linkkejä. Infrastruktuurirajapinnan laitteilla on enemmän resursseja kuin laiterajapinnan älykkäillä laitteilla, jotka saattavat toimia akuilla. Alueelliset datakeskukset ovat suuria ja niissä on monella ytimellä, isolla muistilla ja tallennustilalla varustettuja tietokoneita. Tällaisissa tietokoneissa raskaampien sovellusten suorittaminen on mahdollista. Pienet datakeskukset toimivat yksityisenä tai julkisena pilvenä. Kuvassa 2 on esitetty reunan käsitteen jaottelu rajapintoihin. [17,51]



Kuva 2: Reunalaskennan rajapinnat [51, s. 7]

Reunalaskentaa voidaan suorittaa eri tavoin, eikä kirjallisuudessa ole tarkkaa määritelmää reunalaskennalle [9]. Kuitenkin karkeasti ajatellen IOT-laitteet voivat toimia antureista tai toimilaitteista saadun datan kerääjinä, ja tämä data siirretään analytiikkaa tai päätöksentekoa varten pilveen. Kevyempää datan suodatusta voidaan suorittaa verkon tai laiterajapinnan reunalla yhdyskäytävissä tai muissa laitteissa. Toisaalta kevyttä analytiikkaa voidaan suorittaa myös verkon reunalla. Reunalaskenta mahdollistaa lyhyemmän tiedonsiirtomatkan ja latenssin kuin tiedon siirtäminen pilveen. Tiedonsiirron latenssi laitteiden välillä on mikro- tai millisekuntien luokkaa, kun taas puhelinverkon kautta tiedonsiirron latenssi voi olla jopa satojen millisekuntien luokkaa. [17,51]

Verkon reunalla tapahtuva laskenta pienentää latenssi, mutta myös parantaa IOT-sovellusten palvelunlaatuvaatimuksia: vasteaikaa, resilienssiä, herkkyyttä, kaistanleveyttä ja mahdollistaa suuremman datamäärän siirtämisen. Reunalaskenta on pilvipalvelua täydentävä teknologia, joka tarjoaa parempaa palvelua loppukäyttäjille viiveherkkien sovellusten tapauksessa. Reunalaskennassa ja pilvipalveluissa on samankaltaisuuksia, mutta eroina ovat palveluiden sijainti. Reunalaskennassa palvelut sijaitsevat verkon reunalla, kun taas pilvessä palvelut ovat saatavilla internetistä käsin ja ne sijaitsevat datakeskuksissa. Reunalaskenta myös on sijaintitietoinen ja tarjoaa laadukkaan tuen liikkuville laitteille toisin kuin pilvi. Kolmanneksi reunalaskennassa käytetään palvelimille hajautettua mallia verrattuna keskitetyn palvelinmallin pilveen. Lisäksi reunalaskennassa skaalautuvuus on huonompaa reunan laitekannan rajallisten kyvykkyyksien vuoksi. [9,14]

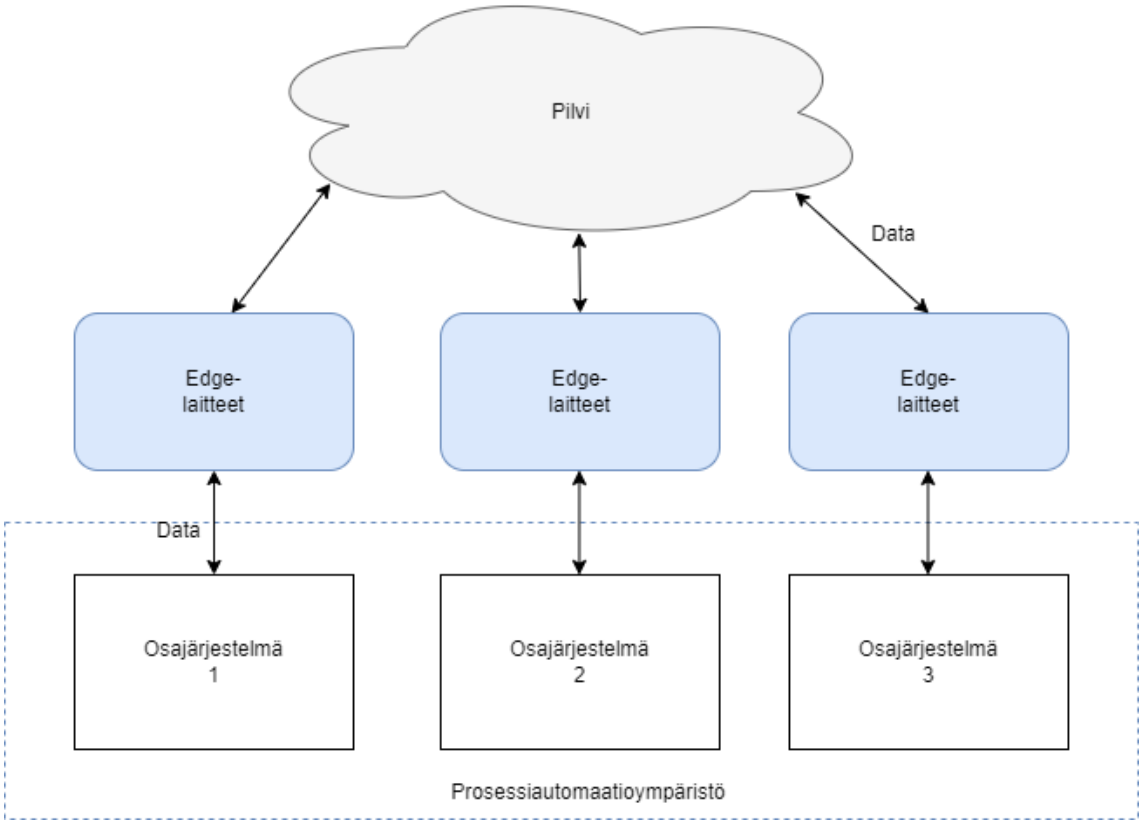
Reunalaskennan erityispiirteitä ovat tuki liikkuville laitteille, maantieteellinen laitteiden tiheys, sijaintitietoisuus, käyttäjäläheisyys, pieni latenssi, kontekstitietoisuus ja verkon heterogeenisyys. Verkkoaseman ja paikan identiteetin eriyttäminen ja hajautettu hakemistojärjestelmä, jonka avulla eriytetyt laitteet löytävät toistensa osoitteet, mahdollistavat tuen liikkuville laitteille. Reunalaskennassa, *big datan*, eli suuren datamassan kerääminen laitteilta ja antureilta, reaaliaikainen analytiikka ja verkon tehokas käyttö on tiheästi hajautettujen laitteiden ansiota. Sijaintitietoisuus ilmenee reunalaskennassa niin, että eri paikoissa olevat käyttäjät voivat käyttää puhelinverkon, GPS:n tai langattoman verkon liityntäpistettä porttina reunalaskennan palvelimelle, joka sijaitsee lähimpänä käyttäjän omaa sijaintia. Tämä on syy, miksi reunalaskennan tuki liikkuville laitteille on parempi kuin pilven. [9]

Käyttäjäläheisyydessä asiakkaat saavat tietoa palvelujen käyttöön ja verkon vaihtoehtoisiin tiedonsiirtotapoihin liittyen. Toisaalta palveluiden tuottajat saavat tietoa asiakkaiden käyttämistä laitteista ja siitä, miten asiakkaat palveluja käyttävät. Tällaisten tietojen perusteella palvelujen tuottajat pystyvät kehittämään sekä palveluja että niiden resurssienhallintaa. Laskennallisten resurssien ja palvelujen tuominen lähemmäs asiakkaita pienentää latenssia resursseja vaativissa ja viiveherkissä palveluissa. Kontekstitietoinen reunalaskennan ominaisuus tuottaa reaaliaikaista tietoa verkon kuormituksesta ja käyttäjän sijainnista, millä voidaan parantaa palvelun laatua. Heterogeenisyys tarkoittaa reunalaskennassa verkossa olevien alustojen, arkkitehtuurien, laskenta- ja kommunikaatioteknologioiden monimuotoisuutta. Erilaisten reunalaskennan palvelimien ohjelmointirajapintojen, API, avulla verkossa eri teknologiat saadaan toimimaan yhdessä. [9]

Reuna- ja pilvilaskennan lisäksi on olemassa sumulaskenta (eng. Fog computing), joka on perinteisen pilvilaskennan laajennus. Sumulaskennassa pilvipalvelua ja sen resursseja, laskentatehoa, muistia, tiedonvälitystä ja päätöksen tekoa tuodaan lähemmäs verkon reunaa. Reunalaskennan ja sumulaskennan erottava tekijä on se, että sumulaskenta sisältää pilven ominaisuuksia eikä vain laskentaa, vaan laajemmin kommunikoinnin, verkkojen muodostamisen, päätöksenteon ja verkon optimoimisen mahdollistavia teknologioita. Reunalaskentaan verrattuna monimutkaisempaa sumulaskentaa voidaan suorittaa, mutta reaaliaikaisemmin kuin kaukana pilvessä. Esimerkiksi IOT-laitteilta kerättyä dataa voidaan suodattaa ja luokitella sumulaskennan laitteilla, kuten reitittimissä, pienissä datakeskuksissa tai palvelimilla verkon reunan läheisyydessä, ennen pilveen siirtämistä. Toisaalta osa laskentatehtävistä voidaan suorittaa sumulaskentana reunalaitteita paremman laskentatehon ansiosta. Keskitetyn pilvilaskennan voidaankin ajatella täydentävän reunalaskennan hajautettua resursointia ja laskentaa, ja sumulaskenta on näiden kahden laskentatavan emergenssi. [3,21]

Prosessiautomaatiossa koneisiin yhdistetyillä laitteilla on rajallinen laskentateho ja tallennustila, jolloin big datan prosessointi, tallennustilan tarve sekä tiedon keruu tuotannon tilasta on erittäin tärkeää alati älykkäämmissä tuotantoympäristöissä. Big data kuvaa tietoa, jota on todella paljon ja usein tällainen tieto voi olla osittain jäsenneltyä ja jäsentelemätöntä, ja vielä tietoa voi generoitua nopeasti. Tällöin tarvitaan raskasta laskentaa, jotta big datasta saadaan tuotettua oleellista tietoa. Tietoverkkoarkkitehtuuri kuormittuisi kaikesta eri laitteilta kerätystä datasta, jos data kerättäisiin vain keskitetysti pilveen. Toinen heikkous keskitetyssä pilvilaskenta-arkkitehtuurissa on se, että pilvi on usein kaukana tehtaan laitteista, milloin tiedonsiirron latenssi on korkea, eikä siltä osin sovellu tyydyttämään tuotannon säätöpäätöksenteon vaatimuksia. [5,41,49]

Reunalaskennan käyttö prosessiautomaatiossa ratkaisee keskitettyyn pilveen liittyviä ongelmia, mutta tuo myös mukaan järjestelmänhallinnan, synkronoinnin, tietoturvan, yksityisyyden ja verkon teknologioiden integroimisen haasteita. Pilvilaskennan edut korkea skaalautuvuus ja yhteensopivuus yhdessä reunalaskennan pienen latenssin yhdistettynä arkkitehtuurina tarjoavat parempaa suorituskykyä kuin kumpikaan teknologia yksinään. Yhdistetty pilvi- ja reunalaskenta, kuvassa 3, mahdollistavat teollisuuden säätöjärjestelmien tietoisuuden järjestelmän virheistä ja tuotannon laadusta, diagnostiikan, toipumisen vikatilanteissa ja tuotannon ennustettavuuden. Lisäksi pilvessä koneoppimisen avulla analysoitu big data tuottaa teollisuuden järjestelmistä parempaa tietoa päätöstenteon tueksi. [5,41,49]

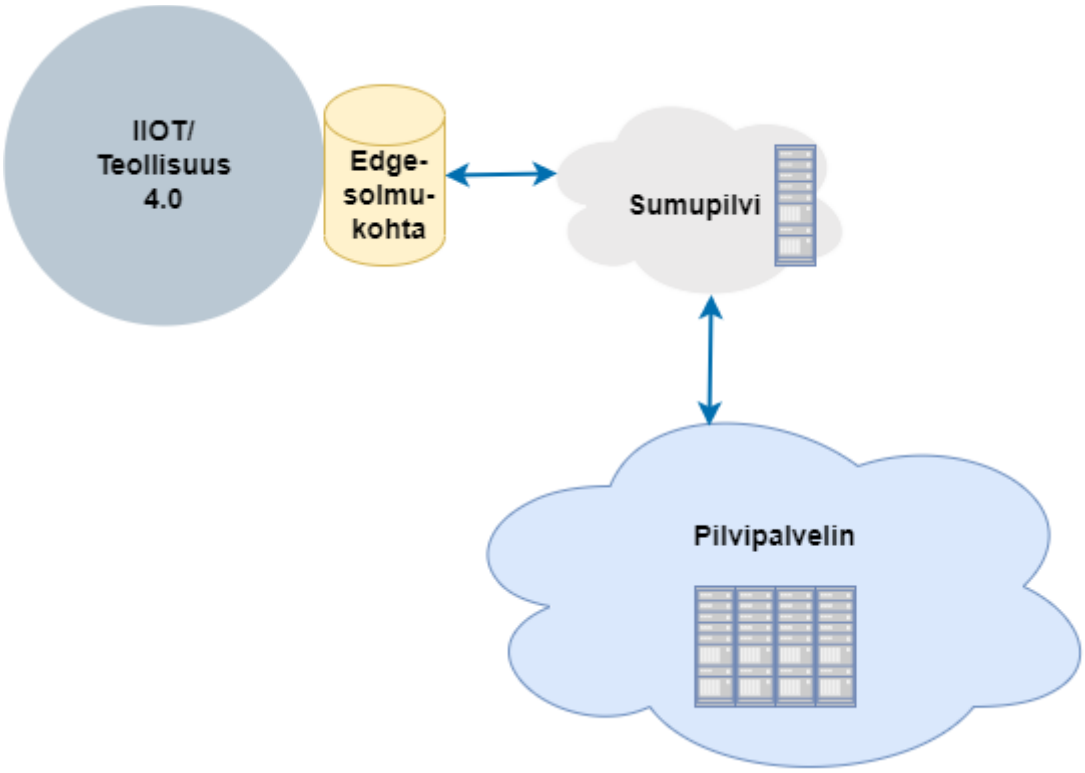


Kuva 3: Yhdistetty reuna- ja pilvilaskenta [6]

Reunalaskennan ”Edge-laitteilla” on erilaiset laskentakapasiteetit, mikä tulee huomioida Edge-laitteiden käyttöönotossa. Prosessiautomaation laitteet suorittavat tehtäviä, joiden suorituskyvyille on omat vaatimuksensa, siksi monenlaisten tehtävien, esimerkiksi monitoroinnin ja säädön, suorittaminen voidaan joutua jakamaan useammalle Edge-laitteelle. Jokaisella verkon Edge-laitteen laskentakapasiteetti viimekädessä määrää sen, millaisia laskentatehtäviä kyseinen reunalaskentalaite kykenee suorittamaan. Tällöin on tärkeää tunnistaa Edge-laitteiden sijainti ja kyvykkyys, jotta koko järjestelmän suorituskyky, laskentatehtävien suorittamiseen kuluva aika ja päätöksenteon eli laskennan tuloksen perusteella reagoinnin tarkkuus voidaan optimoida. Laskentatehtävien hajauttamiseen perehdytään tarkemmin luvussa 2.5. [5]

Toisaalta sumulaskennan käyttäminen automaatiossa parantaa reaaliaikaisuutta ja laitteiden välisen kommunikaation luotettavuutta. Tämän lisäksi sumulaskenta on joustavaa, sillä sen avulla voidaan hyödyntää jo olemassa olevia, vanhempia laitteita osana IIOT-verkkoa. Automaatiossa laitteilla on yleensä pitkä elinkaari, milloin sumulaskenta mahdollistaa tällaisten laitteiden hyödyntämisen pidempään. Keskitettyyn pilveen verrat-

tuna hajautettu sumulaskenta parantaa tietoturvaa ja yksityisyyden suojaa, sillä prosessista tai muusta kerätty tieto voidaan tallentaa palvelimille tai tietokantaan asiakkaan omissa tiloissa. [3]



Kuva 4: Sumupilven, pilvipalvelimen ja reunalaskennan sijoittuminen [3]

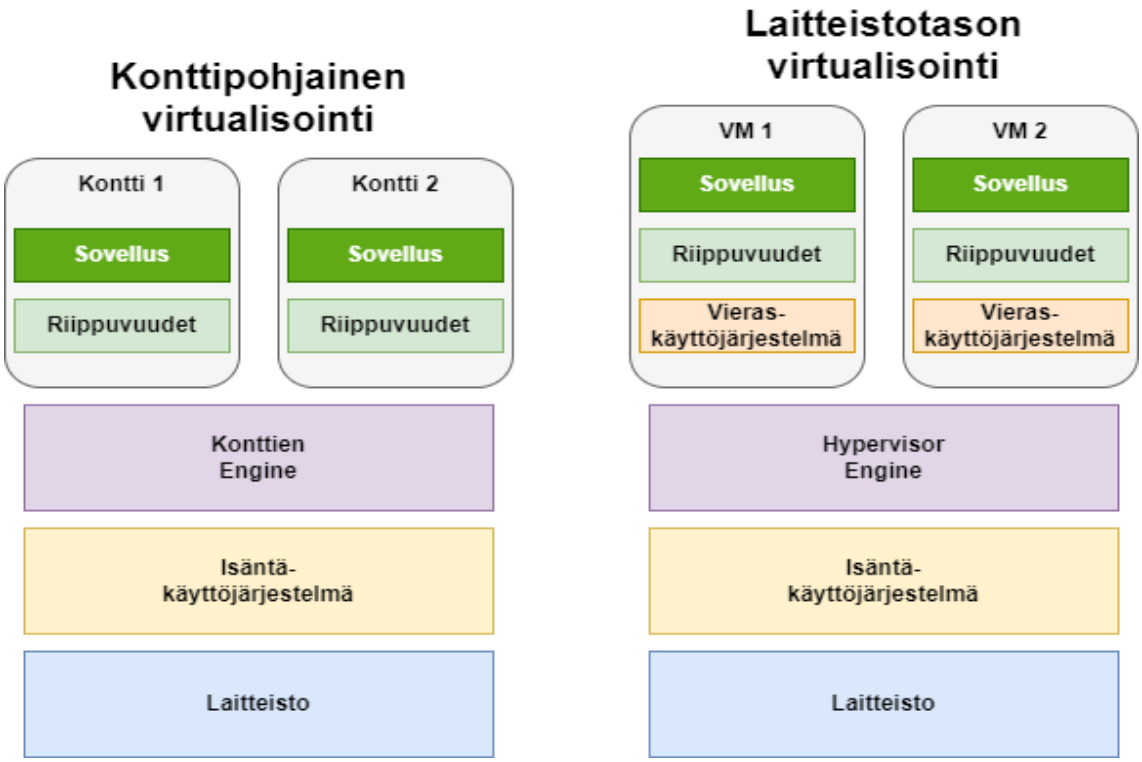
Kuvassa 4 on esitetty, miten sumupilviä voidaan käyttää pilvipalvelimien ja teollisuusverkon välissä. Teollisuus 4.0:n tavoitteena on tehokkaampi tuotanto ja älykkäämmät tehdasjärjestelmät, minkä mahdollistavana yhtenä paradigmana sumulaskenta toimii. Sumupilvet tarjoavat monipuolisuutta pilvi- ja reunalaskentaan, koska sumupilviä voidaan lisätä ja poistaa verkosta tarpeen mukaan sekä laskentatehtävien sijoittelulle on enemmän vaihtoehtoja. Pienissä datakeskuksissa ja palvelintietokoneilla pystytään suorittamaan kompleksisempaa koneoppimista kuin Edge-laitteilla, ja automaatiojärjestelmien tarpeisiin kyetään reagoimaan nopeammin kuin pilvestä käsin. [3,21]

2.4 Prosessiautomaation laskennan ja laskentatehtävien siirtäminen

Aliluvussa 2.4.1 käydään läpi teknologioita virtualisointi ja kontit, joita hyödynnetään koneoppimisprosessin kanavien hajauttamisessa eri laitteiden välillä. Eri MLOps-kanavat (eng. MLOps pipelines) voidaan siirtää suoritettavaksi pilvessä, reunalaskennan tai sumulaskennan laitteilla. Virtualisointiteknologiat mahdollistavat kokonaisten laskentatehtävien jakamisen eri laitteille eristetyissä virtuaaliympäristöissä. Aliluvussa 2.4.2 esitetään, kuinka laskentatehtäviä siirretään tietyssä laitteessa prosessorin eri ytimien välillä. Laskentatehtävien osien jakamista eri sumukerroksen laitteille ja osatehtävien tulosten yhdistämistä esitellään lyhyesti myös aliluvussa 2.4.2

2.4.1 Virtualisointi ja kontit

Virtualisointi on metodologia tai viitekehys, joka mahdollistaa tietokoneen laitteiston resurssien kuten prosessointitehon, muistin ja tallennustilan jakamisen monen suoritusympäristön käytettäväksi. Perinteisesti yhdellä tietokoneella toimii yksi sovellus, milloin osa resursseista jää hyödyntämättä. Virtualisointi mahdollistaa usean sovelluksen toimimisen samalla koneella eristetyissä virtuaaliympäristöissä, VM [24, s.9–13]. Virtualisointia voidaan suorittaa laitteistotasolla, jolloin tietokoneen isäntäkäyttöjärjestelmän päällä voidaan ajaa samanaikaisesti useampia eri vieraskäyttöjärjestelmiä. Vieraskäyttöjärjestelmät ovat paremmin yhteensopivia niillä käytettävien sovellusten kanssa. Virtuaaliympäristöjen toiminnan edellytys on, että kokonainen vieraskäyttöjärjestelmä on asennettu ja että sen ohjelmariippuvuudet ovat määritetty. Kuvassa 5 on esitetty oikealla laitteistotason virtualisointi ja vasemmalla on esitetty konttipohjainen tai käyttöjärjestelmätason virtualisointi. [20,43]



Kuva 5: Konttipohjainen ja laitteistotason virtualisointi [43]

Virtualisointialusta (eng. Hypervisor Engine) mahdollistaa laitteiston virtualisoinnin ja virtuaaliset laiteajurit. Hypervisor antaa virtuaalikoneille pääsyn yhteisiin isäntäkoneen resursseihin. Laitteistotason virtualisointi voidaan toteuttaa kuvassa 5 esitetyllä tavalla isännöitynä virtualisointialustana eli hypervisor sijaitsee isäntäkäyttöjärjestelmän päällä tai ”bare-metal” virtualisointialustana, jossa hypervisor sijaitsee suoraan isäntäkoneen laitteiston päällä, eikä isäntäkäyttöjärjestelmää ole. [43]

Toisaalta virtualisointia voidaan suorittaa käyttöjärjestelmän tasolla, mikä muodostaa abstraktin rajapinnan sovellusten ja käyttöjärjestelmän välillä. Tällaisella virtualisoinnin tasolla voidaan tehdä sovelluksia, jotka kapseloidaan sellaisiin kirjastojen ja prosessoriarkkitehtuurin riippuvuuksien määrittelyyn, että sovelluksen ajaminen lähes käyttöjärjestelmäriippumattomasti on mahdollista. Tällaista riippuvuuksiin kapseloitua sovellusta nimitetään *kontin kuvaksi*. Kun Kontin kuvaa ajetaan, siitä tulee silloin tarkalleen ottaen *kontti*. Kuitenkin tässä työssä käytetään väillä lyhennyksenä kontin kuvastakin termiä kontti. Konttitekniikat, kuten Docker, LXC (eng. Linux Container), Hyper-V ja niin edelleen, mahdollistavat nopean sovelluskehityksen, paremman tietoturvan sovellusten eriyttämisellä toisistaan ja joustavamman sovellusten käyttöönoton automaatio- tai reunalaskennanlaitteilla, jotka käyttävät ARM-pohjaista prosessoriarkkitehtuuria. Konttien Engine huolehtii resurssien jakamisesta konteille ja niiden ajamiseen liittyvistä toiminnoista: ajamisesta, sammuttamisesta ja poistamisesta ynnä muusta sellaisesta. [20,224,43]

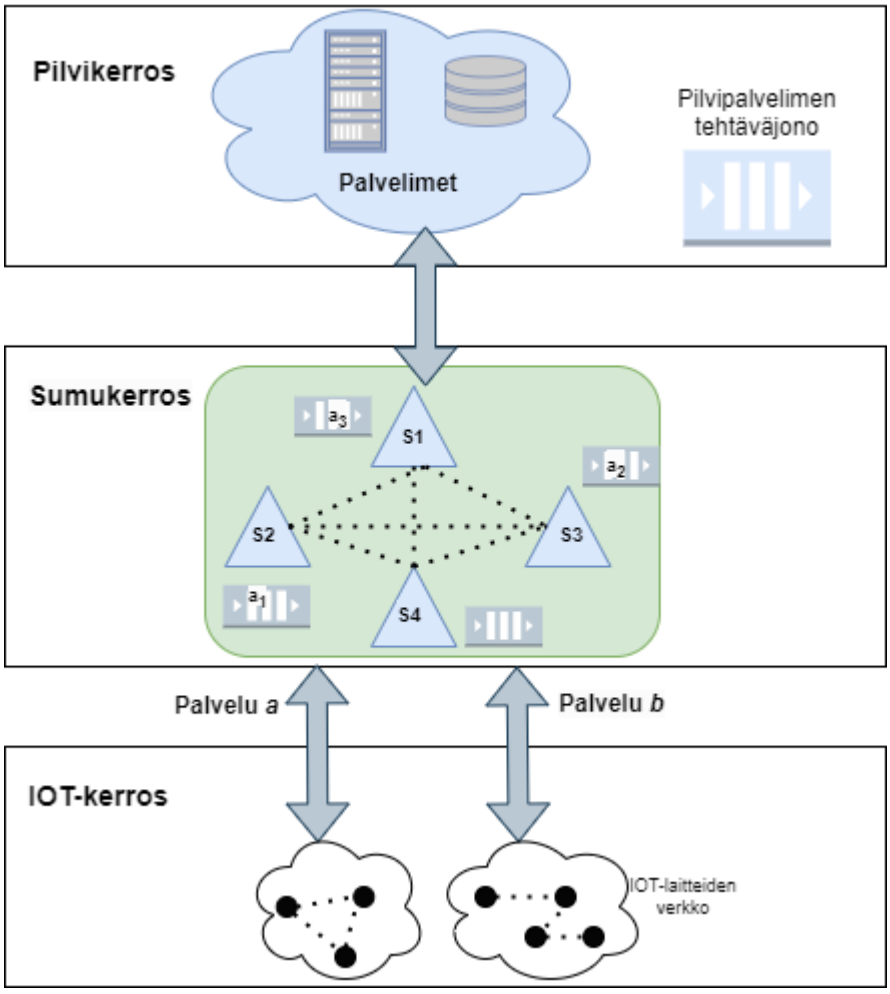
Kontin etu virtuaalikoneeseen verrattuna on pienempi käyttöjärjestelmäriippuvuus ja näin ollen kontin helpompi siirrettävyys laitteelta toiselle. Kontti ei tarvitse toimiakseen kokonaista omaa käyttöjärjestelmää. Tämän vuoksi kontti on kooltaan pienempi kuin virtuaalikone. Konttiin asennetaan käyttöjärjestelmän kuva, ohjelmakoodin kääntäjä ja tarvittavat kirjastot, joita sovellus toimiakseen vaatii. Lisäksi kontin kuva sisältää ohjeen, jonka mukaan konttien engine ajaa konttia. Edellä mainituista syistä kontti voidaan helposti siirtää laitteelta toiselle ajettavaksi, kunhan konttien virtualisointialusta on asennettuna. Konttien avulla voidaan siirtää MLOps:n määrittämiä automatisoituja kanavia eli datakanava, mallinnuskanava ja julkaisukanava, jollekin laitteelle suoritettavaksi pilvi-reunalaskenta jatkumossa.

2.4.2 Laskentatehtävien siirtäminen

Teollisuuden asioiden internet, IOT, tehostaa tuotantoa ja palveluja kytkemällä yhteen tuotannon prosesseja ja mahdollistamalla tehokkaan resurssienjakamisen. Koneoppimisen sovelluksia on laajasti käytetty erilaisissa yhteyksissä, jossa IOT on läsnä. IOT:n tuottamaa dataa voidaan analysoida koneoppimisen avulla esimerkiksi tuotteiden luokittelua, tuotteiden vikojen tarkastuksessa ja prosessien poikkeamien havaitsemisessa. Koneoppimisen sovellukset toteuttavat raskasta laskentaa vaativia laskentatehtäviä, joiden suorittaminen resurssirajoitteisilla laitteilla aiheuttaa korkeaa latenssia. Laskentatehtäviä voidaan toisaalta siirtää korkearesurssiseen pilvipalveluun, jossa laskentaresurssit eivät ole ongelma. Pilvipalvelussa toteutetussa koneoppimislaskennassa on kuitenkin ongelmana pilven etäisyyden vuoksi tehtävien siirrosta aiheutuva latenssi. Lisäksi IOT-laitteiden määrän suuri kasvu ja yhä tiukemmat QoS-vaatimukset hyvin pienestä viiveestä asettavat rajoitteita pilvilaskennalle. Pilvilaskennassa voi ilmetä IIOT-laitteiden määrän takia verkon ruuhkautumista ja liiallista viivettä, koska useilta IOT-laitteilta tietoa pitäisi siirtää keskitetysti pilveen. [10,46]

Ratkaisu pilvilaskennan ongelmiin on reuna- tai sumulaskennan hyödyntäminen. Reunalaskenta tarjoaa tehtävien prosessointipalveluja, joilla on suurempi laskentateho kuin IOT-laitteilla, ja jotka sijaitsevat lähempänä dataa tuottavia laitteita kuin pilvi. Reuna- ja pilvilaskenta yhdessä nopeuttavat tehtävien prosessointia sekä keventävät verkon ja pilvien kuormitusta. Toisaalta reuna- ja pilvilaskennan väliin asemoitu sumulaskenta vähentää myös pilvilaskennan kuormaa. Reunalaskentaan verrattuna sumulaskentalaitteilla on enemmän laskentaresursseja. Sumulaskentakerros koostuu heterogeenisestä tai homogeenisestä joukosta laitteita. Sumulaskennan lisääminen osaksi verkkoa nostaa esille kysymyksen, miten laskenta tulisi jakaa reuna-, sumu- ja pilvilaskennan välillä ja lisäksi miten laskenta tulisi jakaa heterogeenisessä sumukerroksessa. [10,46]

Kuvassa 6 on esitetty IOT- ja pilvikerroksen väliin sijoittuva sumukerros. Sumukerroksessa laitteet ovat heterogeenisiä ja jotkin laitteet voivat olla laskentakapasiteetiltaan rajoittuneita, mutta yleensä rajoittuneet laitteet sijaitsevat IOT-kerroksessa. Näin ollen laskennan jakaminen vain sumulaitteelle, jonka tallennustila, verkon luotettavuus ja prosessointiteho ovat korkeimmat voi johtaa eri palvelupyyntöjen jonojen ruuhkautumiseen. Pitkän jonotusajan vuoksi latenssiherkkien sovellusten reaaliaikavaatimukset eivät toteudu. [10,46]



Kuva 6: Sumulaskennan kolmikerrosmalli [10]

Sumukerroksen arkkitehtuurin rakenne voi olla kolmenlaista: keskitettyä, hajautettua ja hierarkkista. Keskitetyssä arkkitehtuurissa sumukerroksessa on yksi ohjain ja muita sumusolmuja. Ohjain vastaa tiedon yhdistämisestä ja päättää mihin sumukerroksen osaan laskentatehtäviä siirretään. Muut solmut suorittavat ohjaimen jakamia laskentatehtäviä. Keskitetyssä arkkitehtuurissa ohjaimella on myös resurssitaulukko, josta käyvät ilmi avustavien solmujen muisti, kellotaajuus, prosessorityyppi, Round-Trip-Time ja odotettu jonotusaika. Resurssitaulukon ja avustavien laitteiden kuormituksen perusteella ohjain

siirtää laskentatehtäviä vähäkuormaisille sumusolmuille. Hajautetussa arkkitehtuurissa taas sumukerroksen solmut päättävät yhdessä laskentatehtävien siirtämisestä. Hajautetussa arkkitehtuurissa jokaisella solmulla on naapurisolmujensa vapaana olevat resurssit taulukoissa. Hierarkkisessa arkkitehtuurissa on klustereita, joita ohjaava isäntäsolmu jakaa laskentatehtäviä vastaanottaville solmuille. Hierarkkisessa arkkitehtuurissa isäntäsolmuilla on vastaanottavien solmujen resurssitaulukot. [10]

Jokaista laskentatehtävää *k* voidaan kuvata monikolla (A_k,B_k), jossa vektori A_k ilmaisee laskentatehtävän koon bitteinä, sen jaettavuuden osiin sekä datatyyppien määrän. Vektori B_k ilmaisee paljonko, laskentaresursseja tarvitaan tehtävän suorittamiseksi. Laskentatehtävän koon perusteella voidaan määrittää, onko laskenta kevyt, keskiraskas vai raskas tehtävä. Tällöin voidaan jakaa tehtäviä laitteille, jotka kykenevät resurssien puolesta laskennan suorittamaan. Laskennan jaettavuudella tarkoitetaan, voidaanko jokin laskentatehtävä jakaa osakokonaisuuksiin vai ei. Pienempiin paloihin jaetun laskentatehtävän osat voidaan suorittaa samanaikaisesti eri sumulaskennan laitteilla. Esimerkiksi laskentatehtävän *k* osat {a_1, a_2, a_3} voidaan jakaa eri laitteille suoritettavaksi, jonka jälkeen laskennan osatulokset siirretään jollekin laitteelle, jossa osat liitetään yhteen kokonaisuudeksi *f(a)*. Tällaista laskentatehtävien jakamista kutsutaan *laskennan osittamiseksi.* Joissakin tutkimuksissa on tarkasteltu resurssina pelkästään vaadittua CPU:n prosessointitehoa. Monimutkaisemmissa ja raskaammissa koneoppimisen algoritmeissa on tarkasteltu GPU-tehon ja muistin riittävyyttä. [10]

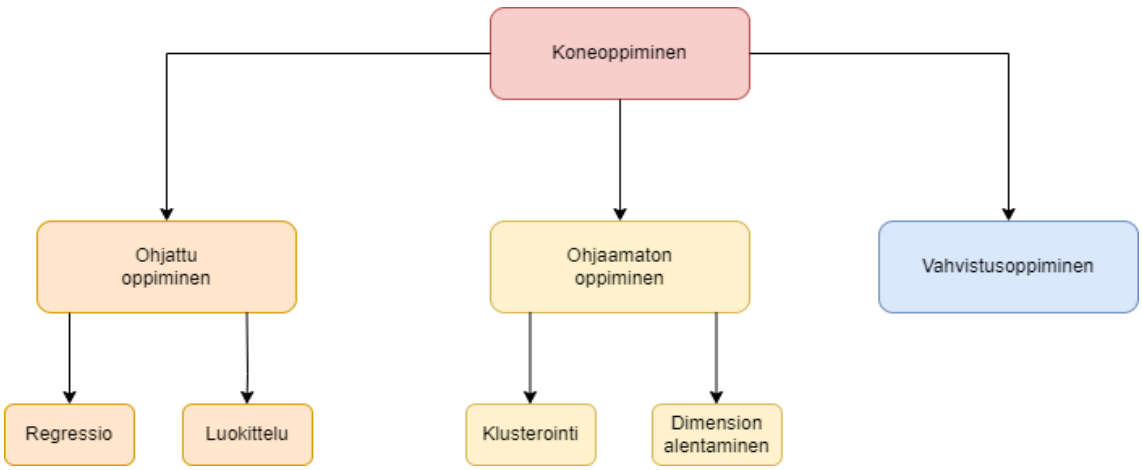
Laskentatehtävien siirtämisessä vaihtoehdot ovat moninaiset kuten edellisessä kappaleessa esitetty laskennan osittaminen osoittaa. Sumukerroksen sisällä tehtävien osia, mutta myös kokonaisia tehtäviä voidaan siirtää laitteelta toiselle resurssitarpeen mukaan. Näiden päälle vaihtoehtoina on suorittaa laskentatehtäviä paikallisesti IOT-laitteissa, siirtää niitä muiden kerrosten kautta tai suoraan pilvipalvelimille. Kun laskentainfrastruktuuri on monipuolista, optimaalinen laskentatehtävien siirtäminen ja osittaminen parantaa systeemin läpäisykykyä, suoritustehoa, vasteaikaa ja resurssien käyttöä. QoS-parametreihin vaikuttaa millä laitteilla erilaisia laskentatehtäviä suoritetaan, mutta myös se mitä reittejä laskentatehtäviä ja niiden käyttämää data siirretään. Tehtävien jakamisen päämääränä on mahdollisimman minimoitu energiankulutus, viive ja kaistanleveyden käyttö. Erilaisien reittien kautta siirrettävän tiedon kulkuun liittyvä viive yhdessä itse laskennan aiheuttaman viiveen kanssa asettaa rajoitteita missä laskenta kannattaa suorittaa. [46]

2.5 Koneoppimisen malleihin perustuva laskenta

Koneoppimisen malleilla ratkaistaan erilaisia ongelmia, joissa tavoiteltu ratkaisu on tiedossa tai datasta halutaan etsiä säännönmukaisuuksia. Koneoppimisen mallin muodostamiseen ja käyttöönottoon liittyy tietty prosessi, jonka tehostamiseen MLOps tarjoaa puitteet. Aliluvussa 2.5.1 esitetään, mitä koneoppiminen tarkoittaa ja minkä tyyppisiä malleja koneoppimiseen liittyy. Lisäksi samassa aliluvussa esitetään syvien neuroverkkojen toiminta periaate ja tutustutaan kuvantunnistuksen kannalta oleellisiin malleihin, Konvoluutionaalisiin neuroverkkoihin. Aliluvussa 2.5.2 käydään läpi tavanomainen koneoppimisen mallin muodostus- ja sen käyttöönottoprosessi. Tämän lisäksi esitetään, miten koneoppimisprosessia voidaan tehostaa käyttämällä MLOps:ia ja lopuksi esitellään jatkuvan oppimisen tuoreempia tutkimuksia.

2.5.1 Koneoppimisen malleja ja neuroverkot

Koneoppimisen perusajatuksena on kerätä yksi tai useampi suuri joukko datapisteitä, joiden avulla harjoitetaan oppiva kone suorittamaan jotakin tehtävää. Koneoppiminen tarvitsee yleisesti ottaen tavoitteen, yleisen mallin ja optimointitekniikan, mutta käytetyt algoritmit ja ratkaisut valitaan datan perusteella. Esimerkiksi koneoppimisen avulla voidaan tunnistaa jollain todennäköisyydellä kuvassa olevia esineitä tai asioita. Datapistejoukkojen lisäksi toinen koneoppimisen termi on ominaispiirre (eng. feature), joka kuvaa datapisteen jotain ominaisuutta. Piirteet voivat olla kategorisia, ordinaalisia tai numeerisia [16]. Kategorinen piirre tarkoittaa ennalta määriteltyjä arvoja, joilla ei ole järjestystä, kuten ikä tai pituus. Ordinaalinen piirre kuvaa ennalta määriteltyjä arvoja, joilla on jokin järjestys. Ordinaalisesta piirteet voidaan laittaa määrän tai suuruuden mukaan järjestykseen. Numeerinen piirre kuvaa vaikkapa reaalilukuna kuvan pikselin värisävyä. Datassa piirteitä voi olla useampia, jolloin piirreavaruus on moniulotteinen. [16,55]



Kuva 7: Koneoppimisen taksonomia [16]

Koneoppimisessa voidaan tunnistaa kolme erilaista ongelmapäätyyppiä: ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen. Kuvassa 7 on esitetty koneoppimisen ongelmapäätyyppien luokittelu ja joitakin näiden päätyyppien alalajeja. Ohjatussa oppimisessa tietokoneen tehtävänä on oppia ennustamaan mallin perusteella entuudestaan tuntemattoman datapisteen luokka tai sen arvo. Toisin sanoen ohjatussa oppimisessa on kyse siitä, että ohjaaja antaa koneelle harjoitusdatan, jossa jokaiselle pisteelle x_n on määritetty jokin ulostulo tai luokka y_n. Opetus tapahtuu niin, että kone oppii muodostamaan kuvauksen syötteiden x_n ja ulostulojen y_n välille. Koneen oppimista ohjataan antamalla sille siis päämäärä, johon oppimisessa tulisi pyrkiä. Ohjatusta oppimisesta on tunnistettavissa kaksi eri ongelmatyyppiä, jotka ovat *luokittelu-* ja *regressio-ongelmat*. Luokitteluongelman tavoitteena on ennustaa harjoitusdataan kuulumattomalle datapisteelle x_m diskreetti arvo y_m. Esimerkiksi voidaan ennustaa, millä todennäköisyydellä uusi datapiste kuuluu johonkin ennalta määritellyistä luokista yksi, kaksi, tai kolme. Datapisteen luokaksi valitaan todennäköisin. Regressio-ongelmassa taas datapisteelle x_m ennustetaan jokin jatkuva lukuarvo y_m opitun mallin perusteella. Esimerkiksi voidaan muodostaa sovite harjoitusdatan pistejoukkoon, ja sovitteen avulla voidaan ennustaa, mikä on syötteen x_m ulostulo y_m. [7,40,55]

Ohjaamattomassa oppimisessa ei tiedetä, mihin luokkaan mikäkin opetusdatan piste kuuluu. Tarkoituksena on tunnistaa datassa esiintyviä hahmoja tai säännönmukaisuuksia ja jakaa yksittäisiä datapisteitä luokkiin. Ohjaamattomassa oppimisessa ei siis anneta koneelle valmiiksi oppimisen päämäärää, vaan kone oppii sen itse. Ohjaamattoman oppimisen alalajeja ovat *klusterointi* ja *dimension alentaminen*. Klusteroinnissa datajoukon pisteitä jaetaan ryhmiin tiettyjen piirteiden tai samankaltaisuuden perusteella. Toinen ohjaamattoman oppimisen alalaji on dimension alentaminen. Datajoukon pisteet voivat sisältää hyvin monia piirteitä ja näin ollen piirreavaruus voi olla liian moniulotteinen visualisoitavaksi tai analyysin tekeminen voi olla liian vaikeaa. Tällöin korkeadimensioinen data voidaan yksinkertaistaa matalampiulotteiseksi dataksi, josta on poistettu epäoleelliset piirteet. Dimension alentamisen jälkeen käsitelty data säilyttää suurimman osan alkuperäisen datan tärkeistä ominaisuuksista. [16,40,55]

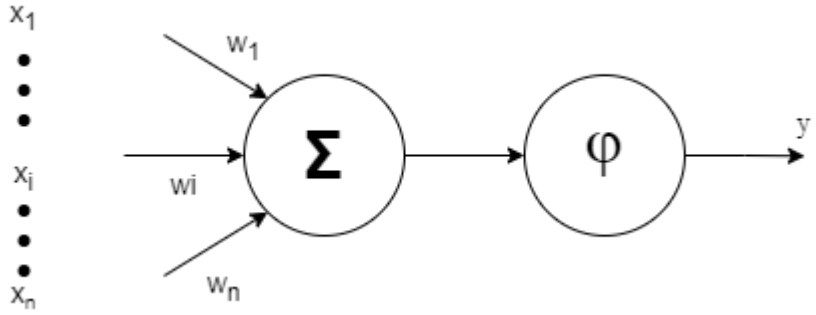
Vahvistusoppimisen ideana on oppia ”optimaalinen” peräkkäisten päätösten muodostama sarja. Päätöstentekoa ohjaa aiempien päätösten perusteella saatavat palkkiot ja rangaistukset. Vahvistusoppimisessa koulutetaan algoritmilla kone suorittamaan tehtävää tilanteessa, jossa päätösten ketjun muodostamiseksi ei ole yhtä oikeaa vastausta. Esimerkiksi toimija voidaan opettaa liikkumaan määrätyssä ympäristössä yrityksen ja erehdyksen kautta. Tällaisessa ympäristössä voi olla erinäinen määrä esteitä, joihin törmäämisestä saa rangaistuksen, ja kulkemalla määrättyyn suuntaan tai lopputilaan saa

tietynsuuruisen palkkion riippuen päätöksen hyödyllisyydestä. Treenaamalla useita kertoja liikkumista ympäristössä toimija oppii, millainen käyttäytyminen on tavoiteltavaa. Vahvistusoppiminen ei ole ohjattua oppimista, sillä oppivalle koneelle ei ole määrätty optimaalisia tapoja toimia tietyssä tilanteessa. Vahvistusoppimisessa oppija saa palautetta palkkioiden ja rangaistusten muodossa, joten kyse ei ole myöskään ohjaamattomasta oppimisesta. [16,40,55]

Neuroverkko, (A)NN, (eng. (Artificial) Neural Network) on koneoppimisen malli, jota voidaan soveltaa sekä ohjatun että ohjaamattoman oppimisen ongelmiin. Neuroverkkoja käytetään muuan muassa piirteiden erottelussa, luokittelualgoritmeissa, kohteentunnistuksessa ja järjestelmien ohjauksessa. Neuroverkko koostuu yksittäisistä neuroneista, jotka ovat kytköksissä muihin neuroneihin. Yksittäisen neuronin sisääntulona on useampia syötesignaaleja x_i, joista jokaisella on oma painokertoimensa w_i. Neuronin ulostulo y on syötteiden painotetun summan funktio, joka esitetään kaavassa 1

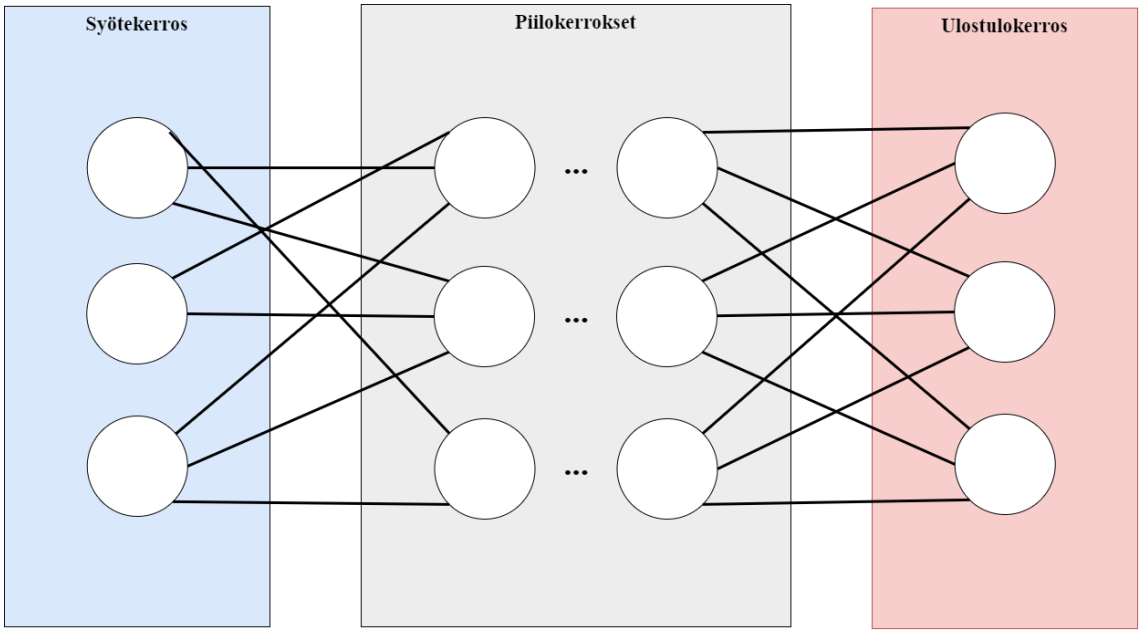
$$y = \varphi\left(\sum_{i=1}^{m} w_i\, x_i\right) \tag{1}$$

ja jossa φ on aktivointifunktio. Yleisesti käytettyjä aktivointifunktioita ovat hyperbolinen tangentti, sigmoid ja ReLU -funktiot, joilla voidaan ratkaista myös epälineaarisia luokitteluongelmia. ReLU on edellä mainituista aktivointifunktioista eniten käytetty, sillä se ei ole yhtä herkkä satunnaisvaihteluille kuin muut vastineet. Kuvassa 8 esitetään yksittäinen neuroni, jossa indeksi i = 1, 2…, n. [35]



Kuva 8: Neuroni [35]

Neuroverkko koostuu syötekerroksesta, piilokerroksesta tai -kerroksista ja ulostulokerroksesta. Jokaisessa kerroksessa on jokin määrä neuroneita rinnakkain ja usein edellisen kerroksen jokainen neuroni on kytketty seuraavan kerroksen neuroneihin, tällöin on kyse *täysin kytketystä* neuroverkosta. Neuroverkko voi olla myös *harva* eli kaikkia edellisen kerroksen neuroneja ei ole kytketty kaikkiin seuraavan kerroksen neuroneihin. Piilokerros voi koostua useammista kerroksista neuroneita ja tällöin puhutaan syvästä neuroverkosta tai syväoppimisesta. Syväoppimisessa on kuitenkin ongelmana *ylisovittaminen*, jossa neuroverkon oppimisen tarkkuus opetusdatalla on hyvä ja testidatalla huono. Tällöin mallin kyky tehdä yleistää oppimaansa kykyä, esimerkiksi tunnistaa kohteita kuvasta, uuteen mutta samankaltaiseen dataan. Ylisovittaminen voi johtua siitä, että opetusdataa on vähemmän kuin testidataa, opetusnäytteiden liiallisesta kohinaisuudesta tai opetuskertoja on liian monta. Syväoppiminen sopii suuren datamäärän käsittelyyn, koska oppimisen suorituskyky kasvaa sitä enemmän, mitä enemmän dataa on. [33,35]



Kuva 9: Syvä neuroverkko [35]

Kuvassa 9 on esitetty syvä neuroverkko, jossa on jokaisessa kerroksessa kolme neuronia rinnakkain [35]. Syvän neuroverkon kerrosten neuronien lukumäärä vaihtelee käytettävän datan mukaan. Esimerkiksi kuvien luokittelussa, jossa kuvan koko on 32 x 32 pikseliä, syötekerroksessa on 1024 neuronia. Syötekerros on kytketty piilokerrokseen, jossa kunkin kerroksen neuronien lukumäärä voidaan periaatteessa valita. Ulostulokerroksen neuronien lukumäärä kuvien luokittelussa on luokkien määrä. Jos halutaan luokitella kuvia kolmeen luokkaan, ulostulokerroksessa on kolme neuronia.

Syväoppimisen algoritmeihin kuuluu *konvoluutionaalinen neuroverkko, CNN*, joka hyödyntää laskennassa konvoluutiota, käyttää jaettuja painokertoimia neuronien välillä ja poolauskerroksia. Syöte- ja ulostulokerroksen lisäksi CNN koostuu konvoluutio-, poolaus- ja täysin kytketystä kerroksesta. Konvoluutionaalinen neuroverkko soveltaa harvaa neuroverkkoa siten, että johonkin neuronien osajoukkoon kytketään jotkin kuvan alueen pikselit ja toiseen osajoukkoon toisen alueen pikselit. Jokainen osajoukko jakaa toisen osajoukon painokertoimet. Poolaus tarkoittaa, että syötedatan kokoa pienennetään esimerkiksi valitsemalla kuvan 2 x 2 alueen pikselien sävyn keskiarvo tai maksimiarvo, jolloin 2 x 2 alueen piirteet tiivistyvät yhteen lukuarvoon. CNN:n harvuudella, painokertoimien jakamisella ja poolauksella voidaan pienentää suuriresoluutioisen kuvan tunnistamiseen tarvittavien parametrien määrää. Konvoluutionaalisia neuroverkkoja käytetäänkin paljon kuvientunnistamis- ja luokitteluongelmissa. [35]

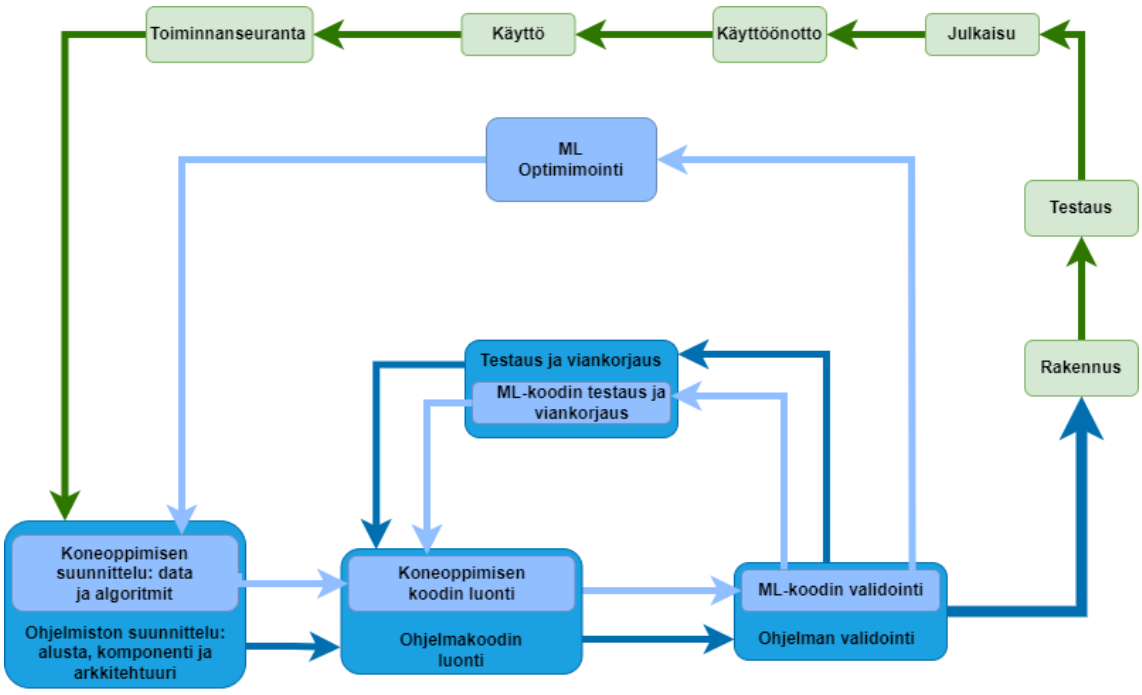
2.5.2 Koneoppimisprosessi ja MLOps

Koneoppiminen prosessina lähtee liikkeelle **raakadatan keräämisestä ja esikäsittelystä**. Raakadata pitää yleensä esikäsitellä siten, että raakadatasta poistetaan virheelliset arvot ja suodatetaan pois kohina. Vasta esikäsitelty data soveltuu mallin opetukseen. Seuraavaksi esikäsiteltyä **dataa analysoidaan**. Millainen on datan rakenne tai miten data on jakautunut? Analyysissä myös tunnistetaan datasta mahdolliset trendit, bias ja rajataan pois ongelman kannalta turhat piirteet. Sitten **dataa prosessoidaan treenausta varten**. Voi ilmetä tarve skaalata datapisteiden arvoja tietylle arvovälille, datasta voi joutua poistamaan poikkeavia datapisteitä (eng. outliers) tai opetusta varten voidaan määrittää datan konteksti. Lisäksi tässä vaiheessa data jaetaan opetus-, testaus- ja validointidataan. Sitten **luodaan, treenataan ja testaan malli**. Tässä vaiheessa luodaan malli, jolle määritetään hyperparametrit ja testataan opetetun mallin suorituskykyä testidatalla. Tämän jälkeen **validoidaan ja säädetään mallia.** Validoinnissa mallin tarkkuutta validoidaan entuudestaan tuntemattomalla datalla, säädetään mallin hyperparametrejä eli muun muassa oppimisnopeutta, opetuskausien määrää tai aktivointifunktiota. Edellisen ohella validoinnissa muutetaan tarvittaessa malli tai sen arkkitehtuuria, ja palataan taaksepäin uudelleen opettamaan mallia, mikäli se ei ole riittävän hyvä. Testausvaiheessa tutkitaan datan ylisovitusta, kun taas validoinnissa enemmänkin hienosäädetään mallia. Muussa tapauksessa edetään **mallin käyttöönotto ja monitorointi** -vaiheeseen. Tässä vaiheessa integroidaan malli osaksi sovellusta, mallin toimintaa seurataan ja kerätään tietoa mallin päivittämistä varten. Alla on listattu vielä koneoppimisprosessin vaiheet. [2]

1) Raakadatan kerääminen ja esikäsittely
2) Data-analyysi

3) Datan prosessointi treenausta varten
4) Mallin luonti, treenaus ja testaus
5) Mallin validointi ja hienosäätö
6) Mallin käyttöönotto ja monitorointi

Ohjelmistokehityksessä DevOps-käytännöt (eng. *Development* ja *Operations)* tuovat yhteen kehittäjä- ja operatiivisen toiminnan tiimejä sekä nopeuttaa tuotekehitystä että tekee siitä joustavampaa. [11] DevOps yhdistää tuotekehityksen, laadunvarmistuksen ja operatiivisen toiminnan yhdeksi jatkuvaksi automatisoiduksi prosessiksi. Jatkuva prosessi sisältää suunnittelun, ohjelmistokehityksen, testauksen, julkaisun ja tuotteen toiminnanseurauksen. Joustavassa ohjelmistokehityksessä jatkuvasti kehitetään (CD, engl. Continuous Delivery) tuotetta, jatkuvasti integroidaan ohjelmistokomponentit osaksi suurempaa ohjelmistoa (CI, eng. Continuous Integration) ja testattujen ohjelmistojen jatkuvan viennin tuotantoon (CD, eng. Continuous Delivery) [4,11] ja tuotannossa ohjelmiston toiminnasta kerätään tietoa, jonka perusteella ohjelmistoa kehitetään paremmaksi. Koneoppimissovelluksen kehittämiselle on omanlaisia vaatimuksia, jotka osittain poikkeavat perinteisen ohjelmistokehityksen vaatimuksista [37]. MLOps (eng. machine learning ja operations) on DevOps:n laajennus [37], jossa koneoppimisprosessi tuodaan osaksi ohjelmistokehityksen jatkuvaa prosessia. MLOps:ssa lisätään jatkuvan kehittämisen takaisinkytkettyyn kanavaan (eng. pipeline) koneoppimisen mallien jatkuva treenaaminen (CT, eng. Continuous Training) [37].



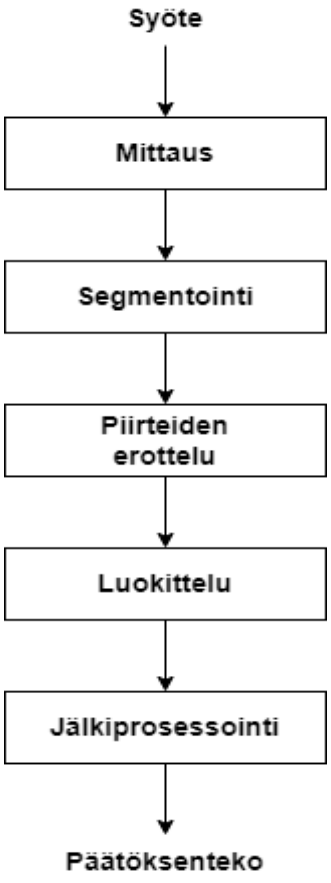
Kuva 10: MLOps-kanava [37]

Koneoppimisen malli tai sitä hyödyntävä sovellus usein rakennetaan rinnakkain sitä käyttävän ohjelmiston kanssa ja tällaisen koneoppimissovellus integroidaan osaksi isompaa ohjelmistoa. Kuvassa 10 on esitetty MLOps-käytäntöjä havainnollistava kuva, jossa on tumman sinisellä esitetty ohjelmistosuunnittelun MLOps-vaiheita, kun taas vaalean sinisellä on esitetty koneoppimissovelluskehityksen vaiheita ja vihreällä on ilmaistu jatkuvan prosessin vaiheet, jotka liittyvät koneoppimisohjelmistokokonaisuuden viemisen tuotantoon. Koneoppimisprosessi alkaa datan keräämisellä, prosessoinnilla ja analysoinnilla, jonka jälkeen luodaan koneoppimisen mallin ja sen hyperparametrit. Kuitenkin koneoppimissovelluksen tuottamisen näkökulma on ennen mallin luontia tärkeää tunnistaa ratkaistava ongelma ja se, millaista dataa on käytettävissä [37]. Ratkaistava ongelma ja käytössä oleva data määrittävät sen millaisella mallilla ja algoritmilla ongelma kannattaa yrittää ratkaista.

Kuvassa 10 tämä vaihe on kuvattu vasemmassa alareunassa. Samassa vaiheessa myös määritetään muun ohjelmiston alusta, komponentit ja arkkitehtuuri. Kun ongelma ja sen ratkaisua tukevat osat ovat määritetty, edetään seuraavaan vaiheeseen, jossa koneoppimisen ja muun ohjelmiston ohjelmakoodi luodaan ja tarvittaessa virheet ohjelmakoodista korjataan tai koodia muutetaan. Sitten validointivaiheessa testataan ohjelmaa ja toteutetun koneoppimismallin suorituskykyä ja jos malli tai algoritmi ei ratkaise ongelmaa riittävän hyvin, niin palataan uudelleen suunnitteluvaiheeseen tutkimaan muita vaihtoehtoja tai optimoimaan nykyistä mallia. Validoinnin jälkeen sekä ohjelmisto, että koneoppimisen osuus integroidaan yhteen ja rakennetaan ohjelmistokokonaisuus. Tämän jälkeen ohjelmisto testataan, julkaistaan, käyttöönotetaan ja käytössä olevan ohjelmiston suorituskykyä seurataan, jos ilmenee tarvetta kehittää tai päivittää ohjelmisto, aloitetaan uudelleen suunnitteluvaiheesta kerätyn tiedon perusteella luoda uutta ohjelmistoa. [37]

2.6 Kuvantunnistus prosessiympäristössä

Prosessiteollisuudessa konenäön avulla voidaan kerätä suuri määrä dataa, jota pystytään hyödyntämään prosessien valvonnassa ja hallinnassa. Konenäkö käsittää automaattisen kuvien hankinnan, prosessoinnin ja analysoinnin. Hahmontunnistuksen metodeja käytetään konenäköjärjestelmissä tunnistamaan ja analysoimaan kuvien sisältöä. Hahmontunnistuksen tarkoitus on määrätä kohteille luokat samankaltaisten ominaisuuksien perusteella. Hahmotunnistuksen ja luokittelun prosessi, kuvassa 11, alkaa syötteiden mittaamisella eli kuvien hankinnalla. Toinen vaihe on kuvien segmentointi, jossa syötekuvasta eristetään oleellinen alue. Kuvasta poistetaan kohteet ja tausta, mikäli ne eivät ole oleellisia hahmontunnistuksen kannalta. Segmentoinnin lisäksi kuvista voidaan suodattaa kohinaa ja näin niiden laatua voidaan parantaa. Kolmas vaihe on piirteiden erottaminen, jossa määritetään kuvat tai niissä olevat kategoriat toisistaan erottavat piirteet. Piirrevektorit tulevat olla sellaisia, että kohteiden samankaltaisuus ja erilaisuus tulee esille. Neljäs vaihe on luokittelu, jossa tuntematon kuva tai siinä oleva hahmo liitetään johonkin kategoriaan. Lopuksi jälkiprosessointivaiheessa luokittelun tuloksen oikeellisuutta arvioidaan soveltamalla jotain validointitapaa. [45,52]



Kuva 11: Hahmontunnistusprosessi [45]

Kuvantunnistukseen liittyy kuvien luokittelu ja esineiden tai hahmojen tunnistaminen kuvasta. Kuvien luokittelussa kone opetetaan harjoituskuvien avulla tunnistamaan koneelle tuntematon kuva. Kuvien luokittelun eräs toteutustapa on hyödyntää ohjattua oppimista, jossa jokaiselle harjoituskuvalle on ennalta määrätty lukuarvo, joka ilmaisee mihin luokkaan kyseinen kuva kuuluu. Kone oppii muodostamaan yhteyden tai funktion kuvien ja niiden luokkien välille. Harjoitetulle koneoppimisen malille tuodaan harjoitusdatassa esiintymätön uusi kuva, joka halutaan luokitella johonkin ennalta määrätyistä luokista. Tuntemattomalle kuvalle määritetty luokka ei ole eksaktin tarkka, vaan luokitteluun liittyy todennäköisyys, jolla kuva kuuluu johonkin luokkaan. Luokittelussa koneelle tuntemattomalle kuvalle määritetään joukko todennäköisyyksiä, joilla kuva kuuluu kuhunkin ennalta määrättyyn luokkaan. Mihin luokkaan kuva näistä ennalta määrätyistä luokista luokitellaan, on se, jolla on suurin todennäköisyys. Toisaalta kuvien luokittelussa voidaan myös hyödyntää ohjaamatonta oppimista, jossa kuvien luokkia ole ennalta määrätty, vaan kuvia luokitellaan samoihin luokkiin niiden samankaltaisuuden, kuten rakenteen ja yhteisen sisällön perusteella. [45]

Kohteentunnistuksessa tunnistetaan kuvasta esineitä tai kohteita (eng. object detection). Kohteentunnistuksessa yritetään arvioida kohteen, kuten jalankulkijan tai kasvojen sijaintia eri kuvissa. Tämän jälkeen arvioidaan, mihin luokkaan kuvasta paikannettu kohde kuuluu. Kuvassa alueelle, jossa kohteen ajatellaan olevan, määritetään kohdetta rajaava suorakaide. Kohteentunnistus on haastavaa, sillä kohde voi olla kuvassa eri katselukulmissa, asennoissa sekä kuvan valaistuskin voi vaihdella. Kohteentunnistus voidaan jakaa kolmeen osaan, jotka muistuttavat kuvan 11 hahmontunnistusprosessin jotain vaiheita. Kolme osaa ovat informatiivisen alueen valinta, piirteiden erottelu ja luokittelu. Eri kuvissa kohteiden kuvasuhteet voivat olla erilaisia, ja niiden koko voi vaihdella, lisäksi kohteet voivat olla missä tahansa asennossa, siksi kohteentunnistuksessa käytetään moniulotteista liukuikkunaa. Piirteiden erottelussa määritetään kohteelle ominaiset piirteet. Johtuen kuvien valaistuksesta ja eri taustoista, kohteita hyvin kuvaavien piirteiden määrittely käsin voi olla vaikeaa. Luokittelussa kohteet erotellaan kategorioihin, jotka lisäävät tietoa kuvan hierarkkisuudesta ja merkityssisällöstä. Kuvan luokittelu ja kohteentunnistus ilman luokittelua ovat esitetty kuvassa 12. [52,56]

Kuva 12: Luokittelu ja kohteentunnistus [52]

Kohteentunnistukseen on kehitetty erilaisia Konvoluutionaalisiin neuroverkkoihin pohjautuvia ratkaisuja, koska niiden mahdollistaman syväoppimisen avulla pystytään oppimaan monimutkaisempia piirteitä kuin perinteisillä, esimerkiksi, tukivektorikonemenetelmillä (eng. Support Vector Machine). SVM-menetelmässä datapisteparvelle tehdään sovite, kuten suora, kahden eri luokan pisteiden välille. Suoran tapauksessa sovitteella on kaksi suoran kanssa samansuuntaista marginaalia tai ikään kuin toleranssirajat. Sovite on asetettu kahden eri luokan pisteiden välille niin, ettei yhtään kummankaan eri luokan datapisteistä ole toleranssirajojen sisällä ja että toleranssirajat ovat mahdollisimman etäällä. Näitä toleranssirajoja ilmaisevia suoria kutsutaan tukivektoreiksi. Toinen etu perinteisiin menetelmiin verrattuna on CNN:n robustit oppimisalgoritmit. Konvoluutionaalisiin neuroverkkoihin pohjautuvia valmiita koneoppimisen malleja on kehitetty, esimerkiksi VGG11, YOLO tai muita, joiden edut vaihtelevat yli 90 prosentin tunnistustarkkuuden tai pienemmän tarkkuuden, mutta paremman reaaliaikaisen tunnistuksen välillä. [52,56]

Kuvantunnistuksessa segmentoinnilla voidaan tarkoittaa kuvasta kiinnostavan alueen erottamista ja toisaalta tarkoittaa kuvantunnistuksen tehtävää, jossa kuvan jokaiselle pikselille määritetään luokat, joihin kyseiset pikselit kuuluvat. Määritetyt luokat erottelevat kohteiden reunat tai niiden taustasta tai kohteessa ilmeneviä pikselejä toisistaan. Kuvien segmentointia pikseleittäin käytetään kohteentunnistuksessa, mutta myös kohteen muodon tai sen lämpötilajakauman määrittämisessä. Segmentoinnin perusteena käytetään kuvan yhtä tai useampaa ominaisuutta, kuten intensiteetti, heijastavuus tai kuvan RGB-kanavan punavihersinisävyjä. Kuvan segmentoinnin menetelmiä ovat muun muassa ryvästysmenetelmä (eng. k-means clustering), jossa datapisteet jaetaan k eri klusteriin datapisteiden etäisyyden kunkin klusterin keskuksesta perusteella, kohteen reunantunnistus ja neuroverkkopohjaiset menetelmät. Kuitenkin syvään neuroverkkoon pohjautuvat

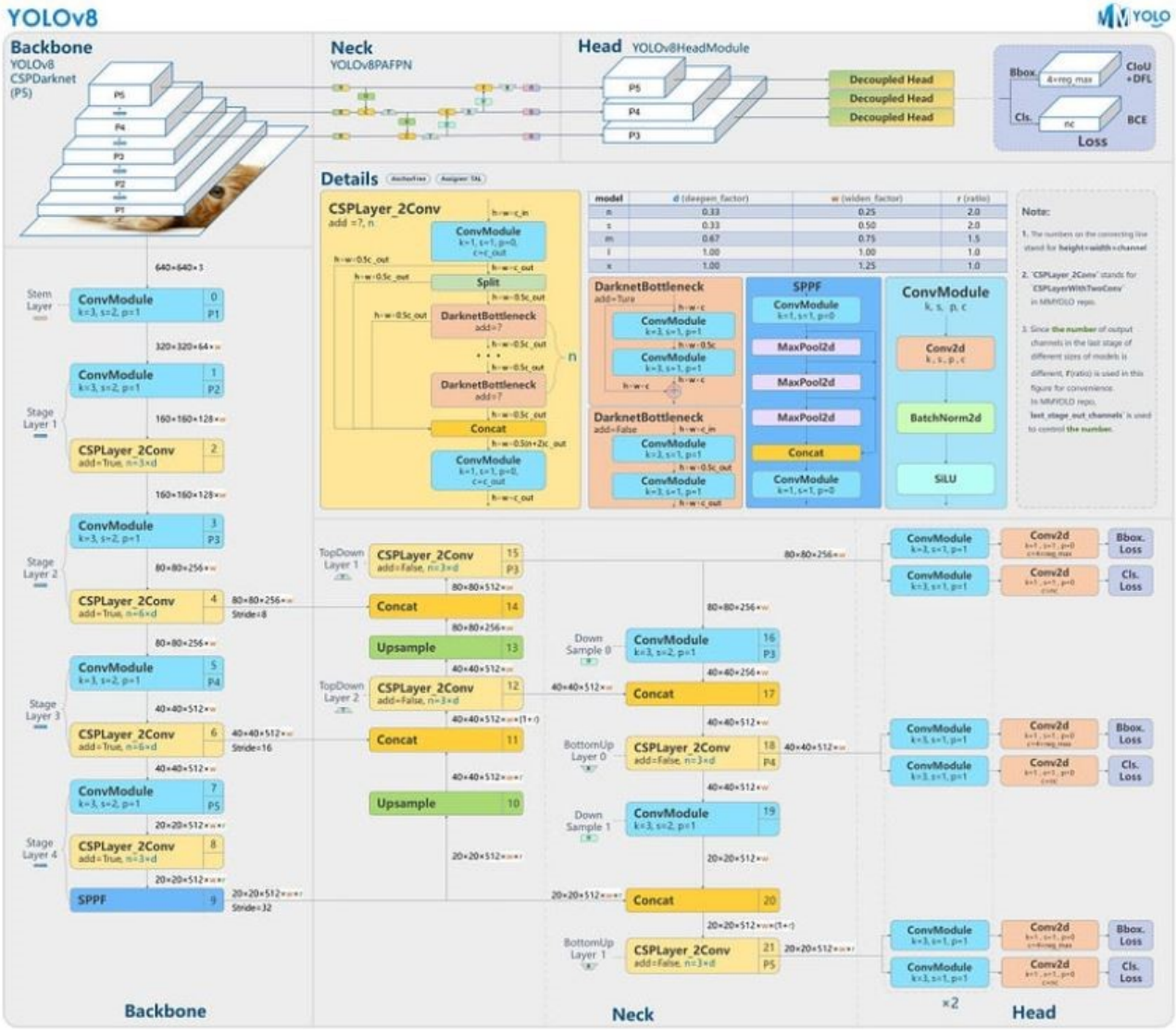
kuvan segmentointimenetelmät ovat päihittäneet muut johtavat menetelmät kuvantunnistuksessa. CNN:ään perustuvan kuvantunnistuksen haasteena on ollut se, että harjoitusdatanäytteitä on vaadittu paljon, jopa miljoonia, jotta kuvantunnistuksen tulos olisi hyvä. [28,12]

Parempia konvoluutionaalisia neuroverkkomalleja on kehitetty ja ne vaativat vähemmän harjoitusdataa opetukseen sekä kykenevät yhtäaikaisesti tarjoamaan sekä hyvän kohteenpaikantamistarkkuuden että hyvän kuvan sisällön analysoinnin tai segmentoinnin. Erilaisista koneoppimisen malleista keskeiseksi teknologiaksi on noussut reaaliaikaisessa hahmontunnistuksessa käytetty YOLO-malliperhe (eng. You only look once), joista uusin versio tällä hetkellä, on YOLOv8-mallit. Hahmontunnistukselle on kaksi tavoitetta: kohteentunnistuksen nopeus ja sen tarkkuus. Näiden kahden tavoitteen yhtäaikainen toteutuminen on yleisesti ottaen kompromissi, jos tunnistustarkkuus on korkea, niin nopeus on pienempi ja päinvastoin. Kuitenkin kehitetty YOLO-malliperhe kykenee yhdistelemään molempia kohteentunnistuksen tarkkuutta ja nopeutta paremmin kuin muut vastaavat koneoppimisen mallit. YOLO-malliperheeseen pohjautuvaa kohteentunnistusta on hyödynnetty monenlaisissa sovelluskohteissa, kuten liikenteen analysoinnissa, lääketieteessä sekä teollisuudessa laadunhallinnassa, esimerkiksi tuotevikojen ja poikkeamien havaitsemisessa. [23]

YOLO-mallin neuroverkkoon ei tarvitse syöttää dataa useampia kertoja, vaan yhdellä kertaa YOLO tunnistaa kohteiden rajat, luokittelee rajatut kohteet sekä poistaa päällekkäiset kohteet. Tunnistuksessa YOLO-malli jakaa kuvan SxS-ruudukkoon, johon se ennustaa B kappaletta kohteita rajaavia suorakaiteita C:lle eri luokalle. Tässä tapauksessa yhteen kuvaan voi tulla päällekkäin useampia kohteita rajaavia suorakaiteita, joista poistetaan päällekkäisten kohteiden rajat. Päällekkäisten kohteiden poistamisessa YOLO hyödyntää jokaiselle suorakaiteelle laskettua luottamusarvoa, joka määrittää kuinka luotettavana malli pitää sitä, että suorakaiteen sisällä on tunnistettava kohde ja kuinka tarkasti kohde on suorakaiteella rajattu. Luottamusarvoja hyödyntävä YOLO käyttää NMS-algoritmia (eng. Non-maximum supression) poistamaan päällekkäiset suorakaiteet, suurimpien luottamusarvojen perusteella. Lopuksi jäljelle jäävät vain erillisiä kohteita rajaavat suorakaiteet. Kuvassa 13 on esitetty kohteentunnistus YOLO-mallilla. [23]

Kuva 13: Kohteentunnistus YOLO-mallilla [23]

YOLO-mallien kehittyessä koneoppimisen malleja alettiin kuvaamaan kolmessa osassa selkäranka, niska ja pää (eng. backbone, neck ja head). Selkäranka perustuu konvoluutionaaliseen neuroverkkoon ja sen tehtävänä on eristää syötekuvasta kohteentunnistuksen kannalta hyödyllisiä piirteitä eri kokoluokissa esimerkiksi pienessä mittakaavassa kohteen reunoja ja tekstuureja sekä suuressa mittakaavassa kohteen osia ja merkityssisältöjä. Nämä piirteet selkäranka syöttää niskalle, joka toimii pään ja selkärangan välissä. Niska yhdistelee ja jalostaa selkärangan tuottamia piirteitä lisätäkseen tunnistustarkkuutta. Lopuksi koneoppimismallin pää tekee ennusteet kohteiden sijainneista ja luokittelee kohteet. Pää huolehtii myös päällekkäisten ennustusten poistamisesta NMS-algoritmilla. Ultralyticsin kehittämässä, tammikuussa 2023 ilmestyneessä, YOLOv8-malllissa käytetään selkärankaneuroverkkona 53 kerroksesta koostuvaa DarkNet-53 CNN:ää. YOLOv8 kykenee tunnistamaan kuvasta tuhat kohdekategoriaa. [22,23]



Kuva 14: YOLOv8-mallin rakenne [22]

Kuvassa 14 on esitetty YOLOv8 mallin rakenne, joka koostuu DarkNet-selkärangasta, niskasta ja pääosasta. YOLOv8-mallissa on monia 2D-konvoluutiokerroksia kaikissa kolmessa mallin osassa ja lisäksi se sisältää 2D-poolausta, näytejoukon normalisointia, sen koon pienentämistä ja kasvattamista. Tässä ei kuitenkaan mennä tarkemmin YOLOv8 mallin toimintaperiaatteeseen rakenteen kautta, vaan kuvan on tarkoitus havainnollistaa modernin neuroverkon kolmea osaa: selkäranka, niska ja pää.

2.7 Jatkuva oppiminen ja laskennan allokointi

Tässä luvussa esitellään tuoreinta tutkimusta MLOps-periaatteeseen kuuluvasta jatkuvasta oppimisesta. Konetta voidaan jatkuvan oppimisen avulla opettaa tunnistaa tarkemmin aiemmin opittuja kohteita tai sille voidaan opettaa vaiheittain uusia tunnistamaan uusia kohteita ja säilyttämään tietyissä rajoin aiemmin opittujen kohteiden tunnistaminen. Jatkuvasta oppimisen tutkimuksia on esitetty aliluvussa 2.7.1. Sitten aliluvussa 2.7.2 esitetään tuoreita tutkimuksia siitä, miten reuna-, sumu- ja pilvilaskentakerroksille on kehitetty menetelmiä, joilla koneoppimisen laskentatehtäviä voitaisiin jakaa optimaalisesti eri kerrosten laitteiden suorittamiseksi. Lisäksi tässä aliluvussa on esitetty toinen lähestymistapa, jossa koneoppimismalleja kevennetään, jotta niitä voitaisiin suorittaa resurssirajoittuneilla reunalaskennan laitteilla.

2.7.1 Jatkuva oppiminen

MLOps-periaatteisiin sisältyy *jatkuva oppiminen*, jossa koneoppimisen mallia kehitetään opettamalla mallia uudelleen, kun mallin suorituskyky heikkenee tai uutta dataa on saatavilla. Aina kun uutta dataa on saatavilla jatkuvaa oppimista, voidaan tehdä niin, että vanhalla ja uudella datalla koneoppimisen malli opetetaan täysin alusta asti uudelleen ja aiemmin opetettu mallia ei enää hyödynnetä. Tällöin tuhlataan laskentaresursseja ja energiaa. Lisäksi ongelmaksi voi muodostua kasvavan opetusdatamäärän aiheuttama laskenta-ajan pidentyminen. Mitä kauemmin uutta opetusdataa on kerätty, sitä kauemmin harjoittamisessa kestää, koska uusi data pitää aina liittää osaksi aikaisempaa datajoukkoa ja malli pitää opettaa uudelleen alusta. Toisaalta uuden mallin opetus voidaan saada suoritettua vasta sen jälkeen, kun riittävästi uutta opetusdataa olisi jo saatavilla mallin seuraavaa kehitysiteraatiota varten. Tällöin mallin suorituskyky voi olla huonompi kuin se olisi viimeisimmällä järjestelmästä kerätyllä datalla. [26]

Tutkimuksissa onkin pyritty kehittämään erilaisia tapoja toteuttaa jatkuvaa oppimista, joissa on pyritty pienentämään opetukseen tarvittavaa harjoitusdatamäärää, opettamaan mallia tuoreimmalla datalla, jotta malli osaisi viimeisimmän datan perusteella tuottaa ajankohtaisia ennusteita tai yhdistelmä näitä molempia. *Ajantasaisessa oppimisessa* (eng. Online learning) malli opetetaan vain tuoreimmalla opetusdatalla alusta asti, kun riittävästi uutta tietoa on saatavilla. Mallin opetus vain tuoreimmalla tiedolla sopii joihinkin sovelluksiin, jossa on kriittistä soveltaa uusinta tietoa, mutta ongelmana tässä tavassa on, että malli ei kykene säilyttämään kykyä esimerkiksi tunnistaa aiemmin opittua. Täysin uudelleen opetuksen ja vain tuoreimmalla tiedolla opetuksen välimaastoon sijoittuu proaktiivinen oppiminen, jossa opetus suoritetaan tuoreella kerätyllä datalla, mutta myös

joillain aiemmilla datanäytteillä. Tutkimuksessa [26] on esitetty ja sovellettu syville neuroverkoille *proaktiivista oppimista*, joka pyrkii säilyttämään kyvyn luoda ennusteita sekä uuden että aiemman opitun tiedon perusteella ja samalla vähentäen oppimisdatan kohtuutonta kumuloitumista. Proaktiivisessa oppimisessa on kuitenkin haasteena se, että uudessa kerätyssä datassa ei välttämättä ole monipuolisesti tai tämän työn kontekstissa, esimerkiksi kuvantunnistuksen kannalta riittävän kattavasti tunnistettavista kohteista erilaisia kuvia. Näin ollen tunnistus tarkkuus voi olla huonompi kuin täysin alusta asti kaikella siihen asti kertyneellä opetusdatalla. [26]

Jatkuva oppiminen tai tässä työssä käytetään nimitystä *elinikäinen oppiminen* (eng. Continual learning), jolla vältetään termien sekaannukseen vaara. Elinikäisessä oppimisessa hyödynnetään jo opetettua laajemmin erilaisia kohteita tunnistavaa tai toisin sanoen erilaisia tunnistustehtäviä suorittavaa koneoppimismallia tai perusmallia, joka opetetaan uudelleen tunnistamaan erilaisia kohteita. Lisäksi uuden kohteen tunnistamisessa tai uuden tunnistustehtävän omaksumisessa ei käytetä vanhaa opetusdataa enää. Perusmalli tarkoittaa siis koneoppimisen mallia, joka on opetettu valtavalla määrällä kuvia tunnistamaan esimerkiksi kymmeniä kohteita kuvista. Tällainen perusmalli voidaan siis elinikäisessä oppimisessa opettaa tunnistamaan uudenlaisia kohteita, jotka ovat samankaltaisia perusmallin tunnistamien kohteiden kanssa eli perusmalli opetetaan tiettyyn kontekstiin. Elinikäisen oppimisen kanssa samankaltainen oppimistapa on *siirto-oppiminen* (eng. Transfer learning), jossa perusmalli opetetaan perusmalli uudelleen tunnistamaan jokin uusi kohde. Kun taas elinikäisessä oppimisessa opetetaan perusmalli vaiheittain esimerkiksi yksi uusi kohde kutakin uudelleen opetuskertaa kohden. Useamman opetuskerran jälkeen malli oppii tunnistamaan useampia uusia kohteita. Toisaalta mallin uudelleen käynnistyksessä voidaan jo opetetun mallin harjoittamista jatkaa uudella datalla, jolloin unohtamisen on pienentyä. [39]

Elinikäisessä oppimisessa haasteena on *katastrofaalinen unohtaminen* (eng. catastrophic forgetting), jossa koneoppimismallin kyky tunnistaa aiemmin opittuja kohteita heikkenee ajan kanssa [39]. Tämä tapahtuu, jos opetettaessa painotetaan uudempaa dataa liikaa tai jos useampien uusien kohteiden tunnistamista on opetettu pidemmän aikaa. Tällöin kone pikkuhiljaa unohtaa aiemmin opitut kohteet. Tuoreemmissa tutkimuksissa on katastrofaalisen unohtamisen aiheuttamaa ongelmaa pyritty ratkomaan esimerkiksi tutkimuksessa [39] soveltamalla rajoitettua elinikäistä oppimista (eng Bounded continual training). Rajoitetussa elinikäisessä oppimisessa pohjamallin data jaetaan luokittain x eri luokkaan ja jokaiselle luokkadatalle opetetaan oma mallinsa, tällöin puhutaan *Indusoidusta siirto-oppimisesta*. Eri mallien ulostuloista koostetaan piirrevektori, jolle

suoritetaan luokittelu. Rajoitetun elinikäisen oppimisen etuna on, kun uutta dataa generoituu, voidaan päivittää vain soveltuvimman luokan malli uudella ja vanhalla datalla. Tällöin mallin vaatima laskenta-aika on paljon pienempi kuin täysin alusta opetetulla siirto-oppimista hyödyntävällä mallilla.

Toinen tutkimus [29] ehdottaa katastrofaalisen unohtamisen ratkaisuksi adaptiivista progressiivista elinikäistä oppimista (eng. Adaptive progressive continual learning). Tutkimus [29] jakaa elinikäisen oppimisen metodit, joilla katastrofaalista unohtamista voidaan estää, kolmeen eri luokkaan: kertaamiseen, regularisointiin ja parametrien eristämiseen pohjautuvat menetelmät. Kertaamisessa osajoukko aiemmin opittujen tehtävien dataa opetetaan tuoreen datan kanssa tai erikseen uudelleen. Regularisoinnissa häviöfunktioon lisätään regularisointitermi, joka torjuu unohtamista ja parametrien eristämisessä tietynkokoista neuroverkosta tunnistetaan aiemmat tehtävät tunnistavat parametrit, jotka pidetään samana. Kun taas dynaamisesti skaalautuvassa neuroverkossa jäädytettyä neuroverkkoa skaalataan uusilla neuroneilla, jotka tunnistavat uusia tehtäviä. Tutkimus [29] laajentaa elinikäistä oppimista käyttämällä vahvistusoppimista, Bayes-optimointia, Gaussin prosessimallia ja tarkkaavaisuutta (eng. Attention). Tarkkaavaisuus kuvantunnistuksessa soveltaa ihmisen kykyä keskittää huomionsa johonkin asiaan ja jättää muu huomiotta. Koneoppimisessa tämä tarkoittaa kykyä keskittyä kuvien tunnistamisessa vain oleellisiin piirteisiin. Tutkimuksessa yleisesti on sovellettu erilaisia menetelmiä elinikäiseen oppimiseen käyttäen tarkkaavaisuutta tai muita menetelmiä, joilla katastrofaalista unohtamista voitaisiin vähentää.

Jatkuva oppiminen on monipuolinen tutkimusalue, jossa on erilaisia yksityiskohtaisia termejä ja menetelmiä, mitä ei välttämättä tässä työssä ymmärretä vielä hyvin tarkasti. Kuitenkin parhaan tämänhetkisen käsityksen mukaan tämän työn jatkuva oppiminen, CT käsittää pääasiassa mallin tunnistamistarkkuuden kehittämistä jatkuvan oppimisen keinoin enemmän kuin elinikäistä oppimista, jossa koneopetetaan tunnistamaan uudenlaisia kohteita vaiheittain samalla mallilla. Tässä työssä tutkitaan siirto-opetetun mallin jatkuvaa opetusta, jossa siirto-opetetun mallin tarkkuutta pyritään parantamaan täysin alusta asti opetusta ja sen lisäksi mallin uudelleenkäynnistystä hyödyntäen ajantasaista opetusta verrataan keskenään.

2.7.2 Laskennan allokointi

Uusimmissa tutkimuksissa on käsitelty laskentatehtävien hajauttamista sumu-, reuna-pilvilaskennan laitteille. Tutkimuksen suuntana on toisaalta optimoida laskentatehtävien jakaminen reuna-, sumu- ja pilvilaskentakerrosten laitteille ja toisaalta on kehitetty tapoja

keventää raskaita koneoppimismalleja, jotta niitä voitaisiin suorittaa rajoittuneilla reunalaskentalaitteilla tai että laskentaa voitaisiin tuoda yhä lähemmäs dataa tuottavia antureita ja laitteita. Laskentatehtävien optimaalinen jakaminen on tutkimuksissa [44,32] tunnistettu ongelmaksi, joka voidaan ratkaista vahvistusoppimisen avulla. Vahvistusoppimisessa siis kone tekee tässä tapauksessa päätöksiä, mihin laitteille mikäkin tehtävä jaetaan. Kustakin päätöksestä kone saa palkkion tai rangaistuksen, joiden perusteella kone oppii korjaamaan päätöksentekoa kohti optimaalista ratkaisua. Esimerkiksi tutkimuksessa [44] esitettiin, että vahvistusoppimisella optimoidulla tehtävien jakamisella tavoitellaan minimoitua energiankulutusta, QoS-parametrien maksimointia ja tunnistustarkkuutta. Tutkimuksessa [32] esitettiin syvään vahvistusoppimiseen perustuva ratkaisu, jolla sekä allokoitiin optimimaalisesti resursseja heterogeenisten reuna-, sumu- ja pilvikerrosten välillä, että huomioitiin suoritettavien tehtävien väliset riippuvuudet, jotka on monesti kuvattu suunnatuilla syklittömillä verkoilla, DAG:lla. Toisin sanoen vahvistusoppimisratkaisu ottaa huomioon myös tehtävien tai osatehtävien suoritusjärjestyksen optimoidessa tehtävien jaottelua eri laskentakerroksissa.

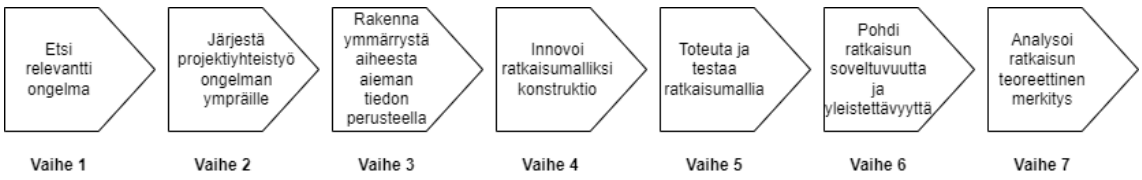
Toiseksi raskaat koneoppimisen mallit sisältävät useampia miljoonia parametrejä, jolloin laskenta vaatii paljon muistia ja prosessointitehoa. Tämän vuoksi jotkin rajoittuneemmat reunalaskennan, varsinkaan CPU:ta käyttävät, laitteet eivät kykene suorittamaan joitain koneoppimistehtäviä. Tutkimuksissa [1, 48] on kehitetty tapoja pienentää mallien parametrimääriä eli keventää malleja, jotta niitä voitaisiin suorittaa myös rajoittuneemmissa reunalaskentalaitteissa tai toisaalta joillakin reunalaskentalaitteilla on GPU-laskentakyvykkyys, mutta kevennetty malli nopeuttaa laskenta-aikaa tällaisilla laitteilla. Tutkimuksessa [1] on esitetty yleisimpiä mallin keventämismenetelmiä: karsiminen (eng pruning), kvantisointi (eng. quantization) ja tiedon tiivistäminen (eng. knowledge distilling). Karsimisessa tunnistetaan ja karsitaan mallista neuroneja tai neuroverkon kerroksia, joilla ei ole suurta vaikutusta tunnistustarkkuuteen ja näin ollen mallin parametrejä tai sen kokoa saadaan pienennettyä. Kvantisoinnissa taas pienennetään mallin laskenta-aikaa muuttamalla liukulukuja kokonaisluvuiksi. Tiedon tiivistämisessä sen sijaan opetetaan ensin kooltaan suurempi ja useampi parametrinen malli tunnistamaan esimerkiksi jokin kohde, jonka jälkeen suuremman mallin tieto tiivistetään ja opetetaan kooltaan pienemmälle mallille. Tutkimuksessa [1] tiivistettiin syvän neuroverkon tieto pienempään kNN-malliin, joka voitiin siirtää ajettavaksi reunalaskennan laitteella. Tuloksena oli, että isompikokoisen mallin (koko 2.8 megatavua) suorittaminen pilvessä tiedonsiirtoineen kesti 207,6 millisekuntia, reunalaskennassa 60,1 millisekuntia ja tiivistetyn mallin (koko 45 kilotavua) suorittaminen mikrokontrollerissa verkon reunalla kesti 9 millisekuntia.

Toisessa tutkimuksessa [25] kehitettiin tuulivoimalan vaihdelaatikon vian havaitsemiseksi syvään neuroverkkoon perustuva kuvantunnistin rajoittuneelle reunalaskennan laitteelle. Reunalaskentalaitteelle soveltuva optimaalinen NN-malli etsittiin neuraaliarkkitehtuurihakuna, NAS (eng. Neural architecture serach). Neuraaliarkkitehtuurihaussa käytettiin vahvistusoppimista, jonka avulla etsittiin Pareto-optimaalisia ratkaisuita, joissa suureet olivat neuroverkon tunnistustarkkuus neuroverkon laskennan vaativuus ilmaistuna vaadittavina liukulukuoperaatioina, (eng. Float-point operations), FLOPs. Tiivistettynä etsittiin siis vahvistusoppimisen avulla neuroverkkoa, jota voisi suorittaa rajoittuneella reunalaskentalaitteella. Kolmannessa tutkimuksessa [27] vastaavasti käytettiin tehokkaampaa reunalaskentalaitetta, jossa on GPU. Konvoluutionaalisella neuroverkolla, jossa on noin 2,4 miljoona parametriä, suoritettiin reunalaskentana kasvojentunnistusta laitteena Nvidia jetson AGX xavier, jossa on 512-ytiminen Volta GPU. Tällä laitteella kasvojentunnistustarkkuudeksi saatiin noin 70–80 prosenttia, mutta laskenta-aikaa ei määritetty.

3. KONSTRUKTIIVINEN TUTKIMUS

Konstruktiivisessa tutkimuksessa pyritään tuottamaan käytännön ongelman ratkaisemiseksi konstruktio. Konstruktio voi olla eri asioita esimerkiksi malleja, suunnitelmia ja tietojärjestelmämalleja. Konstruktiolla on uutuusarvo ja se muodostuu keksimisen ja kehittämisen seurauksena nojaten aiempaan tietämykseen. Konstruktiiviselle tutkimukselle ominaista on keskittyminen ”tosielämän ongelmiin, jotka koetaan käytännössä tarpeelliseksi ratkaista”. Toisena se joko tuottaa reaalimaailman ongelman ratkaisun ja mahdollistaa, että ratkaisun soveltuvuutta käytäntöön voidaan testata. Kolmantena konstruktiivinen tutkimus tulee olla liitetty aikaisempaan teoreettiseen tietämykseen. Neljäntenä empiirisen tutkimuksen tulokset ja havainnot tulee liittää takaisin teoreettiseen viitekehykseen. [31,50]

Konstruktiivisen tutkimuksen vaiheet voidaan määritellä eri tavoin [31], mutta eräs jaottelu seitsemään vaiheeseen on esitetty kuvassa 15. Ensimmäisessä vaiheessa valitaan ongelma, joka on tieteellisesti ja pragmaattisesti merkityksellinen. Toisessa vaiheessa järjestetään projekti käytännön ongelman ympärille ja varmistetaan sidosryhmien sitoutuminen projektiin. Kolmannessa vaiheessa analysoidaan ongelmaa ja rakennetaan ymmärrystä ongelmasta tutkimalla, mitä aiempaa tietoa ongelmasta on saatavilla. Neljännessä vaiheessa rakennetaan kertyneen ymmärryksen ja kokemuksen perusteella konstruktio ongelman ratkaisemiseksi. Viidennessä vaiheessa konstruktio käyttöönotetaan ja testataan sen toimivuutta käytännössä. Kuudennessa vaiheessa pohditaan rakennetun artefaktin yleistettävyyttä muihin samankaltaisiin ongelmiin ja liitetään tulokset teoriatietoon. Seitsemännessä vaiheessa tunnistetaan ja analysoidaan, mitä lisäarvoa konstruktio ja siitä tehdyt havainnot luovat jo olemassa olevaan tutkimukseen.



Kuva 15: Konstruktiivisen tutkimuksen vaiheet [31,50]

Tässä työssä tavoitteena on kehittää prosessiautomaation tarpeisiin ja sen pehmeän reaaliaikavaatimukset huomioiva konstruktio, joka koostuu yhdistelmästä erilaisia teknologioita. Tavoiteltu konstruktio yhdistelee tietotekniikan, automaation ja koneoppimisen teknologioita. Artefaktin rakentamiseksi valitaan potentiaalisten teknologioiden joukosta

sopivat teknologiat tähän työhön. Lisäksi tavoitteena on konstruktion avulla saada tai ainakin yrittää saada vastaus tutkimuskysymyksiin. Tutkimuskysymyksenä on, mitä palvelunlaadullisia eroja on, pehmeän reaaliaikaisuuden näkökulmasta, reaaliaikaisen höyryntunnistuksen suorittamisessa eli inferenssissä reuna- tai sumulaskentalaitteella. Toisena tutkimuskysymyksenä on, mitä hyötyjä ja haasteita on jatkuvan oppimisen hyödyntämisessä koneoppimisessa.

Höyryntunnistusjärjestelmän tarkoituksena on mahdollistaa automaatioprosessin valvonta, jolta vaaditaan pehmeää reaaliaikaisuutta. Reaaliaikaisuuteen sisältyy siirrettävän datan samanaikaisuus sekä datankeräämisen ja arvioinnin oikea-aikaisuus. Oikea-aikaisuus koostuu vielä ennustettavuudesta eli, että järjestelmä toimii ennakoidusti, vaikka data saapuukin satunnaisilla ajan hetkillä. Toisaalta oikea-aikaisen järjestelmän tulee olla luotettava siten, että toiminta jatkuu virheistä ja häiriöistä huolimatta. Molempien itse höyryntunnistuksen toteuttavien laitteiden oikea-aikaisuutta voidaan mitata viiveenä, joka muodostuu konttiin saapuneelle kuvalle suoritettuun inferenssiin kuluvasta ajasta ja ennusteen siirtämisestä asiakkaalle verkon ylin. Inferenssiajan ja kokonaisviiveen erotuksena saadaan tiedonsiirron ja mahdollisen datankäsittelyn viive. Useammilla mittauksilla saadaan palvelunlaadullisille ominaisuuksille keskiarvo.

Toista tutkimuskysymystä, mitä hyötyjä ja haasteita jatkuvan oppimisen hyödyntämisestä koneoppimisessa, tutkitaan mittaamalla sekä koneoppimisenmallin harjoittamiseen kuluvan ajan että mallin kohteentunnistustarkkuuden sekä harjoitus- että testivaiheessa suhteessa harjoituskuvien määrään. Koneoppimisessa käytetään Konvoluutionaalisiin neuroverkkoihin pohjautuvaa Yolov8n-mallia, joka opetetaan tunnistamaan kuvasta höyryä. Kuvantunnistusmallin harjoittaminen tehdään tietokoneklusterilla, jossa neuroverkkoa harjoitetaan näytönohjaimen grafiikkaprosessorilla. Jatkuvaa oppimisen hyödyntämistä kokeillaan ja mitataan kahdella eri tavalla, joista molemmista mitataan harjoitus- ja testivaiheessa tarkkuuksia. Ensimmäinen tapa on jakaa kaikki harjoituskuvat kolmeen eri harjoitusjoukkoon.

Tämän jälkeen kuvantunnistusmallia harjoitetaan ensin yhdellä harjoitusjoukkoista, sitten toisella harjoitusjoukolla ja lopuksi kolmannella. Toisin sanoen ensimmäisen harjoitusjoukon kuvilla harjoitettua mallia opetetaan uudelleen toisen harjoitusjoukon kuvilla. Tällöin voidaan testata, kehittyykö mallin tunnistustarkkuus, kun mallia opetetaan erissä. Kutakin harjoitusdatajoukkoa vastaavaa opetuskertaa kutsutaan tässä opetuskierrokseksi siksi, ettei termien sekaantumista sattuisi mallin opetusparametrin eräkoon ja mallinopettamisessa sen kertaisten harjoituskuvien opetuserän kanssa tulisi sekaannusta. Tässä työssä verrataan kahta erilaista jatkuvan oppimisen menetelmää keske-

nään. Yolov8n-malli siirto-opetetaan tunnistamaan kuvista höyryä ja ensiksi mallin tarkkuutta pyritään parantamaan täysin alusta asti opetettuna, kun uusia höyrykuvia on kerätty. Toiseksi mallia harjoitetaan ”opetuskierroksittain” ajantasaisella oppimisella, jossa jokaisella kierroksella hyödynnetään uudelleenkäynnistämistä. Alusta asti ja ajantasaista oppimista verrataan keskenään.

Esimerkiksi, jos opetuskuvia on 100 kappaletta ja eräkoko on kymmenen, niin opetettaessa ladataan muistiin kymmenen kuvaa kerrallaan. Nämä opetetaan ja sitten ladataan toiset kymmenen kuvaa ja opetetaan ne. Kun taas opetuskierroksella tässä tarkoitetaan sitä, jos on kaiken kaikkiaan 300 opetuskuvaa ja ne jaetaan 100:n kuvan joukkoihin. Ensimmäisellä opetuskierroksella opetetaan mallia 100 kuvilla sitten jo 100 kuvalla opetettu malli opetetaan uudelleen seuraavalla 100:n kuvan joukolla ja niin edelleen. Tällaista jatkuvaa oppimista verrataan siihen, että kaikki opetuskuvat opetetaan mallille kerralla eli esimerkiksi 300:n kuvan joukko opetetaan kerralla. Tällä vertailulla saadaan selville voisiko tässä tutkimuksessa, myöhemmin tarkemmin, esitellyllä teknologiolla soveltaa jatkuvaa oppimista siten, että voitaisiin kerätä automaatio osaprosessista jokin määrä kuvia ja parantaa mallin tunnistustarkkuutta eri opetuskierroksilla sen sijaan, että malli pitäisi opettaa uudelleen sekä vanhemmilla ja tuoreilla kuvilla alusta asti.

Luvussa 4 esitellään tutkimukseen käytettyjä teknologioita ja eteenkin ohjelmistotyökalut ja niiden perustoiminnallisuus. Luvussa 5 on esitetty testilaitteisto, sen sijoittelukaavio ja miten tässä luvussa esitettyjä pehmeän reaaliaikaisuuden osa-alueita on suunniteltu testattavan. Luvussa 6 tehdään havaintoja, joita konstruktion testaamisessa ja käyttöönotossa ilmenee. Luvussa 7 esitetään tulokset ja pohditaan konstruktiivisen tutkimuksen tuloksena tuotetun artefaktin soveltuvuutta laajemmin vastaavien ongelmien ratkaisukeinoksi, ja pohditaan minkä kontribuution konstruktio olemassa olevaan tutkimukseen antaa.

4. KONSTRUKTION OHJELMISTOT TUTKIMUKSESSA

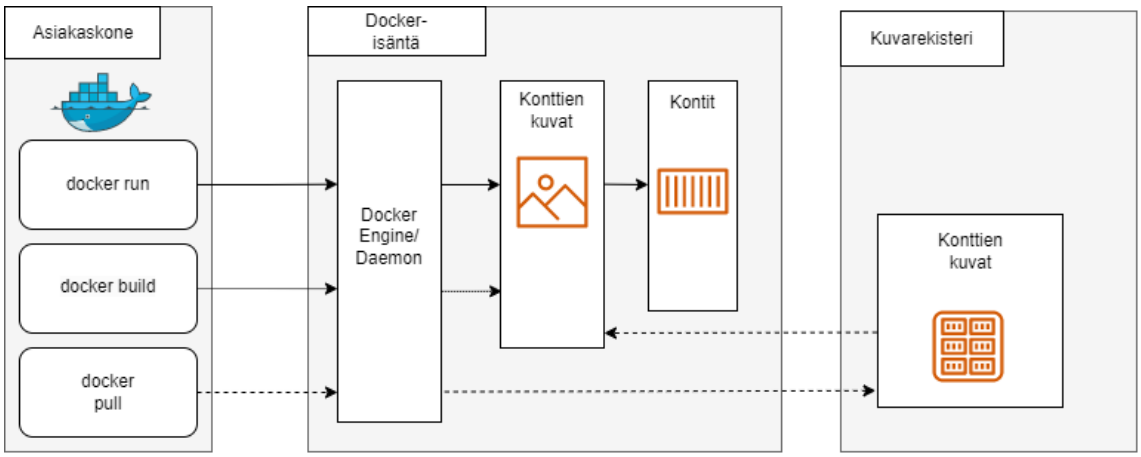
Tässä työssä höyryntunnistusneuroverkon opetuskuvien luokkien ja höyryn sijainnin merkintään käytetään Roboflow'ta, joka on esitetty aliluvussa 4.1. Sovellusten siirtämiseen käytetään yleisesti käytettyä avointa virtualisointialustaa nimeltä Docker, josta on kerrottu aliluvussa 4.2. Sovellukset ohjelmoidaan Python-ohjelmointikielellä, joka käyttää ohjelmakoodin kääntämiseen Python-tulkkia. Aliluvussa 4.3 esitellään höyryntunnistuksessa Ultralytics- ja PyTorch-kirjastot, joiden avulla voidaan käyttää valmista Yolov8n-mallia [77]. Molemmat kirjastot sisältävät datan käsittelyyn, neuroverkkojen harjoittamiseen ja testaamiseen soveltuvia valmiita funktioita ja metodeja. Aliluvussa 4.4 esitetään MLOps jatkuvan harjoittamisen kanavan luonti Dagshub-tietovaraston, sekä ohjelmakoodin versionhallintaohjelman Gitin että koneoppimisen datan versiointityökalun DVC:n avulla. Aliluvussa 4.5 käydään lyhyesti läpi, miten mallin hyperparametrien hienosäätöä voidaan suorittaa kokeellisesti MLflow'n avulla. Lopuksi aliluvussa 4.6 on esitetty, miten REST HTTP-palvelin voidaan toteuttaa FastAPI-, Pydantic-, Uvicorn- ja Requests-kirjastojen avulla. Lisäksi kaikista eri teknologioista esitellään perustoiminnallisuut, joiden koodi on tehty itse. Muutoin teknologioiden esittely perustuu niihin liittyviin lähteisiin, jotka on mainittu kunkin kohdan yhteydessä.

4.1 Roboflow

Roboflow on yleisesti käytetty työkalu koneoppisdatan luokkien tai nimiön asettamiseen [73]. Roboflow'lla harjoitus-, ja validointidatan luokat voidaan määrittää ja lisäksi kyetään asettamaan, missä ovat tunnistettavan kohteen rajat. Toisin sanoen Roboflow'n tarjoamat työkalut soveltuvat niin kuvien luokitteluun kuin kohteiden tunnistukseenkin. Koneoppimisdatan nimiö voidaan määrittää nopeasti ja tarkasti sekä muutaman esimerkin perusteella suuri joukko esimerkiksi kuvia voidaan ”nimetä” tekoälyavusteisesti. Lisäksi datanhallintaan liittyvät roolit huolehtivat siitä, että pääsy dataan on sallittu roolin määrittämällä tavalla, jolloin tietoturvallisuus ja yksityisyys tulee huomioitua, mikäli dataa ei halua julkiseksi. Tämän lisäksi Roboflow mahdollistaa datan esikäsittelyn koneoppimisen parantamiseksi. Datan esikäsittely ja nimeäminen voidaan automatisoida MLOps datakanavaksi, jonka seuranta on mahdollista ohjelmointirajapinnan kautta.

4.2 Docker ja Docker Daemon

Docker tarjoaa alustan konttien (eng. container) rakentamiseen ja ajamiseen. Kuvassa 16 on esitetty Docker-ohjelmiston toimintaperiaate. Docker-ohjelmisto toimii asiakas-palvelinarkkitehtuurina, jossa asiakaskoneella on työpöytäsovelluksena tai komentoriviltä käytettävä käyttöliittymä ja konttien ajamisesta huolehtiva Docker-isäntäpalvelin. Docker Engine huolehtii konttien luomisesta, ajamisesta, ajon keskeyttämisestä ja konttien poistamisesta. Asiakaskoneella komento ”docker build” pyytää Docker Daemonia rakentamaan kuvan Dockerfile-tiedon perusteella. Komento ”docker run” pyytää Docker Daemonia ajamaan jonkin kuvan, jolloin siitä tulee kontti. [62]



Kuva 16: Docker-ohjelmiston toimintaperiaate [62]

”Docker pull” -komennolla voidaan pyytää Docker Daemon:ia hakemaan kuvarekisteristä, esimerkiksi Dockerhub:sta, joko valmiin kontin kuvan tai kontin pohjakuvan, jota käytetään oman kontin kuvan rakennusvaiheessa.

Docker on yleisesti käytetty konttien rakennusohjelmisto. Dockerin avulla sovelluksia voidaan eristää infrastruktuurista, jolloin konttien vieminen tuotantoon on nopeaa. Docker-työkalut tarjoavat tavan, jolla sovelluksia voidaan teollisuudessa standardoida ajettavaksi konteissa [62]. Dockerin etuina ovat helppokäyttöisyys komentorivin tai työpöytäsovelluksen avulla sekä konttien luomisen, ajamisen ja konttien laitteelta toiselle siirron helppous. Lisäksi Docker-ohjelmisto mahdollistaa konttien sisäisten kirjastojen ja kuvien asentamisen nopeuden ja ohjelmakoodin tehokkaan testauksen lyhyessä ajassa. Kolmas etu Docker-työkaluissa on konttien pohjakuville laaja Dockerhub-tietovarasto [74], jossa on sekä virallisten tahojen että Docker-yhteisön jäsenten tuottamia pohjakuvia eri-

laisiin kehittäjien tarpeisiin. Osat pohjakuvista tukevat myös X86- ja ARM64-prosessoriarkkitehtuureja, mikä helpottaa konttien rakentamista eri reuna-, sumu- ja pilvilaskennanlaitteille.

Docker-kontin luominen tapahtuu siten, että ensin toteutetaan itse sovellus, jollain ohjelmointikielellä, sitten ohjelman kanssa samaan kansioon luodaan Dockerfile-niminen tiedosto, joka toimii kontin rakennusohjeena. Docker engine suorittaa Dockerfilen rivit yksi kerrallaan ja asentaa kaikki siinä määritetyt ohjelmistot ja kirjastot usein tavallista käyttöjärjestelmää kevyemmän käyttöjärjestelmän pohjakuvan päälle. Käyttöjärjestelmän omien pakettimanagerien tai pakettien asennusohjelman avulla kontin kuvaan voidaan asentaa halutun ohjelmointikielen tulkki, kuten Python-tulkki, ja kirjastoja, joita ohjelmakoodissa määritetty sovellus vaatii toimiakseen. Kun Dockerfile-tiedostoon on määritetty kaikki oleellinen, Docker Enginen kanssa voidaan rakentaa kontin kuva. Docker Engine rakentaa kuvan kerroksittain alkaen ensimmäisestä Dockerfilen rivistä. Kontin rakennettu kuva sisältää kaiken tarpeellisen kontin ajoa varten ja kuvasta tulee tarkalleen ottaen vasta ajettaessa kontti. Konttia voidaan ajaa missä tahansa laitteessa käyttöjärjestelmästä riippumatta, kunhan laitteeseen on asennettu Docker Engine tai Daemon nimeltä dockerd.

Ohjelma 1 on esimerkki Dockerfile-tiedosta. Ensin Docker-tiedostossa Docker Daemon suorittaa rivin 1, jossa kontin pohjaksi ladataan Dockerhub-tietovarastosta Ubuntu-käyttöjärjestelmän kuva. Kontin pohjakuvaksi voidaan ladata Dockehub:sta muita käyttöjärjestelmiä esimerkiksi pienikokoinen Alpine Linux, suoraan Python-ohjelmointiympäristö, tai pohjakuva kirjastolle tensorflow. Ohjelmassa 1 on asetettu pohjakuvaksi ARM64v8-prosessoriarkkitehtuuria tukeva Ubuntu-käyttöjärjestelmän pohjakuva.

```
1  FROM arm64v8/ubuntu
2
3  COPY qemu-aarch64-static /usr/bin
4
5  RUN apt-get update && \
6      apt-get install -y python3 python3-pip libgl1-mesa-glx libglib2.0-0 && \
7      rm -rf /var/lib/apt/lists/*
8
9  COPY requirements.txt ./
10 RUN pip3 install -r requirements.txt
11
12 COPY . .
13
14 EXPOSE 9000
15
16 CMD ["python3","/main.py"]
```

Ohjelma 1: Kontin rakennustiedosto

Rivillä 3 kopioidaan työkansiosta konttiin kansioon /usr/bin Qemu-emulaattori, joka mahdollistaa ARM64v8-pohjaisen kontin testaamisen laitteilla, jotka pohjautuvat X86- ja AMD64-prosessoriarkkitehtuureihin. Pohjakuvan päälle voidaan Ubuntu:n tai muun käyttöjärjestelmän pakettimanagerilla asentaa kirjastoja. Rivillä 5 päivitetään Ubuntu:n pakettimanageri apt ja sen jälkeen konttiin asennetaan pakettimanagerilla sekä Python3 että Pythonin pakettimanageri pip ja muutama muu kirjasto. Riveillä 9 kopioidaan requirements.txt-tiedosto kontin tiedostojärjestelmään. Rivillä 10 asennetaan Pythonin pakettimanagerilla requirement.txt-tiedostossa määritellyt kirjastot, kuten Ultralytics, Uvicorn, FastAPI ja niin edelleen. Rivillä 12 kopioidaan työkansiosta kaikki tiedostot konttiin. Rivillä 14 määritetään portti 9000, jolla kontista on yhteys kontin ulkopuolelle. Rivillä 16 määritetään, mitä kontti ajettaessa käynnistetään ensimmäisenä. Tässä tapauksessa ensimmäisenä konttia ajettaessa käynnistyvät python3-tulkki ja pääasiallisen sovelluksen sisältävä tiedosto.

4.3 YOLOv8-malli

Tässä työssä käytetään koneoppimisessa Ultralytics-kirjastoa [75], joka toimii neuroverkkojen luomiseen ja ajamiseen erikoistuneen PyTorch-kirjaston [63] kanssa. Ultralytics tarjoaa syväoppimiseen ja konenäköön YOLO-malleja. Uusimpana Ultralyticsin teknologiana on yhä tarkempaan ja nopeampaa reaaliaikaiseen kohteentunnistukseen ja kuvien segmentointiin kehitetty YOLOv8. YOLOv8 tukee kohteentunnistuksen lisäksi luokittelua, kuvien segmentointia ja kohteenseurantatekoälytehtäviä. YOLOv8n on versio kahdeksan malleista pienin ja se käyttää 3,2 miljoonaa parametriä, kun taas suurin mallin YOLOv8x 68,2 miljoonaa parametriä [77]. Näin ollen YOLOv8n ei ole yhtä tarkka kuin YOLOv8x, mutta se on harjoittamisessa ja tunnistuksessa käytettynä nopein saman version malleista. Ultralytics käyttää pohjana PyTorchia, joka on syväoppimiseen erikoistunut kirjasto, jonka etuina on nopeus, joustavuus ja käyttäjäystävällisyys. PyTorch tarjoaa työkaluja datan käsittelyyn, koneoppimisen mallien luontiin, harjoittamiseen sekä mallin parametrien optimointiin. Lisäksi PyTorch tukee CUDA:a, joka mahdollistaa neuroverkkojen ajamisen yhden tai useamman näytönohjaimen grafiikkaprosessoreilla. GPU:lla neuroverkon harjoittaminen on nopeampaa kuin CPU:lla, mutta nopeus riippuu neuroverkon koosta ja vaadittavan muistin määrästä. Vaikka Ultralytics toimiikin PyTorchin kanssa tai osin sen päällä, niin Ultralytics-kirjaston syntaksi ei ole suoraan yhtenevä PyTorchin kanssa. Kuitenkin mallien määrityksessä ja termeissä on paljon samaa.

```
1   from ultralytics import YOLO
2
3   # luo uusi YOLOv8n-malli
4   model = YOLO('yolov8n.yaml')
5
6   # lataa valmiiksi harjoitettu YOLOv8n-model (suositeltu harjoittamiseen)
7   model = YOLO('yolov8n.pt')
8
9   # Harjoita mallia käyttämällä 'coco128.yaml' datasettiä
10  results = model.train(data='coco128.yaml', epochs=3, batch=2)
11
12  # Arvioi mallin suorituskykyä validointi setillä
13  results = model.val()
14
15  # Suorita kuvalle kohteentunnistus
16  results = model('https://ultralytics.com/images/bus.jpg')
17
18  # Vie malli to Pytorch - formaattiin
19  success = model.export(format='')
20
```

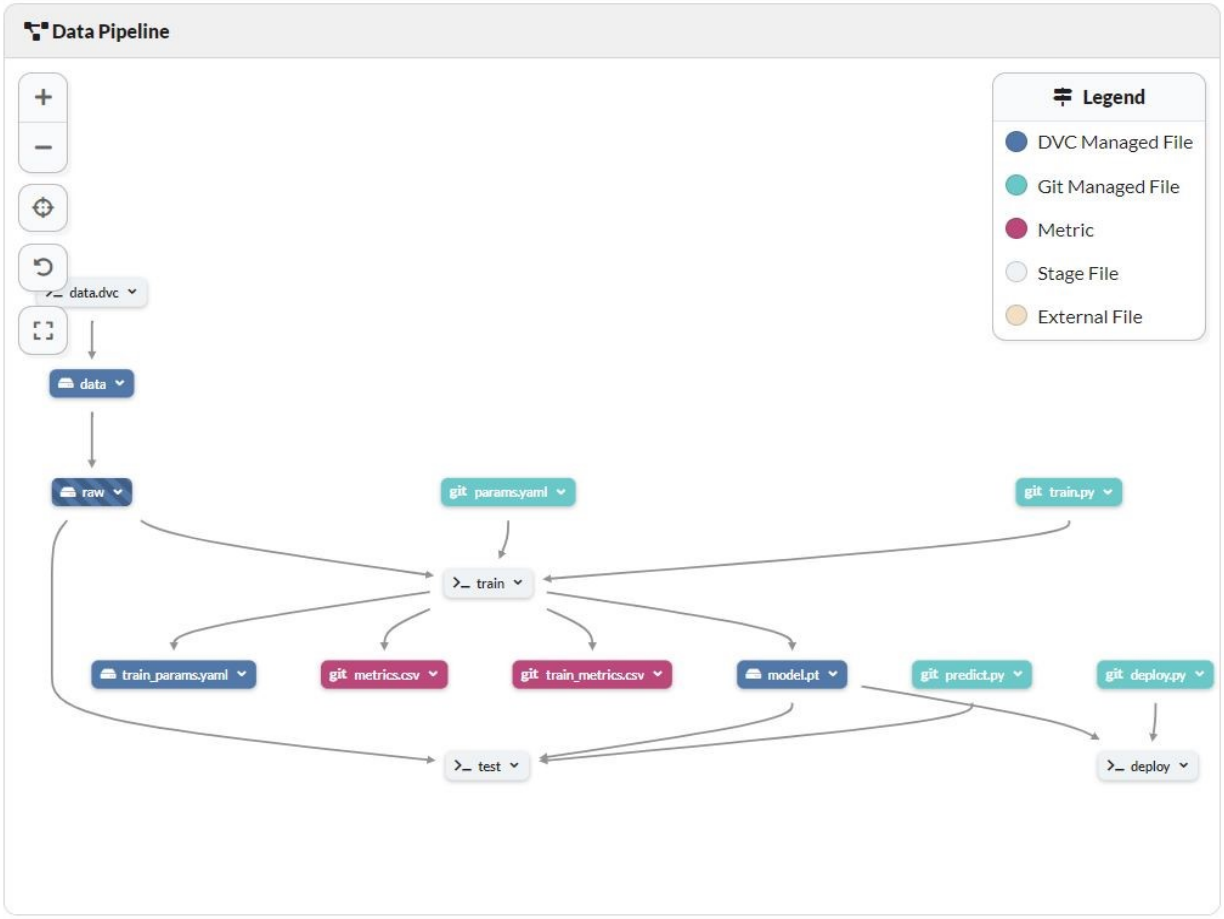
Ohjelma 2: Mallin luominen ja harjoittaminen Ultralytics-kirjaston avulla

Ohjelmassa 2 on esitetty Ultralytics-kirjaston perustoiminnallisuudet. Rivillä 1 tuodaan Python-ohjelmakoodiin käytettäväksi Ultralytics-kirjastosta YOLO-mallit. Rivillä 4 luodaan YOLOv8n-malli YAML-tiedostosta, jossa on määritetty muun muassa syötekuvan koko pikseleinä, käytetty optimoija, mallin oppimisnopeus ja Boolean-arvo sille, onko malli valmiiksi harjoitettu. Ohjelman 2 rivillä 7 tuodaan YOLOv8n-malliin valmiiksi harjoitetut painot. Rivillä 10 harjoitetaan aiemmilla riveillä määritettyä mallia esimerkin vuoksi avoimesti ladattavissa olevalla coco128-datasetillä ja harjoituksen tulokset tallennetaan ”results”-muuttujaan [49]. Samalla rivillä opetuskausiksi on määritetty kolme ja harjoituserän kooksi kaksi. Rivillä 13 validoidaan harjoitetun mallin suorituskykyä validointidatalla, jonka Ultralytics osaa hakea itsenäisesti kansiosta nimeltä *val*. Rivillä 16 suoritetaan mallille ennestään tuntemattomalle kuvalle kohteentunnistus, ja lopuksi rivillä 19 viedään harjoitettu malli ohjelmakoodista tiedostoon PyTorch-yhteensopivassa formaatissa, esimerkiksi *malli.pt*. Malleja voidaan viedä tiedostoon myös muissa formaateissa, kuten ONNX tai Keras-kirjastolle soveltuvassa TF SavedModel -formaatissa.

4.4 MLOps jatkuva oppiminen Dagshubin avulla

Dagshub on datatieteilijöille suunnattu tietovarasto [61], jota voidaan käyttää Git-tietovaraston kanssa. Git-tietovarastoon tallennetaan ohjelmakoodi ja Dagshubiin tallennetaan suurikokoiset tiedostot tai kansiot pääasiallisesti koneoppimisen mallin harjoittamiseen käytettävä data ja opetettu malli. Dagshubin hyödyllinen ominaisuus MLOps:n näkökulmasta on, että Dagshubilla voidaan muodostaa suunnattuja syklittömiä vuokaavioita, lyhennettynä DAG. DAG:n avulla kyetään muodostamaan koneoppimisdatan esikäsittely, mallin opetus, testaus ja validointikanavia. Näin ollen Dagshubin kanssa pystytään muodostamaan MLOps jatkuvan treenaamisen kanavia.

Kuvassa 17 on esitetty esimerkkikanava, joka on tarkoitettu CNN-tyyppisen neuroverkon jatkuvaa treenausta varten. Harmaalla värillä oleva ”laatikko” vuokaaviossa on jatkuvan treenaamisen vaihe *train,* jonka syötteenä on DVC:n seuraama datakansio. Datakansio sisältää kansion raw, jossa on mallin harjoittamiseen ja testaamiseen tarvittavat kansiot opetus- ja testikuville sekä näissä kuvissa olevien höyryä rajaavien suorakulmioiden koordinaatit ja kohteiden luokat. Datakansion kuvien luokat ja höyryn sijaintitieto on määritetty Roboflow’lla.



Kuva 17: Jatkuvan treenaamisen kanava Dagshubissa

Kuvassa 17 *train*-vaihe harjoittaa koneoppimisen mallin syötteinään harjoitusdata ja varsinaisen harjoittamisen suorittava Python-ohjelmakoodi *train.py*. Lisäksi syötteenä on Gitin seuraama tiedosto *params.yaml*, joka sisältää YOLOv8n-mallin käyttämät harjoitusparametrit, kuten harjoituserän koko (eng. batch size), oppimisnopeus (eng. learning rate) ja opetuskausien (eng. epochs) lukumäärä. Harjoittamisvaiheen ulostulona on YOLOv8n-malli, joka tallennetaan tiedostoon *model.pt*. Valmiiksi harjoitettua YOLOv8n-mallia voidaan käyttää pohjana, kun halutaan omalla räätälöidyllä data opettaa YOLOv8 tunnistamaan esimerkiksi höyryä. YOLOv8-malleja on käytetty tunnistamaan metsäpaloja, joten höyryntunnistukseen voidaan soveltaa YOLOv8 mallia, sillä se pystyy tunnistamaan samantyyppisiä kohteita. Lisäksi ulostulona on myös harjoituksen metriikat eli tässä esimerkissä käytetään harjoittamisen metriikkana harjoittamiseen kulunutta aikaa, harjoituskuvien määrää ja oppimistarkkuutta. Nämä Gitin seuraamat metriikat tallennetaan tiedostoihin *metrics.csv* ja *train_metric.csv*. Toiseksi testausvaiheessa nimeltä *test* syötteenä on harjoitettu koneoppimisen malli, datakansiosta testidata, jolla mallin tunnistuksen yleistettävyyttä arvioidaan, sekä itse testauksen suorittava ohjelmakoodi *predict.py*. Testausvaiheen ulostulona on mallin ennustustarkkuus testidatalle.

Kolmantena vaiheena on *deploy*, joka ottaa syötteenä testatun mallin sekä ohjelmakoodin *deploy.py*. Käyttöönottovaiheessa harjoitettu ja testattu malli siirretään REST HTTP-pyyntönä aktiivista höyryntunnistusta suorittavalle palvelimelle, joka sijaitsee tässä tutkimuksessa joko reuna- tai sumulaskentalaitteella. Kuvassa 17 sinisissä laatikoissa on DVC:n [66] hallinnoimia tiedostoja, joista koneoppimisen dataversiohallinnankirjasto pitää kirjaa. Lisäksi DVC-kirjaston avulla voidaan luoda MLOps kanavan vaiheet, jotka Dagshub esittää suunnattuna syklittömänä vuokaaviona. Git-kirjastolla voidaan pitää kirjaa turkooseissa laatikoissa olevista kanavan eri vaiheiden ohjelmakoodeista sekä punaisissa laatikoissa olevista eri vaiheiden metriikoista.

DVC koneoppimisen dataversiointityökalulla yleisesti ottaen seurataan suurien tiedostojen, kuten harjoitus- ja testidatan sekä koneoppimismallin sisältäviä tiedostoja ja Git-kirjastolla seurataan ohjelmakoodia sekä koneoppimisen metriikoita. Seuraaminen tarkoittaa tässä kontekstissa, että kyseinen kirjasto pitää kirjaa esimerkiksi harjoitusdatan muutoksista esimerkiksi, kun uutta harjoitusdataa generoituu.

```
1 dvc init
2
3 # Lisää datakansio dvc:n seurattavaksi
4 dvc add data./
5
6 # Lisää train_model.py git:n seurattavaksi ja
7 # ilmoita git:lle, että dvc seuraa datakansiota
8 git add data.dvc .gitignore train_model.py
9
10 # Lisää projektin snapshot git:iin
11 git commit -m "Esimerkki ohjelmakoodi lisatty"
12
13 # Työnnä muutokset Dagshubiin.
14 git push
15 dvc push -r origin
```

Ohjelma 3: tiedostojen siirtäminen Dagshubiin

Ohjelmassa 3 on esitetty esimerkki, miten tiedostoja voidaan siirtää Dagshubiin DVC:n ja Gitin avulla. Rivillä 1 alustetaan DVC. Rivillä 4 lisätään DVC:n seurantaan kansio *data*, jossa on esimerkiksi kuvan pikselien RGB-sävyt ja kuvien luokat ilmoitettuna .csv- tai .txt-tiedostoissa sekä itse harjoitus- että testikuvat erikseen. Rivillä 8 ilmoitetaan Git-kirjastolle, että DVC seuraa datakansiota, joten Git voi ohittaa tämän kansion seurannan. Tämä on ilmoitettu lisäämällä tiedosto *data.dvc* gitin seurantaan. Lisäksi Gitille lisätään seurattavaksi koneoppimisenmallin harjoittamisen ohjelmakoodi *train_model.py*. Tämän jälkeen rivillä 11 Gitiin lisättyjen tiedostojen tämänhetkinen versio asetetaan odottamaan siirtoa Dagshubiin saatesanoineen. Lopuksi riveillä 14 ja 15 siirretään tiedostot Dagshubiin. Ohjelma 3 on esimerkki, jonka käyttöä edeltää Dagshub-tietovaraston luominen ja jotta tiedostot siirretään oikeaan varastoon, pitää asettaa *dvc* ja *git remote* kirjautumisineen.

4.5 MLflow

MLflow on avoimen lähdekoodin alusta koneoppimisen elinkaaren hallintaan ja kokeiden tekemiseen mallilla [71]. MLflow-alusta koostuu neljästä komponentistä: MLflow-seuranta, MLflow-projektit, MLflow-mallit ja mallirekisteri. MLflow-seurannalla voidaan yhdessä Dagshubin kanssa tehdä kokeita koneoppimisen mallille esimerkiksi voidaan kokeilla, millaisia metriikoita mallin parametrien ja optimoijan muutokset tuottavat. Erinäisien kokeiden tuloksista ja niissä tuotetuiden mallien versioista kyetään pitämään kirjaa. Erilaisia malleja voidaan siirtää tarvittaessa työvaiheesta, tuotantoon ja arkistoon. MLflow'lla suoritettujen kokeiden tuloksia voidaan vertailla keskenään sekä taulukko- että graafimuotoisina. Dagshubissa on itse tietovaraston yhteydessä kokeet-välilehti,

josta voidaan seurata MLflow-kirjaston tuottamia tuloksia, mutta myös erillinen käyttöliittymä kokeiden seurantaan ja mallien tarkasteluun.

```
1 import mlflow
2
3
4 with mlflow.start_run():
5
6     # Tähän tulee esimerkiksi harjoitus- tai evaluointiohjelma.
7
8     # Seuraa mallin parametrejä
9     mlflow.log_param('parametrin nimi', param)
10
11    # Pitää kirjaa mallin versiosta
12    mlflow.log_artifact('artefaktin sijainti')
13
14    # Seuraa mallin metriikoita.
15    mlflow.log_metric('metrics_name', metric)
16
```

Ohjelma 4: MLflow-seurantaesimerkki ohjelmakoodissa

Ohjelmassa 4 on esimerkki, miten MLflow-kirjaston avulla voidaan seurata mallia ja sen hyperparametrejä sekä metriikoita ohjelmakoodissa. Rivillä 4 käynnistetään seuranta ja riville 6 ohjelmoidaan koneoppimisen mallin harjoittaminen tai muu jatkuvan oppimisen vaihe, jonka tietoja halutaan seurata. Sen jälkeen rivillä 9 seurantaan laitetaan jokin mallin parametri, kuten opetuskausi, jota halutaan seurata. Rivillä 12 seurantaan asetetaan artefakti, joka tässä yhteydessä tarkoittaa opetuksen ulostulotiedostoja eli harjoitettua mallia tai datatiedostoja. Rivillä 15 seurantaan asetetaan opetuksen tuottamat metriikat.

MLflow-alustan komponentti MLflow-projektit tarjoavat koneoppimisohjelmakoodin muotoiluun työkaluja, joiden avulla ohjelmakoodia voidaan uudelleen käyttää ja ajaa Python-virtuaaliympäristössä ja Docker-konttiympäristössä. MLflow-mallikomponentti tarjoaa opetettujen mallien tallentamisen eri formaateissa, kuten Pytorch, Scikit-learn, Tensorflow, ONNX, OpenAI ja monille muille koneoppimisen- ja tekoälynteknologioille sopivissa muodossa. Tätä ominaisuutta kutsutaan MLflow'ssa *flavoriksi*. Valmiiksi määritetyillä flavoureilla ja myös Pythonin avulla räätälöityjä malleja voidaan ajaa. Edellä mainittu mallin ajaminen koskee ohjelmakoodista käsin mallin ajamista, mutta MLflow tarjoaa mallien käyttöönoton itsenäisinä Docker-kontteina. Nämä kontit tarjoavat neljä REST API endpointia: /ping, /health, /version ja /invocation. Kahdella ensimmäisellä endpointilla voidaan seurata kontin terveyden tilaa, kolmannella seurataan mlflow:n versiota ja /invocation endpointilla pyytää inferenssiä. Neljäs komponentti MLflow-mallirekisteri on keskitetty mallivarasto, johon voidaan tuoda koneoppimisen malleja ja lisäksi sieltä voidaan hakea malleja käyttöön. [71]

Tässä työssä konstruktion rakentamisessa käytetään MLflow-alustan komponenttia MLflow-seuranta yhdessä Dagshubin kanssa, jotta voidaan ensin kokeellisesti testata erilaisia hyperparametrivaihtoehtoja höyryntunnistusmallin harjoittamiseksi. MLflow'n tarjoamilla työkaluilla selvitetään, miten höyryntunnistusmallista saisi riittävän tarkan ja toisaalta kokeilemalla opittiin Ultralytics-kirjaston käytöstä sekä sen harjoitusprosessista tutkittaessa. MLflow-kirjastoa käytettiin vain ohjelmakoodista käsin, eikä käytetty MLflow-projektit komponentin tarjoamaa API:ta, vaikka se tässä luvussa esitettiinkin tarkoituksena lisätä ymmärrystä MLflow'n toiminnallisuuksista.

4.6 Uvicorn-höyryntunnistuspalvelin

Eri laitteille voidaan jakaa eri koneoppimisen prosessin vaiheita. Yhteen konttiin voidaan laittaa koneoppimisen avulla toteutettu kohteentunnistus eli tällainen kontti tarjoaa palveluna ennusteen uudelle datalle, joka syötetään konttiin, ja toisaalta toisaalla harjoitetun koneoppimisen mallin hyperparametrit pitää siirtää inferenssi-konttiin aina, kun uusi harjoitettu malli on valmis. Inferenssi-kontti voidaan toteuttaa verkkopalvelimena, joka tarjotaan RESTful-tyyppisenä verkkopalveluna. RESTful asettaa verkkopalveluille yhtenäisen rajapinnan, joka parantaa verkkopalveluiden skaalautuvuutta, tehokkuutta ja muokattavuutta. RESTful tarjoaa tavan päästä käsiksi tietoon, lisätä, muokata ja poistaa sitä verkkopalvelimelta. REST http -metodeja ovat *Get*, *Post ja Put*. Get-metodilla voidaan verkkoselaimen tai terminaalin avulla tai ohjelmakoodista käsin pyytää palvelimelta jotakin resurssia, Post-metodilla palvelimeen voidaan viedä uusi resurssi ja Put-metodilla voidaan päivittää, jotain jo olemassa olevaa resurssia verkkopalvelimella.

Tässä työssä käytetään FastAPI-, Uvicorn- ja Pydantic-kirjastoja [65,68], jolla voidaan toteuttaa verkkopalvelin ja sen käyttämät REST http -metodit. FastAPI-teknologia valittiin http-pyyntöjen toteutukseen sen helppokäyttöisyyden, intuitiivisuuden ja nopeuden vuoksi. FastAPI on itsessään nopea ja robusti ohjelmointirajapinta ja siksi se soveltuu käytettäväksi prosessiautomaatiossa, jossa edellytetään reaaliaikaisuutta ja robustiutta. Tämän lisäksi Pydantic-kirjaston kanssa FastAPI kykenee tunnistamaan virheelliset http-pyynnöt ja tiedottamaan niiden virhekoodit, jolloin virheiden paikantaminen ja korjaamien on helppoa. FastAPI myös tukee JSON-formaattia (eng. Javascript object notation), mikä helpottaa palvelimen pyyntöjen laatimista. Näiden lisäksi FastAPI mahdollistaa asynkronisen kommunikoinnin, mikä tehostaa palvelimen käyttöä ja tukee järjestelmän tarvetta samanaikaisuudelle. Verkkopalvelin toteutetaan Uvicorn-palvelimena, koska ensinnäkin se soveltuu käytettäväksi FastAPI kanssa ja toiseksi se tarjoaa paremman tietoturvan

verrattuna esimerkiksi Python-kirjaston omaan http-palvelimeen kirjastonimellä http.server. Uvicorn-palvelimen etuna on myös mahdollisuus ajaa sitä ohjelmakoodissa, jolloin Dockerfile-tiedostossa ei tarvitse erikseen määritellä ajettavaa palvelinta.

Ohjelmassa 5 rivillä 6 luodaan uusi FastAPI sovellus. Riveillä 9–13 luodaan Pydantic-kirjastoa hyödyntämällä pohjamalli luokalle nimeltä DATA, jonka sisältämät tietorakenteet määritetään. Palvelin tarkistaa, onko palvelimelle tullut esimerkiksi post-pyynnön mukana saapuva data pohjamallissa määritetyssä muodossa. Mikäli data on virheellisessä muodossa palvelin palauttaa virhekoodin. Rivillä 16–18 data on palvelinta käynnistettäessä ennalta määritetty tieto, joka on Pydantic-pohjamallin mukaan esitetty.

```
1  from fastapi import FastAPI
2  from pydantic import BaseModel
3  import uvicorn
4  import json
5
6  app = FastAPI()
7
8
9  class Data(BaseModel):
10     id: int
11     name: str
12     x_training: list
13     y_training: list
14
15
16 data = [
17     Data(id=1, name="training_data", x_training=[[1,2,3],[4,5,6],[7,8,9]], y_training=[0,0,1]),
18 ]
19
20
21 @app.get("/data")
22 async def get_data():
23     resource = await get_resource()
24     return resource
25
26
27 @app.post("/data", status_code=201)
28 async def add_data(msg: Data):
29     data.append(msg)
30     return msg
```

Ohjelma 5: http-palvelimen pyyntöjen käsittelijä ja REST-metodit

Ohjelmassa 5 rivillä 21 on määritetty FastAPI:n tarjoama Get-metodi, jolla palvelimelta Ovoidaan hakea resurssia IPv4-osoitteesta käyttäen tiedonsiirtoon http-protokollaa, kuten *http://0.0.0.0:1234/data*, jossa 1234 tarkoittaa palvelun portin numeroa ja /data tietyn resurssin osoitetta palvelimella. Rivillä 22 on määritetty funktio, joka ajetaan, kun palvelin vastaanottaa Get-pyynnön. FastAPI kykenee asynkroniseen viestintään, josta on esimerkki rivillä 23. Jos get_resource() funktion suorittamisessa kestää kauan, niin ilman

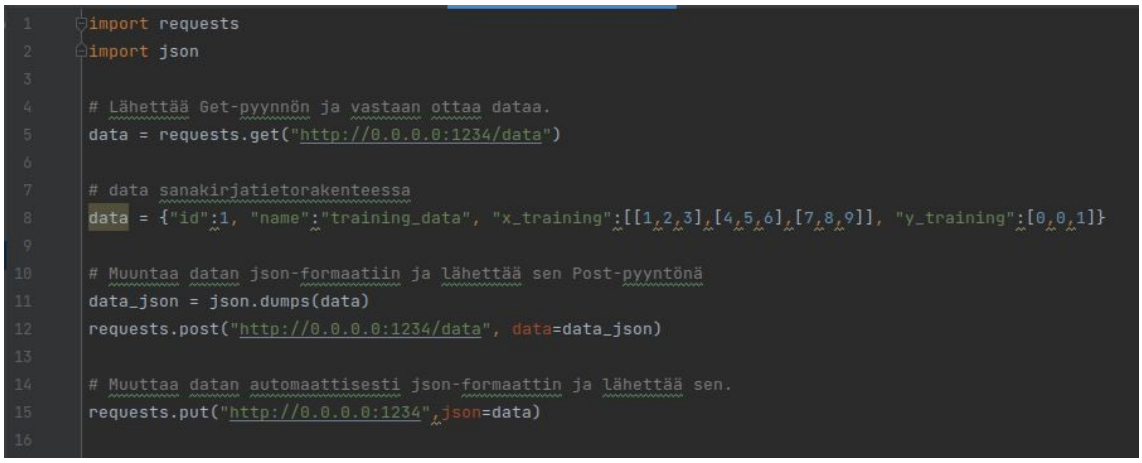
asynkronista viestintää palvelin odottaisi kyseisen funktion suorittamista, kunnes se palauttaa halutun tuloksen. Kuitenkin *await*-syntaksi mahdollistaa sen, että palvelin voi jatkaa muuta toimintaansa, sillä välin kuin get_resource()-funktio on suoritteilla. Kun funktio on suoritettu, palvelin toisen tehtävän suoritettuaan jatkaa palvelin siitä mihin se jäi. Riveillä 27–30 on määritetty palvelimen reagointi Post-metodiin. Post-pyynnön ja sen sisältämän tiedon (msg rivillä 28) vastaanotettuaan palvelin lisää tiedon datataulukkoon ja palauttaa tilakoodin 201.

Ohjelmassa 6 on esitetty ohjelman 5 käyttämä get_resource()-funktio ja verkkopalvelimen luonti ja ajaminen Uvicorn-kirjastolla. Rivillä 38 luodaan palvelin ja ajetaan palvelinta ohjelmakoodista käsin. Palvelimen ajamisen voisi suorittaa myös ajamalla Uvicorn-palvelinta komentorivillä. Ajettaessa määritetään Python-tiedosto, jossa koodi sijaitsee. Ohjelman 5 tapauksessa ohjelmakoodi on tiedostossa main.py ja tiedostosta ajetaan, tässä tapauksessa, FastAPI-sovellusta, joka määritettiin ohjelmassa 5 rivillä 6. Uvicorn-palvelimelle määritetään myös portin numero, johon kohdistuvia pyyntöjä palvelin kuuntelee.

```
32
33  def get_resource():
34      return json.dumps(Data[0]["x_training"])
35
36
37  if __name__ == '__main__':
38      uvicorn.run("main:app", port=1234, reload=True)
```

Ohjelma 6: verkkopalvelimen konfigurointi

Uvicorn-palvelimelle REST-pyyntöjä lähetetään se ulkopuolelta Requests-kirjaston avulla. Ohjelmassa 7 on esitetty esimerkkikoodi Requests- ja JSON-kirjaston käytöstä. Rivillä 5 lähetetään esimerkkipalvelimelle Get-pyyntö ja vastaanotetaan sieltä tieto muuttujaan data. Rivillä 8 luodaan sanakirjatietorakenne, johon on tallennettu koneoppimisen harjoitusdataa.



```
1  import requests
2  import json
3
4  # Lähettää Get-pyynnön ja vastaan ottaa dataa.
5  data = requests.get("http://0.0.0.0:1234/data")
6
7  # data sanakirjatietorakenteessa
8  data = {"id":1, "name":"training_data", "x_training":[[1,2,3],[4,5,6],[7,8,9]], "y_training":[0,0,1]}
9
10 # Muuntaa datan json-formaatiin ja lähettää sen Post-pyyntönä
11 data_json = json.dumps(data)
12 requests.post("http://0.0.0.0:1234/data", data=data_json)
13
14 # Muuttaa datan automaattisesti json-formaattin ja lähettää sen.
15 requests.put("http://0.0.0.0:1234",json=data)
16
```

Ohjelma 7: Requests-kirjaston käytöstä

Riveillä 11–12 muutetaan data JSON-formaattiin ja lähetetään se palvelimelle Post-pyynnöllä. Requests-kirjasto tunnistaa JSON-muotoon sopivan tietorakenteen, jolloin se muotoilee automaattisesti datan oikeaan muotoon ja sitten pyytää esimerkkipalvelinta päivittämään sen tietoja, mikä on kuvattu rivillä 15.

5. TESTILAITTEISTO JA -SUUNNITELMA

Tässä luvussa esitellään työssä käytettävän tutkimuslaitteiston tärkeimpiä teknisiä ominaisuuksia lyhyesti, jotta testilaitteiston laskentaresurssit tulisivat ilmi. Laskentalaitteiden resurssit vaikuttavat siihen, millainen kuinka kompleksista laskentaa laitteella kyetään käytännössä toteuttamaan. Sitten esitetään, miten laitteisto on hajautettu reuna-, sumu- ja pilvilaskennan kerroksiin. Laitteistona tässä työssä on reunalaskennan laitteena Ciscon IR1101-teollisuusreititin, sumulaskennan laitteina toimii tietokoneklusteri. Lisäksi data höyryntunnistukseen kerätään automatisoidusta prosessista IP-kameralla. Aliluvussa 5.2 esitellään testisuunnitelma siitä, miten mitataan koneoppimisen höyryntunnistuksen reaaliaikaisuutta reunalaskennan ja sumulaskennan laitteilla. Tämän lisäksi esitetään suunnitelma miten jatkuvan oppimisen tarkkuutta ja opetusaikaa mitataan. Sitten esitellään miten höyryntunnistuksen suorituskykyä ja jatkuvaa oppimista mitataan, ja mistä kohtaa järjestelmää tai jatkuvan oppimisen kanavan vaihetta tarkemmin ottaen mittaukset tehdään.

5.1 Testilaitteisto

Tässä työssä kuvia sekä koneoppimisen mallin alustamista että sen jatkuvaa harjoittamista varten höyrystä otetaan optisella kameralla. Kuvia halutaan siirtää automaattisesti kameralta, jolloin tässä työssä käytettiin näistä FTP-protokollaa, joka tiedostojensiirtoprotokolla. FTP-palvelin voidaan helposti muodostaa ohjelmakoodilla ja toisaalta kuvia ei tarvita tallentaa palvelimelle todella suuria määriä. Suuri kuvien määrä olisi voitu tallentaa erilliselle NAS (eng. Network Attached Storage) -tallennustilapalvelimelle, joka vaatii erikoituneen NAS-palvelinlaitteen. Kameralta kuvia työnnetään FTP-palvelimelle jotain satoja, jonka jälkeen aletaan alusta uudelleenkirjoittaa aikaisempia kuvia.

Kuvat automaatioympäristöstä siirretään IP-kamerasta FTP-palvelimelle, joka on tehty Python-ohjelmakoodilla ja sen jälkeen ohjelma on laitettu Docker-konttiin. Kontin pohjakuvana on pienikokoinen Alpine Linux -käyttöjärjestelmä. Konttiin laitettu FTP-palvelin siirretään ajettavaksi Intel NUC10 -minitietokoneelle [67] lähemmäs prosessia. Intel NUC 10 on minitietokone, jossa on kymenennen sukupolven Intel-prosessori kuudella ytimellä ja 4,70 gigahertsin maksimikellotaajuudella. Ja tämän lisäksi Intel NUC 10:iin saa maksimissaan 64GB DDR4-2666 muistia. Tiedonsiirtoon Intel NUC 10:ssä Gigabit Ethernet -portti. FTP-palvelin viedään erilliselle laitteelle sen sijaan, että se olisi viety IR1101-rei-

tittimelle suoritettavaksi. On todennäköistä, että Intel NUC 10:llä voisi suorittaa höyryntunnistamista, mutta tässä tutkimuksessa siihen ei ole keskitytty. Syyt tähän ovat se, että reitittimellä suoritetaan itse höyryntunnistusta, joka kuluttaa reitittimen laskentatehoa ja käyttää sen muistia, näin ollen FTP-palvelin samassa laitteessa voisi huonontaa reitittimen kykyä suorittaa reaaliaikaista höyryntunnistusta. Toisaalta kuvia automaatioympäristöstä tarvitaan sekä aktiiviseen höyryntunnistukseen että käytettäväksi jatkuvan oppimisen kanavan harjoitusdatana ja näin ollen tietoliikenne voi hidastaa reitittimen kykyä suorittaa höyryntunnistusta. Toisaalta FTP-palvelin olisi voitu toteuttaa myös tietokoneklusterissa, mutta silloin matka, jota tietoa pitää siirtää pitenee. Tietoa pitäisi siirtää verkossa jopa edestakaisin, mikä voi huonontaa reaaliaikaisuutta.

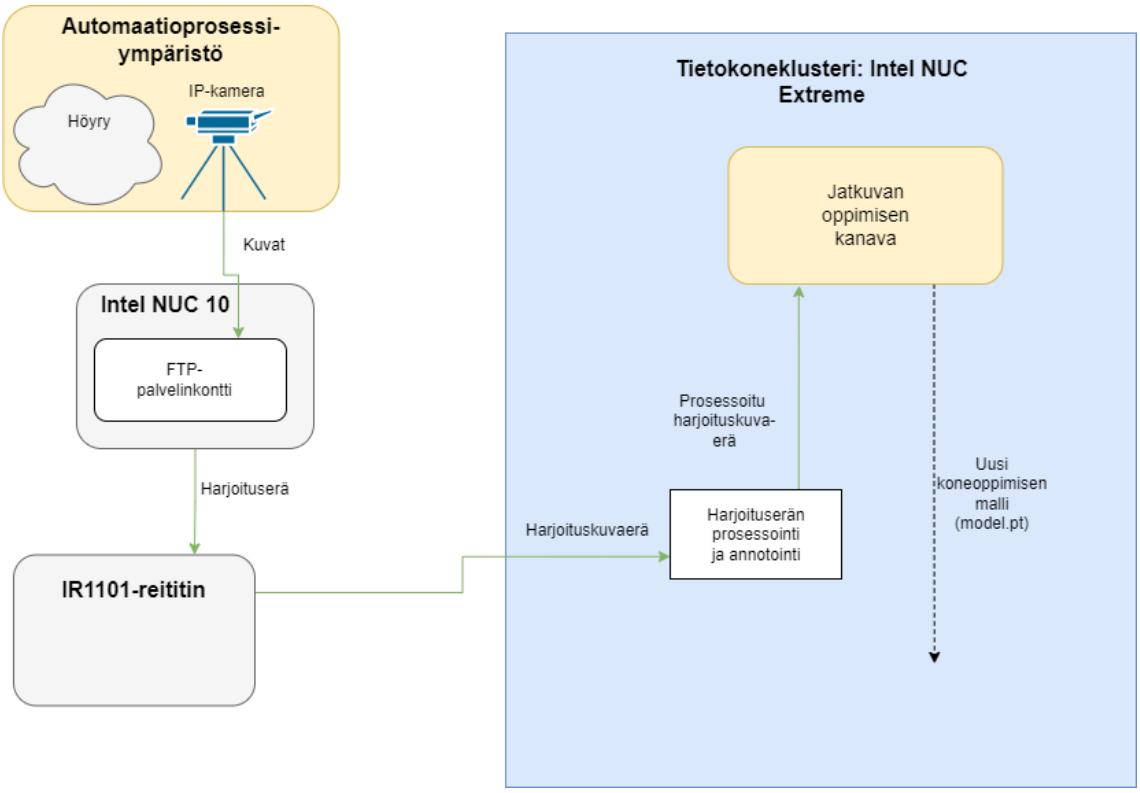
Tässä työssä reunalaskennan testilaitteena käytetään Cisco Catalyst IR1101 -teollisuusreititintä, jolla on seuraavanlaisia teknisiä ominaisuuksia. IR1101 on teollisuuskäyttöön suunniteltu reititin, jossa on Modular LTE, FirstNet ja 5G-verkkovalmiudet. Lisäksi IR1101 sisältää neljä 100Mbit FastEthernet-porttia ja 1Gb GigabitEthernet-portteja, joilla tieto voidaan siirtää myös tiedonsiirtokaapeleita pitkin. IR1101 tukee reunalaskentaa ja siinä on erittäin turvallinen ja joustava Linux-pohjainen Ciscon käyttöjärjestelmä, IOS XE, joka sisältää Docker Daemonin nimeltä dockerd. Näin ollen IR1101 mahdollistaa Docker-konttien ajamisen itse reitittimellä. Lisäksi tutkimuslaitteiston IR1101 sisältää lisämoduulin, jossa on mSATA 100GB -kovalevy. Lisäksi IR1101:ssä on neljä gigatavua sekä DRAM että flash-muistia ja ARM64v8-arkkitehtuurin prosessori. [58]

Ciscon IOS XE käyttöjärjestelmä tarjoaa alustan sovellusten isännöinnille ja alusta toimii SaaS-periaatteella. IR1101:n Hypervisoria hyödyntävä IOx-arkkitehtuuri tarjoaa kyvykkyyden ajaa Docker-ohjelmistolla toteutettuja konttiin laitettuja sovelluksia. IOS XE mahdollistaa sovellusten käyttöönoton eli sovellusten ajon ja hallinnoinnin komentorivillä, CLI, tai verkkokäyttöliittymän avulla. Sovellusten isännöinnin tarjoamia palveluita ovat määrättyjen sovellusten käynnistäminen konteissa, käytettävissä olevien resurssien hallinta ja allokointi, tuki kirjautumiseen konttien konsoleihin, pääsy palveluihin REST API:n avulla, sovellusten isännöinnin mahdollistava infrastruktuuri CAF ja alustariippuvaisen verkkotyöskentelyn pystyttäminen VirtualPortGroup- ja hallintarajapintojen avulla. [59]

Tietokoneklusterina tässä tutkimuksessa käytetään Mac Studio -klusteria, jossa tietokoneilla on monta prosessoriydintä ja useampiytiminen näytönohjain Neural Engine-ytimillä, jotka on optimoitu kompleksisten neuroverkkojen suorittamiseen. Mac Studio käyttää alustana koneoppimisessa MPS-kehystä (eng. Metal Performance Shaders), joka on optimoitu GPU-laskentaan tarjoamalla siihen grafiikkavarjostimia ja muita ominaisuuksia. MPS:n voisi ajatella karkeasti olevan vastine Nvidian CUDA-alustalle. [69,70]

Jos neuroverkkoa harjoitettaessa opetuskuvia on suuri määrä, niin yleisesti ottaen näytönohjaimen GPU pienentää harjoitukseen kuluvaa aikaa verrattuna CPU:n laskenta-aikaan. Paljonko nopeampi GPU on, riippuu harjoituskuvien määrästä, harjoitettavasta koneoppimisen mallista ja käytetystä CPU- ja GPU-mallista, mutta joidenkin tutkimusten [8,13] mukaan GPU kykenee suorittamaan syväoppimista alaspäin arvioiden noin neljästä joihinkin kymmeniin ehkä jopa sata kertaa nopeammin kuin CPU.

Kuvassa 18 on testilaitteiston pohjapiirros, jossa höyryntunnistuskamera ottaa kuvia automaatioprosessista vapautuvasta höyrystä tai kuvia, joissa ei esiinny höyryä. Kuvat kamerasta siirretään Intel NUC 10 -minitietokoneelle, josta FTP-palvelimelta kuvat ohjataan reitittimen kautta kauempana järjestelmässä sijaitsevaan tietokoneklusteriin harjoituserän esikäsittelyä varten. Kun harjoituskuvaerä on käsitelty, prosessoidut kuvat viedään jatkuvan oppimisen kanavaan, jossa harjoitetaan YOLOv8n-malli tunnistamaan höyryä kuvista. Jatkuvan oppimisen kanava sijaitsee vain tietokoneklusterissa, eikä pilvipalvelussa, koska jatkuvan oppimisen suorittamisessa sovelletaan Best Effort -reaaliaikavaatimusta. Tällöin uuden mallin harjoittamiselle ei ole erityisvaatimuksia treenausajan suhteen. Tietokoneklusterissa on riittävästi laskentatehoa, jotta laskeminen voidaan suorittaa siellä, eikä erilliselle pilvipalvelulle ole tarvetta. Lisäksi yksityisessä klusterissa on parempi tietoturva automaatiojärjestelmälle.



Kuva 18: Testilaitteiston pohjapiirustus

Kuvassa 18 Jatkuvan oppimisen kanavassa harjoitetaan koneoppimismalli ja testataan mallin tarkkuutta. Uusi koneoppimisen malli viedään joko reitittimellä tai tietokoneklusterissa olevaan inferenssikonttiin. Testit höyryntunnistuksessa tehdään IR1101-reitittimellä reunalaskennassa ja sumulaskennassa tietokoneklusterissa. Molemmille laitteille tehdään inferenssi-kontti Docker-ohjelmistolla ja inferenssiin käytetään harjoitettua Yolov8n-mallia. Koneoppimisen mallia käytetään Ultralytics-kirjaston avulla, tiedonsiirto suoritetaan http-protokollan avulla, päätepisteet tehdään FastAPI-kirjastolla ja palvelin toteutetaan Uvicorn-palvelimena. Päätöksentekopalvelimeen voidaan siirtää tietoa ja pyytää siltä ennustetta höyryntunnistuskuville. Inferenssi suoritetaan käyttämällä CPU-prosessoria IR1101:lla.

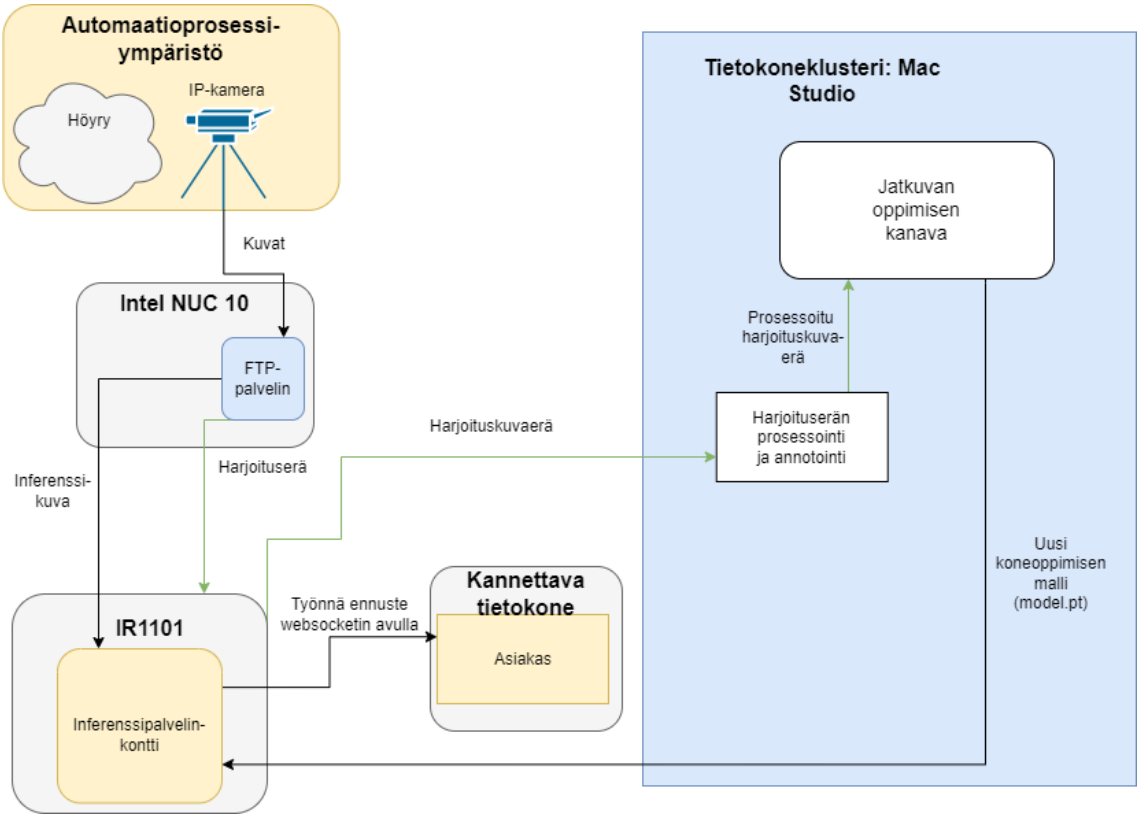
5.2 Testisuunnitelma

Prosessiautomaatioympäristössä tapahtuvassa höyryntunnistuksessa tulee ottaa huomioon, että koneoppimisen avulla suoritettu aktiivinen kohteentunnistus uusista kuvista tulee olla reaaliaikaista. Prosessiautomaatiossa pehmeä reaaliaikavaatimus on **muutamasta muutamaan sataan millisekuntia** riippuen järjestelmästä. Koneoppimispohjainen höyryntunnistus tulisi noudattaa pehmeää reaaliaikaisuutta. Tällöin esimerkiksi voisi koneoppimisen avulla tuotettua tietoa hyödyntää reaaliaikaisessa prosessin valvonnassa. Automaation reaaliaikavaatimuksen vuoksi itse höyryntunnistusta tutkitaan reunalaskennan laitteella lähellä prosessia. Reunalaskentatunnistusta verrataan sumulaskennan, tässä tapauksessa tietokoneklusterilla tapahtuvaan, höyryntunnistamiseen, joka suoritetaan reunalaskentatunnistukseen verrattuna yleensä kauempana prosessista.

Kokeessa viivemittaus suoritetaan kymmenen kertaa, jotta saadaan sekä tiedonsiirrosta että kohteentunnistuksesta aiheutuvalle viiveille keskiarvo ja keskihajonta. Lisäksi useamman mittauksen perusteella voidaan tutkia, paljonko tiedonsiirron ja tunnistuksen viive vaihtelevat näiden mittausten välillä, ja tunnistaa mahdollisia poikkeamia järjestelmän toiminnassa. Samalla joko IR1101-reitittimellä tai tietokoneklusterilla tapahtuvan inferenssiin kuluva aika ja viive mitataan. Oikea-aikaisuus koostuu datan keräämisestä, arvioinnista ja tiedon perusteella reagoimisesta. Tutkittavaa konstruktiota käytetään automaatioprosessin valvonnassa, jolloin höyryntunnistuksen tuottaman tiedon jatkokäyttö on asiakkaan tai käyttäjän vastuulla. Näin ollen oikea-aikaisuuden tutkimisessa keskitytään tiedon keräämisen ja arvioinnin oikea-aikaisuuteen. Viiveellä mitataan palvelimelta ennusteen siirtämiseen asiakkaalle kuluvaa aikaa, joka ilmaisee höyryntunnistuksen oikea-aikaisuutta asiakkaan näkökulmasta. Lisäksi tiedonsiirrossa tapahtuvat häiriöt ja vii-

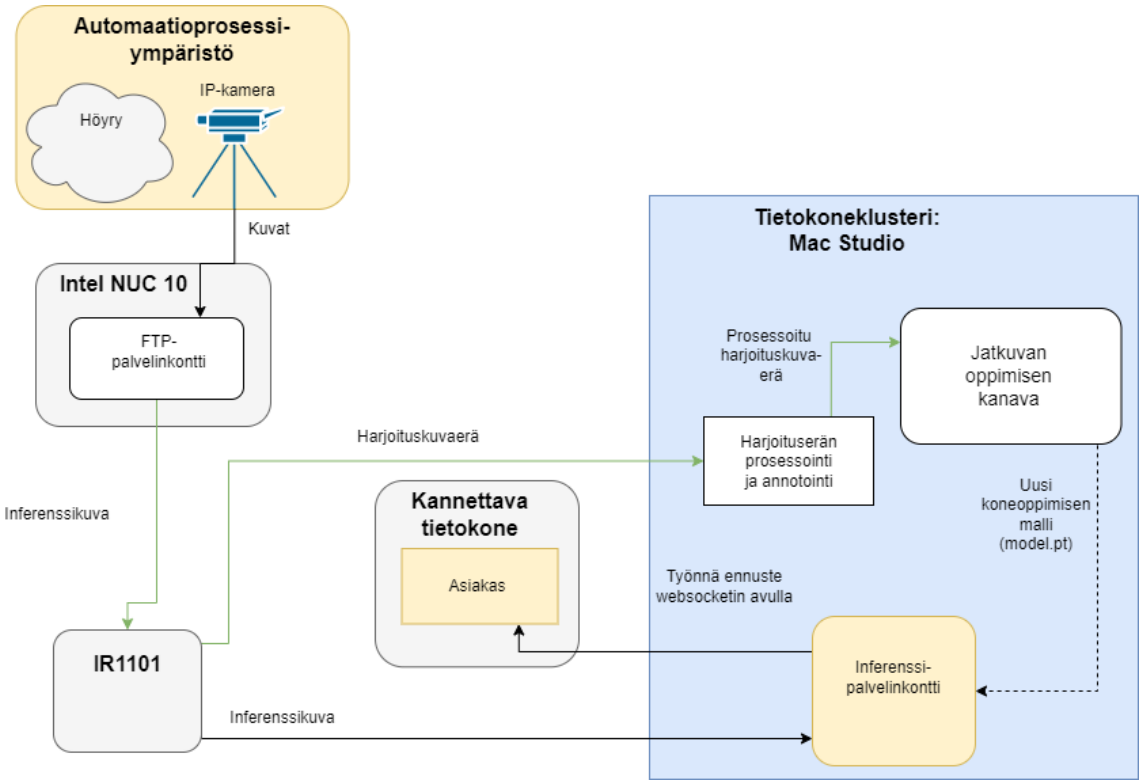
veen vaihtelut mitataan tiedonsiirron viiveen keskihajontana, jotta voidaan arvioida järjestelmän luotettavuutta. Järjestelmän samanaikaisuutta tutkitaan tiedonsiirron- ja inferenssiviiveen avulla epäsuorasti, sillä samanaikainen tiedonsiirto inferenssikonttiin voi aiheuttaa näihin kahteen arvoon pidempää viivettä, mikäli järjestelmän pitää odottaa kuvan tai mallin päivittämistä ennen kuin järjestelmä voi lähettää asiakkaalle tunnistuksen tulokset.

Kuvassa 19 on esitetty, miten höyryntunnistus toteutetaan reunalaskennan laitteella. Mittauksen kannalta oleellinen viesti kuvassa on, että inferenssiviivettä mitataan reunalaskentalaitteella, paljonko aikaa kuluu, kun sovellus suorittaa höyryntunnistuksen ja tiedonsiirtoviive muodostuu, kun palvelin palauttaa asiakkaalle ennusteen ja ajan, joka inferenssin suorittamiseen kului. Lisäksi aktiivista höyryntunnistusta varten inferenssikonttista käsin haetaan FTP-palvelimelta uusin kuva, jolle höyryntunnistus suoritetaan.



Kuva 19: Höyryntunnistus reunalaskentana

Kuvassa 19 olevan järjestelmän samanaikaisuus eli miten järjestelmä reagoi, kun jatkuvan oppimisen kanava tuottaa uuden harjoitetun mallin ja järjestelmän tulisi samanaikaisesti kyetä tuottamaan asiakkaalle ennuste. Samanaikaisuutta tässä tilanteessa voidaan tutkia vaikutuksena inferenssiaikaan ja -viiveeseen.

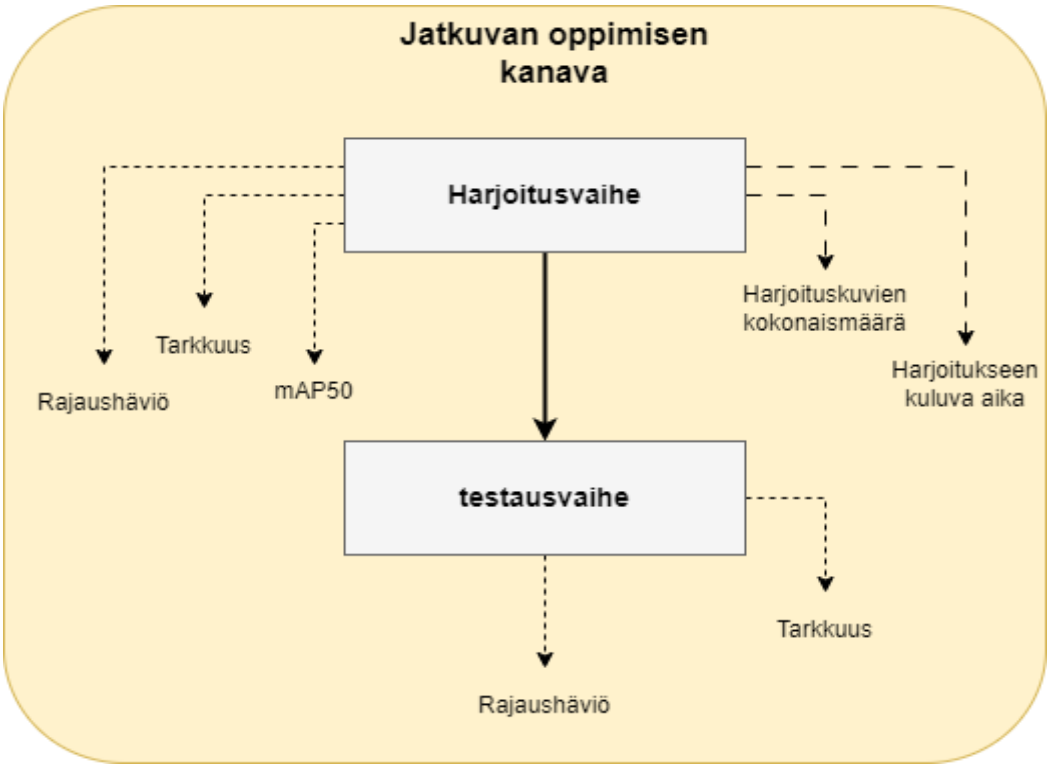


Kuva 20: Höyryntunnistus sumulaskentana

Kuvassa 20 on esitetty järjestelmä kaksi, jossa höyryntunnistus suoritetaan sumulaskentana tietokoneklusterilla. Tässä järjestelmässä tiedonsiirron viive ja inferenssiaika mitataan samoin kuten järjestelmässä yksi, jossa inferenssi suoritetaan reunalaskentana. Järjestelmän samanaikaisuutta tutkitaan samoin kuin järjestelmässä, jossa päätöksenteko suoritetaan reunalaskentana.

Toisena tutkimuskysymyksenä on, mitä hyötyjä ja haasteita on jatkuvan oppimisen hyödyntämisestä koneoppimisessa. Jatkuvan oppimisen avulla tulisi kyetä oletettavasti parantamaan höyryntunnistustarkkuutta, kun opetuskuvien määrä lisääntyy. Toisaalta opetuskuvien määrä vaikuttaa opetukseen kuluvaan aikaan. Pitkä harjoittamiseen kuluva aika, muistinkäyttö, tallennustila ja suoritintyyppi voivat rajoittaa sitä, missä laitteilla jatkuvaa oppimista kannattaa tai voi suorittaa. Suunnitelmana on mitata, mikä on mallin kohteentunnistustarkkuus harjoituskuvajoukon kokoa kohti. Kuvanjoukon koko ilmaistaan 640x640-resoluutioisten kuvien lukumääränä. Mallin kohteentunnistustarkkuutta mitataan höyryä rajaavien suorakaiteiden sijoittelun häviönä (eng. box loss), tarkkuutena (eng. precision) ja mAP50-metriikalla. Tarkkuus tarkoittaa osuutta kohteista, jotka saatiin luokiteltua oikein kaikista kohteiden todellisista luokista. Mallin tarkkuuttakin tarkastellaan erikokoisilla kuvajoukoilla, toisin sanoen, tietyn kokoinen harjoituskuvakuvajoukko

mahdollistaa tietyn tarkkuuden. Opetuskausien lukumäärä yleisesti ottaen riippuu kohteentunnistukseen käytettävästä mallista, opetusdatasta ja sovelluksesta mihin kohteentunnistusta käytetään. Näin ollen mielekäs opetuskausien lukumäärä on parametri, joka pitää määrittää kokeellisesti. Tässä järjestelmässä opetuskausien lukumääräksi valittiin 30, mikä ei välttämättä ole optimaalinen. Harjoituskuvien valinta pohjautui siihen, että valmiiksi opetettu YOLOv8 kykenee tunnistamaan metsäpaloja, joissa esiintyy savua, mikä osin muistuttaa höyryä. Siksi opetuskausien lukumääräksi valittiin vähäinen 30, jotta mallin opetuksessa ei tapahtuisi mallin parametrien ylisovittamista. Oppimisnopeudeksi asetettiin 0,001, optimoijaksi AdamW ja harjoituserän kooksi 64, jotta harjoitus olisi nopeampaa kuin pienemmällä harjoituserällä, mutta samalla muistinkäyttö ei ylittäisi laskentalaitteen resursseja.



Kuva 21: CT-kanavan mittaukset

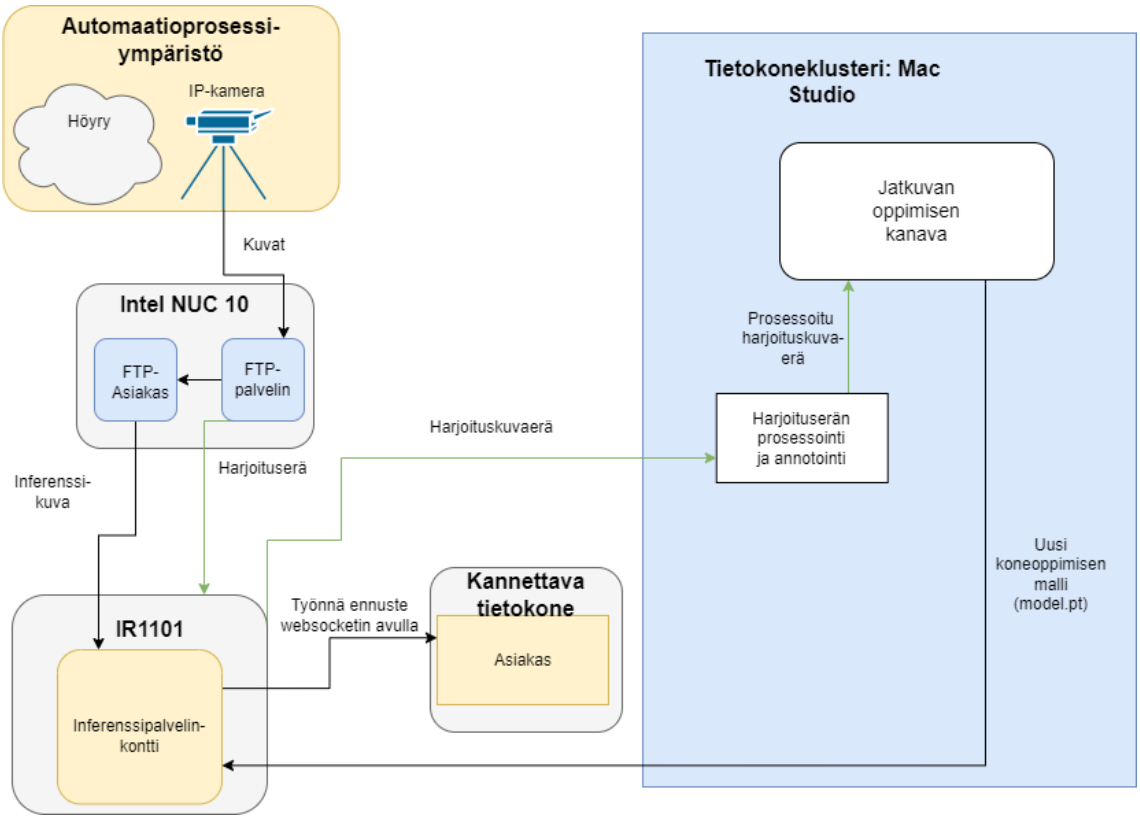
Kuvassa 21 on esitetty mistä CT-kanavan vaiheista mikäkin arvo mitataan. Ensin harjoitusvaiheesta mitataan harjoituksessa käytettävien kuvien lukumäärä ja harjoitukseen kulunut aika. Harjoitusvaiheessa mitataan myös mallin tarkkuus, mAP50-metriikka ja kohteen rajaavien suorakaiteiden häviö. Kuvassa 21 keskellä paksumpi musta nuoli tarkoittaa siirtymää harjoitusvaiheesta testivaiheeseen, kun kanavassa edetään. Harjoitetun mallin testausvaiheessa tutkitaan YOLOv8n-mallin tarkkuutta mallille entuudestaan tuntemattomilla kuvilla ja tästä saadaan tarkkuus sekä suorakaiteiden rajaushäviö mallin testausvaiheessa.

6. HAVAINNOT

Tutkimuksessa kerättiin havaintoja, jotka ilmenivät konstruktion rakentamisen sekä mittaamisen aikana. Konstruktion rakentamisessa ilmeni myös käytännön ongelma, joka ratkaistiin päivittämällä konstruktiota. Mitattaessa havainnoitiin, miten reunalaskentana toteutettu aktiivinen höyryntunnistus IR1101-reitittimellä, tutkimustilanteessa yksi, eroaa toiminnaltaan, tutkimustilanteesta kaksi, jossa aktiivinen höyryntunnistus suoritettiin Mac Studio -tietokoneklusterilla. Lisäksi havaintoja kerättiin koneoppimismallin opetus- ja testausvaiheista.

Alun perin suunniteltu tapa toteuttaa höyryntunnistusta oli kuvassa 19 esitetty tapa, jossa IR1101-teollisuusreitittimelle tehdään Uvicorn-palvelin, jonne FTP-palvelimelta haetaan senhetkinen uusin kuva. Sitten uusimasta kuvasta tunnistetaan höyry rajaamalla se suorakaiteella, mikäli kuvassa ilmenee höyryä. Alun perin oli suunniteltu, että FTP-asiakas hakee uusimman kuvan höyryntunnistuspalvelimelta, mikä ei ollut käytännössä toteutuskelpoinen tapa. FTP-palvelimen ja sen asiakkaan välillä kommunikointi passiivisessa moodissa tapahtuu siten, että FTP-asiakas määrittää komento- sekä datakanavan porttien numerot. Komentokanavan kautta palvelin ja asiakas jakavat keskenään komentoja ja oletusportti komentokanavalle on 21. Kuitenkin työssä käytetty Pythonin ftpdlib-kirjaston avulla toteutettu FTP-asiakas määrittää datakanavaksi jonkin portin numeron satunnaisesti porttialueelta 30000-65535.

Kun höyryntunnistuspalvelinkontin FTP-yhteys katkeaa tai kontti käynnistetään uudelleen, valitsee asiakas uuden portin numeron satunnaisesti samalta porttialueelta. Tästä seurasi käytännön ongelma, sillä IR1101:lla toimivalle höyryntunnistuspalvelimelle pitää määrittää erikseen, mistä portista reitittimeltä voi liikennöidä ulospäin, reitittimeltä muuhun verkkoon. Konttia rakennettaessa pitää myös ioxclientillä kapseloidulle sovellukselle määrittää portin numero, milloin usealle portille IOx-sovelluksen rakennus kestää todella kauan, eikä toiseksi ole tietoturvallista sallia liikennettä niin laajalta porttialueelta. Ongelma ratkaistiin kuvassa 22 olevalla tavalla siirtämällä FTP-asiakas kontista Intel NUC 10:lle, jossa oli mahdollista avata tarvittava datakanavan satunnainen portti dynaamisesti. Sen jälkeen voidaan lähettää uusin kuva HTTPS-pyyntönä höyryntunnistuspalvelimelle päivitettyyn REST-päätepisteeseen, jonka jälkeen kuvalle suoritetaan tunnistus. Kannettavalle tietokoneelle työnnetään, Websocketin avulla, uudessakin ratkaisussa höyrytunnistuksen tulokset.



Kuva 22: Höyryntunnistus reunalaskentana päivitetty järjestelmä

Aktiivisen höyryntunnistuksen päivitetyssä reunalaskentatoteutuksen kokeissa ilmeni, että asiakaskonetta lähellä sijaitsevalta reitittimeltä työnnettävällä ennusteella on hyvin pieni tiedonsiirron viive. Lisäksi sama ennuste saapuu asiakkaalle useita kertoja peräkkäin reaaliajassa, koska IP-kameralta kuva saapuu harvemmin FTP-palvelimelle kuin ennuste siirretään asiakkaalle. Tällöin ennuste päivittyy vasta, kun uusin kuva FTP-palvelimelta siirretään höyryntunnistuspalvelimelle analysoitavaksi. Tämän jälkeen uusi ennuste päivittyy vasta asiakkaalle. Sama pätee sekä IR1101:llä reuna- että sumulaskentana toteutettuun höyryntunnistukseen. Lisäksi asiakkaalle Websocketilla siirretyn ennusteen viive pysyy alle millisekunnissa lukuun ottamatta hetkittäisiä häiriöitä. Tutkittaessa sumulaskentana suoritettua höyrytunnistusta havaittiin pidemmän tiedonsiirtomatkan vuoksi enemmän viivettä kuin reunalaskennassa, mutta tunnistuksen suorittamiseen kuluva aika on vastaavasti pienempi, koska tunnistus voidaan suorittaa näytönohjaimella.

Koneoppimismallia opetettaessa havaittiin, mitä enemmän opetuskausia mallia opetettiin, sitä tarkemmaksi koneoppimismalli harjoitusdatalla tarkentui. Toisaalta opetusdatalla optimoidut mallin parametrit voivat kuitenkin johtaa niiden ylisovittamiseen. Harjoitusvaiheessa mallin tunnistustarkkuus voi olla todella hyvä, mutta testausvaiheessa tun-

temattomilla kuvilla tarkkuus voi olla huono. Tällöin malli ei kykene yleistämään tunnistusta opetusvaiheen kuvista poikkeavaan dataan. Kuitenkin opetuskausien välillä oli havaittavissa oskillaatiota mallin tarkkuudessa. Esimerkiksi rajaushäviö saattoi kasvaa kahden kauden opetuksen välillä ja tarkkuusmetriikka saattoi pienentyä tai päinvastoin. Havaittiin, että opetusnopeusparametrin muuttaminen vähensi opetuskausien välillä tapahtuvaa tarkkuuden värähtelyä. Pienempi oppimisnopeus vähensi tarkkuuden vaihtelua, mutta liian pieni oppimisnopeus voi johtaa mallin parametrien optimoinnissa siihen, että tarkkuus- ja häviömetriikat jäävät vaihtelemaan paikallisen minimikohdan ympärillä, tällöin parempi ratkaisu voi jäädä löytymättä. Lisäksi havaittiin, että opetuksessa käytettävällä optimoijalla on vaikutusta testivaiheen tarkkuuteen. Ultralytics tarjoaa useampia opetuksessa käytettäviä optimoijia, joista tutkimuksessa kokeiltiin neljää vaihtoehtoa: SGD (eng. Stochastic gradient descent), Adam, AdamW ja RMSProp. Näistä neljästä vaihtoehdosta höyryntunnistuksessa ja tässä tutkimuksessa käytetyille kuville parhaat tarkkuudet ja pienimmät häviöt tuottivat optimoija AdamW. Toiseksi parhaat tulokset saatiin SGD-optimoijalla.

Opetettaessa Ultralytics-kirjastolla jo opetettua mallia uudelleen, ensimmäisillä opetuskausilla opetustarkkuus on pienempi ja häviöt ovat suurempia kuin edellisen opetuskerran lopussa. Tämä johtuu osittain siitä, että opetettaessa YOLO-mallia ensimmäiset kolme opetuskautta ovat ikään kuin lämmittelykierroksia ja vasta niiden jälkeen mallin tarkkuus alkaa kasvaa opetuskausien edetessä.

7. TULOKSET

Tutkimuksessa rakennetulla konstruktiolla ensin pyrittiin saamaan vastauksia tutkimuskysymykseen, mitä palvelunlaadullisia eroja on, pehmeä reaaliaikaisuuden näkökulmasta, ajantasaisen höyryntunnistuksen suorittamisessa reuna- tai sumulaskentalaitteella. Taulukossa 1 on esitetty höyryntunnistuksen mittaustulokset reunalaskentalaitteella eli IR1101-teollisuusreitittimellä, jonka kanssa samassa huoneessa sijaitsi asiakastietokone tai kannettava tietokone, jolla valvotaan osaprosessia. Toiseksi höyryntunnistus sumulaskentana suoritettiin eri huoneessa ennusteen Websocketilla vastaanottavasta asiakastietokoneesta, ja näin ollen ennuste siirrettiin siis kauempaa verkosta asiakkaalle. IR1101-reitittimellä ARM64v8-arkkitehtuurin prosessorilla suoritettujen mittausten kymmenen mittauksen keskiarvo oli yli viisi sekuntia ja kymmenen mittauksen keskihajonta oli 777,6 millisekuntia. Kun höyryntunnistus suoritettiin Mac Studio -tietokoneklusterilla, kymmenen mittauksen keskiarvo oli 11,6 millisekuntia ja keskihajonta 0,5 millisekuntia.

Taulukko 1: Höyryntunnistus reuna- ja sumulaskentana

	Reunalaskentana	**Sumulaskentana**
Tunnistusajan keskiarvo (ms)	5307,7	11,6
Tunnistukseen kuluneen ajan keskihajonta (ms)	777,6	0,5
Tiedonsiirron-viiveen keskiarvo (ms)	1,1	322,9
Tiedonsiirron-viiveen keski-hajonta (ms)	1,4	19,0

Mittauksista ilmeni, että höyryntunnistus sumulaskentana on noin 458 kertaa nopeampaa keskiarvoisesti kuin sama suoritettuna reunalaskentana IR1101-reitittimellä rajattuna tässä tutkimuksessa käytetyillä teknologioilla ja koneoppimisen kirjastoilla. Lisäksi höyryntunnistukseen kuluneen ajan keskihajonnasta huomataan, että laskenta-ajan todennäköinen vaihteluväli on tällä kyseisellä reunalaskennan laitteella paljon suurempi kuin sumulaskennassa. Tästä voidaan päätellä, että ainakin IR1101-reitittimellä höyryntunnistuksessa on mittaamisen aikana tapahtunut häiriö laskennassa, sillä yhdeksän mittausta kymmenestä ovat ajallisesti olleet lähempänä viittä sekuntia kuin yksi poikkeava mittaus, jonka arvo oli 7,4 sekuntia. Yhdeksän muun mittauksen keskihajonta on 289,6 millisekuntia, mikä on silti selvästi suurempi kuin sumulaskennassa. Näin ollen reunalaskentaa IR1101-reitittimellä voidaan pitää laskenta-ajan tasaisuudessa epäluotettavampana vaihtoehtona kuin sumulaskentaa Mac Studiolla. Useammalla mittauksella voitaisiin kuitenkin saada selvyyttä siihen, että esiintyykö reunalaskennassa useammin suurempia häiriöitä tunnistusajassa kuin sumulaskennassa.

Sumulaskennassa, tässä tutkimuksessa, tiedonsiirron viive on 288 kertaa suurempi kuin reunalaskennassa. Kuitenkin tutkimuksissa [x2,x3] tiedonsiirron viiveeksi sumulaskennassa on mitattu joistain kymmenistä sataan millisekuntia. Näin ollen tässä tutkimuksessa mitattu viive on paljon ainakin kolme kertaa suurempi. Syynä tälle voi olla se, ettei testilaiston välistä tiedonsiirtokanavaa satu eristettyä riittävästi muussa käytössä olevasta verkosta tai laitteiden varsinkin tietokoneklusterin muut sovellukset vaikuttivat tiedonsiirtoviiveeseen. Tällainen tilanne on hyvin todennäköistä myös reaalimaailman monimutkaisissa verkoissa, joiden osana hyödynnetään sumulaskentaa. Silti sumulaskentaklusteri sijaitsi verrattain lähellä reunalaskennan laitetta ja käytännössä sumupilvi voisi kauempanakin, milloin tiedonsiirtoviive voisi olla pidempikin tietenkin riippuen myös tiedonsiirtoreitistä. Reunalaskennassa tiedonsiirron keskihajonta on suuri suhteessa tiedonsiirtoaikaan, kun taas sumulaskennassa tiedonsiirtoajan keskihajonta on absoluuttisesti suurempi kuin reunalaskennassa.

Pehmeälle reaaliaikaisuudelle ei ole tarkkaa aikarajaa, mutta suuntaa antavana referenssiaikana voidaan pitää muutamasta muutamaan sataan millisekuntia. Reunalaskenta IR1101-reitittimellä ei sovellu pehmeää reaaliaikasuutta vaativaan höyryntunnistukseen, kun käytetään tässä tutkimuksessa esitettyjä koneoppimisen kirjastoja ja mallia. Laskenta-aika jo itsessään ylittää selkeästi suuntaa antavan pehmeän reaaliaikaisuuden rajan. Toisaalta reunalaskenta pelkän tiedonsiirtoajan näkökulmasta soveltuisi käytettäväksi osana pehmeää reaaliaikaisuutta vaativissa valvontajärjestelmissä, mutta ei höyryntunnistussovelluksen kanssa kokonaisuutena. Sumulaskenta tällaisenaan voi soveltua rajoitetusti käytettäväksi pehmeää reaaliaikaa vaativissa valvontajärjestelmissä,

joissa laskenta- ja tiedonsiirtoajan summaa 334,5 millisekuntia voidaan pitää sopivana keskiviiveenä, josta satunnainen poikkeaminen ei aiheuta turvallisuusriskejä. Kuitenkin kannattaa huomioida sumulaskennan soveltamisessa, että pidempi tiedonsiirtomatka voi aiheuttaa lisää viivettä järjestelmään.

Toiseksi konstruktiolla pyrittiin saamaan vastaus toiseen tutkimuskysymykseen, mitä hyötyjä ja haasteita on jatkuvan oppimisen hyödyntämisessä koneoppimisessa. Taulukossa 2 on esitetty koneoppimisen mallin harjoitusvaiheesta kerättyjä metriikoita. Harjoituksen ensimmäisellä opetuskierroksella käytettiin pohjana valmista yolov8n.pt-mallia ja ensimmäisen harjoituskierroksen jälkeen harjoitettu malli ja sen parametrit tallennettiin tiedostoon model.pt. Tämän jälkeen tätä kerran harjoitettua mallia model.pt harjoitettiin uudelleen toisella opetuskierroksella ja tämä toistettiin myös opetuskierroksella kolme. Taulukossa 2 on esitetty myös metriikat tilanteessa, jossa kaikkia 1302:ta kuvaa käytettiin opetuksessa yhdellä kertaa.

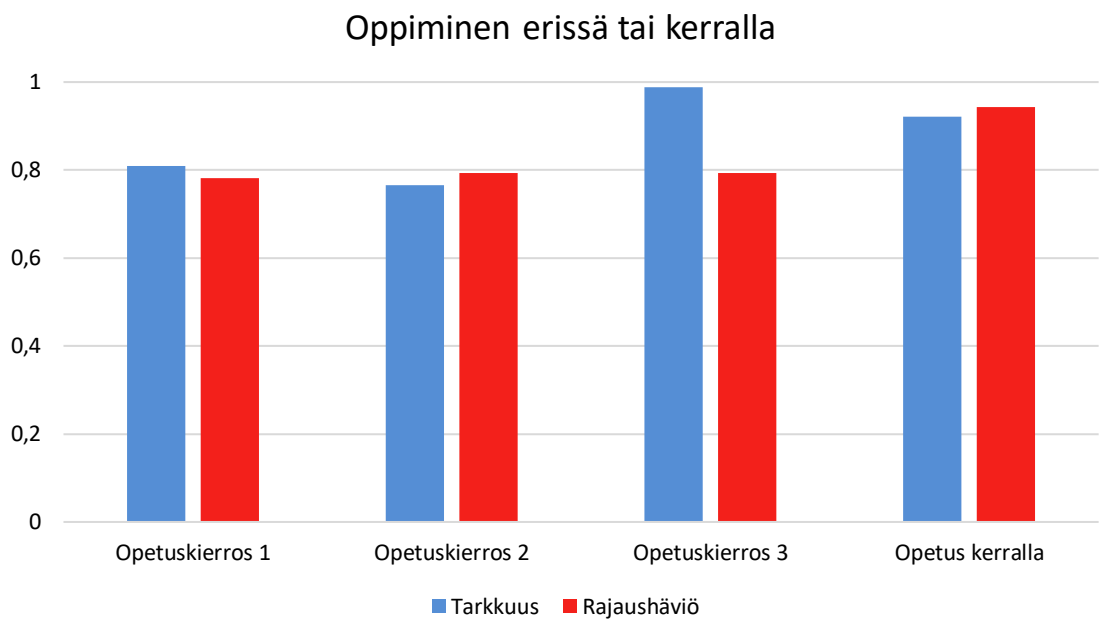
Taulukko 2: Mallin tunnistustarkkuusmetriikoita harjoitusvaiheessa

	Harjoituskuvien määrä	**Harjoitukseen kulunut aika (s)**	**Harjoituskausien lukumäärä**	**Suorakaiteiden rajaushäviö**	**Tarkkuus**	**mAP50**
Ensimmäinen opetuskierros	488	181,71	30	0,56	0,81	0,781
Toinen opetuskierros	407	162,00	30	0,62	0,77	0,79
Kolmas opetuskierros	407	158,25	30	0,63	0,61	0,47
Opetus kerralla	1302	236,80	30	0,58	0,913	0,925

Taulukossa 2 suorakaiteiden höyrynrajaushäviö kasvaa eri opetuskierrosten jälkeen, kun taas tarkkuus ja mAP50 laskevat. Tarkkuuden pieneneminen voi johtua opetuskierroksella kaksi ja kolme siitä, minkälaisia harjoituskuvia opetuskierroksella oli tai mallin ylisovittamisesta. Toisaalta taulukossa 2 viimeisellä rivillä on tulokset opetuskerralle, joka on suoritettu kaikilla kuvilla kerralla, tällöin tarkkuus on kolmanteen opetuskierrokseen nähden parempi ja rajaushäviö on pienempi kuin kolmannella opetuskierroksella. Edellisestä havainnosta voidaan päätellä, ettei Ultralytics-kirjasto suoraan mahdollista jatkuvaa oppimista siten, että mallia voitaisiin parannella pienemmillä kuvaerillä. Sen sijaan uudet kuvat ovat liitettävä osaksi aikaisempaa opetusdataa ja tällä suuremmalla kuvajoukolla pitää harjoittaa malli alusta asti uudelleen.

Mallin tarkkuus harjoitusvaiheessa yleisesti ottaen ilmaisee koneoppimismallin tunnistustarkkuuden kehittymisestä. Mutta tunnistustarkkuus testivaiheessa ilmaisee mallin kykyä yleistää oppimaansa sille entuudestaan tuntemattomille kuville. Mallin tunnistuksen yleistettävyys on tärkeää, jotta mallilla voidaan suorittaa esimerkiksi aktiivista höyryntunnistusta reunalaskentana. Varsinkaan höyry ei ole kiinteä objekti, vaan höyry on muodoltaan anomaalista, tällöin reaalimaailmassa tunnistukselta vaaditaan yleistettävyyttä.

Kuvassa 22 on esitetty koneoppimismallin tarkkuus ja höyryä rajaavien suorakaiteiden rajaushäviö. Ensimmäisellä opetuskierroksen jälkeen luokittelutarkkuus on yli 80 prosenttia ja kohteiden rajaushäviö on hieman alle 0,8. Luokittelutarkkuus pienenee toisella opetuskierroksella ja sen selittää todennäköisesti, että toisessa harjoituskuvaerässä on enemmän höyryttömiä kuvia, joten mallin tunnistaa paremmin höyryttömät kuvat, koska ne ovat yliedustettuina toisen kierroksen opetusdatassa. Kolmannen opetuskierroksen jälkeen testausvaiheessa tarkkuus kasvaa taas luultavasti sen vuoksi, että kolmannessa opetuskuvaerässä on enemmän kuvia, joissa on höyryä paljon höyryä ja etenkin sellaisia, jossa höyryä on vähän.



Kuva 22: Höyryntunnistustarkkuus mallin testausvaiheessa

Oikeanpuolimmaisimmat pylväät kuvassa 22 edustavat tarkkuutta ja höyrynrajaushäviötä tilanteessa, jossa kaikki kolmen opetuskierroksen kuvat ovat opetettu mallille kerralla. Kun otetaan huomioon sekä opetus- että testausvaiheiden tarkkuusmetriikat, kerralla opetuksessa molempien vaiheiden tarkkuusmetriikat ovat samansuuntaiset.

Jatkuvan oppimisen hyödyt koneoppimisessa on se, että yleisesti ottaen mallin tunnistustarkkuus paranee, mitä enemmän kuvia tunnistusmallille opetetaan. Lisäksi jatkuvan oppimisen hyötynä on se, että malli voidaan alustaa tunnistamaan höyryä ensin muualla, jonka jälkeen itse kohdejärjestelmästä kerätyillä kuvilla voidaan tarkentaa mallin tarkkuutta tunnistaa kyseisen kohdejärjestelmän höyryä, mikäli sen tapa tuottaa höyryä ja ympäristö, mistä kuvia kerätään, eroaa muualla alustetun mallin höyrynmuodoista ja ympäristöstä.

Molemmat opetustavat hyödynsivät Yolov8n-mallin siirto-oppimista siten, että malli opetettiin tunnistamaan höyryä. Haasteena Ultralytics-kirjaston hyödyntämisessä jatkuvassa oppimisessa on se, että uudelleenkäynnistetyllä ajantasaisella oppimisella eri harjoituskierroksittain malli unohtaa oppimansa kuvat ja toisin sanoen painottaa oppimisessaan viimeisimmän kierroksen opetuskuvia. Näin ollen oppimisessa suositellaan näistä kahdesta vaihtoehdosta sitä, että Yolov8n-malli opetetaan kerralla. Tämän sen vuoksi, ettei malli kykene uudelleenkäynnistyksellä (eng. warm-start) eli uusi opetuskierroksella käytetään edellisen opetuskierroksen painoja. Näin yritetään ehkäistä mallia unohtamasta edellisellä opetuskierroksella opittuja kuvia.

Jatkossa voisi tutkia voisiko proaktiivista oppimista käyttää Ultralytics-kirjaston kanssa jatkuvassa oppimisessa ja miten sitä kannattaisi toteuttaa ylipäätään. Proaktiivisessa oppimisessa siis opetetaan malli uusilla ja osalla vanhoista kuvista niin, että harjoituskuvien määrä ei kasva kohtuuttomaksi, mutta myös malli säilyttäisi kyvyn soveltaa aiemmin opittua. Proaktiivisessa oppimisessa voisi tutkia, kuinka monta ja miten aiemmin opittujen kuvien joukosta kannattaisi valita kuvia uudelleen opetukseen, jotta aiempi kuva joukko olisi hyvin edustettuna. Toisaalta uusien kuvien poiketessa vanhoista kuvista, pitäisi valita kuvat niin, että höyryntunnistuksen opetuksessa olisi riittävästi kuvia, jotka kuvaisivat höyryn moninaisuutta myös kattavasti.

8. YHTEENVETO

Tässä työssä automaatio-osaprosessin valvontaa varten rakennettiin koneoppimista hyödyntävä konstruktio. Höyryntunnistusjärjestelmän tuottaman tiedon perusteella automaatio-osaprosessin toimintaa voidaan valvoa. Prosessin valvonnalta vaaditaan pehmeää reaaliaikaisuutta, jossa järjestelmästä datankerääminen, datan arviointi ja arvion lopputuloksena tuotetun tiedon siirto valvontakäyttöön tapahtuvat ennustettavasti tietyn aikavälin sisällä tai satunnaisesti sen ulkopuolella. Tämän vuoksi reunalaskentana tai sumulaskentana suoritetulta koneoppimispohjaiselta höyryntunnistamiselta vaaditaan myös pehmeää reaaliaikaisuutta, jossa viiveen pituus on muutamasta muutamaan sataan millisekuntia. Aktiivinen höyryntunnistaminen hajautettiin lähemmäs automaatioprosessia, tavoitteena parempi reaaliaikaisuus. Toisaalta höyryntunnistus malli harjoitettiin sumulaskentalaitteella kauempana automaatioprosessista ilman reaaliaikavaatimuksia. Koneoppimismallin höyryntunnistustarkkuuden parantamiseksi rakennettiin myös jatkuvan oppimisen -kanava, jolla mallia harjoitettiin uudelleen jatkuvasti järjestelmästä kerätyllä datalla.

Jatkuvasti opetettava malli ja itse höyryntunnistuksen suorittava sovellus pohjautuivat Ultralytics-kirjaston tarjoamaan YOLOv8n-malliin ja sen työkaluihin. YOLOv8n-mallin jatkuvaoppiminen suoritettiin Mac Studio -tietokoneklusterissa. Harjoituksen tuloksena saatava uusi koneoppimisen malli siirrettiin järjestelmässä höyryntunnistusmallia harjoittavalta laitteelta itse höyryä tunnistavalle laitteelle joko IR1101-reitittimelle tai toisessa tutkimustilanteessa Mac Studio -tietokoneklusterille. Rakennetun konstruktion avulla pyrittiin saamaan vastauksia tutkimuskysymyksiin. Ensimmäiseksi konstruktion avulla haluttiin saada vastaus kysymykseen mitä palvelunlaadullisia eroja on aktiivisen höyryntunnistuksen suorittamisessa reuna- tai sumulaskentalaitteella.

Tässä työssä reunalaskenta suoritettiin IR1101-teollisuusreitittimellä ja sumulaskenta Mac Studio -tietokoneklusterilla. Lisäksi selvitettiin molempien laskentalaitteiden prosessointitehon ja vaadittavan tiedonsiirron kykyä soveltua pehmeään reaaliaikaiseen valvontaan. Pehmeä reaaliaikaisuus jaoteltiin oikea- ja samanaikaisuuteen, joita tutkittiin palvelunlaatumittareiden avulla. Höyryntunnistusjärjestelmän oikea-aikaisuutta tutkittiin sekä tiedonsiirto- että prosessointiviiveiden avulla. Järjestelmä ei itsessään reagoi höyryntunnistuksen tuottaman tiedon pohjalta, siksi sitä ei huomioitu tutkimuksessa. Höyryntunnistusjärjestelmän samanaikaisuutta eli kykyä vastaanottaa tietoa useammista

lähteistä samalla suorittaen höyryntunnistusta, tutkittiin epäsuorana vaikutuksena tunnistusviiveeseen.

Yolov8n-mallilla suoritettu höyryntunnistus reunalaskentana IR1101-teollisuusreitittimellä ei sovellu tämän työn mukaan käytettäväksi pehmeää reaaliaikaa vaativassa valvonnassa, vaikka itse tiedonsiirtoviive on hyvin pienin, niin laitteen laskenta-aika on selvästi yli muutaman sadan millisekunnin. Sumulaskenta Mac Studio -klusterilla sen sijaan kykenee suorittamaan reaaliaikaista valvontaa, sillä sen laskenta-aika on kymmenen millisekunnin luokkaa ja tiedonsiirronviive on joidenkin satojen millisekuntien luokkaa. Kuitenkin täytyy todeta, että tutkimuslaitteisto pyrittiin eristämään muuhun käyttöön soveltuvasta laitteistosta, joskin ne kasvattivat tiedonsiirron viivettä ja paremmin eristettynä sumulaskennalla päästää vielä pienenpään tiedonsiirronviiveeseen. Jatkossa voisi tutkia, kuinka koneoppimisen mallia voisi keventää niin, että IR1101:llä laskenta-aikaa saataisiin pienennettyä tai sitten voisi tutkia, miten tehokkaampi reunalaskentalaite soveltuu samaan kontekstiin. Sumulaskennassa jatkotutkimuskysymyksenä voisi olla, kuinka kauas sumupilvi voidaan sijoittaa höyryntunnistuksen vastaanottajasta, jotta tiedonsiirron viive ei vielä ylitä pehmeän reaaliaikaisuuden rajaa.

Toiseksi konstruktion avulla pyrittiin saamaan vastaus kysymykseen, mitä hyötyjä ja haasteita on jatkuvan oppimisen hyödyntämisestä koneoppimismallin harjoittamisessa. Tätä tutkittiin koneoppimismallin harjoitusvaiheessa mittaamalla kohteenrajaushäviö, tunnistustarkkuus, mAP50-metriikka, harjoittamiseen kulunut aika ja harjoituskuvien sen hetkinen kokonaismäärä. Testausvaiheessa mallin tunnistustarkkuus, entuudestaan tuntemattomilla kuvilla, mitattiin mAP50- ja tarkkuusmetriikoiden avulla. Näin voitiin selvittää mallin kyky yleistää oppimaansa. Koneoppiminen toteutettiin siirto-oppimisena ja joista ensimmäisessä tavassa sovellettiin uudelleenkäynnistystä ja ajantasaista oppimista kolmen opetuskierroksen verran ja toisessa tavassa malli siirto-opetettiin kaikilla opetuskuvilla kerralla.

Jatkuvan oppimisen hyötynä koneoppimisessa on parempi kohteen tunnistustarkkuus, mutta sen toteuttaminen ajantasaisella oppimisella harjoitusvaiheessa tarkkuus laski joka opetuskierroksella, vaikka testivaiheessa tarkkuus oli parempi kuin tavassa kaksi. Syinä tähän voi olla, että harjoituskuvaerissä oli epätasaisesti höyryllisiä ja höyryttömiä, kuvia, siksi ajantasaisessa oppimisessa malli unohtaa aiemmin opitun ja testivaiheen kuvat olivat sattumalta samankaltaisia opetuskierroksen kolme kuvien kanssa. Opetettaessa kaikilla harjoituskuvilla kerralla tunnistustarkkuus oli yli 90%:ia ja suorakaiteiden rajaushäviö oli alle yksi. Jatkuvassa oppimisessa haasteena on kuitenkin, uuden mallin jatkuva opettaminen tarvitsee aina opetettaessa uudet ja aiemmat kuvat, joiden määrä

kasvaa ajan kanssa. Näin ollen harjoittamisessa myös laskenta-aika pitenee ja tallennustilantarve kasvaa. Lisäksi haasteena on, että uudelleenkäynnistetyssä ajantasaisessa oppimisessa malli unohtaa aiemmilla kierroksilla oppimansa. Jatkossa voisi tutkia voitaisiinko Ultralytics-kirjaston avulla suorittaa proaktiivista oppimista niin, että malli säilyttäisi kyvyn muistaa aiemmin opittua ja toisaalta, että harjoituskuvia ei tarvitse pidemmällä aikavälillä opettaa mielivaltaisen paljon.

LÄHTEET

[1] M. Abdurohman, H. H. Nuha, D. Perdana et al., EdgeSL: Edge-Computing Architecture on Smart Lighting Control With Distilled KNN for Optimum Processing Time, IEEEAccess, 2023

[2] S. K. Adari, S. Alla, Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure, Apress, 2021

[3] M. Aldwairi, M. Ali, M. I. Ashraf, Fog Computing Enabling Industrial Internet of Things: State-of-the-Art and Research Challenges, Sensors, MDPI, 2019, 38 p.

[4] S. Ahlawat, S. Garg, P.K. Grupta et al., On Continous Integration / Continous Delivery for Automated Deployment of Machine Learning Models using MLOps, IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), IEEE, 2021, 4 p.

[5] A. Ahmed, E. Ahmed, S. Hakak, et al., Edge Computing A survey, Elsevier, 2019, 17 p.

[6] A. Badar, C. Barth, U. Graf et al., Intelligent Edge Control with Deterministic-IP based Industrial Communication in Process Automation, IEEE, 15th International Conference on Network and Service Management (CNSM), 2019, 7 p.

[7] S. Badillo, B. Banfai, F. Birzele et al., An Introduction to Machine Learning, Clinical Pharmacology and therapeutics, Vol. 107, No. 4, 2020, 15 p.

[8] P. K. Baruah, L. R. Bonta, A. K. Reddy B., Evaluating Training Time of Inception-v3 and Resnet-50,101 Models using Tensorflow across CPU and GPU, Proceedings of the 2nd International conference on Electronics, Communication and Aerospace Technology, IEEE, 2018, 5 p.

[9] J. Berrocal, J. Díaz, R. López-Viana, J. Pérez, Edge computing: A grounded theory study, Springer, 2022

[10] S. Bhardwaj, D.-S. Kim, T. Radim et al., Reinforcement Learning based Resource Management for Fog Computing Environment: Literature Review, Challenges, and Open Issues, Journal of Communications and Networks, Vol. 24, No. 1, IEEE, 2022, 16 p.

[11] J. Bosch, H. Holmström Olson, M. M. John, Towards MLOps: A framework and Maturity, 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2021, 8 p.

[12] T. Brox, P. Fisher, O. Ronneberger, U-Net: Convolutional Networks for Biomedical Image Segmentation, Springer, 2015

[13] E. Buber, B. Diri, Performance Analysis and CPU vs GPU Comparison for Deep Learning, 6th International Conference on Control Engineering & Information Technology, IEEE, 2018, 6 p.

[14] R. Buyya, S. N. Srirama, Edge and Fog Computing: Principles and Paradigms, John Wiley & Sons, Inc., 2019, 485 p.

[15] D. Cai, Fine-Grained Classification of Low-Resolution Image, Tampere University of Technology, 2017, 64p.

[16] J. P. Campell, M. F. Chiang, R. Y. Choi et al., Introduction to Machine Learning, Neural Networks, and Deep Learning, Transl Vis Sci Technol., 2020

[17] I. Castrillo, D. Rountree, The Basics of Cloud Computing: Understanding the Fundamentals of Cloud Computing in Theory and Practise, Syngress, 2013, 155 p.

[18] H. Chen, S. Li, X. Peng et al., Indoor formaldehyde monitoring system based on fog computing, IOP Publishing, 2019

[19] Z. Chen, C. Liu, Z. Lu et al., Visualization Technology Framework of Industrial Cloud Computing, IOP Publishing, Journal of Physics, 2021, 10 p.

[20] S. Chen, M. Zhou, Evolving Container to Unikernel for Edge Computing and Application in Process Industry, Processes, MDPI, 2021, 18 p.

[21] T. Choudhury, S. Dahiya, A. Katal et al., Fog Computing: Concepts, Frameworks, and Applications, Taylor and Francis Group, CRC Press, 2023, 235 pp.

[22] M. Churuvija, A. Dawood, M. Karkee, Machine Vision-Based Crop-Load Estimation Using YOLOv8, Cornell University Library, arXiv.org, 2023, 23 p.

[23] D. M. Cordova-Esparaza, A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond, Under Review in ACM Computing Surveys, 2023, 27 p.

[24] S. Crosby, J. Garcia, W. E. Williams, Virtualization with Xen, Syngress Publishing Inc., USA, 2007, 364 p.

[25] L. Deng, Y. Shen, B. Tang et al., Hardware-Resource-Constrained Neural Architecture Search for Edge-Side Fault Diagnosis of Wind-Turbine Gearboxes, IEEE Transactions on Industrial Electronics, IEEE, 2023

[26] B. Derakhshan, A. R. Mahdiraji, I. Prapas et al., Continuous Training and Deployment of Deep Learning Models, Springer, 2021

[27] A. Dowd, Na. H. Tonekaboni, Real-Time Facial Emotion Detection Through the Use of Machine Learning and On-Edge Computing, ICMLA, IEEE, 2022

[28] A. Fabijanska, The Recursive Approach to Segmentation of Images Presenting Heat-Emitting Objects, IEEE, 2009

[29] X. Gao, J. Ma, J. Xu et al., Adaptive Progressive Continual Learning, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 44, No. 10, IEEE, 2023

[30] O. Givehchi, J. Jasperneite, H. Trsek, Cloud Computing for Industrial Automation Systems: A Comprehensive Overview, IEEE, 2013

[31] R. A. Gonzalez, K. A. Piirainen, Seeking Constructive Synergy: Design Science and the Constructive Research Approach, Konferenssiartikkelissa: Design Science at the Intersection of Physical and Virtual Design, 8th International Conference, DESRIST, Helsinki Finland, Springer, 2013, 59-72 pp.

[32] M. Goudarzi, M. Palaniswam, A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments, IEEE Transactions on Mobile Computing, IEEE, 2023

[33] L. Guang-sheng, D. Yu-nan, Research and Discussion on Image Recognition and Classification Algorithm Based on Deep Learning, 2019 International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI), IEEE, 2019, 5p.

[34] W. A. Halang, K. M. Sacha, Real-Time Systems: Implementation of Industrial Computerized Process Automation, World Scientific Publishing Company, 1992, 354 p.

[35] J. Han, S. Liu, Y. Liu et al., A Survey of Stochastic Computing Neural Networks for Machine Learning Applications, IEEE Transactions on Neural Networks and Learning Systems, Vol. 32, No. 7, 2021, 16 p.

[36] S. Haq, S. Kaiser, MD., A. S. Tosun, T. Korkmaz, Container Technologies for ARM Architecture: A Comprehensive Survey of the State-of-the-Art, IEEE Access, Vol. 10, 2022, 29 p.

[37] D. Hästbacka, F. Loimio, S. Moreschini et al., MLOps for evolvable AI intensive software systems, IEEE, International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022

[38] P. K. Jana, N. Kumari, A. Yadav, Task offloading in fog computing: A survey of algorithms and optimization techniques, Elsevier, 2022

[39] S. Jang, Y. Kim, A Fast Training Method using Bounded Continual Learning in Image Classification, IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, 2021

[40] M. Kaur, S. Kaur, A. K. Shukla, An Introduction to Machine Learning in a Nutshell, IEEE, Proceedings of the SMART-2021, 2021, 6 p.

[41] T. Kiss, J. Kovacs, R. Singh, To Offload or Not? An Analysis of Big Data Offloading Strategies from Edge to Cloud, IEEE, 2020, 7p.

[42] A. Kishor, C. Chakraborty, W. Jeberson, A Novel Fog Computing Approach for Minimization of Latency in Healthcare using Machine Learning, Unir, Special Issue on Current Trends in Intelligent Multimedia Processing Systems, 2020

[43] J. Kjällman, M. Komu, R. Morabito, Hypervisors vs. Lightweight Virtualization: a Performance Comparison, International Conference on Cloud Engineering, IEEE, 2015, 8p.

[44] Dr. S. M. Kumar, D. Majumder, A Review on Resource Allocation Methodologies in Fog/Edge Computing, 8th International Conference on Smart Structures and Systems (ICSSS), IEEE, 2022L. Lepistö, Colour and Texture Based Classification of Rock Images Using Classifier Combinations, Tampere University of Technology, Publication 593, 2006, 145 p.

[45] L. Lepistö, Colour and Texture Based Classification of Rock Images Using Classifier Combinations, Tampere University of Technology, Publication 593, 2006, 145 p.

[46] S. Li, J. Liu, Y. Liu et al., Joint Task Offloading and Resource Allocation for Accuracy-Aware Machine-Learning-Based IIoT Applications, Internet of Things Journal, Vol. 10, No. 4, IEEE, 2023, 17 p.

[47] A. Lisdorf, Cloud Computing Basics: A Non-Technical Introduction, Apress, 2021

[48] T. L. Liu, Y. Fu, W. Yan, YOLO-Tight: An Efficient Dynamic Compression Method for YOLO Object Detection Networks, ACM, 2021

[49] A. Luckow, S. Jha, K Rattan, Exploring Task Placement for Edge-to-Cloud Applications using Emulation, IEEE, 2021, 5 p.

[50] K. Lukka, Konstruktiivinen tutkimusote, saatavilla (viitattu 25.6.2023): https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/, 2001,

[51] A. Marcham, Understanding Infrastructure of Edge Computing: Concepts, Technologies and Considerations, John Wiley and Sons Ltd., 2021

[52] H. Nakahara, S. Sato, H. Yonekawa, An Object Detector based on Multiscale Sliding Window Search using a Fully Pipelined Binarized CNN on an FPGA, IEEE, 2017

[53] N. B. Ruparelia, Cloud Computing, MIT Press, 2016. 261 p.

[54] T. Sauter, Z. Pang, S. Vitturi, Real-Time Networks and Protocols for Factory Automation and Process Control Systems, Proceedings of IEEE, Vol. 107 No. 6, 2019, 5 p.

[55] O. Simeone, A Brief Introduction to Machine Learning for Engineers, Now Publisher Inc., Vol. 12, Issue 3-4, 20128

[56] X. Wu, S-T. Xu, Z-Q. Zhao, el al., Object Detection with Deep Learning: A Review, Transactions on Neural Networks and Learning Systems, Vol. 30, No. 11, IEEE, 2019, 21 p.

[57] J. Åkerberg et al., Future Industrial Networks in Process Automation: Goals, Challenges, and Future Directions, Applied Sciences, MDPI, 2021, 15 p.

[58] Cisco Catalyst IR1101 Rugged Series Router Data Sheet, saatavilla (viitattu 26.10.2023): https://www.cisco.com/c/en/us/products/collateral/routers/1101-industrial-integrated-services-router/datasheet-c78-741709.html

[59] Cisco Catalyst IR1101 Rugged Series Router Software Configuration Guide, saatavilla (viitattu 26.10.2023): https://www.cisco.com/c/en/us/td/docs/routers/access/1101/software/configuration/guide/b_IR1101config/b_IR1101config_chapter_010001.html

[60] Coco128, saatavilla (viitattu 26.10.2023): https://www.kaggle.com/datasets/ultralytics/coco128

[61] Dagshub, saatavilla (viitattu 27.10.2023) https://dagshub.com/

[62] Docker Overview, saatavilla (viitattu 26.10.2023):: https://docs.docker.com/get-started/overview/

[63] End-to-End Machine Learning Framework, saatavilla (viitattu 26.10.2023): https://pytorch.org/features/

[64] Esittely, saatavilla (viitattu 26.10.2023): https://ubuntu-fi.org/esittely/

[65] FastAPI, saatavilla (viitattu 26.10.2023): https://fastapi.tiangolo.com/

[66] Get Started with DVC, saatavilla (viitattu 26.10.2023): https://dvc.org/doc/start

[67] Intel® NUC 10 Performance kit - NUC10i7FNH, saatavilla (viitattu 26.10.2023): https://www.intel.com/content/www/us/en/products/sku/188811/intel-nuc-10-performance-kit-nuc10i7fnh/specifications.html

[68] Introduction, saatavilla (viitattu 26.10.2023): https://www.uvicorn.org/

[69] Mac Studio, saatavilla (viitattu 26.10.2023): https://www.apple.com/fi/mac-studio/specs/

[70] Metal Performance Shaders, saatavilla (viitattu 26.10.2023): https://developer.apple.com/documentation/metalperformanceshaders

[71] Open Source Platform for the Machine Learning Lifecycle, saatavilla (viitattu 26.10.2023): https://mlflow.org/

[72] Overview, saatavilla (viitattu 26.10.2023): https://docs.pydantic.dev/

[73] Roboflow Annotate, saatavilla (viitattu 26.10.2023): https://roboflow.com/annotate

[74] Ubuntu, saatavilla (viitattu 26.10.2023): https://hub.docker.com/_/ubuntu

[75] Ultralytics YOLOv8 Modes, saatavilla (viitattu 26.10.2023): https://docs.ultralytics.com/modes/

[76] What is Automation?, saatavilla (viitattu 26.10.2023): https://www.isa.org/about-isa/what-is-automation

[77] YOLOv8, saatavilla (viitattu 26.10.2023): https://docs.ultralytics.com/models/yolov8/

LIITE A: