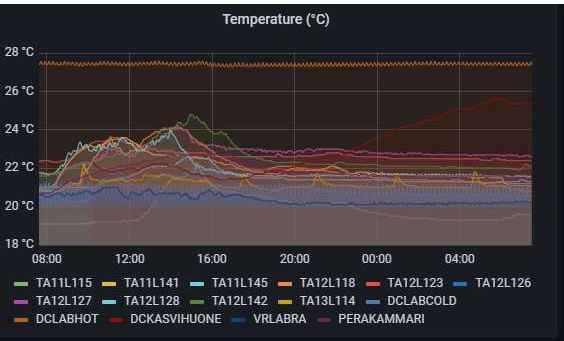
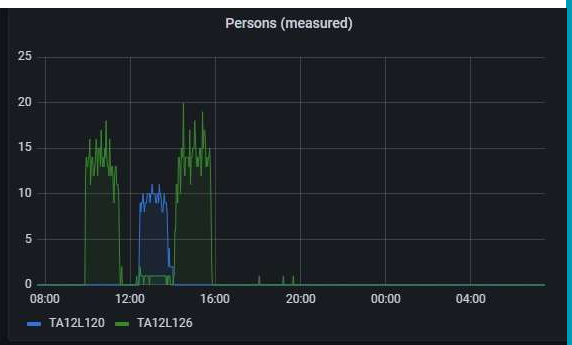
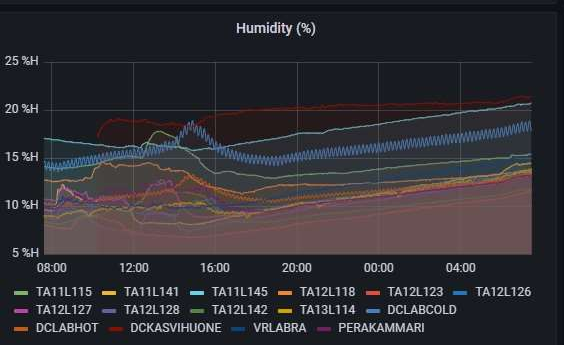
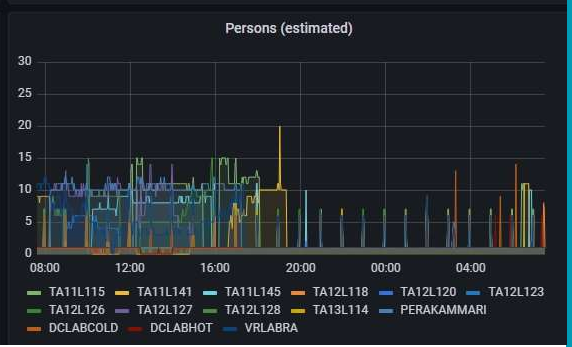
Erik Huuskonen

Ihmismäärän mittaaminen sensoridatasta









Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2023

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Huuskonen Erik

Työn nimi: Ihmismäärän mittaaminen sensoridatasta

Tutkintonimike: esim. Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: Sensoridata, konenäkö, raspberry Pi, Python, koneoppiminen, tietokanta

Opinnäytetyö käsitteli ihmismäärän laskemista sensoridatasta koneoppimismenetelmillä. Ihmismäärää yleensä arvioidaan suoraan laskemalla kameran näköpiirissä olevat ihmiset. Tämän voi automatisoida konenäöllä, mutta ratkaisu on kallis, jos ison rakennuksen jokaiseen huoneeseen pitää asentaa kamera ja pieni tietokone, joka jaksaa pyörittää ihmisentunnistusta. Niinpä ratkaisu hyödyntää huoneista kerättyä sensoridataa, jonka perusteella voisi arvioida huoneen läsnäolijoiden määrän koneoppimismetodeilla ja tekoälyllä. Tavoitteena on saada arvio ainakin n. 5 ihmisen tarkkuudelle.

Opinnäytetyön tietoperusta koostui kirjallisuus- ja verkkolähteistä. Käytännön työvaiheissa tehdyt valinnat perustuivat enemmän luovaan tutkimiseen ja testailuun kuin lopullisen tuotteen toteuttamiseen. Opinnäytetyön toiminnallisena osuutena oli matka ihmismäärädatan keräämisestä sen arviointiin koneoppimismenetelmillä. Tehty työ oli osana KAMKin 2022 SmartCampus-hanketta, joka tehtiin KAMKin kampusalueen tiloissa.

Tulokset olivat lupaavia ja ihmismäärän arviot miellyttävän tarkkoja, relatiivisen naiivista ratkaisusta riippumatta. Arviot lisättiin tietokantaa, josta ne saatiin haettua erilaisiin kojelautoihin näyttämään luokkien ihmismääränarvion. Ihmismääränarviointi laitettiin pyörimään KAMKin Bull-supertietokoneelle, jossa se toivottavasti olisi toiminnassa 24/7. Opinnäytetyössä nostettiin esiin erilaisien ratkaisujen pohtimista, reflektoitiin tutkimus- ja kehitysprosessia, ja kuvattiin mikä ei toimi ja miksi. Päätöksiä on pyritty perustelemaan tuloksien pohjalta.

Johtopäätöksenä opinnäytetyö tarjoaa haasteen aiheen jatkokehitykseen, ns. ratkottavan ongelman, joka ilmeni käytännön pohjalta: Domain-sovitusongelma on suhteellisen vähän tunnettu koneoppimisongelma, johon tässä työssä törmättiin. Sen pääidea on ratkaista, miten erilaiset ympäristöt voidaan yleistää yhdelle koneoppimismallille.

KAMK • University of Applied Sciences

Abstract

Author(s): Huuskonen Erik

Title of the Publication: Deriving Occupancy from Sensor Data

Degree Title: Bachelor of Engineering, information engineering

Keywords: Sensor data, machine vision, raspberry Pi, Python, machine learning, database

The thesis dealt with calculating the number of people from sensor data using machine learning methods. Typically, the number of people is estimated directly by counting the people in a camera's field of view. This can be automated with computer vision, but the solution is expensive if you must install a camera and a small computer in every room of a large building that also has to handle human recognition. Therefore, the solution utilizes sensor data collected from the rooms, based on which the number of people present in the room could be estimated using machine learning methods and artificial intelligence. The goal is to get an estimate of occupants to the accuracy of at least approx. 5 people.

The basis of the thesis consisted of literature and online sources. The choices made in the practical work phases were based more on creative research and testing than on the realization of the final product. The functional part of the thesis was the journey from the collection of population data to its evaluation with machine learning methods. The work performed was part of KAMK's 2022 SmartCampus project, which was carried out on the premises of KAMK's campus area.

The results were promising and the estimates of the number of people pleasantly accurate, regardless of the relatively naive solution. The estimates were added to the database, from which they could be retrieved in various dashboards to show the estimated number of people in the classes. The population estimation was put to work on KAMK's Bull supercomputer, where it would hopefully be in operation 24/7. The thesis highlighted the consideration of different solutions, reflected on the research and development process, and described what doesn't work and why. Decisions have been tried to be justified based on the results.

As a conclusion, the thesis offers a challenge for the further development of the topic, the so-called of a solvable problem that appeared on a practical basis: The domain matching problem is a relatively little-known machine learning problem encountered in this work. Its main idea is to solve how different environments can be generalized to one machine learning model.

Sisällys

Symboliluettelo

.csv	Excel-tiedostopääte
.pkl	Pickle-tiedostopääte
C++	Ohjelmointikieli
ESP32	Mikrokontrolleri
Docker	Virtualisointiteknologia
Grafana	Datan visualisointialusta
HVAC	Heating Heating, Ventilation and Air Conditioning (Lämmitys ja ilmastointi)
Haar-kaskadit	Objektintunnistusalgoritmi
InfluxDB	Aikasarjatietokanta
JupyterLab	Ohjelmistokehitysalusta
KAMK	Kajaanin Ammattikorkeakoulu Oy
Meteostat	Python-säädatakirjasto
OpenCV	Konenäkö ja -oppimis-ohjelmistokirjasto
Python	Ohjelmointikieli
RaspberryPI	Mikrotietokone
Scikit-learn	Koneoppimiskirjasto
Smaca	SmartCampus-hanke
Tensorflow	Koneoppimis- ja neuroverkko-ohjelmistokirjasto
YOLO	Objektintunnistuskirjasto

1. Johdanto

Julkisissa rakennuksissa on paljon huoneita, jotka on pääsääntöisesti suunniteltu olemaan käytettävissä 24/7. Kaikissa huoneissa pyörii ilmastointi ja lämmitys jopa silloin, kun huoneessa ei ole ketään. Ei voi koskaan tietää tarkalleen, milloin huoneet ovat oikeasti käytössä, ellei kuvaa huonetta valvontakameralla. Ei voi myöskään tietää, milloin huone on normaalia täydempi, jolloin lämmitys voi olla liikaa tai ilmastointi tehotonta. Kun kaikki huoneet pidetään samassa tilassa, ne ovat harvoin ideaalisia.

Tämä on SmartCampus-hankkeessa (Smaca) ratkaistava ongelma. Smaca oli Kajaanin Ammattikorkeakoulun (KAMK) hanke, joka on jatkoa edeltävälle työlle, jossa luotiin IoT-sensoreille oma verkko, joka kerää sensoridataa tietokantaa luokkatiloista. Juha Hauhia oli työstänyt tätä IoT-verkkoa ja on nyt Smaca-hankkeen projektipäällikkö. Olin hankkeessa projektityöntekijänä ja tämä opinnäytetyö on tehty Smaca-hankkeen päättymisen jälkeen. Työ on hyvin mielenkiintoinen sen teoreettisuuden vuoksi: On idea, joka vaikuttaa mahdottomalta, mutta kaiken kaikkiaan täysin loogiseltakin.

Tässä työssä esittelen työstämäni ratkaisun, millä huoneen ihmismäärän saisi määriteltyä sensoridatasta tekoäly- ja koneoppimismenetelmien avulla. Käytössä on monia erilaisia teknologioita ja laitteita, mikä luo pohjaa HVAC:in ohjaamiseen tarkoitetulle sulautetulle järjestelmälle joskus tulevaisuudessa. Inspiraationa toimi myös aiemmin kirjoitetut paperit aiheesta, joissa käytettiin hieman erilaisia tekniikoita saavuttamaan sama tavoite, mutta mikään niistä ei sisältänyt mitään koneoppimismallin opettamisesta [1; 2; 3]

Tavoitteena oli saada ihmismääränarvio edes n. 5 ihmisen tarkkuudelle. Tämä tavoite ei perustunut mihinkään ja oli vain pinnallinen keksintö jonkinlaiseksi vaatimukseksi. Olin jo aloitusvaiheessa varma, että ainakin yhdelle huoneelle saataisiin opetettua malli arvioimaan ihmismäärää suhteellisen hyvin. Ongelma, jota en alussa edes ajatellut oli, miten saisin yleistettyä monen huoneen datan yhdelle mallille.

Tutkimus ja työskentely oli suurin osin itsenäistä. Olin projektipäällikkö Juha Hauhiaan yhteyksissä paljon kuvaten nykyistä kehitys ja mahdollisia ongelmia, joita on tullut vastaan. Hänen kanssansa myös säädimme ja asensimme kameroita luokkiin.

2. Käsitteitä & teknologioita

2.1 ESP32

ESP32 on sarja edullisia, vähän virtaa käyttäviä mikrokontrollereita, joissa on integroitu WiFi ja Bluetooth. ESP32-sarjassa käytetään joko Tensilica Xtensa LX6 -mikroprosessoria (kaksiytimisissä yksiytimisissä variaatioissa), kaksiytimistä Xtensa LX7 -mikroprosessoria tai yksiytimistä RISC-V-mikroprosessoria ja se sisältää sisäänrakennetut antennikytkimet, RF-balunin, virran vahvistimen, vastaanottovahvistin, suodattimet ja virranhallintamoduulit. ESP32:n on luonut ja kehittänyt Espressif Systems, Shanghaissa sijaitseva kiinalainen yritys, ja TSMC valmistaa sen 40 nm:n prosessillaan. ESP32:sen edeltäjä on ESP8266. [4.]

2.2 Raspberry PI

Raspberry Pi on sarja pieniä yhden piirilevyn tietokoneita (SBC), jotka Raspberry Pi Foundation on kehittänyt UK:ssa yhteistyössä Broadcomin kanssa. Raspberry Pi -projekti suuntautui alun perin tietojenkäsittelyn perusopetuksen edistämiseen kouluissa ja kehitysmaissa. Alkuperäisestä mallista tuli odotettua suositumpi, ja se myytiin kohdemarkkinoidensa ulkopuolelle esimerkiksi robotiikkaan. Sitä käytetään laajalti monilla aloilla, kuten sään tarkkailuun alhaisten kustannusten, modulaarisuuden ja avoimen suunnittelunsa vuoksi. Sitä käyttävät tyypillisesti tietokone- ja elektroniikkaharrastajat, koska se käyttää HDMI- ja USB-laitteita. [5.]

2.3 InfluxDB

InfluxDB aikasarja-alusta on suunniteltu keräämään, tallentamaan, käsittelemään ja visualisoimaan mittauksia ja tapahtumia [6.]. Pääsääntöisesti sitä käytetään tietokantana tallentamaan aikasarjaista dataa, eli dataa, missä aikaleima on tärkein. Se on yksi suosituimmista valinnoista tällaiselle datalla sen nopeuden ja helppokäyttöisyyden vuoksi. [7.]

2.4 Grafana

Grafana on avonaiseen lähdekoodiin perustuva interaktiivinen kojelauta, jossa voi visualisoida dataa monella eri tavalla [8]. Dataa haetaan kyselyillä, joihin voi määritellä erilaisia rajoja esim. miltä aikaväliltä hakee tai hakee vain, jos jonkun kolumnin arvo on 1. Dataa voi siis suoraan muokata Grafanassa tehokkaasti. Grafanaan saa asennettua ilmaisia lisäosia, joita yhteisö on luonut. Lisäosat tuovat esimerkiksi uusia ominaisuuksia tai graafityyppejä. [9.]

2.5 OpenCV

OpenCV (Open Source Computer Vision Library) on avoimen lähdekoodin tietokonenäön ja koneoppimisen ohjelmistokirjasto. OpenCV luotiin tarjoamaan yhteinen infrastruktuuri konenäkösovelluksille ja nopeuttamaan tietokoneen havainnoinnin käyttöä kaupallisissa tuotteissa. OpenCV on Apache 2 -lisensoitu tuote, joten yritykset voivat helpoksi käyttää ja muokata koodia [10.]. Tässä projektissa käytettiin Python- ja C++ -versioita OpenCV-kirjastosta nimeltään cv2.

2.6 Tensorflow

Tensorflow on avoimeen lähdekoodin perustuva kirjasto, jota käytetään koneoppimiseen ja tekoälyyn. Se on kiistattomasti suosituin tapa koneoppimisongelmien ja tuotteiden luomiseen sen yksinkertaisuuden, tehokkuuden ja selvyyden vuoksi. Tensorflow tukee Pythonia, C++ ja Javaa. [11.]

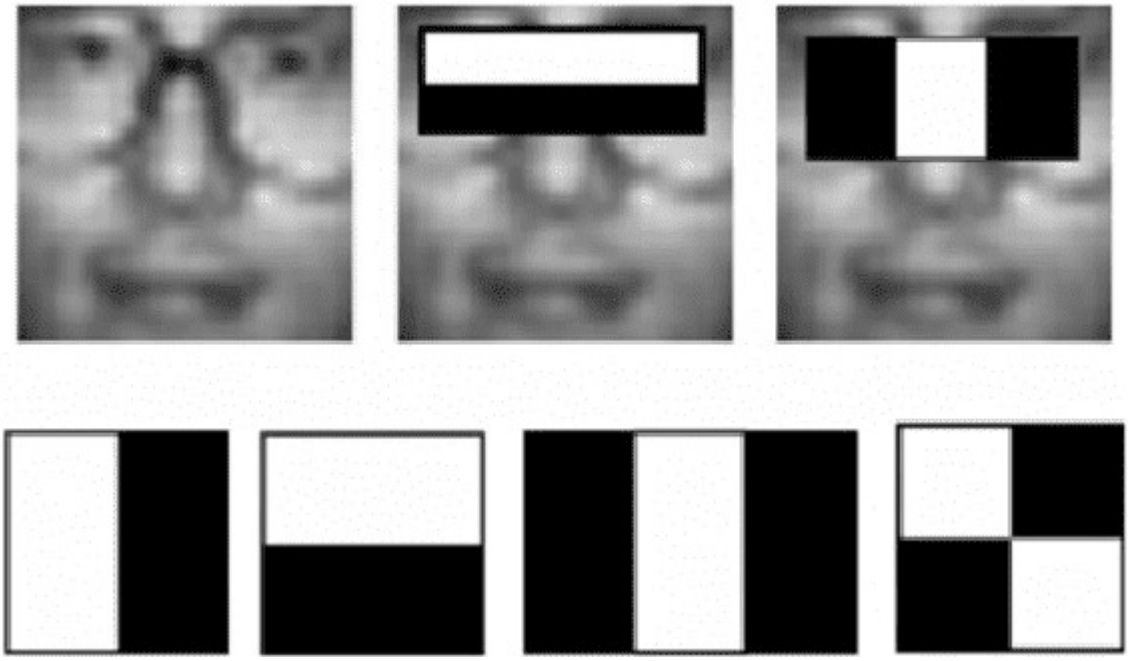
2.7 YOLO-mallit

YOLO on suosittu ja tehokas objektintunnistusmalli, josta on vuosien varrella kertynyt monta eri versiota, joista uusin, on YOLOv7. Alun perin Joseph Redmonin Darknet malliin pohjautuva, YOLO-perheen mallit olivat ensimmäisiä objektintunnistusmalleja, jotka tekivät tunnistuksen ja luokittelun yhdellä kertaa. Tämä oli paljon tehokkaampi ratkaisu muihin verrattuna, varsinkin YOLO:n julkaisu vuonna 2016. Tämä tehokkuus onkin YOLO-mallien hyöty: Ne mahdollistavat reaaliaikaisen objektintunnistuksen. [12.]

2.8 Haar-kaskadit (Haar cascades)

Haar-kaskadi-luokittelijat opetetaan positiivisilla ja negatiivisilla vahvistuksilla: Positiiviset kuvat ovat kohteesta mitä halutaan mallin oppivan (esim. ihminen), negatiiviset ovat kuvia kaikenlaisista muista kohteista, mitä ei haluta mallin oppivan. Haar-kaskadit etsivät näistä positiivisista kuvista tietynlaisia ominaisuuksia ja yhtäläisyyksiä. Kuva 3 havainnollistaa, mitä nämä ominaisuudet voisivat olla, jos kohteena on ihminen.

Haar-kaskadit ovat todella nopeita niiden yksinkertaisuuden vuoksi, mutta ne antavat paljon vääriä positiivisia. Ne ovat olleet olemassa jo 2001 asti, mutta modernit objektintunnistusmenetelmät suoriutuvat tätä tekniikkaa paremmin. [13.]



Kuva 1. Haar-kaskadeja ja mistä algoritmi on löytänyt niitä kasvoista [13].

2.9 Jupyterlab

Project Jupyter on voittoa tavoittelematon, avoimen lähdekoodin projekti, joka syntyi IPython-projektista vuonna 2014, kun se kehittyi tukemaan interaktiivista datatiedettä ja tieteellistä laskemista kaikilla ohjelmointikielillä. Jupyter on avoimen lähdekoodin ohjelmisto ja julkaistaan muokatun BSD-lisenssin ehdoin [14.]. Jupyterlab tarjoaa kehitysympäristön, jossa Python- ja Octave-koodin iterointi ja tutkiminen on erittäin kätevää ja nopeaa.

3. Lähtökohta

Scama-hanke oli jatkoa edeltävälle työlle KAMK:issa. Juha Hauhia oli kasannut sensoriyksikköjä, jotka sai yhdistettyä omaan anturiverkkoon kampusalueella. Sensoriyksiköt yhdistyivät sensoriverkkoon KAMKin kampusalueella ja keräsivät dataa yhteiseen tietokantaan, josta data haettiin visualisoitavaksi nettisivulle Grafanaan. [15, s. 1-41.]

Sensoriyksikkö koostuu ESP32-mikrokontrollerista, jossa on viisi erilaista anturia: Hiilidioksidi-, kosteus-, lämpötila-, valo- ja liikeanturi. Liikeanturi tosin todettiin turhaksi sen epäluotettavien ja epäselvien mittauksien vuoksi. [15, s. 26.] Tässä opinnäytetyössä ei koskettu sensoriyksiköihin sen enempää kuin siirtämällä niitä huoneista remontin aikana.

Sensoriyksiköt lähettävät jatkuvasti kerättyä mittausdataa InfluxDB-tietokantaan. Tätä kautta sensoridataa haettiin esikäsiteltäväksi ja sitten koneoppimismalleihin. Lopputuloksena aikaero, kun sensorin mittaus menee tietokantaan, se haetaan, esikäsitellään ja sille tehdään ihmismäärän arvio, oli alle puoli sekuntia. Kun dataa haettiin ja arviota tehtiin tusinalle sensorille, aikaero oli vielä alle 1 sekunti. Tämä kuvaa aika hyvin, miten koko arkkitehtuuri on hyvin kevyt ja tulevaisuudessa skaalautuva. [15, s. 18.]

Smaca-hanketta edeltävästi Grafana oli asennettu pyörimään edellä mainitun sensoriverkon nettisivuille [15, s. 4]. Näitä sivuja on Grafanalla kaksi: Ensimmäinen näyttää vain sensoriarvoista piirretyt historiagraafit, toisena on Grafanan kojelauta, jossa pystyy säätämään datahakuja, graafeja ja muuta tarpeellista. Kojelautaan pääsee julkisesti käsiksi osoitteessa anturiverkko.fi. [15, s. 19; 16.]

4. Ihmisten laskeminen

Ihmismäärän arviointi sensoridatasta vaatii ensin dataa oikeasta ihmismäärästä. Tätä tarvitaan koneoppimismallin opettamiseen ja kun mallin arvioita verrataan opetusdataan, voidaan mallin oppiminen täten validoida. Ongelma on siis ihmismäärän kerääminen huoneessa, mikä sitten yhdistetään sen sensoridataan.

Hanketta tehtiin KAMKissa, joka antoi loistavia mahdollisuuksia ihmismäärien keräämiseen eri luokissa. Kaikki testit tehtiin kampusalueen Taito 1 -rakennuksessa, pääosin 2. ja 3. kerroksissa. Tiedonkeruuta hidasti 2022 remontti, jolloin luokissa ei ollut opetusta ja sensoriyksiköt piti viedä niistä pois.

4.1 Ihmismäärän manuaalinen kerääminen

Ensimmäinen yritys oli laittaa huoneisiin lappuja, joihin opettajat täyttivät manuaalisesti läsnäolijoiden määrän tunnin alussa ja sitten lopussa. Tällä tavalla saimme n. kahden kuukauden verran dataa, mutta pääongelma ilmeni heti: Lappuja ei aina muistettu täyttää. Tiesimme, milloin luokissa on opetusta, jolloin tiesimme, miltä tunneilta läsnäolijalistaus puuttui. Myös datan siirtäminen manuaalisesti digitaaliseen muotoon (.csv-tiedosto eli Microsoft Excel -taulukko) oli työlästä ja hidasta. Automatisoitu ratkaisu vaadittaisiin.

Lappujakin iteroitiin pari kertaa: Ensimmäisessä versiossa vaadittiin opettajia allekirjoittamaan paperit, mistä näkisi kenen tunneilta ihmismäärä on laskettu. Tämähän oli täysin turhaa dataa ja muutaman kommentin pohjalta otimme allekirjoitusvaatimuksen pois tietoturvasyistä.

4.2 Ihmismäärän automaattinen laskeminen

Läsnäolijoiden laskeminen automaattisesti päätettiin tehdä kameralla, joka asennettiin luokkaan. Meillä oli käytössä vain kaksi kameraa ja Raspberry PI:tä, mikä tarkoitti, että emme saisi opetettua koneoppimismallia kuin kahdelle luokalla. Parempi se kuin ei mitään.

Kokeilin hankkeen aikana kahta eri tekniikkaa ihmismäärän laskemiseen kameralla:

- Liikkuvien objektien laskeminen. Kamera asennettiin oven yläpuolelle ja se laskee, kuinka monta objektia on mennyt sen näkökentässä alas (sisäänkäynti) ja ylös (uloskäynti)

- Ihmisten tunnistus tekoälyllä. Kamera asennettiin luokan liitutaulun yläpuolelle, josta se näkisi mahdollisimman paljon luokkaa yhdellä kertaa.

4.2.1 Tietosuoja & Vaikutuksenarviointi

Koska kyseessä on julkisettilat sekä ihmisten filmaaminen kameroilla, meidän täytyi täyttää KAMKin vaikutuksenarviointilomake. Tässä lomakkeessa piti kuvata, mitä tehdään, miten tehdään, mitä datalla tehdään, miten sitä säilytetään sekä mitä riskejä tietovuodosta voi tulla. Lomakkeen idea on siis varmistaa KAMKin laillinen asema, jos asiasta tulee käräjäjuttu. Lomakkeella mallataan, onko tarkoitus sopiva, onko siihen liittyvät riskit tarpeeksi pienet ja miten se vaikuttaa ihmisiin.

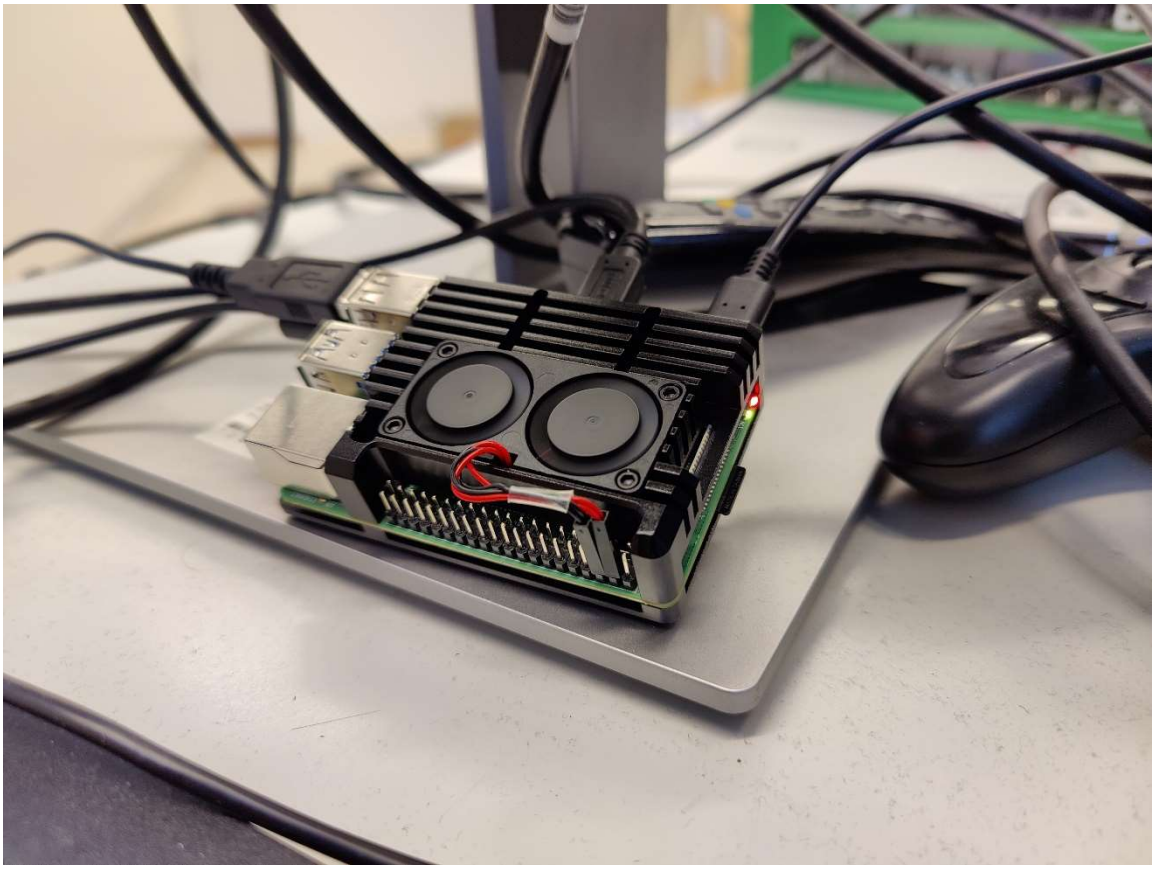
Pääsimme suhteellisen helpolla, kun emme missään vaiheessa tallenna tai lähetä kameran kuvaa minnekään, vaan kaikki pysyy Raspberry PI:llä. Ongelmia vaikutuksenarvioinnissa olisi tullut, jos olisi toisin. Pääsin täyttämään lomakkeen itse tarkistuttaen sen Juha Hauhialla. Oli ihan hauska päästä osallistumaan vähän byrokratiaankin.

Lopulta kun asennettiin kameroita luokkiin, laitoimme luokkien oviin varoituslapun, jossa vaan kerrottiin luokassa olevan tällainen AI-kamera, jonka voi kytkeä pois päältä.

4.2.2 Raspberry PI -ratkaisu

Raspberry Pi:tä päätettiin käyttämään ihmislaskurin pohjana sen helppokäyttöisyyden ja suorituskyvyn takia. Yritin myös tätä päätöstä ennen tehdä laskuria ESP32:lle, mutta objektintunnistuksen tekeminen vaatii enemmän suorituskykyä kuin ESP32 tarjoaa. Myös pelkän kameran yhdistäminen oli hankalaa, normaalia USB-webkameraakaan ei saanut ESP32 yhdistettyä mitenkään yksinkertaisesti. Ottaen huomioon, miten ihmislaskureita ei tarvitse, kun ne ovat keränneet dataa, ei myöskään nähty tarpeelliseksi tehdä “halvempaa” ratkaisua: Raspberry PI on noin kaksi kertaa kalliimpi kuin ESP32 [17].

Kuvantunnistuksen pyörittäminen Raspberry PI:llä on hyvin raskasta, ja pari kertaa laite sammui, koska se ylikuumeni ja muulloin performanssi huononi laitteen yrittäessä vähentää virrankäyttöä lämpeämisen estämiseksi. Onneksi Raspberry PI:lle tehdään integroituja tuulettimia, joten saimme hyvin kätevän ratkaisun tähän ongelmaan. Nyt lämpö ei ole rajana, vaan laitteen prosessointiteho itsessään.



Kuva 2. RaspberryPI tuulettimella käyttöjärjestelmää asennettaessa.

4.2.3 Webbikamera

Käytimme kameroina Logitech BRIO -webbikameroita. Syynä tämän mallin valintaan oli niiden 4K-videonlaatu ja että meillä oli jo kaksi. Koska olemme periaatteessa laskemassa pikseleitä, mitä enemmän niitä on. sitä tarkempia tunnistukset ovat. Tämä vaatimus validoitiin myös huonommalla 1080p:n webbikameralla, joka sopeutui korkeintaan ensimmäiseen toteutukseen.

5. Ihmismäärälaskuri V1-toteutus

Ensimmäinen tapa, jolla yritimme mitata luokan läsnäolijamäärää, oli yksinkertaisesti laskea, kuinka monta kertaa kamera laskee jonkin objektin menevän ovesta sisään & ulos.

5.1 ESP32-ratkaisu

Alussa mietittiin, voisiko ihmisiä laskea asettamalla jonkin sortin anturin/laserin laskemaan, kuinka monta kertaa laser rikkoutuu, kun objekti menee sen ohi. Ongelmana tässä ratkaisussa on, ette objektin kulkusuuntaa voi mitata. Tämä vaatisi kahden anturin asettamista vierekkäin, jolloin mitattaisiin, kumpi niistä laukeaa ensin kuvaten objektin kulkusuuntaa. Mutta koska oviaukko on sen verran ”ohut”, ei oven karmiin saisi kahta anturia järkevän kauaksi toisistaan.

Tässä ideassa oli tosin itua, joten rupesin testailemaan, jos saman voisi tehdä webbikameralla. Sain idean tutkimalla internetistä jo olemassa olevia ratkaisuja inspiraatioksi ja paras esimerkki löytyi PyImageSearch-sivulta [18]. Tässä vaiheessa ei Raspberry PI:tä työstetty, vaan yritettiin saada nimeämätöntä kameramoduulia toimimaan ESP32:lle. Tavoitteena oli selvittää, jos näitä komponentteja voisi käyttää tässä ongelmassa, koska se olisi paljon halvempaa kuin Raspberry PI ja webbikameran käyttö.

Valitettavasti kameramoduulia ei saatu ottamaan videota. Kamerasta ei edes löydetty muuta informaatiota kuin sen tietolomakkeen: Malli oli xd-95 OV2640, joka sekoittui samannimisiin kameramoduuleihin Arducam OV2640 ja Alientek OV2640. Kamera on tehokkaampi kuin paljon suositumpi OV7670-malli [19], mutta siinä oli erilaiset pinnit. Voi olla, että pinnit oli yhdistetty ESP32:een väärin tai itse koodi oli väärin. Päädyttiin sitten jättämään ESP32 sikseen ja siirryttiin kokeilemaan objektinlaskua Raspberri PI:llä ja webbikameralla.

5.2 Raspberry PI ratkaisu

Perusideana on tunnistaa objekti kameran syötteestä ja mitata, milloin se ylittää virtuaalisen rajan. Kamera ottaa videota n. 15 kuvaa sekunnissa.

Jokaiselle kuvalle tehdään seuraavat toimenpiteet:

1. Kuvan muuttaminen mustavalkoiseksi (Greyscale)

    a. Tämä on pääsääntöisesti performanssin optimointiin, koska värilliset pikselit ovat kompleksisempia mustavalkeisiin. Värillinen pikseli on 3-ulotteinen, koska se määritellään RGB (R = Red, G = Green, B = Blue) värikanavien arvoilla. Esim. ”magenta” saadaan R: 255, G: 0, B: 255. Ohjelmoinnissa pikseliä kuvaisi lista ”[255, 0, 255]”.

    b. Mustavalkeaa pikseliä kuvaa vain yksi arvo, periaatteessa kuinka ”päällä” pikseli on. Intuitiivisimmin tämä skaalataan 0 %-100 % välille.

    c. Jos miettii, että 1920x1080 kameran syötössä on 2 073 600 pikseliä ja jokaisella on 3 arvoa, jokaisen värillisen videosyötön framelle pitää tehdä laskutoimitus 6 220 800 kertaa. Jos kuva on mustavalkea, tarvitsee laskutoimitus tehdä vain tuo rapiat kaksi miljoonaa kertaa.

2. Binärisäätiö (Binarization)

    a. Pikselin väriarvot skaalataan joko valkoiseksi tai mustaksi. Jos keskiharmaa on 0.5 (50 %), se skaalataan valkoiseksi. Jos se on 0.49, se skaalataan mustaksi. ”Musta” tarkoittaa tyhjää ja ”valkoinen” objektia. Tämän filtterin skaalan muuttaminen vaikuttaa kaikkein eniten, miten hyvin objekti erottuu taustasta. [20.]

3. Ääriviivojen etsiminen (Contour extraction)

    a. Kuvasta etsitään objektit joiden reunoilla on sama intensiteetti [18].

    b. Etsintä tehdään valkoisten alueiden pinta-alojen perusteella. Jos pinta-ala on liian pieni, sen oletetaan olevan ”kohinaa”. Jos pinta-ala on tarpeeksi iso ja se on toisen ison alueen vieressä, lasketaan alueet yhdeksi.

4. Objektin keskipisteen etsiminen (Centroid)

    a. Piirretään valkoisen alueen ääriviivojen pohjalta sen ympärille laatikko. Sitten laatikon keskipisteen voi laskea ja tätä käytetään kuvaamaan objektin keskipistettä. [18].

b. Nyt pisteellä on kuvassa koordinaatit. Jos videosyötön resoluutio on 1920x1080 ja pisteen koordinaatit ovat sama, olisi piste silloin oikeassa alakulmassa (pikseleitä ruvetaan laskemaan vasemmasta yläkulmasta). Jos piste on keskellä ruutua, sen koordinaatit olisivat (960, 540).

5. Keskipisteiden seuraaminen

   a. Jotta voidaan oikeasti seurata objektia, meidän pitää tietää, että seuraamme varmasti samaa, eikä jotain toista objektia. Tämä voidaan laskea esim. Mean Shift -algoritmilla, joka laskee, kuinka suuri ero pikseleillä on edellisessä kuvassa. Jos ero on suuri, voidaan sanoa, että emme seuraa samaa objektia. Jos se on pieni, seuraamme yhä samaa objektia.

   b. Manuaalisesti pisteen ”reitin” voi laskea ottamalle sen edellisen ja nykyisen koordinaatin euklidisen etäisyyden. Tämä on siis pikseleiden välillä olevan kolmion hypotenuusan pituus [18]. Kun tämä euklidinen etäisyys on tarpeeksi pieni (määrittelin sen 100 pikseliin), on piste edelleen sama.

   c. Logiikka tässä on, ettei objekti voi liikkua yli 100 pikseliä kahden framen välillä. Jos kuvassa kuitenkin on kaksi pistettä ja niiden etäisyys toisistaan on yli 100 pikseliä edellisen ja nykyisen framen välillä, niiden on oltava kaksi eri objektia.

6. Asetetaan objekteille uniikki ID

   a. Pisteiden koordinaateille asetetaan uniikki ID ja se lisätään listaan. Jos piste häviää kuvasta, otetaan sen ID ja koordinaatit pois listasta. Joka framella pusketaan ID:lle uudet koordinaatit, jolloin siitä saa luotua pisteen ”reitin” [18].

   b. Kun pisteen reitti on määritelty, voidaan päätellä, onko reitti ”ylös” vai ”alas”. Esim. ”ylös” tarkoittaa, että koordinaattien jälkimmäinen arvo pienenee (Y-arvo). Tämä määrittelee, onko objekti mennyt ”ulos” vai ”sisään” huoneeseen.

   c. Pidetään listaa pisteistä, jotka ovat jo ylittäneet viivan, jolloin niitä ei lasketa uudestaan.

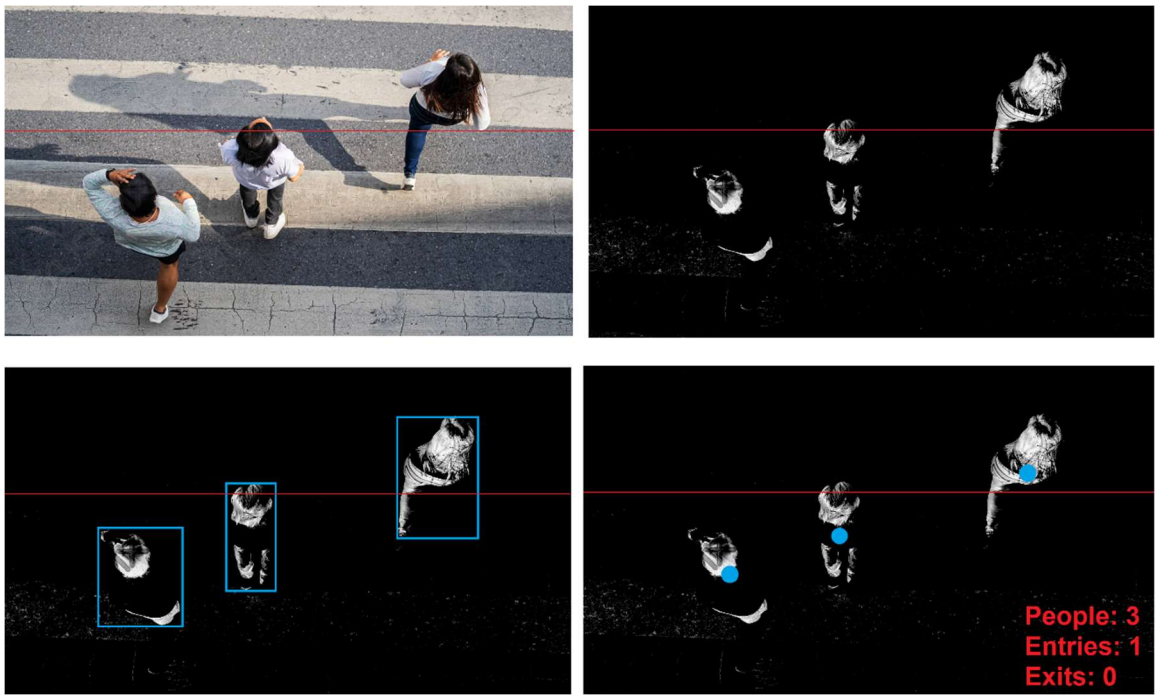
   d. Tämä algoritmi vaatii tarpeeksi korkean ruudunpäivitysnopeuden kameraan, jotta objektit eivät ”hypi” paikasta toiseen sen takia, ettei kamera näe, mitä sadasosasekunnissa tapahtuu.

Algoritmin huono puoli on, että se ei erota, onko objekti oikeasti ”objekti”, koska algoritmi toimii periaatteessa vain kontrastin pohjalta, joten se myös laskee heijastukset ja varjot objekteiksi. Kuvassa voi olla myös paljon ”kohinaa” eli skaalattuja pikseleitä on siellä-sun-täällä. Tunnistettujen ääriviivojen sisällä voi myös olla tyhjiä kohtia.

Kameran kuvan keskelle piirretään viiva sen pikselikoordinaattien perusteella. Jos objektin keskipiste ylittää viivan, nostetaan laskuri ylöspäin. Kokeilin mitata sisään/uloskäyntejä kahdella eri tavalla:

1. ”Sisään” viiva ja ”Ulos” viiva. Riippuen kumman viivan piste ylittää ensin, luokitellaan se sisään/ulos. Esimerkiksi jos piste ylittää ensin viivan ”Ulos” ja sitten viivan ”Sisään”, luokitellaan tapahtuma ”sisäänkäynniksi” ja ”Ihmisiä sisällä” laskuria nostetaan yhdellä. Loogisesti ajatellen ihminen ei voi tulla luokan sisällä ilman, että hän on ulkona, sen takia: ulos => sisään = +1 & sisään => ulos = -1.

2. Yksi viiva, jonka yläpuoli tarkoittaa ”Ulos” ja alapuoli tarkoittaa ”Sisään”. Riippuen kummalla puolella viivaa objekti nähdään ensin, sen statukseksi asetetaan sisään/ulos. Kun se ylittää viivan, sen statukseksi asetetaan nykyisen vastakohta. Riippuen kumpi tämä on, muutetaan ”Ihmisiä sisällä” laskurin arvoa.



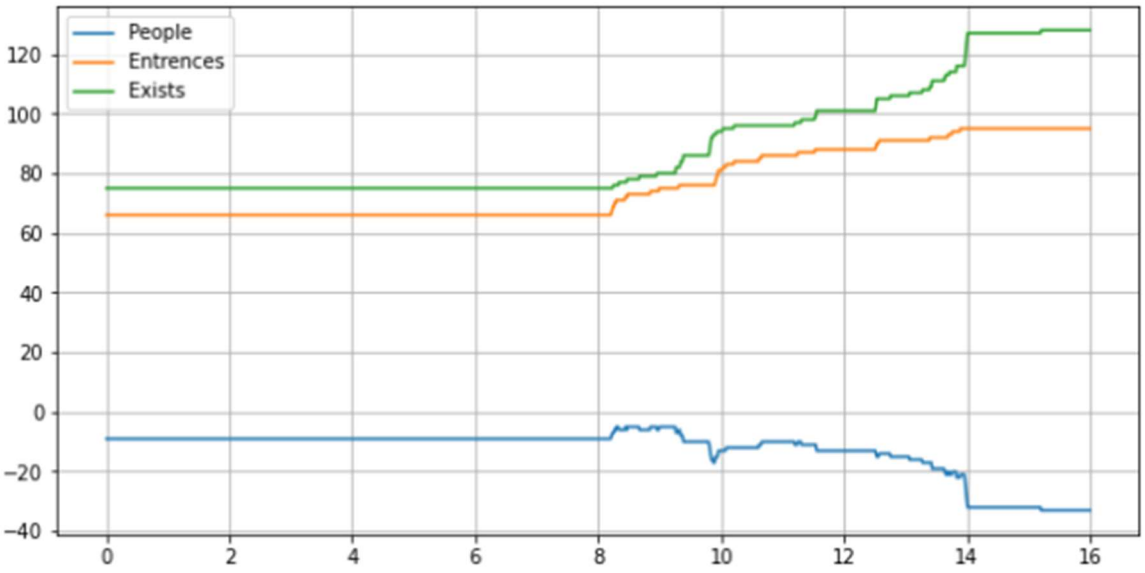
Kuva 3. Esimerkkivaiheet, miten ihmisen seuraus toimii tavalla 2.

6. Ihmismäärälaskuri V2-toteutus

Parin viikon testailun jälkeen tulimme siihen tulokseen, että laskuri oli liian epätarkka. Se vaatisi täydellisen valaistuksen ja lattian, jotta varjot ja heijastukset saisi minimoitua.

Kuva 3:ssa havainnoidaan ihmislaskurin käytöstä. ”Sisäänkäyntejä” (Entrences) on lopussa vähemmän kuin ”uloskäyntejä” (Exits). Kamera todennäköisesti luuli siis sisään tulevan ihmisen olevankin ulosmenevä, laskien ihmisen periaatteessa kahdesti. Lopussa ihmismäärä on n. -30, eli kamera luuli luokasta tulevan enemmän ihmisiä kuin sinne oli mennyt. Arvot myös ovat paisuneet, toinen todiste siitä, että kamera laskee yhden ihmisen monta kertaa.

Kamera laski yhden ihmisen monta kertaa, koska prosessointien jälkeen se saattoi erottaa ihmisen pään, olkapään ja repun yksittäisiksi objekteiksi. Aloitimme tutkimaan vaihtoehtoista tapaa ihmistenlaskuun ja hyvin nopeasti tulin ns. ”bruteforce” -tulokseen, että voisimme vain käyttää ihmisentunnistusta ja laskea sillä suoraan, kuinka monta ihmistä huoneessa on.



Kuva 4. Ihmislaskurin mittaukset. Testattu yhden oppitunnin ajan (n. 2 tuntia).

6.1 Objektintunnistus 101

Objektintunnistus on konenäön alatiede, joka perustuu objektien tunnistamiseen kuvista/videoista. Objektintunnistusmalli on joko neuroverkko- tai koneoppimispohjainen luokittelumalli, joka näkemistään pikseleistä palauttaa listan objekteista ja todennäköisyysarvon. Malli siis vertaa

kuvan pikseleitä omaan visuaaliseen kirjastoonsa ja päättelee, mitä se kaikkein eniten muistuttaa. [21.]

Jokaisella objektilla on jotain tiettyjä piirteitä. Pallot ovat aina ympyröitä, sen 3-ulotteisuuden voi päätellä valaistuksen luoman varjojen avulla. Sama pätee ihmiseen: Silmät ovat yleensä jonkin sortin mantelin muotoisia, nenä on jonkin sortin kolmio ja suu on jonkin sortin ovaali. Ihmiset ovat myös tietyn värisiä, jolloin muiden piirteiden löydyttyä, vihreä ihonväri olisi aika iso rokottaja, päättääkö malli näkevänsä ihmisen. Tämä myös johtaa tekoälyn eettisyyteen, esim. monta uutista on vuosien mittaan tullut, miten joku malli ei tunnista tummaihoisia ihmisiä, koska mallin opetuskuvina käytettiin vain kuvia valkoihoisista ihmisistä. Eli mallin tarkkuus perustuu hyvin pitkälti, mitä kuvia sille on opetettu ja kerrottu ”tämä on ihminen”. Mitä monipuolisempia kuvia ihmisestä, sitä parempi.

Mutta tämä on periaatteessa vasta kuvan luokittelua, ei tunnistusta. Käyttämällä samanlaisia tekniikoita kuin mainittu Ihmismäärälaskuri V1-toteutuksessa, voidaan termiä ”objektintunnistus” käyttää vasta, kun se on erotettu kuvasta ja se on rajattu laatikolla (bounding box) [18]. Tällöin ollaan varmoja, että malli tietää, missä ihminen on, mikä ihminen on ja nähdään visuaalisesti, ettei malli luule abstractia kuvioiden mylläkkää ihmiseksi.

6.2 Toteutus

OpenCV:n cv2-kirjasto on edelleen käytössä tässä versiossa, mutta tällä kertaa oman manuaalisen algoritmin sijaan käytettiin valmiita ratkaisuja. Muutamia eri tekniikoita kokeiltiin, joista viimeisin antoi parhaimpia tuloksia.

6.2.1 Haar-kaskadi-testi

OpenCV:n Haar-kaskadimalli suoriutui melko hyvin, kun sen piti tunnistaa kasvot ja kun ihminen katsoi suoraan kameraan. Mutta itse ihmisen koko kehoa se ei tunnistanut. Kokeilimme yhdistää kolme eri kaskadimallia: Kasvo-, pää-, ja kehontunnistuksen. Nyt malli tunnisti ihmisen muistakin

kuvakulmista, mutta edelleen, jos päätä ei selvästi erottanut kuvista, ei se tajunnut ihmisen kehosta mitään, jos ei ihminen poseerannut kameralle täydellisesti. Haar-kaskadien testaaminen jätettiin tähän ja siirryimme seuraavaan metodiin.

6.2.2 Tensorflow- & YOLO-testit

Aluksi käytettiin taulu 1:stä lähtökohtana YOLO-mallin valitsemiseen. Sitten benchmarkattiin valittuja malleja YOLOv4, TinyYOLOv3 ja YOLOX, koska niissä ruudunpäivitys oli suurimpia Raspberry PI:llä taulu 1:sen mukaan. Tulokset olivat seuraavanlaiset:

Benchmarkkasin malleja YOLOv4, TinyYOLOv3 ja YOLOX. Tulokset olivat tälläiset:

- YOLOv4 antoi parhaimman tunnistuskyvyn, toimien vaikeissakin olosuhteissa, esim. tunnistaen pimeässä nurkassa istuvan ihmisen. Huono puoli oli, että Raspberry PI ei jaksanut oikein pyörittää algoritmia, jolloin kameran ruudunpäivitys putosi hyvin alhaiseksi, n. 0.2 fps eli 1 kuva joka viides sekunti. Tämä oli tosi huono suoritus.

- TinyYOLOv3 on varta vasten tehty käyttämään vähemmän resursseja toimiakseen vähätehoisilla laitteita (kuten Raspberry PI). Ruudunpäivitys pomppasi n. 5 fps, mutta objektintunnistus oli huomattavasti huonompi; välillä kadottaen ihmisen, jos hän vaikka nojasi liikaa sivulle tuolilla istuessaan. Pimeästä nurkasta ei myöskään ihmistä enää tunnistettu.

- YOLOX ei toimi Pythonilla, vaan sitä pitää käyttää C++:lla, mutta se antoi suunnilleen saman objektintunnistuskyvyn kuin YOLOv4 ruundunpäivityksellä 4–7 fps. Tämä oli hyvä keskitie ja päätimme käyttää tätä Raspberry PI:ssä. Ongelma oli tosin, että YOLOX piti kääntää Raspberry PI:lle, joka oli erittäin hidasta (kesti kerran yli 2 tuntia). Lisäongelmia tuli myös, kun Raspberry PI:stä loppui kääntövaiheessa muisti (RAM) kesken. Asiaa tutkittua selvitettiin, että Raspberry PI:stä pitää kasvattaa sen ”swap file”-kokoa eli muistin loputtua, data siirtyy tallennustilaan. Tämä on yksi suurin syy käännön hitauteen. Juha Hauhia päättikin vain suoraan Raspberry PI -käyttöjärjestelmää asentaessa muistikortille kääntää YOLOX tälle kortille. Näin saatiin toimiva objektintunnistukseen kykenevä laite noin tunnissa.

YOLO-malleissa on tosin yksi suuri miinus: Maksimiresoluutio on 640p. Meidän 4K-kameramme oli siis melkein turha, koska ei objektintunnistusmalli pystynyt hyödyntämään kaikkia pikseleitä,

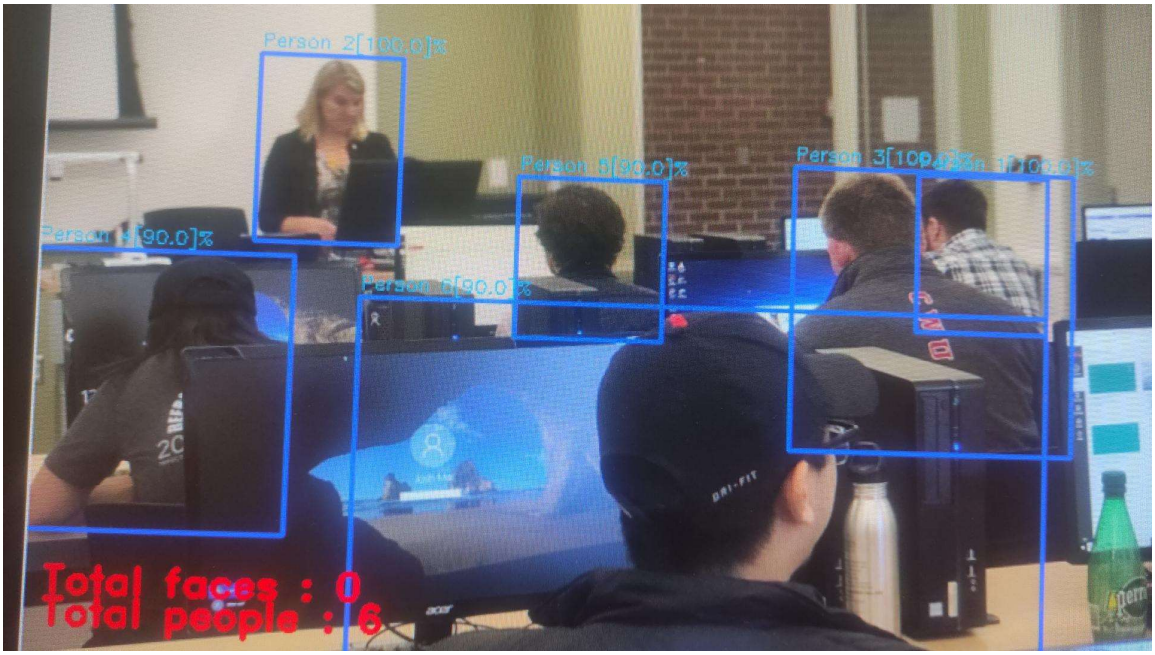
vaan kuva skaalautuu pienemmäksi. Mutta 4K:sta downskaalaaminen 640p:hen antaa silti paremman kuvan kuin natiivi 640p, joten ei 4K ollut täysin turhaa.

Taulu 1. Yolo-mallien benchmarkit [22].

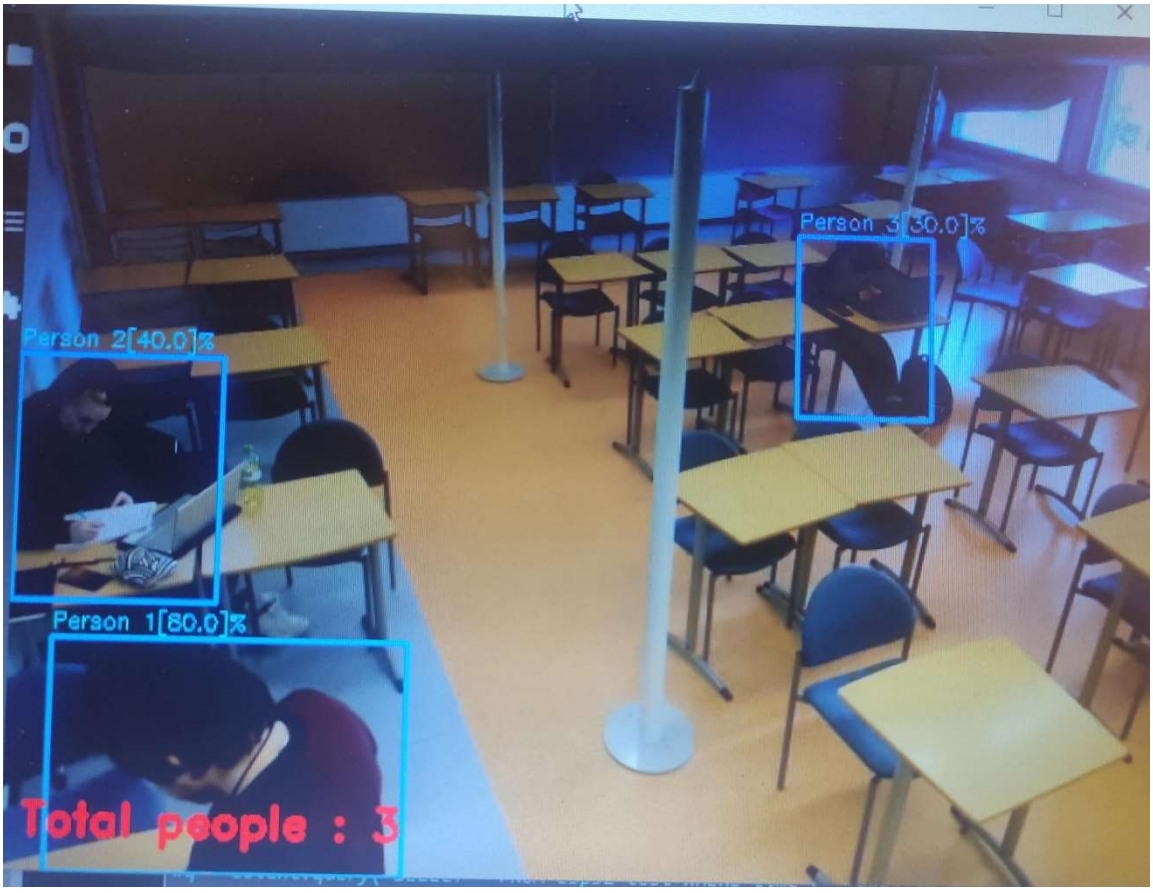
Model	size	objects	mAP	Jetson Nano 1479 MHz	RPi 4 64-OS 1950 MHz
NanoDet	320x320	80	20,6	26.2 FPS	13.0 FPS
NanoDet Pl	416x416	80	30,4	18.5 FPS	5.0 FPS
YoloFastestV2	352x352	80	24,1	38.4 FPS	18.8 FPS
YoloV2	416x416	20	10,1	10.1 FPS	3.0 FPS
YoloV3	352x352 tiny	20	16,6	17.7 FPS	4.4 FPS
YoloV4	416x416 tiny	80	21,7	16.1 FPS	3.4 FPS
YoloV4	608x608 full	80	45,3	1.3 FPS	0.2 FPS
YoloV5	640x640 small	80	22,5	5.0 FPS	1.6 FPS
YoloV6	640x640 nano	80	35,0	10.5 FPS	2.7 FPS
YoloV7	640x640 tiny	80	38,7	8.5 FPS	2.1 FPS
YoloV8	640x640 nano	80	37,3	14.5 FPS	3.1 FPS
YoloV8	640x640 small	80	44,9	4.5 FPS	1.47 FPS
YoloX	416x416 nano	80	25,8	22.6 FPS	7.0 FPS
YoloX	416x416 tiny	80	32,8	11.35 FPS	2.8 FPS
YoloX	640x640 small	80	40,5	3.65 FPS	0.9 FPS

6.2.3 Ihmisentunnistus käytännössä

Aloitin tunnistuksen testaamisen tekemällä sitä kuville. Etsin internetistä kuvia, joissa oli ihmisiä istumassa jonkin sortin huoneessa. Yolo mallit on koulutettu tunnistamaan monenlaisia kohteita (n. 70) ja joka kuvasta malli tunnistaa sitä, mitä vain kykenee. Jokainen tunnistettu kohde puskетaan listaan, jossa sillä on todennäköisyysarvo prosentteina. Ihminen on yksi kohteista, mitä malli tunnistaa, joten jos mallin tunnistuslistassa on tagi ”human” ja sen todennäköisyys arvo on yli 20 (tähän päädyin testien jälkeen), voidaan laskea, että kuvassa on ihminen. Vähintään 20 % todennäköisyysarvon määrittämisellä vältetään väärät tunnistukset, joita on yleensä jonkin verran.

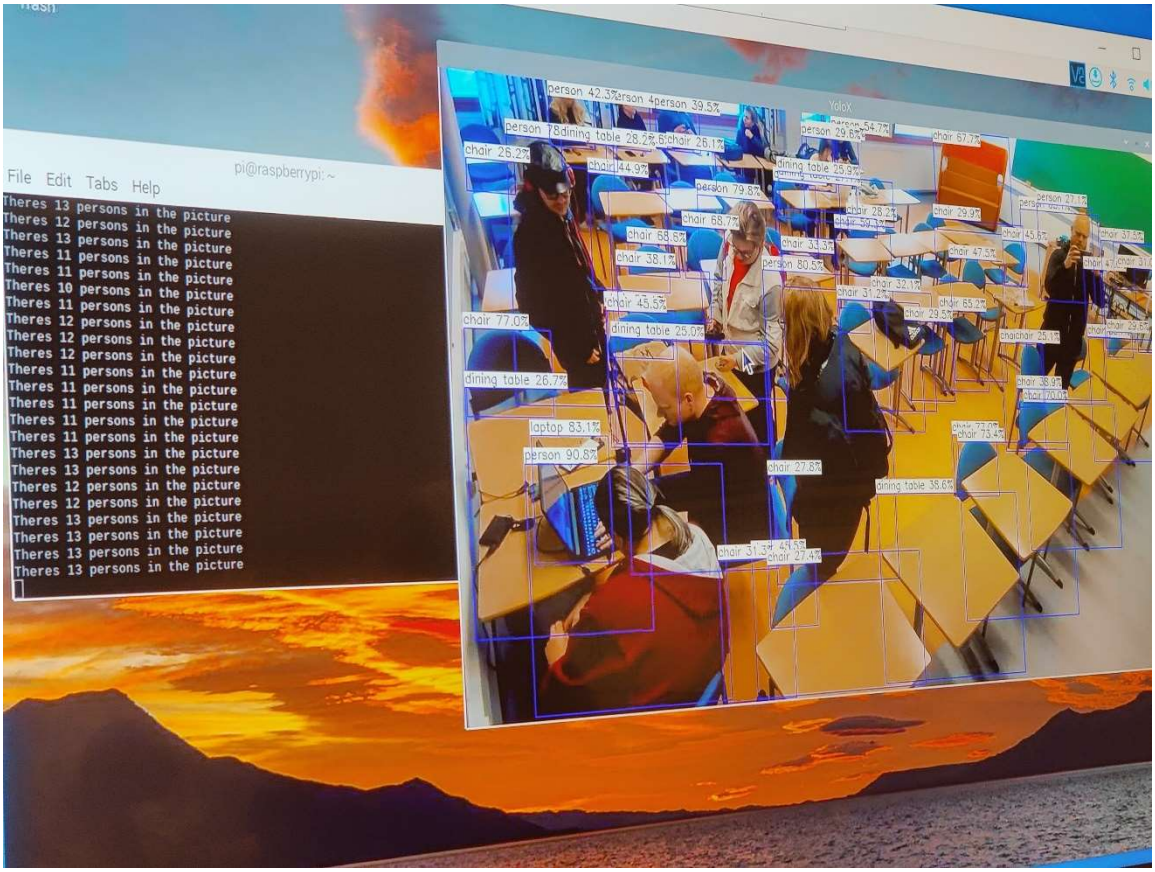


Kuva 5. YoloV4 testi satunnaisella kuvalla, jonka löysin internetistä. Malli on todella hyvä, tunnistaen ihmisen jopa, kun sen kasvoja ei näe, mikä olikin tämän testin tarkoitus.



Kuva 6. YoloV4-testi luokassa, itse säädän kameran asetuksia vasemmassa alakulmassa. Tämä kuva havainnoi melko hyvin, miten tunnistuksen tarkkuus (tai varmuus %) putoaa, mitä kauemmaksi menee esim. ”Person 2[40–0] %”.

Tunnistus näytti tässä vaiheessa erittäin lupaavalta, joten laitoimme kaksi Raspberry PI + webbikameraa kahteen eri luokkaan: Yksi käytti YoloV4:ää Pythonilla ja toinen käytti YoloX:ää C++:lla. YoloX-versio toimi jonkin verran paremmin (n. 1–2 ihmistä tarkempi) kiitos suuremman ruudunpäivityksen.



Kuva 7. YoloX-testi, Juha Hauhian ottama kuva oppitunnillaan. Tämä kuva havainnollistaa, miten Yolo-mallit on opetettu tunnistamaan montaa erilaista objektia samaan aikaan. Kuvasta manuaalisesti laskemalla voi nähdä 12 henkilöä, joten mallin tunnistamat 13 osuu melkein nappiin. Kuvasta näkee myös, missä tuo yksi ylimääräinen ihminen on tunnistettu: Juha Hauhia itse on tunnistettu kaksi kertaa (oikealla reunassa).

Viimeiseksi kameroiden suhteen kokeilimme ratkaista kameran pienen näkökentän ongelmaa. Mihin tahansa luokkaan asensimme kameran, joko takapenkki oli liian kaukana tunnistaakseen sieltä ihmisen tai kameran näkökentän reunoilla olevat paikat jäivät piiloon. Periaatteessa kummassakin n. 5–6 paikkaa jäi aina piiloon. Kokeilimme erilaisia kalansilmälinssiadaptereita webbikameroille, jotka toimivat ihan ok, mutta huononsivat takapenkin tunnistusta vielä enemmän kuin ilman niitä. Testailujen jälkeen päätimme vain olla käyttämättä näitä linssejä, koska niistä ei ollut tarpeeksi hyötyä.

7. Ihmismäärän arviointi koneoppimisella & tekoälyllä

Ihmismäärädatan kerääntyessä aloitettiin tutkimaan, miten ihmismäärää voidaan tästä edespäin arvioida koneoppimismenetelmillä suoraan sensoridatasta. Tätä kehitin koko ajan Jupyterlab Python -kehitysympäristössä, jossa voi ajaa erillisiä ”koodisoluja” kerrallaan nopeaan kehittämiseen ja iterointiin. InfluxDB-tietokantaan saatiin helposti yhteys Pythonilla influxdb-kirjastolla. Sensoridataa oli 15 eri huoneesta, mutta vain kahdessa oli ihmislaskuri, joten koneoppimiseen voitiin käyttää vain 2/15 huoneista. Tämä ”datan puute” olikin tässä osiossa ilmenevä ongelma.

Tässä työssä viitataan ihmismäärää arvioivaan koneoppmismalliin ”arviointimallina”, vaikka projektin aikana käytimme sanaa ”ennustusmalli”. Näiden eron voisin mainita: Tässä projektissa keskittyminen oli vain ihmismäärän arvioinnin tekemiseen nykyisellä ajan hetkellä, ei sen ennustamista tulevaisuudessa. Tarkoitus ei ollut missään vaiheessa projektia yrittää ennustaa ihmismäärää, vaikka hyvin luontevasti se olisi ollut seuraava vaihe. ”Ennustusmalli” myös tuntui asettavan meihin oikeat oletukset, mitä haetaan, mutta nyt kun taaksepäin katsoo, se tarkoittaa hyvin erilaista käyttötarkoitusta arviointiin verrattuna.

7.1 Datan prosessointi

Sensoridataa haettiin kahden kuukauden ajalta ja tein sille perusprosessoinnit:

1) Korjattiin kolumnien tyypit.
2) Korjattiin puuttuvat arvot.
3) Poistettiin rivit klo 17–7 väliltä.
    a. Yöaika, jolloin luokissa ei pitäisi olla ketään muutenkaan.
4) Poistettiin rivit, joissa hiilidioksiditaso on alle 400 ppm (Parts Per Million).
    a. Pitäisi yleensä aina olla yli 400, alle tämän tarkoittaa jotain vikaa.
5) Vähennettiin lämpötila mittauksista 2 astetta.
    a. Sensorivirhe.

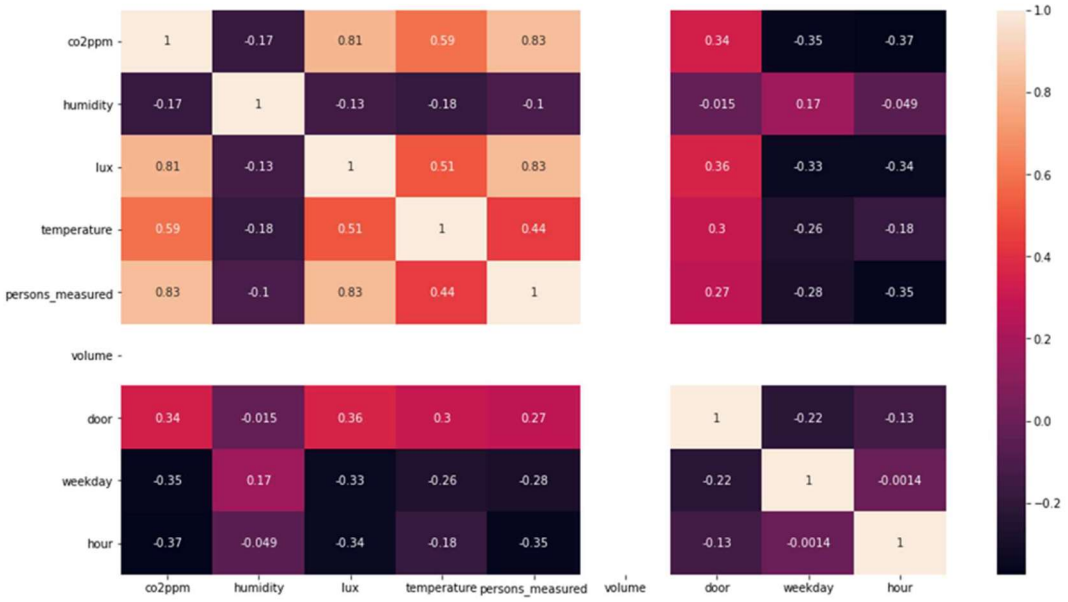
6) Yhdistettiin eri luokkien sensorit yhteen datasettiin.

    a. Lisättiin ”Luokan nimi” -kolumnin.

7) Kun ihmisdataa oli saatu kerettyä, Lisättiin se datasettiin.

    a. Lähimmän aikaleiman perusteella.

Nyt kun data oli siistiä, sitä pystyttiin analysoimaan. Päälöydöt löytyivät yksinkertaisesti korrelaatiomatriisilla. Kuvassa 7 näkyy, miten hiilidioksiditaso ja valonmäärä korreloi vahvasti (81 %) positiivisesti. Myöhemmin kun ihmismäärädata yhdistettiin tähän samaan datasettiin, hiilidioksiditaso korreloi vahvasti (83 %) ihmistenmäärän kanssa. Tämä käytös seuraa intuitiivista: Ihmiset hengittävät ulos hiilidioksidia, jolloin hiilidioksiditaso nousee. Ihmiset tuskin haluavat istua pimeässä, jolloin he laittavat huoneessa valot päälle. Lämpötila ja ilmankosteus korreloivat ihmismääriin jonkin verran (44 % ja 27 %). Ovivahtimme (kuva 7 ”door”) näytti myös korreloivan kaikkien sensoriarvojen paitsi ilmankosteuden kanssa. Emme kuitenkaan voineet käyttää ovivahtia mallien opetukseen, koska se otettiin käyttöön hankkeen loppupuolella ja vain yhdessä luokassa. Meillä ei ollut kerääntynyt siitä tarpeeksi dataa.



Kuva 7 Korrelaatiomatriisi featureista. Mitä vaaleampi, sitä positiivisempi korrelaatio. Mitä tummempi, sitä negatiivisempi

7.2 Featurematriisin luonti

Featurematriisi sisältää kolumnit ja lisäkolumneja, joita datasta on eroteltu. Nämä lisäykset voivat olla esim. tunnin erotus aikaleimoista. Feature, eli ”ominaisuus”, kuvaa muuttujia, jotka tiedetään entuudestaan ja joiden perusteella tehdään ennustus. Featurematriisi siis voi sisältää vain muuttujia, jotka tiedetään ennen ennustuksen tekemistä.

Yksinkertaisimmat featuret, jotka tehtiin, olivat lisäämällä kellonajasta tunti ja luokkahuoneiden tilavuus, joka saatiin rakennuksen pohjapiirustuksista. Päivänkin erottelua kokeiltiin, mutta se itseasiassa huononsi ennustusmallien tarkkuutta.

Toinen kokeilu oli lisätä säädataa datasettiin. Ideana oli selvittää, jos ulkoilma jotenkin vaikuttaisi sensorien mittauksiin tai jopa ihmismääriin. Tähän käytettiin meteostat-kirjastoa säädatan hankkimiseen samalta ajalta, miltä sensoridata oli [23]. Valitettavasti säädata ei juurikaan parantanut ennustusmallin tarkkuutta. Se tosin korreloi sensoridatan ilmankosteuden ja lämpötilan kanssa, joilla ei ole vaikutusta ihmismääriin.

Lopullinen featurematriisi on siis: Hiilidioksidi, valonmäärä, lämpötila, ilmankosteus, huoneen tilavuus, & tunti.

7.3 Koneoppimismallin opetus & tulokset

7.3.1 Datasetin jakaminen

Datasetti pitää jakaa harjoitus- ja testaussetteihin, yleensä 80/20-suhteella. Idea on harjoittaa arviointimalli 80 % datasta ja tarkistaa tulokset uudella datalla, jota malli ei ole koskaan nähnyt. Periaatteessa tällä vältetään lisädatan kerääminen, kun mallin tarkkuutta voi arvioida yhdellä datasetillä. Käytin datanjakamiseen Scikit-learn-kirjastoa [24], joka jakaa datan niin, että harjoitus- ja testiseteissä on tarpeeksi monipuolista dataa. Esimerkiksi, jos sinulla on 100 datapistettä joka tunnille, naiivijako on ottaa tuntien 0–19 (1920 pistettä) väliset pisteet harjoitussettiin ja 20–24 (480 pistettä) testisettiin. Oikeaoppinenjako olisi ottaa 20 pistettä per tunti (480 pistettä) testisettiin.

7.3.2 Regressio-algorytmin valitseminen

Benchmarkkasin erilaisia regressio-algorytmejä, jotka kätevästi löytyivät scikit-learn-kirjastosta. Scikit-learn tarjoaa kattavan valikoiman erilaisia koneoppimismalleja, luokittelusta klusterointiin. Käytin tätä kirjostoa mm. datasetin normalisointiin (arvot skaalataan 0 ja 1 välille) ja sen jakamiseen harjoitus- ja testisetteihin kuin myös seuraavien algorytmien testaamiseen. [24].

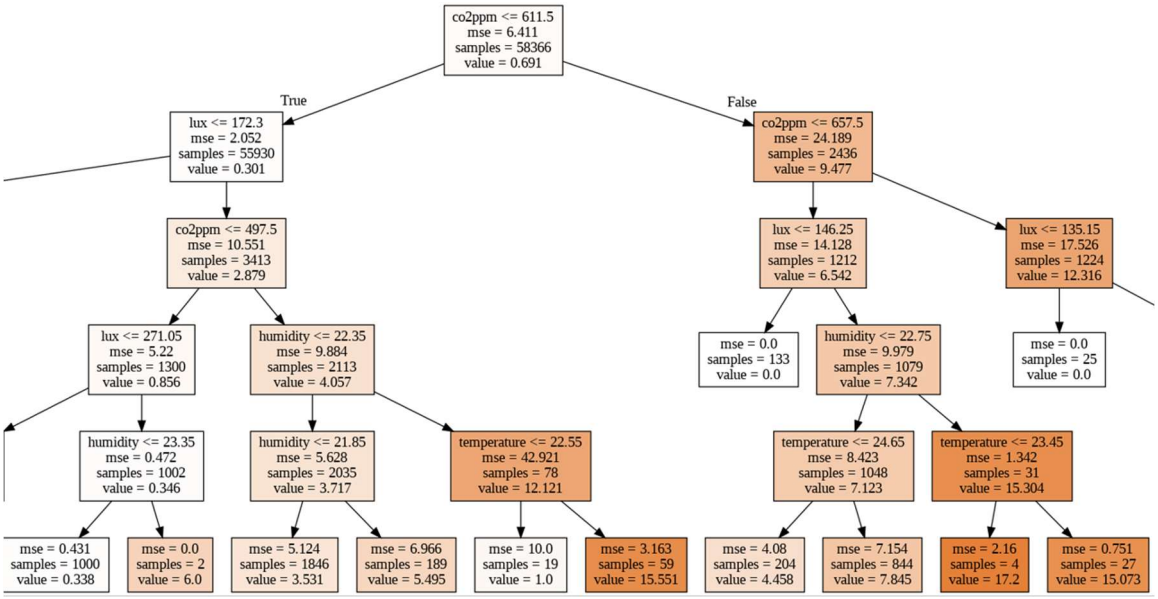
Malli	Tark-kuus	Ennustus	Oikea arvo
Linear Regression	56.7	0.44	0.0
Ridge	56.7	0.44	0.0
RidgeCV	56.7	0.44	0.0
SGDRegressor	56.72	0.44	0.0
Lasso	42.6	0.27	0.0
LassoCV	56.71	0.44	0.0
LarsCV	56.7	0.44	0.0
ElasticNet	-0.03	0.68	0.0
OMP	41.15	0.11	0.0
ARD	56.7	0.44	0.0
Bayesian	56.7	0.44	0.0
Poisson	16.68	0.63	0.0
Tweedie	9.34	0.63	0.0
PassAgg	55.73	0.37	0.0
Bayes_ARD	56.7	0.44	0.0
Bayes_Ridge	56.7	0.44	0.0
DecisionTree	93.75	0.0	0.0
RandomForest	95.61	0.0	0.0

Taulu 2. Benchmarkatut algoritmit. Puuarkkitehtuurimallit suoriutuvat huomattavasti parhaiten.

DecisionTree -algoritmi muistuttaa rakenteeltaan nimellisesti puuta: ”Runkosolmu” edustaa koko näytettä, joka pilkkoutuu pienempiin ”Sisäisiin solmuihin”. Jokainen ”sisäinen solmu” edustaa datasetin featurea ja tekee true/false -päätöksen, luoden uuden ”oksan”. Lopussa oleva ”lehti solmu” on tulos. Tämä algorytmi siis tekee paljon päätöksiä, jotka jakautuvat ja kasvavat uusiin päätöksiin. Näistä päätöksistä otetaan keskiarvo ja tämä on koko algoritmin antama ennustus. [25.]

Yksi DecisionTree -algoritmin ongelmista on ylisovitus, eli malli ei osaa yleistää oppimaansa ja toimii hyvällä tarkkuudella vain opetusdatasetille. RandomForest -algoritmi lieventää tätä ongelmaa. RandomForest -algoritmi toimii ottamalla monen muun antaman tuloksen keskiarvon, joka toimii lopullisena ennustuksena. Verrattuna DecisionTree -algoritmiin, jokainen näistä puista on

satunnaisesti luotu kaavojen sijaan, joka pienetää mallin ylisovittumista. Monta ”puuta” luo ”metsän”, tästä nimi ”Random Forest”. [26.] Kuva 8 visualisoi, miltä RandomForest -regressori näyttää.



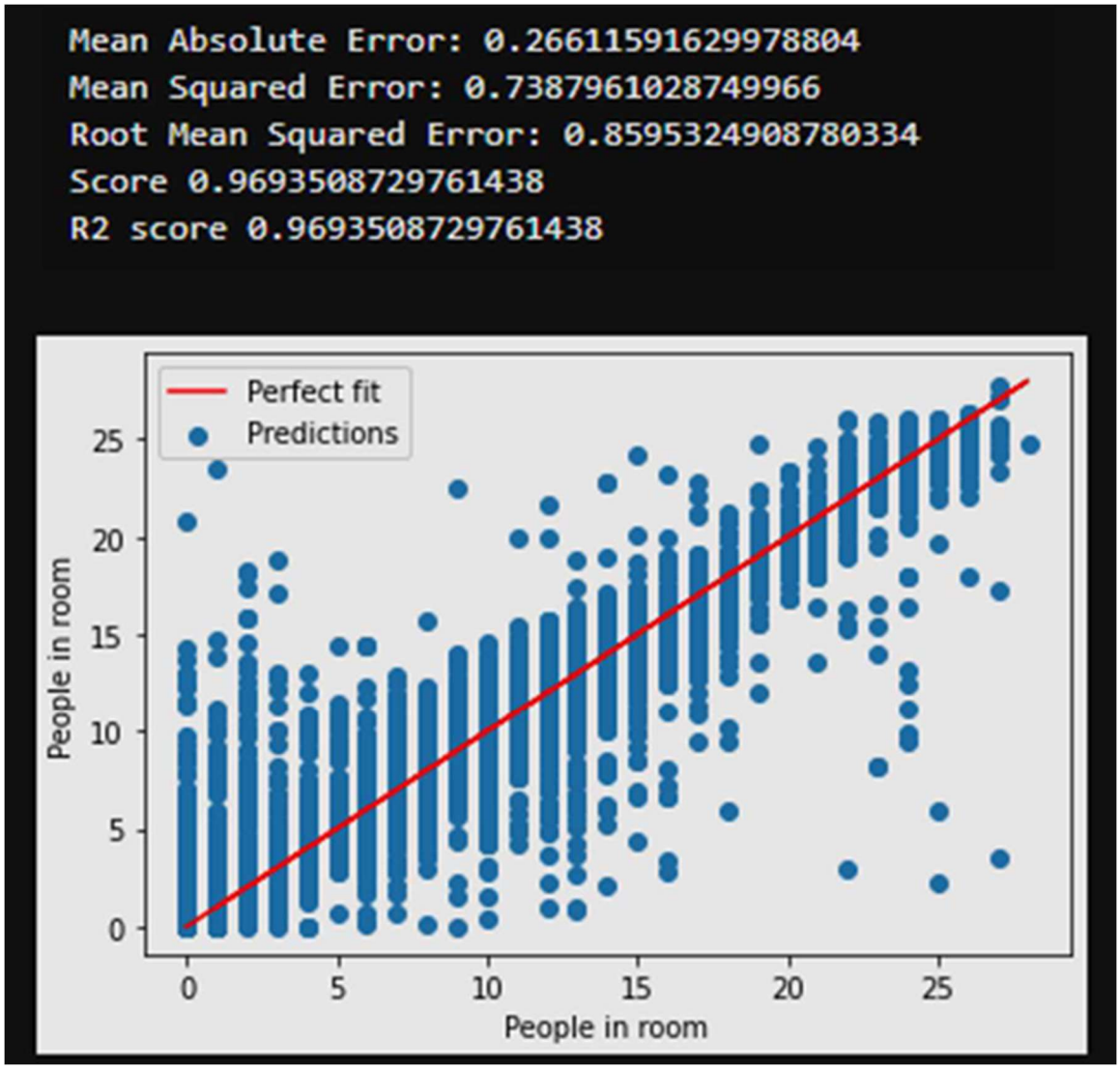
Kuva 8. RandomForest -mallin päätöksiä opetusdatalle. Huomaa ”sisäisissä solmuissa” olevat featureiden nimet.

7.3.3 RandomForest -mallin tulokset

RandomForest -regressori R2 oli n. 0.96, eli mallin voisi sanoa olevan 96 % tarkka. Tämän luulisi olevan erinomainen tulos, mutta pitää ottaa huomioon, että n. 96 % datasta on ns. ”nolladataa”, eli dataa, kun huoneessa ei ole yhtään henkilöä. Malli ei siis oikeasti ennusta 91 % ajasta oikein. Tämä hypoteesi konfirmoitiin ottamalla opetusdatasta pois kaikki ”nolladata”, jolloin ennustustarkkuudet putosivat n. 48 %:iin, 43 %, joka on huonompi kuin koko datalla. Eli koko datalla 4 % väärin tarkoittaa 52 % väärin ”ei nolladatalle”. Tämä näkyy itse ennustuksissa: Ennustukset heittävät yhdellä tai kahdella ihmisellä hyvin usein. Yleisin ongelma on 1 ihmisen ennustus, kun huoneessa on 0 henkilöä. Ennustus voi myös vaihtua, vaikka sensoridata ei muutu. Heittely voi tapahtua milloin tahansa, tosin se ei kuitenkaan anna satunnaista arvoa. Kuva 9 havainnollistaa tätä hajontaa hyvin.

Ennustus kuitenkin on puolet ajasta oikein ja ottaen huomioon, ettei kyseessä ole binäärivalinta vaan liukuluvun ennustus, tulos on miellyttävä eikä sitä luokitella 50/50-mahdollisuus valinnaksi.

Oman ja Juha Hauhian mielipiteen perusteella, ihmismäärän ennustuksen heittely 1–5 välillä o täysin hyväksyttävää.



Kuva 9. RandomForest -mallin ennustukset. Mitä kauempana diagonaalia, sitä suurempi virhe. Ideaalisesti siis kaikki sijoittuisivat diagonaalille

7.4 Neuroverkon opetus & tulokset

Käytin Pythonin Tensorflow-kirjastoa kaikkien neuroverkkojen benchmarkaamiseen. Loin erilaisia verkkoja erilaisilla rakenteilla:

1. Lineaarimalli Sisältää vain yhden tiheän kerroksen. Sisääntulo menee suoraan ulostuloon.

2. Tiheämalli Sisältää 4 kiinteää tasoa, joissa on asteittain vähemmän neuroneja (1000, 100, 50, 1). Jokaisessa kerroksessa on myös joukkonormalisointi ja dropout-kerrokset.

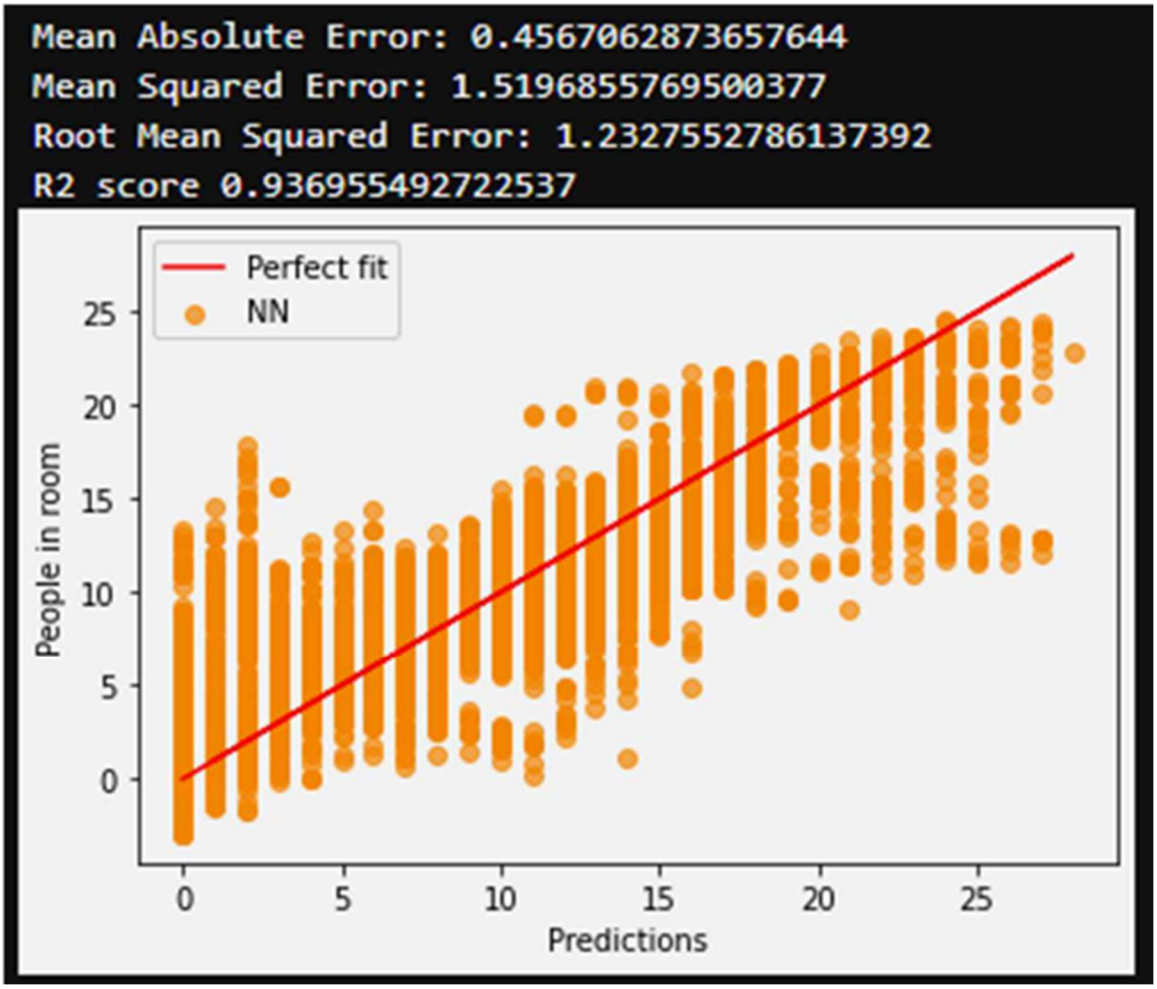
3. Moniaskelinen kiinteämalli Sisältää flatten-kerroksen, kaksi kerrosta 128 neuronilla ja yhden ulostulokerroksen. Koko pino lisätään seuraavaan sisääntuloon ”([outputs] => [1, outputs])”. Tällöin verkko pitää yllä kontekstia nykyisille arvoille sisääntulossa.

4. Konvoluutiomallissa oleva konvoluutiokerros luo ”ikkunan”, joka ottaa tietyltä alueelta sisääntulomatriisista keskiarvon. Nämä keskiarvot siirtyvät eteenpäin toiseen konvoluutiokerrokseen. Nämä keskiarvot yhdistetään ulostuloksi tiheällä kerroksella.

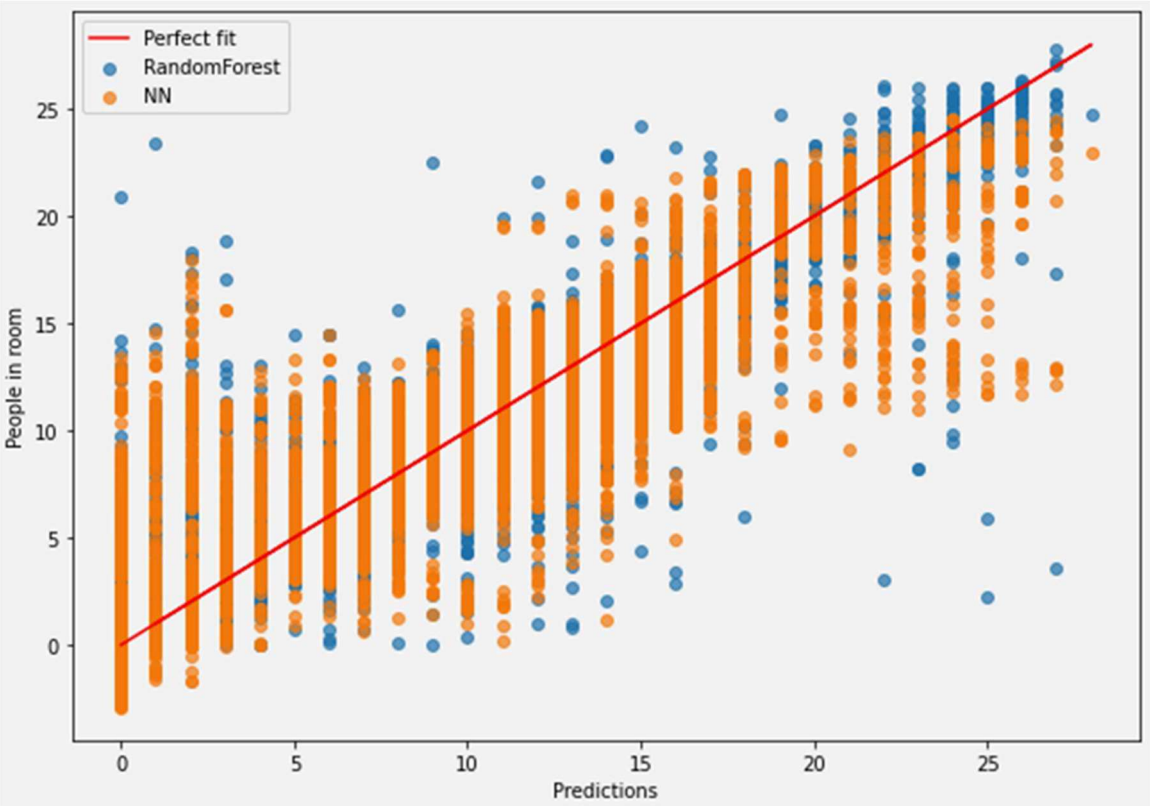
5. LSTM-malli eli Long-Short-Term Memory -malli (LSTM) sisältää LSTM-soluja, jotka päättävät painoarvojen avulla, mitä unohtaa ja mitä muistaa. LSTM-malli pitää täten yllä yleiskäsitystä kaikista edellisistä askelista kuin myös edellisestä luoden hyvän kontekstin nykyiselle askeleelle.

Parhain neuroverkkorakenne oli relatiivisesti yksinkertainen Tiheäverkko 3 tasolla. Neuroverkon ennustustarkkuus oli 93 %, eli 3 % huonompi kuin RandomForest -regressorilla. Mutta jos katsoo kuvaa 10, missä näkyy neuroverkon ennustukset vs. ideaali, sillä on vähemmän radikaaleja virheitä, mutta myös omituisia ”aukkoja”. Aukot tarkoittavat neuroverkon välttelevän tiettyjä arvoja tietylle datalle.

Kuva 11 Näyttää RandomForest ja neuroverkkomallin ennustuksien erot. RandomForestilla on enemmän hajontaa, kun taas neuroverkko ennustaa vain useammin väärin.



Kuva 10. Neuroverkon ennustukset.



Kuva 11. RandomForest- ja neuroverkkomallin (NN) ennustuksien ero.

Näiden tuloksien perusteet päätin valita RandomForest -regressorin. Se oli 3 % parempi kuin neuroverkko ja ennusti vähemmän ajasta väärin. Mieluummin valitsen mallin, joka antaa 5/10 väärin, joista yksi on hyvin radikaali virhe, kuin neuroverkon, joka ennustaa 7/10 väärin.

7.5 Ihmismäärän arviointi -pipeline

Kun sopiva malli oli valittu ja opetettu, tallensin sen ja normalisoijan pickle-tiedostoksi (.pkl). Pickle-tiedosto on serialisoitu Python-objekti [27] ja sillä on todella kätevä tallentaa ja ladata erilaisia koneoppimismalleja. Pythonin konoppimiskirjasto scikit-learn tukee tätä tiedostomuotoa [28]. Tensorflow-malleja suositellaan tallentamaan joko .h5- tai .keras-tiedostoiksi [29], mutta ne voi myös tallentaa .pkl-tiedostoiksi SavedModel-formaatissa. SavedModel-tiedosto sisältää Tensorflow-ohjelman sisältäen opetetut parametrit ja laskennat. Se ei tarvitse täten alkuperäistämallia toimiakseen. [30.]

Malli ja normalisoija ladataan Python-ohjelmalla. Sama ohjelma yhdistää sitten InfluxDB-tietokantaa ja hakee esim. edellisen 3 minuutin ajalta sensoridataa. Dataa täten tulee jopa 500 riviä. Tälle on kaksi vaatimusta:

1. Tietokannassa jokaisen huoneen data on samassa taulussa, eli jos hakee vain 10 riviä, ei saa välttämättä kuin kahden huoneen datan siltä ajanhetkeltä.

2. Ennustus on tarkempi, jos yhdeltä huoneelta on enemmän kuin 1 rivi dataa.

Tälle datalle tehdään sitten ennustus, eli jokainen rivi saa ihmismääränarvion. Tämä sama data sitten tallennetaan uudestaan tietokantaan aikaleimojen avulla. Ohjelma sitten odottaa esim. 15 sekuntia, kunnes hakee uudestaan dataa ja tekee sille ennustuksen. Ohjelmalle voi syöttää argumentteja, joissa määritellään, kuinka pitkältä aikaväliltä dataa haetaan ja sen intervalli. Testailujen jälkeen, tuo esimerkkien 3 minuutin ajalta dataa 15 sekunnin välein oli sopiva, koska ihmismäärän arviota ei tarvitsisi päivittää reaaliajassa ja mallin ennustus heittelee samalla datalla jonkin verran.

Lisäksi ennustuksista tehtiin ”konservatiivisia” pyöristämällä ne alaspäin esim. 0.7 = 0. Tällä yritettiin välttää pomppailua ennustuksissa, joissa tyhjissä huoneissa ennustus kävisi yhdessä tai kahdessa henkilössä vähän väliä.

Koska ennustus kirjoitetaan samaan tauluun tietokannassa, se on todella helppo lisätä Grafanaan. Haut ovat täysin samanlaiset kuin muille sensoriarvoille, vain kolumnin nimi pitää vaihtaa. Kuva 12:sta oikeassa alakulmassa näkyy ihmismääränarvio. Tätä voi helposti verrata sensoridataan ja katsoa, näkeekö silmäpelillä mahdollisia muutoksia siinä, kun ihmismäärän arvio muuttuu.

Tämä kokonaisuus on Python-skripti, joka pyörii Docker-kontissa KAMK:in supertietokone BULLilla. BULLin pitäisi olla päällä 24/7 vähäisillä katkoksilla ja skripti ei kaadu, jos se menettää yhteyden InfluxDB-tietokantaan.

Kuva 12. Grafana-kojelauta, missä näkyy reaaliajassa huoneiden sensoridataa.

8. Domain-sovitusongelma (Domain adaption)

Seuraavaksi esitelty ongelma, mikä ilmeni malleja opetettaessa. Opinnäytetyössä taikka hankkeessa ei riittänyt aika tutkia tätä monimutkaista asiaa tarkemmin.

Emme pystyneet mittaamaan ihmismäärää kuin kahdesta eri luokasta kerrallaan hankeajalla. Meillä oli vain 2 ihmismäärälaskuria, joten mallejakaan ei voitu opettaa kuin kahdelle eri huoneelle. Malli pystyy tekemään ennustusta muillekin huoneille, mutta koska jokaisen huoneen sensoriyksikkö lähettää hieman erilaista ja uniikkia dataa, niiden lähtötasot eivät ole samat. Malli näkee dataa, joka on keskiarvoltaan esim. 2 pistettä korkeampi kuin opetusdatassa. Koska ominaisuudet feature-matriisissa olivat positiivisesti korreloituvia, ennustuksen pitäisi myös olla korkeampi. Tämä näkyy käytännössä: Huoneissa, joissa on esim. normaalisti korkeampi hiilidioksiditaso kuin opetusdatahuoneissa yleensä saa ihmismäärän ennustuksena ”1”, kun taas opetusdatahuone on ”0”. Kumpikin huone on tyhjä, jolloin malli ennustaa väärin toiselle huoneelle. Sensoriyksiköitä oli 15 eri huoneessa, joilla kaikilla on erilainen baseline-taso. Meillä opetusdataa on vain kahdesta eri huoneesta, joilla on myös omat baseline-tasonsa. Kaksi huonetta ei siis kuvaa kaikkia muita huoneita todellisesti.

Ongelma syntyy siis opetus- ja validointidatan puutteesta ja miten sen kerääminen on liian kallista ja työlästä. Tämä on tunnettu koneoppimisen ongelma varsinkin konenäön puolella, jota pyritään ratkaisemaan erilaisilla menetelmillä [31, s. 1; 33, s. 10]. Sitä kutsutaan Domain-sovitukseksi (Domain adaption) [32; 33, s. 2].

Domainin-sovitus on koneoppimistekniikka, joka auttaa siirtämään malleja tietoalueelta toiseen. Sitä käytetään parantamaan mallin tarkkuutta käyttämällä lähdealueen tietoa ja soveltamalla sitä kohdealueelle. Domain adaptaation tavoitteena on kuroa umpeen kahden eri tietoalueen välinen kuilu, jotta malli voi oppia tehokkaasti molemmista [33, s. 1]. Domain-sovituksesta on tullut yhä tärkeämpää syväoppimisen ja big datan aikakaudella, koska se mahdollistaa mallien soveltamisen laajempiin ongelmiin ja on myös tärkeä osa-alue esim. kuvan generoinnissa [32; 33, s. 23–27], jossa halutaan ”yleistää” esim. kuva ja piirustus tuolista.

8.1 Vastakkaisdomainiset neuroverkot (Domain-Adversarial Neural Networks)

Vastakkaisdomainiset neuroverkot (DANN) on syväoppimismalli, joka mahdollistaa valvomattoman toimialueen eli domainin mukauttamisen. DANN:it on koulutettu oppimaan jaetun esityksen kahdessa eri domainissa ja käyttämään ylimääräistä vastakkaisverkkoa erottamaan nämä kaksi aluetta. Vastakkaisverkko on vastuussa kahden domainin ominaisuuksien välisen erojen minimoimisesta, kun taas jaetut ominaisuudet on optimoitu maksimoimaan kohdedomainin ennustus/luokitustarkkuuden. [33, s. 7.] Vastakkaisverkko pyrkii sovittamaan eri domainit mahdollisimman yhtenäisiksi ottaen huomioon niiden luontaiset erot ja monet ratkaisut olettavatkin, että lähdedomainin riskit ovat samat kuin kohdedomainissa [33, s. 2].

Vastakkaisdomainiset neuroverkot ovat vastaus yleistysongelmaan, koska opetusdatan luokittelu ja mallien harjoittaminen joka ikiselle mahdollisuudelle on työlästä ja aikaa vievää. Jos mallit pystyisivät yleistämään syötetyn datan, säästyttäisiin suurelta työltä. [32]

9. Loppupäätelmät

Hankkeen tavoitteet saavutettiin ja erilaisia näkemyksiä tuli ilmi. Ihmismäärän arviointi ja mittaus sensoridatasta on yllättävän monimutkainen ongelma, jota pyrittiin ratkaisemaan koneoppimis- ja tekoälymenetelmillä. Ennustuksia saatiin aikaan miellyttävällä tasolla yhdistettyä kontrollipaneeliin, joista saatiin yllättyneitä kehuja ennustuksien relatiivisesta tarkkuudesta. SmartCampus-hankkeella oli hyvin lepsut odotukset: Yksinkertaista R&D:tä ja ettei ongelmaan välttämättä saataisi ratkaisua. Suoraan sanottuna hyppäsimme odotuksien yli hyvin korkealta, kun edes todistimme ihmismäärän arvioin derivoinnin sensoridatasta mahdolliseksi.

Opinnäytetyö esittelee jatko-ongelman Domain-sovituksesta ja pyrkii tuomaan ongelman yleiseen tietoon ja toimii myös jatkokehityslähtökohtana, jos hanketta jatketaan tulevaisuudessa. Ihmismääränarvion tarkkuuden optimointi Domain-sovituksen suhteen olisi yksi jatkokehityskohde ja myös tärkein ongelma, jota ei pääse karkuun ainakaan tällä metodilla.

Toinen ratkaisu voisi olla myös halvempien ihmislaskurien rakentaminen, eli halvempaa rautaa, joka lähettäisi videokuvan prosessoitavaksi, vaikka KAMKin supertietokone BULLille. Tästä keskustelimme vaikutuksenarviointia tehdessä. Tässä tulee tietoturvaongelma, kun lähetetään videokuvaa jonnekin eri paikkaan verkon kautta prosessoitavaksi. Todennäköisesti olisi vaikeampi saada lupia laittaa kameroita täten luokkiin, vähintään niiden turvallisuuteen pitäisi panostaa paljon. Kameroille pitäisi luoda jonkin sortin oma suljettuverkko, minne ei pääsisi käsiksi ulkopuolelta millään konstilla; aika lähellä, miten CCTV toimisi. Tällaisen saaminen hyväksytysti vaatisi hyvin paljon turvatarkastuksia ja kaikenlaisten ihmisten osallistumista, rehtorista juristeihin.

Tämä ratkaisu tosin hyppää kokonaan sensoridatan yli: Todennäköisesti ihmislaskureita käytettäisiin suoraan, eikä vain opetusdatan keräämiseen. Eli tämän voisi ajatella ”bruteforce” ratkaisuna, jolla välttää Domain-sovitus haasteen kokonaan.

10. Lähteet

1. Mohammadhossein Toutiaee. Occupancy Detection in Room Using Sensor Data [Internet]. 10.1.2021 [viitattu 23.3.2023]. Saatavilla: https://doi.org/10.48550/arXiv.2101.03616

2. Chenli Wang, Jun Jiang, Thomas Roth, Cuong Nguyen, Yuhong Liu, Hohyun Lee. Integrated sensor data processing for occupancy detection in residential buildings [Internet]. 15.4.2021 [viitattu 23.3.2023]. Saatavilla: https://doi.org/10.1016/j.enbuild.2021.110810.

3. Pressac. Types of occupancy monitoring sensors and their uses. [Internet]. 3.2.2021 [viitattu 23.3.2023]. Saatavilla: https://www.pressac.com/insights/types-of-occupancy-monitoring-sensors-and-their-uses/

4. Espressif. ESP32. [Internet]. 2023 [viitattu 4.1.2023] Saatavilla: https://www.espressif.com/en/products/socs/esp32

5. What Is the Raspberry Pi?. [Internet]. 28.9.2021 [viitattu 27.3.2023]. Saatavilla: https://www.howtogeek.com/754492/what-is-raspberry-pi/

6. influxdata. InfluxDB. [Internet]. 2023 [viitattu 4.1.2023] Saatavilla: https://www.influxdata.com/products/influxdb-overview/

7. Adith Bharadwaj. Introduction to InfluxDB. [Internet]. 20.1.2021 [viitattu 27.3.2023] Saatavilla: https://www.section.io/engineering-education/introduction-to-influxdb/

8. GrafanaLabs. What is Grafana?. [Internet]. 2023 [viitattu 30.3.2023]. Saatavilla: https://grafana.com/oss/grafana/?plcmt=footer

9. GrafanaLabs. Plugins. [Internet]. 2023 [viitattu 30.3.2023]. Saatavilla: https://grafana.com/grafana/plugins/?plcmt=footer

10. OpenCV. About. [Internet]. 2023 [viitattu 27.3.2023] Saatavilla: https://opencv.org/about/

11. Daniel Johnson. What is TensorFlow? How it Works? Introduction & Architecture. [Internet]. 25.3.2023 [viitattu 30.3.2023]. Saatavilla: https://www.guru99.com/what-is-tensorflow.html

12. Rohit Kundu. YOLO: Algorithm for Object Detection Explained [+Examples]. [Internet]. 2.3.2023 [viitattu 26.3.2023]. Saatavilla: https://www.v7labs.com/blog/yolo-object-detection

13. Adrian Rosebrock. OpenCV Haar Cascades. [Internet]. 12.4.2021 [viitattu 30.3.2023]. Saatavilla: https://pyimagesearch.com/2021/04/12/opencv-haar-cascades/

14. Jupyter. [Internet]. 2023 [viitattu 27.3.2023]. Saatavilla: https://jupyter.org/about

15. Juha Hauhia. KAMK-anturiverkko – sensoriverkko luokkatilojen monitorointiin. [Internet]. 2020 [viitattu: 27.3.2023]. Saatavilla: https://urn.fi/URN:NBN:fi:amk-2020060316554

16. KAMK-Anturiverkko. [Internet]. 2023 [viitattu 26.3.2023]. https://anturiverkko.fi

17. Slant. ESP32 vs. RaspberryPI 4 model B. [Internet]. 2023 [viitattu 30.3.2023]. Saatavilla: https://www.slant.co/versus/36414/34414/~esp32_vs_raspberry-pi-4-model-b

18. Adrian Rosebrock. OpenCV People Counter. [Internet]. 13.8.2018 [viitattu 15.2.2023]. Saatavilla: https://pyimagesearch.com/2018/08/13/opencv-people-counter/

19. Lee Jackson. OV2640 vs OV7670: Spec Comparison, Sample Images, Datasheets, Lens Options, Pinouts, Compatibility, and Tutorials. [Internet]. 1.8.2021 [viitattu 15.2.2023]. Saatavilla: https://www.arducam.com/ov2640-vs-ov7670-detailed-comparisons-and-resources/

20. OpenCV. Image Thresholding. [Internet]. 2023 [viitattu 03.02.2023]. Saatavilla: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html

21. GeeksforGeeks. Object Detection vs Object Recognition vs Image Segmentation. [Internet]. 28.7.2022 [viitattu 21.3.2023]. Saatavilla: https://www.geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/

22. Qengineering. YoloV4 Raspberry Pi 4. [Internet]. 16.1.2023 [viitattu 03.02.2023]. Saatavilla: https://github.com/Qengineering/YoloV4-ncnn-Raspberry-Pi-4

23. Pypi. Meteostat. [Internet] 2023 [viitattu 30.3.2023]. Saatavilla: https://pypi.org/project/meteostat/

24. API Reference. Scikit-learn. [Internet]. 2023 [viitattu 1.3.2023] Saatavilla: https://scikit-learn.org/stable/modules/classes.html

25. IBM. What is a Decision Tree?. [Internet]. 2023 [viitattu 23.3.2023]. Saatavilla: https://www.ibm.com/topics/decision-trees

26. Nima Beheshti. Random Forest Regression. [Internet]. 2.3.2022 [viitattu 23.3.2023]. Saatavilla: https://towardsdatascience.com/random-forest-regression-5f605132d19d

27. Python.org. pickle — Python object serialization. [Internet]. 2023 [viitattu 26.3.2023] Saatavilla: https://docs.python.org/3/library/pickle.html

28. Jason Brownlee. Save and Load Machine Learning Models in Python with scikit-learn. [Internet] 8.6.2016 [viitattu 26.3.2023]. Saatavilla: https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/

29. Tensorflow. Save and load Keras models. [Internet]. 2023 [viitattu 26.3.2023]. Saatavilla: https://www.tensorflow.org/guide/keras/save_and_serialize

30. Tensorflow. Using the SavedModel format. [Internet]. 2023 [viitattu 26.3.2023]. Saatavilla: https://www.tensorflow.org/guide/saved_model

31. Harsh Maheshwari. Understanding Domain Adaptation. 4.5.2021 [viitattu 26.3.2023]. Saatavilla: https://towardsdatascience.com/understanding-domain-adaptation-5baa723ac71f

32. Rohit Kundu. Domain Adaptation in Computer Vision: Everything You Need to Know. [Internet]. 2.3.2023 [viitattu 26.3.2023]. Saatavilla: https://www.v7labs.com/blog/domain-adaptation-guide

33. Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, Victor Lempitsky . Domain-Adversarial Training ofNeural Networks. [Internet]. 26.5.2016 [viitattu 26.3.2023] Saatavilla: https://doi.org/10.48550/arXiv.1505.07818