Tampere University

Leevi Hietamäki

Indoor Localisation Using Ion Mobility Spectrometry

Master's thesis
Faculty of Information Technology and Communication Sciences (ITC)
Examiners: Professor Joni Kämäräinen
Professor Esa Rahtu
July 2025

Abstract

Leevi Hietamäki
Indoor Localisation Using Ion Mobility Spectrometry
Master's thesis
Tampere University
Master's Programme in Electrical Engineering: Advanced Studies in Signal Programming and Machine Learning
July 2025

This thesis investigates the use of machine learning methods for indoor location classification based on scent profiles of different locations, measured using Ion Mobility Spectrometry (IMS). The IMS measurements were performed using an electric nose (eNose) device called ChemPro100i. The thesis's purpose is to assess the efficiency of several classification algorithms and their capacity to generalize to real-world scenarios. Four primary methods, K-Nearest Neighbours (KNN), Random Forest (RF), Gradient Boosting Decision Tree (GBDT), and Convolutional Neural Networks (CNN), were tested on a dataset collected from various measurement locations mostly within university campus. While this thesis uses existing implementations for KNN, RF and GBDT algorithms, it introduces a novel CNN model specifically designed for captured IMS measurements.

The results show that RF achieved the highest overall accuracy. While the proposed CNN model demonstrated strong performance in certain runs, its high variance led to a lower average accuracy, making its overall performance less consistent and reliable. In addition to the baseline study, data augmentation approaches were used to increase the training set and improve generalization, especially for our CNN model.

Confusion matrices were analyzed to better understand classification results across different locations. The findings indicate that, while machine learning models can learn to classify IMS-based location data to some extent, external factors present in real-world environments significantly disrupt IMS measurements, reducing the classification methods ability to effectively distinguish between measurement locations. Future research should concentrate on combining multi-sensor data, enhancing model robustness, and collecting a larger data set before these methods can be reliably implemented in practical applications.

Keywords: Indoor localisation, Ion mobility spectrometry, Machine Learning, Convolutional Neural Network, Data augmentation

Tiivistelmä

Leevi Hietamäki
Ionin liikkuvuus spektrometrin hyödyntäminen sisätilapaikannuksessa
Diplomityö
Tampereen yliopisto
Sähkötekniikan DI-ohjelma: Signaalinkäsittelyn ja koneoppimisen syventävät opinnot
Heinäkuu 2025

Tämä diplomityö tutkii koneoppimismenetelmien hyödyntämistä sisätilapaikannuksessa, jossa paikannus perustuu eri tilojen ominaistuoksuihin. Työssä otetut mittaukset suoritettiin ionin liikkuvuus spektrometri (IMS) -tekniikkaa hyödyntyvän elektronisen nenän (eNose) ChemPro100i-laitteen avulla. Työn tavoitteena on arvioida useiden luokittelualgoritmien tehokkuutta ja niiden kykyä yleistää todellisissa käyttötilanteissa. Neljää pääasiallista koneoppimismenetelmää, K-Nearest Neighbors (KNN), Random Forest (RF), Gradient Boosting Decision Tree (GBDT), and Convolutional Neural Networks (CNN), testattiin aineistolla, joka kerättiin useista mittauspisteistä pääsäätöisesti yliopistokampuksen alueella. KNN-, RF- ja GBDT-algoritmeille käytettiin valmiita toteutuksia, mutta IMS-mittausten luokittelua varten kehitettiin uusi, diplomityötä varten suunniteltu CNN-malli.

IMS mittauksista saadut luokittelutulokset osoittavat, että RF saavutti parhaan kokonaisluokittelutarkkuuden. Vaikka ehdotettu CNN-malli saavutti ajoittain hyviä tuloksia, suuri vaihtelu sen suorituskyvyssä johti heikompaan keskimääräiseen tarkkuuteen ja luotettavuuteen eri opetuskertojen välillä. Diplomityö sisältää myös toisen kokeilun, jossa hyödynnettiin datan augmentointimenetelmiä. Toisen kokeilun tavoitteena oli parantaa erityisesti CNN-mallin yleistämis- ja suorituskykyä.

Tuloksia analysoitiin myös sekaannusmatriisien avulla, jotta mallien käyttäytymistä eri sijainneissa voitiin ymmärtää paremmin. Tulokset viittaavat siihen, että koneoppimismallit kykenevät oppimaan IMS-pohjaista sijaintiluokittelua jossain määrin, mutta todellisten käyttöympäristöjen ulkoiset tekijät häiritsevät IMS-mittauksia ja heikentävät mallien kykyä erottaa eri paikat toisistaan. Mahdollisissa tulevissa tutkimuksissa tulisi keskittyä monianturillisen datan yhdistämiseen, mallien toimintavarmuuden parantamiseen sekä suuremman aineiston keräämiseen, ennen kuin näitä menetelmiä voidaan luotettavasti hyödyntää käytännön sovelluksissa.

Avainsanat: Sisätilapaikannus, Ionin liikkuvuus spektrometri, koneoppiminen, konvoluutiohermoverkko, datan augmentointi

PREFACE

I would like to express my sincere gratitude to my supervisors professor Esa Rahtu and professor Joni Kämäräinen. I am grateful for the opportunity to pursue this thesis topic and for the support I received throughout the process. Thank you for the guidance and support which made this thesis possible.

I also want to thank research group for Emotions, Sociality and Computing, as well as professor Veikko Surakka for lending me the ChemPro100i IMS device used in this thesis. I also want to express my gratitude towards Philipp Müller for his assistance in setting up the device.

Tampereella, 31st July 2025

Leevi Hietamäki

Contents

List of Figures

LIST OF SYMBOLS AND ABBREVIATIONS

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DIMS	Differential Ion Mobility Spectrometry
DMA	Differential Mobility Analyzer
DMS	Differential Mobility Spectrometry
DTIMS	Drift Time Ion Mobility Spectrometry
ELU	Exponential Linear Unit
eNose	Electrical Nose
FAIMS	Field Asymmetric Waveform Ion Mobility Spectrometry
GBDT	Gradient Boosting Decision Tree
IMCell	Ion Mobility Cell
IMS	Ion mobility Spectrometry
KLD	Kullback-Leibler Divergence
KNN	K-Nearest Neighbours
LR	Learning Rate
LReLU	Leaky Rectified Linear Unit
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RF	Random Forest
RMSProp	Root Mean Square Propagation
SGD	Stochastic gradient descent
SVD	Singular Value Decomposition
SVM	Support Vector Machine
t-SNE	T-distributed Stochastic Neighbour Embedding
tanh	Hyperbolic Tangent
TIMS	Trapped Ion Mobility Spectrometry
TWIMS	Travelling-wave Ion Mobility Spectrometry

Use of artificial intelligence in this work

Artificial intelligence (AI) has been used in generating this work:

☒ Yes
☐ No

I hereby declare, that the AI-based applications used in generating this work are as follows:

Application	Version
ChatGPT	GPT-4o
QuillBot Paraphrasing Tool	unknown version
Writefull for Overleaf	2.1.0

Purpose of the use of AI

ChatGPT was used during the thesis writing process to help with text clarity, fluency, and grammar of written content .

QuillBot was used selectively to rephrase short passages or individual sentences for improved readability and variation in expression.

Writefull for Overleaf feature provided automated language feedback, primarily focused on grammar and clarity directly within the LaTeXwriting environment.

Parts of this work, where AI was used

All of the AI tools were used throughout the text content of thesis. As Writefull is integrated with Overleaf, its features were available across all sections of the thesis. ChatGPT and QuillBot were used selectively whenever a part or sentence of the thesis needed structural feedback and refinement.

Acknowledgement of risks

I hereby acknowledge, that as the author of this work, I am fully responsible for the contents presented in this thesis. This includes the parts that were generated by an AI, in part or in their entirety. I therefore also acknowledge my responsibility in the case, where use of AI has resulted in ethical guidelines being breached.

1 Introduction

Indoor localization is the process of identifying a subject's location inside an indoor setting which enables navigating inside environments. Some animals are able to locate themself and navigate using olfactory cues from their environment [4], [49]. However, us humans use our technological advantages for indoor localization. These technologies include numerous radio signals, such as Bluetooth, ultra-wideband, and wireless local area networks. Alternative measurement methods contain data from light, sound, and magnetic fields.[9], [14], [53]

Since most animals have a better sense of smell than humans, we have developed electronic noses (eNose) for use in artificial olfaction. These electronic devices were designed to resemble the olfactory system of mammals. Typically, an eNose consists of an information processing unit, a multisensor array, databases from reference libraries, and software with digital pattern recognition algorithms. ENoses can identify and classify an analyte mixture from its distinct aroma signature. The eNoses are more precise than the human sense of smell and can detect chemicals in the analyte mixtures that are odorless. [74]

As we humans cannot use our sense of smell to navigate or locate ourselves, can we use the previously mentioned eNoses to help us with that? In this master's thesis, we explore the possibilities of ion mobility spectrometry (IMS) based eNose in indoor localisation.

In this paper, we focus on nonparametric fingerprint-localisation for localization using eNose measurements. The nonparametric fingerprint-localisation uses a database that includes measurements made at specified locations. These measurements are called fingerprints and they are gathered through a training phase. After that during a localisation phase, the user's location can be determined by comparing these fingerprints with measurements from the user's current location.[52], [53]

In this thesis, we use K-nearest neighbour (KNN), random forest (RF), and gradient boosting decision tree (GBDT) algorithms for the localisation. additionally, we implore the possibility of using convolutional neural networks (CNN) for the localisation problem.

Figure 1.1: ChemPro100i handheld chemical detector made by Environics that was used in this thesis. The image is from it's user manual [18].

2 Scent Detection

While gas sensor arrays are a feature of traditional eNoses, the same function can be achieved with ion mobility spectrometry technology [36]. When compared to gas sensor-based eNoses, IMS-based eNoses have the advantage that their electrodes and sensors do not deteriorate with time [53]. In this thesis, we use a ChemPro100i handheld chemical detector made by Environics as an eNose. This device is based on IMS and it was created for classifying and detecting chemical warfare agents in the field [19].

We can use the ChemPro100i to separate and identify ionized molecules, allowing it to analyze the air in a given area. The type of IMS that the device utilizes is a modified aspiration type IMS configuration patented by Environics. The modified aspiration type IMS works in an open-loop configuration that utilizes multisensor technology. [18] The multiple sensors on ChemPro100i are:

- Miniaturized IMS sensor with 16 measuring channels
- Semiconductor sensors
- MOS sensors
- Field Effect sensor
- Pressure sensor
- Humidity sensor
- Temperature sensors
- Flow sensor

With these sensors, the eNose is capable of real-time detection of chemical warfare agents from a sample. During analysis, these samples are continuously pumped to the Ionization chamber, where the radiation source of Am-241 ionizes it.[18] Then the ionized sample is forwarded to the ion mobility cell (IMCell). Inside the IMCell is an electric field that gives the mobility of the ions based on the ions' charge, size, and shape [17]. Ions of both polarities can be detected simultaneously by alternating the polarity of the electric field [51]. On the walls of the IMCell are 8 pairs of electrodes that correspond to the measurement channels. Information from the sample can be collected from 14 of these electrodes as the last pair is only used to control the speed of the airflow [19]. Depending on the Ion's mobility the Ion will hit one of these electrodes. The electrodes will absorb the ions from their surface, leading to a change in their electric resistance [19]. Electric current can be measured from each of these electrodes, which are then used to create a chemical pattern of the sample. The chemical patterns provide both quantitative and qualitative information about the sample under analysis [12].

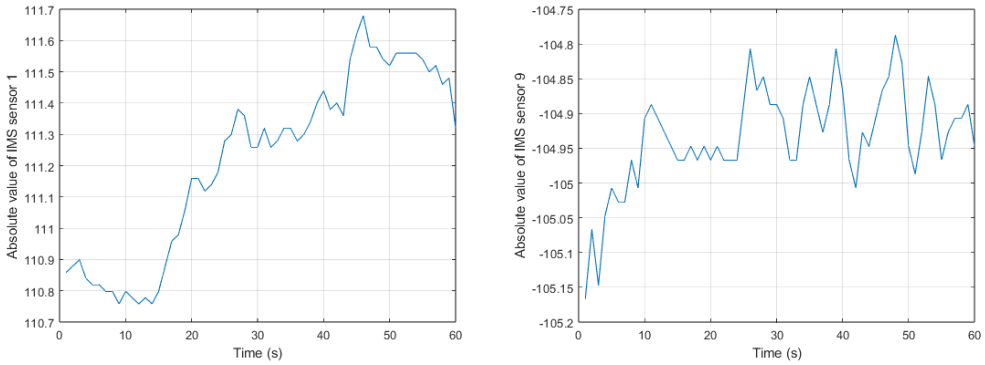


Figure 2.1: Example plots of IMS measurements recorded using the ChemPro100i device. The absolute values of IMS sensors 1 and 9 are plotted over time. Mean value of IMS sensor 1 is 111.2187 and the mean value of IMS sensor 9 is -104.9333.

2.1 Ion Mobility Spectrometry

The definition of ion mobility spectrometry is the study of ions advancing in gases in the presence of an electric field. The IMS can also be described as a technique of ion separation as the ions separate according to variations in ion mobilities in the presence of an electric field. The process of ion separation can be utilized in sample identification. In this section, we go over the fundamentals of ion mobility spectrometry.

At the start of an analysis, the IMS device collects a sample, which is a tiny but representative portion of an unidentified gas combination. After acquiring the sample, it needs to be ionized. The radioactive atmospheric pressure chemical ionization technique is the most widely used ionization method in IMS. The Radioactive ionization is usually done by a β-source from Ni-63 and tritium or α-source of Am-214. Other suitable ionization methods for IMS include Corona discharge atmospheric pressure chemical ionization, laser desorption/ionization technique, and Pressure photo ionization. [12]

Ions are directed toward the separation region after they have ionized. The separation of ions in a buffer gas under the influence of an electric field is the fundamental idea behind IMS instrumentation [15]. In IMS, friction with the buffer gas precisely balances the coulomb forces an electric field E applies to an analyte ion, resulting in a steady-state analyte velocity v_d [12]. The analyte's mobility K can be expressed as:

$$K = \frac{v_d}{E} \tag{2.1}$$

Usually, the ion's mobility K is an expression of experimental variables (temperature T and pressure ρ), which frequently get normalized to standard conditions to determine reduced mobility K_0 [15]:

$$K_0 = K\frac{\rho}{\rho_0}\frac{T_0}{T} \tag{2.2}$$

The collision cross section Ω and number density N (the number of molecules per unit volume) both affect an ion's mobility [12]. K can be expressed as a Mason-Schamp equation when v_d is small concerning the ion thermal velocity [24]:

$$K = \frac{3}{16}\sqrt{\frac{2\pi}{\mu k_B T}}\frac{ze}{N\Omega} \tag{2.3}$$

where μ is the ion-gas pair's reduced mass, k_B is Boltzmann constant, z is the ion's charge, and e is the charge of an electron. The μ can be calculated from the ion m and particle M masses $\mu = \frac{mM}{m+M}$.

After the ions have separated according to their mobilities the ions will finally reach a detector. Typically, an aperture grid or a Faraday's plate serves as the detector, but other types of detectors can be used to detect ions i.e. pairs of electrodes or beam stoppers [12]. The ion's mobility determines where the ion collides with the detector (i.e., which grid plate or electrode). The faster ions travel further before colliding with the corresponding grid plate or electrode [51]. When the ion collides with the detector, the ion's charge causes a change in a measured current from the detector and the ion is detected. The measured currents create a chemical pattern of the sample. The quantitative information is provided by the sum of the currents flowing through each grid plate/electrode. At the same time, the qualitative analysis is based on the pattern's shape, which is independent of concentration [70].

2.2 Ion Mobility Spectrometer Types

Recognizing that there are various IMS types is crucial. The ChemPro100i IMS devices utilize a modified aspiration-type IMS technology. In this section, we summarise the most common IMS technologies.

Drift time IMS (DTIMS) is the conventional setup wherein ions travel through an electric field in a drift tube while neutral gas molecules are present [12]. The uniform electric field that propagates through the drift region is the essential element of DTIMS. The analytes pass through the pressurized drift region, where there is no buffer gas flow, while subjected to a uniformly applied weak electric field. [15] Mobility K in DTIMS can be decreased, demonstrating that number density N influences mobility in low electric fields rather than E and using the Mason-Schamp equation to determine the corresponding Ω values for analytes. The biggest benefit of DTIMS is probably its capacity to compute Ω from first principles. Comprehensive ion collection, in which all analyte mobilities are gathered in a single pulsed experiment, is another benefit of DTIMS.[12], [15].

Traveling-wave IMS (TWIMS) is schematically similar to DTIMS except that the electric field is not uniform. In TWIMS a stacked-ring ion guide provides traveling voltage waves [12]. These voltage waves propel ions through the drift region. Mobility separation of ions is accomplished by adjusting the traveling voltage wave's speed and magnitude [12]. One limitation with TWIMS devices is that, before determining the Ω values of unknown ions, each instrument needs to be calibrated

using ions with known mobility [15]. TWIMS has two main benefits: it can manipulate ion motion into long path length separations without significant ion loss, and it requires low voltage due to constant wave heights [15].

Trapped IMS (TIMS) is based on the application of a non-uniform electric field to hold ions stationary against a moving gas (trapping ions in place), allowing the ion packages to be separated according to their size-to-charge ratio [12]. For structural analysis, the electric field strength in a TIMS is gradually reduced to expel ions with particular mobilities from the mobility region. TIMS can only analyze the expelled ions. This characteristic makes TIMS extremely selective in separation efficiency, but it can also be advantageous in situations where ions have very similar mobilities. [15] Another benefit of TIMS is its compactness. The IMS devices utilizing TIMS are usually smaller than other devices that utilize other technologies.

Field asymmetric waveform IMS (FAIMS) is an IMS type that capitalizes on variations in ion mobility in strong electric fields. A strong time-dependent electric field is used in FAIMS as a periodic asymmetric waveform. FAIMS takes advantage of the variations in mobilities at high and low electric fields, a method that filters for the change in mobility of a specific analyte with field strength as opposed to absolute mobility. [12], [15] Also, **differential mobility spectrometry** (DMS), and **differential ion mobility spectrometry** (DIMS) function using the same electronics mechanism as FAIMS, with the only distinction being in the shapes of their electrodes. High selectivity can be achieved with FAIMS/DMS/DIMS devices, which may not be achievable with other low-field-only techniques. Furthermore, FAIMS, DMS, and DIMS are scanned filtering devices that allow only those analytes to match the applied compensation voltage and have a specific response to the changing electric field to pass through the drift region and exit through the aperture. [15]

Differential mobility analyzers (DMA) is used to separate ions in space with different electrical mobilities [12]. DMA devices are characterized by a well-defined perpendicular gas flow, ambient pressure operation, and scanning for the desired molecule [15]. Through a slit, ions enter the DMA and mix with the perpendicular gas flow that is passing between two parallel plate electrodes. Ions are propelled by the combination of electrical force and fluid drift when a perpendicular electric field is superimposed. Only the ions with a particular electrical mobility are released from the DMA through a slit in the outer electrode [12]. High carrier flow rates are used in DMA classification, which allows for the use of space rather than drift time and may result in higher sensitivities and resolving powers [12].

3 Methods

3.1 K-Nearest Neighbors

K-nearest neighbours classifier (KNN) has a describing name of what the classifier does: it classifies samples based on the class of their nearest neighbours, K in front of the name refers to how many nearest neighbours are taken into account determining the class. The KNN is possibly the simplest and most well-known classifier to use [13]. The KNN was first introduced in 1951 when Fix and Hodges presented the k-nearest neighbor rule, a non-parametric approach to pattern classification [21].

Samples are usually divided into training and testing categories when using kNN for classification. During classification, kNN uses the class labels of the training data, making it a supervised classification technique. [59] Techniques for unsupervised classification do not use the training data's class labels. During testing, the classifier predicts the class of each test sample. The test sample's predicted class is determined by taking the most frequent true class out of the k-nearest training samples. In some cases, giving the closest neighbours more consideration when determining the sample's class is more beneficial. in this case, closer neighbours of the sample will have a greater influence than neighbors who are further away.

The distance between a test sample and the training samples is the basis of the kNN classifier. The most popular distance measures are Euclidean and Manhattan distances [13]. Let X_i be a test sample, and X_t be a training sample, both with n features. The Euclidean distance between these samples is defined as:

$$D_E(X_i, X_t) = \sqrt{(X_{i1} - X_{t1})^2 + (X_{i2} - X_{t2})^2 + ... + (X_{in} - X_{tn})^2} \tag{3.1}$$

The Manhattan distance between those same samples can be defined as:

$$D_M(X_i, X_t) = |X_{i1} - X_{t1}| + |X_{i2} - X_{t2}| + ... + |X_{in} - X_{tn}| \tag{3.2}$$

3.2 Random Forest

Random forest (RF) classifier consists of several tree-structured classifiers, each of which casts one vote to determine which class is the most common. The RF is based on bagging where each tree is built independently using a bootstrap sample of the data set and subsequent trees are independent of one another. The benefit of collective tree classifiers is that they outperform single classifiers regarding accuracy and noise resistance. [7] In RF, a further degree of randomness is added to bagging by random feature selection [8]. Breiman, the latter in 2001, introduced both Bagging and RF methods [7], [8].

Random forests modify the construction of the classification trees in addition to employing distinct bootstrap samples of the data set for each tree. Each node in a standard tree is divided using the variable that yields the best split. Each node in a random forest tree is divided using the best of a subset of randomly selected features at that node. [42] As a result, this may weaken each tree independently, but it also reduces the correlation among the trees, which lowers the generalization error. An additional noteworthy feature is that an RF classifier's trees grow without pruning, making it lightweight in terms of computation. [8] Each tree classifier can be expressed as $\{h(x, \Theta_k), k = 1, ...\}$ where x is the input vector and $\{\Theta_k\}$ are the independent and identically distributed random vectors [8]. As a result, some data might be used more than once to train classifiers, while other data might not be used at all. Increased classifier stability can thus be attained, strengthening its resistance to slight variations in input data. [8].

Selecting an appropriate feature selection measure that maximizes class dissimilarity is necessary for tree design. The best split selection is typically determined by an RF using the Gini Index, which assesses an element's impurity concerning the other elements in the class [62]. Choosing a single case at random and claiming that it belongs to a class C_i for a given training set T, the Gini index can be expressed as:

$$\sum \sum_{j \neq i} (f(C_i, T)/|T|)(f(C_j, T)/|T|) \tag{3.3}$$

where $f(C_i, T)/|T|$ is the probability that the selected case belongs to class C_i. Other frequently used approximations for selecting features are gain-ratio and Chi-square [62].

3.3 Gradient Boosting Decision Tree

Gradient Boosting Decision Trees (GBDT) is an ensemble learning method, similar to RF, that builds a powerful predictive model by combining many weak learners. The GBDT consists of several decision trees, which are trained in sequence [23]. The predicted outcome is determined by the sum of the values that each individual tree predicted. The GBDT is based on boosting, an alternative bagging approach. In boosting a series of decision trees are trained iteratively, whereby the success of the previously built trees determines how much weight to give the next learner's training examples [22]. Gradient boosting uses regression on a function of the gradient vector of the loss function that was assessed at the previous iteration to fit the subsequent decision tree. The algorithm was developed in 1999 by Friedman and it is still widely used due to its efficiency, accuracy, and interpretability.[23]

In GBDT a single node is the starting point for the tree-building process, which iteratively adds branches to the tree until a criterion is satisfied. Let $D = (x_i, y_i)(|D| = n, x_i \in \mathbb{R}^d, y_i \in \mathbb{R})$ be our dataset with n examples and d features. Training the model in an additive manner, GBDT greedily adds the best f_t, which corresponds to a decision tree, that, at the t-th iteration, minimizes the following objective function. [40].

$$\tilde{L}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3.4}$$

where g_i and h_i are the first and second order gradient statistics on the loss function and $\Omega(f_t)$ is a regularization term that penalizes the model's complexity and controls over-fitting.

Using an iterative process, a greedy algorithm begins with a single leaf and builds the tree's branches. Branches are added to each leaf after the split in order to maximize the loss reduction [10]. Let's assume the instance sets of the left and right nodes following the split are I_L and I_R. Letting $I = I_L \cup I_R$, we obtain the split's gain as

$$L_{split} = \frac{1}{2}[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}] - \gamma \tag{3.5}$$

Where λ and γ are hyper-parameters. GBDT uses this formula to find the split that maximizes the gain by iterating through all the feature values.

3.3.1 LightGBM

For each feature, traditional implementations of GBDT must scan all data instances to estimate the information gain of all potential split points. The number of features can be efficiently decreased using Ke et al.'s suggested method, LightGBM, without compromising split point determination accuracy [35]. They implemented two novel techniques: Gradient-based One-side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

Ke et al. noticed that larger gradient instances will contribute more to the information gain, per the definition of information gain. Therefore, GOSS randomly samples the instances with small gradients while retaining all the instances with large gradients. When calculating the information gain, GOSS adds a constant multiplier for the data instances with small gradients to account for the influence on the data distribution. This technique leads to greater accuracy in estimating gain compared to uniformly random sampling at the same target sampling rate. [35]

The EFB can be used to reduce the number of features effectively. This method uses the fact that high-dimensional data are usually sparse and in a sparse feature space, many features are mutually exclusive (i.e., they never accept nonzero values at the same time.). EFB bundles these exclusive features into a single feature in a nearly lossless strategy to cut down on the number of features. This algorithm reduces the optimal bundling problem to a graph coloring problem, which is then solved by a greedy algorithm using a constant approximation ratio. This greatly increases the GBDT training speed without sacrificing accuracy. [35]

3.4 Artificial and Convolutional Neural Network

A convolutional neural network (CNN) is a class of artificial neural networks (ANN) that is mainly used to solve challenging pattern recognition tasks. It has developed into one of the deep learning field's most representative neural networks. ANNs are computational processing systems that draw significant inspiration from the functioning of biological nervous systems. The primary component of ANNs is a large number of interconnected computational neurons that collaborate in a distributed manner to learn from the input and optimize their output [56]. The first ANN was created in 1958 when Rosenblatt suggested a single-layer perceptron model by adding learning ability to McCulloch's mathematical model of neurons [50], [65]. The first one and two-dimensional CNNs were proposed in 1987 and 1988 by Waibel and Zhang[71], [77]. The term "convolution" was first used in LeCun et al.'s landmark study establishing CNNs current structure which was published in 1989 [39].

3.4.1 Artificial Neural Networks

The basic components of neural networks are called neurons. Taken from the nervous system, the effects on the synapses are represented by connection weights that adjust the impact of the related input signals, and a transfer function represents the neurons' nonlinear behavior. The weighted sum of the input signals, converted by the transfer function, is then used to calculate the neuron impulse. By modifying the weights corresponding to the selected learning algorithm, an artificial neuron can learn new information. [2] When the unidirectional signal flow from inputs is $x_1, x_2, ..., x_n$, The neurons output signal O can be expressed as:

$$O = f(net) = f(\sum_{j=1}^{n} w_j x_j) \tag{3.6}$$

where w_j is the weight vector and the $f(net)$ is the transfer function. The variable net is a scalar product of the weight and input vectors $net = w^T x$, where T is a matrix's transpose. The output value O is calculated as follows in the most basic scenario, where this neuron is a linear threshold unit:

$$O = \begin{cases} 1 & \text{if } w^T x \geqslant \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

where θ is the threshold level.

The basic structure of ANN consists of a group of neurons at each layer. These layers can be divided into input, one or more hidden, and output layers. Deep learning is the general term for having several of these hidden layers stacked on top of one another. The signal flows strictly in a feed-forward direction from the input to the output layer in feed-forward networks [2]. The input layer typically takes the input as a multidimensional vector and distributes it to the hidden layers. After making decisions based on the previous layer, the hidden layers will assess

whether a stochastic change within itself will result in a worse or better final output in the output layer [56]. The structure of a neural network must be set up so that the desired set of outputs is produced when a set of inputs is applied. For example, the number of hidden neurons influences the network's ability to separate the data. A large number of hidden neurons ensures correct learning, and the network can accurately predict the data it has been trained on. However, its performance on new data, or its capacity to generalize, is weakened. A network that has too few hidden neurons may not be able to understand the connections between the data, and the error will not be able to drop below a reasonable standard. [2] The basic structure of an ANN can be seen in figure 3.1.

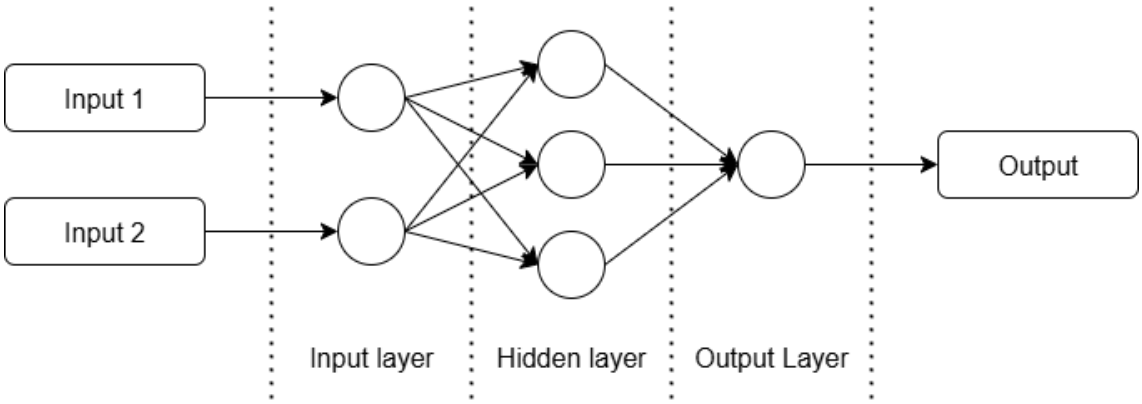


Figure 3.1: Example of simple feedforward artificial neural network.

The learning aspect of ANNs is achieved by adjusting the weights of the connections between neurons, according to some modification rule. Hebb's work from 1949 states that The modification of synaptic connections between neurons was the foundation of learning. Especially, a growth process or metabolic alteration occurs in one or both cells when an axon of cell A is close enough to excite cell B and participates in firing it repeatedly, increasing A's efficiency as one of the cells firing B. [29] The majority of neural network learning methods are essentially variations of the Hebbian learning. The fundamental principle is that two neurons must be more strongly connected if they are both active at the same time. Mathematically, the Hebbian learning rule can be stated as:

$$\Delta w_{ij} = \eta x_i x_j \tag{3.8}$$

where w_{ij} is the weight of the connection from neuron i to neuron j, η is the learning rate, and x_i and x_j are the binary excitation levels of neuron i and j.

A single-layer neural network whose biases and weights could be adjusted to generate the appropriate target vector when presented with the corresponding input vector is called a perceptron [65]. These perceptrons can be trained using the perceptron learning rule. When considering a binary classification task, where x is an input vector from learning samples and $d(k)$ is the desired output, the perceptron learning rule can be stated as follows:

1. Start by assigning the connections random weights.
2. From the collection of training samples, pick an input vector x.

3. If the perceptron provides an incorrect output $y_k \neq d(k)$, adjust all connections w_i in accordance with the formula $\delta w_i = \eta(d_k - y_k)x_i$, where η is the learning rate.
4. Return to step 2.

Although the Hebbian and the perceptron learning rules are similar, the most notable difference is that no connection weights are changed when the network reacts appropriately in the perceptron learning rule.

These previous learning rules work better with tackling simple problems with the correct network topology. However, when more parameters and complex topologies are taken into account, the computational effort required to determine the ideal weight combination significantly increases. The backpropagation algorithm is a popular learning method that can solve such problems. This algorithm was first used with neural networks in 1986 by Hinton et al. [66] The algorithm looks for a combination of the weights that minimizes the error function using the method of gradient descent [64]. The weights are adjusted by calculating the partial derivative of the network's error for every weight. The algorithm takes the derivate's negative and proceeds to add it to the weight which results in the decrease of the error until it reaches a local minima. Taking these partial derivatives and applying them to each weight in feedforward order (input layer $\triangleright$ hidden layer $\triangleright$ output layer) is required because to alter these weights, we must acknowledge the partial derivatives computed in the downstream layer [2].

There are two different modes in which neural networks can be trained that update weight changes differently. In the online method, weight updates are calculated for each input data sample, and the weights are adjusted after each sample. In the batch method, weight updates are calculated for each input data sample, but these values are stored throughout the batch. Batch is usually a small user defined amount of data samples. The weights will be updated only after all contributions have been added at the end of the batch. [2] In essence, training entails passing training samples through a neural network as input vectors or matrices, figuring out the output layer's error, and then modifying the network's weights to reduce the error. When using the batch method and E is the average of all squared errors, the weights are adjusted one by one based on the gradient ∇E for the total training set following:

$$\Delta w_{ij}(n) = -\eta^* \frac{\delta E}{\delta w_{ij}} + \alpha^* \Delta w_{ij}(n-1) \tag{3.9}$$

where η is the learning rate and α is the momentum. Momentum establishes how previous weight changes have affected the movement's current direction in the weight space. Learning rate affects the size of the step that is taken in the weight space. If the selected learning rate is too large, the local minima might be stepped over repeatedly, resulting in slow convergence to the lower error state and oscillations. Too small a learning rate might increase the number of iterations required, resulting in slow performance. It is also possible that backpropagation gets stuck at a local minimum and never reaches the global minimum, which is the optimal solution.

3.4.2 Convolutional Neural Networks

Similar to conventional ANNs, convolutional neural networks are composed of neurons that learn to optimize themselves. CNNs are a type of feedforward neural network that uses convolution structures to extract features from data. Unlike traditional feature extraction methods, CNNs can extract these features automatically [3], [43]. CNNs have numerous benefits over ordinary ANNs, these advantages include local connections, weight sharing, and downsampling. In local connections neurons are only coupled to a small number of neurons in the previous layer, resulting in faster convergence and reduced parameters. In weight sharing sharing weights among connections further decreases parameters. In downsampling a pooling layer uses local correlation to decrease image data while maintaining relevant information and it can reduce the number of parameters by deleting unnecessary features. [41]

The CNNs are mostly used in the field of pattern recognition within images as their architecture is inspired by visual perception [32]. A biological neuron is analogous to an artificial neuron, with CNN kernels, which refer to the learnable parameters used in convolution processes, representing distinct receptors that respond to diverse features. Activation functions replicate brain signals that exceed a particular threshold for transmission to the next neuron. Loss functions and optimizers were developed to train the CNN system to perform as desired.[41]

The overall architecture of CNNs is formed of three types of layers. They consist of convolutional, pooling, and fully connected layers. The convolutional layer calculates the scalar product of neurons' weights and the input volume to determine their output. The pooling layer will then perform downsampling along the input's spatial dimensionality, lowering the number of parameters in that activation. Fully-connected layers mimic normal ANNs, producing class scores from activations for classification purposes. [56]

Convolution

The convolutional layer is designed to learn feature representations of the inputs. Several convolution kernels comprise the convolution layer, which computes several feature maps. In particular, a feature map's neurons are linked to a group of nearby neurons in the previous layer. Using a learned kernel to convolve the input first, followed by an element-wise nonlinear activation function applied to the convolved results, yields a new feature map. The kernel is shared by all input spatial locations to produce each feature map, and several distinct kernels are used to produce the full feature maps. [26] The feature value $z^l_{i,j,k}$ in the k-th feature map of the l-th layer can be calculated by:

$$z^l_{i,j,k} = {w^l_k}^\top x^l_{i,j} + b^l_k \tag{3.10}$$

where w^l_k is the weight vector of the k-th filter of the l-layer, b^l_k is the bias term of the k-th filter of the l-layer, and $x^l_{i,j}$ is the input patch centered at position (i, j) of the l-th layer. The feature map $z^l_{:,:,k}$ is generated by a shared kernel w^l_k because

the weight sharing mechanism can simplify the network and lower the model's complexity.

Activation functions are added to CNNs to detect nonlinear features. They are usually added right after linear operations such as convolution. When $a()$ denotes a nonlinear activation function, The convolutional feature $z^l_{i,j,k}$'s activation value $a^l_{i,j,k}$ can be calculated as follows:

$$a^l_{i,j,k} = a(z^l_{i,j,k}) \tag{3.11}$$

More about activation functions can be found in Chapter 3.4.2.

The convolution operation lowers the output feature map's height and width compared to the input tensor because the kernels' centers cannot overlap with the input tensor's outermost elements thus information is lost in the border. An operation called padding is used to combat this issue. In particular, zero padding adds rows and columns of zeros to the input tensor to center the kernel on the outermost element and maintain the same in-plane dimension during convolution [75]. To adjust the density of convolving, the stride is used. The stride is the distance between two successive kernel positions. Setting a stride of 1 results in a highly overlapping receptive field and large activations. Setting the stride to a higher value reduces overlapping and produces lower spatial dimensions. While kernels are automatically learned during training, the kernel size, amount, padding, and stride hyperparameters must be set before the training process starts.

Pooling

The goal of pooling layers is to reduce the dimensionality of convolutional responses and provide a minor degree of translational invariance to the model. A pooling layer reduces feature maps' in-plane dimensionality to reduce parameter quantity, introducing translation invariance to tiny shifts and distortions [75]. Pooling layers are typically placed between two convolutional layers. A pooling layer's feature maps are linked to the feature maps of the previous convolutional layer. [26]

The most common pooling procedure is max pooling, which takes patches from input feature maps, outputs the maximum value, and discards all other values. A max pooling procedure with a filter size of 2×2 and a stride of 2 reduces the in-plane dimension of feature maps by a factor of two [75]. An another common pooling procedure is average pooling. The average pooling outputs the average value of the input feature map. Visualization of these two pooling procedures can be seen in Figure 3.2. There are many more pooling methods which include L_p pooling and mixed pooling [20], [72].

Fully connected

The last convolution or pooling layer's output feature maps are often flattened into a one-dimensional array of numbers before being connected to one or more

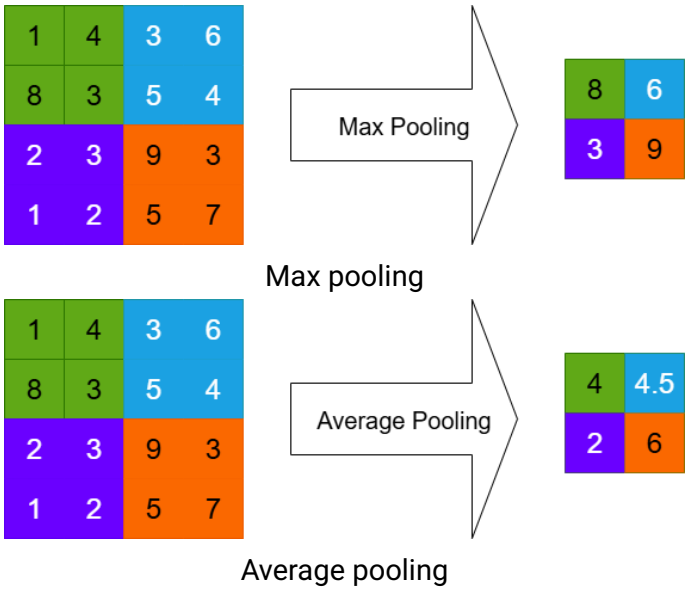


Figure 3.2: Example of average and max pooling operations on the same feature map with 2×2 filters and stride of 2.

fully connected layers. In a fully connected layer, every input is connected to every output by a learnable weight. After convolution and pooling layers extract and downsample features, a selection of fully connected layers translates them to the network's final outputs, including class probabilities in classification tasks. Typically, the output nodes in the final fully connected layer match the number of classes. An activation function follows the output fully connected layer. [26] A softmax activation function is often utilized in multi-class classification problems [67]. Other classification functions could also be used depending on the classification task, such as support vector machines (SVM) [69].

Let θ represent all CNN parameters, including weight vectors and bias terms. To determine the optimal settings for a task, minimize the loss function associated with it [26]. Minimizing the loss function allows us to identify the optimal set of parameters. In this example we have N desired input-output relations $((x^n, y^n); n \in [1, 2, ..., N)$, where x^n is the n-th input data, y^n is its corresponding target label, and o^n is the output of the CNN. The CNN loss can be estimated as follows:

$$L = \frac{1}{N} \sum_{n=1}^{N} l(\theta; y^n, o^n) \tag{3.12}$$

Training CNN is a problem of global optimization. Stochastic gradient descent (SGD) is a popular approach for optimizing CNN networks [75]. The SGD, like ordinary gradient descent described in Chapter 3.4.1, uses gradients to determine which direction to travel in the search space. The difference between them is that, while gradient descent goes through all of he samples in the training set to do a single update for a parameter in a particular iteration, The SGD chooses random samples to calculate the approximate gradient to do a single update for a parameter in a particular iteration [6]. Other popular gradient descent based algorithms include root mean square propagation (RMSProp) and adaptive moment estima-

tion (ADAM) [30], [37].

Activation functions

Convolutional neural networks use many activation functions to represent complicated features. The activation function, similar to the neuron model in the human brain, determines which information is to be transferred to the next neuron. [41] Previously, smooth nonlinear functions like the sigmoid or hyperbolic tangent (tanh) functions were used to mimic biological neuron behavior [75]. However, the rectified linear unit (ReLU) and its variations are now the most commonly used activation functions.

The sigmoid function is a logistic function whose graph has an S-shaped curve. As x approaches 0, the gradient grows steeper. The sigmoid function can map a real number to (0, 1), making it suitable for binary classification challenges [41]. The following formula can define the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.13}$$

The tanh function is also S-shaped, but unlike the sigmoid, it can map a real number to (-1, 1). Tanh's output has a mean value of 0, allowing for a kind of normalization and easier learning in the next layer [41]. The tanh function can be expressed as:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.14}$$

ReLU is a piecewise linear function that cuts the negative part to zero while maintaining the positive part [55]. The ReLU function has a substantial benefit over the sigmoid and tanh functions, as it helps accelerate learning. Sigmoid and tanh employ exponential operations that require division to compute derivatives, but ReLU's derivative is constant. The ReLU activation function is defined as follows:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{3.15}$$

The ReLU unit has a possible drawback: when x is less than 0, the gradient is 0. This means that the back-propagated error is multiplied by zero, and no error is transferred to the preceding layer. In this case, neurons will be considered inactivated or dead. Leaky ReLU (LReLU) may prevent neuron inactivation. When x is smaller than 0, LReLU produces ax instead of zero, where a is a fixed parameter in the range (0,1). [46] The LReLU function may be expressed as:

$$f(x) = \begin{cases} ax & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{3.16}$$

The Exponential Linear Units (ELU) function is an improved version of ReLU. Unlike ReLU, ELU has a negative component that encourages fast learning. The ELU function converges faster than ReLU since its average output value is near zero due to its negative value. [11] The ELU function may be expressed as:

$$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leqslant 0 \\ x & \text{otherwise} \end{cases} \tag{3.17}$$

where α is a preset parameter that regulates how much an ELU will saturate when given negative inputs.

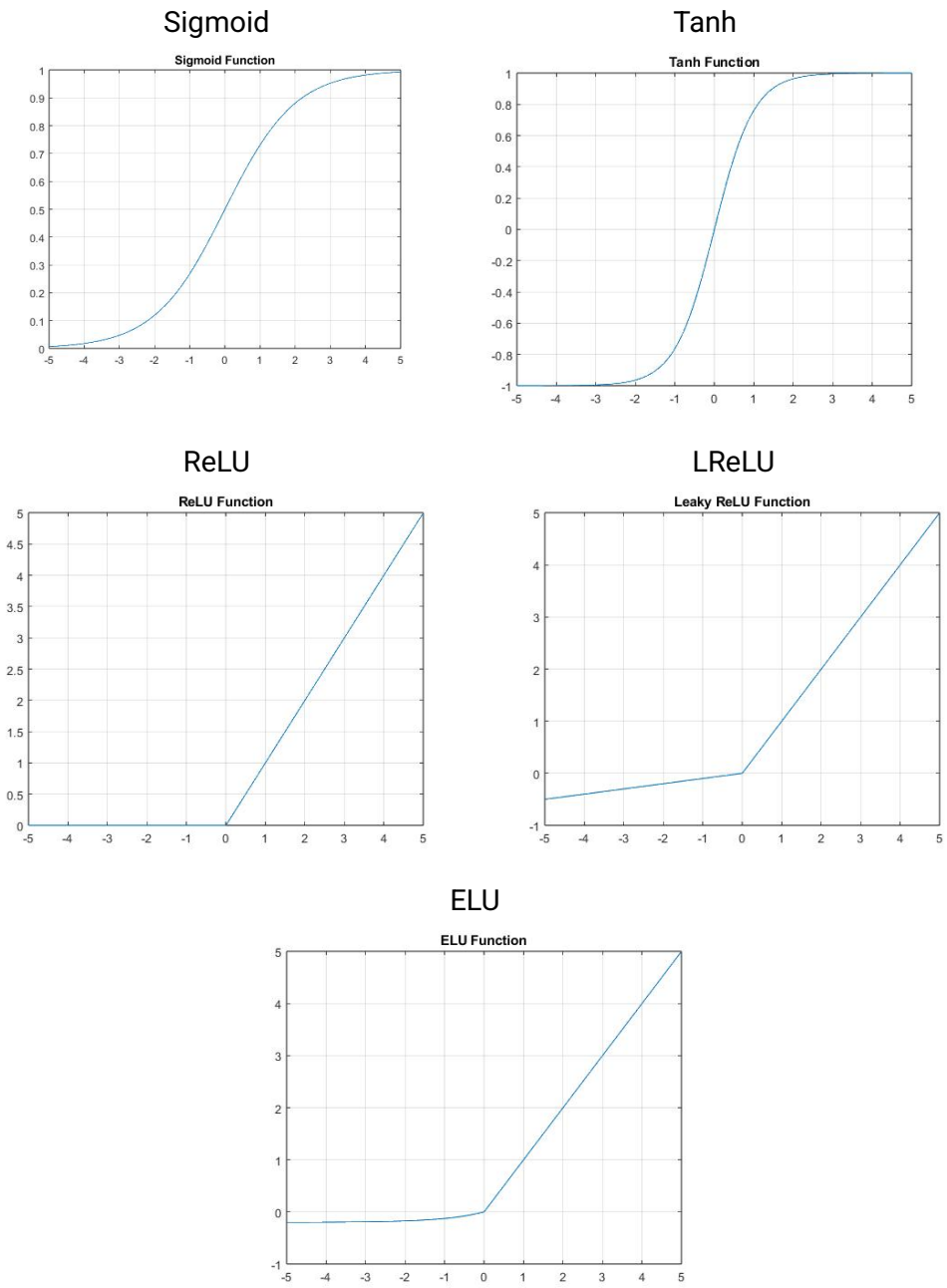


Figure 3.3: Diagrams of Sigmoid, Tanh, ReLU, Leaky ReLU, and ELU activation functions. Leaky ReLU's $a = 0.1$ and ELU's $\alpha = 0.2$.

Loss functions

The loss function measures the difference between expected and actual values. The loss function is often employed as a learning criterion for optimization problems. [41] The objective of convolutional neural networks is to minimize the loss function, which can be utilized for solving classification problems. Selecting the right loss function for a particular problem is crucial. This chapter introduces a couple of common loss functions.

One of the most common loss functions for classification tasks is cross-entropy loss, also known as softmax loss [41]. The cross-entropy loss combines the multinomial logistic loss and softmax activation function. Let the training set be $(x^{(i)}, y^{(i)}); i \in 1, ..., N, y^{(i)} \in 1, ..., K$, where $x^{(i)}$ is the input patch and $y^{(i)}$ is the target class label within K classes. The j-th class prediction for the i-th input is modified using the softmax function $p_j^{(i)}$. Softmax converts predictions to non-negative values and normalizes them to generate a probability distribution across classes. [26] The cross-entropy loss can be expressed as:

$$L_{cross-entropy} = -\frac{1}{N}(\sum_{i=1}^{N}\sum_{j=1}^{N} 1[y^{(i)} = j] log p_j^{(i)}) \tag{3.18}$$

Contrastive loss is a weakly-supervised method for learning a similarity measure between pairs of data examples labeled as matching or not matching. [27]. The contrastive loss is mainly used in Siamese networks, where two or more networks with the same parameters and weights work together for the same problem. It can help reduce dimensionality in convolutional neural networks. After dimensionality reduction, previously related samples maintain their similarity and two samples that were once different remain distinct [27]. if the i-th pair of data is $(x_\alpha^{(i)}, x_\beta^{(i)})$ and $(z_\alpha^{(i,l)}, z_\beta^{(i,l)})$ denotes its corresponding output pair of the l-th ($l \in [1, ..., L]$) layer, the contrastive loss is:

$$L_{contrastive} = \frac{1}{2N}\sum_{i=1}^{N}(y)d^{(i,L)} + (1-y)max(m - d^{(i,L)}, 0) \tag{3.19}$$

where $d^{(i,L)} = \| z_\alpha^{(i,L)} - z_\beta^{(i,L)} \|_2^2$ and m is a margin parameter that affects pairs that do not match. $y = 1$ if $(x_\alpha^{(i)}, x_\beta^{(i)})$ is a matching pair, and otherwise $y = 0$.

A non-symmetric metric for comparing two probability distributions, $p(x)$ and $q(x)$, over the same discrete variable x, is called Kullback-Leibler Divergence (KLD) [38]. The KLD between $q(x)$ and $p(x)$ can be expressed as:

$$L_{KLD} = \sum_x p(x) log \frac{p(x)}{q(x)} \tag{3.20}$$

Optimizers

Convolutional neural networks frequently need optimizing non-convex functions. Mathematical approaches need significant computational power, thus, optimizers are utilized during training to minimize the loss function and get optimal network parameters within a reasonable time frame.[41] In this section, we introduce few gradient descent based optimization algorithms: Momentum, RMSprop, and Adam.

The Momentum algorithm adds a momentum term to SGD, which helps with faster convergence [61]. The algorithm simulates physical momentum by updating weights based on the gradient's exponential weighted average. The Momentum method can prevent one-dimension oscillation, resulting in faster convergence. The momentum term also helps reaching the global minimum instead of the local minimum. The formula for the momentum can be stated as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\delta L}{\delta w_t} \tag{3.21}$$

where m_t is the gradient aggregate at t, β_1 is the momentum, δL is the loss function derivation, and δw_t the weights derivation at t.

The RMSProp algorithm optimizes the learning rate adaptively. It is an extension of the popular Adagrad Algorithm that aims to significantly minimize the computational effort required to train neural networks[16]. The RMSprop updates the parameters using exponentially weighted moving averages of squared gradients [30]. Formula for the RMSprop can be stated as:

$$w_t = w_{t-1} - (\frac{\eta}{\sqrt{v_t}})\frac{\delta L}{\delta w_t} \tag{3.22}$$

where w_t is the gradient aggregate at t, v_t is the moving average of the squared gradients, $v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\frac{\delta L}{\delta w_t})^2$, e, η is the learning rate, and β_2 is moving average parameter.

The Adam algorithm is a combination of the Momentum and the RMSprop algorithm [37]. Adam, unlike RMSprop, adapts learning rates depending on the average second moment of the gradients instead of the first moment. Adam Optimizer takes the strengths of the previous two methods and improves on them to produce a more optimal gradient descent. The formula for the Adam can be denoted as:

$$w_t = w_{t-1} - \eta(\frac{\hat{m}_t}{\sqrt{\hat{v}_t}} + e) \tag{3.23}$$

where $\hat{m}_t$ is bias corrected m_t, $\hat{m}_t = \frac{m_t}{1-\beta_1}$, $\hat{v}_t$ is bias corrected v_t, $\hat{v}_t = \frac{v_t}{1-\beta_2}$, and e is constant.

Regularization

Overfitting is a common problem in CNNs, where the model learns too much from the training set. While overfitting the model inevitably memorizes the data's outliers and unwanted noise, making it perform poorly on unseen data. [75] The opposite of overfitting is called underfitting, where the model is too simple to grasp the intricate details of the data. An effective solution to reduce overfitting is regularization. The most common regularization techniques include dropout and L_2 regularization.

The dropout regularization technique was first introduced by Hinton et al in 2012 [31]. The dropout selects activations during the training and sets them to 0, which reduces the model's sensitivity to particular network weights. Dropout prevents the network from becoming overly reliant on a single neuron or a small group of neurons, ensuring accuracy even in the absence of particular information.

The L_2 regularization is generally referred to as weight decay, although these methods are only identical for SGD [45]. The L_2 regularization works by modifying the objective function by introducing the squared magnitude of the coefficient that penalizes the model's complexity [26].

Other common regularization techniques include batch normalization, where an additional supplemental layer is added to the model, which adaptively normalizes input values for the subsequent layer, and stochastic pooling, which selects activations at random from a multinomial distribution during training„ is parameter-free, and can be used in conjunction with other regularization techniques. [34], [76]

4 Experiment and models

In this chapter, we introduce the localization problem that we intend to undertake. First, we go through the data and its collection before we introduce the experiment. Lastly, we introduce the models, based on the methods of the previous chapter 3, that we used to tackle the localization problem.

4.1 Data Collection and Preprocessing

To achieve place recognition using scents, we had to measure the scents of each location. The measurements were done by using the previously mentioned ChemPro100i handheld IMS device. In total, we made measurements of ten different indoor locations and ten different measurements for each location. Each of the measurements was 5 minutes long. Nine of these locations were inside Tampere University's Hervanta campus and the final location was inside my apartment. For the locations we chose to take the measurements, we wanted a good mix of locations: Some that would be easier to differentiate from each other (i.e., in different school buildings) and some that might be harder to differentiate (i.e., different cafeterias or different lobbies in the same building). A better description of these locations can be found in the Table 4.1. Measurement information for dates and times can be found in the appendix A.

Although we kept the locations the same throughout all of the measurements, we wanted deviations in the measurements of the same location to represent real-life situations better. To achieve these deviations, we made these measurements at varying times (i.e., weekend evenings vs. in the middle of the school day) and at least one hour apart from the previous measurement from the same place. This was done to ensure that in some measurements the locations were vacant of people, and in some the chosen locations were crowded with people so they could walk close to the device. With these deviations in the measurements in mind we wanted to find a machine learning model that could identify the underlying essence of the scent of the location that it could use to differentiate it from the other measurements, no matter how similar the places were.

The ChemPro100i saved these measurements as log files. In the end, 100 measurements were recorded, from each sample (location at a time), using a 1-second logging interval. But from these 300-second measurements, we discarded the first and last 60 seconds because the measurements had a lot of volatility at the start and end of each measurement. This happens because our IMS measurements have a transient phase where the measured currents progressively decrease and increase [54]. To increase available data, each of the remaining 180-second measurements was divided into three 60-second measurements.

Every one of the 1-second interval logs had in total of 81 different entries ranging from time and device serial number to battery voltage and resistance of the MOS sensors [19]. For the simplicity of the experiment, we were only interested in the

Class	Name	Full name and Description
0	F2	F2 lobby, A corridor near stairs located on the second floor in Festia.
1	Home	Home, A living room of a first-floor apartment.
2	K	K lobby, A table at the end of a cafeteria on the second floor in Konetalo.
3	P1	P1 lobby, A table in a large lobby near a student kitchen on the first floor in Päätalo.
4	PB	PB212, A corner of a computer classroom on the second floor in Päätalo.
5	Res	Restaurant, A table just outside of a cafeteria on the third floor in Kampusareena.
6	RH	RH2 lobby, A table in a lobby between lecture halls on the second floor in Rakennustalo.
7	RL	RLa3, A table in a Lobby of the temporary Signal Processing Research Center location on the third floor in Rakennustalo.
8	SM	SM2 lobby, A table in a small lobby near electronics classrooms on the second floor in Sähkötalo.
9	TB	TB205, A corner of a computer classroom on the second floor in Tietotalo.

Table 4.1: Class names, real names, and descriptions for each measurement location.

absolute values of each of the 16 IMS channels and we ended up using information from 14 of these channels, as the last pair is only used to control the speed of the IMS devices airflow [19]. In the end, we were left with 30 measurements of size 60×14 (logs in the measurement $\times$ number of used IMS channels) of each room, totaling of 300 measurements of size 60×14.

The next step of preprocessing is data normalization. Normalization is done to ensure that all features contribute equally to the final prediction of the room of the measurements. To achieve this data normalization equalizes the range of values for each feature with a certain scale [57]. For this work, we chose a Z-score

standardization. Z-score standardization is based on the data's mean value μ and standard deviation σ. In essence, this normalization method expresses features as the number of standard deviations differing from the mean. The formula for Z-score normalization can be expressed as:

$$X_{standardized} = \frac{X - \mu}{\sigma} \tag{4.1}$$

where X represents the data to be normalized. Another popular normalization is called Min-Max scaling, which transforms data features into a specific range, i.e., between 0 and 1. The formula of the Min-Max scaling is:

$$X_{Normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.2}$$

where X represents the data to be normalized and X_{Min} and X_{max} are the minimum and the maximum value of the data.

The final preprocessing step was to split the measurements into training and testing data. Training data teaches the machine learning model how to make predictions or carry out a particular task. Before producing predictions, the model needs to become adept at recognizing patterns within the data. The output of the model for each data point is known because it is typically identified. Testing data is used to evaluate the machine learning model's performance. In the model's output, each data point is unknown. The model's predictive accuracy is evaluated using the testing data.

The testing data were chosen randomly from the measurements so that each room had 6 measurements as testing data and 24 measurements as training data. While selecting the testing data, we made sure that all three of the originally divided measurements were either testing or training data and not mixed between both. This was done to ensure that the used model could not learn from the data from the same measurements used in the evaluation. Considering all locations, this equals 240/60 (80%/20%) measurements split between training and testing data.

4.2 MODELS

For the localization problem, we intend to use the data gathered and preprocessed in the previous section. The models are trained with the training data, aiming to classify each room from the testing data correctly. In total, the models were evaluated using 60 IMS measurements from 10 different rooms.

To give each model a fair comparison against the others, all models were trained and tuned to achieve the best classification results. This was done to ensure the best fit for our dataset for the chosen models. We used trial and error and a randomized search for hyperparameter tuning. A randomized search is a hyperparameter tuning algorithm that analyzes random combinations among hyperparameters within defined distributions [5]. The Algorithm takes random values from

the relevant distribution for each hyperparameter and uses those values to train a model. The ideal hyperparameter values are selected depending on the model's performance after this procedure is carried out a predetermined number of times.

For most of the methods, we used an implementation from the general-purpose machine learning library called Scikit-learn [58]. It effectively implements cutting-edge algorithms that are universally accessible and reusable across application domains and research disciplines.

4.2.1 K-Nearest Neighbors

We used the KNeighborsClassifier from Scikit-learn [58]. Their implementation of the KNN classifier has eight parameters, of which we used six to tune the classifier for our dataset and used the default option for the rest. The most suitable parameters for this task are defined in the following table. As the best value for the amount of neighbours is 1, The K-nearest neighbour classifier denotes to nearest neighbour classifier.

parameter	best option
neighbours	1
weights	uniform
algorithm	BallTree
leaf size	40
metric	Minkowski
p	1

Table 4.2: Best parameters for Scikit-learn KNeighborsClassifier, where neighbours is the number of neighbours used, uniform weights means that all points in each neighborhood are weighted equally, BallTree is the algorithm that is used to compute the nearest neighbours, leaf size is the number of points at which BallTree switch to brute-force, metric is the method used for distance computation, and p is the power parameter for Minkowski metric, where 1 is equivalent of using manhattan distance.

4.2.2 Random Forest

The Scikit-learn's RandomForestClassifiers trees in the forest employ the best split strategy for tree construction [58]. In total, the Scikits RF classifier has 19 parameters, of which we used 9 parameters to tune the RF classifier. The best parameters found for our dataset can be found in the following table:

4.2.3 Gradient Boosting Decision Tree

For the implementation of the gradient boosting classifier, we used one from Scikit-learn called GradientBoostingClassifier [58]. In total, this classifier has 23 parameters, of which we used 10 for tuning and left the rest of the parameters as default

parameter	best option
estimators	320
criterion	Gini
max depth	none
min samples split	2
min samples leaf	1
max features	square root
max leaf nodes	none
bootstrap	true
CCP alpha	0

Table 4.3: Best parameters for Scikit-learn RandomForestClassifier, where estimators are the number of trees in the forest, criterion is the approach to measure the quality of the split, where Gini is the Gini impurity measure, max depth is the maximum depth of a tree, where none means that tree nodes are grown until all leaves are pure or contain fewer than the minimum number of samples split, min samples split is the minimum number of samples required to split an internal node, min samples leaf is the minimum number of samples required to be at a leaf node, max features is the amount of features to consider when looking for the best split, where square root means that the amount of features is the square root of all the features in our dataset, max leaf nodes is the maximum amount of leaf nodes that trees can have, where none means unlimited amount of leaf nodes, and CCP alpha is the complexity parameter for Minimal Cost-Complexity Pruning, where 0 means that the trees are not pruned at all.

options. After the tuning, the best parameters for classification accuracy can be seen in the following table:

LightGBM

The LightGBM is a gradient-boosting framework that utilizes tree-based algorithms for learning, made by Ke et al. [35]. The framework has many parameters, from learning parameters to the device type used for tree learning. From all of the possible parameters, we chose 9 parameters to tune that we thought would have the biggest impact on the classification accuracy and still achieve it in a tolerable time. The chosen parameters and their best values can be seen on the table below:

4.2.4 Convolutional Neural Network

Despite the advances in convolutional neural networks for image classification, existing architectures are not designed to handle scent-based localization problems. Therefore, there is a need for a lightweight yet accurate alternative tailored for IMS data. To address the lack of CNNs made for these kinds of problems, this thesis proposes a novel architecture designed to solve the localization problem

parameter	best option
loss	log loss
learning rate	0.12
estimators	200
criterion	Friedman MSE
min samples split	3
min samples leaf	1
max depth	4
min impurity	0
max features	log_2
CCP alpha	0

Table 4.4: Best parameters for Scikit-learn GradientBoostingClassifier, where loss is the loss function to be optimized, and log loss corresponds to binomial and multinomial deviance, learning rate is the value that decreases the contribution of each tree, estimators is the number of boosting stages to performed, criterion measures to quality of a split, and Friedman MSE is the Friedman's revamped version of mean square error, min split is the number of samples required to split an internal node, min samples leaf is the minimum number of samples required to be at a leaf node, max depth is the maximum depth of the individual regression estimators, min impurity is value, that dictates that a node will split if this split results in an impurity decrease larger than or equal to this value, max features is the amount of features to consider when looking for the best split, and log_2 is the binary logarithm of all the features in our dataset, and CCP alpha is the complexity parameter for Minimal Cost-Complexity Pruning, where 0 means that the trees are not pruned at all.

with good accuracy and with a limited amount of IMS data.

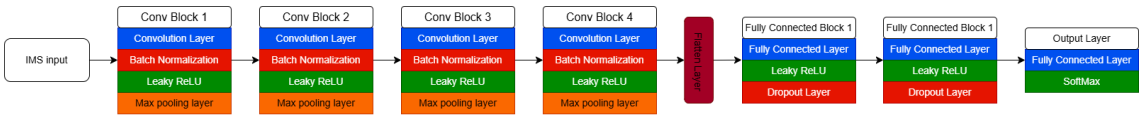


Figure 4.1: An architectural image of our CNN model.

Our model consists of four convolutional blocks and two fully connected blocks. An architectural image of our CNN model can be seen in Figure 4.1, and a more detailed figure is found in appendix B. Each convolutional block has a convolutional layer followed by a batch normalization layer, a leaky ReLU activation layer, and a max pooling layer. Notice that the batch normalization happens before the activation function. Each convolutional layer has twice as many filters as the convolutional layer from the previous convolutional block, starting from 16 filters (16 -> 32 -> 64 -> 124). While the number of filters doubles in each convolutional layer, kernel size, padding, strides, and kernel regularizer stay the same. The parameters for these can be found in the table 4.6 below :

parameter	best option
estimators	250
num leaves	31
min data in leaf	10
boosting type	GBDT
objective	multiclass
sample strategy	GOSS
learning rate	0.12
tree learner	feature
max bins	480

Table 4.5: Parameters used for LGBMClassifier, where estimators is the number of boosting stages to performed, num leaves is the number of leaves in each tree, min data in leaf is the minimum number of observations that must fall into a tree node for it to be added, boosting type is the boosting used, where GBDT is traditional gradient boosting decision tree, objective is set to multiclass for multiclass classification, where the number of classes is set to 10, Data sample strategy is set to gradient-based one-side sampling, learning rate is the step size of an optimization algorithm, tree learner is the distributed learning algorithm used, where feature is the traditional algorithm which aims to parallelize finding the best split in each decision tree, and max bins is the maximum number of bins that feature values will be bucketed in.

Between the convolutional blocks and the fully connected blocks is a flattening layer, which flattens the features from the convolutional blocks for the fully connected blocks. Each fully connected block consists of a fully connected layer, a leaky ReLU activation, and a dropout layer. Both fully connected blocks are similar except that the number of neurons in the latter fully connected layer is a quarter of the first fully connected layer (256 -> 64). The parameters of the fully connected blocks can be seen on the table 4.7 below:

The output layer of our model is a fully connected layer with 10 neurons and a softmax activation function. Our model used categorical cross-entropy as a loss function, which is similar to a cross entropy loss presented in 3.4.2 except that the class labels are one-hot encoded [78]. We used AdamW optimizer, which is an improved version of Adam that separates weight decay from the gradient update step. Instead of incorporating weight decay into the loss function, weight decay is applied directly during the parameter update, resulting in more consistent regularization and greater generalization [45]. We used a decreasing learning rate for our model, which decreased the learning rate by a factor of 0.5 after every 40 epochs. Output layer and other hyperparameters can be found in the table 4.8 below:

parameter	best option			
Convolutional layer				
filters	16	32	64	128
kernel size	(6,2)			
padding	same			
strides	(1,1)			
regularizer	L_2 = $1e^{-5}$			
Batch Normalization layer				
Activation				
LReLU	$\alpha = 0.2$			
Max Pooling layer				
pool size	(2,1)			
strides	(2,1)			
padding	same			

Table 4.6: Layers in each convolutional block and their parameters, where filters is the number of filters used in the convolution, where the first number corresponds to the first convolution block, the second number to the second convolution block, and so on, padding and stride values selected determine that the output of the convolution has the same size as the input of the convolution, The kernels use L_2 regularization with penalty value of $1e^{-5}$, batch normalization layer normalizes its output using the mean and standard deviation of the current batch of inputs, leaky ReLU activation function is used with α value of 0.2, max pooling layer downsamples the input along its spatial dimensions by taking the maximum value over an input window, pool size indicates the size of the pooling window, stride is stride lenght of the pooling window, padding with value of same applies even padding to the input and determine that the output of the pooling has same size as the input.

parameter	best option	
Fully connected layer		
Neurons	256	64
regularizer	L_2 = $1e^{-5}$	
Activation		
LReLU	$\alpha = 0.2$	
Dropout Layer		
dropout rate	0.1	

Table 4.7: Layers of each fully connected block and their parameters, where fully connected layers neurons is the number of neurons, where the first number corresponds to the first fully connected block and the second number corresponds to the second fully connected block, Regulizer is the regularizer function applied to the kernel weights, leaky ReLU activation function is used with α value of 0.2, The dropout rate is the fraction of the input units to drop.

parameter	best option
Output layer	
neurons	10
activation	softmax
Other	
loss function	categorical cross entropy
optimizer	AdamW
starting learning rate	0.001
batch size	8
input shape	(60,14,1)
epochs	150

Table 4.8: Output layer and other hyperparameters of our model.

5 Results

This chapter describes the experimental findings of machine learning classification models designed for the localization challenge. The purpose of this study was to accurately predict the location using IMS readings. Several supervised learning methods were trained and assessed using the dataset described in the previous chapter. The results are presented using accuracy and confusion matrices. This chapter is designed to present the performance of each model and compare its effectiveness in addressing the localization task.

5.1 Results From Each Method

The nearest-neighbor classifier was able to achieve an accuracy of 38.33 % for our testing data. The confusion matrix 5.1 of the results can be seen below. From the confusion matrix we can observe that the KNN classifier predicted all test data from labels 1 and 7 correctly, but from labels 0 and 3 the classifier got none correctly. From the other labels the KNN classifier predicted a minor amount correctly.

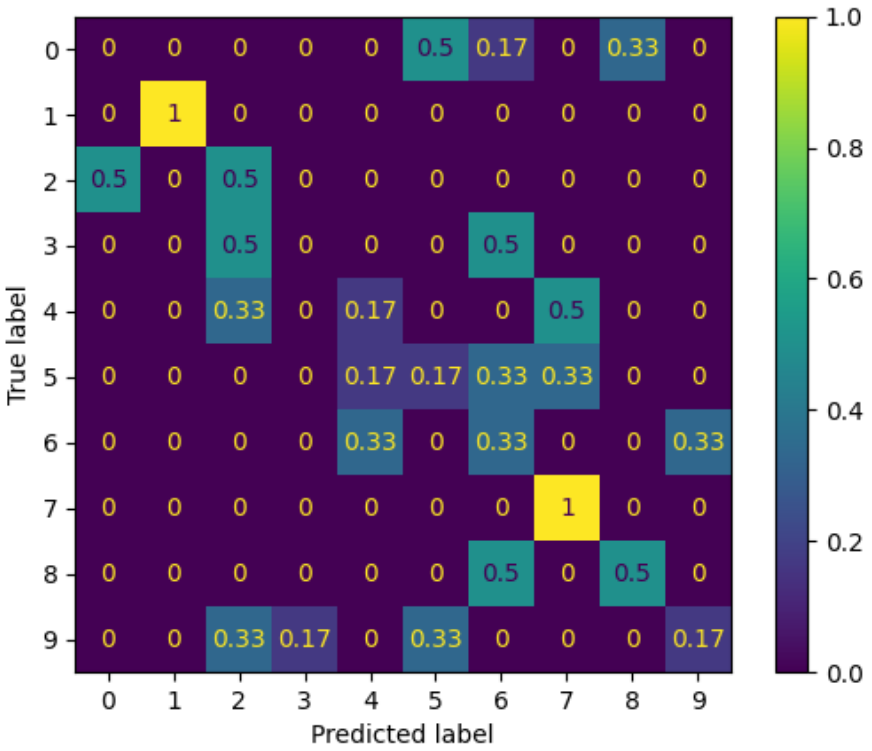


Figure 5.1: Confusion matrix of the results from the K-nearest neighbour classifier with classification accuracy of 38.33 %. Description for the each label can be found in the Table 4.1.

The random forest classifiers results differ each run due to the randomness of the algorithms. This occurs because each tree in the forest is trained on a random subset of the training data, and each split in a tree considers a random subset of features for splitting. For methods with varying results the results are presented as the best result of 10 training runs and the average result of those runs. The best result of the RF classifier is 63.33 % and the average is 60.67 %. The confusion matrix below indicates that the RF classifier classified all the test data correctly from labels 0, 1 and 7 and none correctly from labels 3 and 6. The classifier also got the majority (50 % or over) correctly from the remaining labels.

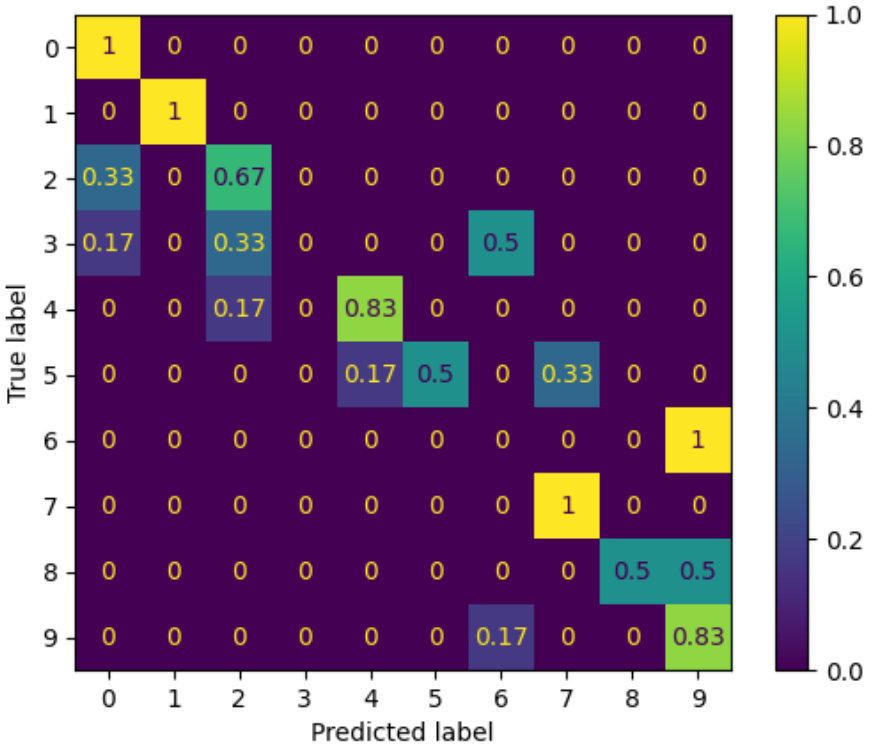


Figure 5.2: Confusion matrix of the best result of the random forest classifier with classification accuracy of 63.33 %. Description for the each label can be found in the Table 4.1.

The gradient boosting decision tree classifier also shows varying results due to randomness and therefore we will be using the best and average results mentioned above. The best result of the GBDT classifier is 53.33 % and the average result is 50.00 %. We notice from the confusion matrix 5.3 that the GBDT classifier got all the test data exact from labels 1, 2 and 7 and none correctly from labels 0, 3 and 6. The classifier predicted majority correct from labels 4 and 9 and minority correct from labels 5 and 8.

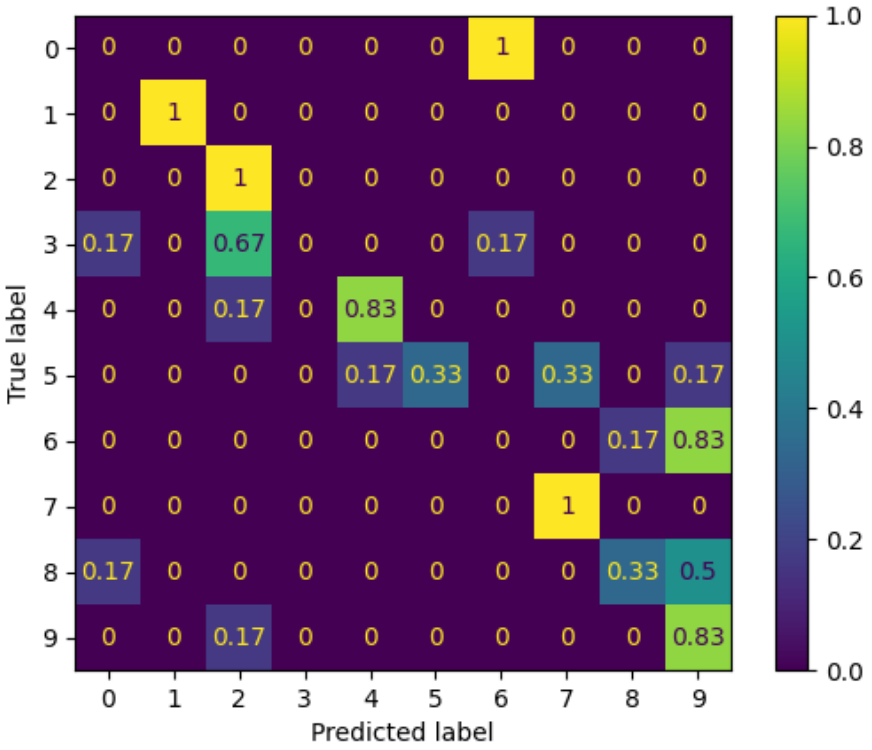


Figure 5.3: Confusion matrix of the best result of the gradient boosting decision tree classifier with classification accuracy of 53.33 %. Description for the each label can be found in the Table 4.1.

Although the LightGBM classifier has more randomness than the GBDT classifier on which it is based, due to its GOSS algorithm, the results are constant over the runs [35]. Our chosen parameters and data provided the results of a best score of 56.67 % and a 10-run average score of 56.67 % with LGBM. The confusion matrix below shows that the LGBM classifier correctly predicted all test data from labels 1, 2 and 7 and none correctly from labels 3 and 6. The LGBM classifier also predicted half of the test data correctly from labels 4, 5 and 8, minority correctly from label 0 and majority correctly from label 9.

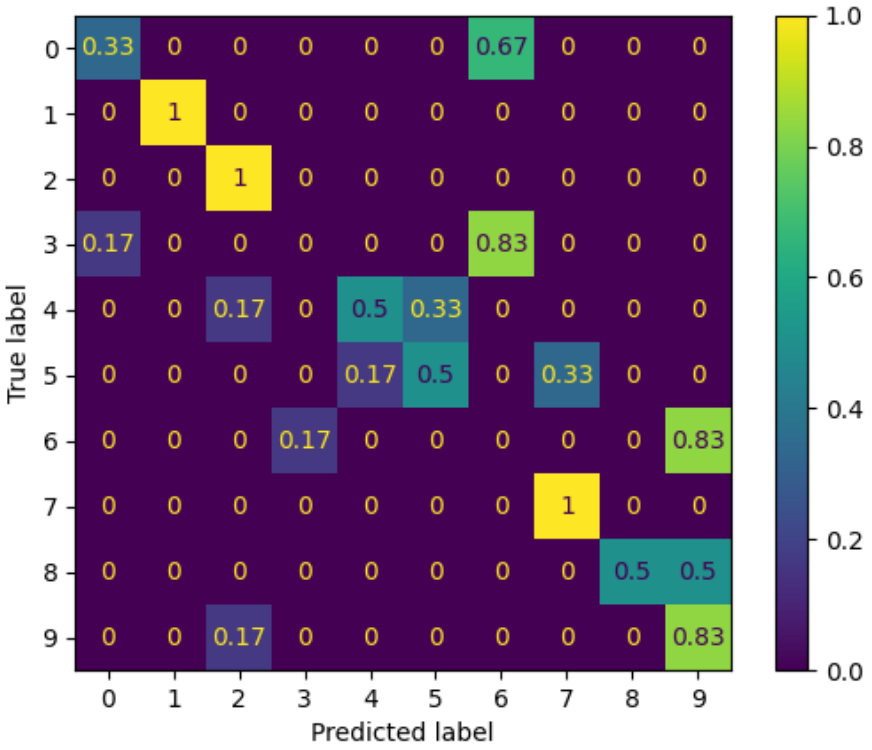


Figure 5.4: Confusion matrix of the best result of the LightGBM classifier with classification accuracy of 56.67 %. Description for the each label can be found in the Table 4.1

There are many sources of randomness for our CNN model, some layers introduce randomness to the model like dropout and batch normalization layers, there are also random initialization of weights in convolution and fully connected layers. Our CNN models best score is 60 % and the average score is 50.83 %. The confusion matrix of the models best result can be seen in figure 5.5. From the matrix we can observe that the model can correctly label all the test data from labels 0, 1, 4 and 7 and that model miss-labeled every test data from labels 3 and 5. From the remaining labels in the test data the model labeled majority correctly from label 2 and labeled correctly half or less from labels 6, 8 and 9. We also provide the training accuracy and loss of our models best run which can be seen in figure 5.6.

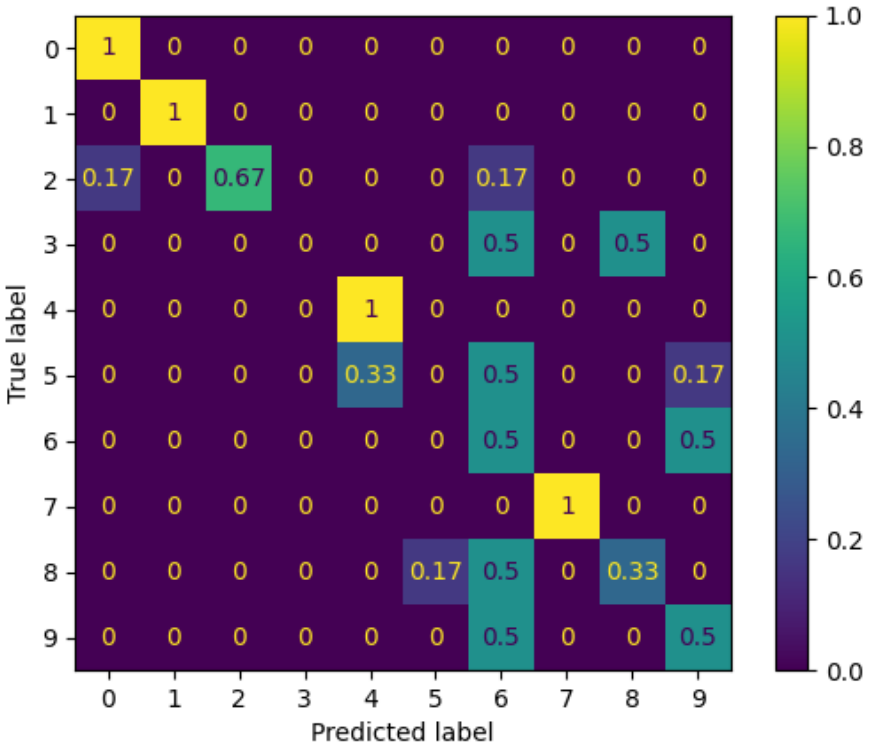


Figure 5.5: Confusion matrix of the best result of our CNN model with classification accuracy of 60.00 %. Description for the each label can be found in the Table 4.1

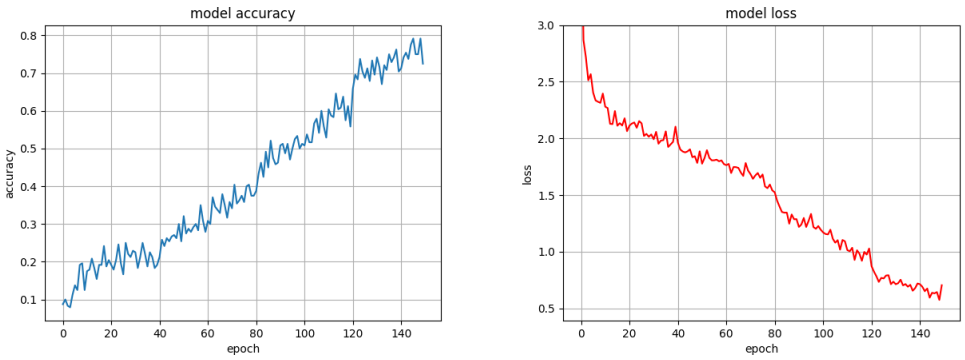


Figure 5.6: The training accuracy and loss of our CNN model's best run.

6 Discussion

This chapter discusses the key findings presented in the previous chapter in relation to the localization problem. This chapter summarizes the results and then interprets what they indicate and what could have caused them. From these findings we formulate additional experiments to test our hypothesis and to bring conclusion to our research problem.

6.1 Discussion of Results

The previous Results chapter indicates that the results are not quite ideal. For the created training and testing data sets, KNN classifier had the worst performance of the tested methods with average and best classification accuracy of 38.33 % and the RF classifier had the best performance with best classification accuracy of 63.33 % and average classification accuracy of 60.67 %. While the remaining classifiers had their average and best classification accuracies between 50-60 %. We had hoped for more accurate results for better future use scenarios.

Method	Best	Average
KNN	38.33 %	38.33 %
RF	63.33 %	60.67 %
GBDT	53.33 %	50.00 %
LightGBM	56.67 %	56.67 %
CNN	60.00 %	50.83 %

Table 6.1: Summary of classification results.

To determine the causes of the results, we must delve further into the results. From the previous chapters confusion matrices we can notice that the test data from labels 1 and 7 were the easiest to correctly label for every method. What differentiates these locations from the rest and makes them more easily detectable? We have brief descriptions of every location in the attachment A, but we go more in detail for the more interesting findings. Label 1, Home, differentiates from the rest of the locations because it is the only location outside of the Tampereen Yliopisto Hervanta campus. The location being also a private apartment, makes it significantly less busy than the other locations, which should account for fewer deviations in the IMS measurements compared to the more busy locations. What the label 1 had in common with label 7, RL, is that the location is also behind a restricted access and less crowded than the other locations which were publicly accessible to all students. Other notable aspect of the location RL, that could separate it from some of the other locations, was that it usually had noticeable coffee scent lingering around the measurement location.

The most challenging locations for the methods to accurately predict were those

associated with labels 3 and 6. Label 3, P1, located near a student kitchen in the Päätalo was usually the most crowded location during the measurements. Measurements were conducted twice during an event held in the student kitchen, during which the area was heavily congested. Of particular concern is the fact that none of the confusion matrices show even a single correct prediction for data from label 3. The location was most commonly mistaken with location 2 and 6. The both locations from labels 2 and 3 are located near kitchen/cafeteria which might be the reason for their mix up. The location from label 6, RH, is a small lobby between lecture halls that often had students waiting or engaging in schoolwork. The nearby corridor also experienced significant foot traffic during peak hours. What locations from labels 3 and 6 had in common is that the locations had a lot of student activity close to the measurement locations. The location 6 was most often mistaken with location 9 although they are quite dissimilar locations. The only notable similarities between locations labeled 6 and 9 are that both are situated on the second floor, albeit in different buildings, and both typically had some degree of student activity.

Some of the measurement sites were purposely chosen to be similar with each other to test the methods classification robustness. It can be observed from the attachment A that locations from labels 3 and 4 are both located in the Päätalo and labels 6 and 7 are both located in the Rakennustalo. The visual presentation of the confusion matrices shows that are no difficulties for our method to differentiate the in building locations from each other which is promising as it shows that IMS device localization can be used in a smaller scale than a building wide localization. Measurement locations from labels 4 and 9 were both similar computer class rooms and the chosen methods are able separate them from each other. Another common factor among the measurement sites is that the locations corresponding to labels 2, 3, and 5 were all situated near cafeterias or kitchens, and each exhibited a noticeable scent of food during lunchtime. According to the confusion matrices, data from label 3 is occasionally confused with label 2, while no other instances of miss classification were observed among labels 2, 3, and 5. It seems that the chosen methods are relatively robust classifying labels from seemingly similar locations and the differences in the ion level are large enough that the methods are able to differentiate the locations from each other.

The proposed CNN model is quite simple in design, but it shows good results compared to the other chosen methods. Despite yielding the second-best classification accuracy among the methods evaluated, the high variability in its performance negatively impacted its average accuracy. The high variability is caused by the randomness and increased with the regularization in the model. The regularization were added to the model to combat some of the overfitting that can be observed from figure 5.6 in the difference of training accuracy and the resulting accuracy. Although increasing regularization or simplifying the model architecture can reduce overfitting, it lead to a decrease in classification accuracy. Conversely, reducing regularization or increasing model complexity may improve training accuracy, but it also makes the model more susceptible to overfitting, ultimately resulting in lower overall performance. The same kind of behavior can be observed when trying to address the classification problem with a complex models, like ResNet50 [28]. These high-capacity architectures also suffer from poor general-

ization when trained on small datasets, as they tend to learn irrelevant patterns and noise present in the training data.

The reasons why classification accuracy can't get much over 60 % despite the classification method used seems to point to deeper data related issues. Usually CNN's need a lot of data to generalize and on small data, simpler models like Random Forest often outperform CNNs—until variance or feature richness exceeds tree capacity [44]. Larger datasets might give better results with some methods and the next chapter experiments with larger datasets using data augmentation. Other issues with the data might include high in-class variance or low inter-class separability which makes it hard for the models to learn clear boundaries. High in-class variance can be explained by high student activity during the IMS measurements and environmental factors (e.g. barometric pressure, temperature, and air currents) that affect the movement of molecules [48]. While the IMS device includes sensors capable of monitoring various environmental factors, their measurements were intentionally excluded from the experiment. This decision was made to prevent the models from potentially learning to identify locations based on these auxiliary sensor readings rather than relying on the IMS-specific measurements (e.g. to recognize a location because its temperature is much higher than the other locations). The high in-class variance can also cause problems if the randomly selected test dataset includes measurements that are completely different from the measurements in the training data set.

6.2 Larger Datasets with Data Augmentation

This section presents a small-scale experiment utilizing an expanded training data set obtained through data augmentation. The aim of the experiment is to ensure that the selected methods have sufficient data to achieve effective generalization. Only the KNN, RF, and CNN methods were included in this experiment. KNN was selected as a baseline due to its simplicity, RF was chosen because it yielded the best results in previous experiment, and CNN was included to assess whether it suffers from poor generalization. The chosen machine learning methods were retuned to accommodate the new dataset with augmented data. The experiment evaluates the performance of the chosen methods in 5-run average classification accuracy and with the best classification accuracy of these runs.

Data augmentation is a technique for artificially expanding a training dataset by performing various alterations on existing data. Its goal is to improve the generalization of the model and prevent overfitting by increasing the variety of data [73]. While numerous data augmentation algorithms exist, this study focuses specifically on methods that are applicable to IMS data. The chosen data augmentation algorithms include noise injection, slicing, region deletion, and magnitude scaling.

The noise injection is data augmentation algorithm that injects random noise noise to the data sample which simulates measurement errors or sensor noise [68]. The algorithm is implemented for this experiment by adding random Gaussian noise to each sample with scale of 0.03. Data slicing is used to cut a piece of each data sample to create a unique new sample [33]. The implemented method of data slicing slices a random 70 % portion of the original sample and then zero

padding it to the original length, creating a new sample. Region deletion is a data augmentation algorithm inspired by dropout regularization that completely erases a part of the original sample [68]. Region deletion is implemented by zeroing out a random length continuous spot (max length of 5 seconds) of a random section of the original samples temporal axis. Magnitude scaling is a technique that applies a variable scaling to the sample [33]. The implementation of magnitude scaling multiplies the entire original sample by a random scalar from a uniform range of 0.85-1.15.

This experiment uses aggressive amounts of data augmentation; each sample in the training data set is augmented with every chosen data augmentation algorithm. The new data set is five times the size as the original training data set as it includes the original data set and the data set from each chosen augmentation algorithms.

6.2.1 Data Augmentation Results

After re-tuning the KNN classifier, turns out that the same classifier described in table 4.2 produced the best results. Although using larger training dataset the classification accuracy worsened; as there are no variance of KNN classifiers results both the best and 5-run average classification accuracy is 33.33 %. This occurs because augmented samples that are not exactly representative of the class can pollute the neighbourhood surrounding a true sample. This may lead to KNN misclassifications.

With the augmented dataset, the optimal hyperparameter settings for the random forest classifier were largely consistent with those presented in Table 4.3, except for the parameters criterion and max features. In this case, the criterion was set to entropy, which utilizes information gain based on entropy from information theory, and max features was set to log_2, meaning the maximum number of features considered at each split is determined by the base-2 logarithm of the total number of features in the dataset [58]. Using the augmented dataset, the RF classifier got best classification accuracy of 51.67 % and 5-run average classification accuracy of 49.67 %. RF classifier also had worse accuracy with augmented dataset than with the regular dataset. This could be because data augmentation introduces noise, redundancy, or subtle label disparities that Random Forest classifiers are not designed to handle. Random Forests, unlike deep learning models, lack the architectural capacity to learn invariant features from augmented data, resulting in worse classification accuracy.

The larger dataset created through data augmentation made it possible to allocate a validation set from the training data when using the CNN classifier. The creation of the validation set happens before data augmentation so as not to distort evaluation. The validation set is defined as 20.5 % (50 measurements) of the original training set, maintaining the same label distribution through random allocation. The new validation set is used in the evaluation, early stopping and in a learning rate scheduler in our model. This scheduler reduces the LR by a factor 0.5 when validation loss stops improving. The scheduler waits 8 epochs without improvements in validation loss before reducing the learning rate. Starting LR was set to be the same as in the baseline experiment. Early stopping halts the train-

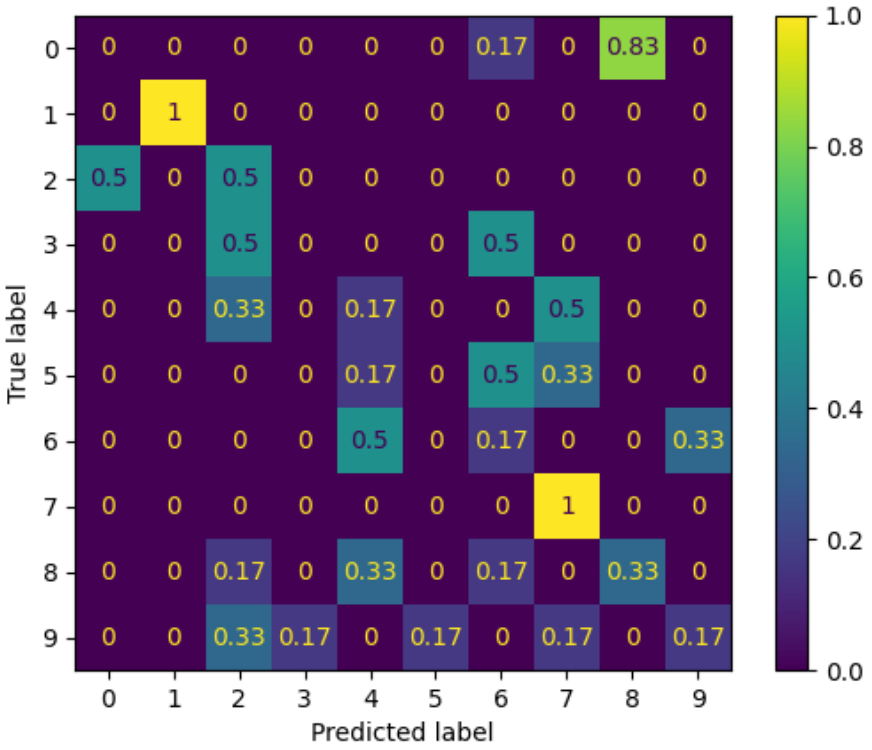


Figure 6.1: Confusion matrix of the results from the K-nearest neighbour classifier during the second experiment with classification accuracy of 33.33 %. Description for the each label can be found in the Table 4.1

ing process once the model's performance stop improving preventing overfitting [60]. In our model, early stopping is configured to occur after 15 epochs without improvement in validation loss. Other parameters that differ from tables in 4.2.4 can be found in table 6.2 below. The 5-run average classification accuracy of the CNN classifier with augmented data set is 53 % and the best classification accuracy is 60 %. The best classification accuracy remained the same compared to the data set without augmentation but the average accuracy did improve.

Figure 6.4 shows that training loss steadily decreases, which is expected, but validation loss is volatile especially in the early stages of training. Validation loss still decreases, which suggests some level generalization, but the volatility hints to instability. Some of the volatility can be explained by the small validation set used because it may cause instability in batch statistics. The accuracy curve shows that both accuracies improve over the training but validation fluctuates more and lacks behind training accuracy. The gap between validation and training accuracy indicates overfitting and that the generalization isn't optimal. Adding more validation data would benefit validation accuracy and generalization but it isn't feasible in this experiment as it would lead to smaller training data set and thus worse classification accuracy of the test data set.

This data augmentation experiment shows that using data augmentation with KNN or RF classifier hinders their classification accuracy. This happens because

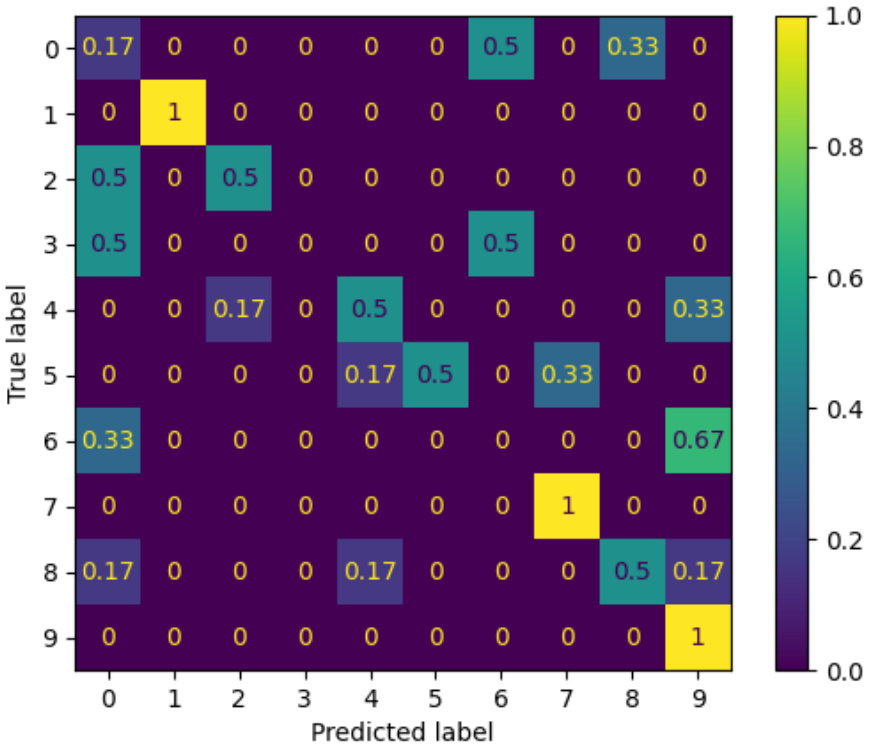


Figure 6.2: Confusion matrix of the best results from the random forest classifier during the second experiment with classification accuracy of 51.67 %. Description for the each label can be found in the Table 4.1

parameter	best option	
Fully connected layer		
Neurons	64	64
Dropout Layer		
dropout rate	0.2	0.3
batch size	16	

Table 6.2: Parameters of our CNN model that were changed for augmented data set. Rest of the parameters are same as in Section 4.2.4

both of these methods rely heavily on training data distributions. The synthetic and noisy samples introduced by data augmentation may reduce KNNs ability to find true nearest neighbors and may cause RF's to split on irrelevant or less informative features. On the other hand data augmentation usually improves classification accuracy of CNN's. Augmentation helps CNNs see many variations of the same class, making them robust to noise and transformations. This, combined with CNNs inherent ability to generalize better when trained on diverse datasets, results in improved performance. In this experiment, the average classification accuracy of our model improved; however, more extensive testing with larger datasets would be necessary to confirm these findings regarding IMS mea-

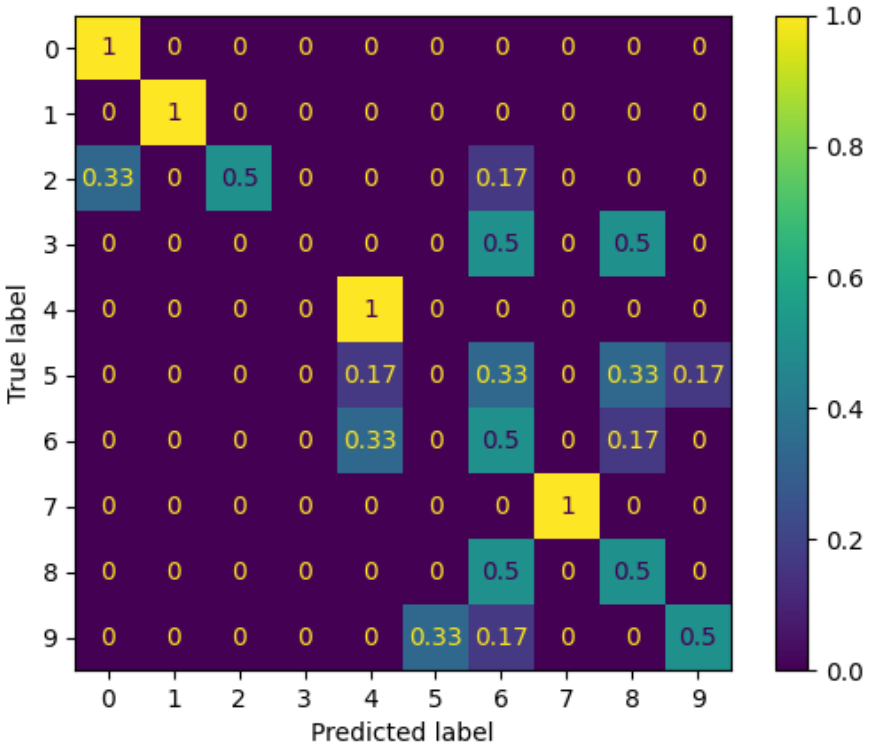


Figure 6.3: Confusion matrix of the best results from the convolutional neural network classifier during the second experiment with classification accuracy of 60.00 %. Description for the each label can be found in the Table 4.1

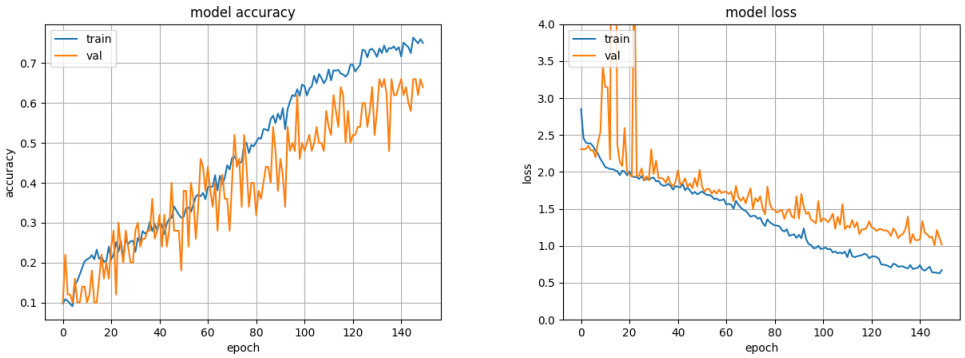


Figure 6.4: The training and validation accuracy and loss of our CNN model's best run during the second experiment.

surements.

6.3 Visual Data Set Analysis

This section focuses on analyzing the IMS measurements in order to determine if the main cause for mediocre classification accuracy is the IMS data itself and not the ML methods employed. The IMS data is visually analyzed by using principal

component analysis (PCA) and t-distributed stochastic neighbour embedding (t-SNE).

Principal Component Analysis is a multivariate statistical technique that reduces the dimensionality of data by combining information from multiple correlated variables into a smaller set of uncorrelated variables known as principal components. These principal components containing only a few linear combinations of the original variables are constructed to capture the maximum possible variance present in the original variables. [1] Principal components are obtained by standardizing the original data and taking singular value decomposition (SVD) of the standardized data [25]. The standardization ensure that the results are not influenced by the scale of the variables. The SVD yields set of positive singular values and two sets of singular vectors, the left and right singular vectors. The first two left singular vectors scaled by the corresponding singular values gives the principal components. [25]

T-distributed stochastic neighbour embedding is a dimensionality reduction algorithm meant for non-linear data, that generates a low-dimensional representation of that data [47]. Noticeable groupings of data points in the low-dimensional representation correspond to observations that are similar in the original high-dimensional space and help to visualize the structure of a data set. The algorithm starts by computing pairwise similarities between data points using a Gaussian probability distribution that represents Euclidean distance of the data points in high-dimensional space. Then it places data points in the low-dimensional space and defines similar probability distribution. Then the t-SNE optimizes mismatch between the data points in the low-dimensional space using he Kullback-Leibler divergence, so that similar points stay close together on the low-dimensional presentation. [47] Visualizing the low-dimensional presentation show that clusters and local neighborhoods of data points in the high dimensional data are visually preserved.

6.3.1 Visual Data Set Analysis Results

For the analysis with previously mentioned method, All of the IMS measurements were firstly standardized using Z-score standardization. The analysis used PCA and t-SNE algorithms employed by Scikit-learn [58]. The PCA was set to produce two principal components for scatterplot presentation. The t-SNE was also set to produce two-dimensional results with perplexity of 30. Both he t-SNE's and PCA's scatterplot present the entire data set. These methods were chosen because they compliment each other: PCA is a linear method whereas t-SNE is a non-linear method and it might reveal non-linear patterns that PCA can't. PCA is effective at preserving large-scale patterns and overall variance whereas t-SNE has focuses strongly on neighbourhood relationships.

Inspecting Figures 6.5 and 6.6 it is noticeable that measurements are usually found in groups of three in both figures. This is because each 5-minute measurement were divided to 3 1-minute usable measurements. It is certain that these 3 measurements from the same 5-minute measurement are more similar to each other compared to the measurements from other 5-minute measurements. It can also be observed from the figures, that variability between the 5-minute measure-

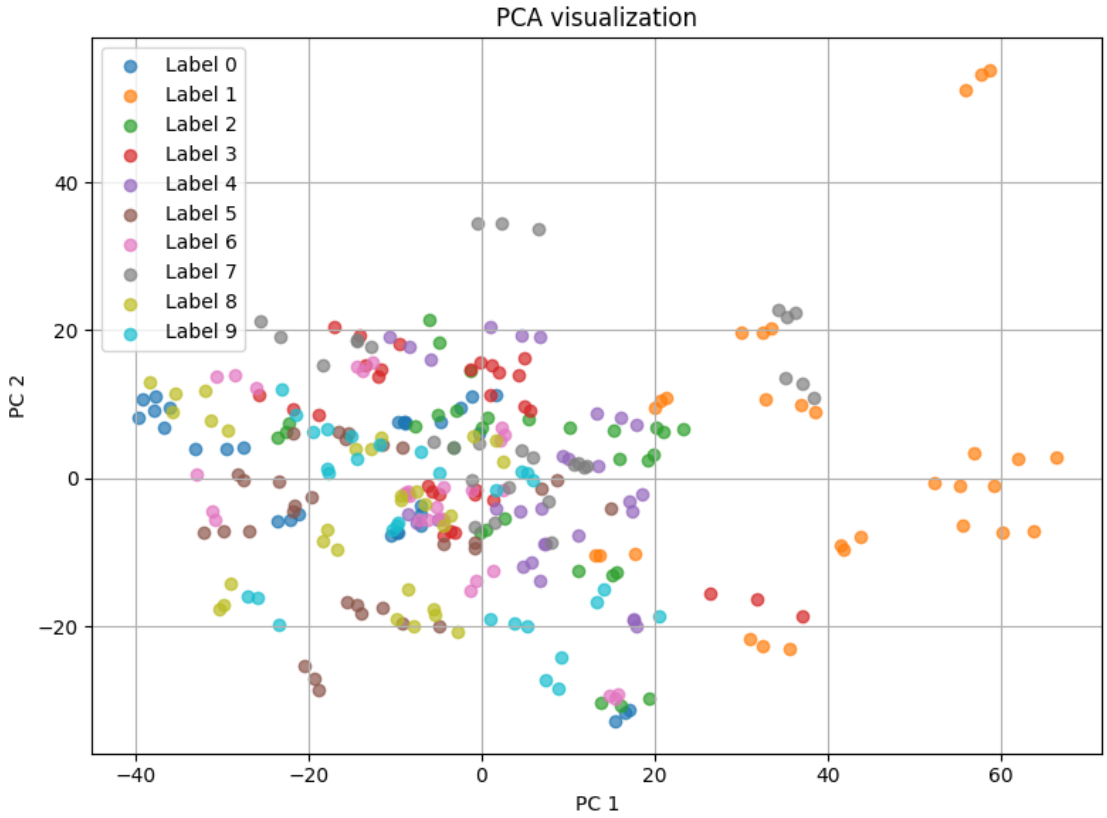


Figure 6.5: All of the IMS measurements visualized with PCA.

ments is high as the measurements don't appear in distinct clusters but scattered over both t-SNE and PCA spaces. Overview of the PCA shows that most labels overlap significantly, meaning that labels also overlap in the original high dimensional space or that PCA isn't able to clearly separate them. As the axes range in PCA is quite high, it indicates that PCA components have large variance differences. This would also indicate that the original measurements has large variance in the direction of the principal components [47]. The labels overlap less with t-SNE but some overlapping is still observable. This is mostly because t-SNE tries to preserve local structure but not the global structure. Overlapping of labels with both PCA and t-SNE figures, combined with mediocre classification accuracies of the classifiers, would indicate that the IMS measurements also have label overlapping.

When examining the individual labels from both t-SNE and PCA figures, some labels form more clearly defined clusters than others. Label 1 forms a tight cluster in t-SNE plot and is also separated from other labels in PCA plot although having high variance in both principal component directions. This could confirm why it was one of the most accurately classified labels for the classification methods. The other easy label to classify was label 7. The PCA a plot has only half of the 5-minute measurements moderately clustered near the origin while the rest aren't as clustered but have similar second principal component. In t-SNE plot label 7 measurement seem to form multiple smaller clusters of couple of measurements. Especially PCA show significant overlapping for label 7 with other labels. Although

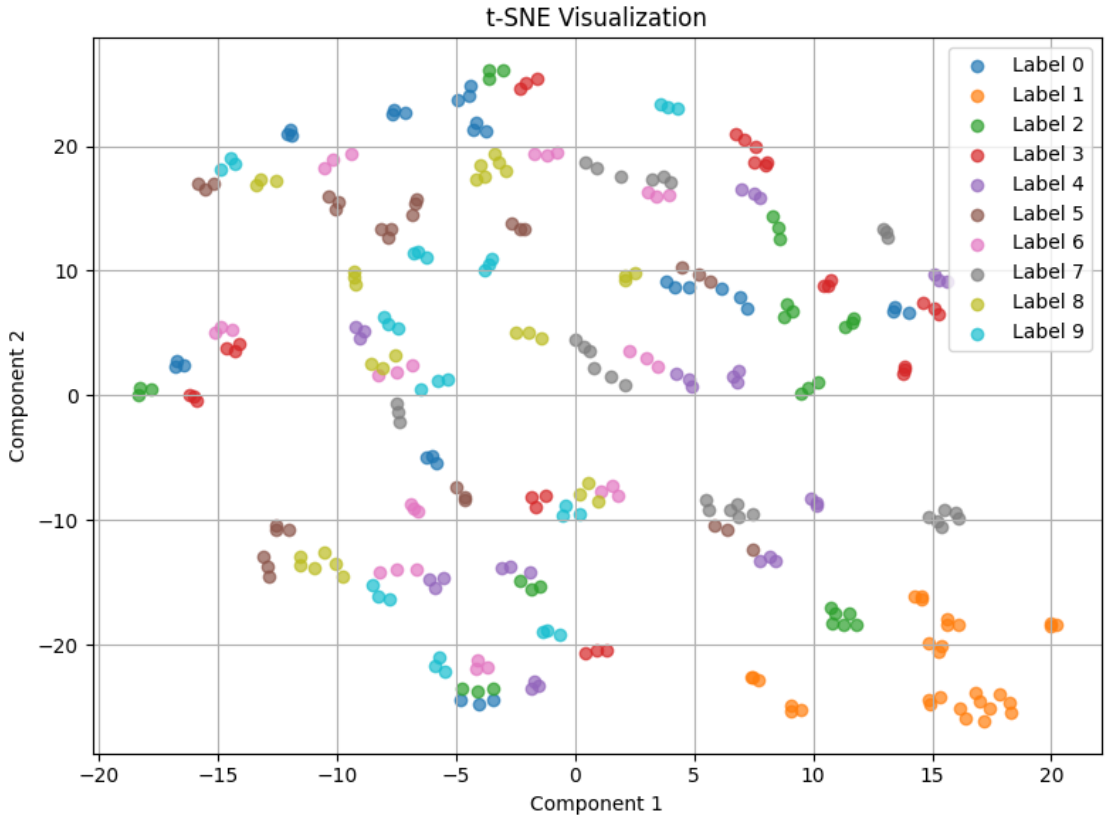


Figure 6.6: All of the IMS measurements visualized with t-SNE.

not forming clear separation to other labels, the classification methods are able to classify measurements from label 7 accurately. This may happen because in PCA space flattens high dimensional data onto 2 dimensional space and t-SNE is a stochastic method and it only tries to preserve local structures [1], [47]. The PCA and t-SNE projections may cause labels to lose separating features from the original data and poor separation from other labels in both projections doesn't necessarily correlate to poor classification accuracy.

The most problematic measurements for the classification methods to classify correctly were from labels 3 and 6. Both t-SNE and PCA plots show serious overlapping with other labels for label 3. The label 3 forms moderately tight cluster and PCA space but not in t-SNE space. From the previous analysis of the confusion matrices, it is known that measurements from label 3 are most frequently mistaken from measurements from labels 2 and 6. It is observable from PCA plot that label 1 measurements overlaps with both 2 and 6 measurements. The different 5-minute measurements of label 3 are scattered across of the t-SNE plot and the scattering is more similar to label 6 than label 2. The measurements from label 6 show similar cluster as label 3 measurements near origin in the PCA plot and appear scattered in the t-SNE plot similar to measurements of label 9. The measurements from label 6 were mostly mistaken for label 9's measurements and some overlapping near origin can be observed in the PCA plot.

7 Conclusion

This thesis focused on machine learning used in ion mobility spectrometry measurement classification. The initial chapters 1-2 gave the thesis statement and the theoretical foundation of IMS and brief summary of the most common IMS methods. The third chapter introduced the theoretical foundation behind the ML methods chosen for the thesis. Chapter 4 focused on the localization problem, introducing the IMS data set created for this experiment. Chapter 4 also presents how the chosen methods were tuned for the best classification accuracy for the created data sets. The following chapter covers the classification results of the chosen methods. The results are presented in average and best accuracy evaluation and confusion matrices of each methods results are introduced for further analysis. Chapter 6 presents an analysis of the confusion matrices and compares the performance of the different methods. In addition, the data augmentation experiment is described, and its results are reported.

In addition to IMS measurement classification, another key objective for this thesis was designing our own CNN model capable of identifying the measurement locations from each other. Due to the amount of IMS data more simple models were observed to perform better. Our model achieved results comparable to the other ML methods chosen. Although best classification accuracy of our model was good the variance of the classification greatly varies between different training cycles, resulting in lower average accuracy. The data augmentation experiment showed that the average classification accuracy can be improved with data augmentation.

Overall, the results of the IMS location classification have not yet reached the desired level of accuracy. The poor accuracy is most likely result of IMS measurements sensitivity to human activity and environmental factors. The high variability between measurements and the overlapping of labels were confirmed in the visual analysis of the IMS measurements. The methods showed good accuracy to locations where external disturbance was minimal and locations with a lot of external disturbance showed poor results. It is likely that strong results could be achieved in controlled laboratory conditions by focusing solely on IMS measurements with minimal disturbances. However, the poorer performance observed in real-world scenarios indicates that further improvements are necessary before the methods can be reliably applied in more advanced or practical applications.

For future work, gathering a larger data set could help the methods to yield better results as seen in CNN's case with augmented data set. Larger data sets would also allow to use more high-capacity model as they suffer from overfitting with smaller data sets. Additionally, use of other than IMS sensors could improve the classification accuracy as IMS measurements are sensitive to environmental factors. Including measurements from additional sensors in the training process could help the machine learning methods identify useful patterns and correlations. However, this approach carries the risk that the models may rely primarily on the additional sensor data, rather than learning from the IMS measurements

themselves. The visual data analysis showed that the IMS measurements were not optimal in classification as the measurements have high variability between measurements. This indicates that alternative sensing technique might be more suitable for scent-based localization.

References

[1] H. Abdi and L. J. Williams, "Principal component analysis", *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[2] A. Abraham, "Artificial neural networks", *Handbook of measuring system design*, 2005.

[3] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition", *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 12, pp. 2037–2041, 2006.

[4] S. Åkesson, J. Boström, M. Liedvogel, and R. Muheim, "Animal navigation", *Animal movement across scales*, vol. 21, pp. 151–178, 2014.

[5] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization", *The journal of machine learning research*, vol. 13, no. 1, pp. 281–305, 2012.

[6] L. Bottou, "Stochastic gradient descent tricks", in *Neural Networks: Tricks of the Trade: Second Edition*, Springer, 2012, pp. 421–436.

[7] L. Breiman, "Bagging predictors", *Machine learning*, vol. 24, pp. 123–140, 1996.

[8] L. Breiman, "Random forests", *Machine learning*, vol. 45, pp. 5–32, 2001.

[9] R. F. Brena, J. P. García-Vázquez, C. E. Galván-Tejada, D. Muñoz-Rodriguez, C. Vargas-Rosales, and J. Fangmeyer Jr, "Evolution of indoor positioning technologies: A survey", *Journal of Sensors*, vol. 2017, no. 1, p. 2 630 413, 2017.

[10] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[11] D.-A. Clevert, "Fast and accurate deep network learning by exponential linear units (elus)", *arXiv preprint arXiv:1511.07289*, 2015.

[12] R. Cumeras, E. Figueras, C. Davis, J. I. Baumbach, and I. Gracia, "Review on ion mobility spectrometry. part 1: Current instrumentation", *Analyst*, vol. 140, no. 5, pp. 1376–1390, 2015.

[13] P. Cunningham and S. J. Delany, "K-nearest neighbour classifiers-a tutorial", *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.

[14] P. Davidson and R. Piché, "A survey of selected indoor positioning methods for smartphones", *IEEE Communications surveys & tutorials*, vol. 19, no. 2, pp. 1347–1370, 2016.

[15] J. N. Dodds and E. S. Baker, "Ion mobility spectrometry: Fundamental concepts, instrumentation, applications, and the road ahead", *Journal of the American Society for Mass Spectrometry*, vol. 30, no. 11, pp. 2185–2195, 2019.

[16] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.", *Journal of machine learning research*, vol. 12, no. 7, 2011.

[17] G. A. Eiceman and Z. Karpas, *Ion mobility spectrometry*. CRC press, 2005.

[18] Environics, *Chempro100i handheld chemical detector operator and unit support manual*, English, Environics Oy, 76 pp.
[19] [. Environics. (), [Online]. Available: https://www.environics.fi/cbrn-products/chempro100i/.
[20] J. B. Estrach, A. Szlam, and Y. LeCun, "Signal recovery from pooling representations", in *International conference on machine learning*, PMLR, 2014, pp. 307–315.
[21] E. Fix and J. L. Hodges, "Discriminatory analysis, nonparametric discrimination", 1951.
[22] Y. Freund, R. E. Schapire, *et al.*, "Experiments with a new boosting algorithm", in *icml*, Citeseer, vol. 96, 1996, pp. 148–156.
[23] J. H. Friedman, "Greedy function approximation: A gradient boosting machine", *Annals of statistics*, pp. 1189–1232, 2001.
[24] V. Gabelica and E. Marklund, "Fundamentals of ion mobility spectrometry", *Current opinion in chemical biology*, vol. 42, pp. 51–59, 2018.
[25] M. Greenacre, P. J. Groenen, T. Hastie, A. I. d'Enza, A. Markos, and E. Tuzhilina, "Principal component analysis", *Nature Reviews Methods Primers*, vol. 2, no. 1, p. 100, 2022.
[26] J. Gu, Z. Wang, J. Kuen, *et al.*, "Recent advances in convolutional neural networks", *Pattern recognition*, vol. 77, pp. 354–377, 2018.
[27] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping", in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, IEEE, vol. 2, 2006, pp. 1735–1742.
[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[29] D. O. Hebb, "The first stage of perception: Growth of the assembly", *The Organization of Behavior*, vol. 4, no. 60, pp. 78–60, 1949.
[30] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent", *Cited on*, vol. 14, no. 8, p. 2, 2012.
[31] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *arXiv preprint arXiv:1207.0580*, 2012.
[32] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex", *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.
[33] G. Iglesias, E. Talavera, Á. González-Prieto, A. Mozo, and S. Gómez-Canaval, "Data augmentation techniques in time series domain: A survey and taxonomy", *Neural Computing and Applications*, vol. 35, no. 14, pp. 10 123–10 145, 2023.
[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in *International conference on machine learning*, pmlr, 2015, pp. 448–456.
[35] G. Ke, Q. Meng, T. Finley, *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree", *Advances in neural information processing systems*, vol. 30, 2017.

[36] S. Kiani, S. Minaei, and M. Ghasemi-Varnamkhasti, "Application of electronic nose systems for assessing quality of medicinal and aromatic plant products: A review", *Journal of Applied Research on Medicinal and Aromatic Plants*, vol. 3, no. 1, pp. 1–9, 2016.

[37] D. P. Kingma, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[38] S. Kullback and R. A. Leibler, "On information and sufficiency", *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[39] Y. LeCun, B. Boser, J. Denker, *et al.*, "Handwritten digit recognition with a back-propagation network", *Advances in neural information processing systems*, vol. 2, 1989.

[40] Q. Li, Z. Wen, and B. He, "Practical federated gradient boosting decision trees", in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 4642–4649.

[41] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects", *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021.

[42] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest", *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[43] T. Lindeberg, "Scale invariant feature transform", 2012.

[44] T. Liu, A. Abd-Elrahman, J. Morton, and V. L. Wilhelm, "Comparing fully convolutional networks, random forest, support vector machine, and patch-based deep convolutional neural networks for object-based wetland mapping using images from small unmanned aircraft system", *GIScience & remote sensing*, vol. 55, no. 2, pp. 243–264, 2018.

[45] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization", *arXiv preprint arXiv:1711.05101*, 2017.

[46] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models", in *Proc. icml*, Atlanta, GA, vol. 30, 2013, p. 3.

[47] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne", *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[48] R. F. Maestre, "Calibration of the mobility scale in ion mobility spectrometry: The use of 2, 4-lutidine as a chemical standard, the two-standard calibration method and the incorrect use of drift tube temperature for calibration", *Analytical Methods*, vol. 9, no. 29, pp. 4288–4292, 2017.

[49] A. C. Marin, A. T. Schaefer, and T. Ackels, "Spatial information from the odour environment in mammalian olfaction", *Cell and Tissue Research*, vol. 383, pp. 473–483, 2021.

[50] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[51] P. Mochalski, J. Rudnicka, A. Agapiou, M. Statheropoulos, A. Amann, and B. Buszewski, "Near real-time vocs analysis using an aspiration ion mobility spectrometer", *Journal of breath research*, vol. 7, no. 2, p. 026 002, 2013.

[52] P. Müller, M. Raitoharju, S. Ali-Löytty, L. Wirola, and R. Piche, "A survey of parametric fingerprint-positioning methods", *Gyroscopy and Navigation*, vol. 7, no. 2, pp. 107–127, 2016.

[53] P. Müller, S. Ali-Löytty, J. Lekkala, and R. Piché, “Indoor localisation using aroma fingerprints: A first sniff”, in *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, IEEE, 2017, pp. 1–5.
[54] P. Müller, K. Salminen, A. Kontunen, *et al.*, “Online scent classification by ion-mobility spectrometry sequences”, *Frontiers in Applied Mathematics and Statistics*, vol. 5, p. 39, 2019.
[55] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines”, in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.
[56] K. O’Shea, “An introduction to convolutional neural networks”, *arXiv preprint arXiv:1511.08458*, 2015.
[57] S. Patro, “Normalization: A preprocessing stage”, *arXiv preprint arXiv:1503.06462*, 2015.
[58] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in python”, *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
[59] L. E. Peterson, “K-nearest neighbor”, *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.
[60] L. Prechelt, “Early stopping-but when?”, in *Neural Networks: Tricks of the trade*, Springer, 2002, pp. 55–69.
[61] N. Qian, “On the momentum term in gradient descent learning algorithms”, *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.
[62] V. F. Rodriguez-Galiano, B. Ghimire, J. Rogan, M. Chica-Olmo, and J. P. Rigol-Sanchez, “An assessment of the effectiveness of a random forest classifier for land-cover classification”, *ISPRS journal of photogrammetry and remote sensing*, vol. 67, pp. 93–104, 2012.
[63] L. Roeder, *Netron: Visualizer for neural network, deep learning, and machine learning models*, https://github.com/lutzroeder/netron, 2017.
[64] R. Rojas and R. Rojas, “The backpropagation algorithm”, *Neural networks: a systematic introduction*, pp. 149–182, 1996.
[65] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.”, *Psychological review*, vol. 65, no. 6, p. 386, 1958.
[66] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
[67] O. Russakovsky, J. Deng, H. Su, *et al.*, “Imagenet large scale visual recognition challenge”, *International journal of computer vision*, vol. 115, pp. 211–252, 2015.
[68] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning”, *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.
[69] Y. Tang, “Deep learning using linear support vector machines”, *arXiv preprint arXiv:1306.0239*, 2013.
[70] M. Utriainen, E. Kärpänoja, and H. Paakkanen, “Combining miniaturized ion mobility spectrometer and metal oxide gas sensor for the fast detection of toxic chemical vapors”, *Sensors and Actuators B: Chemical*, vol. 93, no. 1-3, pp. 17–24, 2003.
[71] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks”, in *Backpropagation*, Psychology Press, 2013, pp. 35–61.

[72] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, “Regularization of neural networks using dropconnect”, in *International conference on machine learning*, PMLR, 2013, pp. 1058–1066.
[73] Z. Wang, P. Wang, K. Liu, *et al.*, “A comprehensive survey on data augmentation”, *arXiv preprint arXiv:2405.09591*, 2024.
[74] A. D. Wilson and M. Baietto, “Applications and advances in electronic-nose technologies”, *sensors*, vol. 9, no. 7, pp. 5099–5148, 2009.
[75] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: An overview and application in radiology”, *Insights into imaging*, vol. 9, pp. 611–629, 2018.
[76] M. D. Zeiler and R. Fergus, “Stochastic pooling for regularization of deep convolutional neural networks”, *arXiv preprint arXiv:1301.3557*, 2013.
[77] W. Zhang, J. Tanida, K. Itoh, and Y. Ichioka, “Shift-invariant pattern recognition neural network and its optical architecture”, in *Proceedings of annual conference of the Japan Society of Applied Physics*, Montreal, CA, vol. 564, 1988.
[78] Z. Zhang and M. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels”, *Advances in neural information processing systems*, vol. 31, 2018.

Appendix A: Measurement descriptions

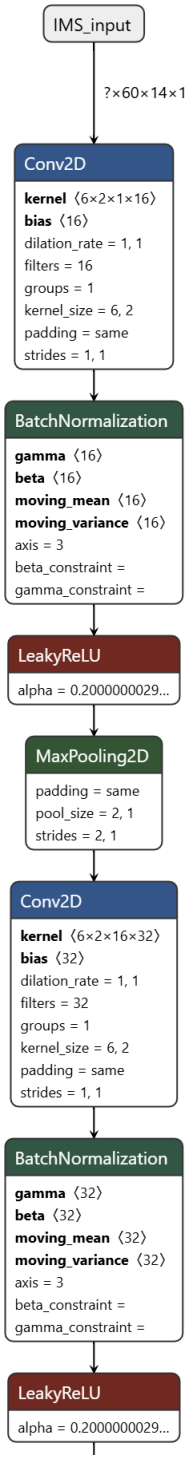
Name	Measurement	Date	Time
F2	1	29.11.2023	14.54
	2	1.12.2023	12.57
	3	4.12.2023	13.31
	4	5.12.2023	13.25
	5	11.12.2023	18.07
	6	22.3.2024	13.30
	7	22.3.2024	15.41
	8	26.3.2024	13.34
	9	26.3.2024	15.02
	10	27.3.2024	13.46
Home	1	28.11.2023	18.23
	2	3.12.2023	12.47
	3	4.12.2023	11.43
	4	5.12.2023	15.03
	5	11.12.2023	19.44
	6	13.2.2024	12.25
	7	22.3.2024	16.17
	8	24.3.2024	16.22
	9	26.3.2024	15.36
	10	27.3.2024	15.30
K	1	29.11.2023	14.41
	2	1.12.2023	12.46
	3	4.12.2023	13.20
	4	5.12.2023	13.15
	5	11.12.2023	17.57
	6	22.3.2024	13.19
	7	22.3.2024	15.51
	8	26.3.2024	13.23
	9	26.3.2024	15.12
	10	27.3.2024	13.36

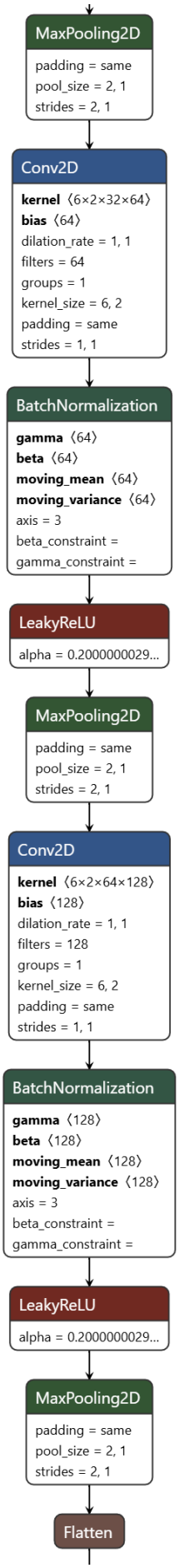
Name	Measurement	Date	Time
P2	1	29.11.2023	15.06
	2	1.12.2023	13.12
	3	4.12.2023	13.42
	4	11.12.2023	18.18
	5	22.3.2023	13.42
	6	22.3.2024	15.30
	7	24.3.2024	13.29
	8	22.3.2024	14.51
	9	26.3.2024	13.45
	10	27.3.2024	13.56
PB	1	29.11.2023	16.25
	2	1.12.2023	13.22
	3	4.12.2023	13.53
	4	5.12.2023	13.38
	5	11.12.2023	18.29
	6	22.3.2024	13.59
	7	22.3.2024	15.20
	8	24.3.2024	14.41
	9	26.3.2024	13.55
	10	27.3.2024	14.06
Res	1	29.11.2023	16.12
	2	1.12.2023	14.26
	3	4.12.2023	14.52
	4	5.12.2023	14.38
	5	11.12.2023	19.17
	6	22.3.2024	14.09
	7	24.3.2024	13.41
	8	24.3.2024	15.02
	9	26.3.2024	14.50
	10	27.3.2024	15.03
RH2	1	29.11.2023	15.19
	2	1.12.2023	13.38
	3	4.12.2023	14.04
	4	5.12.2023	13.49
	5	11.12.2023	18.39
	6	22.3.2024	14.56
	7	24.3.2024	14.19
	8	24.3.2024	15.43
	9	26.3.2024	14.04
	10	27.3.2024	14.16

Name	Measurement	Date	Time
RL	1	28.11.2023	18.14
	2	29.11.2023	15.29
	3	1.12.2023	13.48
	4	4.12.2023	14.14
	5	5.12.2023	14.00
	6	22.3.2024	15.06
	7	24.3.2024	14.30
	8	24.3.2024	15.53
	9	26.3.2024	14.14
	10	27.3.2024	14.26
SM	1	29.11.2023	15.47
	2	1.12.2023	14.02
	3	4.12.2023	14.28
	4	5.12.2023	14.23
	5	11.12.2023	18.52
	6	22.3.2024	14.35
	7	24.3.2024	14.07
	8	24.3.2024	15.29
	9	26.3.2024	14.28
	10	27.3.2024	14.40
TB	1	29.11.2023	16.00
	2	1.12.2023	14.14
	3	4.12.2023	14.41
	4	5.12.2023	14.26
	5	11.12.2023	19.05
	6	22.3.2024	14.22
	7	24.3.2024	13.52
	8	24.3.2024	15.14
	9	26.3.2024	14.40
	10	27.3.2024	14.52

Table A.1: Measurement dates and times for the IMS measurements

APPENDIX B: MODEL ARCHITECTURE




MaxPooling2D
padding = same
pool_size = 2, 1
strides = 2, 1
Conv2D
kernel 〈6×2×32×64〉
bias 〈64〉
dilation_rate = 1, 1
filters = 64
groups = 1
kernel_size = 6, 2
padding = same
strides = 1, 1
BatchNormalization
gamma 〈64〉
beta 〈64〉
moving_mean 〈64〉
moving_variance 〈64〉
axis = 3
beta_constraint =
gamma_constraint =
LeakyReLU
alpha = 0.2000000029...
MaxPooling2D
padding = same
pool_size = 2, 1
strides = 2, 1
Conv2D
kernel 〈6×2×64×128〉
bias 〈128〉
dilation_rate = 1, 1
filters = 128
groups = 1
kernel_size = 6, 2
padding = same
strides = 1, 1
BatchNormalization
gamma 〈128〉
beta 〈128〉
moving_mean 〈128〉
moving_variance 〈128〉
axis = 3
beta_constraint =
gamma_constraint =
LeakyReLU
alpha = 0.2000000029...
MaxPooling2D
padding = same
pool_size = 2, 1
strides = 2, 1
Flatten

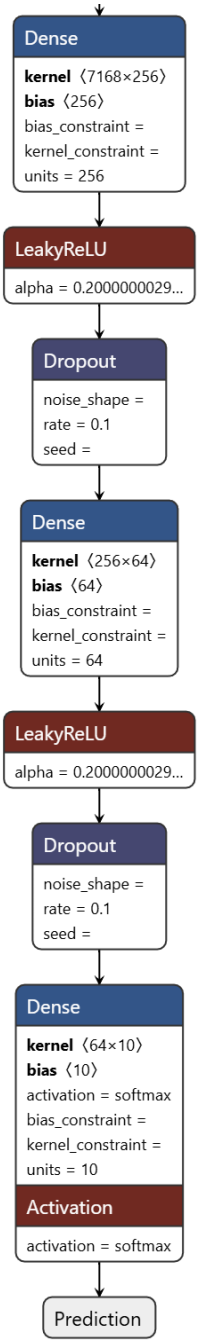


Figure B.1: Architechtural model of the CNN created for the localization problem. The figure was created by using Netron [63].