Tampere University

Viljami Romppanen

DEEP NEURAL NETWORKS FOR JAMMING AND INTERFERENCE CLASSIFICATION IN 5G PHYSICAL LAYER

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Tuomas Virtanen
DSc. Toni Levanen
September 2023

ABSTRACT

Viljami Romppanen: Deep Neural Networks for Jamming and Interference Classification in 5G Physical Layer
Master of Science Thesis
Tampere University
Master´s Degree Programme in Information Technology
September 2023

The fifth generation (5G) of cellular networks is bringing major performance improvements and connecting new industries with wider application areas than the previous generations. The exposed nature of every wireless technology, including 5G, makes them vulnerable to being interfered. Interference can be intentional by jamming attacks or unintentional by other devices in the network. These physical layer threats can cause denial-of-service problems in the network. To ensure the security and availability of 5G communications, it is important to develop identification methods for jamming and interference.

In this thesis, three deep learning (DL) approaches are proposed for classifying different jamming and interference models. To train and evaluate those DL approaches, a 5G spectrogram-related jamming and interference dataset is generated. The proposed DL architectures are convolutional neural network (CNN), long short-term memory (LSTM), and combined CNN-LSTM. The goal is to find out which DL model works the best and if there is a significant difference between two different classification tasks. Those tasks are binary and multi-class classifications. Binary classification denotes whether the input is jammed or not, whereas multi-class classification recognizes the type of the jamming model.

CNN and CNN-LSTM results are almost identical with binary classification accuracies around 97% and multi-class classification accuracies around 90%. LSTM is notably worse with accuracies around 90% and 70% respectively. Overall, all models show sufficient performance, but CNN performs the best in terms of results and efficiency. Also, performance differences between the two classification tasks are not alarming, so either task is suitable for the proposed approaches.

Keywords: Deep learning, Jamming, Interference, Classification, 5G, Physical layer, Neural network

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Viljami Romppanen: Syvät neuroverkot häirintä- ja häiriöluokitusta varten 5G fyysisellä kerroksella
Diplomityö
Tampereen yliopisto
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Syyskuu 2023

Mobiiliverkkojen viides sukupolvi (engl. fifth generation, 5G) tuo mukanaan merkittäviä suorituskyvyn parannuksia ja yhdistää uusia toimialoja laajemmilla sovellusalueilla kuin aiemmat sukupolvet. Kaikki langattomat teknologiat, mukaan lukien 5G, ovat haavoittuvaisia häiriöille. Häiriöt voivat olla tahallisia häirintähyökkäyksiä tai tahattomia verkon muiden laitteiden toimesta. Nämä fyysisen kerroksen uhat voivat aiheuttaa verkossa palvelunesto ongelmia. 5G viestinnän turvallisuuden ja saatavuuden varmistamiseksi on tärkeää kehittää tunnistustoimia häirintää ja häiriöitä vastaan.

Tässä työssä ehdotetaan kolmea syväoppimisen (engl. deep learning, DL) lähestymistapaa erilaisten häirintä- ja häiriömallien luokitteluun. Näiden syväoppimisen lähestymistapojen kouluttamiseksi ja arvioimiseksi luodaan 5G spektrogrammiin liittyvä häirintä- ja häiriötietojoukko. Ehdotetut syväoppimisen arkkitehtuurit ovat konvoluutioneuroverkko (engl. convolutional neural network, CNN), pitkäkestoinen lyhytkestomuisti (engl. long short-term memory, LSTM) ja yhdistetty CNN-LSTM. Tavoitteena on selvittää, mikä syväoppimisen malli toimii parhaiten ja onko kahden eri luokitustehtävän välillä merkittävää eroa. Nämä tehtävät ovat binääri- ja moniluokkainen luokitus. Binääriluokittelu ilmaisee, onko syöte häirinnän alainen vai ei, kun taas moniluokkainen luokitus tunnistaa häirintämallin luokan.

CNN ja CNN-LSTM tulokset ovat lähes identtiset binääriluokituksen tarkkuuksien ollessa noin 97% ja moniluokkaisen luokituksen noin 90%. LSTM on huomattavasti huonompi noin 90% ja 70% vastaavilla tarkkuuksilla. Kaiken kaikkiaan kaikki mallit osoittavat riittävää suorituskykyä, mutta CNN suoriutuu parhaiten tulosten ja tehokkuuden suhteen. Myöskään kahden eri luokitustehtävän suorituserot eivät ole huolestuttavia, joten kumpi tahansa tehtävä sopii ehdotetuille lähestymistavoille.

Avainsanat: Syväoppiminen, Häirintä, Häiriö, Luokittelu, 5G, Fyysinen kerros, Neuroverkko

PREFACE

I am pleased to present this Master's thesis, conducted in collaboration with Nokia Solutions and Networks. The opportunity to work with Nokia has been a remarkable experience, allowing me to gain practical insights into 5G and DL.

First, I would like to thank my supervisors, Toni Levanen from Nokia and Tuomas Virtanen from Tampere University, for their expertise and guidance throughout this research. I would also like to thank all my colleagues at Nokia, whose support has been valuable during this thesis.

I am grateful to my family and friends for their constant support and encouragement, which has been very important in the successful completion of this academic journey.

Tampere, 1st September 2023

Viljami Romppanen

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

5G	Fifth generation
Adam	Adaptive moment estimation
AI	Artificial intelligence
CNN	Convolutional neural network
CP	Cyclic prefix
DL	Deep learning
DoS	Denial-of-service
FN	False negative
FP	False positive
GRU	Gated recurrent unit
JNR	Jamming-to-noise ratio
LSTM	Long short-term memory
ML	Machine learning
NR	New radio
OFDM	Orthogonal frequency-division multiplexing
PRACH	Physical random access channel
PRB	Physical resource block
PUCCH	Physical uplink control channel
PUSCH	Physical uplink shared shannel
QAM	Quadrature amplitude modulation
RE	Resource element
ReLU	Rectified linear unit
RL	Reinforcement learning
RNN	Recurrent neural network
SCS	Subcarrier spacing
SIR	Signal-to-interference ratio
SNR	Signal-to-noise ratio

Tanh	Hyperbolic tangent
TDL	Tapped delay line
TN	True negative
TP	True positive
UCI	Uplink control information
UE	User equipment

1. INTRODUCTION

The fifth generation (5G) of cellular networks is expected to be applied in a broad field of applications such as healthcare, education, entertainment, internet of things, autonomous vehicles, and smart cities [1]. Just as we became more dependent on the previous cellular technologies, we will most likely become even more dependent on 5G. That is why we must ensure its security and availability whenever it is needed. Also, 5G can be utilized for mission-critical communications such as public safety, power grid, and military systems. Unfortunately, 5G, like any other wireless technology, can be easily interfered intentionally by jamming attacks or unintentionally by other devices in the network. Jamming and interference can cause denial-of-service (DoS) problems in the network. Also, jamming attacks can be targeted to very specific signals that are crucial for the operation of the network. [2] The major advancements in machine learning (ML) research have attracted the use of ML in various fields of wireless communications, including 5G. However, ML can also be used for creating even more intelligent security threats, such as smart jamming attacks. [3]

This thesis proposes deep learning (DL) approaches, which is a subset of ML, for classifying different jamming and interference models. For example, this information could be used to develop some anti-jamming strategies or interference management systems. However, introducing different use cases or modeling the above-mentioned smart jamming attacks are not in the scope of this thesis. Also, different unintentional interference scenarios are not introduced in this thesis, but the main focus is to study intentional jamming attacks. The purpose of this thesis is first to generate a dataset with multiple different jamming and interference models in a 5G physical layer uplink waveform. Then the purpose is to implement and compare three different DL models in binary and multi-class classification tasks with the generated dataset. Binary classification denotes whether the input is jammed or not, and multi-class classification recognizes the type of the jamming model. Furthermore, the purpose is to provide proof of concept results for the above-mentioned scheme, which is the first of its kind to the best of found knowledge. The goal is then to find out which of the proposed approaches works the best and if there is a significant difference between the two classification tasks. Then these results can give DL architectural and data generation ideas for future works in similar schemes.

The generated dataset consists of time-frequency power heatmaps. Each heatmap consists of 133 physical resource block (PRB) wise average powers over 14 orthogonal frequency-division multiplexing (OFDM) symbols. Four different jamming models are used that vary their internal parameters for each heatmap. Noise is also considered as a class for scenarios without jamming. To minimize the pre-processing and feature selection steps, DL based approaches are proposed. Only the physical layer power measurements are used, which are relatively easy to compute. The used DL architectures are convolutional neural network (CNN), long short-term memory (LSTM), and combined CNN-LSTM. Each model has separate instances for the two different classification tasks, but the only architectural difference between the two instances of the model is the activation of the output. CNNs are well known for their efficient feature extraction capabilities whereas LSTM is known for its time dependency processing capabilities. The proposed architectures are designed to be relatively efficient for minimizing computational cost and to reduce training times.

The system block diagram is presented in Figure 1.1. Signals from the jamming device and user equipment (UE) go through their own channels and are then mixed in the air interface before arriving at the 5G base station receiver. Then the mixed signal is demodulated in the demodulator and a time-frequency power heatmap is produced in the power measurements block. The produced heatmap is then the input for the ML block. Outputs from the ML block are then binary or multi-class classification results. The dashed line in the UE illustrates the real-world scenario, but the solid line represents the simulated scenario in the dataset. Signals for the UEs are not actually created but the corresponding resources in the power heatmap are set to zero. Only the power measurements from non-allocated resources are then visible for the ML block.

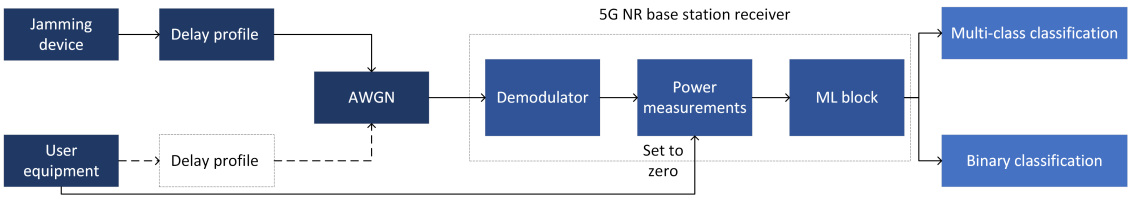


Figure 1.1. *System block diagram*

This thesis is organized as follows. Chapter 2 discusses related works with different types of approaches and application areas, but as mentioned above, this thesis is the first of its kind to the best of found knowledge. Chapter 3 provides the basic background theories and concepts regarding DL, 5G, jamming, and interference. Chapter 4 describes the technical details of creating the dataset and implementing the DL models. Chapter 5 presents and analyses the results for the proposed models. Chapter 6 concludes this thesis and suggests future work with improvement ideas.

2. RELATED WORK

The existing jamming and interference classification methods can be divided into three categories: traditional signal processing [4, 5, 6, 7], ML [8, 9, 10, 11, 12] and DL [13, 14, 15, 16, 17, 18, 19] based approaches. The traditional approaches are mainly focused on binary classification tasks to detect whether the input signal is jammed or not. They are based on the use of some parameters and strategies, for example, power thresholds [4, 5], fuzzy logic [6], game theory [7], etc. To distinguish different jamming types from each other, meaning multi-class classification task, pattern recognition capabilities from the ML and DL based approaches are more robust.

ML based approaches usually include two parts: feature extraction and pattern recognition. The extracted features can be some cross-layer measurements from the protocol stack [8, 9], time-frequency domain analysis [10], radiometric estimates [11], etc. Then the classifications are made by a decision tree [8, 9, 10, 11], support vector machine [8, 9, 10], neural network [8, 11] or any other type of pattern recognition method. The classification performance of these approaches is heavily dependent on the selected features and pre-processing steps, so human factors have a large impact. Also, it is possible to use reinforcement learning (RL) [12], which is one training method for ML algorithms. But due to the nature of RL, it has problems with continuous types of inputs or outputs and hence requires a lot of feature engineering. Although, one general advantage of RL is that it might not need external datasets.

DL based approaches can directly use the raw input signals and then require a minimal amount of pre-processing or feature engineering. That solves the problem of manually selecting and processing suitable features. The input data can be IQ samples [13], power spectrums [14, 15], time-frequency heatmaps [16, 17, 18], etc. Often DL algorithms are based on multi-layer neural network architectures, including CNN [13, 14, 16, 17, 18] or LSTM [18] or both [15]. Usually DL and some ML models require large amounts of data. So, despite the problem of feature engineering being removed, a lot of engineering is required to create those big datasets. However, some approaches are trying to tackle this problem, for example, designing models that could learn from a limited amount of samples [13]. Also, DL algorithms can be used with RL [19, 20]. That might solve the problems with big external datasets, feature engineering, and continuous types of inputs or outputs. However, implementing deep RL models is generally quite complicated and heavy.

The jamming and interference classification in this thesis is focused on binary and multi-class classification tasks with time-frequency power heatmap inputs. Even more precisely, those heatmaps present 5G uplink physical layer waveforms with different resource allocation levels. Because of the nature of those inputs and tasks, DL based approaches are proposed in this thesis. To the best of found knowledge, there is no exactly similar work that meets all of the above-mentioned descriptions. For example, most of the above presented DL based jamming classification methods [13, 14, 15, 16, 17, 18, 19] are related to radar technology or similar setting. And the ML based methods [8, 9, 10, 11, 12] are pre-processing the inputs with some specific feature-engineering logic. The 5G and DL relation can be easily found in other signal classification tasks, for example in modulation classification [21, 22] or cellular technology classification tasks [23, 24]. However, the development steps in physical layer classification tasks with DL are quite similar, despite the tasks or application fields might be different.

This thesis is mostly built around a work of an automatic modulation classification by the authors in [21]. They proposed a hybrid neural network that combined efficient feature extraction from CNN and time sensitivity of LSTM. Also, they compared the individual performances of CNN and LSTM against the combination of these two. As an input, they used IQ samples with 11 different modulation schemes, and they were successfully able to classify different modulation types from each other. A very similar type of approach in modulation classification is proposed by the authors in [22]. The idea of combining both CNN and LSTM is also proposed by the authors in [23], where they implemented a CNN-LSTM network for detecting non-orthogonal multiple access signals. They proposed the input signals to be a stream of 2-D image-like grid, which is a very similar concept as in this thesis. Also, the authors in [15] proposed a CNN-LSTM method for jamming identification in unmanned aerial vehicle networks. They compared the traditional method against the DL model and showed more robust results with DL. As an input they used power spectrums from a narrow band channel and the outputs were mostly related to channel condition classification. The used jamming signals were classified just as good or bad jamming and the jamming models were not as diverse as in this thesis.

3. THEORY

This chapter presents the most relevant background theories for this thesis. Sections 3.1-3.4 focus on deep learning and performance metrics. Sections 3.5-3.7 focus on telecommunications and jamming techniques.

3.1 Deep learning

DL is a subset of ML based on multi-layer neural networks. Also, ML is a further subset of artificial intelligence (AI). [25] The main differences and relations between these concepts are summarized in Figure 3.1.

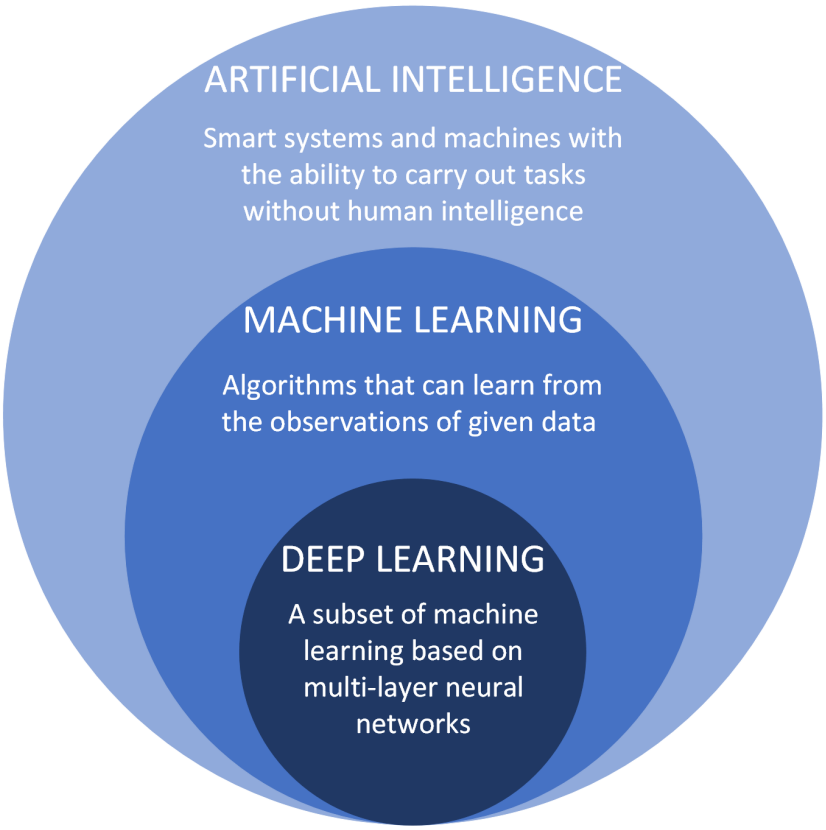


Figure 3.1. *Main differences and relations between AI, ML and DL. Adapted from [26].*

In general, the definition of AI is very broad. Every machine or system, that can carry out tasks without human intelligence, can be considered to have AI. Therefore, depending on

the definition of AI, it can be seen to exist even before computers. For example, the earliest automated machines were invented in the 17th century, whereas the first operational computer was built in 1940. However, the birth of AI as an official research field was in 1956 at a workshop at Dartmouth College. [27]. The recent availability of large amounts of data and increased computational power have accelerated the development of AI, especially in the ML and DL domains. These factors have then allowed the development of complex models and revolutionized the solutions of AI. [25]

ML is a subset of AI that focuses on developing algorithms and statistical models that enable systems to learn from data without being explicitly programmed to do so. Learning means that ML algorithms can improve their performance on the specific task. That is achieved by gathering experience from the observations of given data. There are various learning, also known as training, approaches for ML algorithms including supervised learning, unsupervised learning, and reinforcement learning. How well the ML model eventually learns depends on many things and details, such as the used data, the used ML model and the task to be solved. [25]

As mentioned above, DL is a subset of ML based on multi-layer neural networks. Those multi-layer neural networks are also called deep neural networks. The deep multi-layer structure enables models to process and analyze more complex data such as images, audio, and text. That allows end-to-end learning, which means that difficult problems are solved more straightforwardly without extra stages. For example, traditional ML and AI approaches might need to split the problem into smaller sub-problems, like feature extraction and classification. DL approaches can automatically solve both of these problems. This ability has led to significant improvements in various fields, including computer vision, natural language, and speech processing. Usually neural network is called deep when it has more than two or three layers and often it is trained with large amounts of data. [25] Different types of layers and neural networks are presented more closely in Section 3.2.

3.2 Neural networks

Artificial neural networks, in this context simply called neural networks, are computing systems loosely modeling biological neural networks in brains. Neural networks are constructed with basic units usually called neurons, also known as nodes or perceptrons. [25] Basic structure of a neuron is presented in Figure 3.2, where the key components between inputs and outputs are shown.

Neurons are connected to each other so that the output of a neuron is an input to another neuron. Between those connections, learnable weights are applied. Learning is achieved by optimizing the values of those weights. Weighted inputs are summed together and passed to an activation function in the neuron. The activation function usually does a non-linear mapping between the inputs and outputs. Neuron output y can be defined as

$$y = f(w_0 + \sum_{i=1}^{n} w_i x_i), \tag{3.1}$$

where f is the activation function, w_0 is the weight for a bias, w_i is the i:th weight and x_i is the i:th input. [28] Different types of activation functions are presented more closely in Section 3.3.1. The bias term is usually an optional input value of $+1$. Adding the bias term might help to get better results by shifting the activation function. [29]

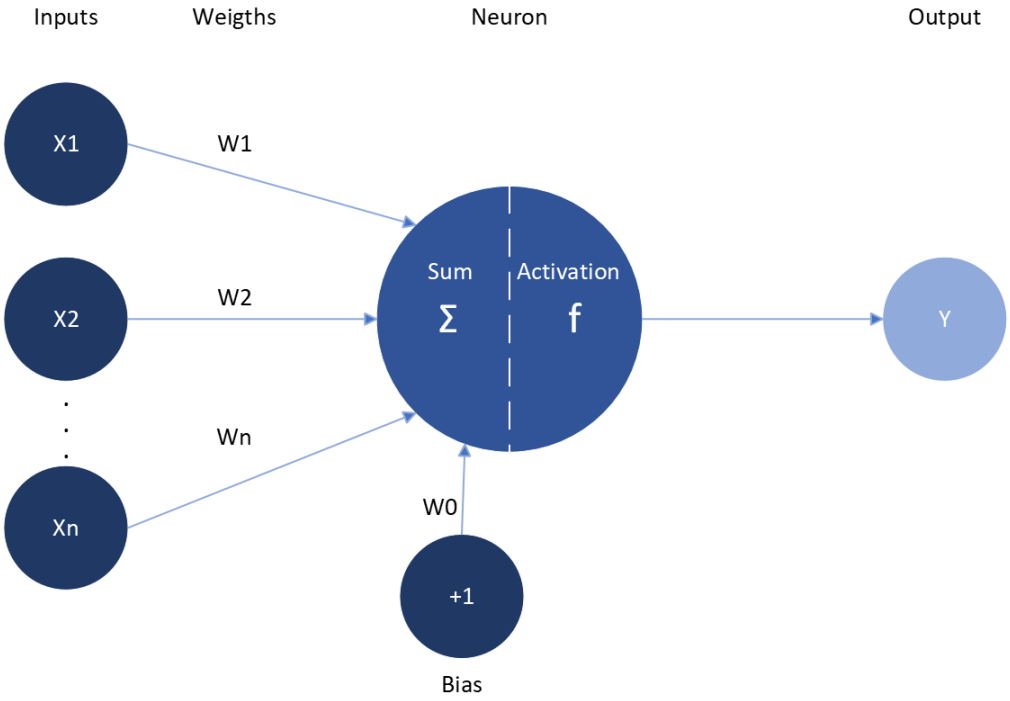


Figure 3.2. *A typical neuron with bias and weighted inputs. Adapted from [28].*

Usually, neurons are aggregated into layers to form the whole neural network. Different layers may perform different types of operations depending on the type of neural network. In the following subsections, three of the most relevant neural networks of this thesis are presented.

3.2.1 Fully connected neural networks

As mentioned above, usually neurons are aggregated into layers to form the whole neural network. Those layers are called an input layer, a hidden layer, and an output layer. In general, the name of the layer describes the position or main functionality of the layer. Input data is presented in the input layer, processed by the hidden layers, and finally, outputs are formed in the output layer. [30] In a fully connected neural network all of the layers are fully connected to each other. That means that every neuron in the past layer is an input to every neuron in the following layer. Then any input value can affect any output

value. [31] An example of the fully connected neural network is presented in Figure 3.3, where the bias term is not added to every neuron for simplicity reasons.

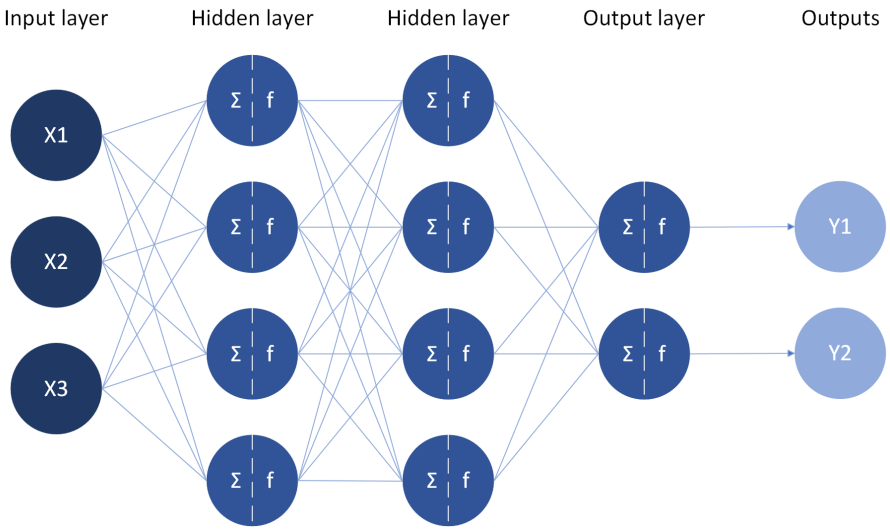


Figure 3.3. *Four-layered fully connected neural network. Adapted from [30].*

Fully connected neural networks usually have many learnable parameters even with a relatively small amount of inputs. That tends to make the fully connected structure computationally heavy. However, one property of those full connections is that they do not assume a structure or order in the inputs. Therefore, fully connected neural networks are often used in the last layers of deep neural network architectures to mix the inputs and make final classifications. [31]

3.2.2 Convolutional neural networks

CNNs are neural networks that are specialized for processing data that has a known grid-like structure. [32] Often CNNs are suitable in problems where the input order and location matters. That is because CNNs consider that the points in the same region are closely related and that the identified patterns can be found translated in the space. CNNs have many successful applications. Especially in image processing problems where the data can be thought of as a 2-D or 3-D grid of pixels. Also, CNNs can be suitable for analyzing time-series data. Often the time-series is then considered as an image by in example time-frequency transformations. [33]

A basic CNN architecture consists of a convolutional layer, a pooling layer, and a fully connected layer as presented in Figure 3.4. It can be seen that successively applying convolutional layers and pooling layers can be called a feature extraction part. And in the end, fully connected layers can be called as a classification part. [33] The flattening operation between the feature extraction and classification flattens the data to 1-D for the classification part, which does not assume a structure or order in the inputs.

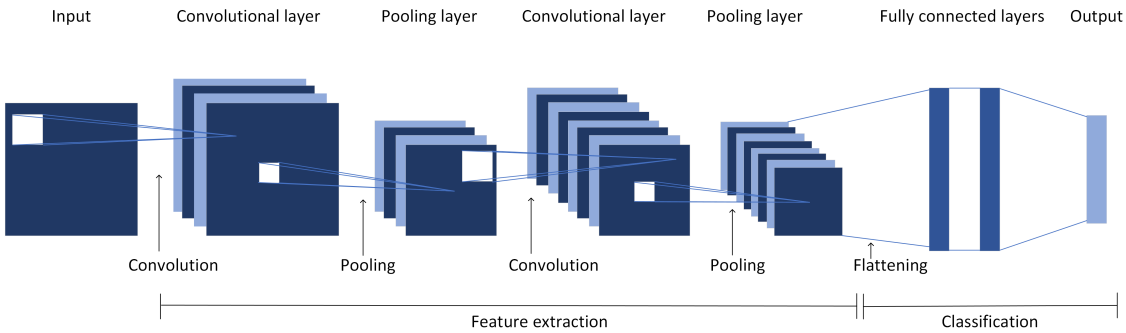


Figure 3.4. *A basic CNN strucure. Adapted from [33].*

Convolutional layers apply convolutions by sliding a kernel over the input grid. A kernel is a matrix containing weight values. Each input channel, meaning the depth dimension, gets a different kernel. A filter is a set of those unique kernels along the input channels. At each step, the weights in a kernel are element-wise multiplied with the input and summed together, until the whole input region is covered. Intermediate results from different kernels in a filter are element-wise summed, bias term is added and passed to an activation function. A feature map or output is then formed from a filter. That map presents the strength of a certain feature in the input regions. Usually, each convolutional layer has multiple different filters to present different features of the input. [33] A basic convolutional operation between an input and a kernel is presented in Figure 3.5.

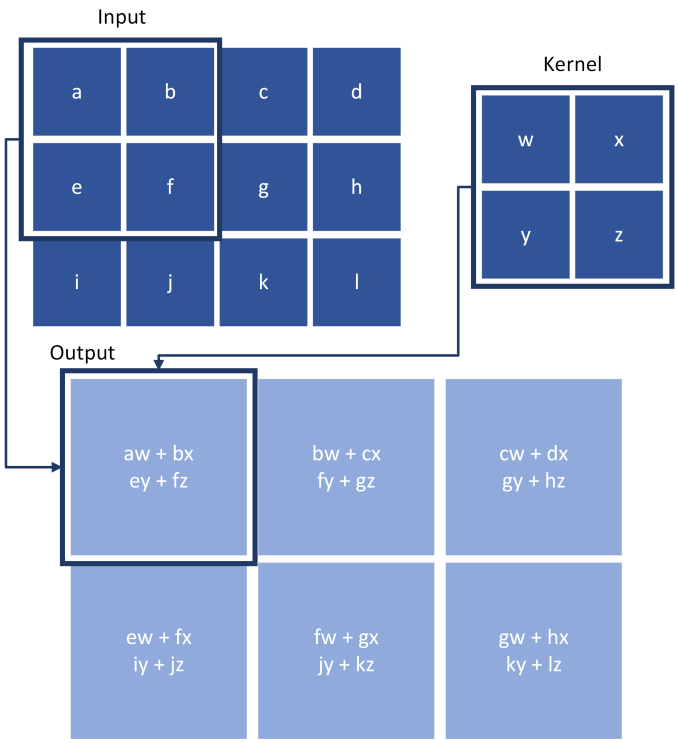


Figure 3.5. *A basic convolutional operation in a CNN. Kernel size is 2x2 and stride is 1. Multiple input channels would be processed similarly with different kernels and then the outputs would be element-wise summed. Adapted from [32].*

The pooling layer is often used in a CNN for sub-sampling the feature maps. It helps to reduce the computational load and number of parameters in the network. Also, pooling has a significant impact on identifying spatial patterns. [33] One of the most used pooling operations is max pooling, which selects the maximum value over the input region. Other popular pooling operations are average pooling, L^2 norm pooling, and weighted average pooling based on the distance from the center. [32] An example of the max pooling is presented in Figure 3.6.

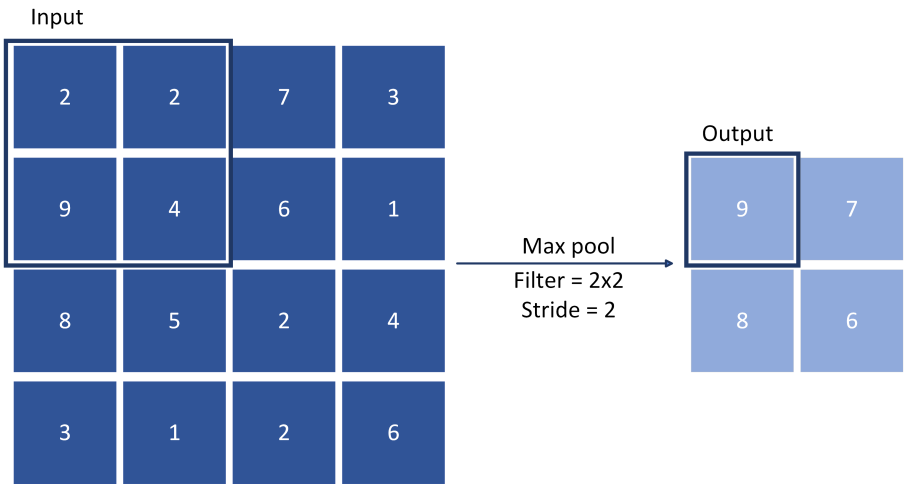


Figure 3.6. *Max pooling. Adapted from [34].*

3.2.3 Recurrent neural networks

Recurrent neural networks (RNNs) are neural networks that are designed to process temporal or sequential data with time dependencies. Usually, the input data is configured to be a 2-D tensor containing the time steps and corresponding input features. RNNs can utilize the time history and context of the data by processing the input time step by time step. The output of the current time step is passed to the next time step as a state. With that ability to carry information between the time steps, RNNs are often used in language processing, computer vision, and forecasting problems. [33] A general illustration of the RNN unit is presented in Figure 3.7, where it can be seen that possible inputs and outputs are dependent on the used RNN unit type.

RNN is usually constructed with a combination of multiple RNN units with stacked architecture, as presented in Figure 3.8. The most popular RNN units are Elman RNN, LSTM, and gated recurrent unit (GRU). Operations inside of those RNN units are relatively complex including multiple activations, additions, and multiplications for each input stream. The most complex structure is with LSTM, the simplest with Elman RNN and GRU in between. RNNs tend to suffer from exploding or vanishing gradient problems. However, the complexity of a RNN unit is usually proportional to the ability to tackle those problems. On the other hand, a more complex unit also means a computationally heavier structure. [35]

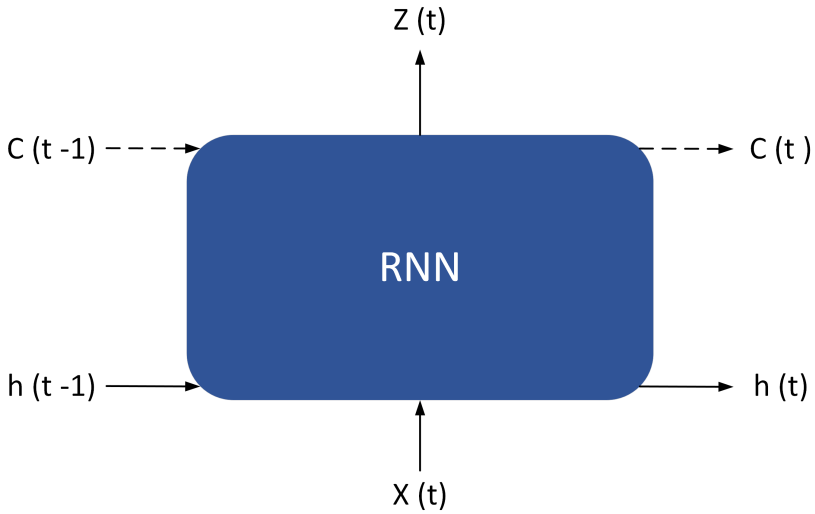


Figure 3.7. *A general recurrent unit's inputs and outputs, where Z(t) is the output, X(t) is the input and h(t-1) or h(t) is the hidden state. The dashed lines in C(t-1) and C(t) illustrate the cell states that are present in LSTM. Adapted from [35].*

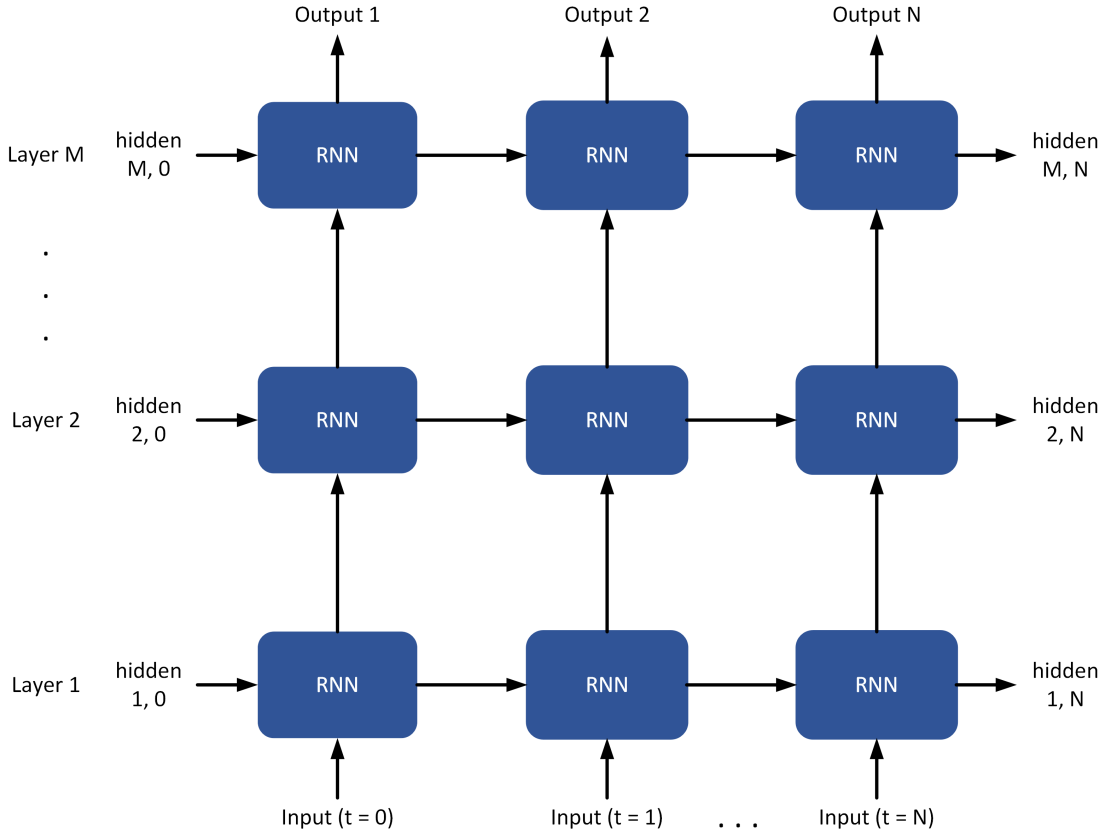


Figure 3.8. *A basic multi layered RNN with stacked architecture. Adapted from [35].*

The used RNN unit in this thesis is LSTM. Of the three RNN units presented above, LSTM is the most complex and robust against vanishing or exploding gradient problems. As seen from Figure 3.7 above, LSTM also uses the cell state component, which is not

present in Elman or GRU units. The hidden state component is considered as the short-term memory component and the cell state component corresponds to the long-term memory component. To control the memorizing process, LSTM has a gating mechanism that includes a forget gate, an input gate, and an output gate. Those gates regulate the flow of input and output information. The forget gate decides how much old information should be removed from the current cell state. The input gate decides how much new information should be added to the current cell state. The output gate generates the output and hidden state based on the current cell state. [35]

Equation for the LSTM cell state component C_t is

$$C_t = i_t \odot \widetilde{C}_t + f_t \odot C_{t-1}, \tag{3.2}$$

where i_t is the input gate vector, $\odot$ is element wise multiplication, $\widetilde{C}_t$ is the candidate cell state vector, f_t is the forget gate vector and C_{t-1} is the previous cell state vector. Equation for the LSTM hidden state component h_t and output Z_t is

$$h_t = o_t \odot \tanh\left(C_t\right) = Z_t, \tag{3.3}$$

where o_t is the output gate vector. [35] Neuron-like calculations with activations, weights, and biases are applied in i_t, f_t, o_t and $\widetilde{C}_t$, where the previous hidden state h_{t-1} and input data X_t are used. Equations for those can be checked in more detail here [35].

3.3 Hyperparameters

Hyperparameters are high-level parameters that are set manually before starting the training process. Hyperparameter tuning is one of the most important steps for building efficient models. In neural network-based approaches possible hyperparameters are, for example, number of layers, number of neurons, number of epochs, learning rate, batch size, activation functions, loss function, optimization algorithm, normalization, and regularization techniques. [36] A few of these hyperparameters are discussed in the following subsections. Also, depending on the type of the neural network some model-specific hyperparameters may occur.

3.3.1 Activation functions

As previously mentioned, activation functions are often used for a non-linear mapping between the inputs and output of a neuron. Activation functions are used in thousands or even millions of neurons and their derivatives are used during the training. So, both the function and its derivative should be computationally efficient. [36]

One of the most common activation functions and their gradients, which are also used in

this thesis, are sigmoid

$$sigmoid(x) = \frac{1}{1 + e^{-x}}, \tag{3.4}$$

$$\frac{d(sigmoid(x))}{dx} = sigmoid(x)(1 - sigmoid(x)), \tag{3.5}$$

hyperbolic tangent function (Tanh)

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{3.6}$$

$$\frac{d(Tanh(x))}{dx} = 1 - Tanh^{2}(x) \tag{3.7}$$

and rectified linear unit (ReLU)

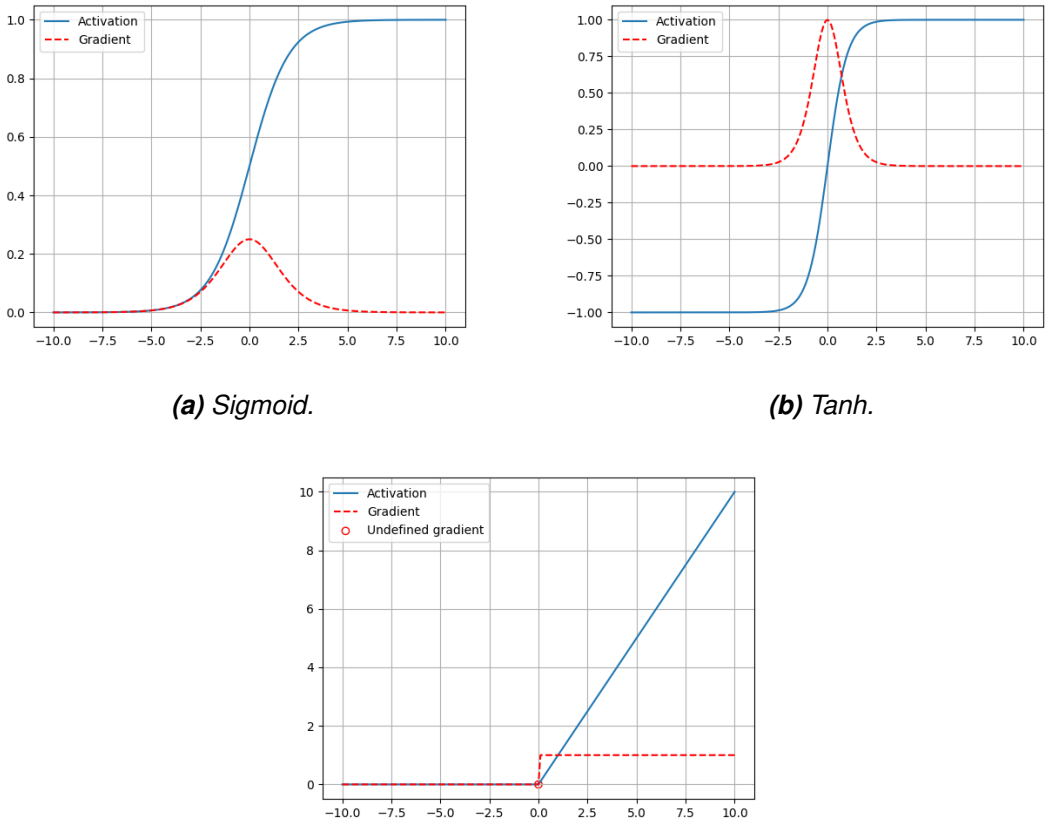
$$ReLU(x) = max(0, x), \tag{3.8}$$

$$\frac{d(ReLU(x))}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undefined} & \text{if } x = 0. \end{cases} \tag{3.9}$$

Visual presentation of those activations and gradients are presented in Figure 3.9. Sigmoid was originally designed for binary classification tasks and its output can be interpreted as a probability distribution over a binary variable. Computing the derivative of the sigmoid is relatively simple but one major problem is the vanishing gradient. Tanh, also called a symmetric sigmoid, might provide better performance and faster convergence than the sigmoid because of larger gradients and zero-centered nature. Especially with RNNs where the vanishing gradient problem is more present. In practice, ReLU and its variants are the most widely used activation functions. The biggest advantages of ReLU are gradient stability and computationally efficient structure. However, ReLU activations are biased to be positive or can lead to dead neurons if the activations are always zero. [37] In the output layer of a multi-class classifier, the most relevant activation function is softmax

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}, \tag{3.10}$$

which can be seen as a generalization of the sigmoid function. Softmax represents a probability distribution over C number of different classes. It is often used as the final output of a multi-class classifier but rarely used inside the model. [32]



(a) *Sigmoid.*

(b) *Tanh.*

(c) *ReLU.*

Figure 3.9. *Activation and gradient comparison.*

3.3.2 Loss functions

A loss function, also called a cost function, measures the difference between the correct and predicted output as a number. That measurement is then used by an optimization algorithm to adjust the model's weights to minimize the loss function. [25] Choosing a suitable loss function mainly depends on the used activation in the output layer and on the task, whether it is binary or multi-class classification or regression task, and so on. [36]

A couple of the most common loss functions are mean squared error and cross-entropy loss, also known as log loss. Mean squared error calculates the mean from squared differences between the predicted and correct values. Cross-entropy loss utilizes probabilities and logarithms for measuring the difference. [25] Since this thesis focuses on binary and multi-class classification-related tasks, cross-entropy loss is used and adapted for each classification task. For multi-class classification cross-entropy loss can be defined as

$$cross - entropy\ loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{p}_{ij}), \tag{3.11}$$

where N is the batch size, C is the number of classes, y_{ij} represents the ground truth value from one-hot encoded format and $\hat{p}_{ij}$ refers to the predicted probability from softmax. For binary classification, cross-entropy loss can be defined as

$$cross - entropy\ loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i), \tag{3.12}$$

where N is the batch size, y_i represents the ground truth in binary (0 or 1) format and $\hat{p}_i$ refers to the predicted probability of class being 1 from sigmoid. [38]

3.3.3 Optimization algorithms

As mentioned above, optimization algorithms, also called optimizers, are used to adjust the model's weights to minimize the loss function. Those adjustments are done during the training process and usually, weight updates are performed after processing a batch size amount of samples. [36] Some of the most common optimization algorithms are based on a gradient descent algorithm.

The basic idea of gradient descent is to minimize the loss function L, parameterized by the model's weights w, by updating the weights in the opposite direction of the gradient of the loss function with respect to the weights. The equation of the gradient descent weight update is

$$w_t = w_{t-1} - \alpha\frac{\partial L(w)}{\partial w}, \tag{3.13}$$

where w represents the weights, α is the learning rate and L is the loss function. [39] The used optimization algorithm in this thesis is adaptive moment estimation (Adam) [40], which is an advanced variant of the gradient descent. Adam is designed to be computationally efficient and to be used with large datasets and/or high dimensional weight space. The equation of the Adam weight update is

$$w_t = w_{t-1} - \alpha\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{3.14}$$

where w represents weights, α is the learning rate, $\hat{m}$ is the bias-corrected first moment estimate, $\hat{v}$ is the bias-corrected second moment estimate and ϵ is a small constant to avoid division by zero. The loss gradient with respect to the weights is used in the moment estimates, $\hat{m}$ and $\hat{v}$, but the exact equations that contain a couple of steps and additional parameters can be checked from the original paper of Adam. [40] With gradient descent-based approaches, usually a backpropagation algorithm is used for efficiently calculating the loss gradient with respect to the weights. The backpropagation algorithm partially

calculates the gradient from output to input, one layer at a time. It avoids redundant calculations by reusing the computations of the current layer for the next layer. [32]

3.3.4 Regularization techniques

Regularization techniques are used to prevent overfitting and underfitting the model. The goal is to minimize the test error, possibly at the expense of increasing the training error. [32] There are many different methods for regularization but the ones that are used in this thesis are called data augmentation, dropout, and early stopping.

Data augmentation is a technique to artificially increase the size of the dataset. Training the model with more data is often the best way to better generalization and hence increase the performance. Usually, existing data samples are duplicated and modified using various techniques, such as adding noise, rotation, or masking elements. [32]

Dropout is used during the training to randomly drop neurons from certain layers of the model with some dropout rate. That means that some neuron outputs are set to zero. Different combinations of randomly dropped neurons can be seen to create a new model whenever neurons are dropped. That is why dropout can be seen as a practical and efficient way of combining several different models into one model. [32]

One of the most commonly used regularization techniques is early stopping. During training, the idea is to observe when the validation loss starts to increase, despite the training loss might be steadily decreasing. The model is saved whenever the validation loss has decreased from the lowest value so far. To save time, training is usually stopped if the validation loss has not decreased after a predefined number of patience epochs. Test results should be better when obtaining the model at the lowest validation loss rather than at the latest epoch, which might have the lowest training loss. [32]

3.4 Performance evaluation metrics

This thesis focuses on multi-class and binary classification tasks. Multi-class classification means that the input is classified into one, and only one, over C non-overlapping classes. Binary classification means that the input is classified into one, and only one, over two non-overlapping classes. To evaluate the correctness of classifications three different metrics are used in this thesis. Those metrics are precision, recall, and accuracy. Depending on the classification task and used definitions, those metrics have a little bit different meanings and equations. In general, accuracy is the most often used metric and there is no well-developed analysis of precision and recall with multi-class classification. [41] For these reasons, only accuracy metrics are used in multi-class classification in this thesis.

Usually, the comparisons of ground truth and predicted values are presented in a matrix form that is called a confusion matrix. The confusion matrix provides the key numbers for calculating almost any type of evaluation metric. An example of the confusion matrix for binary classification is presented in Table 3.1, where it can be seen that the main diagonal elements from the top left represent the correct classifications. [41] For multi-class classification the confusion matrix can be generalized over all classes. An example of the confusion matrix for multi-class classification is presented in Figure 3.2, where it can be seen that the notation of each cell depends on the class to observe. Again, only the main diagonal elements from the top left are usually considered as the correct classifications. [42]

Table 3.1. *An example confusion matrix for binary classification.*

	Predicted as positive	**Predicted as negative**
Ground truth is positive	True positive (TP)	False negative (FN)
Ground truth is negative	False positive (FP)	True negative (TN)

Table 3.2. *An example confusion matrix for multi-class classification observing class B.*

	Predicted as A	**Predicted as B**	**Predicted as C**
Ground truth is A	TN_B	FP_B	TN_B
Ground truth is B	FN_B	TP_B	FN_B
Ground truth is C	TN_B	FP_B	TN_B

For binary classification, the above-mentioned three metrics can be defined as

$$precision = \frac{TP}{TP + FP}, \tag{3.15}$$

$$recall = \frac{TP}{TP + FN}, \tag{3.16}$$

and accuracy as an overall measurement, usually simply called just as an accuracy,

$$overall\ accuracy = \frac{TP + TN}{TP + FN + FP + TN}, \tag{3.17}$$

or as an average measurement, also called as a balanced accuracy,

$$average\ accuracy = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right). \tag{3.18}$$

Precision measures the correctness of positive predictions and recall measures the completeness of positive measurements. Precision and recall can be seen as trade-off met-

rics between each other, since maximizing the performance with another metric usually lowers the other one. Accuracy measures the total effectiveness of a classifier. [41] If the test dataset has a significant imbalance, performances with the most frequent classes have a greater impact on the overall accuracy. With average accuracy, every class has an equal impact despite the imbalances in the test dataset. [42] That is why accuracy can be defined as an overall or average measurement and both measurements are used in this thesis.

For multi-class classification, the accuracy metrics, as well as the others, could be generalized similarly from the binary classification metrics [41]. However, often the accuracies for multi-class classification are simply defined as fractions of correct classifications. Although, then the multi-class accuracies are technically multi-class recall measurements. The overall accuracy for multi-class classification can be then defined as a fraction of correct classifications

$$overall\ accuracy = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} TP_i + FN_i}, \tag{3.19}$$

where C is the number of classes. The average accuracy can be defined as an average over the accuracy of each class

$$average\ accuracy = \frac{\sum_{i=1}^{C} \frac{TP_i}{TP_i + FN_i}}{C}, \tag{3.20}$$

where C is the number of classes. [42]

With a perfectly balanced test dataset, meaning that each class has an equal amount of samples, overall and average accuracies would be identical. Small differences between the average and overall accuracies indicate that the imbalance in the test dataset or the bias towards certain classes in the model predictions is not significant. On the other hand, big differences between the average and overall accuracies indicate that the imbalance in the test dataset and the bias towards certain classes are significant. Imbalances in the training data might cause a bias towards certain classes in the model predictions and then cause insufficient accuracy levels. [42]

3.5 Orthogonal frequency-division multiplexing

OFDM is a multi-carrier modulation scheme that is widely used in many wireless telecommunication applications. In OFDM data is transmitted as a parallel stream of multiple orthogonal narrowband signals known as subcarriers. Because of the orthogonality, the subcarriers are not disturbing each other despite being overlapped and then save bandwidth. That is achieved by carefully selecting a constant subcarrier spacing (SCS) in

frequency between all of the subcarriers. The SCS is defined as an inverse of the OFDM symbol duration, which contains all of the subcarriers. Equivalently, an inverse of the SCS defines the OFDM symbol duration. [43] An example of the OFDM symbol and its subcarriers in the time and frequency domain is presented in Figure 3.10.

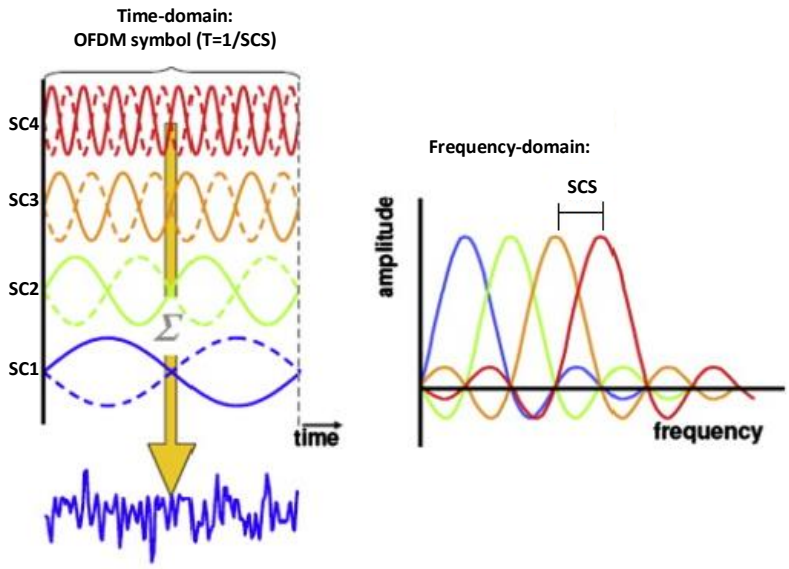


Figure 3.10. *An example of the OFDM symbol and its subcarriers, denoted as SC, in time and frequency domains. Adapted from [44].*

In OFDM each subcarrier is modulated with a conventional digital modulation, usually with a quadrature amplitude modulation (QAM), at a low data rate. Since multiple parallel subcarriers are transmitting at a low data rate, the total data rate with OFDM is similar to what could be achieved with a conventional single carrier modulation. Before transmitting an OFDM symbol, usually a few samples from the end are added to the beginning in the time domain. That operation is called a cyclic prefix (CP) and it provides a guard interval between two OFDM symbols. CP helps to reduce the inter-symbol interference and processing operations in the receiver. The benefits of OFDM are spectral efficiency, ease of sharing optimal resources, robustness against multipath fading, and a simple channel equalization process. However, OFDM waveforms are sensitive to phase noise and frequency offsets. [43]

3.6 5G New Radio

5G new radio (NR) is the 5th generation of mobile networks. 5G is designed to deliver capacity enhancements with multi-Gbps peak data rates, massive connectivity, ultra-reliable, and low-latency communications. Also, the increased performance of 5G enables new use cases and applications in various fields. The waveform of 5G in uplink and downlink is based on OFDM, with multiple SCS options from 15 kHz up to 240 kHz. [1]

To satisfy the requirements of various use cases in 5G, multiple different numerologies are defined and possibly mixed and used simultaneously depending on the needs. The numerology is defined based on the certain SCS and CP. For example, lower SCS would be suitable to cover greater areas, whereas higher SCS would be suitable for shorter latency. [1] An example of the supported numerologies is shown in Table 3.3, where it can be seen that the SCS is doubled in every step.

Table 3.3. *An example of 5G numerologies. [1]*

Numerology (μ)	**SCS** = $2^{\mu} * 15$ kHz	**Cyclic prefix**
0	15	Normal
1	30	Normal
2	60	Normal/extended
3	120	Normal
4	240	Normal

Because of the variating numerology, the frame structure of 5G is also adaptable. However, the length of a radio frame is always 10 ms and the length of a subframe is 1 ms. With normal CP the number of OFDM symbols in a slot is always 14 symbols. Also, a PRB always consists of 12 subcarriers. The number of slots, and hence the number of symbols, in a subframe depends on the used numerology. Higher SCS has a shorter symbol duration and then the number of slots in a subframe increases. An example of the 5G frame structure is presented in Figure 3.11. A basic scheduling unit in 5G is a PRB over one or many OFDM symbols. One subcarrier over one OFDM symbol is also called a resource element (RE). Multiple PRBs in the same bandwidth form a resource grid as presented over a slot in Figure 3.12. [1]

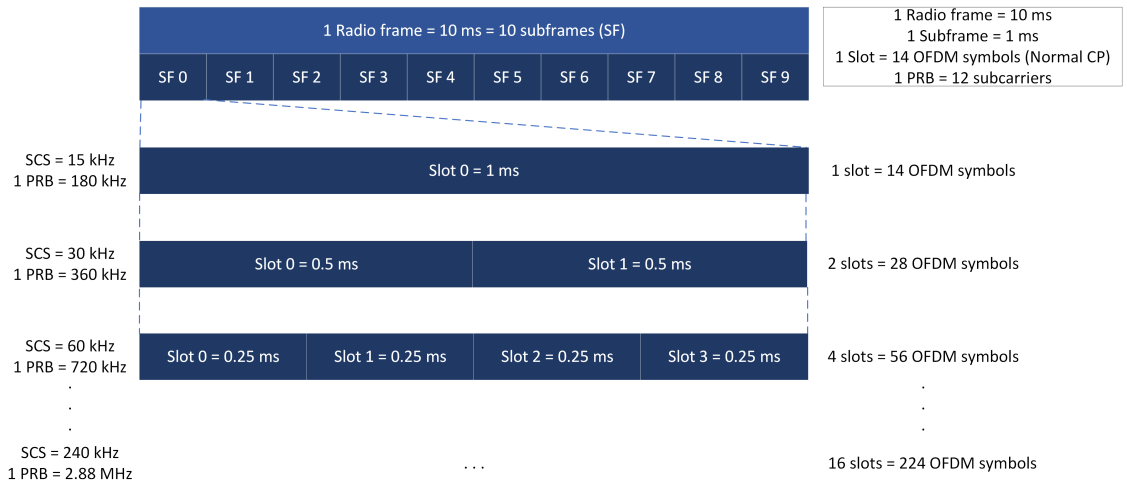


Figure 3.11. *An example of the 5G frame structure. Adapted from [1].*

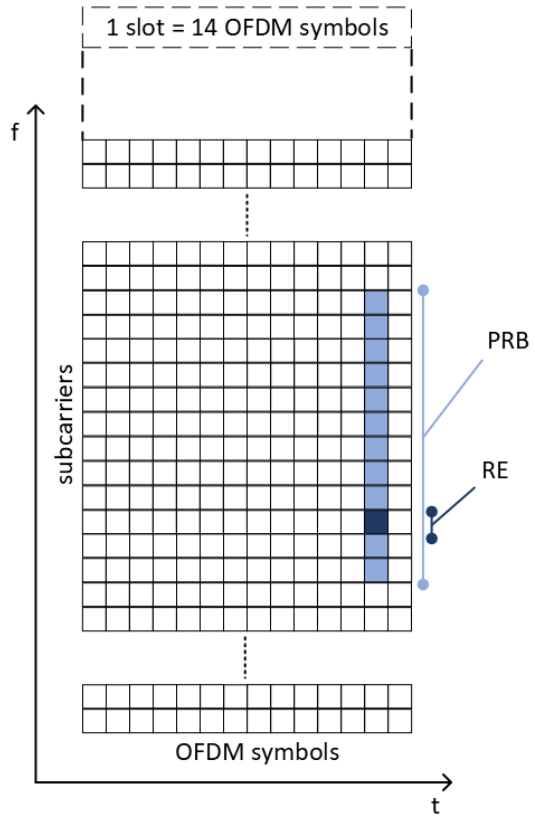


Figure 3.12. *An illustration of the 5G resource grid. Adapted from [1].*

Resources from the resource grid are occupied and scheduled to different physical layer channels. Physical layer channels in the uplink are used to transmit data from the UEs to the base station. Some of the key physical layer uplink channels in 5G are physical uplink shared channel (PUSCH), physical random access channel (PRACH) and physical uplink control channel (PUCCH). PUSCH is the primary uplink channel to transmit user data and represents the majority of the allocated resources. PRACH is used by the UE to initiate the connection with the base station. PUCCH is used to transmit control information from the UE to the base station. However, uplink control information (UCI) can also be carried in the PUSCH instead of PUCCH. [2] An example of the 5G uplink channel mapping between different layers is presented in Figure 3.13. Since this thesis focuses only on the uplink physical layer, channels in the other layers or in the downlink are not further introduced. Also, the exact contents or resource allocations of the 5G uplink channels are not further described. Resource allocations in this thesis are simulated by randomly masking resources in the grid with certain masking rates. More details of the simulations are presented in Chapter 4.

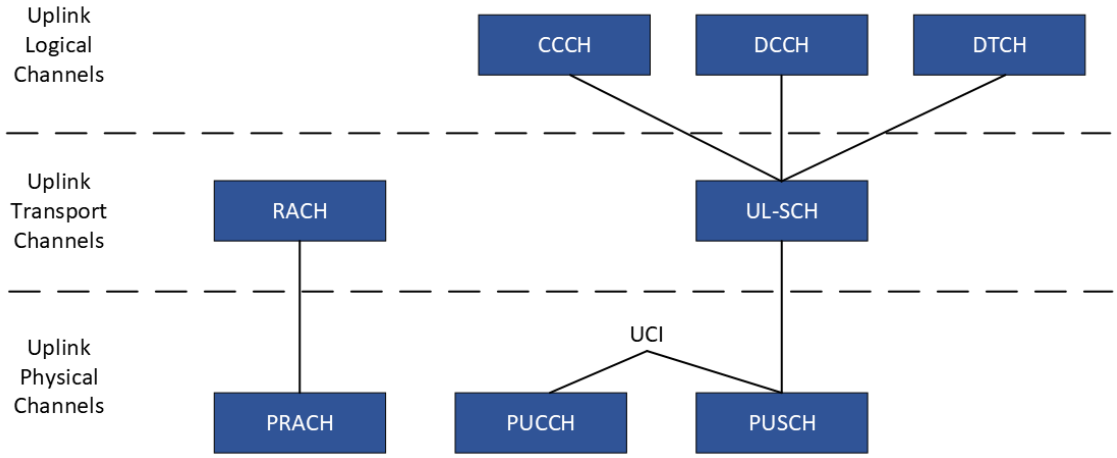


Figure 3.13. *An example of the 5G uplink channel mapping. Adapted from [45].*

3.7 Jamming and interference

Jamming is an intentional interference generated by an external source that tries to disrupt legitimate communications in the networks. Jamming attacks are a subset of DoS attacks which may cause several other higher-layer security problems. Because of the exposed nature of wireless links, all wireless networks can be easily attacked by jamming techniques. Interference can also be unintentional by other nodes or devices in the wireless network. However, the main conceptual difference is that jamming is conducted by an attacker who intentionally attacks the network, and regular interference is unintentional interference. [46] Different unintentional interference scenarios are not further described, since the main focus is to study intentional jamming attacks.

There are several different jamming models, which are based on many different attributes as illustrated in Figure 3.14. Even some more advanced jamming models are developed for electronic warfare purposes that are not publicly available. An ideal jamming model should have a high success rate, low probability of detection, low power consumption, and be resilient to anti-jamming techniques. [3] Since the jamming and interference models in this thesis are quite simple and generic, the most important attributes are active time, frequency selection, and frequency range.

Active time refers to the selection of time slots where the jammer is active. In constant jamming, the jammer is transmitting continuously, which makes it energy inefficient. In random jamming, the transmission happens at randomly selected time slots and then sleeps otherwise, which makes it more energy efficient than constant jamming. In pulse jamming, the durations of the sleeping phase and jamming phase are predefined. [3]

Frequency selection refers to a strategy to move from one frequency to another. Static selection means that the jammer is targeting certain frequencies all the time without switching the frequencies. Static selection is simple to implement, but it can be easily avoided

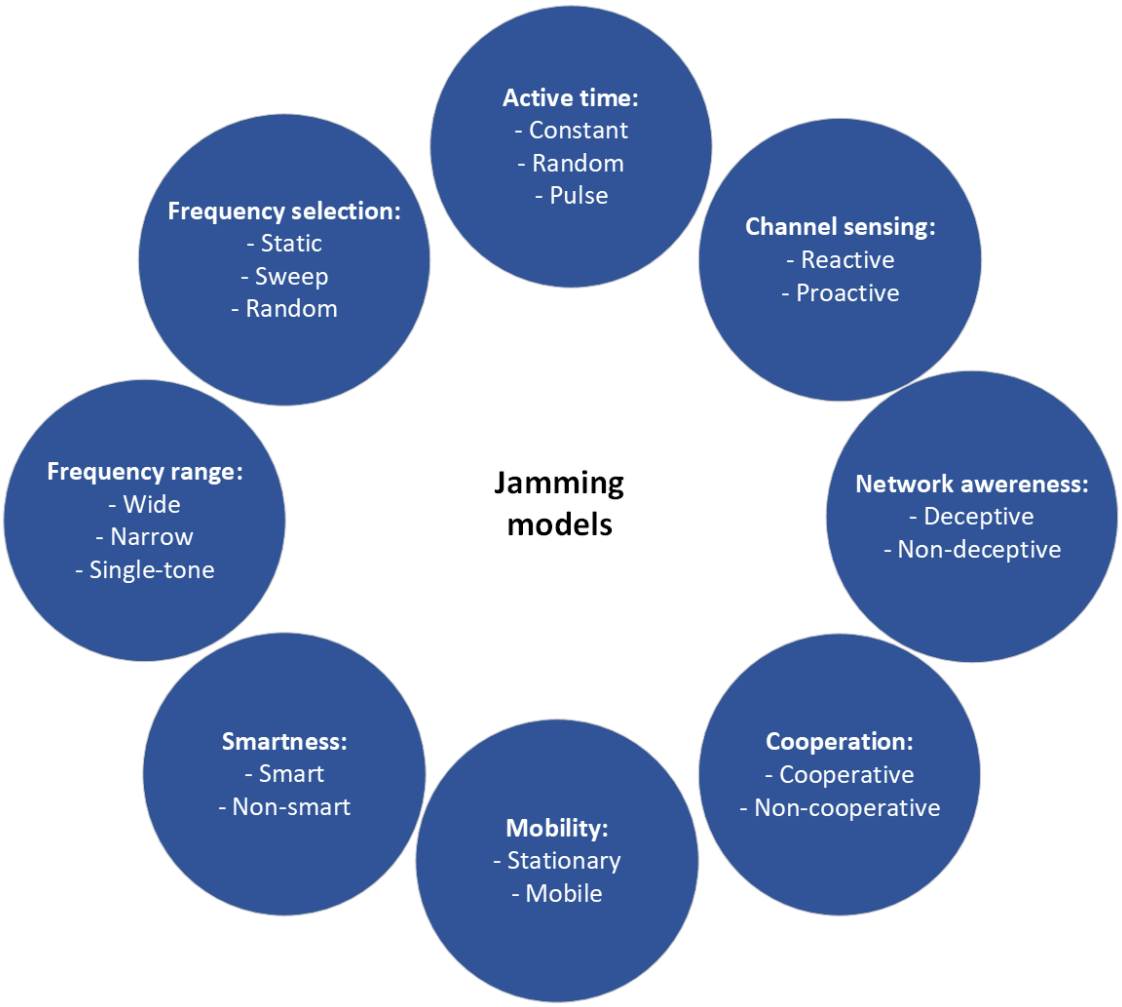


Figure 3.14. *An overview of different attributes in jamming models. Adapted from [3].*

by changing the operating channel. In sweep jamming, the jammer sweeps across the selected spectrum sequentially or non-sequentially. Sweeping the selected spectrum might be useful when the target frequencies are unknown or if the target frequencies are also changing. Random frequency selection means that the jammer randomly chooses a frequency band to disrupt from a set of targeted frequency bands. Random jamming may not be very effective, since it might keep jamming idle bands. However, it might be more difficult to find an anti-jamming strategy for that random behavior than for static and sweep jamming. Some special cases of random selection might be jammers that are not completely random but follow some logic, for example, the Markov chain. [3]

Frequency range refers to the instantaneous frequency range that the jammer can attack. Wide range refers to the ability to disrupt wide bandwidth, whereas narrow range refers to narrow bandwidth. In single-tone jamming, the jammer is targeting only one frequency. Wide bandwidth jamming requires a lot of energy, but it is almost impossible to avoid it in the target network. Narrow bandwidth and single-tone jamming can be very energy efficient, but again easy to avoid by changing the operating channel. [3] Usually, the

frequency range is divided into two categories, single-tone and multi-tone, but for this thesis, three different ranges are defined. The limit between the wide and narrow range is not strictly defined and depends on the available bandwidth.

Channel sensing refers to the ability to detect activity in the wireless channel. A reactive jammer starts its jamming procedure after detecting certain activity in the channel, whereas a proactive jammer has no knowledge of the channel activity. Most of the jammers are targeting the physical layer. However, network awareness refers to prior knowledge about the network protocol and the jammer may target the attacks to other layers. A deceptive jammer is aware of the existing protocol in the targeted network and transmits legitimate packets at a high rate over a period of time. A non-deceptive jammer transmits random bits without following any communication protocol. [3]

With cooperation, the jamming attack can be generated in a coordinated way using multiple jammers or just a standalone jammer. Mobility refers to the capability of moving the jammer from one location to another. Smart jammers can quickly learn the transmission patterns in the network and adjust jamming strategies to maximize the damage or efficiency. Smart jammers might use ML to learn these patterns in the targeted network and then select optimal attributes for the jamming model. A non-smart jammer follows a fixed strategy and may not have any intelligent capabilities. [3]

The jammers that are used in this thesis are called sweep, spot, pulse, and barrage. Sweep jammer has a constant active time but changes its frequency sequentially over time, hence sweeps the bandwidth. Spot and barrage jammers are also constant in active time but static in frequency selection. The difference between the spot and barrage is that the spot has a narrow frequency range and the barrage has a wide frequency range. Pulse jammer has a static frequency selection, but as the name says, has a pulse type active time. An example of those jammers in a resource grid is presented in Figure 3.15. All of the implementation details are presented in Section 4.1 below. The used jammers do not follow any type of advanced or intelligent attributes like channel sensing, network awareness, cooperation, or smartness. However, all of the jammers are generated over multiple randomly variating parameters. So, the jamming attacks that might have some intelligent attributes, could look very similar in the physical domain to the current ones. The used jammers can also present an unintentional interference, but for simplicity reasons, all of the used jammers are considered to be jamming attacks.

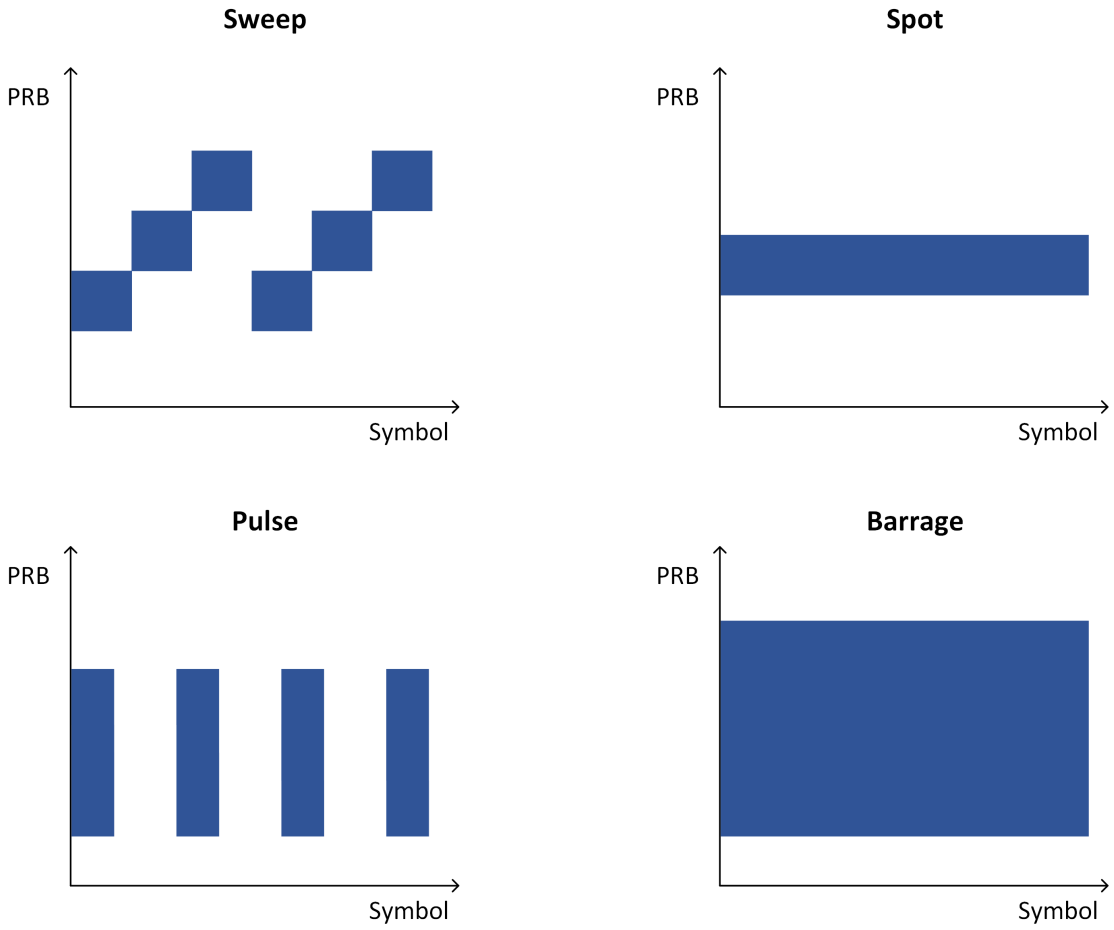


Figure 3.15. *An example of the used jammers in a resource grid.*

4. IMPLEMENTATION

This chapter presents the technical implementation details and processes of this thesis. Dataset generation and detailed information are discussed in Section 4.1. Architectures, training, testing, and optimization of the DL models are presented in Sections 4.2-4.4.

4.1 Dataset

The used dataset models scenarios of 5G NR uplink waveform with and without jamming attack. The dataset is generated with Matlab R2022a using 5G Toolbox. Four types of jammers are used and the samples without any jammer contain only noise. So, the dataset contains five classes, which are sweep, spot, pulse, barrage, and noise. The used features are time-frequency power heatmaps. Each input heatmap contains 133 PRB wise average powers over 14 OFDM symbols. The maximum bandwidth is configured to 50 MHz. For modulating the jamming signals 30 kHz SCS and 16QAM with random bits are used. All of the jamming signals are going through the TDLC300 delay profile, with random seeds, and noise is added in the receiver. The signal-to-noise ratio (SNR) is set to 10 dB and the signal-to-interference ratio (SIR) varies between 0 to 20 dB with steps of 1 dB. The used definition for jamming-to-noise ratio (JNR) is

$$JNR(dB) = SNR(dB) - SIR(dB), \tag{4.1}$$

so the JNR variate between -10 to 10 dB. All of the general system parameters are presented in Table 4.1.

Data for each jammer type is randomly generated over certain parameter limits. Each jammer type has five different bandwidths. With one bandwidth value rest of the jammer parameters are randomized until the desired amount of samples is achieved for a certain bandwidth. That process is repeated over all JNR points. Parameters for each jammer type are presented in Figure 4.1. The total amount of heatmaps without data augmentation is $21 * 100 * 5 = 10\,500$, where 21 is the amount of JNR points, 100 is the number of randomized heatmaps and 5 is the number of classes including noise. The final dataset includes data augmentation, which brings some imbalance between the occurrences of each class. Data augmentation is presented in more detail below.

Table 4.1. *General system parameters.*

Parameter	Value
SNR	10 dB
SIR	0, 1, 2, ..., 20 dB
JNR	-10, -9, -8, ..., 10 dB
Max bandwidth	50 MHz
SCS	30 kHz
Subcarriers per PRB	12
Number of PRBs	133
Number of OFDM symbols	14
Modulation type	16QAM
Delay profile	TDLC300
Tx & Rx antennas	1

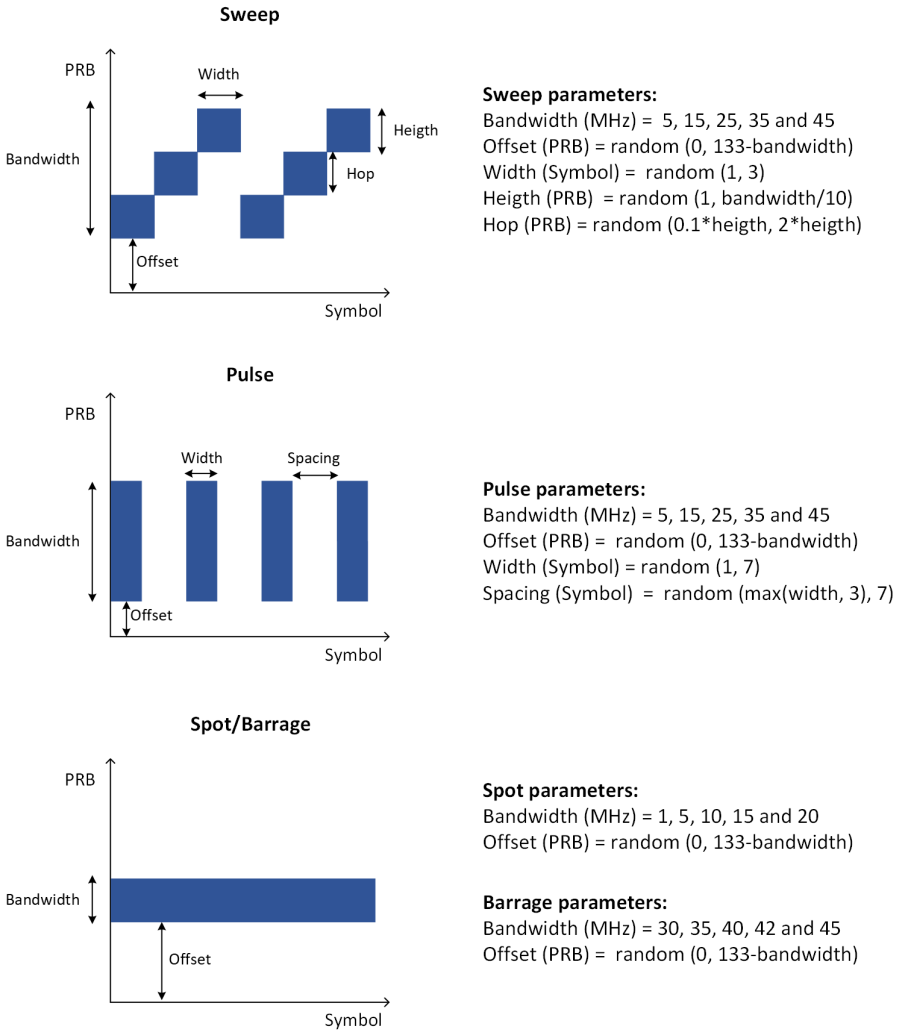


Figure 4.1. *Parameters for each jammer type.*

The dataset is further divided into two different sets before the data augmentation. Based on the classification task those datasets are called binary and multi-class datasets. That also defines the target or label for the input heatmap. The division between the training, validation, and testing data is 60%, 20%, and 20% respectively. The division is done in an equal way so that every split has an equal amount of different classes with different parameters.

For binary classification, the target is 1 (jammed) if the heatmap contains any jammer or 0 (not jammed) if the heatmap contains only noise. Since there are more classes for jamming than noise, the binary dataset is balanced by taking an equal amount of noise and jammer heatmaps. Also, the sampling of the jammers is done equally. Every jammer has an equal amount of heatmaps with different parameters. Before data augmentation, the binary dataset has $2\ 100(not\ jammed) + 2\ 100(jammed) = 4\ 200$ heatmaps. For multi-class classification, every class is denoted with a unique label to distinguish the different classes. The size of the multi-class dataset, before data augmentation, is the same as mentioned in the previous paragraph, which is 10 500 heatmaps.

The data augmentation includes masking, which simulates the resource allocations for the channels. Masking means that a certain amount of the elements in the input heatmap are set to zero in a random pattern. Masked heatmaps are then added to the dataset, so the final dataset is larger than the original dataset. Masking levels are set to mask 10%, 30%, 40%, and 50% of the input heatmap. Jamming models, which have under 15 MHz bandwidth are not masked. The random masking operation would likely mask too much or all of the jamming in these scenarios, which may not be desirable for DL model training.

After the data augmentation, the final datasets are formed. The sizes of the final datasets are presented in Table 4.2. The occurrences of each class in the final binary dataset are presented in Table 4.3 and for the final multi-class dataset in Table 4.4. It can be seen that the imbalance from the data augmentation is now present. All classes do not get an equal amount of masked samples, since that depends on the bandwidths. The final datasets could have been balanced by reducing the most frequent classes or increasing the least frequent classes. However, no further balancing was executed, since the imbalance is relatively small [47] and all classes have at least a sufficient amount of samples. Also, in DL more data with a small imbalance is often a better option than less data with perfect balance.

Table 4.2. *Number of heatmaps for the final dataset splits.*

	Train	**Validation**	**Test**	**Total**
Binary	10 572	3 520	3 536	17 628
Multi-class	23 400	7 792	7 868	39 060
Original	N/A	N/A	N/A	10 500

Table 4.3. *Number of heatmaps over classes for the final binary dataset splits.*

(Binary)	Sweep	Pulse	Noise	Barrage	Spot	Total
Train	1 127	1 019	6 300	1 575	551	10 572
Validation	357	361	2 100	525	177	3 520
Test	333	365	2 100	525	213	3 536
Total	1 817	1 745	10 500	2 625	941	17 628

Table 4.4. *Number of heatmaps over classes for the final multi-class dataset splits.*

(Multi-class)	Sweep	Pulse	Noise	Barrage	Spot	Total
Train	4 384	4 220	6 300	6 300	2 196	23 400
Validation	1 404	1 456	2 100	2 100	732	7 792
Test	1 352	1 464	2 100	2 100	852	7 868
Total	7 140	7 140	10 500	10 500	3 780	39 060

4.2 Models

In this thesis, three deep neural network-based models are proposed. Those models are CNN, LSTM and combined CNN-LSTM. Each model has separate instances for the two different classification tasks. The only architectural difference between the two instances of the model is the output activation in the last layer. Therefore, a total of six different model instances are trained and evaluated. Both the CNN and LSTM are well-known approaches for this type of input containing time and frequency information as a 2-D grid. The combined CNN-LSTM utilizes feature extraction from the CNN and time-dependency analysis from the LSTM. All of the models are trained and evaluated with the same dataset, which depends on the classification task. Also, the number of layers in each model is within the same range. Models are trained and implemented with Tensorflow 2.10 framework and NVIDIA Tesla M40 graphics card. In this section, all of the presented model architectures are the final ones after the optimization steps. Optimization steps are described in Section 4.4 below.

The architecture of the CNN model is presented in Figure 4.2. The CNN model consists of three convolutional layers and two fully connected (dense) layers. Between the convolutional and fully connected layers, a flatten layer is used to transform the data into 1-D. All three convolutional layers use stride 1x1, kernel size 3x3, and padding same. Therefore, convolutional layers do not perform down-sampling before max pooling. The first convolutional layer uses 16 filters, the second 32, and the third 64. All convolutional layers are followed by a ReLU activation and max pooling with a 2x2 window and valid padding. ReLU activation is also used in the first fully connected layer. Between the two fully connected layers a dropout of 0.8 is used. As mentioned above, each model has

separate instances for the two classification tasks and the only architectural difference is the output activation in the last layer. For binary classification, a sigmoid activation with 1 output is used in the last fully connected layer. For multi-class classification, a softmax activation with 5 outputs is used.

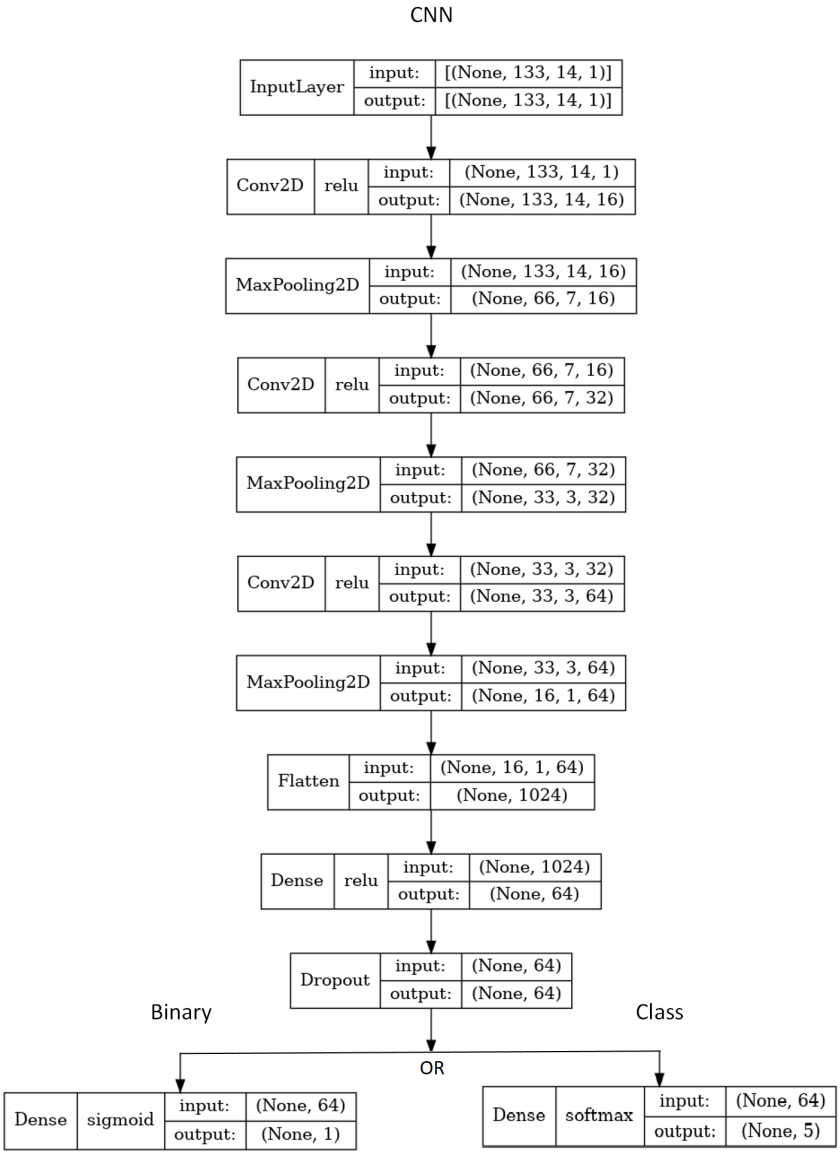


Figure 4.2. *CNN model architecture, where the final output is dependent on the classification task. The last output dimension in the Conv2D denotes the number of filters. The first dimension in the input and output denotes the pre-defined batch size. That is usually None since the model should be able to handle any batch size if not explicitly defined.*

The architecture of the LSTM model is presented in Figure 4.3. The LSTM model consists of three LSTM layers and one fully-connected layer. Again, a flatten layer is used before the fully connected layer. First, the permute layer swaps the axis of the input features and time steps corresponding to the required form for the LSTM. That operation could have been implemented outside of the model but for convenience reasons, it was implemented inside of the model. All three LSTM layers use 32 units and return full sequences.

Therefore, the output shapes of all LSTM layers are the same, where 14 corresponds to the number of time steps and 32 to the number of output features. All LSTM layers use Tanh activation for the outputs and sigmoid activation for the inner recurrent activations. Dropout of 0.7 is used before the fully connected layer. Activation in the last fully connected layer is again dependent on the classification task, as described with the CNN.

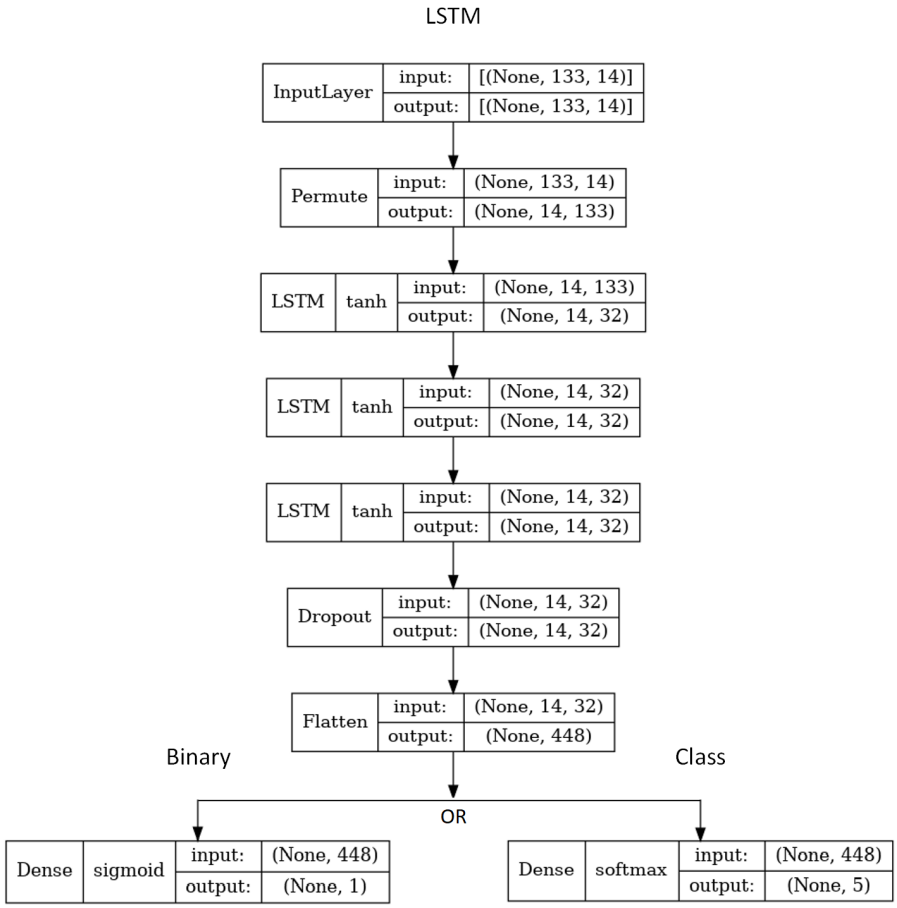


Figure 4.3. *LSTM model architecture, where the final output is dependent on the classification task. The last output dimension in the LSTM layer denotes the number of units.*

The architecture of the CNN-LSTM model is presented in Figure 4.4. The CNN-LSTM model consists of two convolutional layers, two LSTM layers, and one fully connected layer. It basically combines the first two layers of the above-described CNN and LSTM architectures. Also, the parameters for each layer are similar to the above descriptions. The permutation and reshape layers in the middle are used to map the 3-D outputs from the convolutional layers into 2-D input for the LSTM layers. Both convolutional layers again use stride 1x1, kernel size 3x3, and padding same. The first convolutional layer uses 32 filters and the second only one filter to avoid additional down-sampling before the reshape. Max pooling is performed only once after the first convolutional layer. Both LSTM layers again use 32 units and return full sequences. Because of the max pooling, the output shapes of the LSTM layers now include 7 time steps and 32 features. Dropout of 0.5 is used after every layer before the fully connected layer. Activation in the last fully

connected layer is again dependent on the classification task. The number of parameters, the used optimizer, and the losses are presented in Section 4.3 below.

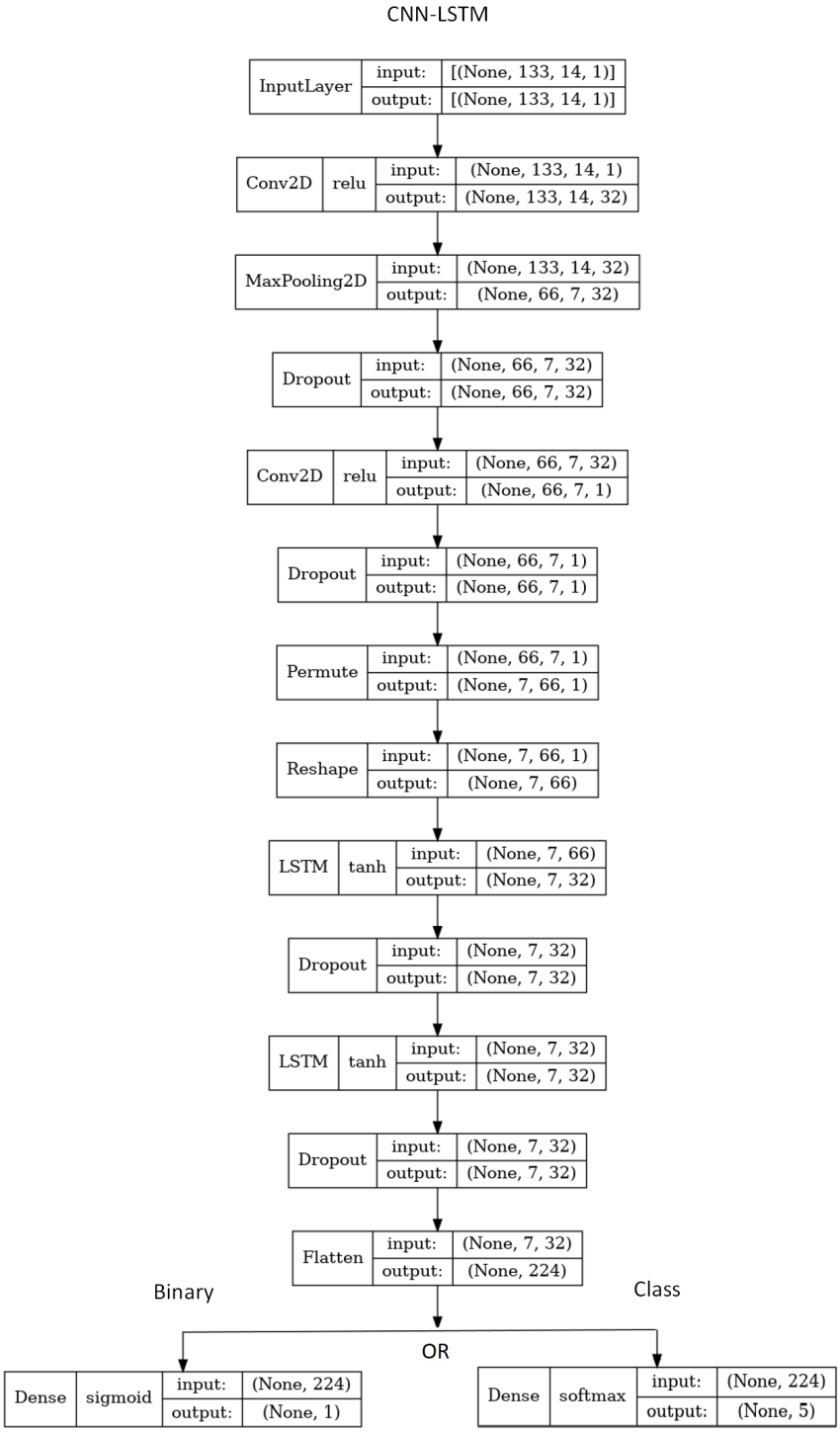


Figure 4.4. *CNN-LSTM model architecture, where the final output is dependent on the classification task. The reshape layer is used to transform the input into 2-D for LSTM.*

4.3 Training and testing process

As described above, the used dataset is dependent on the two different classification tasks, which are binary and multi-class. Those datasets are further divided into training, validation, and testing datasets. Training data is used for training the models. Batch processing and training data shuffling are used to enhance the training process. Validation data is used to evaluate the model performance after every epoch during the training. Also, the early stopping condition is checked based on the validation data loss. Testing data is used after all of the training and optimizations are completed. This section observes only the final models, which have already gone through the optimization process. Optimization steps are discussed in Section 4.4 below.

The general training parameters are presented in Table 4.5. The number of patience epochs is set to 30. That means that the training is stopped if the validation loss is not decreased during that time. The used optimizer is Adam with Tensorflow default parameters. The used loss function is binary or categorical cross-entropy depending on the classification task. The categorical cross-entropy loss is based on the one-hot encoded targets against the outputs of the softmax. The binary cross-entropy loss is based on the single target value of 0 or 1 against the output of the sigmoid. The used performance metric is binary or categorical (overall) accuracy, which are based on similar types of comparisons as the used loss.

Table 4.5. *General training parameters.*

Parameter	Value
Max epochs	100
Patience epochs	30
Batch size	32
Learning rate	1e-3
Optimizer	Adam
Loss	Binary/Categorical Cross-entropy
Performance metric	Binary/Categorical (Overall) Accuracy

After early stopping the best state of the model is chosen based on the lowest validation loss. And that state is then used for the final evaluations and tests. The comparisons of the final training and testing times are presented in Table 4.6 for the binary dataset. In Table 4.7 are similar results for the multi-class dataset. All of the classification results are presented in Chapter 5. The prediction time is an average over all of the test samples when the batch size is set to 1. Despite the CNN model having the highest number of parameters, its prediction and training times are the lowest. That might be explained because of the heavier and more memory-intensive computations in the LSTM layers. Also, CNNs are usually computed more efficiently when a graphics card is used. The number

of training epochs is the biggest with the CNN-LSTM, which points to the slowest convergence. The CNN and LSTM converge relatively fast with under 10 training epochs. Since the number of dropout layers is the biggest with the CNN-LSTM, that kind of difference can be expected.

Table 4.6. *Training and testing times comparison for binary classification.*

(Binary)	CNN	LSTM	CNN-LSTM
Number of trainable parameters	88 961	38 337	21 826
Training time per epoch (s)	2	5	4
Training epochs	9	2	16
Prediction time (ms/sample)	3	15	8

Table 4.7. *Training and testing times comparison for multi-class classification.*

(Multi-class)	CNN	LSTM	CNN-LSTM
Number of trainable parameters	89 221	40 133	22 726
Training time per epoch (s)	5	9	8
Training epochs	8	6	37
Prediction time (ms/sample)	3	14	8

4.4 Model optimization

The architecture and hyperparameter tuning of the models were made manually by trial and error. The optimization workflow included starting with baseline architectures and generally known parameters. Then the next step was to make changes that optimize the previous design. The models are trained with the multi-class training dataset during the optimizations. Also, the previously mentioned early stopping condition with the validation dataset is used. The test dataset was not used during the optimizations. The first step is to make changes to the architectures by adding or removing layers and changing the number of filters or units. Then the best option is chosen by comparing the validation loss, accuracy, and number of parameters. The second step is to add or remove dropout layers with different rates and add or remove batch normalizations in different locations. The last step is to try different batch sizes, learning rates, activations, and kernel sizes. Generally, the differences between the losses and accuracies were relatively small with different trials. Often the most significant difference was in the number of parameters.

The initial values of the learning rate and batch size were also the most optimal ones. Those are mentioned in Table 4.5 above. The batch normalization was tried in the CNN optimization phase but then completely dropped. The results were significantly worse with every combination of batch normalization with or without dropout. For this reason,

batch normalization is not discussed in more detail in this thesis. Comparisons between the initial designs and final optimized designs are presented in Table 4.8. The final designs after the optimizations are the same as presented in Section 4.2 above. To avoid confusion between different architectures in different steps, the details of the initial designs are not further presented. The most important aspect is just to illustrate how the final architectures were formed and how much they improved during the optimization. It can be seen that the biggest optimization impact was in the number of parameters, as mentioned above. However, the CNN-LSTM model gained notable improvement in the loss and accuracy. With the LSTM the loss slightly increased after the optimization. On the other hand, accuracy increased and the number of parameters halved. Also, almost every time there is some variation in the loss and accuracy between each training run, so slight differences are not usually that meaningful.

Table 4.8. *Performance comparison before and after optimization with multi-class validation data.*

	CNN	**LSTM**	**CNN-LSTM**
Num parameters before optimization	618 501	88 197	31 833
Num parameters after optimization	89 221	40 133	22 726
Loss before optimization	0.26	0.68	0.37
Loss after optimization	0.24	0.69	0.23
Accuracy before optimization	91%	73%	87%
Accuracy after optimization	91%	75%	91%

The workflow for optimizing the models might not have been the most thorough. The different changes were made step by step and not crosswise. That means that variations in the later step were not tried with all of the variations in the previous step. Only the best variations were used for the next step. However, the results were quite similar with different types of architectures and parameters. So, the final results from some crosswise optimization workflow might not have been significantly different.

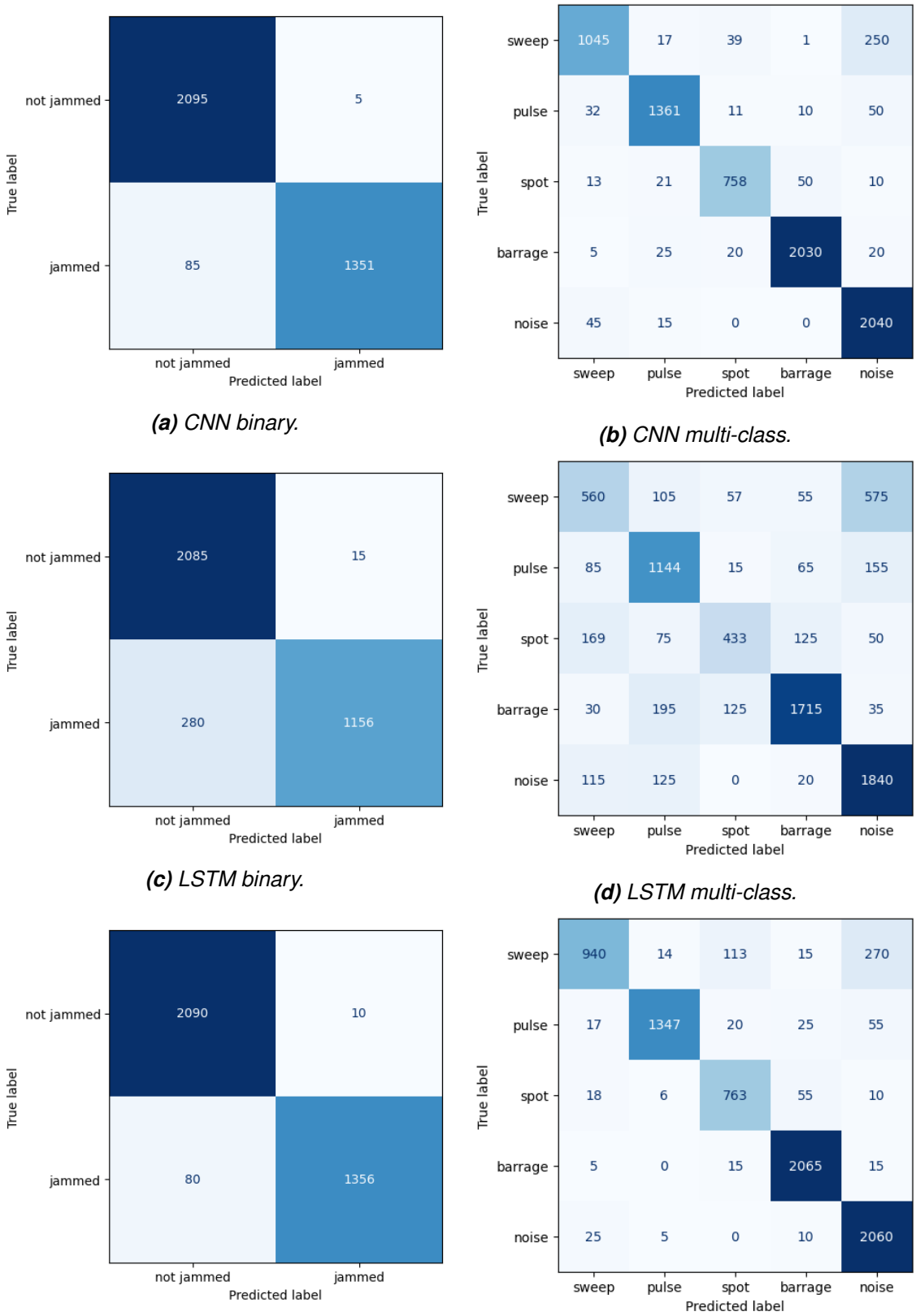
5. RESULTS AND DISCUSSION

This chapter presents and discusses the final results that are based on the metrics introduced in Section 3.4. All of the final results are evaluated with the test dataset. The test data was not used during the training or optimization steps of the models. Input for the models is always one heatmap, which contains 133 PRBs and 14 OFDM symbols.

Confusion matrices are presented in Figure 5.1. It can be seen that the results between the CNN and CNN-LSTM are quite similar. The LSTM stands out from those two models by providing notably more false classifications. For binary classification, it can be seen that the highest false classification numbers happen when the prediction is not jammed and the true label is jammed. That refers to higher precisions than recalls, but those results are presented more closely below. For multi-class classification, it can be seen that the highest false classification numbers happen when the prediction is noise but the true label is sweep. Since the sweep jammer is changing its frequency over time and some elements are also masked, the sweep heatmap might look very similar to the noise heatmap. Also, there might be scenarios where all of the jamming is masked and only noise is left, but still, the true label is denoted for some jammer. Variations in the JNR are also affecting the results, which are discussed below.

Accuracies for each class in multi-class classification over the JNR are presented in Figure 5.2. In general, the accuracies are getting better as the JNR increases, which is also expected. Jamming power in relation to noise power is growing, so it should be easier to recognize different jamming patterns from the noise. Difficulties with the sweep jammer and LSTM model are again visible. Also, it can be seen that the fastest convergences are achieved with the noise and barrage classes. Accuracies with the noise class are behaving kind of like reverse manner. The noise accuracies are around 100% with the lowest JNR points and then fluctuating a bit as the JNR increases. With the lowest JNR points the models might make a lot of predictions towards the noise class, since the jamming power is weaker than the noise power. After some JNR point this kind of bias towards the noise class might stabilize and cause some fluctuation for the noise accuracy. Because the barrage jammer has high energy, meaning wide bandwidth and constant pattern, it might be easier to recognize the barrage jammer even with the lowest JNR points. Those assumptions might explain why the noise and barrage classes are at higher accuracy levels already in the beginning. However, noise and barrage classes are also the most

frequent ones in the training and testing data, which might have a slight impact.



(a) *CNN binary.*

(b) *CNN multi-class.*

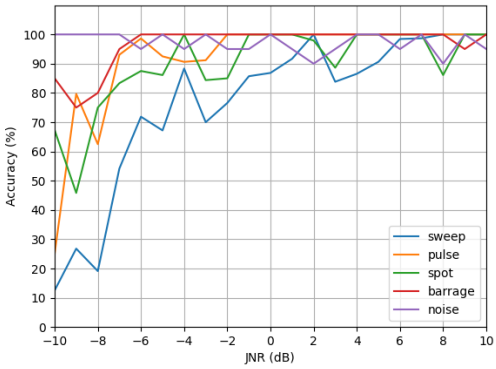
(c) *LSTM binary.*

(d) *LSTM multi-class.*

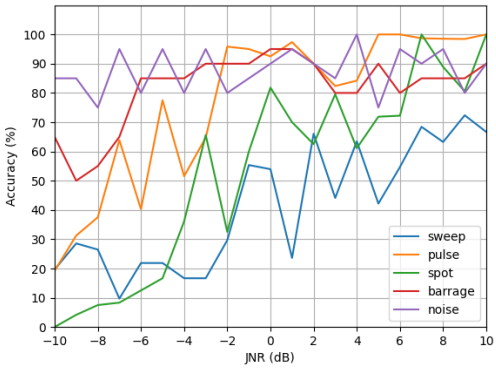
(e) *CNN-LSTM binary.*

(f) *CNN-LSTM multi-class.*

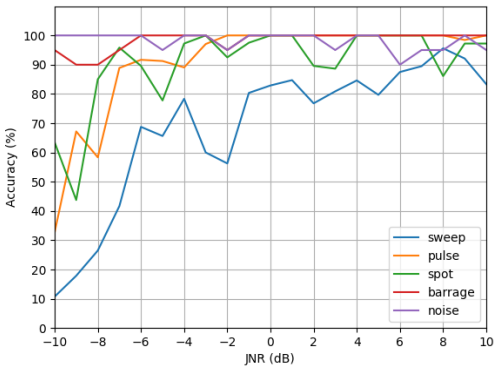
Figure 5.1. *Confusion matrices.*



(a) *CNN.*



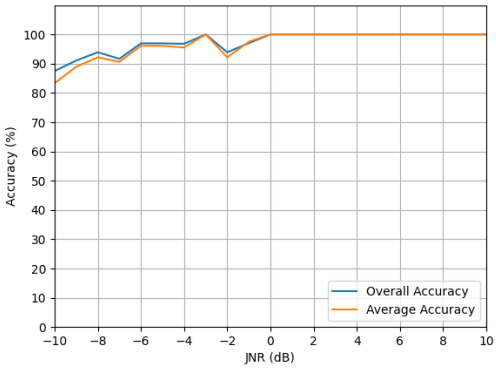
(b) *LSTM.*



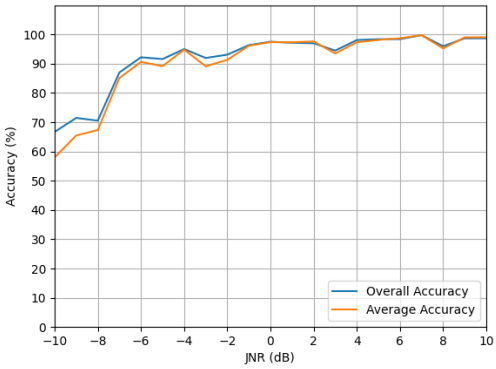
(c) *CNN-LSTM.*

Figure 5.2. *Accuracies per class over JNR.*

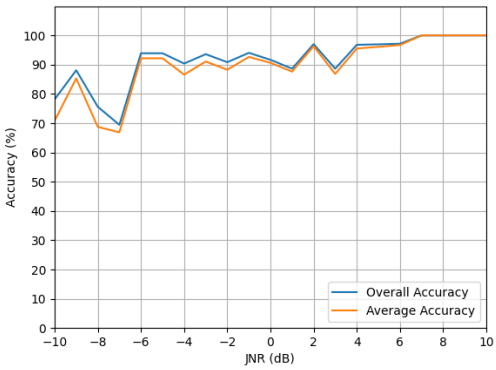
Comparison between the overall and average accuracies are presented in Figure 5.3. To recap from Chapter 3.4, overall accuracy is a fraction of correct classifications divided by the total amount of items to classify. Average accuracy is an average over the individual accuracies of each class. If the test dataset has an imbalance, performance with the most frequent classes has a greater impact on the overall accuracy. With average accuracy, every class has an equal impact despite the imbalances in the test dataset. It can be seen that differences between the overall and average accuracies are relatively small, especially with the binary classifications. With the CNN and CNN-LSTM multi-class, the biggest differences are around 10% with a couple of the lowest JNR points and then the accuracies are almost identical. With the LSTM multi-class, the biggest difference is also around 10% until -2 dB JNR and then the accuracies are in a few percent margin. Because the overall and average accuracies are quite similar and at sufficient levels, the imbalance in the test or training dataset does not have a significant impact on the results. That is also stated in Section 4.1 above.



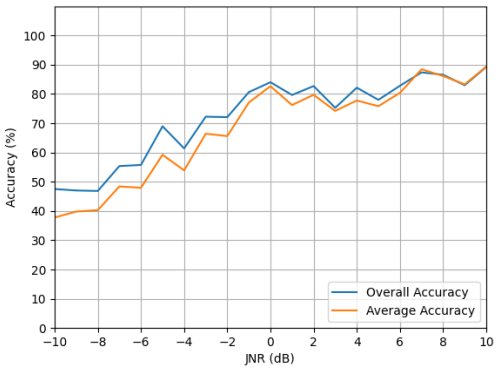
(a) *CNN binary.*



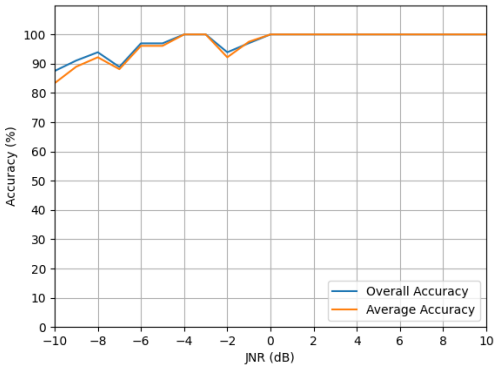
(b) *CNN multi-class.*



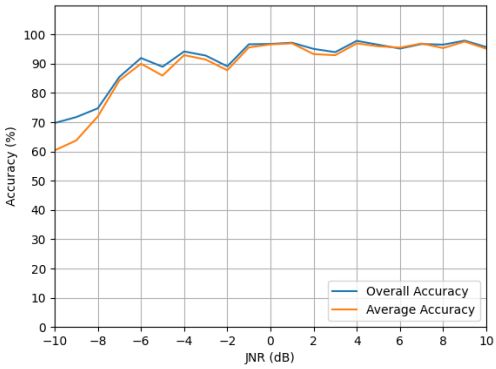
(c) *LSTM binary.*



(d) *LSTM multi-class.*



(e) *CNN-LSTM binary.*

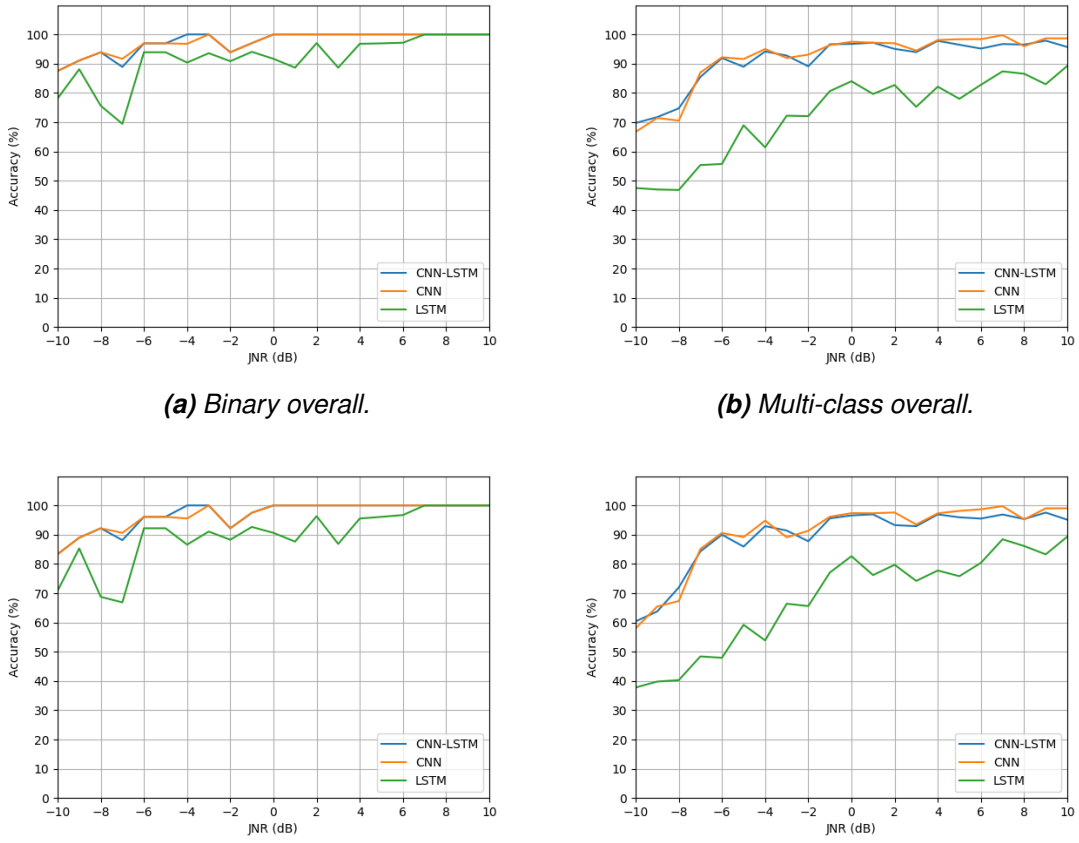


(f) *CNN-LSTM multi-class.*

Figure 5.3. *Overall and average accuracy comparison over JNR.*

All accuracies over the JNR are summarized in Figure 5.4. Again the accuracies are increasing as the JNR increases. The CNN and CNN-LSTM are providing almost identical accuracies. The underperformance of the LSTM is now even more visible when compared to those two. For binary classification, all of the models are showing relatively good results already in the beginning with the lowest JNR values. The CNN and CNN-LSTM are achieving 100% accuracies around 0 dB JNR. The LSTM achieves 100% accuracy around 7 dB JNR. For multi-class classification, the results are also at a relatively good

level already in the beginning. The CNN and CNN-LSTM are achieving 90% accuracies around -6 dB JNR and then fluctuating between 90% and 100%. The LSTM accuracies are steadily increasing and achieving 90% accuracy at the very end around 10 dB JNR. The multi-class classification results are not as high as with the binary classification, but the objective is more complex. However, the difference between the binary and multi-class classification with the CNN and CNN-LSTM is not that significant, especially when considering above 0 dB JNR. When the JNR is below 0 dB, the noise power is greater than the jamming power, which means that dealing with the noise might be a bigger problem than jamming. When the JNR is above 0 dB, it is more important to be able to classify the jammers. For binary classification with the CNN and CNN-LSTM it can be seen that the accuracies are 100% after 0 dB JNR. Similarly, for the multi-class classification with those models, accuracies are around 95% after 0 dB JNR.



(a) *Binary overall.*

(b) *Multi-class overall.*

(c) *Binary average.*

(d) *Multi-class average.*

Figure 5.4. *Summary of accuracies over JNR.*

The key metrics over all JNR points are presented in Table 5.1. For binary classification also the precisions and recalls are shown. As mentioned above, when observing the binary confusion matrices, the precisions are a bit higher than the recalls. Precisions for all of the models are almost 100%, especially with the CNN and CNN-LSTM models. Also, the recalls with those two models are almost as high as the precisions, which are

close to ideal performance when both metrics are 100%. Recall for the LSTM is notably worse than the precision, but still at a relatively good level. All in all, the CNN model provides the best results with just a little margin ahead of the CNN-LSTM. Although, the CNN-LSTM model provides the best recall and binary average accuracy values. Despite the LSTM model providing relatively good results, it can not compete against the CNN based approaches. The results with the CNN and CNN-LSTM are almost identical, so the combination of CNN and LSTM seems to be a valid option for this type of classification problem. However, as seen from Tables 4.6 and 4.7 above, the CNN model is more efficient due to the heavier computations in LSTM units. Utilizing longer time dependencies or having some sort of sequence to sequence or forecasting problem might have been more suitable for the LSTM based approaches.

Table 5.1. *Key metrics for each model.*

	CNN	**LSTM**	**CNN-LSTM**
Multi-class Overall Accuracy	**91.9%**	72.3%	91.2%
Multi-class Average Accuracy	**90.5%**	68.1%	89.4%
Binary Overall Accuracy	**97.5%**	91.7%	**97.5%**
Binary Average Accuracy	96.9%	89.9%	**97.0%**
Binary Precision	**99.6%**	98.7%	99.3%
Binary Recall	94.1%	80.5%	**94.4%**

6. CONCLUSIONS

The purpose of this thesis was first to create a jamming and interference dataset in the 5G uplink physical layer. Then the purpose was to implement and compare three different DL models in two different jamming and interference classification tasks with the generated dataset. Each model had separate instances for the two different classification tasks, but the only architectural difference between the two instances of the model was the activation of the output. Since any wireless connection can be jammed and interfered, including 5G, it is important to develop methods against those threats. The goal was to find out which of the proposed models works the best and if there is a significant difference between the two tasks. The tasks were binary and multi-class classifications. The binary classification task denotes if the input is jammed or not and the multi-class classification task recognizes the type of the jammer. To the best of found knowledge, this thesis is the first of its kind in terms of the used inputs, tasks, and DL models.

The generated dataset modeled scenarios of 5G uplink physical layer signals with and without jamming. Four different jamming models were used and noise was also considered as a class, so the dataset contained five classes. The jamming models varied their parameters for each input and each input was a time-frequency power heatmap. Each input heatmap was a 2-D grid consisting of 133 PRB wise average powers over 14 OFDM symbols. To change the jamming power in relation to the noise power, JNR was varied from -10 dB to 10 dB. To simulate the resources allocated to UEs, some of the elements in the heatmaps were set to zero with different resource allocation rates. Datasets for each of the two tasks were a bit different, to balance the occurrence of jammed and not jammed inputs. Despite the occurrence of each class was not exactly the same in the datasets, the imbalance was not alarming and did not have a significant impact on the results.

The proposed DL architectures were CNN, LSTM and CNN-LSTM. Since the data can be considered as an image like a 2-D grid with time-frequency dependencies, utilizing feature extraction from the CNN and time-dependency analysis from the LSTM could be a suitable approach. By using DL, there was no need for complicated pre-processing or manual feature selection steps. So, the model inputs could be just simple power measurements from the input signals. However, training DL algorithms can be time-consuming and in general require large amounts of training data. Also, selecting sufficient hyperparame-

ters for optimal performance can be difficult. That is why the proposed DL models were designed to be relatively small and efficient.

The CNN model was the most efficient and provided the best results. Also, there was no significant difference between the binary and multi-class classification tasks. Performance of the LSTM model was the poorest but the CNN-LSTM model provided almost identical results with the CNN model. That means that adding time-dependent processing capabilities from LSTM was not useful but not harmful either. However, due to the nature of computations in LSTM units, the LSTM or CNN-LSTM models were not as efficient as the CNN. With the CNN and CNN-LSTM binary classification accuracies were around 97% and multi-class accuracies around 90%. LSTM results were notably worse around 90% and 70% accuracies respectively. There is a notable difference between binary and multi-class classification accuracies, which can be expected since the objectives are different. Since the difference is relatively small, except with LSTM, both tasks are suitable for the CNN and CNN-LSTM. Results with the CNN and CNN-LSTM are satisfying well the expectations in both tasks, especially when considering the performances above 0 dB JNR. The biggest room for improvement is with below 0 dB JNR, but then the noise might become a greater concern than the jamming. So, the most interesting results are with above 0 dB JNR, since then the jamming power becomes greater than the noise power.

If the input heatmaps had been generated to be longer in time, LSTM might have shown more comparable performance. Especially CNN-LSTM might have been more robust than the CNN. As mentioned above, the biggest room for improvement is under 0 dB JNR, which is not actually that useful to improve. With the CNN and CNN-LSTM results were already 100 % in binary and around 95 % in multi-class classification when the JNR was above 0 dB. Also, in a practical system in 5G base station, longer input sequences would mean bigger memory buffers. So, increasing the input size might not be feasible if it is not necessary.

Overall, the models performed well in both classification tasks with the generated dataset. CNN provided the best performance in terms of results and efficiency. The worst performance was with the LSTM, although CNN-LSTM provided almost identical results with the CNN. However, the efficiency of the CNN was more robust. Also, there was no significant difference between the two classification tasks.

For future improvements, more engineering towards the data and jamming attacks would be recommended. It would be useful to have a larger training dataset with more diverse jamming models. Based on the larger dataset, current approaches could be re-evaluated and fine-tuned to be even more robust. With the CNN efficiency could be improved, for example using depth-wise separable convolutions and/or replacing fully connected layers with convolutional layers and global average pooling. Also, strided or dilated convolutions could be tried instead of max pooling. Using more advanced hyperparameter tuning

methods, like grid or random search, might provide more robust models. In future work, it would be interesting to implement DL architecture to jamming pattern extraction, meaning a semantic segmentation task, that would indicate the PRBs that are being jammed. Also, a jamming pattern forecasting network would be interesting, that would predict the future PRBs that might be jammed. Modeling smart ML-based jamming attacks would also be interesting in future work. Using LSTM with the forecasting problem might be more relevant again since that task could be more suitable for LSTM. Based on the results of this thesis, the use of LSTM should also be considered in future works. Especially when combining it with CNN.

REFERENCES

[1] Ahmadi, S. 5G NR. Academic Press, 2019. ISBN: 9780128134023.

[2] Lichtman, M., Rao, R. M., Marojevic, V., Reed, J. H. and Jover, R. P. 5G NR Jamming, Spoofing, and Sniffing: Threat Assessment and Mitigation. IEEE International Conference on Communications Workshops, 2018, pp. 1–6. DOI: https://doi.org/10.1109/ICCW.2018.8403769.

[3] Aref, M., Jayaweera, S. and Yepez, E. A Survey on Cognitive Anti-jamming Communications. IET Communications, 2020, pp. 3110–3127. DOI: https://doi.org/10.1049/iet-com.2020.0024.

[4] Lichtman, M., Czauski, T., Ha, S., David, P. and Reed, J. H. Detection and Mitigation of Uplink Control Channel Jamming in LTE. IEEE Military Communications Conference, 2014, pp. 1187–1194. DOI: 10.1109/MILCOM.2014.199.

[5] Xu, W., Trappe, W., Zhang, Y. and Wood, T. The feasibility of launching and detecting jamming attacks in wireless networks. Association for Computing Machinery, 2005, pp. 46–57. DOI: https://doi.org/10.1145/1062689.1062697.

[6] Vijayakumar, K. P., Ganeshkumar, P., Anandaraj, M., Selvaraj, K. and Sivakumar, P. Fuzzy logic–based jamming detection algorithm for cluster-based wireless sensor network. International Journal of Communication Systems, 2018. DOI: https://doi.org/10.1002/dac.3567.

[7] Xu, X., Gao, K., Zheng, X. and Zhao, T. A zero-sum game theoretic framework for jamming detection and avoidance in Wireless Sensor Networks. International Conference on Computer Science and Information Processing, 2012, pp. 265–270. DOI: https://doi.org/10.1109/CSIP.2012.6308845.

[8] Arjoune, Y., Salahdine, F., Islam, M. S., Ghribi, E. and Kaabouch, N. A Novel Jamming Attacks Detection Approach Based on Machine Learning for Wireless Communication. International Conference on Information Networking, 2020, pp. 459–464. DOI: https://doi.org/10.1109/ICOIN48656.2020.9016462.

[9] Puñal, O., Aktaş, I., Schnelke, C.-J., Abidin, G., Wehrle, K. and Gross, J. Machine learning-based jamming detection for IEEE 802.11: Design and experimental evaluation. Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2014, pp. 1–10. DOI: https://doi.org/10.1109/WoWMoM.2014.6918964.

[10] Xin, M. and Cai, Z. A feature fusion-based communication jamming recognition method. Wireless Networks, 2023. DOI: https://doi.org/10.1007/s11276-023-03272-1.

[11] Pawlak, J., Li, Y., Price, J., Wright, M., Al Shamaileh, K., Niyaz, Q. and Devabhaktuni, V. A Machine Learning Approach for Detecting and Classifying Jamming Attacks Against OFDM-Based UAVs. Association for Computing Machinery, 2021, pp. 1–6. DOI: https://doi.org/10.1145/3468218.3469049.

[12] Sedjelmaci, H. Attacks Detection Approach Based on a Reinforcement Learning Process to Secure 5G Wireless Network. IEEE International Conference on Communications Workshops, 2020, pp. 1–6. DOI: https://doi.org/10.1109/ICCWorkshops49005.2020.9145438.

[13] Shao, G., Chen, Y. and Wei, Y. Convolutional Neural Network-Based Radar Jamming Signal Classification With Sufficient and Limited Samples. IEEE Access, 2020, pp. 80588–80598. DOI: https://doi.org/10.1109/ACCESS.2020.2990629.

[14] Qu, Q., Wei, S., Liu, S., Liang, J. and Shi, J. JRNet: Jamming Recognition Networks for Radar Compound Suppression Jamming Signals. IEEE Transactions on Vehicular Technology, 2020, pp. 15035–15045. DOI: https://doi.org/10.1109/TVT.2020.3032197.

[15] Viana, J., Farkhari, H., Campos, L. M., Sebastião, P., Cercas, F., Bernardo, L. and Dinis, R. Two methods for Jamming Identification in UAV Networks using New Synthetic Dataset. IEEE Vehicular Technology Conference, 2022, pp. 1–6. DOI: https://doi.org/10.1109/VTC2022-Spring54318.2022.9860816.

[16] Cai, Y., Shi, K., Song, F., Xu, Y., Wang, X. and Luan, H. Jamming Pattern Recognition Using Spectrum Waterfall: A Deep Learning Method. IEEE International Conference on Computer and Communications, 2019, pp. 2113–2117. DOI: https://doi.org/10.1109/ICCC47050.2019.9064207.

[17] Tang, Y., Zhao, Z., Ye, X., Zheng, S. and Wang, L. Jamming Recognition Based on AC-VAEGAN. IEEE International Conference on Signal Processing, 2020, pp. 312–315. DOI: https://doi.org/10.1109/ICSP48669.2020.9320987.

[18] Gecgel, S., Goztepe, C. and Kurt, G. Jammer Detection based on Artificial Neural Networks: A Measurement Study. Association for Computing Machinery, 2019, pp. 43–48. DOI: https://doi.org/10.1145/3324921.3328788.

[19] Li, K., Jiu, B., Wang, P., Liu, H. and Shi, Y. Radar active antagonism through deep reinforcement learning: A Way to address the challenge of mainlobe jamming. Signal Processing, 2021, p. 108130. DOI: https://doi.org/10.1016/j.sigpro.2021.108130.

[20] Arulkumaran, K., Deisenroth, M. P., Brundage, M. and Bharath, A. A. A Brief Survey of Deep Reinforcement Learning. IEEE Signal Processing Magazine, 2017, pp. 26–38. DOI: https://doi.org/10.1109/MSP.2017.2743240.

[21] Liao, K., Zhao, Y., Gu, J., Zhang, Y. and Zhong, Y. Sequential Convolutional Recurrent Neural Networks for Fast Automatic Modulation Classification. IEEE Access, 2021, pp. 27182–27188. DOI: https://doi.org/10.1109/ACCESS.2021.3053427.

[22] Zhang, Z., Luo, H., Wang, C., Gan, C. and Xiang, Y. Automatic Modulation Classification Using CNN-LSTM Based Dual-Stream Structure. IEEE Transactions on Vehicular Technology, 2020, pp. 13521–13531. DOI: https://doi.org/10.1109/TVT.2020.3030018.

[23] Ali, A. H., AL-Musawi, R. S. and Al-Majdi, K. Development of CNN-LSTM Hybrid Deep Learning Network for the Joint Detection of Non-Orthogonal Multiple Access Signals in 5G Uplink Receivers. International Journal of Intelligent Engineering and Systems, 2022, p. 479. DOI: http://dx.doi.org/10.22266/ijies2022.0831.43.

[24] Zhang, W., Krunz, M. and Ditzler, G. Intelligent Jamming of Deep Neural Network Based Signal Classification for Shared Spectrum. IEEE Military Communications Conference, 2021, pp. 987–992. DOI: https://doi.org/10.1109/MILCOM52596.2021.9653072.

[25] Mueller, J. P. and Massaron, L. Deep learning for dummies. Wiley, 2019. ISBN: 1119543045.

[26] Simplifying the difference: machine learning vs deep learning. Singapore Computer Society, 2021. URL: https://www.scs.org.sg/articles/machine-learning-vs-deep-learning (visited on 01/27/2023).

[27] Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. Pearson, 2016. ISBN: 9781292153964.

[28] Du, K.-L., Leung, C.-S., Mow, W. H. and Swamy, M. Perceptron: Learning, Generalization, Model Selection, Fault Tolerance, and Role in the Deep Learning Era. Mathematics, 2022, p. 4730. DOI: https://doi.org/10.3390/math10244730.

[29] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. and Arshad, H. State-of-the-art in artificial neural network applications: A survey. Heliyon, 2018. DOI: https://doi.org/10.1016/j.heliyon.2018.e00938.

[30] Wang, H. and Raj, B. A survey: Time travel in deep learning space: An introduction to deep learning models and how deep learning models evolved from the initial ideas. arXiv preprint arXiv:1510.04781, 2015. DOI: https://doi.org/10.48550/arXiv.1510.04781.

[31] Ramsundar, B. and Zadeh, R. B. TensorFlow for Deep Learning. O'Reilly Media, 2018. ISBN: 9781491980446.

[32] Goodfellow, I., Bengio, Y. and Courville, A. Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org (visited on 01/27/2023).

[33] Cunha, B., Droz, C., Zine, A., Foulard, S. and Ichchou, M. A Review of Machine Learning Methods Applied to Structural Dynamics and Vibroacoustic. Mechanical Systems and Signal Processing, 2022. DOI: https://doi.org/10.1016/j.ymssp.2023.110535.

[34] Yu, D., Wang, H., Chen, P. and Wei, Z. Mixed Pooling for Convolutional Neural Networks. Springer, 2014, pp. 364–375. DOI: https://doi.org/10.1007/978-3-319-11740-9_34.

[35] Hewamalage, H., Bergmeir, C. and Bandara, K. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. International Journal of Forecasting, 2021, pp. 388–427. DOI: https://doi.org/10.1016/j.ijforecast.2020.06.008.

[36] Agrawal, T. Hyperparameter optimization in machine learning : make your machine learning and deep learning models more efficient. Apress, 2021. ISBN: 1-4842-6579-3.

[37] Goyal, M., Goyal, R., Reddy, P. and Lall, B. Activation Functions. Springer, 2020, pp. 1–30. DOI: https://doi.org/10.1007/978-3-030-31760-7_1.

[38] Wang, Q., Ma, Y., Zhao, K. and Tian, Y. A Comprehensive Survey of Loss Functions in Machine Learning. Annals of Data Science, 2022, pp. 187–212. DOI: https://doi.org/10.1007/s40745-020-00253-5.

[39] Ruder, S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2017. DOI: https://doi.org/10.48550/arXiv.1609.04747.

[40] Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980, 2017. DOI: https://doi.org/10.48550/arXiv.1412.6980.

[41] Sokolova, M. and Lapalme, G. A systematic analysis of performance measures for classification tasks. Information Processing & Management, 2009, pp. 427–437. DOI: https://doi.org/10.1016/j.ipm.2009.03.002.

[42] Grandini, M., Bagli, E. and Visani, G. Metrics for Multi-Class Classification: an Overview. arXiv preprint arXiv:2008.05756, 2020. DOI: https://doi.org/10.48550/arXiv.2008.05756.

[43] Li, Y. G. and Stuber, G. L. Orthogonal Frequency Division Multiplexing for Wireless Communications. Springer, 2006. DOI: https://doi.org/10.1007/0-387-30235-2.

[44] Eng, J. S. and Kocot, C. Chapter 14 - VCSEL-Based Data Links. Academic Press, 2013, pp. 501–554. DOI: https://doi.org/10.1016/B978-0-12-396958-3.00014-7.

[45] Kim, Y.-H., Ju, H., Jeong, C. B. and Lee, M.-S. Performance comparison of DTX detection schemes for 5G NR PUCCH. International Conference on Information and Communication Technology Convergence, 2020, pp. 1391–1394. DOI: https://doi.org/10.1109/ICTC49870.2020.9289538.

[46] Grover, K., Lim, A. and Yang, Q. Jamming and anti–jamming techniques in wireless networks: a survey. International Journal of Ad Hoc and Ubiquitous Computing, 2014, pp. 197–215. DOI: https://doi.org/10.1504/IJAHUC.2014.066419.

[47] Johnson, J. M. and Khoshgoftaar, T. M. Survey on deep learning with class imbalance. Journal of Big Data, 2019, pp. 1–54. DOI: https://doi.org/10.1186/s40537-019-0192-5.