Domain-independent Activity Recognition using WiFi CSI Data

Žinys, Augustinas

Award date:
2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Interconnected Resource-aware Intelligent Systems

Domain-independent Activity Recognition using WiFi CSI Data

Master Thesis

Augustinas Zinys

Supervisors:
prof. dr. Nirvana Meratnia
Bram van Berlo, MSc

Abstract

Throughout the thesis project, device free learning-based WiFi CSI activity recognition has been analyzed. Based on existing literature analysis, it was noticed that methods tend to show performance degradation in presence of domain change issue and, thus, researchers in the field propose learning-based models, with complex WiFi CSI data pre-processing. In this thesis we analyse the impact of pre-processing module, CNN feature extractor and adversarial domain adaptation on WiFi activity recognition performance in presence of domain change. As a result we propose (i) a simplified data pre-processing module, (ii) a fine-tuned Convolutional Neural Network (CNN), which outperforms Widar3.0 [52] in presence of domain change of subjects and rooms, when both using the simplified CSI data pre-processing and, (iii) a new adversarial domain adaptation model, which outperforms the (ii).

Acknowledgement

It has been a almost a year since we started working together, and it was my pleasure to collaborate and get full support from my supervisors Nirvana and Bram. They always used to steer my way out of a lot of "unknowns" and show me the right direction. I sincerely want to say thank you for your patience and full theoretical and technical support throughout entire thesis project.

Contents

Acronyms

BMI Body Mass Index. 26

CNN Convolutional Neural Network. iii, 6, 7, 23, 33, 34, 35, 36, 37, 38, 39, 40, 41, 45, 46, 48

CSI Channel State Information. iii, vii, 1, 2, 3, 4, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 32, 34, 36, 40, 41, 42, 47, 48

DWT Discrete Wavelet Transform. 22, 29, 30, 31

FFT Fast Fourier Transform. 22

GAN Generative Adversarial Network. 8

IFFT Inverse Fast Fourier Transform. 22

LeakyReLU Leaky Rectified Linear Unit. 34, 35

OFDM Orthogonal Frequency Division Multiplexing. 2

PCA Principal Component Analysis. 22

ReLU Rectified Linear Unit. 12

VAE Variational Auto-Encoder. 27, 31

Chapter 1

Introduction

With the accelerating development of new sensing and communication technologies, monitoring human activities in everyday life has become relevant over the last decade in various fields such as surveillance, entertainment, and healthcare. In particular, sensing technologies are nowadays used to monitor and secure buildings, to track physical activities, such as running or walking, or to recognize human motion and gestures in virtual reality games. A such, there are number of different applications, in which various sensing technologies are required. This indicates that human-computer interaction has become an inevitable part of human lives.

Sensing technologies in the field can be categorised into two sub-categories: device-based and device-free. While device-based sensing refers to a situation in which sensors are attached to human body to measure and monitor an activity, the device-free sensing refers to situations in which not the human body, but the environment in which a human is present in is monitored.

Although most of device-based sensing systems became quite popular, in some situations it is impractical and cumbersome to wear them all the time. For instance, wearable watches may be taken off before sleep, or at industrial (production) environments, where wearable watches are forbidden due to safety reasons and inconvenience. Additionally, wearable-based solutions may not continuously monitor the activity accurately or correctly due to the fact that users may forget to wear them. This is relevant especially in surveillance and security field, when continuous monitoring is required.

In order to overcome the limitations of the device-based sensing, device-free sensing such as visual-based sensing (cameras) has been considered. Although this technology is quite popular as computer vision algorithms (object detection/ recognition) are advancing quite rapidly, it only operates in scenarios, in which a subject is in line-of sight and no occluding obstacles are in the view. Additionally, it requires robust and continuous lighting conditions, as a subject may not be visible throughout the entire day. Moreover, visual-based sensing devices are intrusive as they impact privacy of an individual. Therefore in order to overcome all of these limitations, device-free solutions, using radio signals such as WiFi, are considered, which will be discussed more in Section 1.1. In particular IEEE 802.11 protocol contains channel state information CSI, which characterizes how well wireless signals propagate from a transmitter to a receiver at certain carrier frequency [28]. CSI contains carrier signal amplitude and phase. Recently, it is commonly used for fine-grained activity recognition in combination with data-driven, learning-based models [25][22][6][52][29].

Over the last years, one of the popular learning-based methods - deep learning - has shown great success in various real world applications, for instance in computer vision and natural language understanding [10][15], and WiFi-based activity recognition [21][48]. However, research has shown that performance of these learning-based approaches in the context of WiFi-based activity recognition significantly degrades due to change of various factors. For instance, research has shown

that performance of WiFi-based activity recognition systems are impacted by the change of the environment in which the activity is performed [21][52][56], change of orientation with respect to the sensing device [52], CSI data quality [28], different physical properties of human subjects or slight difference in movement patterns of a user [6][29]. Even the time of the day may have a big impact as electromagnetic waves are impacted differently during the day and night in office or home environments [43][6]. Therefore, the focus of this thesis is on addressing the performance degradation issue discussed above, which is referred to as "domain change".

Typically, the domain change problem can be addressed by collecting more data and learning the data distribution that covers multiple factors impacting the performance. However, activity recognition data collection is a very expensive and time consuming task as there are too many known and unknown factors impacting the performance. In particular, each new environment setup has multiple paths from the transmitter to receiver [57]. Therefore, a learning-based system trained once on one particular environment, activity, or human subject at a specific time may not be sufficiently robust and consistent against change. This indicates the need of a robust recognition models, which are capable of performance well independent of the factors that a recognition system is exposed to.

1.1 Channel State Information

As already mentioned in the previous section, CSI characterizes signal propagation from a transmitter to a receiver at a certain carrier frequency. Data is transmitted at a certain frequency band (commonly $2.4GHz$), which is divided into N number of channels. Each channel has its own carrier frequency (central frequency), which is split by OFDM [40] into a number of sub-carrier frequencies and used for sending modulated signal. Thus, usually multiple sub-carriers are used for multiple data streams to be transmitted over one frequency band. This allows more efficient and faster data transmission compared to using only a single sub-carrier. As it can be seen in Figure 1.1, CSI is a three dimensional matrix $H^{N\times M\times K}$, with N denoting the receiving antennas, M transmitting antennas, and K number of sub-carriers. This matrix represents amplitude attenuation and phase shift of multi-path channels and is equivalent to a digital image with spatial resolution of $N \times M$ and K color channels [28]. Each entry in H matrix represents the channel frequency response as shown in Equation 1.1 and can be represented by a complex number.

$$H(f,t) = \sum_{n}^{N} a_n(t) e^{-j2\pi f \tau_n(t)} \tag{1.1}$$

, where $a_i(t)$ is the amplitude attenuation factor, $\tau_n(t)$ is the propagation delay, and f is the carrier frequency [44] [28].

For each sub-carrier, the WiFi channel is modeled by $y = Hx + n$, where y is the received signal, x is the transmitted signal, H the CSI matrix, and n is the noise vector [28]. As received and transmitted signals are known, CSI matrix can be estimated by solving Equation 1.2.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} h_{11} & \cdots & h_{1T} \\ h_{21} & \cdots & h_{2T} \\ \vdots & \ddots & \vdots \\ h_{R1} & \cdots & h_{RT} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_n \end{bmatrix} \tag{1.2}$$

Finally, as depicted in Figure 1.1, CSI matrices are filled packet by packet, forming a fourth dimension. This provides additional information over time with a series of CSI samples. One may note that this has an analogy with a video stream, which is a series of images. It provides additional temporal information to capture the action taking place. Therefore, some standard video or image processing methods in combination with learning-based models can be applied for extracting beneficial information from the CSI data.

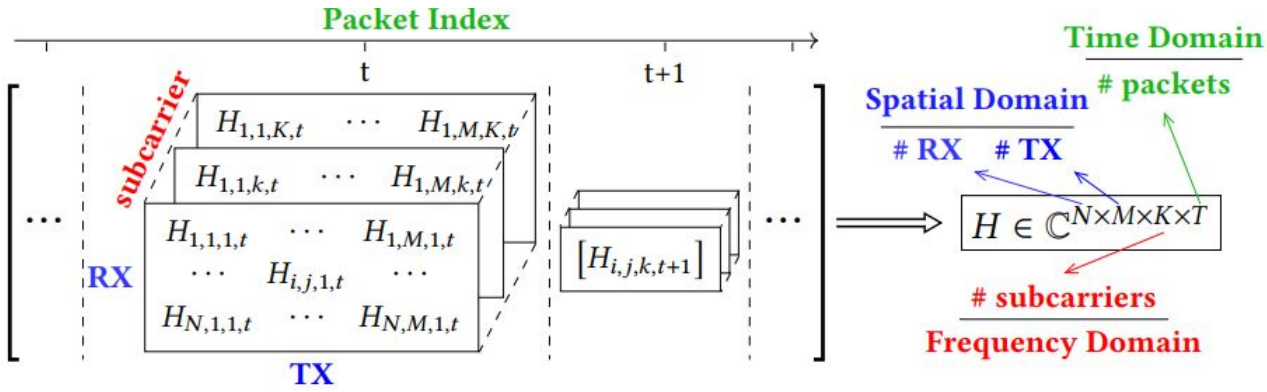


Figure 1.1: Four dimensional tensor representing a time series of CSI matrices, taken from [28]

1.2 Problem Statement

In order to formalize the "domain change" problem, we present it using clear mathematical notations. First of all, a WiFi-based activity recognition system operates in domain $\mathcal{D}(a_1, a_2, ..., a_n)$, which contains n number of domain attributes a_n that impact performance either negatively or positively.

In general, for a given domain, denoted by $\mathcal{D}$, an activity recognition task, denoted by $\mathcal{T}$, can be written as a set $\{Y, P(Y|X)\}$, where P is a function of conditional probabilistic model, Y is a label space and $X \in \mathcal{D}$ is the input data space. Using supervised learning techniques, $P(Y|X)$ is learnt from the labeled data $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in Y$.

Let us assume that source domain, denoted by $\mathcal{D}^s$, is used for training, target domain, denoted by $\mathcal{D}^t$, is used for testing, and a recognition task is defined as $\mathcal{T}^s = \{Y^s, P(Y^s|X^s)\}$ and $\mathcal{T}^t = \{Y^t, P(Y^t|X^t)\}$. In normal cases, i.e., if $\mathcal{D}^s = \mathcal{D}^t$ and $\mathcal{T}^s = \mathcal{T}^t$ [6][55][45], the learning-based model of activity recognition there will have no performance degradation. In case of domain change, however, the tasks do not change (, which means $\mathcal{T}^s = \mathcal{T}^t$), but source and target domains are no longer the same (, which means $\mathcal{D}^s \neq \mathcal{D}^t$) [55], resulting in activity recognition performance degradation.

The domain shift problem can further be categorized into two categories: (i) homogeneous, where input data space of domain attributes a_n are the same (, which means $X^s = X^t$), but the data distribution is not (, which means $P(X^s)^s \neq P(X^t)^t$) and (ii) heterogeneous, where input data space is different $X^s \neq X^t$ [55][9]. In the former case there is an assumption that domains differ only in marginal distributions. Therefore, domains can be adapted by correcting the sample selection bias. The latter case, however, is more challenging, as input data space of a_n is available from source domain, but it is represented in a different way than that of the target [9].

In order to relate the general domain change problem to the context of WiFi CSI-based activity recognition, both homogeneous and heterogeneous cases will be discussed with the focus on major body activities such as various hand gestures or body movement/exercises, as described in Section 4.2 further explored in this thesis.

Regarding the homogeneous case, marginal probability distributions data space are different $(P(X^s)^s \neq P(X^t)^t)$. For instance, they are different if recognition system is trained to classify major hand movements in an/a environment/room, where electromagnetic wave interference is lower and less frequent (suburbs) than in the target domain (city). Moreover, probability distributions may differ, for example when a group of people (source domain) performs specific body activities less often than another group of people (target domain). The performance of a learning model may degrade in both examples as the marginal probability distributions between the source and target domains (environments and people respectively) are different. Regarding the

heterogeneous case, the input data space in source domain is different compared to the target domain ($X^s \neq X^t$). For instance, assuming a learning model is trained on one group of people - female (source domain) with different physical properties a_n than the other group of people - male (target domain), then the way how movements of the activities are performed would be different. Consequently, this would cause a domain change problem, as input data space of a_n (physical body properties) between group of female and male are not same. Additionally, physical properties may be different regionally based on the average human height or any other physical property that is relevant to a one specific geographical region or human race, etc.

1.3 Research Questions

To address domain change problem in the context of activity recognition using WiFi CSI data, and based on the literature analysis and gaps identified in Chapter 3, we further define the following research question:

- To what extent can convolutional deep learning methods reduce performance degradation caused by domain change? By performance we refer to accuracy of activity recognition.

  To answer this question, we define the following sub-questions, which correspond to each of the following chapters and linked together as illustrated in Section 1.5:

  1. What is the impact of data pre-processing on the performance of convolutional deep learning architecture in presence of the domain change? (Chapter 5)
  2. What is the impact of different objective function parameters, feature extraction parameters, and dropout regularization on performance of the convolutional deep learning model in presence of domain change? (Chapter 6)
  3. What is the effect of adversarial domain adaptation on performance of the convolutional deep learning model in presence of domain change? (Chapter 7)

1.4 Contributions

The main contributions of these thesis are:

- A simplified set of CSI pre-processing steps for convolutional neural network to function for WiFi CSI data.
- A convolutional neural network model with combination of triplet loss, which outperforms Widar3.0 [52], when both using simplified set of pre-processing steps, in presence of domain change.
- A new adversarial model for domain adaptation with UNet architecture and simplified set of CSI pre-processing steps, which improves performance of the model mentioned in the second bullet point and surpasses Widar3.0 in presence of domain change.

1.5 Thesis Outline

Figure 1.2 represents the thesis outline. Beginning with the introduction (Chapter 1), motivation and research questions are discussed, which were formulated based on the outcome of the literature review (Chapter 3). Then background information (Chapter 2) is presented, which explains concepts and techniques used in Chapters 3, 4, 5, 6, and 7. Backed up by Chapters 2 and 3, our system overview (Chapter 4) is introduced to illustrate and discuss major components of WiFi activity recognition system pipeline. Our analysis and solutions to questions regarding pre-processing and feature extraction are presented in Chapter 5 and Chapter 6, respectively. Finally, based on the outcome of the Chapters 5 and 6, our approach towards adversarial domain adaptation is presented in Chapter 7. Chapter 8 presents a summary of our contributions and highlights future research directions.

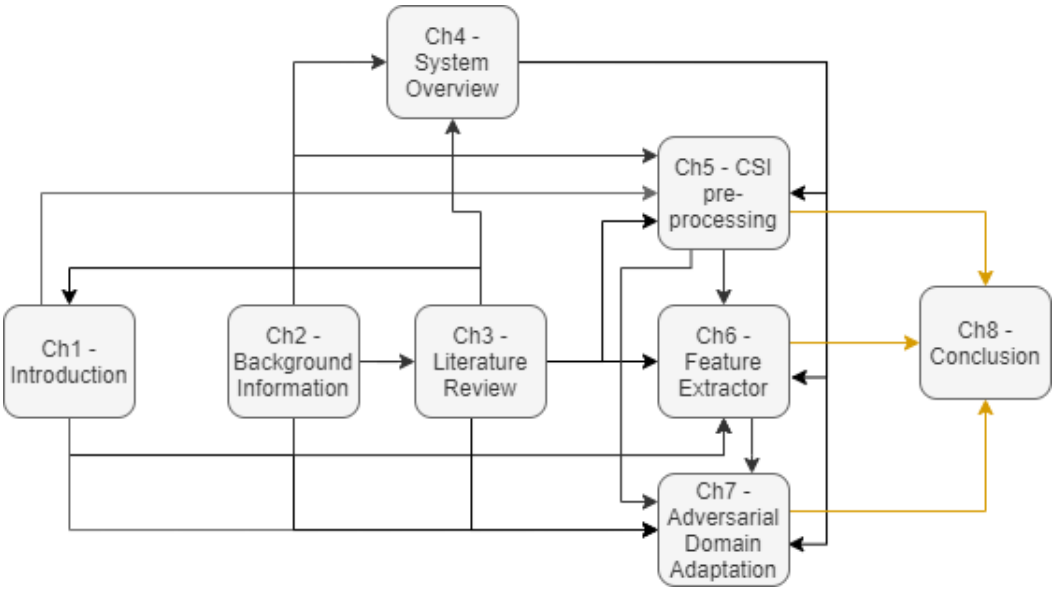


Figure 1.2: Thesis outline

Chapter 2

Background Information

In this chapter, we present the main background information related to deep learning, which will be used throughout the thesis. Beginning with a high level overview, deep learning is a part of Artificial Intelligence (AI) field [2]. As it can be seen in Figure 2.1, it belongs to a brain-inspired machine learning family, which is called "Neural Networks". This type of algorithm consists of multi-stack of layers, which contain number of neurons with weighted connections, resulting in a mapping function $y = f(x; \theta)$, where x is the input data, y ground truth labels, and θ network parameters (neuron connection weights), which are learnt throughout the learning process [15].

Different neural network architectures, with different configurations of layers and objective functions exist. Therefore, a high level overview of some of these architectures will be discussed in Section 2.1, followed by definitions of different layers and activation functions in Section 2.2 and Section 2.3, respectively. Finally, common objective functions of neural networks will be discussed in Section 2.4.

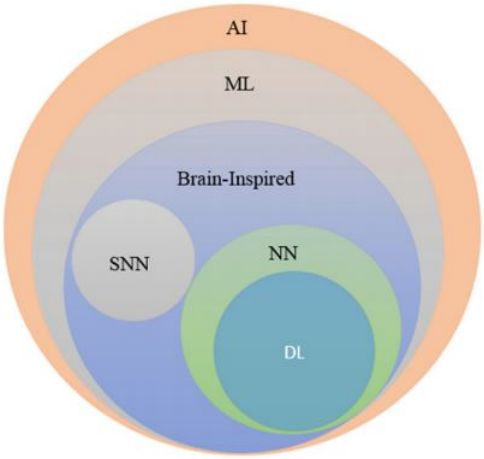


Figure 2.1: Artificial Intelligence taxonomy, taken from [2]

2.1 Deep Learning Models

2.1.1 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) is a type of artificial neural network, which consists of a stack of convolutional layers (as described in Section 2.2.2). As an example, one of the first introduced CNNs, called "LeNet-5" [27] can be seen in Figure 2.2. This network consists of 5 convolutional

layers stacked one after another, followed by "densely connected layers" (refer to Section 2.2.1) for classifying 10 hand writing digits (grayscale images of size 32x32). The architecture is inspired by visual cortex, where convolutional layer kernels correspond to different receptors, which allow automatically extract features from an input data [27].

Moreover, this type of network provides flexibility for a wide range of design choices and applications. In fact, throughout the history of deep learning, high number of different CNN architectures were proposed, with high variety of model topologies, different activation functions, or layer combinations (such as pooling as described in 2.2.3) in various fields such as computer vision 2.3 or WiFi activity recognition [6][52][50].

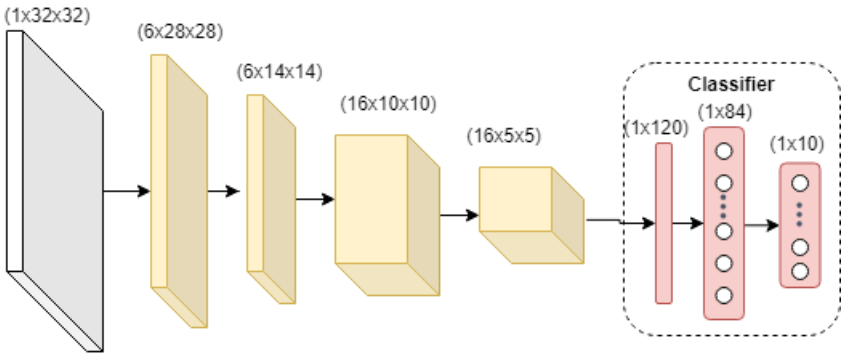


Figure 2.2: An example of a CNN network (so called LeNet-5) [27]

2.1.2 Auto-Encoders

Auto-encoder is a type of artificial neural architecture, which leverages the idea of reconstructing the input data as closely as possible, by encoding the data into lower dimensional space. As it can be seen in Figure 2.3, it mainly consists of two networks: encoder and decoder. While encoder $f(x)$ learns to encode the data x into representation h, resulting in $h = f(x)$, the decoder produces the reconstruction $r = g(h)$ [15]. Thus, if a model is constrained, such that it can only approximately copy the input data to output, then it commonly learns useful properties that help to generalize training data [15].

Many different auto-encoders have been designed for different applications. Based on a survey [5], there exist different types of auto-encoders such as Regularized Auto-Encoders (de-noising, sparse or contractive auto-encoders) or Variational Auto-Encoders [24], which can be used for generating synthetic data, clustering or pre-training data in unsupervised manner for classification tasks [5].

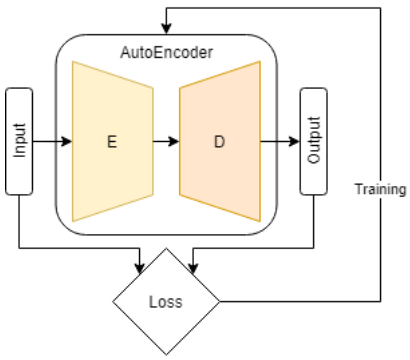


Figure 2.3: Standard auto-encoder architecture

2.1.3 Generative Adversarial Networks (GAN)

Generative Adversarial Network (GAN) [16] is a deep learning model, based on a game scenario, when one network competes against the other.

As it can be seen in Figure 2.4, a GAN consists of two networks: (i) the generator $G(z)$, which learns the representation of the input data x given the prior input noise z, and (ii) the discriminator $D(x)$, which outputs a single probability whether the input data sample x is from the original data distribution rather than from the generator G.

Both networks are trained through an iterative procedure. D is trained to maximize the probability of assigning the correct label to both original training samples and samples from G. G is trained to minimize $log(1 - D(G(z)))$ [16]. The final objective loss function can be defined as in the equation 2.1.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{2.1}$$

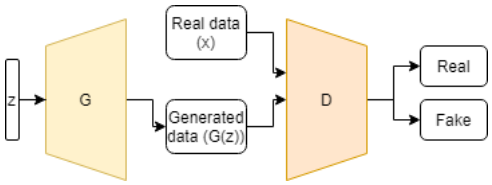


Figure 2.4: Generative Adversarial Network architecture

2.2 Layers

This section describes basic building blocks, called "layers" of deep neural network, which will be used throughout the thesis.

2.2.1 Dense Layer

A "dense" layer (sometimes referred to as "linear" layer) is the basis of a artificial neural network. It consists of a stack of densely connected computational units, called "neurons" depicted in Figure 2.5 [30]. As it can be seen, each neuron gets an input signal x, which is then passed to a neuron through synaptic weight w. After that, a sum operation of the weighted signal is performed, followed by a non-linear activation function, which is also sometimes referred to as "squashing function" (to be discussed more in the Section 2.3). The main purpose of activation function is to limit the range of an output signal [30].

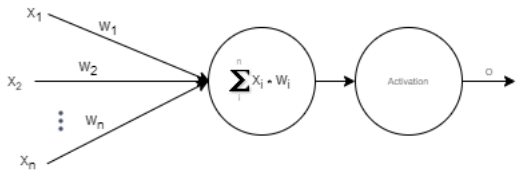


Figure 2.5: An example of an artificial neuron [30]

The densely connected layer can be constructed as depicted in Figure 2.6. Each input x is connected to each neuron and can then be passed to another layer for further processing.

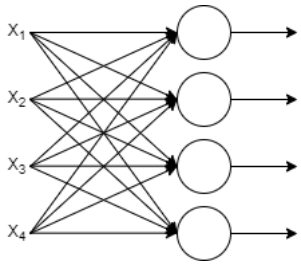


Figure 2.6: Representation of a densely connected layer

2.2.2 Convolutional Layer

Convolutional layer is another type of neural network layer, which leverages the idea of the mathematical operation, called convolution [15]. One motivation behind this layer is that it improves a simple "Dense" layer. While "dense" layer (discussed in Section 2.2.1) is quite memory expensive, as every neuron unit is connected to every input, convolutional layer reuses a particular set of weights. An example of a convolutional layer is shown in Figure 2.7, with a 2-dimensional data. This operation contains a set of weights, commonly referred to as "kernel", which is convolved through the input data, resulting to an output matrix R. By performing this operation weights are re-used all over the input data and can be represented with a mathematical equation 2.2, where I is the input data, K - kernel (weights) and R is the resulting matrix [15]. Similar to densely connected layers (2.2.1), the output of convolution layers is passed to an activation function, which will be discussed in Section 2.3.

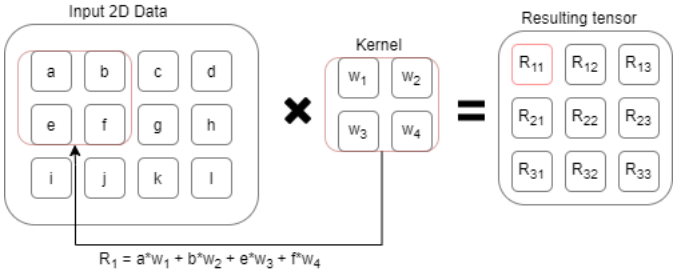
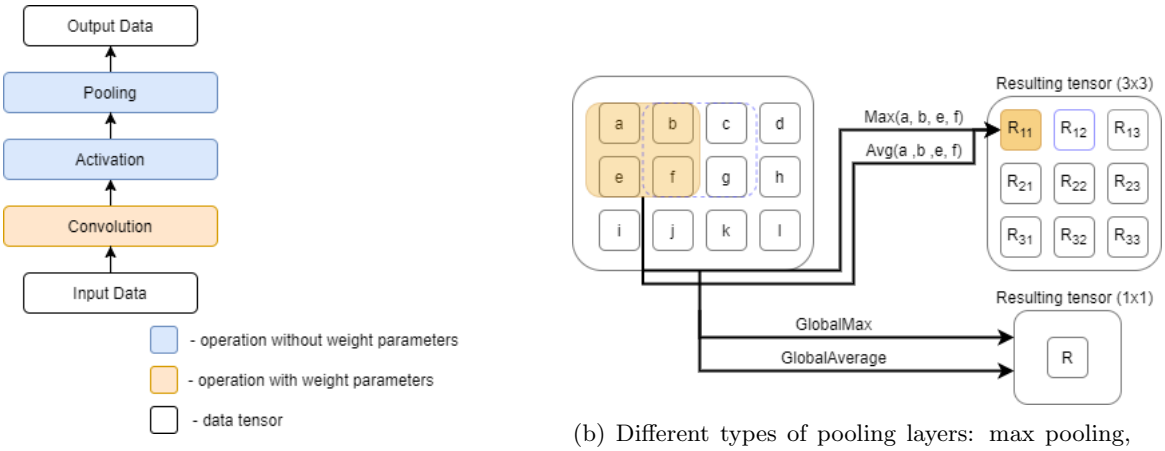


Figure 2.7: An example of a convolution operation

$$R(i,j) = (K * I)(i,j) = \sum_m \sum_n I(m,n) * K(i-m, j-n) \tag{2.2}$$

2.2.3 Pooling Layer

As represented in Figure 2.8a, pooling operation is commonly used together with a convolutional layer described in Section 2.2.2. The main purpose of a pooling layer is to replace the output of the convolutional layer at certain locations with a summary statistics of the nearby outputs [15]. For instance, as depicted in Figure 2.8b, pooling utilizes a sliding window to extract different types of statistics such as maximum or average. Based on the book [15], motivation behind this operation is to make the output tensor approximately invariant to small changes of the input. Therefore, this operation can be beneficial to extract certain features and to know whether a particular feature is present.



(a) Convolutional layer combination with the pooling layer.

(b) Different types of pooling layers: max pooling, average pooling, global max pooling and global average pooling.

Figure 2.8: An example of a pooling layer diagram

2.2.4 Batch Normalization Layer

Batch normalization layer is another operation, which is commonly used in deep neural networks. Its main functionality is to normalize the outputs of intermediate network layers within a batch of data [19]. This operation improves generalization ability, training stability as well as optimization efficiency [18] of the deep learning model. The mathematical definition of this operation can be found in Equation 2.3, where ϵ is a small number to avoid numerical instabilities, $u = \frac{1}{m}\sum_{i=1}^{m} x^i$ is the mean value, and $\sigma = \frac{1}{m}\sum_{i=1}^{m}(x^i - u)^2$ is the variance. Finally, γ and β are learnt scaling parameters throughout the training process [18].

$$\hat{x}^i = \gamma \frac{x^i - u}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.3}$$

2.2.5 Transpose Convolutional Layer

Transpose convolution layer (a.k.a. deconvolution) is an up-sampling technique, which transforms the data in an opposite way of normal convolution [47], as described in Section 2.2.2. An example of this operation is shown in Figure 2.9. Firstly, the original data size is doubled from $2x2$ size to $4x4$, by filling the missing data with zeroes. Then the expanded data is zero-padded and convolved (see Section 2.2.2) with a $2x2$ kernel marked as red, resulting in a data matrix R (size $4x4$).

Similar to convolution networks, transpose convolution has a few hyper-parameters, which have to be set before training. In particular, parameters such as kernel size, kernel stride, and padding can be varied depending on the application.

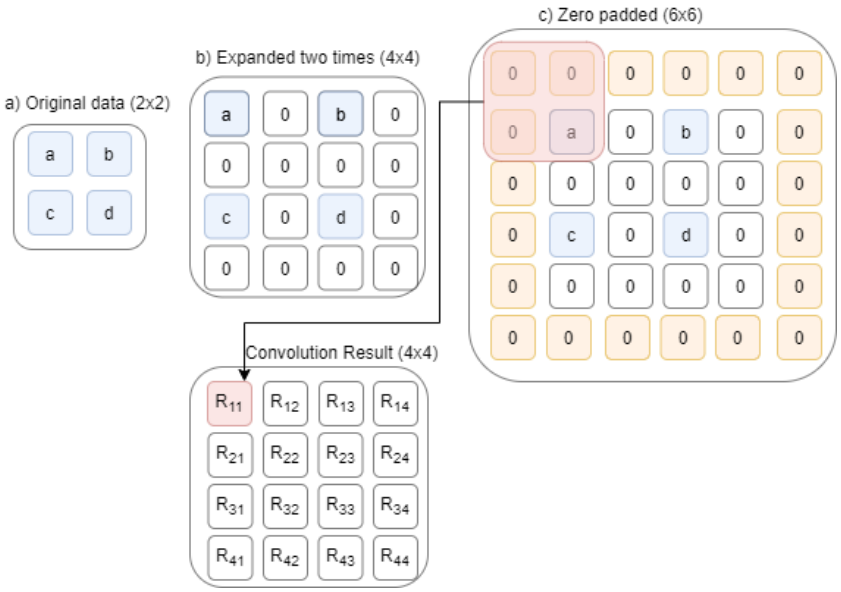


Figure 2.9: An example of a transpose convolution operation

2.2.6 Dropout Layer

Dropout layer is a regularization layer used during training a deep learning model. Commonly, deep learning models are sufficiently large, with high number of connections that could almost perfectly learn the mapping from input to output (especially when training set is limited), leading to an issue called "overfitting" [17]. Thus as depicted in Figure 2.10, dropout regularization drops certain number of neurons, with probability p in the artificial neural network during the training time in order to reduce the network capacity [17].

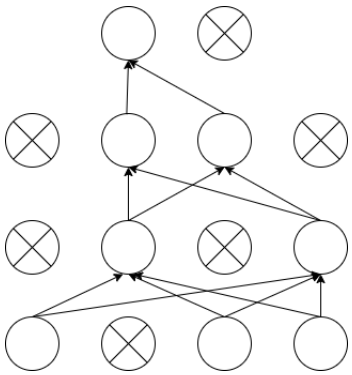


Figure 2.10: Dropout regularization

2.2.7 Embedding Layer

Embedding layer is an operation originally introduced for converting text data to continuous vector representation, which is learnt throughout the training process [31]. As it can be seen in Figure 2.11, it takes one-hot encoded word, which is multiplied by its learnt weight matrix, resulting in a weight vector (embedding vector) of that particular word.

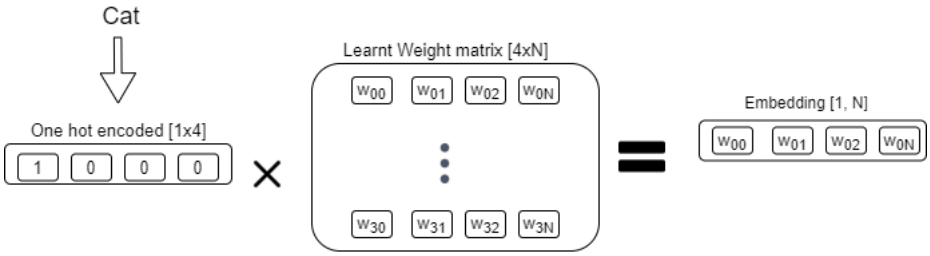


Figure 2.11: An example of an embedding Operation

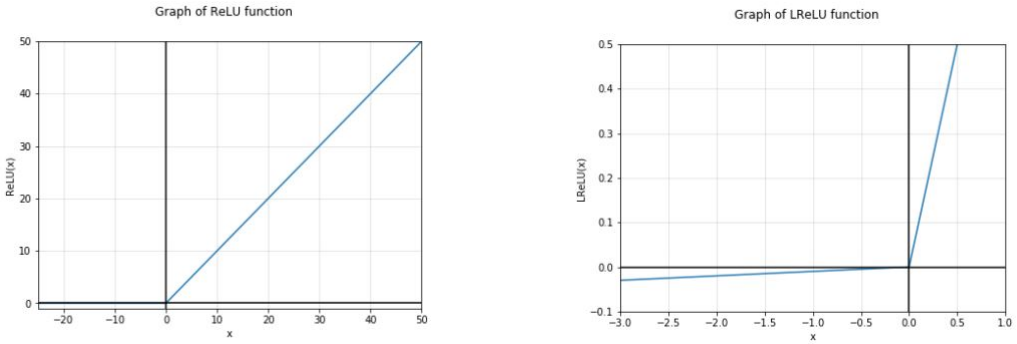
2.3 Activation Functions

As already mentioned in previous sections, an activation function (a.k.a transfer function) is a non-linear function, applied to the output of each layer in artificial neural network [8]. In this section, we present some common activation functions used in this thesis.

2.3.1 Rectifier Linear Unit (ReLU)

Rectified Linear Unit (ReLU) (see Figure 2.12 for an example) is commonly used in intermediate layers of an artificial neural network. It was first proposed by [33] and can be defined as $f(x) = \max 0, x$, where x is input data. While ReLU does not change the positive input values, negative input values are pushed to 0. Although it is simple and computationally cheap, its gradient for all negatives input values is 0, introducing an issue commonly known as dying ReLU, which shuts down neurons during the training process. Therefore, a modified version, called Leaky (a.k.a Parametric) Rectified Linear Unit [8] was introduced, and can be seen in Figure 2.12b. This improved version of ReLU, is defined as Equation 2.4 and its main advantage is the constant parameter a, which controls the gradient of negative input values.

$$f(x) = \begin{cases} ax & \text{for } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{2.4}$$



(a) Rectified linear unit activation function (b) Leaky rectifier linear unit activation functions

Figure 2.12: An example of rectified linear unit proposed by [8]

2.3.2 Softmax

Softmax activation is another popular function used in artificial neural networks. It is commonly used for multi-class classification tasks, as its main purpose is to compute probability distribution from an input vector x [34]. Thus this function usually is used after the last layer of artificial

neural network to transform output to a probability distribution. Its mathematical definition can be found in Equation 2.5 [34].

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.5}$$

2.3.3 Hyperbolic Tangent

Hyperbolic tangent (Tanh) is another activation function, defined in Equation 2.6, which maps wide range of input data x to a range of $[-1, 1]$ (Figure 2.13) [8]. It is commonly used for natural language processing [34] or computer vision task, using generative adversarial networks [46].

$$f(x_i) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{2.6}$$

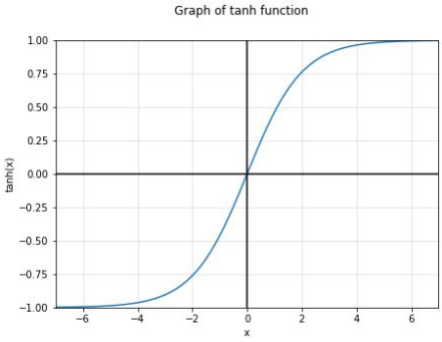


Figure 2.13: Hyperbolic Tangent activation function, taken from [8]

2.4 Objective Loss Functions

While a deep learning model can be defined as a mapping function $y = f(x; \theta)$, where y is output, x - is input data, and θ model weight parameters, an objective loss function $J(\theta)$ has to be defined in order to optimize training the model $f(x; \theta)$ with weight parameters θ for a given task $T = \{y, f(x; \theta)\}$, where model has to learn targets y [15]. In general terms, supervised learning objective function J is defined as in Equation 2.7, where it is computed over an average of the training set and L is the loss function per single training sample [15].

$$J(\theta) = \mathbb{E}_{(x,y)\sim p_{data}} L(f(x; \theta), y) \tag{2.7}$$

Different loss functions exist and we discuss several of them, used in this thesis.

2.4.1 Cross-Entropy and KL Divergence

Assuming that a deep learning task needs to know how different two probability distributions are, a certain measure is needed. For instance, a classification task may require such a quantity in order to compare the predicted probability distribution of categories of the given target probability distribution (labels).

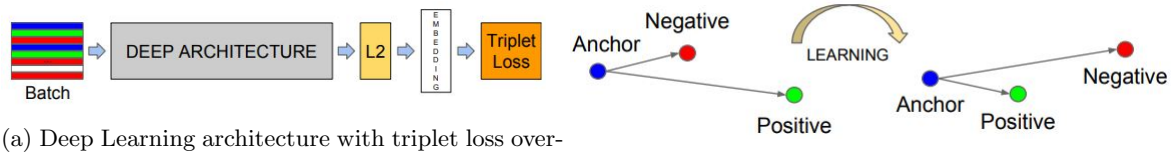
Let us beginning with definition of entropy. It is a measure of uncertainty in a probability distribution, defined as $H(x) = -\mathbb{E}_{x\sim P} P(x)$ [15]. Then, for two given probability distributions $P(x)$ and $Q(x)$ over same random variable, a measure of how different these distributions are can be defined (see Equation 2.8). This measure is referred to as Kullback-Leibler (KL) divergence [15].

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)] \tag{2.8}$$

Cross-entropy, denoted by $H(P,Q)$, is a quantity closely related to KL divergence and can be defined as $H(P,Q) = H(P) + D_{KL}(P||Q)$ [15]. Based on definition of $H(P)$, $H(P,Q)$ can be further reduced to $H(P,Q) = -\mathbb{E}_{x\sim P} \log Q(x)$ [15].

2.4.2 Triplet Loss

Authors of the paper [41] were one of the first people, who coined the idea of triplet loss. It was originally used for human face verification in scalable and efficient way. As can be seen in Figure 2.14, triplet loss leverages the idea to embed input sample x into d-dimensional Euclidean space, such that a person face x^a (anchor) would be within close distance (L_2) in the embedding space with a similar person face x^p (positive), however within large L_2 distance with a differently looking face x^n (negative).



(a) Deep Learning architecture with triplet loss overview.

(b) Triplet loss visualization.

Figure 2.14: Triplet los, taken from [41]

The mathematical definition of such a loss function can be found in Equation 2.9, where f represents embedding, N is number of pairs and α is the loss margin enforced between negative and positive pairs [41].

$$L = \sum_{i}^{N} \left[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha\right] \tag{2.9}$$

Chapter 3

Related work

Different methods have been developed in the past for domain independent activity recognition using WiFi CSI data. Those methods can be categorized in two main categories, i.e., (i) model-based, and (ii) learning-based [43]. Since most of the learning-based approaches have shown huge success in the recent years, model-based approaches will not be reviewed in this thesis.

Learning-based approaches are data-driven and can be trained to perform an activity recognition task using a machine learning algorithm. These learning-based approaches may be categorized into three sub-categories, i.e., supervised, semi-supervised, and unsupervised learning [2]. Based on this categorization, we present in the following sections an overview of various deep learning methods, designed for activity recognition using WiFi CSI data.

3.1 Supervised Learning Approaches

Various supervised learning algorithms have been successfully applied for activity recognition using WiFi CSI data. Supervised learning algorithms are based on training with labeled data sets, where each input data x has an associated label y and a model can learn to extract relevant features to map each input x to a corresponding output label y. This algorithm training setting is quite common in the literature.

Beginning with the year of 2018, one of the baseline papers, called SignFi [29], was published. Authors designed a deep learning model based on convolutional neural networks (CNN) to recognize hand sign gestures. In total, 5 people volunteered and CSI traces of 276 different sign gestures in two different environments, i.e., home and lab, were collected. Authors showed that performance of classical machine learning methods, such as k-Nearest Neighbour, degrades with the increasing number of gesture categories. Deep learning CNN model was able to achieve 98.01%, 98.91%, and 94.81% accuracy for the lab, home, and combined lab and home data, respectively. Although their model showed high performance in each environment, based on leave-one-subject-out cross validation they showed their model degrades on a new user that the model has not been trained on. The authors did not show how environmental independent (leave-one-environment-out) their model was as they did not train the model on one environment and tested on another.

An activity anomaly detection model, called FallDeFi [36], came out the same year. Authors focused on detecting the anomaly, i.e., a rare event, when a person is falling in any environment. They invited three volunteers to perform falling and non-falling events to collect CSI traces in five different environments. They first did feature selection and then used Support Vector Machine (SVM) [7] to perform a binary classification. FallDeFi's model performed quite well on data from different environments, achieving approximately 80% accuracy when trained on one environment and tested on another. However, their results showed performance degradation (approximately 70% overall accuracy), when the model was trained with one person in one environment and tested

with another person in a different environment, indicating that future work could be focused on addressing this issue.

Authors of [6] experimented with a supervised transfer deep learning [55] method to address scalability and stability of the human activity recognition using WiFi CSI data. They invited nine volunteers to collect CSI traces of body movements over multiple days. They used deep learning CNNs to solve a multi-class classification problem. Their model showed high performance on classification of individual movements (the same person for training and testing), however, results on cross-participant was not as robust. Based on their conclusion, the overall accuracy of the CNN model decreased for different participants and over different days. This indicates that raw CSI is not transferable as it is sensitive to different characteristics of participants and fluctuations in the WiFi CSI signal over days [6].

Another paper [50] proposed quite a different supervised deep learning architecture for WiFi CSI-based activity recognition, as depicted in Figure 3.1. The goal of the paper was to address the domain change problem in the field of hand gesture recognition. According to the authors, their model worked well in new environments with minimal tuning and few additional training samples. In order to achieve this, they proposed to use Siamese neural network architecture [25], where two identical twin networks were used with the shared weights and two input samples were fed to each of the two networks. They had two main contributions to the Siamese architecture. First they utilized convolution neural network to extract spatial information and coupled it with the recurrent neural network BILSTM (Bi-directional Long-Short Term Memory) to capture temporal information. Secondly, they proposed to use a pairwise loss function with the combination of Mk-MMD (multiple kernel maximum mean discrepancies) [50]. While this pairwise loss function maximizes the distance between the samples of different classes and minimizes the distance between the gesture samples of the same class, MMD aids better domain adaptation. In order to test their architecture, they invited 10 volunteers to collect data of 6 major hand movements in two different meeting rooms (large and small). Regarding the environmental robustness, they achieved 65.3% and 75.3% accuracy for large and small rooms, respectively, when 100% data in one environment and 20% of data in the new environment were used for training and other for testing. Additionally, they formed 2 groups of people: A and B. When trained on group A and tested on B and vice-versa, they got 85.7% and 83.5% accuracy, respectively. Overall, authors concluded that their architecture improved existing methods on very small sample conditions, however its performance degraded with fine grained finger gestures.

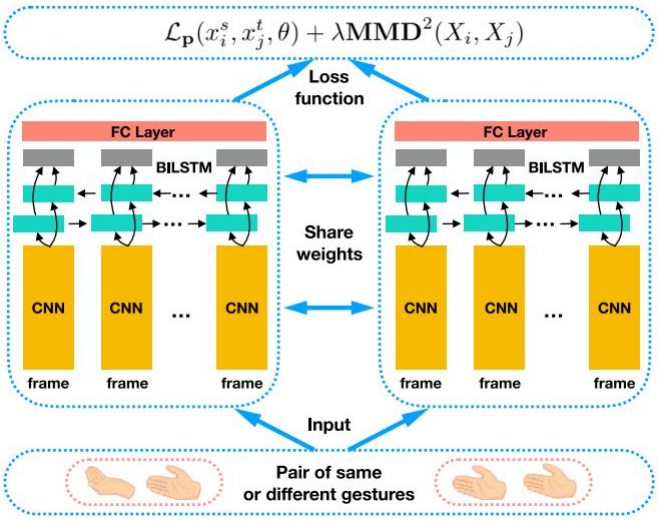


Figure 3.1: Siamese architecture, taken from [50]

Authors of [52] introduced a robust supervised cross-domain recognition system. They collected one of the largest open-source CSI datasets, which will be described in more details in Section 4.2 and will be used throughout this thesis. It consists of seventeen subjects performing various hand gestures in three different environments. The dataset was used to study domain independent features, which was found to be the body velocity profile (BVP). Authors developed a pre-processing module, which transforms CSI data to signal power distribution over velocity components in the body coordinate system. Regarding the classification, a hybrid spatial-temporal deep learning model was designed. As it can be seen in Figure 3.2, their model takes the BVP as an input and outputs prediction of the gesture. Based on the experiments, their entire recognition system achieves quite robust results, showing environment, person, location and orientation independency, with an average accuracy exceeding approximately 80%. The major contributing of this paper is use of the BVP as an input feature. It was shown that without this pre-processing module the accuracy drops to approximately 40%. Therefore, this model as well as their rich collected dataset (described in Section 4.2) will be used as a benchmark for the experiments of this thesis (in Chapters 5, 6, 7) to study domain independent learning.

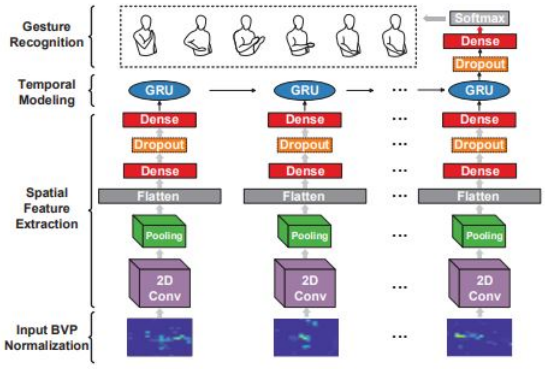


Figure 3.2: The spatial-temporal Widar3.0 deep neural network, taken from [52]

3.2 Un/Semi - Supervised Learning Approaches

Unsupervised and semi-supervised learning have shown huge success, outperforming most of the supervised learning activity recognition models using WiFi CSI data. Both semi-supervised and unsupervised learning are beneficial when labeled data is not available or is too expensive or time consuming to label the data. In case of unsupervised learning, algorithms are trained on a dataset without labels and the goal is to learn useful properties or features of that particular dataset. However, most of practical scenarios are based on the semi-supervised learning, when only part of the dataset has labels. Then the goal is to make use of unlabeled data, enhancing the model robustness, scalability, and performance.

Authors of [22] achieved model robustness in different environments by utilizing the unsupervised domain adaptation [13]. They employed labeled and unlabeled parts of the dataset to train three neural networks for feature extraction, activity recognition, and domain discriminator, as represented in Figure 3.3. The learning was set through adversarial learning, in which feature extractor extracts features that minimizes the performance of the domain discriminator, at the same time enhancing the performance of the activity recognizer. In order to test performance of their model, they collected human body activity CSI traces of 40 subject-room pairs, corresponding to 40 different domains [22]. Based on the experiments, their model kept improving with an increasing number of domains and achieved an accuracy of over 70% for 18 domains used for testing and 22 domains for training. Although the proposed model uses both labeled and unlabeled data, each environment had to be labeled to get a clear distinction. This may lead to scalability problem, when the system is going to be deployed and used in real life and as such needs to be

addressed in the future.

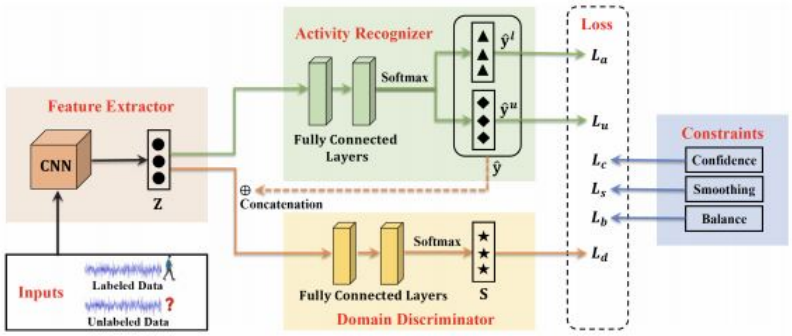


Figure 3.3: Proposed architecture of [22]

Authors of [56] introduced WiADG - WiFi-enabled device-free adaptive gesture recognition scheme. The goal of their work was to identify human gestures accurately under different environmental dynamics using adversarial domain adaptation (ADA) [13]. Their main system architecture was divided into 3 steps, as illustrated in Figure 3.4. In the first step, authors assumed that only source domain training data is available, so they trained a source encoder and source classifier to get high performance only in one specific source environment. Then, in the second step, an unsupervised adversarial domain adaptation technique was applied, by utilizing trained source encoder, new target encoder, and an environment discriminator. Inspired by GAN (to be discussed in Section 3.2.1), where the discriminator distinguishes between the fake and real images, the WiADG discriminator separates source and target domains. The main learning objective was set such that target encoder forced the discriminator to classify unlabeled target samples as source samples, while discriminator seeks the opposite. Finally, in the last step, trained target encoder and source classifier were used to identify gestures in the target domain. In order to test performance, authors collected CSI data of various hand gestures in two different environments, i.e., office and conference room. They showed that with adversarial domain adaptation, their system improved the overall accuracy at least by 30% for cross-domain inference, achieving 83,3% TPR (true positive rate), when target domain was the office room and 66,6% TPR, when target was the conference room. Although, adversarial domain adaptation improved robustness of their model at different environments, authors did no experiment with different users.

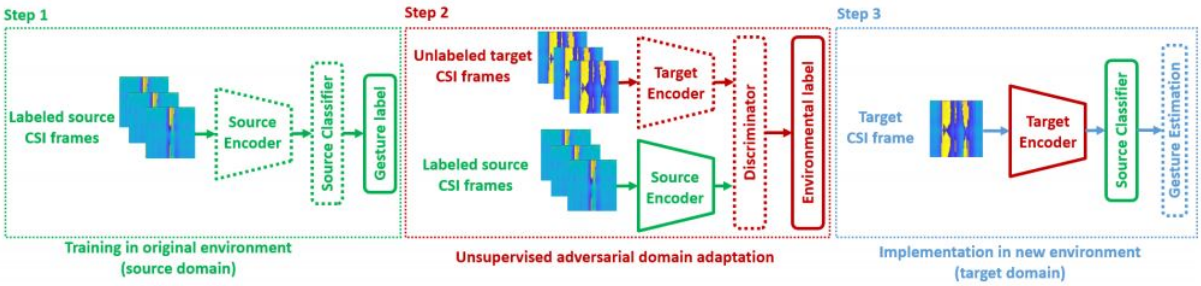


Figure 3.4: Architecture of WiADG, taken from [56]

3.2.1 GAN Based Architectures

A quite common unsupervised/semi-supervised algorithm used in recent years is Generative Adversarial Network (GAN) [16]. Different variations of GANs, such as Cycle GAN [54], Wasserstein

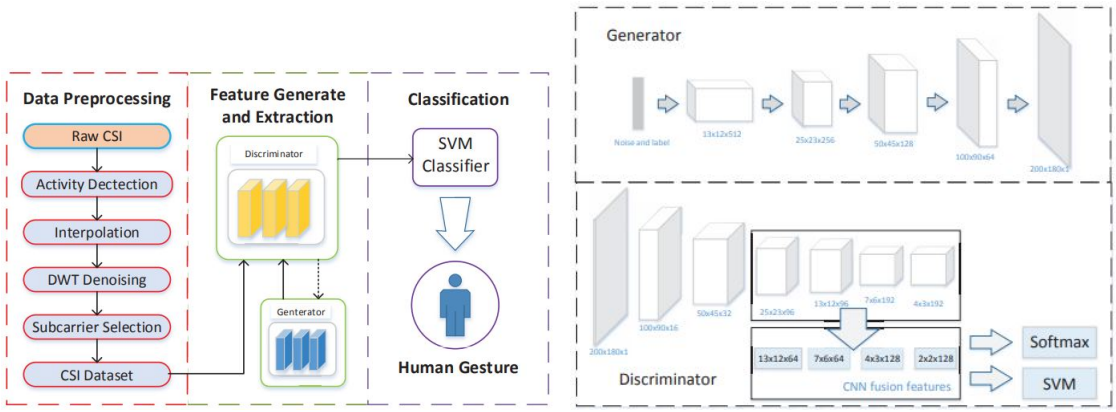
GAN [4], Deep Convolutional GAN [38], are used for domain transfer in unsupervised or semi-supervised settings. However, all of them are based on the same fundamental concept, i.e., a game theory concept that generator network competes against an adversary. This concept is described in Section 2.1.3.

A recent GAN-based semi-supervised architecture was proposed in [48]. The authors addressed performance degradation of leave-one-subject-out, when data of one person was used for training (source domain) and another for testing (target domain). This is a common and realistic scenario as the end user would always be the one who is left out. The main contribution of the paper is that they used two generators in their GAN architecture. One generator is a vanilla GAN [16] to generate fake samples, and another generator is a complement GAN, trained using the "CycleGAN" [54], which generates the source domain samples in accordance with data of the left-out user (target domain). In order to test their performance, authors used SignFi [29] and Falldefi [36] datasets and achieved an accuracy of over 84% and 86% for SignFi and FalldeFi datasets, respectively for leave-one-subject-out validation. Although, they had quite a low performance degradation when performing leave-one-out-subject validation, they did not test performance of their model on data from different environments. Finally, they faced some stability issues during training.

Another recent successful method based on GAN is WiGAN [21]. The authors showed that their proposed model, as illustrated in Figure 3.5b, is environmental and user independent. There are three major contributions of the model itself compared with the original DCGAN model. First, the authors combined structure of DCGAN with the main characteristics of Conditional GAN [32]. This combination resulted in altered generator G input, which consists of the prior latent z together with the sample label $y_i \in \mathcal{Y}$. This improved sample generation, solving the small sample problem. Second contribution was the discriminator network D, for which they proposed a convolutional neural network module to fuse the features maps of the last four layers of D. Those compressed feature maps were then further processed by the "softmax" activation function to output the probability distribution of each category, which is used in GAN training. Taking the feature maps of the last four layers instead of only the last one, provided an effective way to learn by choosing an optimal set of features. Third contribution was related to the fact that instead of using the discriminator D directly as a classifier, the authors proposed to use Support Vector Machine (SVM) [42]. Based on the experiments with Widar3.0 dataset [53], they showed that SVM outperforms CNN by 7% under small sample conditions. The trade-off, however, was that performance was degraded while testing on a large number of activity categories. The authors did not further explore different classification methods based on deep learning and left it for future studies.

Furthermore, the authors of [21] proposed to add data pre-processing module for training deep learning model (see Figure 3.5a), whose main purpose was to convert raw CSI data into the sanitized CSI amplitude. This module consists of several data pre-processing steps, such as activity detection, interpolation, Discrete Wavelet Transform and sub-carrier selection. Several experiments were performed to check the impact of data pre-processing module on overall model performance. It was showed that without it, 10% to 20% performance degradation was experienced. Although, this indicates that raw data already contains unnecessary noisy components and it should be pre-processed, authors did not further explore which pre-processing steps are the most optimal ones and did not experiment with other methods available in the literature.

Based on the experiments with SignFi [29] and Widar 3.0 [53] [52] datasets , their model achieved state of the art results. They used 10-fold cross-validation to test environmental independence on the Widar3.0 dataset and their model achieved an average recognition accuracy of 83%, 71%, and 75.2% for 'classroom', 'hall', and 'office', respectively. Regarding the SignFi dataset, the same test was done and an accuracy of 98% and 90% for home and lab environments, respectively, was achieved. Additionally, leave-one-out subject validation was performed on the Widar3.0 dataset. Their experiments showed that the model can reach 91% accuracy, when tested on left-out subject and trained on all remaining subjects.



(a) WiGAN System (b) WiGAN model architecture

Figure 3.5: WiGAN system and model architecture, taken from [21]

3.3 Research Gaps

After a thorough literature review, some research gaps in the field of learning-based activity recognition models using WiFi CSI data can be identified, which are presented in the list below.

- Current commonly used learning-based activity recognition models using WiFi CSI data commonly require high amount of data.
- Learning-based activity recognition models using WiFi CSI data found in the literature tend to be robust only in one particular domain (or limited number of domain combinations), such as change of subject or room [22][48].
- Current learning-based activity recognition models for WiFi CSI data require a complex and computationally expensive data pre-processing module. It is not clear, which pre-processing methods are the most optimal for a given CSI traces of certain activities [52][21].
- Un/semi-supervised activity recognition models for WiFi CSI data, such as GAN, tend to be unstable and require careful parameter tuning [48].
- Some learning based models show performance degradation in presence of larger number of activity categories [21].

Chapter 4

System Overview

In this chapter, we first explain standard pipeline of learning-based activity recognition models using WiFi CSI data and its common building blocks. We then present its adaptation in deep leaning-based approaches, as well as our modification made in this thesis. Additionally, we explain the dataset used for our experiments.

4.1 Standard Pipeline of Activity Recognition using WiFi CSI Data

The first phase of a learning-based activity recognition system using WiFi CSI data is data collection. Figure 4.1 illustrates a data collection setup, in which a human subject performs activities in a room in front of a transmitter and a receiver. The WiFi signals sent by the transmitter are reflected by the floor, walls, ceiling, objects in the room, as well as body of the human subject. These reflections together with the reflection resulted from the activity that the person performs, are then received at the receiver. The received signal variations are stored as raw CSI traces.

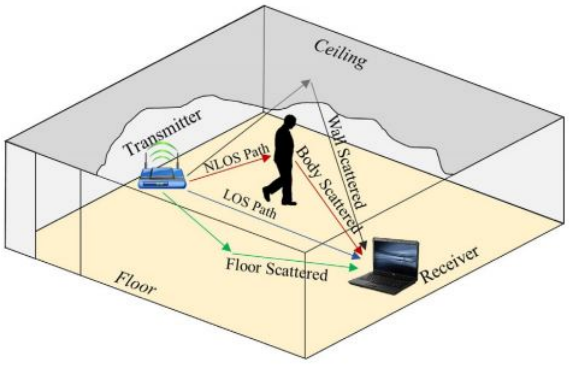


Figure 4.1: An overview of the data collection setup from the user point of view, taken from [43] [11]

The second phase is generally known as (activity/gesture/context) recognition phase, which has multiple steps, as illustrated in Figure 4.2. It starts commonly with signal pre-processing and feature extraction designed for achieving better recognition accuracy.

Based on existing survey papers, e.g., [43] [11], data pre-processing step involves signal segmentation, de-noising, dimensionality reduction, or any other techniques contributing to a more robust and accurate recognition by improving data quality or its transformation. Initially, idle or absent

signal is removed by the process of the signal segmentation. After that, it is quite common to remove high frequency noise from the raw CSI [43]. Various techniques, such as butterworth low pas filter [51], Hampel filters [37], or DWT [21], have been utilized for this purpose.

After that, the feature extraction step is commonly applied on the de-noised signal. Some common data transformation methods, such as Fast Fourier Transform (FFT), Inverse Fast Fourier Transform (IFFT) or Principal Component Analysis (PCA)[49],[43] have been used for this purpose.

Consequently, a set of extracted features are either manually or (semi-)automatically selected in such a way that recognition accuracy can be improved, after which they are passed to a classification algorithm (the third step in the standard pipeline), such as Support Vector Machine, K-Nearest Neighbors [43].

Overall, this entire recognition pipeline includes quite a long list of methods, beginning with the CSI data pre-processing, feature extraction and ending with the type of the classification algorithms that can be used. Each manual selection impacts performance of the recognition system and it is quite complex to find the optimal combinations. Therefore, in recent years it has become quite common to use learning-based methods such as deep learning artificial neural networks, to replace some steps in the recognition pipeline shown in Figure 4.2.

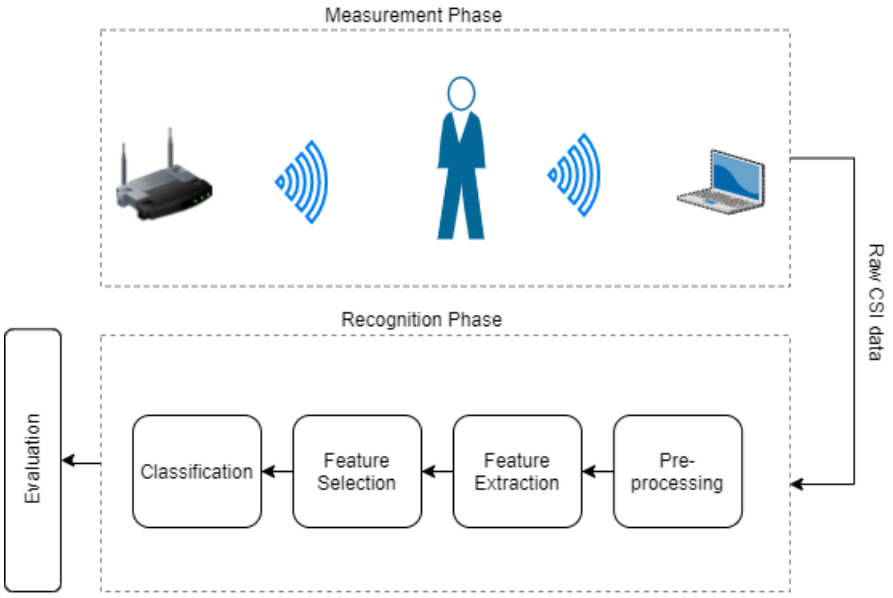


Figure 4.2: Standard pipeline of activity recognition using WiFi CSI data

4.1.1 Adaptation of Standard Activity Recognition Pipeline by Deep Learning Approaches

Recent deep learning-based activity recognition using WiFi CSI data have adapted the standard pipeline presented above by replacing some parts with artificial neural networks. For instance, Figure 4.3 depicts Widar3.0 [52] deep learning-based activity recognition pipeline, which is used as a reference for comparison in this thesis. It can be seen from the diagram that feature extraction, feature selection, and classification are replaced by a deep learning model. As mentioned in Chapter 3, authors of the paper have used spatial and temporal deep learning extraction techniques, which automatically learn optimal features based on a pre-defined loss function and ground truth labels. Their data pre-processing phase consists of several stages. Firstly, noise and phase offset are removed in the pre-processing phase. Then using pre-processed CSI data, body velocity profile (BVP) is generated. Finally, BVP is normalized for the input of deep learning model.

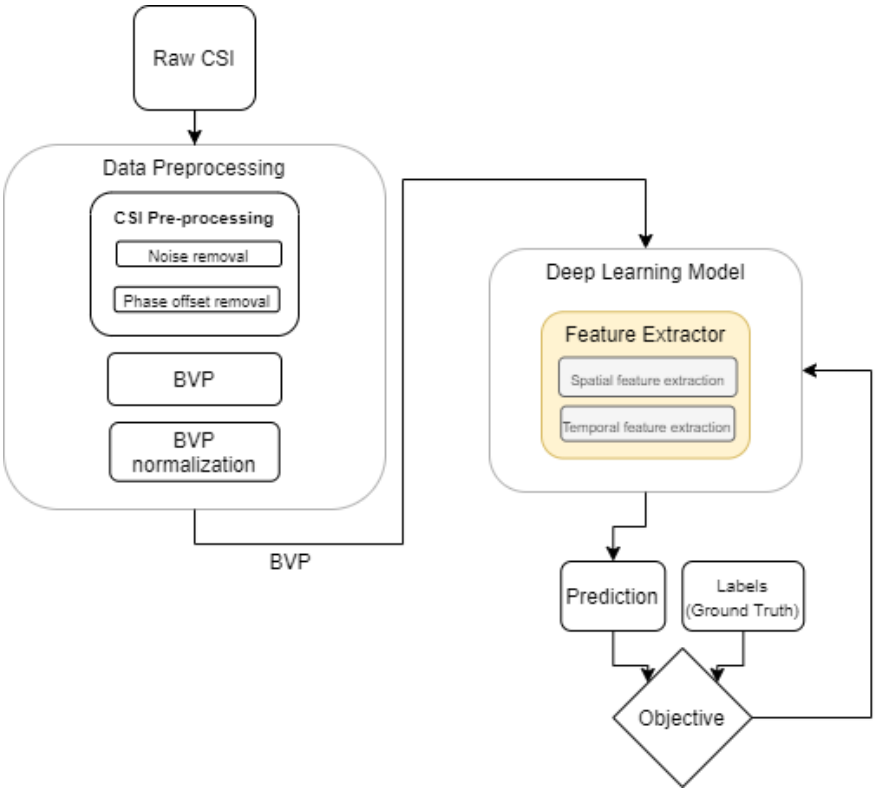


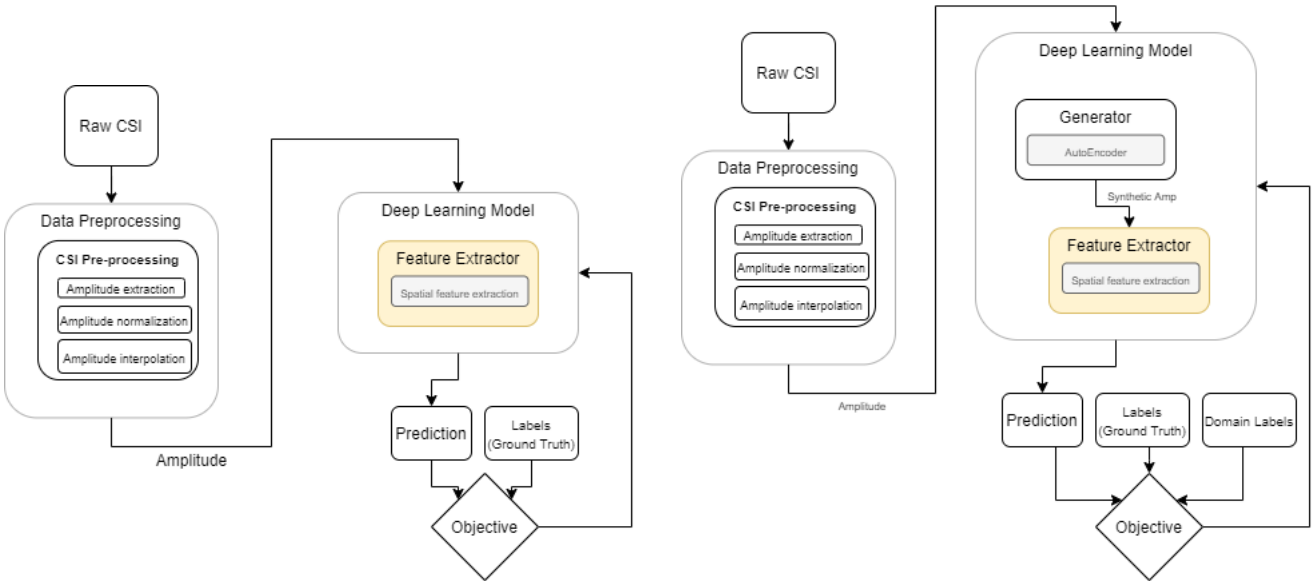
Figure 4.3: Pipeline of Widar3.0 paper's activity recognition using WiFi CSI data, taken from [52]

In this thesis, we further modify the Widar3.0 paper's activity recognition pipeline by simplifying the data pre-processing phase and proposing a new domain adaptation step. Our activity recognition pipeline is illustrated in Figure 4.4, which will be discussed more in the following Chapters 5, 6 and 7.

Our data pre-processing begins with amplitude data extraction from raw CSI, followed by amplitude normalization and interpolation. The former is required due to the fact that neural networks work with standardized data for more stable training process [18]. Regarding the latter, amplitude sample constant shape is required for the input of artificial neural network. Whilst there are methods to construct a CNN model with varying input shape [15][27], this is out of the scope of this thesis and is left for future work.

Our deep learning model, as it can be seen in Figure 4.4a, is different than deep learning model of Widar3.0 paper, as it utilizes only spatial feature extraction - a Convolutional Neural Network (CNN), which directly takes normalized and interpolated CSI amplitudes as input.

Figure 4.4b depicts our deep learning-based pipeline, which will be analyzed in Chapter 7, for adversarial domain adaptation. As it can be seen, data pre-processing and deep learning feature extractor of Chapters 5 and 6 are re-used. A major difference between our pipeline and the one from Widar3.0 paper is the adversarial training using a generator with additional domain labels. More detailed analysis of this method will be presented in Chapter 7.



(a) Pipeline used in Chapters 5 and 6.

(b) Pipeline used in Chapter 7.

Figure 4.4: Pipeline of activity recognition system using WiFi CSI data used in this thesis

4.2 WiFi CSI Dataset and Computing Resources

In this section, we describe the WiFi CSI dataset used in this thesis for the purpose of model training and performance evaluation. We also briefly explain computing infrastructure and resources used for our experiments.

We used a large and rich open-source WiFi CSI dataset, called Widar3.0 [53] [52], which has been used by many other researchers as well. The dataset contains WiFi CSI data collected from 17 people performing different hand gestures such as push-pull, sweep, clap, (see Figure 4.6 for some example gestures) in three different environments, i.e., classroom, hall and office (identified as room 1, room 2, and room 3, respectively). Experiments were performed at each room, following the setup depicted in Figure 4.5 consisting of one transmitter and six receivers placed at different positions. Each human subject had to perform each gesture at 5 different locations and orientations with respect to the transmitter.

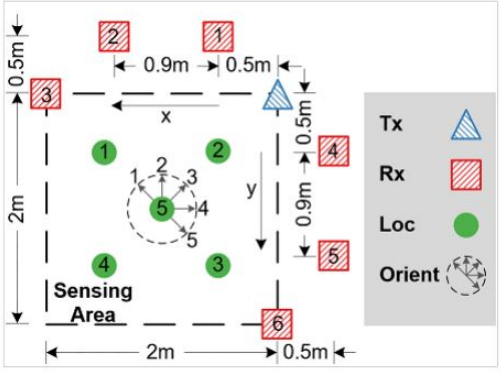


Figure 4.5: Experiment setup of Widar3.0 dataset, taken from [52]

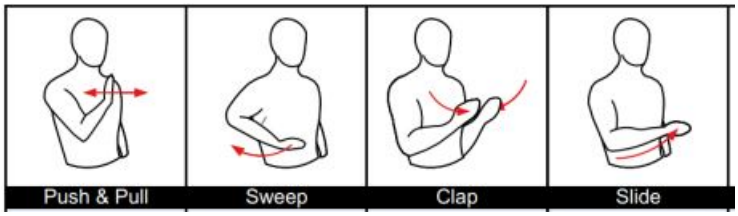
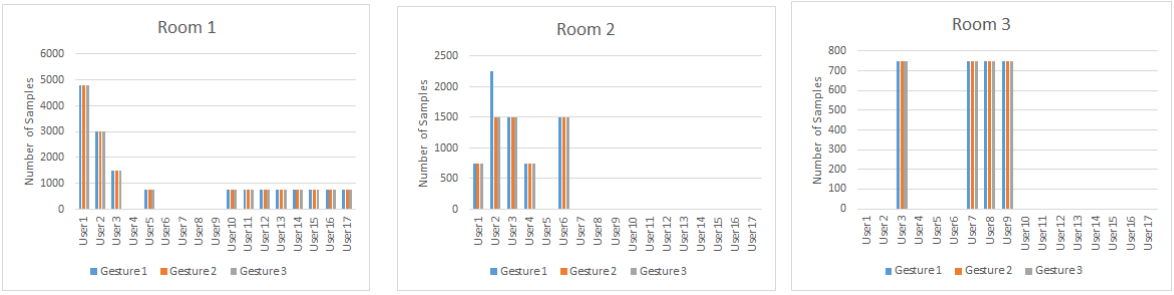


Figure 4.6: Widar3.0 dataset gesture samples, taken from [52]

Distribution of gesture samples (push/pull, sweep, clap) taken by human subjects at each room is presented in Table 4.1. It can be noticed that (i) only subject 3 performed experiments in every room, and (ii) subject 1 and 2 collected data in two rooms, i.e., room 1 and room 2. The rest of the subjects participated in the data collection only in one of the rooms. Data sample distribution per subject in each room is depicted in Figure 4.7. As it can be seen, the largest amount of data samples was collected by subject 1, 2 and 3. The other subjects collected approximately 2000 CSI samples each. Also, it can be noticed that the data set is balanced: approximately equal number of three gesture types were performed by each subject in each room. Overall, this data set will be used for our experiments in the following chapters, as it allows us to study the impact of the domain change problem caused by change of rooms, change of human subjects, and change of user orientation and location with respect to a transmitter.

Person ID	Room 1	Room 2	Room 3
1.	x	x	-
2.	x	x	-
3.	x	x	x
4.	-	x	-
5.	x	-	-
6.	-	x	-
7.	-	-	x
8.	-	-	x
9.	-	-	x
10.	x	-	-
11.	x	-	-
12.	x	-	-
13.	x	-	-
14.	x	-	-
15.	x	-	-
16.	x	-	-
17.	x	-	-

Table 4.1: Presence of users in different rooms in Widar3.0 data, taken from [52]



(a) Data distribution per subject in room 1

(b) Data distribution per subject in room 2

(c) Data distribution per subject in room 3

Figure 4.7: Data distribution of Widar3.0 dataset per subject in different rooms, taken from [52]

Additionally, Figure 4.8 depicts user distribution based on the body mass index (BMI). As it can be noticed, human subjects cover quite wide range of body height and weight. This allows to experiment with different people BMI and observe if physical body properties such as height and weight impact the recognition system performance. For instance, two clusters of people (A and B) can be taken, with different BMI metrics and experiments can be conducted with training set of A and test set of B.

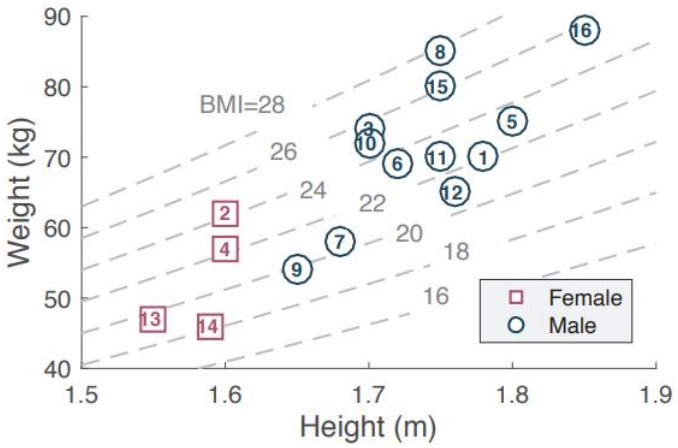


Figure 4.8: User body mass index distribution in Widar3.0 dataset, taken from [52]

Finally, as training models tend to be computationally expensive, research could be limited by available computing power. Depending on the workload, personal machine, server or any other cloud computing platform may be used. To train and test our approaches presented in Chapter 5, 6 and 7, using the Widar3.0 dataset, we used a Linux server (Ubuntu 18.04.5), with Intel(R) Core(TM) i7-6900K CPU (3.20GHz) and 126GB RAM, that has two NVIDIA "TITAN X" graphic processing units, with 12GB of video RAM each.

Chapter 5

WiFi CSI Data Pre-processing

Data pre-processing plays an important role in addressing domain change problem in learning-based activity recognition systems using WiFi CSI data. Based on recent research papers, modern deep learning approaches discussed in Chapters 3 and 4, proposed data pre-processing pipelines that increase recognition system performance by approximately 20% [21] or 40% [52]. However, as discussed in Section 4.1, commonly used pipelines in the literature consist of a long list of different methods and it is quite complex to find an optimal set of a combination of these techniques. Thus, we strive to automate the pre-processing phase by allowing the convolutional neural network to learn the optimal set on its own.

With this idea, we further analyze different methods to reduce complexity of the standard pre-processing pipeline presented in Section 4.1.1. We aim to find minimal CSI data pre-processing operations required by the convolutional neural network to function well in the presence of domain change. We further use this pipeline, to be presented in Section 5.3, in Chapters 6 and 7.

5.1 WiFi CSI Data Pre-processing Pipeline

Figure 5.1 shows our initial data pre-processing operations, which will be experimented and consist of i) amplitude extraction, ii) amplitude normalization, iii) amplitude interpolation, iv) de-nosing and v) VAE pre-training.

As raw CSI is represented in complex numbers (see Section 1.1), amplitude extraction becomes a necessary operation in order to get raw numerical values that neural network could work with. Using a complex number, represented as $x + yj$, amplitude $A = \sqrt{x^2 + y^2}$ and phase $\theta = \arctan \frac{y}{x}$ can be computed. Although having phase information may be beneficial, it is not studied in this thesis and left for future work.

The normalization operation, as discussed in Section 4.1.1, ensures that the input data has certain statistical properties, such that training stability could be achieved [18]. In the experiments in Sections 5.2.2 and 5.2.3, training is performed in batches of data, thus each batch is normalized, such that it is between 0 and 1.

Subsequently, amplitude is interpolated to achieve constant input shape. Due to the fact that authors of Widar3.0 paper used 30 sub-carriers when collecting CSI data and receivers used had 3 antennas, the final CSI dataset is three dimensional with the varying shape $[N, 30, 3]$, where variable N is the number of discrete time-steps. Each sample in Widar3.0 dataset contains different N and, therefore, we use interpolation to study effect of different variations of N, with methods described in Section 5.2.1.

Finally, we experiment the effects of de-nosing the data and pre-training with Variational Auto-Encoder (VAE) in Sections 5.2.2 and 5.2.3 respectively.

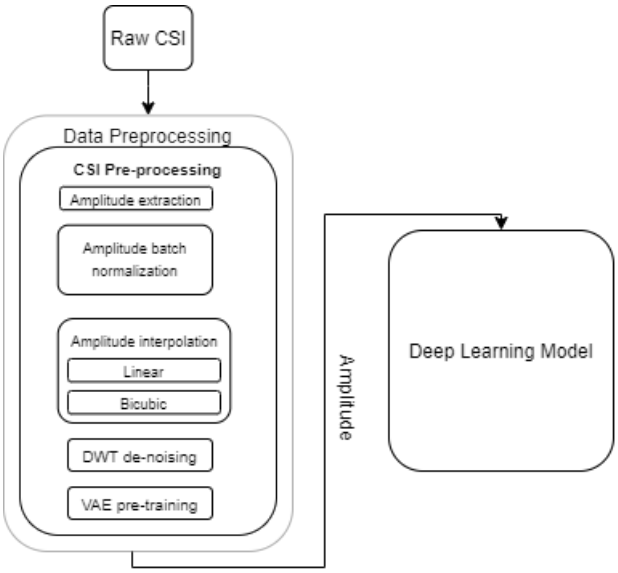


Figure 5.1: System overview with a data pre-processing pipeline used for experiments

5.2 Effect of Selected Pre-processing Steps on Activity Recognition

To study what effects our choice of data pre-processing steps will have on the overall performance of the activity recognition system, we use a simple 4-layer convolutional neural network, as depicted in Figure 5.2 for activity classification. As it can be seen, network receives an input of pre-processed CSI amplitude. Then pre-processed sample is further convolved by four convolutional layers, with a kernel size of $(3, 3)$, stride $(2, 2)$ and the "LeakyReLU" activation ($\alpha = 0.1$) function. Subsequently, last convolutional layer output (with shape $[100, 2, 256]$) is flattened to a vector of size 51200, which is then passed to 3 densely connected layers, with 256, 128, and 3 neurons each. Finally, output of the neural network is passed through a Softmax activation function to get probability of each gesture category.

Regarding the training process, the network weights are initialized with the Glorot uniform distribution [14] and trained using Adam [23] optimizer, with learning rate of 10^{-3} and objective loss function (categorical cross-entropy) presented in Chapter 2.

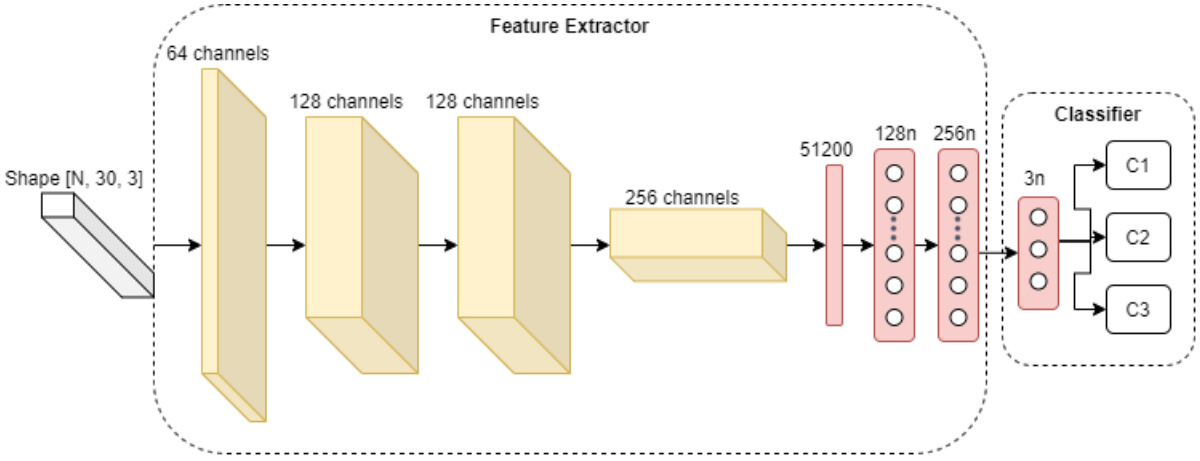


Figure 5.2: Convolutional neural network used for activity classification

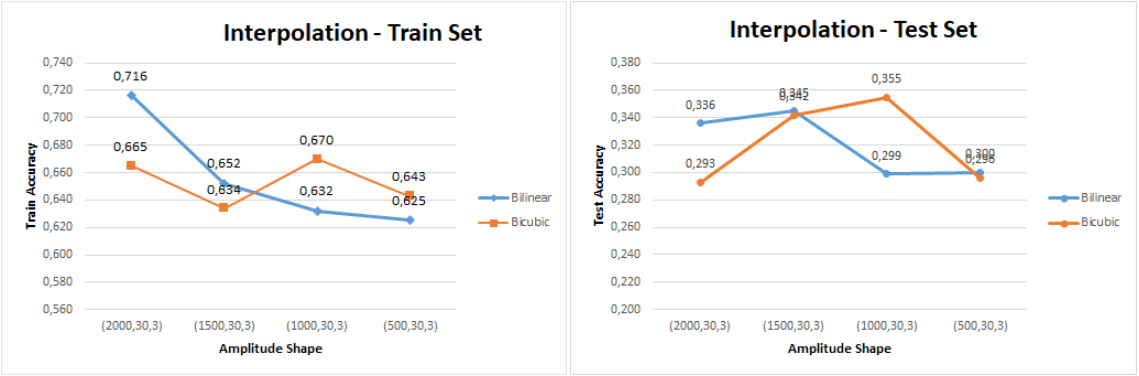
Regarding the data, part of Widar3.0 is used for experiments, represented in Table 5.1. Two people are taken for the experiments with the user IDs' 1 and 2 for training set and test set respectively. Users performed three hand gestures: push/pull, sweep and clap (Figure 4.6), in room 1 (classroom) and location 5 (Figure 4.5).

Number of categories	Train set	Test set	Room location	Room type	Face orientation w.r.t Tx	Amplitude
3 (Push/Pull, Sweep, Clap)	User ID 1	User ID 2	5	Room 1	All	+

Table 5.1: Widar3.0 data-set for pre-processing pipeline experiments

5.2.1 Effect of Interpolation

As discussed in Section 1.1, CSI sample shape is equivalent to a digital image with three channels $[Height, Width, 3]$. Thus image processing interpolation methods are used in accordance with finding of two survey papers, i.e., [3][39]. These survey papers indicate that bi-linear and bi-cubic interpolation methods are computationally cheap, having relatively low peak signal to noise ration. Therefore, we study impact of bi-linear and bi-cubic interpolation (without normalization step) on accuracy of our convolutional neural network trained over 11 epochs with varying input size $[N, 30, 3]$, when N decreases from 2000 to 500. As it can be seen from Figure 5.3a, while bi-linear interpolation training accuracy gradually decreases, accuracy of bi-cubic interpolation remains approximately the same over all varying values of N. This indicates that bi-cubic interpolation is more robust against varying resolution of input data and, therefore, will be used for final pre-processing module presented in Section 5.3. Regarding the test set accuracy, as it can be seen in Figure 5.3b, we observe that the model performed almost like random guessing, when both interpolation methods were used, with bi-cubic performing slightly better for lower values of N.



(a) Model accuracy with training dataset

(b) Model accuracy with test dataset

Figure 5.3: Effect of interpolation with varying input size N

5.2.2 Effect of Normalization and DWT de-noising

In this section, impact of batch normalization and DWT de-noising on training accuracy of our convolutional neural network model is studied. Firstly, de-noising helps to remove high frequency components from input data, which do not contribute to model performance. In particular, Figure 5.4 depicts comparison of amplitude sample with and without Discrete Wavelet Transform. Amplitude sample is first pre-processed with bi-cubic interpolation (shape $[1600, 30, 3]$), then reshaped for visualization purposes (shape $[320, 150, 3]$). As the final step, DWT is applied, resulting in

smoothed shades and colors. As it can be seen in Figure 5.5, four different pre-processing configurations of bi-cubic interpolation with and without DWT de-noising are compared, when CNN model was trained over 11 epochs. In particular, we experiment with (i) bi-cubic interpolation, (ii) bi-linear interpolation, (iii) bi-cubic with batch normalization, and (iv) bi-linear with batch normalization. First of all, it can be observed, that average training accuracy (62.75 %) of all configurations with DWT is higher than without it by approximately 7.5%. This shows that DWT de-noising of input data contributes to a better model training process, leading to a higher recognition accuracy than without it. Although, DWT shows positive results on model training accuracy, it will not be used and studied in further Chapters in this thesis, due to the fact that, our main objective, as it was already mentioned, is to simplify the pre-processing pipeline by finding minimal pre-processing steps that convolutional neural network could work with.

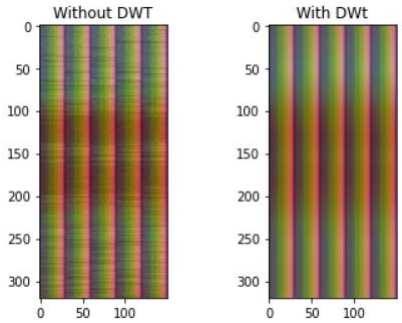


Figure 5.4: Amplitude sample visualized with and without Discrete Wavelet Transform (wavelet: "sym3", decomposition threshold 0.5)

Additionally, we noticed that training a model with batch normalization showed higher training accuracy than without. It can be seen in Figure 5.5 that the model trained with bi-cubic interpolation and bi-linear interpolation with batch normalization show approximately 2 to 3 percent higher training accuracy. This indicates that normalization of the input data contributes to a better model performance, and therefore pre-processing pipeline with bi-cubic interpolation and batch normalization will be used for further analysis in Chapters 6 and 7.

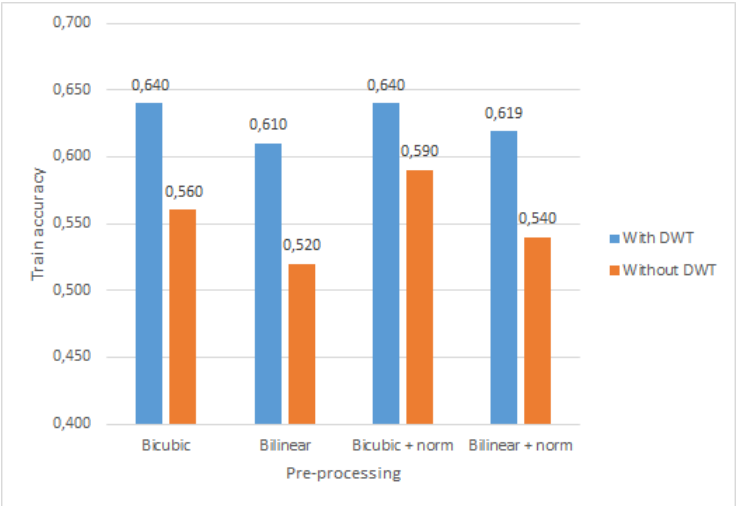


Figure 5.5: Discrete Wavelet Transform and batch normalization pre-processing results

5.2.3 Effect of Variational Auto-Encoder Pre-training

We also study impact of using Variational Auto-Encoder [24] (mentioned in Chapter 2, Section 2.1.2) pre-training as pre-processing step on model performance. As it can be seen in Figure 5.6, VAE pre-training takes place in two steps. Firstly, convolutional part of the model depicted in Figure 5.2 is replaced by a VAE encoder E. Then in the first training step, the auto-encoder is trained using the training dataset, with an objective to reconstruct the input amplitude as closely as possible, allowing the model learn optimal model parameters to encode input amplitude data to a latent vector z. Subsequently, in the second step, the trained convolutional part of the auto-encoder is taken and same densely connected layers are built on top as shown in Figure 5.2, resulting in the same model configuration and training procedure as discussed in Section 5.2. It is also important to note that batch normalization and bi-cubic interpolation pre-processing steps are included in the pre-processing pipeline of this experiment.

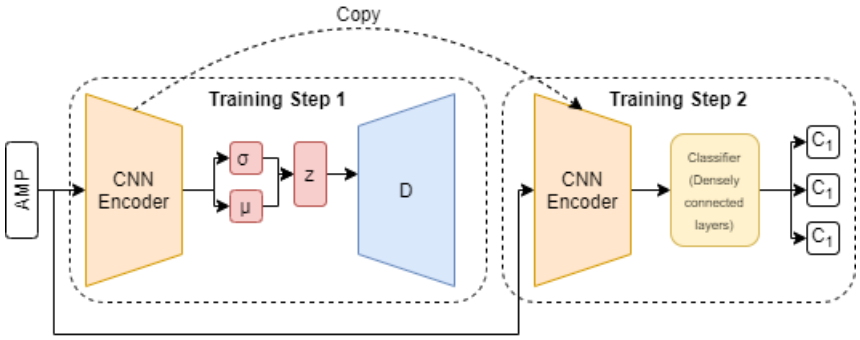


Figure 5.6: Variational Auto-Encoder pre-training diagram.

Our results of such VAE pre-training procedure, described above, are represented in Figure 5.7. As it can be observed, the graph represents model accuracy with varying latent vector z from size 128 to size 512. Based on the results, model training accuracy positively correlates with the increasing size of latent vector z, achieving approximately 65% training accuracy, which is comparable with result of DWT pre-processing (Figure 5.5). However, VAE pre-training does not show any signs of addressing the domain change problem, as model tested with different user (ID 2) shows an accuracy just slightly higher than random guessing. Thus, VAE pre-training will not be used as additional pre-processing step in further experimentation in Chapters 6 and 7.

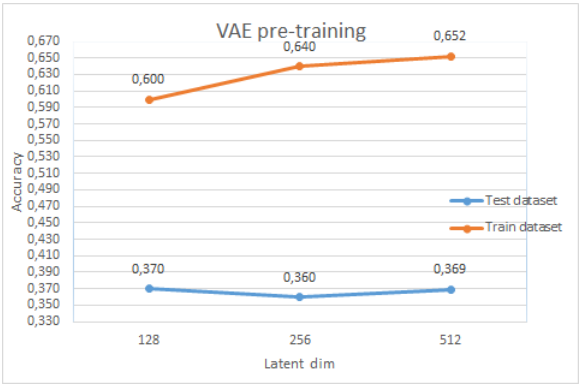


Figure 5.7: Variational Auto-Encoder pre-training results

5.3 Final pre-processing

As an outcome of this chapter, in this section we present final and simplified pre-processing module in Figure 5.8, which will be used in further chapters 6 and 7. As it can be seen, only three operations are included, such as amplitude extraction, normalization and bi-cubic interpolation for the input of deep learning model.

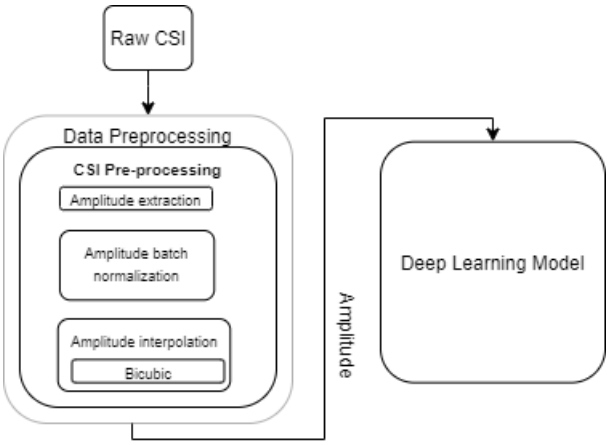


Figure 5.8: Simplified CSI pre-processing.

Chapter 6

Deep Learning-based Feature Extraction

In this chapter we focus on the deep learning model used in the feature extraction part of the pipeline. In Chapter 5, we used a simple 4-layer CNN as a feature extractor (see Section 5.2) to find out minimal pre-processing operations that a deep learning model could work with and yet achieve an accepted activity recognition accuracy on train data set. However, based on the experimental results, described in Section 5.1, we observed that the CNN model used was not robust in presence of domain change - change of user, when the model was trained on one person and validated on another. Thus, as it can be seen in Figure 6.1, in this chapter we re-use data pre-processing module found in Chapter 5 and propose a different model for feature extraction. In particular, we improve the Convolutional Neural Network (CNN) model and its objective function in order to address the domain change problem.

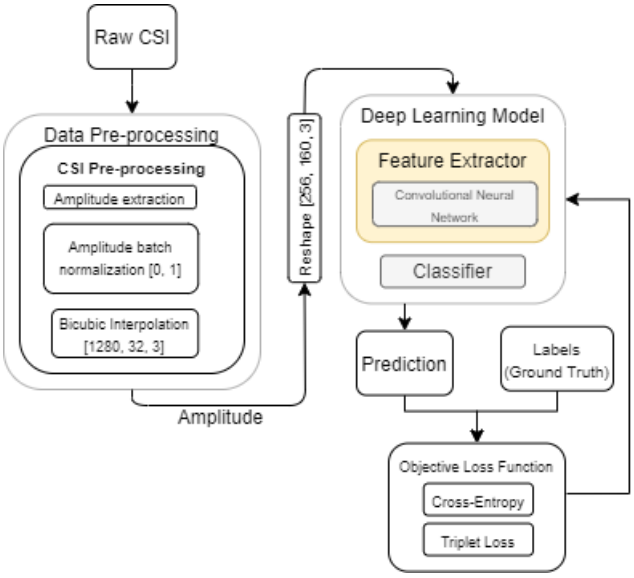


Figure 6.1: Feature extractor model overview

6.1 A CNN-based Model for Feature Extraction

6.1.1 CNN Architecture

Base architecture of the CNN model used for our feature extraction component is shown in Figure 6.2, which will be used and updated with each experiment in this chapter. CNN feature extractor consists of 5 CNN blocks, where first block takes an input of CSI amplitude in a reshaped form $[256, 160, 3]$. Each block reduces its input by approximately twice and contains three operations, i.e., convolution, batch normalization, and LeakyReLU activation. Then convolutional part of the model is followed by a stack of 4 layers, which starts with an operation to flatten out feature maps of the last CNN layer, followed by densely connected layer of size variable z (latent dimension), which will be varied in the experiments. After that the vector z is processed with L_2 normalization, which is then passed to a densely connected layers of size 64. Finally, the output of feature extractor is then passed to a classification module, which contains a densely connected layer, with 3 neurons, followed by softmax activation, in order to output probability distribution of three type of gestures.

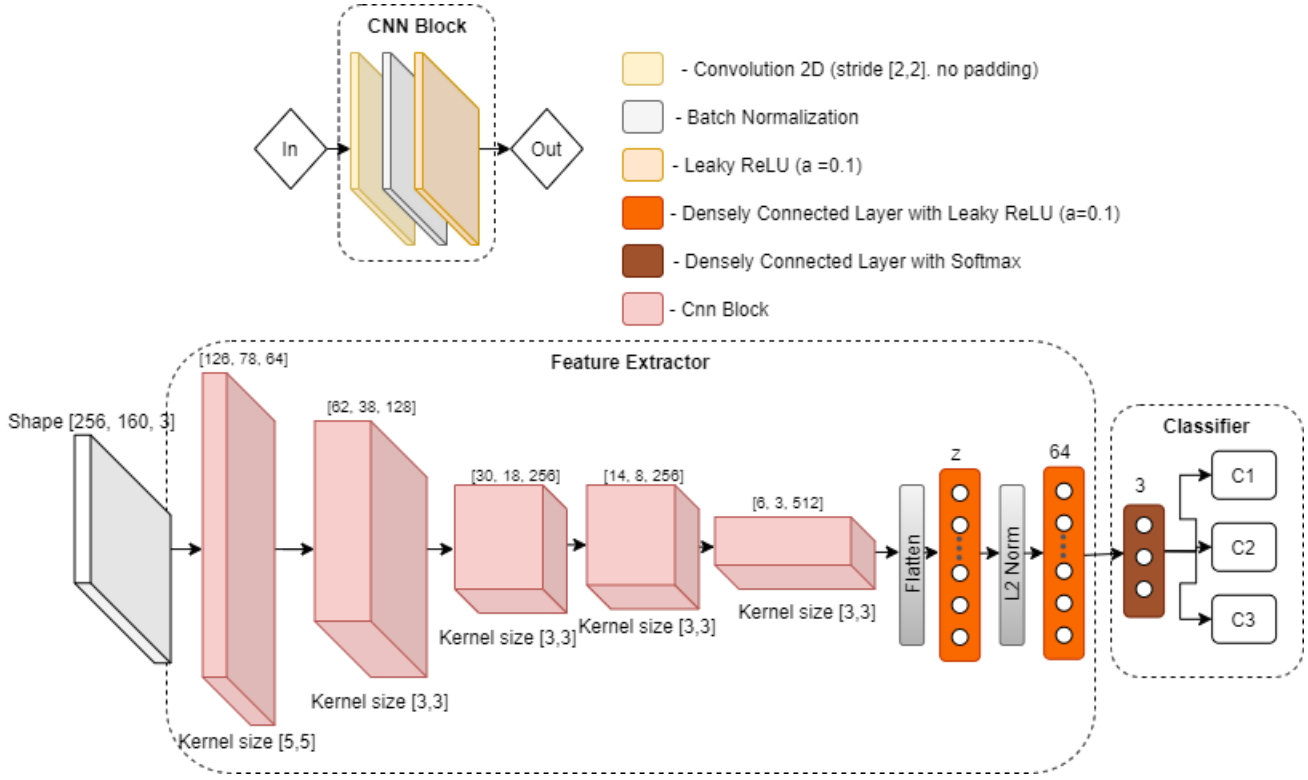


Figure 6.2: CNN feature extractor architecture

6.1.2 Training and Testing CNN Model

Regarding the training configuration, the network showed in Figure 6.2 weights were initialized with Glorot uniform distribution [14] and trained using Adam [23] optimizer, with learning rate of 10^{-3} for all experiments. Since, the task is to classify person gestures, i.e, push/pull, sweep, and clap, as presented in Table 6.1, we use a combination of cross-entropy $\mathbb{L}_c$ and triplet loss $\mathbb{L}_T$, with margin $\alpha = 1$ (as discussed in Section 2.4.2) as an objective loss function $\mathbb{L} = \mathbb{L}_c + \mathbb{L}_T$ and aim to minimize it. By adding triplet loss in the objective function, we seek to improve model training convergence, by forcing it to group CSI gesture samples into clusters in the embedding space, with z dimensions (a.k.a. latent dimension). This will be discussed in more details in Section 6.2.3.

For model training, we use data that creates conditions of domain change issue. For that we use samples from all users in room 1 except user 11, which was used only for testing purposes. The

Number of categories	Train set	Test set	Room location	Room type	Face orientation w.r.t Tx	Repetition ID
3 (Push/Pull, Sweep, Clap)	U1, U2, U3, U5, U10, U12, U13, U14, U15, U16, U17	User ID 11	All	Room 1	All	1-3

Table 6.1: Widar3.0 data-set for feature extractor experiments

samples include all gesture samples collected at all locations and face orientations with respect to the transmitter. Due to highly time consuming training procedure, only first 3 user gesture repetitions were used in training and test datasets.

6.2 Analyzing Feature Extractor in Presence of Domain Change

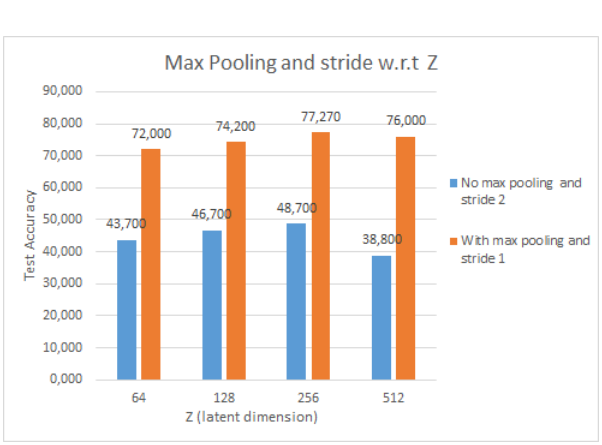
To study effect of different parameters and design choices of our CNN model on accuracy of activity recognition in presence of domain change, we perform a number of experiments. In particular, in Section 6.2.1, we investigate effect of adding max pooling layer, changing convolutional layer kernel strides, and adding dropout regularization. In Section 6.2.3, we analyze the impact of different triplet loss margins (i.e. α). In Section 6.2.4, we study performance of Bayesian optimization for hyper-parameter tuning. Finally, overall result comparison with tuned network of this Chapter, Widar3.0 [52] and initial CNN (from Chapter 5) is presented, which are based on leave one out user/room validation.

6.2.1 Effect of Max Pooling and Kernel Stride

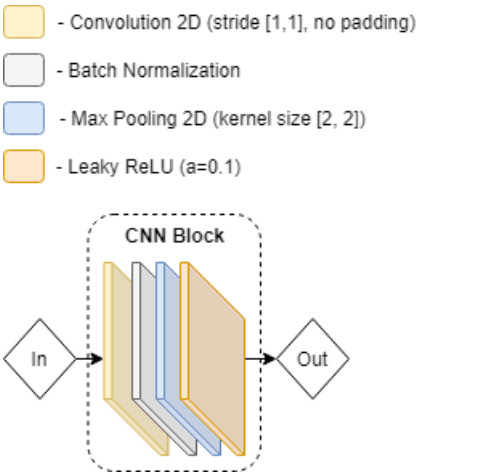
In the very first experiment, we strive for analysing the effect of changing CNN block parameters of the base model in Figure 6.2. As can be seen in Figure 6.2, base model CNN block consist of convolution operation, followed by batch normalization and LeakyReLU ($a = 0.1$, Section 2.3.1). Then, as depicted in Figure 6.3b, we alter the CNN block, by inserting max pooling operation and changing convolution kernel stride to 1. Based on the background information in Section 2.2.3, the max pooling layer allows to extract statistics that is invariant to small changes of the input to the convolutional layer. This may help to address the shortcomings of the minimal pre-processing (discussed in Chapter 5), which were found to be not contributing to address the domain change issue.

In order to test the changes, we perform experiments with the data set, described in Table 6.1. As it can be seen in Figure 6.3a, it depicts a bar chart, which represents comparison of test accuracy between our CNN model with and without max pooling, with increasing latent dimension z of the densely connected layer in feature extractor module.

As the first observation, we notice that the CNN model with max pooling and convolution kernel stride 1 outperforms the CNN model without max pooling and convolution kernel stride 2 by on average approximately 30%. This shows that the former has high impact on model performance when tested on a "not seen" user. Moreover, the impact of latent dimension z can be noticed. We observe that model test accuracy positively correlates with the increasing latent dimension z, from 64 to 512 in both CNN block types, with the test accuracy peaking at $z = 256$. As a result, based on this experiment, we further use $z = 256$ and modified version of CNN block, depicted in Figure 6.3b. Finally, regarding the training set accuracy, it reached 99.99% for all experiments.



(a) CNN feature extractor max-pooling and stride w.r.t latent dimension z experiment

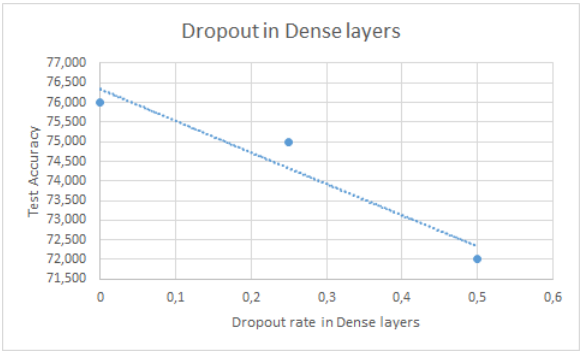


(b) CNN feature extractor CNN block

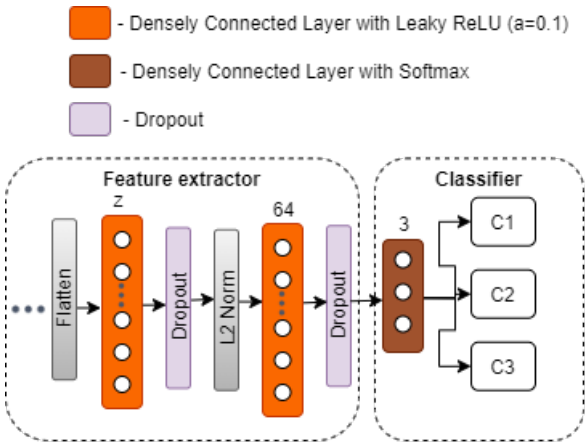
Figure 6.3: Effect of adding max pooling and changing convolution kernel stride

6.2.2 Effect of Dropout Regularization

In this section, we evaluate effect of adding the dropout regularization to parts of the feature extractor module (Figure 6.2). Based on the previous experiment (Figure 6.3a), best CNN architecture was chosen and, as represented in Figure 6.4b, dropout layer (Section 2.2.6) was added after each densely connected layer. The test accuracy of the resulting new model with varying dropout probabilities p are shown in Figure 6.4a. It can be observed that test accuracy negatively correlates with increasing p, showing the best test accuracy when $p = 0$. This indicates that dropout regularization after each densely connected layer in feature extractor is not contributing towards better accuracy, when user 11 is the left out for testing. Therefore, dropout regularization in will not be used in further experiments in this chapter.



(a) Dropout regularization results



(b) Classifier module with dropout regularization

Figure 6.4: Dropout regularization effect study

6.2.3 Effect of Triplet Loss Function

The triplet loss function aims to group CSI gesture samples into three clusters (3 categories) in the embedding space of z dimensions. Based on Section 2.4.2, this is achieved by maximizing the $L2$ distance between the samples belonging to different categories at least with a margin α

and minimizing the distance samples belonging to the same category. As depicted in Figure 6.2 embedding of each gesture sample is computed by passing it through the CNN module, followed by densely connected layer, with z neurons, and L_2 normalization.

Based on the previous experiments, we take the best model architecture, which contains max pooling operations in each CNN block and embeds gesture samples into size $z = 256$ vector (embedding space). Figure 6.5 shows test accuracy with varying α. It can be seen that increasing margin α from 0.1 to 1 results to decreasing test accuracy, on the other hand, increasing α from 1 to 1.75 shows and increasing trend, peaking at 80%, when $\alpha = 1.75$. This indicates that triplet loss with sufficiently high margin contributes to addressing the domain change problem. Thus, margin $\alpha = 1.75$ will be used for further experimentation in Chapter 7. Regarding the model accuracy with training dataset, it achieved approximately 99.99% in all experiments.

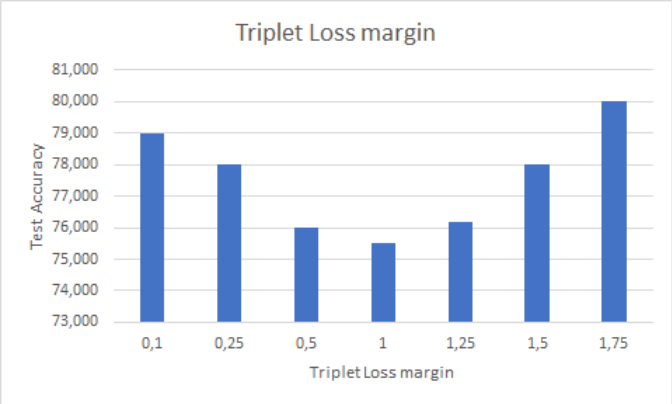


Figure 6.5: Effect of triplet loss margin

We further analyze effect of the triplet loss by visualizing the embedding vectors. Firstly, we take the trained model with triplet loss margin $\alpha = 1.75$ to compute embedding vectors of each gesture sample in train data set and test data set. Then each embedding sample vector of size $z = 256$ was reduced to the size of $z = 3$, using Principal Component Analysis (PCA) for visualization purposes in three dimensional space. The resulting visualization can be seen in Figure 6.6. It can be observed that three clusters corresponding to each gesture category were obtained. While in the training set no outliers were observed (see Figure 6.6a), forming almost perfect clusters, test set showed clusters, which are quite scattered and have a number of outliers (see Figure 6.6b). This confirms the lower test accuracy (80%) compared with the training accuracy (99.99%), indicating worse model performance in presence of domain change.

Overall, triplet loss forces the model to automatically group all gesture samples into clusters in the embedding space, regardless of who, where, and when the gesture samples were taken. By doing so, it contributes to address the domain change problem.

(a) Training set embedding vectors visualized in 3-dimensional space with PCA

(b) Test set embedding vectors visualized in 3-dimensional space with PCA

Figure 6.6: Visualization of effect of triplet loss. Each cluster corresponds to a gesture type.

6.2.4 Effect of Bayesian Optimization

With this experiment, we further seek to optimize the CNN architecture towards a better test accuracy. Due to the fact the there are too many parameters to tweak, causing a large search space, we use Bayesian optimization as described in [12] to find the best set automatically. This optimization algorithm is commonly used for tuning hyper-parameters of a machine learning model in order to obtain optimal test accuracy or loss on a validation data set.

We aim at fine-tuning the hyper-parameters of the CNN model towards the best model test accuracy using Keras tuner [35].

We provide to the Bayesian optimizer a search range for each hyper-parameter for each convolutional block in our CNN model and each densely connected layer. These search ranges can be found in Table 6.2 and Table 6.3, respectively. The learning rate provided to the Bayesian optimizer for the Adam optimizer was $[10^{-2}, 10^{-4}]$ and the loss margin α for the triplet loss was $[0, 1.75]$.

	Number of kernels	Kernel size	Padding type	Strides	Leaky ReLU α
Range	[128, 512]	[3, 5]	Padding/No Padding	[1, 3]	[0.1, 0.4]

Table 6.2: Convolutional block hyper parameter range provided to Bayesian optimizer. Note: max pooling layer with kernel size (2,2) is automatically added after each convolutional operation if Bayesian optimizer chooses the *stride* = 1. This makes sure that feature map sizes are always reduced at least by half after each CNN block.

	Number of neurons	Leaky ReLU α
Range	[64, 512]	[0.1, 0.4]

Table 6.3: Hyper-parameters of the densely connected layer provided to Bayesian optimizer

In total the model was trained 90 times, with different set of hyper-parameters. Each training run was stopped as soon as validation loss did not decrease for at least 5 training epochs. The resulting optimal hyper-parameters found by Bayesian optimizer are presented in Tables 6.4 and 6.5, with learning 10^{-4} and triplet loss margin $\alpha = 1.5$.

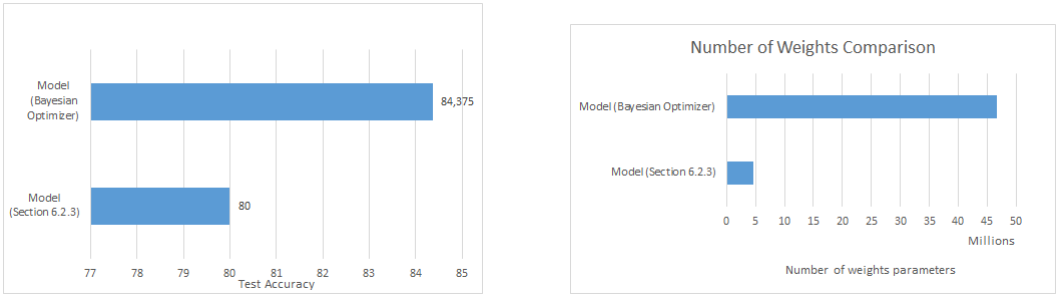
As it can be seen in Figure 6.7a, the optimized model with Bayesian optimizer outperformed the model found in Section 6.2.3 by over 4 percent, achieving the best performance with approximately 84.4% test accuracy. This indicates that Bayesian optimization is a well suited algorithm for finding optimal hyper-parameters, especially when each training run is computationally expensive. Additionally, Bayesian optimized model showed further contribution to improving the model performance in presence of domain change. On the other hand, the Bayesian optimized model found to be quite memory consuming. This is due to high number of (i) kernels in each CNN block and (ii) neurons in densely connected layer (see Tables 6.4, 6.5). This results in quite a large model compared with the model of Section 6.2.3. A comparison between number of weight parameters of these two models can be seen in Figure 6.7b. It can be clearly noticed that Bayesian optimized model contains approximately 10 times more weight parameters compared with the model of Section 6.2.3, reaching the limits of hardware (GPU memory), described in Section 4.2 using Tensorflow framework [1]. Thus, we will use the Bayesian optimized model as a benchmark for further studies and comparisons in Section 6.3, but the model architecture itself will not be used for further investigations due to hardware limitations as it cannot be used in combination with other neural networks for adversarial domain adaptation that will be discussed in the following Chapter 7. Instead, we use the model found in Section 6.2.3 for further analysis.

Layer	Number of kernels	Kernel size	Padding type	Strides	Leaky ReLU α
Conv Block 1	512	5	No Padding	1	0.1
Conv Block 2	512	3	No Padding	1	0.4
Conv Block 3	512	3	Padding	1	0.4
Conv Block 4	512	5	Padding	1	0.1
Conv Block 5	512	5	No Padding	3	0.1

Table 6.4: Optimal hyper-parameters of CNN found by Bayesian optimizer

Layer	Number of neurons	Leaky ReLU α
Dense 1	512	0.1
Dense 2	64	0.4

Table 6.5: Optimal densely connected layer hyper-parameters found by Bayesian optimizer



(a) Test accuracy comparison

(b) Number of weight parameters comparison

Figure 6.7: Model of Section 6.2.3 comparison with the model optimized with Bayesian optimizer

6.3 A Robust CNN-based Feature Extractor against Domain Change

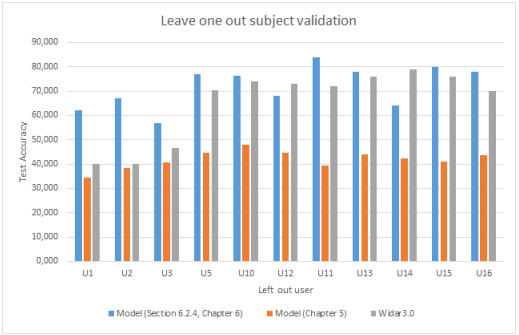
Based on results of the previous experiments, our modified CNN model looks like Figure 6.2, with a modified CNN block as in Figure 6.3b, which adds max pooling operation and utilizes convolution operation with stride $[1, 1]$. Additionally, based on the experiment in Section 2.4.2, this model architecture is used with triplet loss margin $\alpha = 1.75$ for further experiments in chapter 7. **Note**: as already mentioned in Section 6.2.4, this architecture will not be used for performance comparison instead Bayesian optimized model will be used as it performed better by 4% in terms of test accuracy, with a trade-off of high number of weights, indicating that it cannot be used for further investigations in chapter 7 due to hardware memory limitations.

In order to compare the performance in presence of domain change, we perform leave-one-out subject/room validation tests for three models, i.e., the model based on Bayesian optimizer (see Section 6.2.4), the model presented in Widar3.0 paper [52], and the initial model used in Chapter 5. Results of these tests are depicted in Figure 6.8.

For each model, the same minimal CSI pre-processing steps found in Chapter 5 were used. These minimal steps include amplitude batch normalization in range [0, 1], followed by bi-cubic interpolation, resulting in sample shape of $[1280, 32, 3]$.

Regarding leave one out subject validation, CSI gesture samples of all subjects in room 1 were included in the train set, except the left out, who was used for testing. As it can be seen in Figure 6.8a, the Bayesian optimizer-based model obtained in Section 6.2.4 achieves an average test accuracy of approximately 72%, outperforming the Widar3.0 model and the initial model of Chapter 5, with an average test accuracy of 65% and 42%, respectively. This indicates that our feature extractor model is quite robust in presence of domain change - change of subject, with an average classification accuracy of approximately 72%.

Regarding leave one out room validation, CSI gesture samples of all subjects in room 1 were included to the train set and CSI samples of one chosen subject, but in different room were taken in the test set. The experimental results can be seen in Figure 6.8b. While model obtained in experiment 3 classifies subject gestures in different rooms, with an average test accuracy of 71.3%, Widar3.0 and model experimented in Chapter 5 achieved an average test accuracy of 61% and 40%, respectively. This indicates that the model described in Section 6.2.4 outperforms Widar3.0 and initial model (of Chapter 5) by on average 10.3% and 31.3%, respectively.



(a) Leave-one-out subject validation, where x-axis represents left out subject in room 1 for testing. Data set used: room locations - all, face orientations w.r.t Tx - all, gesture sample repetitions - first 3.

(b) Leave-one-out room validation, where x-axis represents left user in room x for testing, e.g. U1_R2 - user 1 in room 2. Data set used: room locations - all, face orientations w.r.t Tx - all, gesture sample repetitions - first 3.

Figure 6.8: Leave-one-out subject/room validation

Chapter 7

Adversarial Domain Adaptation

Results described in Chapter 6 show that a certain degree of robustness in domains of different subjects and rooms was achieved by fine tuning different hyper-parameters of the feature extractor CNN model. In this chapter, we further seek to improve the model performance in presence of domain change by utilizing domain information and adversarial training. As it can be seen in Figure 7.1, we re-use the data pre-processing module of Section 5.3 and the feature extractor model of Section 6.2.3 in this chapter. The major difference is the adversarial training (to be described in Section 2.1.3), using which the generator competes against the discriminator to classify CSI gestures and to categorise the domains such as subject ID or room ID. The generator is generating CSI gesture amplitude samples in such a way that it confuses the discriminator to classify from which domain the sample is from leading the discriminator to a classification performance independent of the domain change.

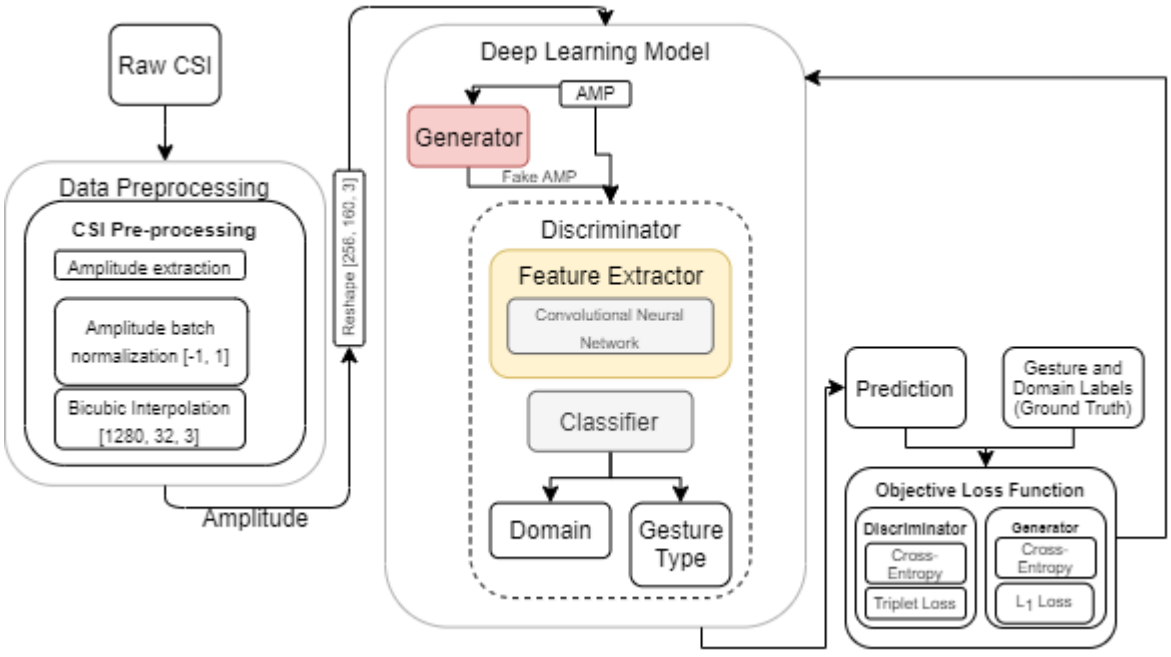


Figure 7.1: Adversarial domain adaptation model overview

7.1 Adversarial Domain Adaptation Model

7.1.1 Adversarial Network Architecture

Figure 7.2 shows a high-level overview of the architecture of our adversarial network. It consists of two artificial neural networks, i.e., (i) a generator and (ii) a discriminator, which compete against one another. For the discriminator part of the model, we use the same model as our feature extractor and classifier as described in Section 6.3. We then borrow the idea from [22] to discriminate between the domains and gestures using domain discriminator (DD) and gesture discriminator (GD) respectively in our classifier module. In our case, we use subject ID as domain labels and leave out other possible domain labels such as room ID, face orientation, or subject location in a room for future studies.

For the generator, we use a U-Net based architecture [20], which was originally designed for translating an input image to its corresponding output image by sharing information between intermediate layers of encoder and decoder. Based on this idea, we propose a UNet based architecture variant that translates CSI amplitude x into a fake amplitude x_f. The proposed architecture is depicted in Figure 7.3. It takes an input of pre-processed CSI amplitude sample, one hot encoded gesture label, and subject ID. Then based on the provided input, it translates amplitude sample x into a fake amplitude sample x_f, which is then passed to discriminator.

The overall objective functions, which have to be minimized by the discriminator and the generator, are described in Equations 7.1 and 7.2, respectively. Regarding the discriminator it outputs probability classes of three gesture categories and k probability classes for domain categories, where $(k+1)$th is for unknown. As it can be seen Equation 7.1, the loss $\mathbb{L}_{\mathbb{D}}$ consists of four terms. The first term corresponds to gesture discriminator (GD) loss with real gesture samples (x, y_g). The second term is the loss term for domain discriminator (DD) with real gesture samples (x, y_d), where y_d corresponds to subject ID. Finally, the third and the fourth terms correspond to domain discriminator (DD) loss with input of fake sample (x_f, y_d), where $y_d = k+1$ - unknown domain category and triplet loss $\mathbb{L}_T$ (margin $\alpha = 1.75$), respectively.

$$
\begin{aligned}
\mathbb{L}_D = & - \mathbb{E}_{x,y_g \sim p_{data}(x,y_g)} \log \left[p_{GD}(y_g|x) \right] \\
& - \mathbb{E}_{x,y_d \sim p_{data}(x,y_d)} \log \left[p_{DD}(y_d|x, y_d < k+1) \right] \\
& - \mathbb{E}_{x_f \sim p_G(G(x_f|(x,y_g,y_d)))} \log \left[p_{DD}(y_d = k+1|x_f) \right] \\
& + \mathbb{L}_T \\
& \text{, where } y_g \text{ for gesture type, } y_d \text{ for domain (subject id), } k \text{ - number of domain categories}
\end{aligned}
\tag{7.1}
$$

The generator objective function $\mathbb{L}_G$ consists of three loss terms. The first and the second terms motivate the opposite of discriminator, as described in $\mathbb{L}_{\mathbb{D}}$. The fist term corresponds to the domain discriminator (DD) with fake sample (x_f, y_d) as input, where the domain (subject ID) is $y_d < k+1$. This ensures that the generator is penalized if its generated sample x_f is classified by the domain discriminator as an unknown $(k+1)th$ domain category. The second term defines gesture discriminator with (x_f, y_g) as input, where y_g is the corresponding gesture category. Finally the last loss term is the L_1 loss between the original CSI amplitude x and fake amplitude x_f, weighted with constant β.

$$
\begin{aligned}
\mathbb{L}_G = & - \mathbb{E}_{x_f \sim p_G(G(x_f|(x,y_g,y_d)))} \log \left[p_{DD}(y_d|x_f, y_d < k+1) \right] \\
& - \mathbb{E}_{x_f \sim p_G(G(x_f|(x,y_g,y_d)))} \log \left[p_{GD}(y_g|x_f) \right] \\
& + \beta [\mathbb{E}_{x \sim p_{data}(x), x_f \sim p_G(G(x|(y_g,y_d)))} ||x - x_f||_1] \\
& \text{, where } y_g \text{ for gesture type, } y_d \text{ for domain (subject id), } k \text{ - number of domain categories}
\end{aligned}
\tag{7.2}
$$

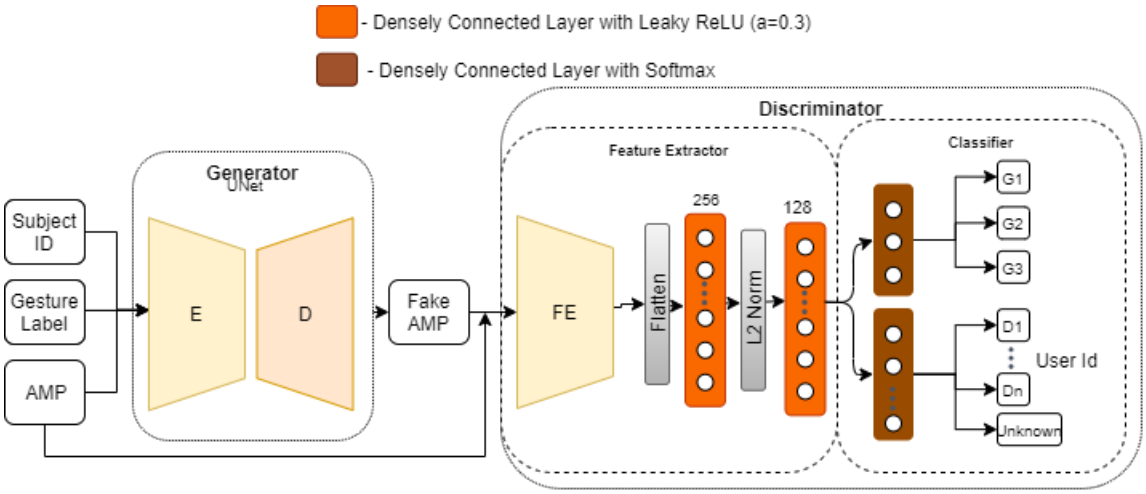


Figure 7.2: Adversarial network model

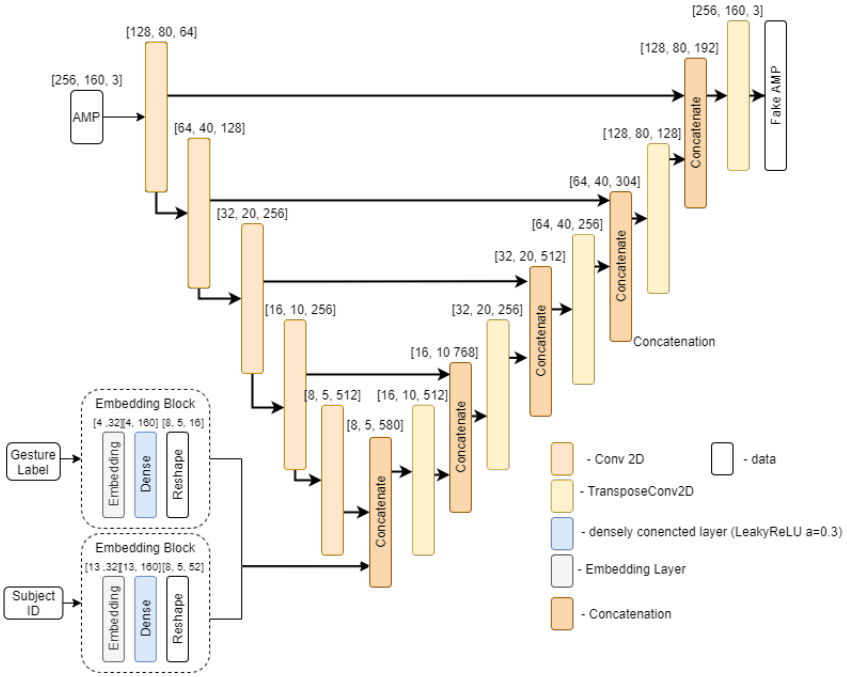


Figure 7.3: UNet model architecture

The above described generator and discriminator networks are trained one after another with a data batch of size 32 and Adam optimizer (with learning rate 0.0002 and $\beta_1 = 0.5$), until it is observed that the gesture classification test accuracy does not get improved anymore. We use the data set as the same used in experiments of Chapter 6, which is presented in Table 6.1. It takes all gesture samples performed by all subjects in room 1, except subject 11, which is used only for testing purposes.

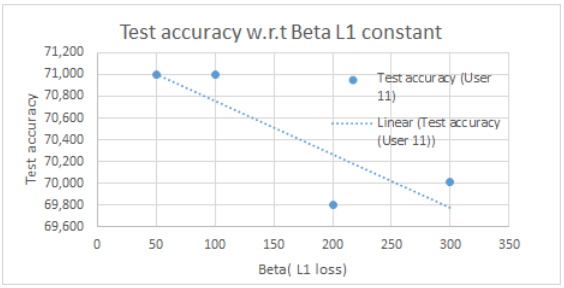
7.2 Analyzing Adversarial Network in Presence of Domain Change

To study effect of different design choices of our adversarial network architecture on accuracy of activity recognition in presence of domain change, we perform a number of experiments. In particular, we study effects of $\mathbb{L}_1$ loss term in Section 7.2.1, UNet regularization in Section 7.2.2, and Discriminator regularization in Section 7.2.3.

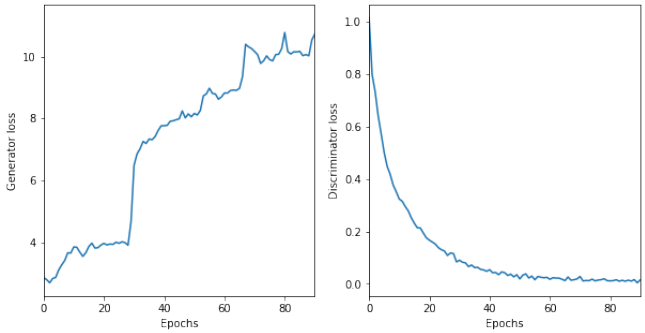
7.2.1 Effect of $\mathbb{L}_1$ Loss

In the first experiment, we analyze effect of the $\mathbb{L}_1$ loss term of the generator, which describes how similar the original input sample x and a fake sample x_f are. In our model, we vary the constant weight β. As it can be seen in Figure 7.4a, the model test accuracy of gesture classification shows decreasing trend when β is increased from 50 to 300, peaking at points $\beta = 50$ & $\beta = 100$. As generator aims to translate an input sample x in a way that confuses the discriminator, higher values of β force the generator to produce samples that are too similar, resulting in worse performance. Therefore, we select $\beta = 50$ for further analysis in the following experiments in this chapter.

Regarding the domain discriminator classification accuracy, it was observed that it quite quickly reaches approximately 99%, over performing generator, which loss showed increasing trend over all training session (Figure 7.4b). As a consequence, it can be observed that generator and discriminator are inbalanced. Thus in order to maintain the balance, we further investigate various regularization methods in the following experiments.



(a) Generator $\mathbb{L}_1$ β constant w.r.t test accuracy



(b) Comparison between generator and discriminator losses during training, when $\beta = 50$

Figure 7.4: Effect of $\mathbb{L}_1$ loss

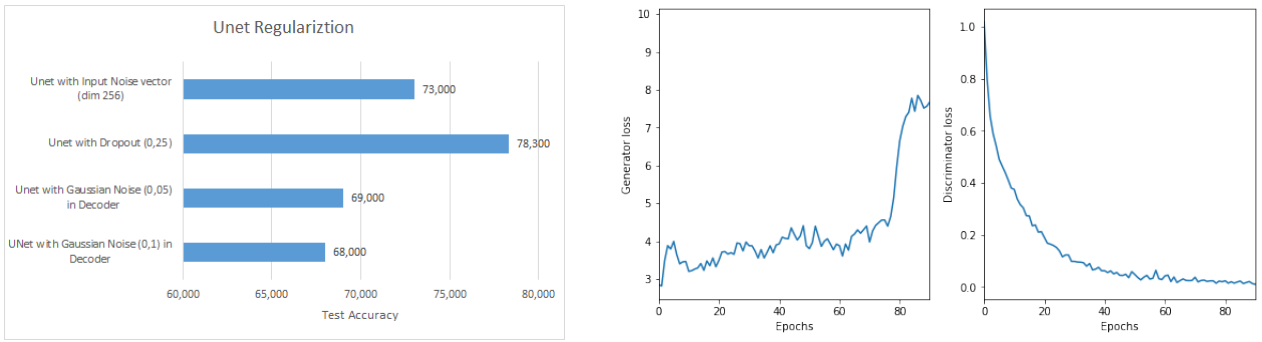
7.2.2 UNet Regularization

In this experiment, we further seek to improve the adversarial training balance between generator and discriminator, by firstly conducting experiments with different regularization methods used in the generator network, which would introduce some "randomness" for fake sample generation. In particular, for each UNet generator layer we try different regularization methods: dropout, with $p = 0.25$ and addition of Gaussian noise (with standard deviation $\sigma = 0.1$ & $\sigma = 0.05$). Additionally, we experiment with adding a 4_{th} input, i.e., a Gaussian noise vector of size 256 ($\sigma = 0.1$) to the generator and concatenating it with UNet bottleneck layer the same way as gesture label, as depicted in Figure 7.3.

The test accuracy, using each method is shown in Figure 7.5a. It can be clearly noticed that adding Gaussian noise to each layer output of UNet decoder part results in the worst performance out of all methods. The standard deviation $\sigma = 0.05$ shows slightly better results than $\sigma = 0.1$,

reaching approximately 68% and 69% test accuracy. On the other hand, adding i) Gaussian noise as input to the Unet network and ii) the dropout shows higher results, reaching approximately 73% and 78%, respectively, outperforming the test accuracy of previous experiment. Therefore, we use $p = 0.25$ dropout after each UNet layer as the regularization method for further experiments.

We also analyze the balance between generator and discriminator during training, with the dropout regularization in UNet. As it can be seen in Figure 7.5b, the generator loss with some fluctuations stayed at approximately same value until *epoch* = 60, followed by increasing trend till end of the training. This indicates that dropout regularization contributes to improving performance of the generator to compete against the domain discriminator.



(a) Experiments of various regularization methods in UNet (generator)

(b) Comparison between generator and discriminator losses during training, when using dropout in UNet and L_1 loss $\beta = 50$

Figure 7.5: Effect of UNet regularization

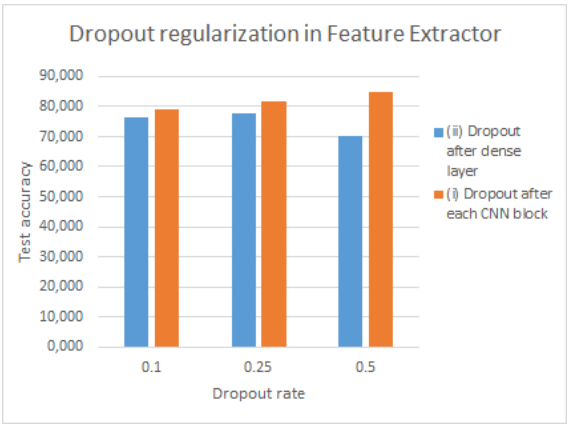
7.2.3 Effect of Discriminator Regularization

Experiments of the section showed that the dropout positively affect balancing between the generator and the discriminator during the training phase, leading to 78.3% test accuracy. In this section, we further investigate effect of the dropout on the discriminator network.

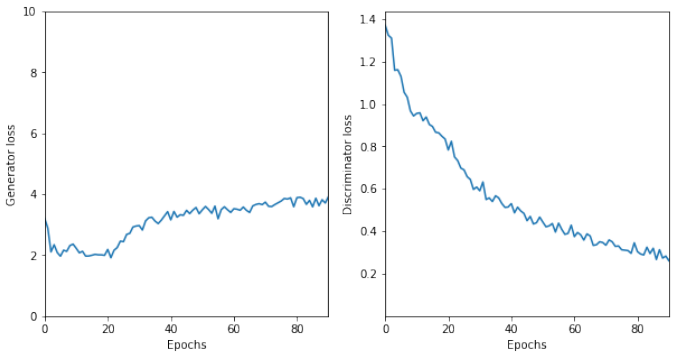
Effect of dropout regularization with varying dropout probabilities p from 0.1 to 0.5 is depicted in Figure 7.6a. As it can be seen, we apply the dropout to different parts of the discriminator network, i.e. (i) after each CNN block of the feature extractor, and (ii) after each densely connected layer in feature extractor (see Figure 7.2). Applying the dropout to the (i) leads to an increasing accuracy, with test accuracy peaking at 84.7%, when dropout $p = 0.5$. On the other hand, applying the dropout to the (ii) leads to a decreasing accuracy with the lowest test accuracy 70.1%, when $p = 0.5$. Therefore, we select (i) dropout regularization with $p = 0.5$ for the CNN feature extractor for our final discriminator network.

The training loss plots are shown in Figure 7.6b. Compared with the previous experiment (Figure 7.5b), the generator loss shows more steady trend over 80 epochs of training. Additionally, the discriminator loss shows a decreasing trend, which was not that steep compared with the previous experiment, when no dropout in CNN was used. This indicates that the dropout regularization has a positive effect on balancing training of generator and discriminator. As a result, the test accuracy of 84.7% was achieved, which outperforms the best accuracy obtained in Section 6.2.3.

As a final note it is important to indicate that, even though our generator network was keeping up with the domain discriminator in terms of training loss, its loss started to increase after training *epoch* = 80. This indicates that domain discriminator starts over performing the generator at that point. Thus we further study different regularization techniques for label smoothing.



(a) Results of applying dropout regularization to the CNN feature extractor and classifier module



(b) Comparison between generator and discriminator losses during training, using dropout in UNet and CNN feature extractor, with L_1 loss $\beta = 50$. **Note:** networks were trained for 400 epochs in total until test accuracy did not increase anymore. The plots are zoomed for a better comparison with UNet regularization experiments.

Figure 7.6: Effect of discriminator regularization

It has been shown in [26] that smoothed domain labels prevents over-fitting of an artificial neural network by softening the training labels, in order to penalize overconfidence. Thus with this experiment we utilize a uniform domain label smoothing technique [26], in which one hot encoded label q is smoothed and can be defined as $\hat{q} = (1-\epsilon) * q + \epsilon \frac{1}{K}$, where ϵ is a smoothing strength and K is a number of classes. In this experiment, we use $\epsilon = 0.055$ and $K = 12$ as we have 11 subjects in our training set (see Table 6.1) and one as unknown, resulting in one hot encoded vector with smoothed value of 0.95 as ground truth.

The test accuracy with varying β of L_1, when smoothed labels are used for computing domain discriminator loss term during training are shown in Figure 7.7. As it can be seen, test accuracy fluctuates when β is varied from 0 to 200, showing slightly higher performance for $\beta = 10$ & $\beta = 0.5$ and reaching around 62.7% and 65.5%, respectively. Regarding the training loss, it was observed that discriminator loss decreased during training phase until around *epoch* = 20 (for both experiments showing the best accuracy), and then started to increase, indicating training instability. This indicates that label smoothing did not contribute to a better model performance in presence of domain change and it will not be utilized further in our experiments.

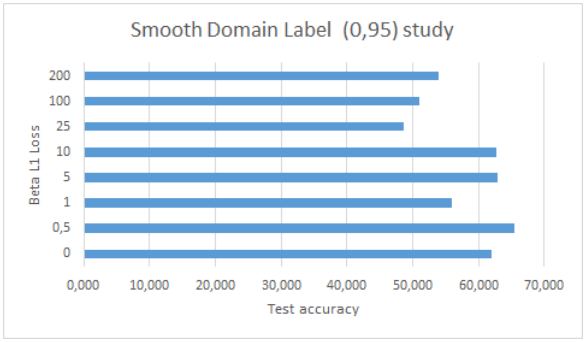


Figure 7.7: Effect of domain label smoothing

7.3 A Robust Adversarial Network against Domain Change

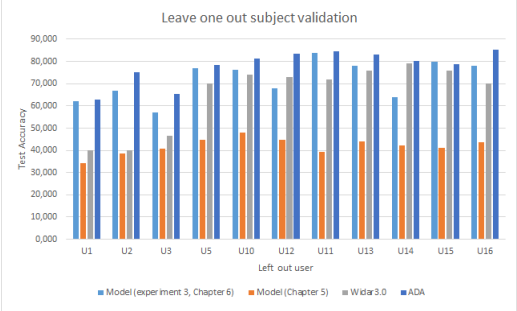
Based on results of the previous experiments, our final generator and discriminator networks look like Figures 7.2 and 7.3, with $\mathbf{L_1}$ $\beta = 50$ (Section 7.2.1), UNet dropout, with $p = 0.25$

(Section 7.2.2) and discriminator, using feature extractor from Section 6.2.3 and dropout, with $p = 0.5$ (Section 7.2.3). Figure 7.8 shows the overall performance of this model using leave-one-out subject/room validation. We compare the best performing models obtained in Chapters 5, 6, the adversarial domain adaptation network (proposed in Chapter 7), and Widar 3.0 [52].

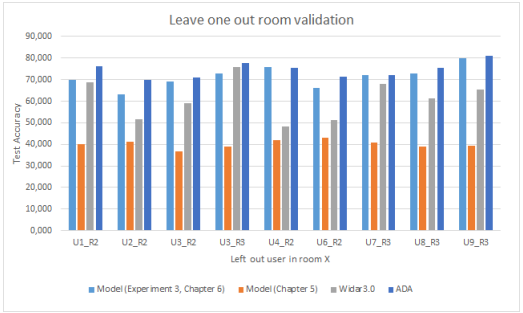
Results of leave-one-out validation on subject domain are shown in Figure 7.8a. In each experiment all subjects in room 1 except the one left out is used for training and the left out subject is used for testing. It can be seen that the worst performing model is the one obtained in Chapter 5, with an average test accuracy 41.9%. The Widar 3.0 model achieves an average test accuracy of 65.2% while the Bayesian optimized model obtained in Chapter 6 shows a higher performance, with an average test accuracy of 71.9%. Finally, even though a lower performing feature extractor was used in adversarial domain adaptation due to hardware limitations, it can be seen that its performance is by far the best. It outperforms our Bayesian optimized model (of Chapter 6) by approximately 6%, showing an average test accuracy of 78.05%.

Results of leave-one-out room validation on room domain are shown in Figure 7.8b. In each experiment CSI gesture samples of all subjects in room 1 were used during training and CSI samples of one chosen subject, but in different room were used for testing. Similarly as in leave-one-out subject validation in Figure 7.8a, the worst and the second worst performing models were the ones experimented in Chapter 5 and Widar3.0, with an average test accuracy of 40.1% and 61%, respectively. While our Bayesian optimized model of Chapter 6 outperforms Widar3.0 model by almost 10%, with an average test accuracy of 71.3%, the adversarial domain adaptation has by far the highest accuracy, with an average test accuracy of 74.5%.

Overall, adversarial domain adaptation model shows the highest average test accuracy in leave-one-out validation for both: subject and room domains. Since the domain labels were used as subject ID, the model shows on average better robustness with unseen subjects rather than unseen rooms.



(a) Leave-one-out subject validation, where x-axis represents left out subject in room 1 for testing. Data set used: room locations - all, face orientations w.r.t Tx - all, gesture sample repetitions - first 3.



(b) Leave-one-out room validation, where x-axis represents left user in room x for testing, e.g. U1_R2 - user 1 in room 2. Data set used: room locations - all, face orientations w.r.t Tx - all, gesture sample repetitions - first 3.

Figure 7.8: Leave-one-out subject/room validation

Chapter 8

Conclusions

Overall, with the accelerating development of learning-based methods, device free activity recognition systems, using WiFi CSI became possible. Based on the related work chapter 3, a wide range of learning based methods were developed over the past years, which achieve sufficiently high recognition performance. However, due to the issue of domain change discussed in chapter 1, these methods tend to show performance degradation. Thus researchers in the field are aiming to address the issue, not only by experimenting with different learning based models, but also proposing different and highly complex CSI data pre-processing.

After analysis of related work, we found some existing gaps in Section 3.3 and focused on filling some of the gaps, such as (i) highly complex CSI data pre-processing and (ii) model robustness in one particular domain. Subsequently, research questions were raised in Section 1.3, which were answered by chapters 5, 6 and 7, where we propose (i) a simplified CSI data pre-processing for CNN to function in presence of domain change, (ii) a fine-tuned CNN, which shows higher robustness than Widar3.0, when both using simplified set of operations for pre-processing in presence of domain change of subject and rooms, and (iii) a new adversarial domain adaptation model, which outperforms the (ii).

8.1 Future work

Although a lot of effort and time was dedicated in order to get some results, there is a number of gaps, which could be filled for the future work. Beginning with the CSI data, only amplitude was utilized, without analyzing and studying the phase information or even combination of both. Secondly, data pre-processing might be simplified even further by removing interpolation step, which was used for getting a constant CSI amplitude sample shape. This may be achieved by improving Convolutional Neural Network (CNN), which could be invariant to the input shape, by utilizing pooling operations.

Regarding the model with adversarial domain adaptation, only subject ID was used as domain labels. Perhaps more fine grained labels, such as subject orientation with respect to transmitter or location in the room might result in better performance. Finally, adversarial training stability might be improved and data efficiency could be studied in future work.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: http://tensorflow.org/.

[2] Md Zahangir Alom, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul A. S. Awwal, and Vijayan K. Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3), 2019. URL: https://www.mdpi.com/2079-9292/8/3/292, doi:10.3390/electronics8030292.

[3] Angelos Amanatiadis and Ioannis Andreadis. A survey on evaluation methods for image interpolation. *Measurement Science and Technology*, 20(10):104015, sep 2009. doi:10.1088/0957-0233/20/10/104015.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017. arXiv:1701.07875.

[5] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *CoRR*, abs/2003.05991, 2020. URL: https://arxiv.org/abs/2003.05991, arXiv:2003.05991.

[6] Jeroen Klein Brinke and Nirvana Meratnia. Scaling activity recognition using channel state information through convolutional neural networks and transfer learning. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, AIChallengeIoT'19, page 56–62, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3363347.3363362.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995. doi:10.1023/A:1022627411411.

[8] Leonid Datta. A survey on activation functions and their relation with xavier and he normal initialization. *CoRR*, abs/2004.06632, 2020. URL: https://arxiv.org/abs/2004.06632, arXiv:2004.06632.

[9] Oscar Day and Taghi M. Khoshgoftaar. A survey on heterogeneous transfer learning. *Journal of Big Data*, 4(1):29, Sep 2017. doi:10.1186/s40537-017-0089-0.

[10] Shi Dong, Ping Wang, and Khushnood Abbas. A survey on deep learning and its applications. *Computer Science Review*, 40:100379, 2021. URL: https://www.sciencedirect.com/science/article/pii/S1574013721000198, doi:https://doi.org/10.1016/j.cosrev.2021.100379.

[11] Hasmath Farhana Thariq Ahmed, Hafisoh Ahmad, Swee King Phang, Chockalingam Aravind Vaithilingam, Houda Harkat, and Kulasekharan Narasingamurthi. Higher order feature extraction and selection for robust human gesture recognition using csi of cots wifi devices. *Sensors*, 19(13), 2019. URL: https://www.mdpi.com/1424-8220/19/13/2959, doi:10.3390/s19132959.

[12] Peter I. Frazier. A tutorial on bayesian optimization, 2018. arXiv:1807.02811.

[13] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation, 2015. arXiv:1409.7495.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, October 2020. doi:10.1145/3422622.

[17] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL: http://arxiv.org/abs/1207.0580, arXiv:1207.0580.

[18] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *CoRR*, abs/2009.12836, 2020. URL: https://arxiv.org/abs/2009.12836, arXiv:2009.12836.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL: http://arxiv.org/abs/1502.03167, arXiv:1502.03167.

[20] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.632, doi:10.1109/CVPR.2017.632.

[21] Dehao Jiang, Mingqi Li, and Chunling Xu. Wigan: A wifi based gesture recognition system with gans. *Sensors*, 20(17):4757, Aug 2020. URL: http://dx.doi.org/10.3390/s20174757, doi:10.3390/s20174757.

[22] Wenjun Jiang, Chenglin Miao, Fenglong Ma, Shuochao Yao, Yaqing Wang, Ye Yuan, Hongfei Xue, Chen Song, Xin Ma, Dimitrios Koutsonikolas, Wenyao Xu, and Lu Su. Towards environment independent device free human activity recognition. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 289–304, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3241539.3241548.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. arXiv:1412.6980.

[24] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014. arXiv:1312.6114.

[25] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015.

[26] Weizhi Li, Gautam Dasarathy, and Visar Berisha. Regularization via structural label smoothing, 2020. arXiv:2001.01900.

[27] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2021. doi:10.1109/TNNLS.2021.3084827.

[28] Yongsen Ma, Gang Zhou, and Shuangquan Wang. Wifi sensing with channel state information: A survey. *ACM Comput. Surv.*, 52(3), June 2019. doi:10.1145/3310194.

[29] Yongsen Ma, Gang Zhou, Shuangquan Wang, Hongyang Zhao, and Woosub Jung. Signfi: Sign language recognition using wifi. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(1), March 2018. doi:10.1145/3191755.

[30] Anke Meyer-Baese and Volker Schmid. Chapter 7 - foundations of neural networks. In Anke Meyer-Baese and Volker Schmid, editors, *Pattern Recognition and Signal Analysis in Medical Imaging (Second Edition)*, pages 197–243. Academic Press, Oxford, second edition edition, 2014. URL: https://www.sciencedirect.com/science/article/pii/B9780124095458000078, doi:https://doi.org/10.1016/B978-0-12-409545-8.00007-8.

[31] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL: http://arxiv.org/abs/1301.3781.

[32] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL: http://arxiv.org/abs/1411.1784, arXiv:1411.1784.

[33] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814. Omnipress, 2010. URL: https://icml.cc/Conferences/2010/papers/432.pdf.

[34] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018. URL: http://arxiv.org/abs/1811.03378, arXiv:1811.03378.

[35] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Keras tuner. https://github.com/keras-team/keras-tuner, 2019.

[36] Sameera Palipana, David Rojas, Piyush Agrawal, and Dirk Pesch. Falldefi: Ubiquitous fall detection using commodity wi-fi devices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4), January 2018. doi:10.1145/3161183.

[37] Kun Qian, Chenshu Wu, Zheng Yang, Yunhao Liu, Fugui He, and Tianzhang Xing. Enabling contactless detection of moving humans with dynamic speeds using csi. *ACM Trans. Embed. Comput. Syst.*, 17(2), January 2018. doi:10.1145/3157677.

[38] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016. arXiv:1511.06434.

[39] L. ROSZKOWIAK, A. KORZYNSKA, J. ZAK, D. PIJANOWSKA, Z. SWIDERSKA-CHADAJ, and T. MARKIEWICZ. Survey: interpolation methods for whole slide image processing. *Journal of Microscopy*, 265(2):148–158, 2017. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jmi.12477, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/jmi.12477, doi:https://doi.org/10.1111/jmi.12477.

[40] Tony J. Rouphael. Chapter 3 - common digital modulation methods. In Tony J. Rouphael, editor, *RF and Digital Signal Processing for Software-Defined Radio*, pages 25–85. Newnes,

Burlington, 2009. URL: https://www.sciencedirect.com/science/article/pii/B9780750682107000035, doi:https://doi.org/10.1016/B978-0-7506-8210-7.00003-5.

[41] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. URL: http://dx.doi.org/10.1109/CVPR.2015.7298682, doi:10.1109/cvpr.2015.7298682.

[42] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, Jun 1999. doi:10.1023/A:1018628609742.

[43] Hasmath Farhana Thariq Ahmed, Hafisoh Ahmad, and Aravind C.V. Device free human gesture recognition using wi-fi csi: A survey. *Engineering Applications of Artificial Intelligence*, 87:103281, 2020. URL: http://www.sciencedirect.com/science/article/pii/S0952197619302441, doi:https://doi.org/10.1016/j.engappai.2019.103281.

[44] David Tse and Pramod Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, 2005. doi:10.1017/CBO9780511807213.

[45] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, Oct 2018. URL: http://dx.doi.org/10.1016/j.neucom.2018.05.083, doi:10.1016/j.neucom.2018.05.083.

[46] Zhengwei Wang, Qi She, and Tomás E. Ward. Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Comput. Surv.*, 54(2), February 2021. doi:10.1145/3439723.

[47] Zhihao Wang, Jian Chen, and Steven C. H. Hoi. Deep learning for image super-resolution: A survey. *CoRR*, abs/1902.06068, 2019. URL: http://arxiv.org/abs/1902.06068, arXiv:1902.06068.

[48] C. Xiao, D. Han, Y. Ma, and Z. Qin. Csigan: Robust channel state information-based activity recognition with gans. *IEEE Internet of Things Journal*, 6(6):10191–10204, 2019. doi:10.1109/JIOT.2019.2936580.

[49] Yang Xu, Wei Yang, Jianxin Wang, Xing Zhou, Hong Li, and Liusheng Huang. Wistep: Device-free step counting with wifi signals. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4), January 2018. doi:10.1145/3161415.

[50] J. Yang, H. Zou, Y. Zhou, and L. Xie. Learning gestures from wifi: A siamese recurrent convolutional architecture. *IEEE Internet of Things Journal*, 6(6):10763–10772, 2019. doi:10.1109/JIOT.2019.2941527.

[51] Yunze Zeng, Parth H. Pathak, and Prasant Mohapatra. Analyzing shopper's behavior through wifi signals. In *Proceedings of the 2nd Workshop on Workshop on Physical Analytics*, WPA '15, page 13–18, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2753497.2753508.

[52] Yue Zheng, Yi Zhang, Kun Qian, Guidong Zhang, Yunhao Liu, Chenshu Wu, and Zheng Yang. Zero-effort cross-domain gesture recognition with wi-fi. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, page 313–325, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3307334.3326081.

[53] Zheng Yang; Yi Zhang; Guidong Zhang; Yue Zheng. Widar 3.0: Wifi-based activity recognition dataset, 2020. URL: https://dx.doi.org/10.21227/7znf-qp86, doi:10.21227/7znf-qp86.

[54] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, 2017. doi:10.1109/ICCV.2017.244.

[55] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021. doi:10.1109/JPROC.2020.3004555.

[56] H. Zou, J. Yang, Y. Zhou, L. Xie, and C. J. Spanos. Robust wifi-enabled device-free gesture recognition via unsupervised adversarial domain adaptation. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8, 2018. doi:10.1109/ICCCN.2018.8487345.

[57] H. Zou, J. Yang, Y. Zhou, L. Xie, and C. J. Spanos. Robust wifi-enabled device-free gesture recognition via unsupervised adversarial domain adaptation. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8, 2018. doi:10.1109/ICCCN.2018.8487345.