People's Democratic Republic of Algeria.
Ministry of Higher Education and Scientific Research

Machine learning for Sign Language Recognition: Application on Smart Buildings

Submitted September 2020, in partial fulfillment of
the conditions for the award of the degree **Master Software engineering.**

Mohamed Mehdi BOURAHLA
ID: M201532027293

Supervised by Roufaida LAIDI and Abdallah KAMECH

Computer Science department
Saad Dahleb Blida University

Date: 23 / 09 / 2020

In front of the jury:

Hafida ABED

Fatima Zahra ZAHRA

Promotion: 2019/2020

Abstract

Recent advances in mobile computing, wireless sensing and communication technologies, consumer electronics have modernized our cities and living environments. Buildings, roads, and vehicles are now empowered with a variety of smart sensors and objects that are interconnected via machine-to-machine communication protocols, accessible via the Internet, to form what is known as the Internet of Things (IoT). The power of IoT expands when coupled with Machine Learning, since the later offer techniques that allow analyzing the vast amount of data generated by sensors and actuators. Smart buildings are an appealing example of IoT and machine learning applications offering higher energy saving and occupants satisfaction through dynamic control.

Vocal virtual assistants (e.g., Amazon Alexa, Google Home) are now a central component of the smart house. However, they are not adapted to deaf and mute people who communicate using sign language. Efficient alternative communication means inside the house are required to assist the interaction of deaf and hearing-impaired people.

The main goal of this thesis is to conceive and realize a solution based on machine learning for sign language recognition that allows the control of a smart home environment through gestures.

Keywords: Smart buildings, Machine learning, Sign language, Disabled people, Human-Computer interaction.

Résumé

Les progrès récents dans l'informatique mobile, la détection sans fil et les technologies de communication, l'électronique grand public ont modernisé nos villes et nos milieux de vie. Les bâtiments, les routes et les véhicules sont désormais dotés de divers capteurs et objets intelligents interconnectés par des protocoles de communication machine à machine, accessibles via Internet, pour former ce qu'on appelle l'Internet des objets (IoT). La puissance de l'IoT se développe lorsqu'il est couplé avec le Machine Learning, puisque ce dernier offre des techniques qui permettent d'analyser la grande quantité de données générées par les capteurs et les actionneurs. Les bâtiments intelligents sont un exemple attrayant d'applications d'IoT et d'apprentissage automatique offrant des économies d'énergie plus élevées et la satisfaction des occupants grâce au contrôle dynamique.

Les assistants vocaux virtuels (par exemple, Amazon Alexa, Google Home) sont désormais une composante centrale de la maison intelligente. Cependant, ils ne sont pas adaptés aux personnes sourdes et muettes qui communiquent en langage gestuel. Des moyens de communication alternatifs efficaces à l'intérieur de la maison sont nécessaires pour faciliter l'interaction des personnes sourdes et malentendantes.

L'objectif principal de cette thèse est de concevoir et de réaliser une solution basée sur l'apprentissage automatique pour la reconnaissance du langage des signes qui permet le contrôle d'un environnement domestique intelligent à travers des gestes.

Mots clés: Bâtiments intelligents, apprentissage automatique, langue des signes, personnes handicapées, interaction homme-machine.

ملخص

لقد حدّث التطور الأخير فى مجال الحوسبة المتنقلة والاستشعار اللاسلكى وتقنيات الاتصال, والإلكترونيات الاستهلاكية مدننا وبيئتنا المعيشية. أصبحت المبانى والطرق والمركبات الآن مزودة بمجموعة متنوعة من المستشعرات والأجسام الذكية المتصلة ببعضها عبر بروتوكولات الاتصال من جهاز إلى جهاز, والتى يمكن الوصول إليها عبر الإنترنت, لتكوين ما يُعرف بإنترنت الأشياء. تتوسع قوة إنترنت الأشياء عند اقترانها بتعلم الآلة, حيث إن أحدث تقنيات تتيح تحليل كمية كبيرة من البيانات التى تولدها أجهزة الاستشعار والمشغلات. تُعد المبانى الذكية مثالاً جذابًا على تطبيقات إنترنت الأشياء وتعلم الآلة التى توفر طاقة أعلى ورضا راكبى السيارة من خلال التحكم الديناميكى.

أصبح المساعدون الافتراضيين الصاخب (على سبيل المثال, Amazon Alexa و Google Home) الآن مكوناً أساسياً فى المنزل الذكى. ومع ذلك, فإنها غير مكيفة مع الصم وكتم الأشخاص الذين يتواصلون باستخدام لغة الإشارة. وسائل الاتصال البديلة الفعالة داخل المنزل مطلوبة للمساعدة فى تفاعل الصم وضعاف السمع.

والهدف الرئيسى لهذه الفرضية هو تصور وتحقيق حل يستند إلى تعلم الآلة للتعرف على لغة الإشارات التى تسمح بالتحكم فى بيئة منزلية ذكية من خلال الإيماءات.

الكلمات المفتاحية: المبانى الذكية , التعلم الآلى , لغة الإشارة , المعاقون , التفاعل بين الإنسان والحاسوب.

Dedications

I dedicate this thesis to my parents for whom a whole book of dedications would not be enough to express my gratitude to them. For all of their support, encouragement, love and sacrifices that have brought me to this point in my life and continue to keep me moving forward. May Allah preserve them as well as all moms and dads in the world.

To my brothers and beloved sister who always believed in me and saw me as the little computer kid in the family.

My dear friends Abdelouahab, Chakib, Chouaib, Hadjer, Mohamed, Oussama, Rym for their love and support, may they find here the expression of my gratitude.

All my CSC club family who over the past five years have taught me to overcome challenges and make a difference and have opened the door to many opportunities for me.

Acknowledgements

First of all, I would like to thank ALLAH the greatest for all the patience, health and strength he has given me to do this modest work. Every letter typed on this thesis and every information withheld is by his grace glory to him.

I would like to thank Ms. Roufaida LAIDI my supervisor at CERIST for her help, her infinite patience, her availability and her golden advice which fueled my reflection, and especially for her efforts and all her contributions and corrections along the way.

I would like to express my gratitude to my supervisor Mr. Abdallah KAMECHE at Saad Dahleb university for having mentored me for this project as well as for all the knowledge I have acquired from him.

I also thank all the professors of the computer science department of Saad Dahleb University and any contributor to the knowledge I have acquired in recent years without whom I will not be where I am.

My gratitude to our delegate Mr. Imade ANANE and to Ms. Rym BOUCHETERA for their support to all our promotion.

All my thanks to Mr. Chakib KERRI for supporting me and helping me prepare the presentation of this project. Thanks to Mrs. Imene HENNI MENSOUR for inspiring me and opening the door to many opportunities.

Contents

List of Tables

List of Figures

GENERAL INTRODUCTION

Context

In the last decade, technology has taken a massive step in the Smart Homes sector, with the development of mobile computing, wireless sensing, communication technologies, and smart sensors. Smart home technology is currently being implemented for the entire house, in particular, the kitchen and the living room. It facilitates users with security, comfortable living, and energy management features as well as added benefits for disabled individuals. Research has already provided for a smart connected house where several devices cooperate in pampering the wishes of users with little or no effort. The research done to date has focused on voice control of the house, with a simple voice command, for example, "Increase the temperature" the system would interact and increase the temperature of the room. However, other researchers have focused on the control of connected devices through gestures made by the user's hands, which is called Hand Gesture Recognition (HGR). The motivation of the researchers was not only to bring a new way of interacting with machines but rather to offer a way for disabled people to control their home like a non-disabled person.

According to the statistics of the World Federation of the Deaf and the World Health Organization, approximately 70 million people in the world are deaf-mute. A total of 360 million people are deaf, and 32 million of these individuals are children [1]. The majority of speech- and hearing-impaired people cannot read or write in natural languages [2]. Fortunately, hearing-impaired people use sign languages to communicate with each other. However, this language is unfamiliar to hearing people.

Recently, a significant amount of effort in human-computer interaction (HCI) has been dedicated to the development of user-friendly interfaces employing voice, vision, gesture, and other innovative I/O channels. HCIs allow users to freely control devices via simple operations without requiring the user's full attention. Therefore, HGR is an essential feature of HCIs because it allows users to control devices with simple hand gestures efficiently. A computer or machine's ability to understand the hand gestures is the key to unlocking other potential applications [3]:

- Sign language recognition (SLR)—Communication medium for the deaf. It consists

of several categories, namely fingerspelling, isolated words, a lexicon of words, and continuous signs.

- Robotics and Tele-robotic—Actuators and motions of the robotic arms, legs, and other parts can be moved by simulating a human's action.
- Games and virtual reality—Virtual reality enable realistic interaction between the user and the virtual environment. It simulates the movement of users and translates the movement in the 3D world.
- Human-computer interaction—Includes application of gesture control in the military and medical field, manipulating graphics, design tools, annotating, or editing documents.

Problem Statement

This work focuses on SLR. With the advance of technology, new systems based on machine learning can translate sign language into natural language. In the context of SLR the challenge would be to acquire the signs that the user produces and to predict in real-time which sign has been made and then to make an act of it. Our system will leverage signs produced by the user in a home environment to predict a sentence that represent an action on connected peripherals.

Thesis structure

To better present our work, we have organized this thesis as follow:

- Chapter 1: gives a background information about the main concepts related to our work namely sign language, smart homes and machine learning and deep learning. Furthermore, it establishes the context of the research, justifies the need for conducting the study, and summarizes what the study aims to achieve.
- Chapter 2: illustrates the process of HGR and presents three different approaches in the literature namely vision, sensor and wireless-based. Next, it gives the current state of the art of each approach. Finally, it compares and summarizes the existing approaches.
- Chapter 3: gives an overview about our solution to translate sign language into smart home action.

- Chapter 4: covers the implementation details and evaluates the performance of the proposed solution.

Chapter 1

Background and Context

1.1 Introduction

This chapter overviews the main concepts related to our work: namely sign language, smart homes, machine learning and deep learning. Furthermore, it establishes the context of the research, justifies the need for conducting the study, and summarizes what the study aims to achieve.

1.2 Sign Language

Sign Language (SL) is considered as the most organized and structured form out of various gesture categories. It is the native language for a deaf or a person with a communication disability. SL relies primarily on gestures to communicate rather than speech, combining the use of finger shapes, hand movements and facial expressions. There is no universal SL; different sign languages are used in different countries or regions. For example, British Sign Language (BSL) is a different language from American Sign Language (ASL), and Americans who know ASL may not understand BSL. The 2020 edition of Ethnologue [4] lists 144 sign languages, but there are more than that. For example, Rwandan Sign Language is not listed in the Ethnologue, although it is an established language within the country. Nevertheless, the most widely used sign language is ASL, used in the USA, Canada, parts of Mexico and, with modifications, in other countries in Central America, Asia and Africa. Moreover, ASL has, like English, become a colonizing language, with missionaries bringing it to Africa and the Philippines, among other places.

The classification system for sign languages of the world is outdated: it was put together in 1991, by Henri Wittmann [5]. He based it on the Ethnologue edition at the time (1988) which listed 69 sign languages. Wikipedia has a simplified list of language families, which is reflective of the state of knowledge we have today in SL linguistics. The most significant sign languages family is the French Sign Language (French: Langue des Signs Française (LSF)) which includes LSF and ASL.

Algerian Sign Language (French: Langue des Signs Algérienne (LSA)) is the sign language used by communication disability persons and their close relatives in Algeria. The LSA is linked to the LSF family and has no direct link with Arabic sign languages.

SL symbols can be classified into single-handed or double-handed signs, static or dynamic. For representing one handed signs, single dominant hand is used. It can be represented by any static gesture or a gesture with motion. They can further be classified as type 0 or type 1 sign. In type 0 sign, both the hands are active, whereas, in type 1 dominant hand is more active as compared to non-dominant hand. SL consists of manual and non-manual elements as well. In manual signs, only hands are used to express any sign. In non-manual signs body postures, mouth gestures and face expressions are used.

The significant deficiencies in this language are a lot of hand movements, a limited vocabulary, and learning difficulties [2]. Besides, SL is unfamiliar to those who are not deaf and mute, and people with disabilities face severe challenges in communicating with able individuals. This interaction barrier significantly affects the lives and social relations of deaf people.

1.3 Smart Homes

Human beings communicate with the surrounding environment, in multiple ways. They are aware of the environmental conditions and act, react or adjust accordingly. If the environment can reciprocate this behavior and adapt to human behavior, it can bring many benefits. Such actions, as well as introducing new programs and facilities, will automate different tasks that humans currently perform manually. A smart home can be defined as a residence equipped with computing and information technology that anticipates and responds to the needs of the occupants, working to promote their comfort, convenience, security and entertainment through the management of technology within the home and

connections to the world beyond [6].

The terms "Home Automation," "Connected Devices" and "Internet of Things" are distinct parts of the Smart Home concept. Home Automation is where home electrical devices are connected to a central system that automates those devices based on user input. Connected devices are smart electrical devices that are connected to the Internet and sensors. These devices know or anticipate what the user's needs. The Internet of Things (IoT) transforms an automated home into a smart home. With a combination of sensors, smart devices and systems, IoT connects everyday objects to a network, enabling them to complete tasks and communicate with each other, with no user input.

There are many types of smart homes in the current state with three major application categories. The first category provides services to residents by identifying and recognizing their actions or by detecting their health conditions. Such smart homes act as testing grounds for the collection of information to support the well-being of the residents of the home. The second category of smart homes is an emerging trend of smart homes, which can help the occupants to reduce the energy consumption of the house by monitoring and controlling of the devices and rescheduling their operating time according to the energy demand and supply. The third category is surveillance, where the data captured in the environment are processed to obtain information that may help to raise alarms, in order to protect the home and the residents from burglaries, theft and natural disasters such as floods.

1.4 Machine Learning

While classical programs use the rules to process the data, and output answers, Machine Learning (ML) programs receive data and the answers to output the rules (see Fig. 1.1). These rules can then be applied to unseen data to produce answers [7]. ML is training computers to learn from data collected through experience. It is the most appropriate alternative when the solution is not a priori known, but can only be developed using data or experience. It is typical in problems where human expertise does not exist, or when it is difficult to express it. Traditional domains where ML has mostly been used include speech/face-recognizing, language processing, spam filtering [8].

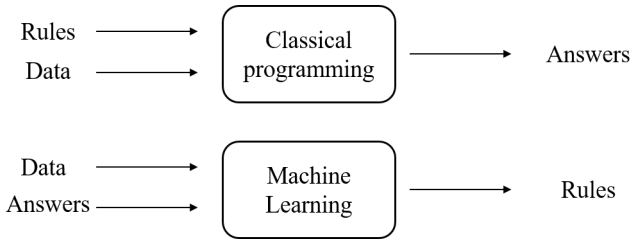


Figure 1.1: Machine learning: a new programming paradigm [7]

A machine learning algorithm aims to create a model, which is the representation of a phenomenon that the ML algorithm has learned. Hence, in order to train a model, a specific process is followed as shown in Fig. 1.2

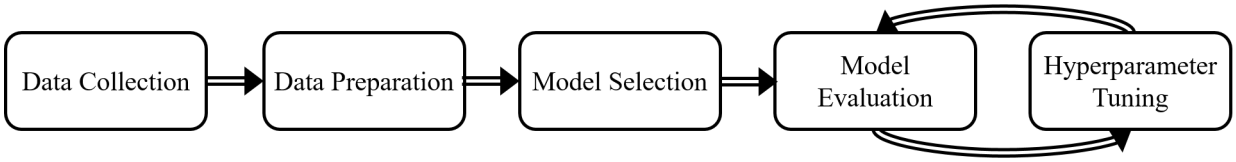


Figure 1.2: Machine learning algorithm process

1.4.1 Data Collection

The goal of data collection (also called "data acquisition") is to gather datasets, which are a set of data samples that contain features essential to solving a problem, and can be used to train machine learning models. There are mainly three approaches in the literature: data discovery, data augmentation, and data generation. Data discovery is when one wants to share or search for new datasets and has become essential, as more datasets are available on the Web and corporate data lakes. Data augmentation complements data discovery where existing datasets are enhanced by adding more external data. Data generation can be used when there is no available external dataset, but it is possible to generate crowdsourced or synthetic datasets instead [9].

1.4.2 Data Preparation

Data preparation (or "data preprocessing") aims at making the raw data gathered from the previous step more manageable and convenient to the training phase. It includes vectorization, normalization and dimensionality reduction.

Vectorization

It is the process of transforming data into tensors. In the context of binary image recognition, each image in the dataset is represented by a matrix. The size of this matrix depends on the number of pixels we have in any given image. Thus, the pixel values denote the intensity or brightness of the pixel.

Normalization

The data collected in a dataset usually comes from different sources, and each feature has its range of values. Heterogeneous data (a mix of large and small digits) may dramatically decrease the performance of some ML algorithms, hence the need for normalization. After normalization, each feature contributes approximately equally to the final distance.

Dimensionality reduction

Data reduction techniques reduce the representation of the data set into a smaller volume while keeping the integrity of the original data [10]. The goal is to produce identical (or almost identical) outcomes when applying the ML algorithm over reduced data and original data. Feature selection and feature extraction are two primary methods for dimensionality reduction. Feature selection removes irrelevant or redundant features. The goal is to find a minimum set of the most significant attributes, such as the resulting probability distribution is as close as possible to the original one. It facilitates the understanding of the selected pattern and accelerates the learning stage. On the other hand, feature extraction creates new features by merging the old ones that could better represent the decision boundaries in supervised learning [11].

1.4.3 Model Selection

One important step to select a model to fit the collected data is to categorize the problem. ML algorithms that we cover in our thesis are described in the following sections.

a) Supervised learning

Supervised learning algorithms experience a dataset containing features, and each example is also associated with a label or target. Supervised learning problems are further grouped into regression and classification problems. A classification problem is when the output variable is a category, such as “red/blue” or “disease/no disease.” A regression problem is when the output variable is a real value, such as “dollars” or “weight.” Some famous

examples of supervised machine learning algorithms are Linear regression for regression problems. Random forest for classification and regression problems. Support vector machines for classification problems. In the following of this section, we give an overview about the supervised learning algorithms mentioned in our thesis.

Support vector machine

Support Vector Machine (SVM) is a supervised ML model mostly used for classification, and sometimes for regression. Binary linear classification is performed by searching the hyper-plane that optimally differentiates two classes, i.e., that maximizes the distance to the nearest data point on each of its sides. Two parallel hyperplanes (that separate the two classes of data with maximum distance) can be selected if the training data are linearly separable. Non-linear classification can be assured by SVM and kernel trick with non-linear kernel functions. The most popular, general-purpose kernel functions are polynomials of degree q, Radial-Basis Functions (RBf) and sigmoidal functions [12]. There are many variants of SVM, such as SVC (Support Vector Clustering) that is used in unsupervised learning clustering problems, and SVR (Support Vector Regression) that is used for regression [13].

Decision trees

Decision Trees (DT) build classification or regression models in the form of a tree-like graph or a flowchart-like structure, where "tests" on attributes are represented by nodes, class labels (numerical data) by leaves, and the outcome of the test by branches. Carefully crafted questions on attributes of the test record are used in a series to feed the tree. The C4.5 algorithm is the most used in DTs. It adopts a top-down divide-and-conquer recursive approach. In every node, the algorithm chooses the best split among the features and possible split points. The split maximizing the normalized information gain is selected. This procedure is repeated for every node until reaching a "stop condition", which may be the minimum number of leaves in a node, the tree height, etc. [12]. The composition of multiple trees leads to more efficient algorithms such as Random Forest or Gradient Tree Boosting. Random Forests, on the other hand, are collections of independently trained DTs outputting the mode class of the classes in case of classification, or the mean of the individual trees predictions in case of regression. DTs are unstable as they tend to overfit the training data, i.e., small changes in the data lead to largely different trees. This model

provide more robust prediction compared to the use of single trees. The forest is called random because a random sample of the complete dataset is used to train each tree, which is known as "bootstrap aggregating or bagging" [12].

b) Unsupervised learning

Unsupervised learning algorithms experience a dataset containing many features, and then learn useful properties of the structure of this dataset. Unsupervised learning problems are grouped into clustering and association problems. A clustering problem discovers inherent groupings in data, such as grouping customers by purchasing behavior. An association rule learning problem finds rules that describe large portions of data, such as people that buy X also tend to buy Y. Some famous examples of unsupervised learning algorithms are K-Nearest Neighbor for clustering problems and Apriori algorithm for association rule learning problems. In the following of this section, we define unsupervised learning algorithms that are mentioned in our thesis.

K-Nearest Neighbors

The K-Nearest Neighbor (KNN) Algorithm is a method that does not require a training phase. The classification or the numerical value prediction for a new instance is made by searching through the entire dataset for the K closest instances using similarity measures (e.g., Euclidean distance). The output is summarized in these K nearest instances. For regression, the value of the new instance is the average on its K nearest neighbors. In classification, it is assigned to the class to which belong most of its K nearest neighbors (the mode class) [12].

Hidden Markov Model

Hidden Markov Model (HMM) is a probabilistic sequence model that associates a sequence of observations to a sequence of labels. It computes, for a given sequence, a probability distribution over possible sequences of labels for the purpose of select the best one. HMM is a Markov process with unobserved (i.e., hidden) states. Only the token output (that dependents on the state) is visible. HMM generates a sequence of tokens to inform about the sequence of states [12]. HMMs can be trained both in an unsupervised and a supervised form. First, when only observed output sequences are available for training, the model's conditional probabilities from this indirect evidence can be estimated through the Baum–Welch algorithm [14], a form of unsupervised learning, and an instantiation of

the expectation maximization algorithm [15]. When instead aligned sequences of hidden variables and output variables are given as supervised training data, both the output emission probabilities and the state transition probabilities can be straightforwardly estimated from frequencies of co-occurrence in the training data [16].

1.4.4 Model Evaluation

In machine learning, the goal is to achieve models that generalize, hence it is crucial to be able to measure the generalization power of the model reliably. Evaluation techniques test the model's performance on unseen data. Data is split into three sets: training, validation, and test. The training and validation sets are used during the training to tune the hyperparameters, while the test set evaluates the final model. Classic evaluation techniques are Simple holdout validation, K-fold cross validation. Therefore, model evaluation metrics are required to quantify model performance. The choice of evaluation metrics depends on a given machine learning task (such as classification, regression, clustering). Classification accuracy and confusion matrix are standard metrics on classification tasks, whereas Root Mean Square (RMS) for regression tasks.

The fundamental issue in machine learning is the balance between the optimization and the generalization. The optimization refers to the process of adjusting a model to get the best performance possible on the training data. In contrast, the generalization refers to how well the trained model infers on unseen data. At the beginning of training, they are correlated, and the model is said to be underfitting because it has not learned all relevant patterns in the training data yet. However, when the generalization stops improving, the model is overfitting, i.e., it learns patterns specific to the training data, but misleading or irrelevant for new data. To avoid overfitting, it is recommended to increase the size of the training set [7].

1.4.5 Hyperparameter Tuning

In machine learning, we have both model parameters and parameters we tune to make the model train better and faster. These tuning parameters are called hyperparameters [17]. Hyperparameters tuning is an iterative process in the machine learning algorithm life cyle that uses the feedback of the model's performance on the validation data to set the most optimal hyperparameters for the learning algorithm. The most commonly used

techniques for searching the best configuration are Grid search and Random search. For example, the main hyperparameter of the SVM is the kernel. It maps the observations into some feature space. Ideally the observations are more easily (linearly) separable after this transformation. There are multiple standard kernels for these transformations, e.g. the linear kernel, the polynomial kernel and the radial kernel. The choice of the kernel and their hyperparameters affect greatly the separability of the classes (in classification) and the performance of the algorithm.

1.5 Deep Learning

For decades, the machine learning system required careful engineering to design a feature extractor that transformed the raw data into a suitable internal representation from which the learning subsystem, could detect patterns in the input. Deep learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transforms the representation at one level into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned. For classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations. The key aspect of deep learning is that human engineers do not design these layers of features; they are learned from data using a general-purpose learning procedure [18]. In the following of this section, we explain the major components of deep learning namely neurons, activation functions and artificial neural networks. Therefore, we give an intuition about Convolutional neural networks and Recurrent neural networks.

1.5.1 Neurons

Artificial neural networks were inspired by the neural architecture of a human brain, and like in a human brain, the basic unit is called an Artificial Neuron. It is a mathematical function that takes one or more inputs that are multiplied by values called weights and added together. This value is then passed to a function, known as an activation function, to become the neuron's output. As shown in Fig. 1.3.

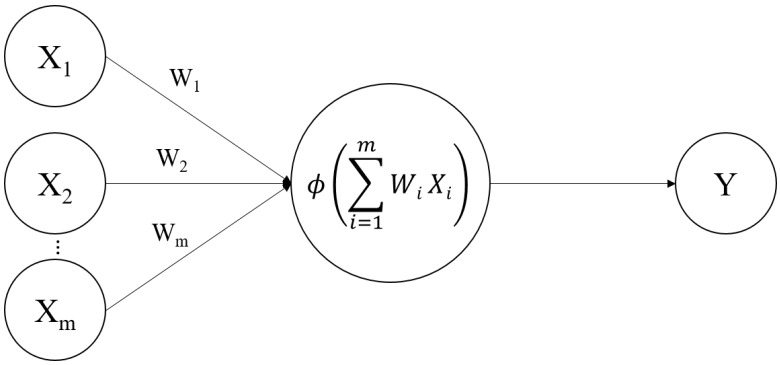


Figure 1.3: Artificial neuron

1.5.2 Activation Functions

As seen above, an activation function is a function applied by a neuron to determine the output of the neural network. It maps the resulting values in between 0 to 1 or -1 to 1 depending upon the function. Indeed, the activation functions are mainly divided on the basis of their range or curves. The following table lists the mainly used activation functions namely Threshold function (or Binary step), Sigmoid, Tanh (or Hyperbolic Tangent) and Rectified Linear Unit (ReLU).

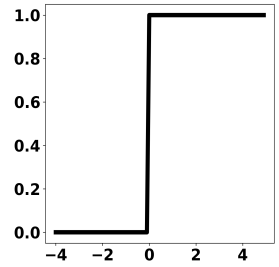
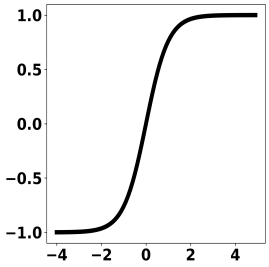
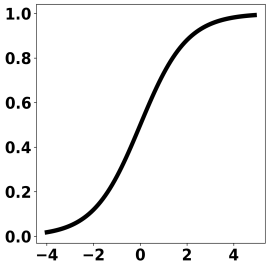
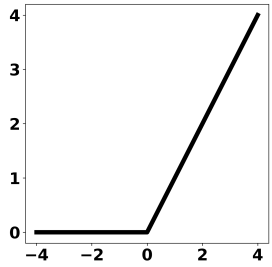
Threshold	Tanh	Sigmoid	ReLU
$g(z) = \begin{cases} 1 \ \ if \ \ z \geq 0 \\ 0 \ \ if \ \ z < 0 \end{cases}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g\,(z) = \frac{1}{1+e^{-z}}$	$g\,(z) = \max(0, z)$
			

Table 1.1: The commonly used activation functions

Among the four activation functions in the latter table, the default recommendation is to use ReLU. As for the output neurons, Sigmoid is recommended in binary classification tasks. Nevertheless, in multi-classification problems, Softmax activation function is used, which can be seen as a generalization of the sigmoid function [19].

1.5.3 Artificial Neural Networks

The ANN architecture is composed of an input layer, an output layer and at least one hidden layer, which are fully connected; each neuron in one layer connects with a weight to every neuron in the next layer. At each layer, there is also a bias that can help better fit the data. More layers generally increase the depth of the network (Hence the term deep learning) and allow it to provide different levels of representation and feature extraction. The input layer dimension represents the amount of feature that every data sample has. As for the output layer dimension, it contains only one neuron for binary classification and regression tasks and N neurons for N-classification problems.

Updating the weights is the primary way the neural network learns new information. Initially, the weights of the network are assigned random values, so the network merely implements a series of random transformations. The process of learning is re-adjusting the weights and biases, this helps the model to learn which features are related to which outcomes, and adjusts the weights and biases accordingly. Neural networks learn these relationships blindly by making a guess based on the inputs and weights and then measuring how accurate the results are. A loss function (also called a cost function) quantifies how close a given neural network is to the ideal toward which it is training. It is a metric based on the error observed in the network's predictions. The goal is finding the parameters (weights and biases) that will minimize the "loss" incurred from the errors. In this way, loss functions help reframe training neural networks as an optimization problem [17]. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. It is an iterative optimisation algorithm used to find the minimum value for a function.

In deep learning, the process of entering information into the input layer and applying transformations to get the output value and calculating the loss function is referred as forward propagation. At the opposite, the process of using gradient descent on the weights to minimize the error on the output of the network is referred as Backpropagation.

1.5.4 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are designed to process data that comes in the form of multiple arrays, for example, a colour image composed of three 2D arrays con-

taining pixel intensities in the three colour channels. Many data modalities are in the form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images.

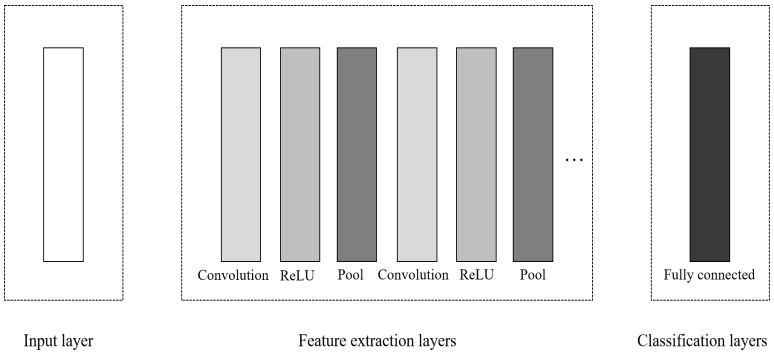


Figure 1.4: Convolutional neural network architecture

The architecture of a typical CNN (Fig. 1.4) is structured as a series of stages. The first few stages in charge of feature extraction are composed of two types of layers: convolutional layers and pooling layers. The second stages are classification layers in which we have one or more fully connected layers to take the higher-order features and produce class probabilities or scores.

Convolutional layers

A convolution is defined as a mathematical operation describing a rule for how to merge two sets of information. It takes input, applies a convolution kernel, and gives us a feature map as output. The convolution operation is known as the feature detector of a CNN. The input to a convolution can be raw data or a feature map output from another convolution [17]. We commonly refer to the sets of weights in a convolutional layer as a filter. This filter is convolved with the input and the result is a feature map. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of filters. All units in a feature map share the same filter and different feature maps in a layer use different filters. The result of this local weighted sum is then passed through a non-linear activation function such as a ReLU.

Pooling layers

Pooling layers are commonly inserted between successive convolutional layers to reduce the data representation progressively over the network and help control overfitting. For instance, max pooling computes the maximum of a local patch of units in one feature map independently. With a (2×2) size, the max pooling operation is taking the largest of four numbers in the filter area. Neighbouring pooling units take input from patches that are shifted by more than one row or column, thereby reducing the dimension of the representation and creating an invariance to small shifts and distortions.

Backpropagating gradients through a CNN is as simple as through a regular deep network, allowing all the weights in all the filters to be trained.

1.5.5 Recurrent Neural Network

In practice, neural networks take a fixed size input and use it to predict a fixed size output. For example, a 256x256 image to predict a single Boolean output. Simple neural networks like these cannot accept variable length entries like text, or an image of unknown size or a sentence of unknown length. There are different techniques for building networks to take inputs of varying sizes, including Recurrent Neural Networks (RNNs). RNNs are another class of neural networks that dominate ML problems that involve sequences of inputs. It is a type of neural network where the connections between units form a directed cycle. This creates an internal state of the network, which allows it to exhibit dynamic behaviour. Unlike neural networks, RNNs can use their internal memory to process and work on arbitrary sequences of inputs. This makes them applicable to tasks such as non-segmented connected handwriting recognition, voice recognition, natural language processing, machine translation, etc. RNNs process an input sequence one element at a time, maintaining in their hidden units a 'state vector' that implicitly contains information about the history of all the past elements of the sequence.

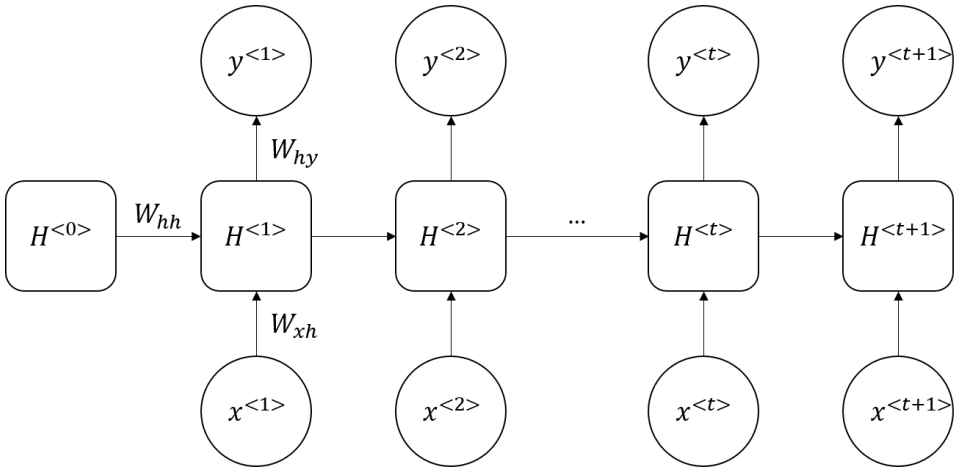


Figure 1.5: Recurrent neural network architecture

A simple RNN has three layers i.e. input layer, hidden recurrent layers and the output layer. At each time step of sending input through a recurrent network, nodes receiving input along recurrent edges receive input activations from the current input vector and from the hidden nodes in the network's previous state. The output is computed from the hidden state at the given time-step. The previous input vector at the previous time step can influence the current output at the current time-step through the recurrent connections [17]. At each time step t, the hidden state $H^{<t>}$ and the output $y^{<t>}$ are computed as follow:

$$H^{<t>} = g_1\left(W_{hh}.H^{<t-1>} + W_{xh}.x^{<t>} + b_h\right) \tag{1.1}$$

$$y^{<t>} = g_2\left(W_{hy}.H^{<t>} + b_y\right) \tag{1.2}$$

Where g_1 , g_2 are activation functions, b_h , b_y are biases parameters and W_{hh} , W_{xh} , W_{hy} are weights parameters.

Forward propagation in a RNN is relatively straightforward. Backpropagation through time is actually a specific application of back propagation in RNNs. It requires expanding the network one time step at a time to obtain the dependencies between model variables and parameters. Then, based on the chain rule, we apply backpropagation to compute and store gradients. However, since sequences can be rather long, the dependency can be rather lengthy. For instance, for a sequence of 1000 characters, the first symbol could potentially have significant influence on the symbol at position 1000. This is not compu-

tationally feasible and it requires over 1000 matrix-vector products before we would arrive at that very elusive gradient [20]. The basic problem is that gradients propagated over many stages tend to either vanish or explode. The most effective way to get around this issue is to use the Long Short Term Memory or Gated Recurrent Unit variants of RNNs.

Gated Recurrent Unit

The main difference between RNNs and Gated Recurrent Units (GRUs) is that the latter support gating of the hidden state. It means that there is a dedicated mechanism for when a hidden state should be updated and when it should be reset. These mechanisms are learned and they address the vanishing and exploding gradient issues. For instance, if the first symbol is of great importance it learns not to update the hidden state after the first observation. Likewise, it learns to skip irrelevant temporary observations. Last, it learns to reset the latent state whenever needed.

At each time step, we will consider overwriting the hidden state with $\widetilde{H}^{<t>}$ that represent a candidate to replace $H^{<t>}$. Knowing that $\widetilde{H}^{<t>}$ is defined as follow:

$$\widetilde{H}^{<t>} = \tanh\left(x^{<t>}.W_{hx} + (R^{<t>} \bigodot H^{<t-1>}).W_{hh} + b_h\right) \tag{1.3}$$

Where W_{hx} , W_{hh} are weights parameters, b_h is a bias parameter and $\bigodot$ denotes the Hadamard product between two vectors and $R^{<t>}$ is a gate in the GRU unit which is responsible for reinitialization. This variable comes from a sigmoid function that returns values between 0 and 1, if the value of $R^{<t>}$ equal to 0 this would mean that we want to reset the memory cell, and if it were equal to 1, this would mean that we want to keep the information received from the previous states. $R^{<t>}$ is defined by the following formula:

$$R^{<t>} = \sigma\left(x^{<t>}.W_{xr} + H^{<t-1>}.W_{hr} + b_r\right) \tag{1.4}$$

Where W_{xr} , W_{hr} are weights parameters and b_r is a bias parameter. As for $H^{<t>}$, it is defined by the following formula:

$$H^{<t>} = U^{<t>} \bigodot H^{<t-1>} + \left(1 - U^{<t>}\right) \bigodot \widetilde{H}^{<t>} \tag{1.5}$$

Where $U^{<t>}$ is the second gate of the GRU unit that is responsible for updating the

memory cell and assigning it a new value. Such as the reset variable, the update variable comes from a sigmoid function, if the value of $U^{<t>}$ is equal to 0 the old value of the memory cell is overwritten and is replaced by the candidate value $\widetilde{H}^{<t>}$, and if the value of $U^{<t>}$ is equal to 1, no change takes place. $U^{<t>}$ is defined by the following formula:

$$U^{<t>} = \sigma\left(x^{<t>}.W_{xu} + H^{<t-1>}.W_{hu} + b_u\right) \tag{1.6}$$

Where W_{xu} , W_{hu} are weights parameters, b_u is a bias parameter.

Long Short Term Memory

Developed by Hochreiter and Schmidhuber in 1997 [21], Long Short Term Memory (LSTM) shares many of the properties of the GRU. It introduces three gates: the input gate, the forget gate, and the output gate. In addition to that, it introduces the memory cell that has the same shape as the hidden state. The motivation for such a design is the same as GRUs, namely to be able to decide when to remember and when to ignore inputs in the latent state via a dedicated mechanism.

At each time step t, the introduced LSTM gates namely input gate, forget gate and output gate are respectively computed as following:

$$I^{<t>} = \sigma\left(x^{<t>}.W_{xi} + H^{<t-1>}.W_{hi} + b_i\right) \tag{1.7}$$

$$F^{<t>} = \sigma\left(x^{<t>}.W_{xf} + H^{<t-1>}.W_{hf} + b_f\right) \tag{1.8}$$

$$O^{<t>} = \sigma\left(x^{<t>}.W_{xo} + H^{<t-1>}.W_{ho} + b_o\right) \tag{1.9}$$

Where $W_{xi},\ W_{xf}, W_{xo}$ and $W_{hi},\ W_{hf}, W_{ho}$ are weight parameters and b_i, b_f, b_o are biases parameters.

The first step is to decide what information to throw away from the cell state. This decision is made by the forget gate layer. It looks at $H^{<t>}$ and $x^{<t>}$, and outputs a number between 0 and 1 for each number in the cell state $C^{<t-1>}$. A 1 represents “keep it” while a 0 represents “forget it” . The next step is to decide what new information to store in the cell state. This has two parts. First, the input gate decides which values to update. Next, a tanh gate creates a vector of new candidate values, $\widetilde{C}^{<t>}$, that could be

added to the state.

$$\widetilde{C}^{<t>} = \tanh\left(x^{<t>}.W_{xc} + H^{<t-1>}.W_{hc} + b_c\right)) \tag{1.10}$$

Where W_{xc} and W_{hc} are weights parameters and b_c is a bias parameter.

Therefore, It updates the old cell state, $C^{<t-1>}$, into the new cell state $C^{<t>}$ using the result of the previous computation:

$$C^{<t>} = F^{<t>} \bigodot C^{<t-1>} + I^{<t>} \bigodot \widetilde{C}^{<t>} \tag{1.11}$$

If the forget gate is always approximately 1 and the input gate is always approximately 0, the past memory cells $C^{<t-1>}$ will be saved over time and passed to the current time step. Finally, it decides to output. This output will be based on the cell state, but will be a filtered version. First, the output gate decides what parts of the cell state to output. Then, it puts the cell state through tanh, and multiply it by the result of the output gate.

$$H^{<t>} = O^{<t>} \bigodot \tanh\left(C^{<t>}\right) \tag{1.12}$$

1.6 Context

SLR is a collaborative research area that involves machine learning, computer vision, natural language processing and linguistics. Its objective is to build various methods and algorithms in order to identify already produced signs and perceive their meaning. SLR systems are HCI based systems that are designed to enable effective and engaging interaction. Such systems can be deployed in buildings to offer disabled and deaf persons the possibility to control connected devices as able individuals with voice commands.

Chapter 2

Literature Review for Sign Language Recognition

2.1 Introduction

In literature, three main categories for HGR can be distinguished: vision, sensors, and wireless-based approaches. The vision-based class makes use of video camera(s) to capture the images of hands, which are then processed and analyzed using computer vision techniques. This type of HGR is simple, natural, and convenient for users and is currently the most popular. However, several challenges should be addressed, for instance, illumination change, background clutter, partial or full occlusion, and the risks related to cameras and other recording devices deployment on personal privacy [22]. This approach is the object of Sec. 2.2.

Sensor-based approaches generally rely on the use of sensors physically attached to the user. These sensors collect the position, the motion, and the trajectories of the fingers and the hands. Features such as the flex angle of digits, the orientation, and the absolute position of the hands are often in 3D space. Hence, it contains the depth information useful in telling distance of gesture away from the sensors. These instruments are set up before the recognition, and these often limit the approaches to a laboratory setup [3]. The main disadvantage of this approach is the need to continuously wearing a glove or a bracelet by the user, which is not comfortable for daily life. Sec. 2.3 explains each stage of the sensor-based approach and shows how the researchers have overcome this constraint.

Finally, wireless-based (also called "device free") gesture recognition systems do not require the user to install camera infrastructures or to wear sensors. It relies on a particular architecture of transmitter and receiver, knowing that the transmitter broadcasts a signal (e.g. Wi-Fi), and the receiver initiates a process that will analyse how the wireless signals are reflected by surrounding objects. From there, it can be used to recognize hand and finger gestures. Still, those systems face several challenges, for example, how to deal with other humans in the environment, long distances, and obstacles. Sec. 2.4 reviews solutions in the wireless-based category and describes the latest development in the area.

2.2 Vision-Based Gesture Recognition

Many contributions considered gesture recognition by analysing visual frames. Researchers studied both static gestures using single frames of images and dynamic gestures using videos, which are continuous frames of images. The goal in these works was to minimize the system response time, increase identification accuracy, recognize more gestures, and enhance robustness. This approach has the advantage of being well known to the public. For example, Kinect from Microsoft [23] used to identify players' movements. The familiarity of the users with this type of input device makes their integration easier. However, the variety of gestures included in SL is the main challenge in vision-based hand gesture recognition. Recognizing a large number of gestures requires handling multiple Degrees of Freedom (DoF), a huge variability of the 2D image based on the point of view of the camera (even with the same gesture), various silhouette dimensions (i.e. spatial resolution) and many temporal aspect measurements (i.e. the variability of gesture speed). The accuracy-performance-usefulness trade-off has to be adjusted according to the type of application, the expense of the solution and different requirements such as real-time performance, robustness, scalability and consumer freedom [24]. The process of gesture recognition in the vision-based approach is composed of the following steps: the data acquisition, the image processing, the segmentation and tracking, the dimensionality reduction, and the classification (as shown in Fig. 2.1). In the following, we explain each step by referring to the most recent solutions proposed in this approach.

Figure 2.1: Vision based hand gesture recognition process

2.2.1 Data Acquisition

Images or videos of the hand gestures are captured through:

- Single-camera—Webcam, video camera and smartphone camera [25] [26]
- Stereo-camera—using multiple monocular cameras to provides depth information such as Kinect [27] [28]
- Invasive techniques—In addition to cameras, some authors include body markers such as coloured gloves, wrist bands, and LED lights [26], [29].

2.2.2 Image Processing

In this section, image processing is limited to the modifications made on images or video to improve the overall performance and does not include segmentation and tracking, which are discussed independently in the next two sections. Median and Gaussian filters are one of the commonly used techniques to reduce noises in images or videos [3].

Pansare et al. [30] first applied gray threshold on skin pixels with specific probability, then the filtering of gray image is done using median filter for preserving the edges followed by Gaussian filter. Oyedotun et al. [31] obtained binary photos (black and white) by thresholding the original images at a 0.5 gray level. Furthermore, the resulted binary images are filtered using a median filter to remove the noise. In [28], the joint bilateral filtering algorithm is used to filter the collected gesture images and improve the quality of the image.

2.2.3 Segmentation

Segmentation (also called "Hand detection") splits images into distinct parts where the Region of Interest (RoI) is separated from the rest of the picture [3]. In our context, the RoI is the hand. To this purpose, several features and their combinations are utilized. Such features are skin color, shape, motion, and anatomical models of hands.

In [32], the hand segmentation follows three steps: 1) converting the image from RGB to YCbCr, 2) extracting the RoI using a skin segmentation algorithm, 3) using a threshold to transform the image pixel values into binary values. In [26], the authors used red gloves to capture images. Therefore, the segmentation is done by extracting the red band of the RGB image to isolate it from the background. Then edge detection is performed using Sobel edge filter [33]. However, color-based methods face the challenge of removing other parts of the user's body with similar colors, such as the face and arms. As a solution, [34] used the Viola-Jones method [35] to detect the user's face, replace it with a black circle, and detect the skin area using the HSV. In [36], the face of the user is detected and removed from the second frame of the video using the Viola-Jones algorithm. Next, skin filtering is performed to obtain the skin-colored objects in the frame. On the other hand, a three-frame differencing algorithm is applied to the first three frames. The results of the skin filtering and three-frame differencing are combined to obtain the desired hand from the background.

2.2.4 Tracking

Tracking is only concerned by dynamic HGR since it is about the frame-to-frame correspondence of the segmented hand regions or features, it is the process of analysing sequential video frames and output the movement of the hand between the frames. The importance of robust tracking is twofold. First, it provides the inter-frame linking of hand/finger appearances, giving rise to trajectories of features in time. These trajectories carry essential information and might be used either in a raw form (e.g., virtual drawing) or after further analysis (e.g., recognition of some hand gestures). Second, in model-based methods, tracking provides a way to maintain estimates of the model's variables and features that are not observable all the time [24]. Typical hand tracking algorithms are optical flow, CamShift, Kalman filtering, particle filtering, condensation algorithm, etc. These algorithms are focused on specific hypotheses, and often, the combination of multiple algorithms produces better results [22].

[37] Used a Kalman filter and hand blobs analysis for hand tracking to obtain motion descriptors and hand regions. [38] Combine optical flow, blob analysis, and noise filtering techniques, including dilation and erosion operations, to recognize six hand gestures. In their experimental results, the authors found that their algorithm performs good results at

detection, tracking, and classification. [36] Performed hand tracking using the modified Kanade-Lucas-Tomasi (KLT) feature tracker. Velocity and orientation were added to remove the redundant feature points. In their experimental results, they compared with CamShift and their proposed tracking algorithm was more robust.

2.2.5 Dimensionality Reduction

Before feeding the data collected from images to a classifier, a dimensionality reduction step is required. Since for images, we think of the number of features as the number of pixels. Therefore, for a 64x64 image we have 4096 features. Such amount of features could slow down the system. The most commonly used algorithm to extract relevant features are Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Shift-Invariant Feature Transform (SIFT), and Speeded Up Robust Feature (SURF).

[32] used the PCA method for feature extraction to reduce the dimensionality of the image while preserving much of the information. [34] Used the SIFT algorithm to extract the key points (vectors) for each training image. However, some authors rely on neural networks to classify gestures, and this method does feature extraction automatically. In [36], ANalysis of VAriance (ANOVA) and Incremental Feature Selection (IFS) techniques were used to select the optimal features from the 44 existing features. [25] Used Histogram of Oriented Gradients (HOG) feature extraction technique to determine the intensity gradients distribution (i.e. edge directions) in the given image, which describes the appearance and shape of the object in the picture.

2.2.6 Classification

The overall goal of HGR is the interpretation of the semantics that the hand(s) location, posture, or gesture conveys. Vision-based techniques are classified under static and dynamic gestures. The classification uses the information extracted and processed by previous steps and outputs the recognition results.

[31] Applied a CNN with two hidden layers to recognize 24 ASL static signs obtained from a public database, a batch size of 5 is used to achieve stochastic gradients computations for optimizing the mean-square-error cost function. The authors achieved 91.33% accuracy on testing data. Singha et al. [36] merged three classifiers: ANN, SVM and k-Nearest Neighbours (kNN) to build a dynamic HGR system. The ANN includes one

hidden layer. Its input layer consists of 40 neurons, which represent the 40 features, and the output layer has 40 neurons expressing 40 gesture classes. The results of the individual classifiers were combined to get the classifier fusion result. Majority voting technique were used to combine the different classifiers. Their experiment achieved an accuracy of 92.23%. [37] Performed gesture recognition using Pseudo two Dimensions Hidden Markov Models (P2-DHMMs) to classify 36 gestures and achieved an average accuracy of 98%. The authors deduced that P2-DHMMs perform much better when the recognition task contains various gestures with varied sizes.

2.2.7 Results

In Tab. 2.1, we summarize the results obtained by various researchers in HGR based on vision, where the lines represent the works, and the columns are the stages and methods used by each work.

Ref	Data acquisition	Database	Image processing	Segmentation & Tracking	Dimensionality reduction	Classification	Acc.
[25]	Webcam	4 static	Cropping	Skin filtering	HOG	kNN	81%
[28]	Kinect	5 static	Joint Bilateral filtering	Sking filtering		CNN	98.52%
[34]	Video camera	6 static	Resizing, Converting to portable gray map	Skin filtering, Viola-Jones method	SIFT	SVM	96.23%
[36]	Webcam	40 dynamic		Skin filtering, Viola-Jones method, KLT	ANOVA, IFS	SVM, ANN, kNN	92.23%
[26]	Webcam, Red glove	6 static	Cropping	Sobel edge filter		ANN	89.48%
[37]	Video camera	36 dynamic		Kalman filter, Blob analysis	Discrete cosine transform	P2-DHMMs	98%
[29]	Video camera, green background, green wrist	26 static	Cropping	Skin filtering	Discrete wavelet transform and F-ratio	SVM	98.64%
[39]	Kinect	24 static		Per-pixel classification	Distance adaptive scheme	Random forest	90%

Table 2.1: Results obtained by various vision-based gestures recognition

2.2.8 Discussion

Among the works cited in the previous table, only [25], [26] and [34] do not concern the recognition of sign language. On the other hand, [28], [29] and [39] used the signs of the alphabet and [36], [37] used the alphanumeric signs. In ASL, letters are expressed with static signs, with the exception of "j" and "z" which are expressed with dynamic signs. However in [29], the authors considered all letters as static signs. A confusion could arise between the letters "i" and "j" which both have the little finger raised. Tab. 2.1 cites two dynamic HGR works [36], [37].

Although widely used, vision-based SLR techniques face many drawbacks. The presented works tried to overcome them by improving one or many steps of the HGR process. In SL, a single term may be expressed by a combination of more than a single gesture. By consequence, works based on static gestures are not suited such terms. For example, the word "Light" start with closed fingers pointing downwards, then spreading them. In addition, for such solutions to be efficient, the user should face the camera permanently. Thus, complete visual coverage is required. The privacy and low-cost requirements make these solutions unadapted to smart buildings.

2.2.9 Conclusion

This section provides an overview of the steps in the gesture recognition process using the vision approach, namely, data acquisition, image processing, segmentation and tracking, dimensionality reduction, and classification. HGR faces many obstacles, including poor lighting conditions, camera's inability to capture dynamic gesture in focus, occlusion due to finger movement, color variations due to lighting conditions, which need to be handled in the processing steps. The Microsoft Kinect device provides an effective way to solve the above issues and provides the skeleton for more convenient hand segmentation and tracking. However, it elevates the cost of the solution. We consider the vision-based gesture recognition not adaptable to the smart home context due to the privacy issues that may arise from placing cameras inside the house.

2.3 Sensor-Based Gesture Recognition

Sensor-based solutions use sensors physically connected to the users that capture finger and hand location, movement, and trajectories. Features such as the flex angle of fingers, the orientation, and the absolute position of the hand are often in 3D space. This approach, however, requires the users to wear a glove or a wrist (depending on the type of sensors) connected to the user's arm with sensors, this requirement tends to limit the approach to a laboratory setup [3]. As a solution, some researchers tried to give the user more freedom and less invasive possibilities. This section presents works from the sensor-based hand gesture recognition category following the steps shown in Fig. 2.2.

Figure 2.2: Sensor-based gesture recognition process

2.3.1 Data Acquisition

The sensor-based approach requires the use of sensors to capture the motion, position, and velocity of the hand. For that, different types of sensors are used:

- Inertial Measurement Unit (IMU)—Measure the acceleration, the location, degree of freedom and acceleration of the fingers. Which requires the use of a gyroscope and an accelerometer [40], [41].
- Electromyography (EMG)—Measures the human muscle's electrical pulses and harness the bio-signal to detect finger movements [42], [43].
- Others—Utilizes flex sensors, mechanical, electromagnetics, and haptic technologies [44].

2.3.2 Signal Preprocessing

Data collected from various sensors contains some degree of erroneous, noisy, and redundant information caused by dead batteries, sensor read failures and loss of intermittent communication. Signal preprocessing operations mainly include Data cleaning techniques to remove the artifacts and outliers, in addition to data interpolation to cope with miss-

ing readings and data transformation to put data into the proper format. This section presents the sensor signal preprocessing methods used by researchers in gesture recognition.

Data cleaning is an umbrella word that groups a set of techniques used to smooth out a raw signal by filtering out artifacts and removing unwanted information to keep only the characteristics of signals that carry relevant information. Multiple filters are used that must be following the sensor's characteristics [45]. [46] applied a bandpass filter between 20 and 450 Hz on EMG signals to remove noise. In addition, a notch filter is used to remove the 50 Hz power line interference. In [44], the authors presented a new gesture recognition method using a flexible epidermal tactile sensor based on strain gauges to sense deformation. As this distortion may reduce the accuracy, they used a median filter to remove the noise. [47] Is an SLR system based on accelerometers and gyroscopes of a smartwatch. The authors applied moving average filters [48] to the sequence of collected signals to suppress the impact of jitter triggered by different activities.

In some works, researchers combined different sensors. For example, the Cyberglove [49] contains 18 to 22 sensors, each of which has its value range. A wide variety of values may dramatically influence the performance of some ML algorithms, e.g. when the Euclidean distance is used, hence the need for normalization. After normalization, each feature contributes approximately equally to the final distance. In [43], the data collected from an EMG is normalized before introducing it in a real-time HGR model. In [50], the authors leverage a light-weight off-the-shelf wearable device, Myo armband, that provides nine inertial sensing unit signals (3-axis accelerometer, gyroscope, and orientation) and eight-channel EMG signals. Since the EMG signal is much more noisy and stochastic, the authors used the RMS to process the EMG signal.

2.3.3 Data Segmentation

Sensor data usually comes as a continuous flow of raw data because sensors provide instant measurements of the controlled phenomenon, either on request or at regular intervals. Segmentation methods try to achieve a proper division of this raw and continuous data flow into smaller blocks of information. The appropriate selection and parameterization of segmentation techniques have a drastic impact on the success of feature extraction and ML algorithms [45]. In the works we have cited, two distinct approaches are commonly

used namely temporal-based segmentation and activity-based segmentation. Activity-based segmentation consists in dividing the signal data stream by identifying the start and end points of each gesture. Various methods are proposed in the literature to identify the beginning and ending points of gestures. Setting a threshold to detect changing points is the most common method. On the other hand, temporal based segmentation is further classified into Time interval-based segmentation and Sliding window segmentation. Time interval-based approach consists in dividing the sensorial datasets into blocks of equal time duration, while Sliding window segmentation divides the continuous signal data flow into windows with either static or dynamic sizes.

In [42], the authors developed an EMG based interface for HGR to recognize control signs in the gestures. To capture only relevant patterns in EMG waveforms, they designed an adaptive thresholding. [41] Proposed a gesture segmentation method based on Euclidean distance between each acceleration data time step. The acceleration is relatively stable when there is no hand movement. In contrast, it varies dramatically when in a hand motion state. The mean Euclidean distance in motion state is much higher than in status of no movement. Hence, it can be used to segment a gesture motion. MyoSign [50] is a deep learning model that enables end-to-end ASL recognition at both word and sentence levels. The continuous signal stream of sensors is segmented into a series of overlapping clips. They introduce a no sign class to represent the state without signs or the movement epenthesis between two continuous signs. Then Connectionist Temporal Classification (CTC) is applied on top of an LSTM which allows MyoSign to translate word or sentence in an end-to-end way.

2.3.4 Feature Extraction

In the sensor-based approach, the goal of this step is to extract new features from the signal. For example, acceleration signals are highly fluctuating and oscillatory, which makes it difficult to recognize the underlying patterns using their raw values. Time series analysis methods allow extending the signal to multiple features. They may be divided into two classes: frequency-domain methods and time-domain methods. Time domain analysis is about looking how time series evolves over time. It can include analysis of width, heights of the time steps, and other characteristics. On the other hand, many signals are better represented not by how they change over time, but what amplitudes

they have in it and how they change and that is the purpose of Frequency domain analysis. Time domain includes Auto-correlation and cross-correlation analysis, and Frequency domain includes Fourier and Wavelet analysis. Tab. 2.2 summarizes the main features that can be extracted in each of these domains:

Domain	Extracted Features
Time domain	Mean, Median, Average, Variance, Standard Deviation, Min, Max, Range, RMS, Correlation, Cross-Correlation, Integration, Differences, Angular Velocity, Signal magnitude area, Signal vector magnitude, Zero-crossing.
Frequency domain	DC component, Key Coefficients, Coefficients sum, Dominant frequency, Energy, Info Entropy.

Table 2.2: Taxonomy of extracted features in three domains

In [44], the authors based their system on strain gauges to sense deformation that are transduced to electric signals. From there, the authors extract two time-series, the Difference Absolute Mean Value (DAMV) and the Mean Absolute Value (MAV), to distinguish between gestures. In [42], the authors select features in the time domain (i.e., Max, Min, MAV, variance, RMS, and signal length) and features in the frequency domain obtained by using Fast Fourier Transform (FFT). They calculated: fundamental frequency and Fourier variance. Georgi et al. [51] developed a HGR system using IMU and EMG sensors, They compute for each sliding window a 28 dimensional feature vector. The first twelve dimensions are mean and standard deviation for each of the six channels of the IMU. The remaining 16 dimensions are the standard deviation of each EMG channel. [52] Summarize the main feature extraction techniques useful for acceleration, environmental measurements, and vital signs. They describe time-domain features like the ones mostly used on environmental measures, such as those obtained by binary sensors. In the case of acceleration data, time-domain and frequency-domain features are commonly used, whereas time-based features such as the number of heartbeats are used on vital signals [45].

2.3.5 Classification

All methods from above can get features for any ML model we have. However nowadays researchers do not want to rely on human-biased mathematical models and features. Neural networks are extremely powerful tools in biosignals analysis, they trade better than we do as well. Thus, they are able to automatically learn parameters and features to find effective solutions for complex problems. Besides, they are very fast to run in the inference stage even when the number of classes is very large.

In [50], the authors proposed a mixture of CNN, to abstract representations from inputs of different sensory modalities, and a bidirectional LSTM, to model temporal dependencies. The system has been evaluated on 70 ASL words and 100 ASL sentences and achieved an average accuracy of 92.4%. In [40], an RNN hand gesture classification based on accelerometer data is proposed. A combination of Convolutional LSTMs and standard LSTM cells was used to exploit both temporal and spatial information in the accelerometer signals. Experiments on a database consisting of 40 gestures performed by 20 able-bodied subjects and 2 amputees reveal an average recognition accuracy of 89.8% for able-bodied and 85.4% for amputees. In [42], an EMG based interface for HGR is presented. The authors combine two linear classifiers, k-NN and Bayes, to perform real-time classification. The combination allowed the recognition accuracy to reach 94%. In [44], the authors used an algorithm based on SVM and selected Radial Basis Function (RBF) kernel for non-linear classification. Only two features were used as input representing strain gauges deformation. The average accuracy for gesture recognition was 97.8% for six different gestures.

2.3.6 Results

This section summarizes some of the existing works in hand gesture recognition based on sensors, and this through the following table, where the lines represent the works, and the columns are the stages and methods used by each work. We did not consider the segmentation and feature extraction stages, since almost all works rely on sliding windows and extract features from both time and frequency domains when not using neural networks.

Ref	Data Acquisition	Database	Signal preprocessing	Classification	Acc.
[42]	EMG	4	Detrending function	k-NN, Bayes	94%
[51]	EMG, Accelerometers, Gyroscopes	12	Z-normalization	Hidden Markov Models	97.8%
[47]	Accelerometers, Gyroscopes	103	Moving average filter	B-LSTM	99.2%
[50]	EMG, Accelerometers, Gyroscopes	70	Median filter	CNN, B-LSTM	92.4%
[43]	EMG	5	Butterworth filter, muscle detection function	ANN	98.7%
[44]	Strain gauges	8	Median filter	SVM	97.8%
[40]	Accelerometer	40	Normalization	LSTM	89.8%

Table 2.3: Results obtained by various sensor-based gestures recognition

2.3.7 Discussion

In the previous table, we cited seven works using sensors to identify gestures. Among these works, two of them were interested in the problem of SLR namely MyoSign [50] and SignSpeaker [47]. These two works, unlike the systems cited in the vision-based approach, try to identify words and sentences instead of being limited to the alphabet. However, although these two works offer the possibility of recognizing a wide range of words and phrases, the fact remains that users must wear a smartwatch or an armband, respectively to Signspeaker [47] or MyoSign [50], in order to use the systems. Although these two devices are less invasive than others are, they are nevertheless cumbersome.

Besides, the use of B-LSTM has a practical impact on sentence level recognition by removing or adding information during learning such that the sequence of events that preceded and followed are taken into account when given the present. These systems offer a more extensive range of gestures compared to the previous category and are more

suitable for SLR in smart homes context.

2.3.8 Conclusion

In this section, we provide an overview of the steps in the gesture recognition process using the sensor approach, namely, data acquisition, Signal preprocessing, segmentation, feature extraction, and classification. The researchers proposed solutions for data collection, innovative noise reduction algorithms to remove the effects of interference, and to increase the number of identifiable gestures and recognition accuracy. Promising works have been cited that could suit SLR systems in a smart home context. Even if the sensor used are cheap and precise, this approach requires a device attached to the users' arms constantly so that they can communicate with the system. This constraint makes this solution invasive.

2.4 Wireless-Based Gesture Recognition

Contrary to the two previous categories, wireless systems require neither instrumentation worn by the user nor the installation of cameras. This approach's architecture consists of a transmitter broadcasting a signal (e.g., Wi-Fi), a transmitting antenna, a receiving antenna (often the same antenna is used for transmitting and receiving) and a receiver and processor to determine properties of the object(s). The deformations in the signals can be used to recognize hand and finger movements. Several works exist, and the main challenges are the ability to extract the gestures from wireless signals and dealing with multiple humans in the environment. The remainder of the section describes the wireless-based SLR process. First, we detail the types of wireless communication signals used for signs recognition. Next, the methods to extract proper information. Finally, the segmentation, the classification models, and their results.

2.4.1 Types of Wireless Transmission Used in SLR

The present works on wireless-based hand gesture recognition systems are categorized into three main trends:

Received Signal Strength (RSS)

It is the strength of the signal measured at the receiver's antenna and can be measured for each received packet. The signal's energy is quantized to form the Received Signal Strength Indicator (RSSI). There are four parameters associated with RSSI: dynamic range, accuracy, linearity, and averaging period. The RSSI dynamic range is specified in dB and indicates the minimum and maximum received signal's energy that the receiver is capable of measuring [53]. Abdelnasser et al. [54] and Haseeb et al. [55] proposed a gesture recognition system by leveraging changes in RSS due to hand motions. Several tools are used to collect RSSI measurements, [55] used a wireless extension for the Linux user interface.

Radio Frequency (RF)

Radio frequencies are generated and processed within many functional units such as transmitters, receivers, computers, televisions, and mobile phones. In the gesture recognition context, many works leverage RF waves to detect motion and gestural input. Examples include sensing via disturbances of GSM [56]. Google's soli work [57] relies on a high-frequency radar with a central frequency of 60 GHz, which allows for more fine-grained gesture sensing. The Doppler shifts and multi-path distortions that occur in Wi-Fi signals from human motion in the environment are exploited in [58]. RF-based systems are sometimes called Software Defined Radio (SDR) based since raw signals are collected using devices such as Universal Software Radio Peripheral (USRP) and Radio-frequency IDentification (RFID) readers.

Channel State Information (CSI)

In wireless communications, CSI refers to known channel properties of a communication link. This information describes how a signal propagates from the transmitter to the receiver and represents the combined effect of, for example, scattering, fading, and power decay with distance. CSI-based sensing systems have been designed for various purposes, such as localization, human motion detection, and counting humans. Recently, CSI has been extended to recognize human activities such as fall detection, daily activity recognition, micro-movement recognition, and gesture recognition. In [59] and [60], the authors leverage the fluctuations in the CSI of Wi-Fi signals caused by hand motions to classify gestures with high accuracy even in scenarios where the signal passes through multiple

walls. The most commonly used software to gather CSI is Linux 802.11n CSI Tool [61].

2.4.2 Signal Preprocessing

The challenge in wireless communications signal preprocessing is to increase the capacity in bps/Hz of bandwidth through reducing noise, interference and multipath distortion. Signal processing methods play the central role in removing or compensating for noise and thereby improving capacity [62].

In [59], two walls separate the transmitter and receiver from the user who lies within a distance of 8 m. The authors use a Butterworth filter to remove noise but preserve the real trend of CSI caused by hand motion. Besides, to focus on gestures from a particular user, the authors rely on Multiple-Input Multiple-Output (MIMO) capability that is inherent to 802.11n. It provides throughput gains by enabling multiple transmitters to send packets to a MIMO receiver concurrently. If we consider the wireless reflections from each human as signals from a wireless transmitter, then they can be separated using a MIMO receiver. [55] Process the RSSI signal by comparing its values with a threshold. All windows that have a variance less than the threshold will be predicted as no gesture or noise. [58] Leverages the property of Doppler shift, which is the frequency change of a wave as its source moves relative to the observer. Similarly, the human hand gestures result in minimal Doppler shifts. To overcome this challenge, the authors transform the received Wi-Fi signal into a narrowband pulse with the bandwidth of a few Hertz. The receiver then tracks the frequency of this narrowband pulse to detect the small Doppler shifts. [60] Explains that the 3D CSI matrix is similar to a digital image with spatial resolution of $N \times M$ and K color channels, so CSI-based Wi-Fi sensing can reuse the signal processing techniques and algorithms designed for computer vision tasks.

2.4.3 Segmentation

The segmentation (Also called gesture detection) algorithm plays an important role in the overall performance of the gestures identification system. It is important to detect the gestures first because in this way we can properly label the data we collect and train the system. Incorrect segmentation of the received signals can trigger incorrect classifications. In addition, through gesture detection, we segment the signal samples we collected, so that only those segments corresponding to a gesture will be processed using the ML

algorithm. This naturally reduces the computational cost. Wireless communication data comes as a continuous flow of raw data. Therefore, the segmentation algorithms used in this approach are similar to sensor based approach namely temporal-based segmentation and activity-based segmentation.

In [58] the receiver computes the average energy in the positive and negative Doppler frequencies. When the ratio between this average energy and the noise level is greater than 3 dB, the receiver detects the beginning of a segment. If this ratio falls below 3 dB, the receiver detects the end of the segment. In [55], the incoming RSSI stream is split into overlapping windows of T seconds length, and d seconds gaps between window starts. In [59] the authors first design a multi-level wavelet decomposition algorithm to decompose CSI and analyze its frequency and time components to obtain a unique pattern for each gesture. Then, apply a short-time energy algorithm to the results of wavelet decomposition of CSI, which can classify silent and communication periods.

2.4.4 Feature Extraction

As the sensor based approach, the goal of this step is to extract new features from the signal. The receiver collects a signal that is not enough to distinguish between gestures. Time series analysis methods allow extending the signal to multiple features.[56] Developed an algorithm to convert the reflected GSM pulses to a continuous signal that can be used for gesture recognition. Combining the data points from four antennas, they generated a feature vector with 18,000 elements. They reduced the feature space by downsampling the feature vector to 80 elements (i.e., 20 elements per channel) using only time-domain data. On the other hand, [54] uses a discrete wavelet transform to capture both frequency and time domain data from RSSI values. [63] Introduces Higher order statistics based third-order cumulant features that are robust to background noises and signal interferences. They demonstrated that using this feature extraction method could achieve a better classification accuracy even without applying any preprocessing methods.

2.4.5 Classification

The works presented in this approach differ in the range of the system, For example, the range of the Soli [57] radar is limited to 30 cm. As well as SignFi [60], which relies on CSI measurements to predict gestures, separates the transmitter and receiver from

230 cm and 130 cm, respectively, for the laboratory and the domestic environment. On the other hand WiGer [59] increases the distance between the user and the receiver. In their experiments, the authors test their system in a scenario where two walls separate the transmitter and the receiver from the user who is at a distance of 8 m. Works that cover a large area test their system in scenarios where the transmitter and receiver are in the Line Of Sight (LOS) as well as in the None Line Of Sight (NLOS). For instance, [64] built a system to identify four different gestures by leveraging CSI measurements, the authors use SVM with RBF kernel to classify the gestures. They also used the LIBSVM [65], an open source machine learning library that applies a method named one-against-one method to construct $k(k-1)$ classifiers where each one is trained on data from two classes to classify these two classes and k is the number of classes. Then a voting strategy is used to predict the test data. In experimental results, their system achieved an average accuracy of 92% in LOS and 88% in NLOS. [55] Developed a system to control a smartphone with three different gestures. To classify each gesture, they used an LSTM model. The model's hyperparameters were selected by performing a grid search in the parameters space. Each parameters' setting is evaluated using four-folds cross-validation. The experiments showed an average accuracy of 78%. In [57], the authors proposed a combination of Deep Convolutional Neural Network (DCNN) and RNN to identify 11 gestures. Where the DCNN has eight layers and the last fully connected layer is directly connected to the input of the LSTM cell. They achieved an average accuracy of 87%.

2.4.6 Results

This section summarizes some of the existing works in HGR based on wireless communication, and this through the following table, where the lines represent the works, and the columns are the stages and methods used by each work. As works on sensor based, we did not consider the segmentation and feature extraction stages, since almost all works rely on sliding windows and extract features from both time and frequency domains when not using neural networks. The accuracy of each work covering a large perimeter was mentioned in the NLOS and LOS scenarios respectively.

Ref	Database	Wireless transmission	Signal preprocessing	Classification	Acc.
[54]	7	RSS	Threshold-based on Stein unbiased risk estimate	Pattern Matching	87.5%
[55]	3	RSS	Compare with a threshold	RNN	78%
[66]	25	RSS	Low pass filter	DTW	96%
[56]	14	RF (GSM)	Savitzky-Golay filter	SVM with PUK kernel	87.2%
[57]	11	RF (Radar)	Signal normalization, Logarithmic data scaling, Max-min truncation	CNN + RNN	87%
[67]	9	RF(RFID)	Phase Shift Calibration, Wavelet Denoising	Random Forest	96%
[60]	276	CSI	Multiple linear regression	CNN	98.91%
[59]	7	CSI	Butterworth low-pass filter	DTW	89%
					97.28%
[63]	276	CSI	Without SP	SVM with RBF kernel	98.26%

Table 2.4: Results obtained by various wireless-based gestures recognition

2.4.7 Discussion

In the previous table, we cited nine works based on wireless communications to perform hand gestures recognition. Among these works, four of them have interpreted SL gestures namely [63], [60], [66] and [67]. These works, like the outstanding systems cited in the sensor based approach, try to identify words and sentences. [63], [60] and [66] are based on CSI measurements while [67] relied on RFID technology. The latter focused on the interpretation of sentences unlike the other that handle only words. [63] and [60] use the same dataset composed of 276 terms, these terms can further be composed to construct sentences.

On the other hand, the main income of such approaches (compared to vision and sensor based systems) is the freedom of the user from carrying or deployment new hardware. However, some wireless-based systems require specific materials. For example, [58] used a USRP-N210s to implement and evaluate a prototype of the system. Others require the user to perform the sign language in front of the receiver. In [57], the user has to perform his gestures 30cm above the radar. Other approaches allow the user to be far from the transmitter or the receiver, which brought new challenges (e.g., distance, obstacles, and interferences). For instance, the authors of [59] test their system in a scenario where two walls separate the transmitter and the receiver from the user who is at a distance of 8m. The latter system is based on CSI measurements, if we manage to find a compromise between the number recognized gestures in [63],[60] and the robustness and range in [59], relying on CSI measurements would be ideal for an SLR system in the environment of a smart home.

2.4.8 Conclusion

In this section, we passed through the most common stages of gesture recognition based on wireless transmission, namely signal preprocessing, segmentation, feature extraction, and classification for each type of transmission used in works. This approach resolves previous approaches limitations since neither vision nor wearable devices are required. Dealing with obstacles like walls, other human motions, and various interferences may be challenging. Using CSI to identify gestures have made an impact on wireless-based systems. Therefore, leveraging the fluctuations in the CSI of Wi-Fi signals makes it a suitable option for SLR in the smart home context.

2.5 Summary and Discussion

2.5.1 Used Approaches in Existing Works

This section presents a summary of the currently used approaches in hand gesture recognition cited above with their input methods, and this through the following table, where the lines represent the works, and the columns are the approaches and methods used by each work. X Indicates the approach (columns) used by each work (rows).

Ref.	Vision			Sensor				Wireless		
	Kinect	Webcam	Video camera	Strain gauges	Gyroscopes	Accelerometers	EMG	RF	RSS	CSI
[25][26][36]		**X**								
[28][39]	**X**									
[29][34][37]			**X**							
[42][43]							**X**			
[50][51]					**X**	**X**	**X**			
[47]					**X**	**X**				
[44]				**X**						
[40]						**X**				
[54][55]									**X**	
[56][57][58]								**X**		
[59][60][63]										**X**

Table 2.5: Summary of the currently used approaches in hand gesture recognition

2.5.2 Comparison Criteria

We establish three categories of comparison criteria related to smart homes, sign language, and system performance.

Smart home systems' criteria are privacy, affordability, and easy installation. Smart home systems control the entire house connected devices, from lights to door locks. Besides, this kind of system records users' data and habits continuously. Data falling into malicious hands is hazardous for the house occupants. On the other hand, smart home devices and connected peripherals are expensive and hard to deploy.

Sign language recognition systems' criteria are the recognition of static vs. dynamic signs and the invasiveness of the acquisition method. SL symbols are classified into static and dynamic gestures, as well as constructing sentences requires the sequence of multiple signs. For optimal experience, the SLR systems should handle dynamic signs.

Additionally, SLR systems may be used for long and anytime. Thus a comfortable device would make the system better.

System performance criteria are the recognition accuracy, the robustness against the environment, and the number of handled gestures. All the cited approaches above rely on machine learning algorithms to predict the correct sign produced by the user and the more accurate the system, the better it is. Besides, the studied systems face several challenges related to the environment that could reduce the system performance. Therefore robustness is an important criterion to compare between the works. On the other hand, the number of gestures handled by the system must be considered, since the number of classes affects the recognition rate. In our comparison, we consider the minimum number of recognized gestures by the approaches of 100, since our goal is not to develop a communication tool, but a way to control devices. Similarly, we consider 90% as the minimum recognition accuracy, since the current voice-based devices' accuracy is between 90% and 95%.

2.5.3 Discussion

In Tab. 2.6, we compare the current approaches in the literature to recognize hand gestures following the criteria that we proposed above.

	Vision	**Sensor**	**Wireless**
Privacy issue		✓	✓
Affordable		✓	✓
Ease of installation			✓
Dynamic	✓	✓	✓
Noninvasive	✓		✓
Accurate	✓	✓	✓
Robust		✓	
Number of gestures		✓	✓

Table 2.6: Comparison between the existing approaches in hand gesture recognition

Vision-based systems have a drastic impact on gesture recognition and achieved commercial success (e.g., Microsoft's Kinect [23]). However, due to the installation cost and

personal privacy, most of the vision-based sensing devices are hard to deploy at scale, for example, throughout an entire home or building. Sensor-based systems responded to those problems since there is no vision, and the sensors used are generally cheap with high precision. However, it requires the permanent wear of a glove or a wristband to communicate with the system, which makes the solution invasive. Wireless systems require neither instrumentation nor infrastructure of cameras. However, the range is limited and can be increased by adding access points, which increases the cost.

2.5.4 Adopted Approach

The study of the current approaches for hand gesture recognition and smart homes and sign language constraints, we opted to use CSI measurements present in WiFi signals to recognize a sentence of signs that represents an action on connected peripherals. We exploit CSI measurements among other wireless trends, since CSI measurements are used for different sensing purposes. Besides, WiFi sensing exploits the infrastructure used for wireless communication, so it is easy to deploy with a low cost. Moreover, unlike sensor-based and vision-based solutions, WiFi sensing is not invasive or sensitive to lighting conditions.

2.6 Conclusion

In this chapter, we presented an overview of hand gesture recognition techniques, which opens a wide range of options to deploy in a smart home in particular for hearing and speaking impaired occupants. Sign language recognition is one of hand gesture recognition applications where machine learning has good potential. In the literature, three different approaches have stood out (i.e., vision, sensor, and wireless-based). For the three approaches, we presented methods proposed by the researchers following the stages of the recognition process. Therefore we summarized works in the three categories and proposed comparison criteria related to smart homes, sign language, and system performance. Based on the latter comparison, we opted to develop a sign language recognition system following the wireless approach by leveraging CSI measurements present in WiFi signals.

Chapter 3

Solution Description

3.1 Introduction

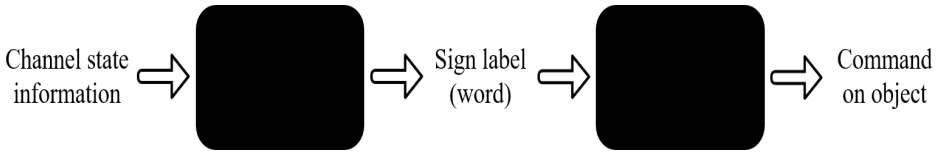
During our study, we developed two different architectures to use CSI measurements to predict the actions desired by the user. In this chapter, we present the design of those architectures that translates sign language into smart home action using CSI measurements present in the WiFi signal. The chapter describes all the stages of the process, namely the collection of CSI traces, signal preprocessing, and the machine learning algorithms.

3.2 System Overview

The objective of our work is to develop a system for translating sign language into commands that can be executed by the smart home. We have considered two different architectures for this, namely Word-based and Sequence-based architectures which will be the objects of sections Sec. 3.3 and Sec. 3.4, respectively.

The format of the input data is what makes the difference between these two architectures. Indeed, as its name indicates, the first architecture processes CSI measurements sign by sign. Subsequently, once a sequence of signs is recognized, It is processed separately in order to predict the desired action. As for the second architecture, in order to predict the action wanted by the user, it directly processes a sequence of CSI measurements reflecting a sequence of signs.

The following figures illustrate, using black boxes, the process of the two architectures proposed in our solution.



(a) Word-based architecture process

(b) Sequence-based architecture process

Figure 3.1: Architectures' black boxes representation

3.3 Word-Based Architecture

This architecture consists of the following stages, as shown in Fig. 3.2:

- Preparation: In this stage, the system collects CSI measurements from the transmitter (i.e., WiFi access point). Raw CSI measurements are preprocessed to remove noises. Then, the amplitude and the phase are extracted.
- Word-level recognition: The purpose of this stage is to predict word by word the performed signs reflected by the CSI measurements. For this, we rely on a CNN model to achieve both feature extraction and word classification.
- Sentence-level recognition: In this stage, we manage to assemble the predicted words from the latter stage in a vector that represents an action on the connected peripherals. Thus, we rely on an RNN model to perform the commands prediction from the constructed sequence of words.

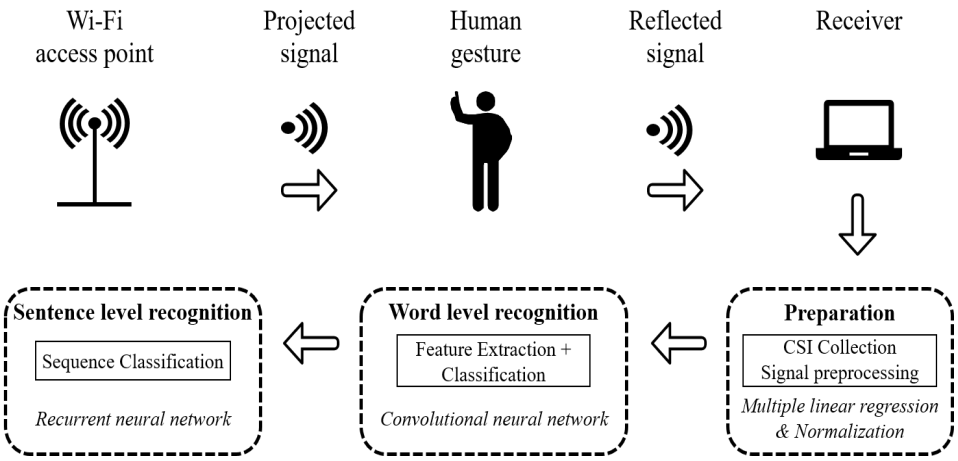


Figure 3.2: Word-based architecture overview

3.3.1 Preparation

CSI Collection

Channel state information is the physical layer information of the WiFi signal. In contrast with the RSSI information, CSI gives information on different channels instead of a cumulative signal strength indication. WiFi devices following 802.11n standards work with Orthogonal Frequency Division Multiplexing (OFDM), achieve increased data rates, improved capacity, and a reduced bit error rate of the system. Besides, devices starting from IEEE 802.11n support MIMO with the OFDM scheme, enabling them to send and receive information over multiple antennas (as shown in Fig. 3.3). The OFDM extracts the channel frequency response in the format of CSI, which contains both amplitude and phase information of the signal in the subcarrier level.

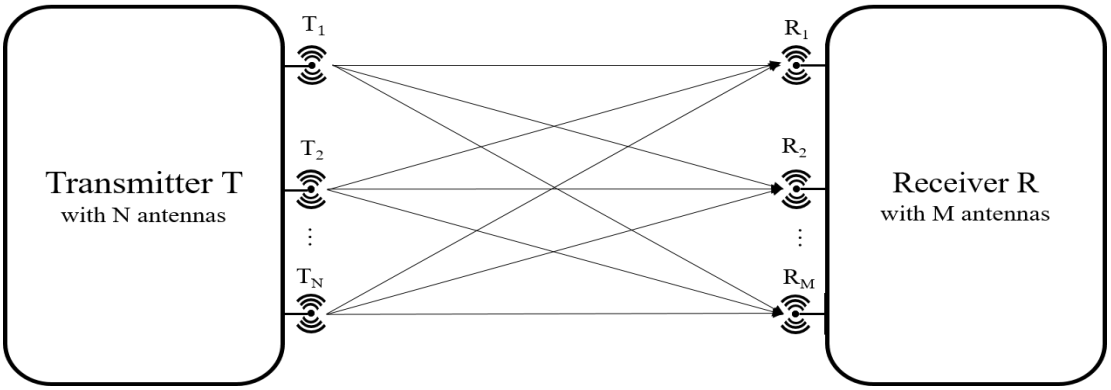


Figure 3.3: CSI representation of MIMO

Wireless communication systems are subject to attenuation, delays, and phase shifts. To maintain the rate adaptations and transmit power, WiFi devices have to monitor the channel state for each MIMO channel continuously. The narrowband flat fading channel model of the MIMO system is given by [68]:

$$Y_{ij} = H_{ij} \times X_{ij} + N \tag{3.1}$$

Where H denotes the channel state information. X and Y express the transmitter and receiver, respectively, N is the noise, i is the stream number, and j is the subcarrier number. In a MIMO system with T number of transmit antennas, R number of receive antennas, and C number of channel subcarriers, CSI is a 3-dimensional matrix with a size of $T \times R \times C$. Each element in the matrix is a complex number with the amplitude and phase information of the corresponding receiver-transmitter pair of antennas for a particular OFDM sub-carrier. The H matrix could be represented as:

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} & \dots & H_{1R} \\ H_{21} & H_{22} & H_{23} & & H_{2R} \\ & \vdots & & \ddots & \vdots \\ H_{T1} & H_{T2} & H_{T3} & \dots & H_{TR} \end{bmatrix} \tag{3.2}$$

Where $H_{ij} = h_1, h_2, h_3, \dots, h_C$ is the CSI value for transmitting antenna i and receiving antenna j and h_k denotes channel sate for k_{th} sub-carrier [68]. Each complex h_k could also be represented by:

$$h_k = \|h_k\| \cdot e^{j \angle h_k} \tag{3.3}$$

Where $\|h_k\|$ is the amplitude, $\angle h_k$ is the phase. If either the amplitude or phase of at least one path changes, the CSI value changes. Since CSI captures how surrounding objects reflect wireless signals, it can be used to recognize hand and finger gestures. For instance, Fig. 3.4 shows the result of the wavelet decomposition algorithm and the changes in amplitude caused by three different air-drawn gestures.

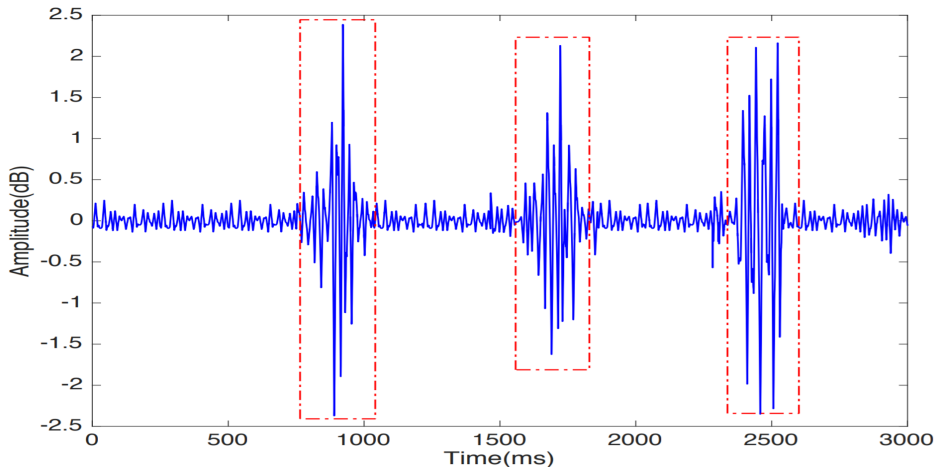


Figure 3.4: Wavelet analysis results of air-drawn gestures [59]

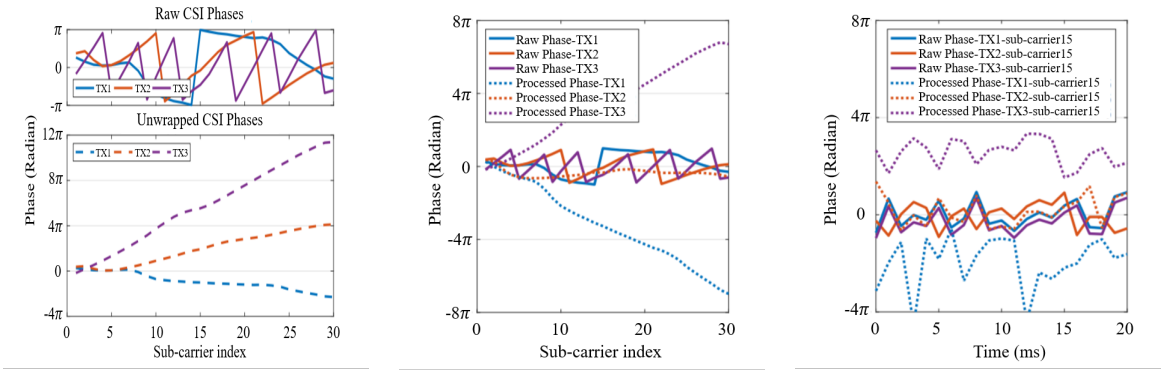
In our work, we rely on changes in amplitude and phase extracted from CSI measurements to predict gestures, since the CSI phase is much more sensitive to environment dynamics due to its inherent short wavelength [69]. However, although CSI is included in WiFi since IEEE 802.11n, not all standard WiFi cards report it. Thus, the present work uses raw CSI traces of the SignFi dataset [60], where the gesture movements refer to American Sign Language. More details about this dataset are given in the implementation section.

Signal preprocessing

Wireless communication systems are designed to be coherent systems. The receiver needs to know the frequency and phase of the transmitted signal. However, perfect coherence cannot be achieved as the oscillator on the transmitter and receiver differ. Thus, raw CSI measurements contain phase offsets due to hardware and software errors. For instance, Sampling Time/Frequency Offsets (STO/SFO) are due to unsynchronized sampling clocks/frequencies of the receiver and transmitter. It introduces estimation errors for angle-of-arrival and time-of-flight, which are used to track and localize humans and objects. Besides, the raw measured CSI phases give redundant information about how CSI phases change. Thus, phase offset removal improves performance for binary and multi-class classification applications since it recovers CSI phase patterns over subcarriers and sampling time [70].

In SignFi [60] dataset, the authors have already preprocessed the CSI samples using

multiple linear regression to remove phase offset. The phase offsets are estimated by minimizing the fitting errors across C subcarriers, N transmit antennas, and M receive antennas. Preprocessed CSI phases are obtained by removing the estimated phase offsets from the measured CSI phases. Fig. 3.5 shows the effectiveness of signal preprocessing on the CSI phase. On the other hand, no filter method has been mentioned, since the authors built their dataset in two different environments (i.e. laboratory and home) as the layout, surrounding environment, and displacement of the transmitter and receiver of the lab and home are different, leading to different noise signals. Afterward, CSI measurements are manually segmented for each sign gesture. Nevertheless, in our work, in contrast to SignFi, we normalized the data since the given datasets have a different range of values.



(a) Raw and Unwrapped CSI Phases

(b) CSI Phase vs. sub-carrier Index

(c) CSI Phase vs. Sampling Time

Figure 3.5: Raw CSI measurements do not capture how CSI phases change over subcarriers and sampling time [60]

3.3.2 Word-level recognition

The main goal of this stage is to process CSI measurements in order to predict one sign performed by the user at a time. Processing CSI measurements requires extensive computation resources. For example, $size(H) = 3 \times 3 \times 52 \times 100 \times 32/8 = 187,200$ *bytes* for a WiFi channel with 3 transmitter/receiver antennas, 52 subcarriers, and 100 CSI samples with each value represented by 32 bits. Besides, CSI measurement processing is a time-frequency analysis task. Thus, signal transformation and dimensionality reduction techniques are essential before further processing. Before the machine learning and deep learning era, people were creating mathematical models and approaches for time-series

and signals analysis. Time-frequency analysis methods such as Fast Fourier Transform and Discrete Wavelet Transform are widely used to find the distinct dominant frequencies. However, in our work we do not rely on such methods but rather take advantage of deep learning to extract new features and reduce the dimensionality.

Convolutional Neural Network

A CNN model can be defined as a combination of two components: the feature extraction part and the classification part. The feature extraction layers have a generally repeating pattern of the sequence composed of the convolution layer, the ReLU activation function and the pooling layer. There could be multiple of these units connecting for large and complex datasets. In our proposed method, only one unit is used to learn essential features by convolving the CSI input, since we tested multiple of these units in our experimentation and It did not improve the system performance.

The convolutional layers are designed to extract important features from the multi-channel CSI data based on the findings of existing studies on CSI [71]. Besides, the 3D CSI matrix is similar to a digital image with a spatial resolution of $N \times M$ and K color channels. Thus, CSI-based WiFi sensing can reuse signal processing techniques and algorithms designed for computer vision tasks. CNN models have shown significant performance in recognition and detection tasks. The Rectified Linear Unit layer provides fast and effective training for deep neural networks, since its activation function is easy to compute and optimize. It has been shown more effective than traditional activations, such as logistic sigmoid and hyperbolic tangent, and is widely used in CNNs [19]. The combination of the convolutional layer and a nonlinear activation function (ReLU in our case) is sometimes referred as the detector stage. A pooling layer follows this stage to reduce the spatial size (width and height) of the data representation and help control overfitting.

Among the convolutional unit, we included in our model a Batch Normalization (BN) layer. BN layer accelerates training in CNNs by normalizing the output of the previous layer at each batch. It applies a transformation that keeps the mean activation close to zero while also keeping the activation standard deviation close to one. BN also reduces the sensitivity of training toward weight initialization and acts as a regularizer [17]. This approach leads to faster learning rates since normalization ensures no activation value is

too high or too low, as well as allowing each layer to learn independently of the others.

As for the classification part, we can have one or more fully connected layers to take the higher-order features and produce class probabilities or scores. However, these layers process vectors, which are 1D, whereas the current output is a 3D tensor. First we have to flatten the 3D outputs to 1D using a flattening layer and then add a fully connected layer on top that connects all of its neurons to the neurons in the previous layer. The purpose is to combine all the features learned by previous layers (i.e. Convolutionial unit) to classify the input.

The fully connected layer typically uses either a softmax or sigmoid activation function for classification. Softmax functions are most often used as the output of a classifier, to represent the probability distribution over n different classes [19]. Since we have a multiclass modeling problem, we use a softmax output layer to get the highest score of all the classes. The softmax output layer gives us a probability distribution over all the classes.

Fig. 3.6 shows how the layers are overlapped in our model. More details about the parametrization are given in the next chapter.

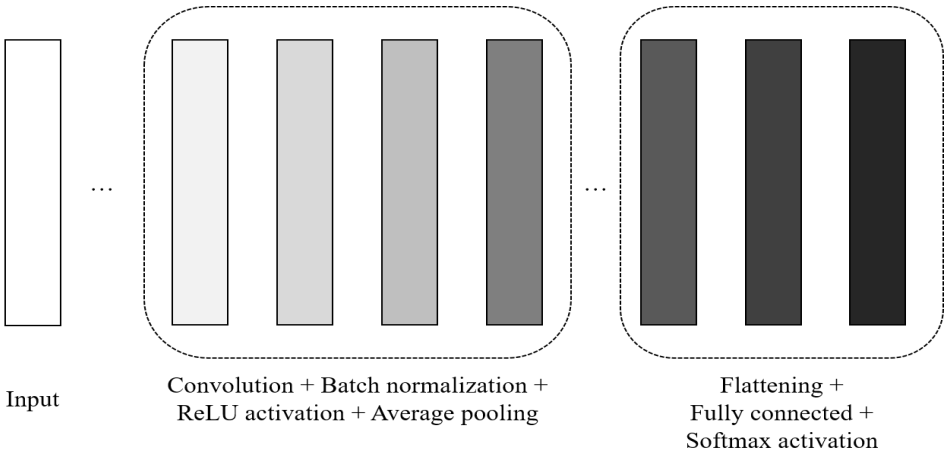


Figure 3.6: Architecture of the CNN model

3.3.3 Sentence-level recognition

Through this work, our contribution is to perform a sentence-level sign language recognition system to execute actions on smart home peripherals. Thus, the purpose of this stage is to process a set of words or sentence to predict the action wanted by the user.

These words come from the previous stage i.e. the word-level recognition. The sentences must contain the objects to control, the action, and, eventually, the room, for instance, "Start air conditioner" or "Open kitchen door." Although the dataset on which we rely offers a wide range of objects, actions, and rooms, it nevertheless remains addressed to word-level recognition problems since labels represent single words. Thus, we must first build sentences from the latter dataset to proceed to sentence-level recognition.

To build the sentences, we first selected words from the dataset that are the best suited to our context. More details about the word selection are given in the implementation section. Therefore, we combined all the latter words using the following sentence structures:

Time	+	Verb	+	Indirect object	+	Direct object
Afternoon		Stop		Apartment		Air conditioner

Verb	+	Indirect object	+	Direct object
Stop		Apartment		Air conditioner

Verb	+	Direct object
Stop		Air conditioner

Figure 3.7: Sentence structures

Recurrent Neural Network

Sentence-level sign language recognition is a sequence-learning task since we are given a series of words that represent the sequential data. Recurrent Neural Networks (RNNs) are well suited for sequence learning problems due to their ability to handle temporal dependencies of the input data. The state of the RNN is reset between processing two different, independent sequences (such as two different sentences). Thus, in the design of our system, we use an RNN model to support sequence prediction.

Nevertheless, before feeding our sequence of raw words to the RNN layer, each word must be represented by a vector of integers. A popular and powerful way to associate a vector with a word is the use of dense word vectors, also called word embeddings. Whereas

the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros), and high-dimensional (same dimensionality as the number of words in the vocabulary), word embeddings are low-dimensional floating-point vectors. Unlike the word vectors obtained via one-hot encoding, word embeddings are learned from data [7]. We use word-embedding techniques because they are low-dimensional and compact. In an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. The position of a word within the vector space is learned from the text. In our work, we used an embedding layer at the beginning of the neural network. It is initialized with random weights and will learn an embedding for all of the words in the training dataset.

Afterward, the embedded sequence of words is fed into an RNN layer that iterates over the words of a sequence, while maintaining an internal state that encodes information about the previous words. The output of the RNN layer contains a single vector per sample. This vector is the last's RNN cell output containing information about the entire input sequence. It is sometimes useful to stack several recurrent layers one after the other in order to increase the representational power of a network [7]. However, in our experimentation It did not improve the system performance.

The RNN layer is followed by a fully connected layer and softmax activation function, which will determine the probability for each sequence. In the literature, It is deprecated to use softmax activation function for classification where the number of classes is large (e.g many thousand). It is recommended to use the variant of the softmax activation function called the hierarchical softmax activation function. This variant decomposes the labels into a tree structure, and the softmax classifier is trained at each node of the tree to direct the branching for classification [17]. However, this is not used in our work for implementation reasons and will be considered for future works.

Fig. 3.8 shows how the layers are overlapped in our model. More details about the parametrization are given in the next chapter.

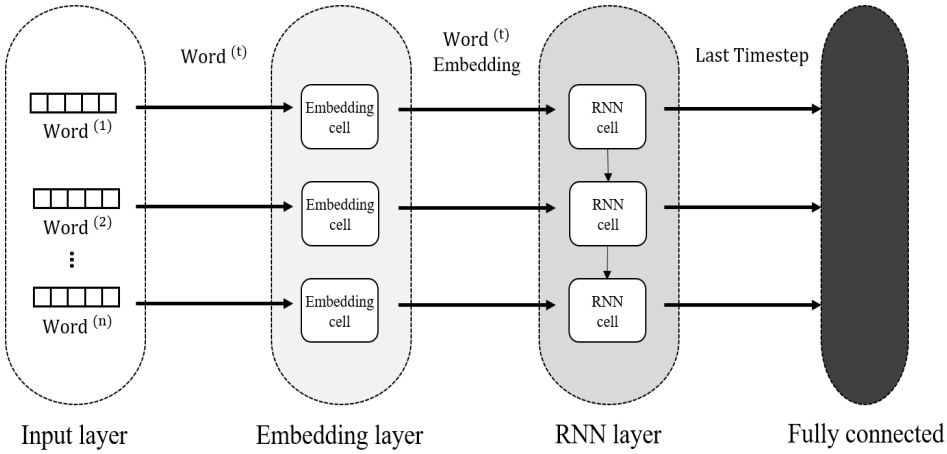


Figure 3.8: Architecture of the RNN model

3.4 Sequence-Based Architecture

This second architecture consists of the following stages, as shown in Fig. 3.9:

- Preparation: As in the word-based architecture, we use CSI measurements, then we have to go through a preprocessing step, which is similar to the one seen above, thus it is not described in this section. Instead, we prepare CSI sequences that reflect a sentence expressing the desired command.
- Sequence-learning: At this stage, we use a sequence of preprocessed CSI measurements to directly predict the desired action. For this, we use a Convolutional Recurrent Neural Network, which is described in the following section.

The particularity of this second method is that it uses a single-step architecture, i.e. the preprocessed CSI measurements go through a single deep learning model to predict the desired action.

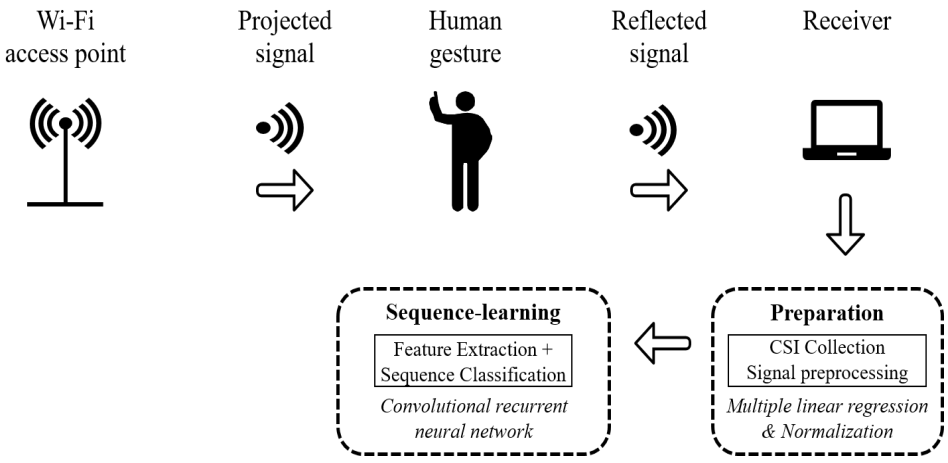


Figure 3.9: Sequence-based architecture overview

3.4.1 Preparation

The dataset on which we rely in our work is composed of preprocessed CSI measurements segmented word by word. Hence, a similarity with the word-based architecture, we generated a set of sentences with the words proposed in the dataset. Subsequently, we transformed the words of each sentence by their proper CSI value. The result of this operation is a dataset of CSI sequences representing the sentences and are labeled by the desired commands. More details about this dataset are given in the implementation section.

3.4.2 Sequence-learning

Although the RNN layer has proven powerful for handling temporal correlation, it contains too much redundancy for spatial data. To address this problem, we propose an extension of the RNN which has convolutional structures in both the input-to-state and state-to-state transitions. Thus, in the sequence-based architecture, we use a combination of these two prominent neural networks CNN and RNN to achieve a one-step sign language sequence recognition.

The Convolutional Recurrent Neural Network (CRNN) is similar to an RNN but has convolutional structures in both the input-to-state and state-to-state transitions. A distinguishing feature of this design is that all the inputs $X_1, \ldots, X_t$, cell outputs $Y_1, \ldots, Y_t$, hidden states $H_1, \ldots, H_t$ (and gates i_t, f_t, o_t in case of Convolutional LSTM) are 3D ten-

sors whose last two dimensions are spatial dimensions (rows and columns) [72]. The key equations of Convolutional LSTM (ConvLSTM) are shown below, where '$*$' denotes the convolution operator and $\odot$ denotes the Hadamard product:

$$I^{<t>} = \sigma\left(x^{<t>} * W_{xi} + H^{<t-1>} * W_{hi} + C^{<t-1>} \odot W_{ci} + b_i\right) \tag{3.4}$$

$$F^{<t>} = \sigma\left(x^{<t>} * W_{xf} + H^{<t-1>} * W_{hf} + C^{<t-1>} \odot W_{cf} + b_f\right) \tag{3.5}$$

$$O^{<t>} = \sigma\left(x^{<t>} * W_{xo} + H^{<t-1>} * W_{ho} + C^{<t-1>} \odot W_{co} + b_o\right) \tag{3.6}$$

$$\widetilde{C}^{<t>} = \tanh\left(x^{<t>} * W_{xc} + H^{<t-1>} * W_{hc} + b_c\right)) \tag{3.7}$$

$$C^{<t>} = F^{<t>} \odot C^{<t-1>} + I^{<t>} \odot \widetilde{C}^{<t>} \tag{3.8}$$

$$H^{<t>} = O^{<t>} \odot \tanh\left(C^{<t>}\right) \tag{3.9}$$

Once the data reshaped, it is fed to a deep learning model composed of a CRNN layer. The output of this layer is a combination of convolution and an RNN output, i.e. a 3D tensor that is the last CRNN cell output containing information about the entire input sequence. This layer is followed by a batch normalization to normalize the output of the previous layer at each batch, ReLU activation function, pooling, fully connected layer and softmax function to determine the probability for each sequence.

Fig. 3.10 shows how layers are structured in this model. More details about the parametrization are given in the next chapter.

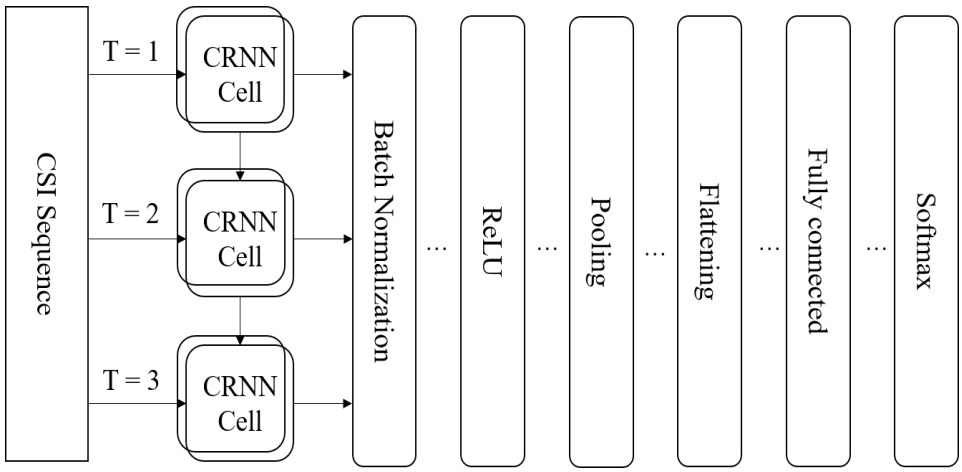


Figure 3.10: Architecture of the CRNN model

3.5 Conclusion

In this chapter, we reviewed two different architectures that translate sign language into smart home actions. The first one is done in two steps using a CNN model to perform word-level recognition from the CSI measurements. A sequence of these words is then fed to an RNN model that performs sentence-level recognition to predict the desired action. As for sequence-based architecture, it is done in a single step. We use a CRNN model that, through a sequence of CSI measurements, can directly predict the desired action. In the next chapter, we describe the details of the implementation of these two architectures and evaluate their performance.

Chapter 4

Implementation & Evaluation

4.1 Introduction

This chapter covers the details of the implementation of the two architectures proposed in the previous chapter, including the presentation of the datasets and the tools used as well as the experimentation settings. It also covers performance evaluation. This through the evaluation of the system on different datasets, as well as the study of the deep learning models parameters. We also present a comparison between the two proposed architectures and work present in the literature. We conclude with an overview of the deployment.

4.2 Implementation

This section describes different implementation stages. It covers the SignFi [60] dataset and the used libraries. We justify the choice of the used framework for the development of the deep learning models with a comparative study between the different existing frameworks. Afterward, more details about the deep learning networks present in our system are given.

4.2.1 SignFi Dataset

As mentioned in the previous section, although CSI is included in WiFi since IEEE 802.11n, not all standard WiFi cards support it. The most commonly used tool to collect CSI measurements in the literature is the Linux 802.11n CSI Tool [61]. It is built on the Intel WiFi Wireless Link 5300 802.11n MIMO radios, using a custom modified firmware

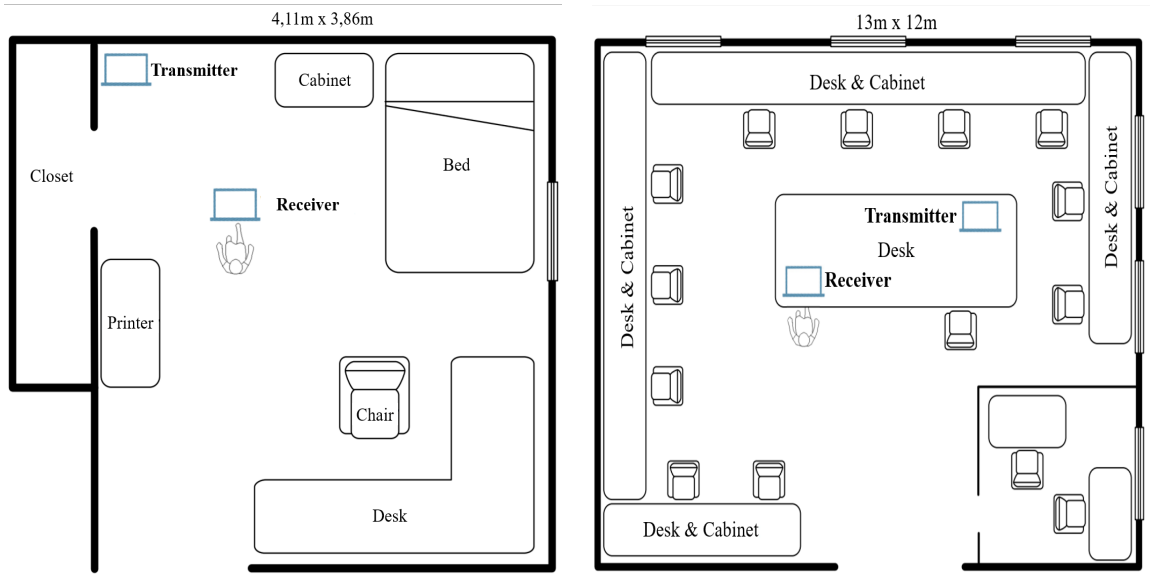
and open-source Linux wireless drivers. Thus, the tool relies on the custom firmware image that only works on the IWL 5300 card. The present work uses CSI measurements of the SignFi dataset [60], following ASL.

The authors of the SignFi dataset captured the raw CSI measurements for 276 signs that are frequently used in daily life, with 20 and 10 instances for each gesture in the laboratory (lab) and home environments, respectively. The SignFi dataset is the dataset of choice in the present work, as it is the only dataset containing CSI traces with 276 gestures and a significant number of gesture samples with high clarity in data acquisition. There are 8280 gesture instances, 5520 from the lab, and 2760 from the home environment, for 276 sign gestures in total from 1 user. Tab. 4.1 summarizes the 276 sign words used in our experiments divided into different categories. According to the authors, these sign words are the essential words for ASL beginners since they are not composed of more than three signs per word.

Common	Animals	Colors	Descriptions	Family	Food	Home	People	Questions	School	Time	Others	Total
16	15	12	32	31	54	17	13	6	26	31	23	276

Table 4.1: SignFi dataset summary

CSI traces are measured in both lab and home environments. The dimension of the lab and home is $13m \times 12m$ and $4.11m \times 3.86m$, respectively. The lab has more surrounding objects, leading to a more complex multi-path environment than the home. The distance between the transmitter and receiver is 230cm and 130cm, respectively, for the lab and home environment. For the home environment, the transmit antenna array is orthogonal to the direction from the transmitter to the receiver. For the lab environment, the angle between the transmit antenna array and the direct path is about 40 degrees. Fig. 4.1 shows the floor plan and measurement settings for the lab and home environments.



(a) Home environment (b) Lab environment

Figure 4.1: Floor plan and measurement settings of the lab and home environments from SignFi [60]

The WiFi transmitter and receiver are two laptops with Intel WiFi Link 5300 installed. They both operate at 5GHz, and the channel width is 20MHz. Note that the 802.11n CSI tool [61] only provides CSI values of 30 sub-carriers even though a 20MHz WiFi channel has 52 sub-carriers. The transmitter has three external antennas, and the receiver has one internal antenna. The transmitting power is fixed at 15dBm. All experiments are conducted in the presence of other WiFi signals.

4.2.2 Sentences Dataset

The SignFi dataset [60] consists of 276 ASL words; each of these words is segmented individually and is associated with a set of CSI instances. The CNN model presented in the previous chapter manages the prediction of a word using the received CSI measurements. Now, to be able to execute commands on connected objects, we need sentences. For this, we rely on a set of words present in the previous dataset to generate sentences. We select these words based on our context, namely home automation.

Regular conversations tend to follow Subject-Verb-Object or Subject-Verb order. Although the word order in ASL and English can be similar, ASL does not use the “to be” verb (am, is, are, was, were) or anything to indicate the state of “being.” Nor does it use

articles (a, an, the). The following table summarizes the selected words from the SignFi dataset and groups them in four different categories.

Category	Words
Direct	Door, Shower, Air Conditioner, Fan, Radio, TV, Refrigerator, Window, Washer, Computer, Aunt, Boyfriend, Brother, Cousin(Female), Cousin(Male), Daughter, Uncle, Wife, Father, Girlfriend, Granddaughter, Grandfather, Grandmother, Grandson, Husband, Mother, Nephew, Niece, Roommate, Sister, Son.
Indirect	Apartment, Bath, Room, Home, House, Kitchen, Room, Garage, Toilet.
Verb	Finish, Open, Close, Continue, Play, Read, Start, Up, Down, Stop, Telephone, Clean, Next, Previous, Help.
Time	Afternoon, Always, Friday, Hour, Midnight, Minute, Monday, Morning, Never, Night/Evening, Noon, Now, Saturday, Sunday, Thursday, Today, Tomorrow, Tuesday, Wednesday.

Table 4.2: The selected ASL words

After combining these words, we managed to build 16,287 sentences made up of at most four words, from which 2,421 classes were raised. These classes are the different commands that our system can handle. For instance, “Play Radio” and “Start Radio” are considered as the same class.

4.2.3 Tools & Libraries

To develop our system, we used the Python programming language. Python libraries include several features that allow the user to evaluate and analyze datasets and produce effective results. The Python programming language has become a robust and powerful tool for parsing data using these libraries. We use several libraries for the manipulation, the organization, and visualization of data. We rely on the following tools and libraries in our implementation:

NumPy

NumPy [73] is a Python library, which supports large multidimensional arrays and matrices, as well as an extensive collection of high-level mathematical functions for operating

on these arrays. It contains, among others: a powerful N-dimensional matrix object, sophisticated (diffusion) functions, a useful linear algebra, a Fourier transform, and random number capacities. NumPy is distributed under a liberal BSD (Berkeley Software Distribution License) license. It is developed and maintained publicly on GitHub.

Matplotlib

Matplotlib [74] is a Python library intended to plot and visualize data in the form of graphs. It can be combined with the python scientific computing libraries NumPy and SciPy [75]. Matplotlib is freely distributed under a BSD license.

Scikit-learn

Scikit-learn [76] (also known as Sklearn) is a free Python library for machine learning; it features various supervised and unsupervised learning algorithms and also modules for data preprocessing and model evaluation. Scikit-learn is distributed under the 3-Clause BSD license and is currently maintained by a team of volunteers.

Pandas

Pandas [77] is an open-source, BSD-licensed library. It is used for data manipulation based on the NumPy data structure. It also provides various functions in the analysis of finance, statistics, social sciences. This library offers tools for transforming raw data into useful data sets. It also provides several functions to access, index, merge, or group data easily.

Google Colab

Google colab is a Google research project created to help spread machine learning education and research. It offers a notebook to write and execute Python code in the browser, with no configuration required, free access to GPUs, and easy sharing. Colab makes the task easier since all the most commonly used libraries are installed and updated, besides, it is composed of an Intel Xeon processor with two cores @2.3 GHz and 12 GB RAM. It is equipped with an NVIDIA Tesla K80 (GK210 chipset), 12 GB RAM, 2496 CUDA cores @560 MHz. This configuration allows developing and training neural networks faster.

Deep learning frameworks

A deep learning framework is an interface, library, or a tool that allows us to build deep learning models more easily and quickly, without getting into the details of underlying

algorithms. They provide a clear and concise way of defining models using a collection of pre-built and optimized components. Below are some of the critical features of a deep learning framework:

- Optimized for performance.
- Easy to understand and code.
- Good community support.
- Parallelize the processes to reduce computations.
- Automatically compute gradients.

In the following, we present three of the most popular deep learning frameworks, namely TensorFlow, Keras, and PyTorch. Afterward, we conduct a comparative study between the three to choose which one suits our methodology the most.

TensorFlow

TensorFlow [78] is a software framework for numerical computations based on dataflow graphs. It is designed primarily as an interface for expressing and implementing machine learning algorithms, chief among them deep neural networks. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and developers quickly build and deploy ML-powered applications. TensorFlow was designed with portability in mind, enabling these computation graphs to be executed across a wide variety of environments and hardware platforms. It provides stable Python and C++ Application Programming Interfaces (APIs), as well as non-guaranteed backward compatible API for other languages. Pure TensorFlow, however, abstracts computational graph-building in a way that may seem both verbose and not-explicit. Although TensorFlow is flexible and portable, it remains difficult to implement since it is low-level and requires lengthy code.

Keras

Keras [79] is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model. Keras was initially developed for researchers to enable fast experimentation. It is a model-level library, providing high-level building blocks for developing deep-learning models. It does not handle low-level operations such as tensor manipulation and differentiation. Instead, it relies on a spe-

cialized, well-optimized tensor library to do so, serving as the backend engine of Keras. Rather than choosing a single tensor library and tying the implementation of Keras to that library, Keras handles the problem in a modular way; thus, several different backend engines can be plugged seamlessly into Keras. Currently, the three existing backend implementations are the TensorFlow backend, the Theano [80] backend, and the Microsoft Cognitive Toolkit (CNTK) backend [81][7]. The Keras ease of implementation and abstraction of the deep learning complexities comes at the price of flexibility. Keras is indeed more readable and concise than other frameworks, allowing to build deep learning models faster while skipping the implementation details. Not considering these details, however, limits the opportunities for exploration of the inner workings of each computational block in the deep learning pipeline, which makes it difficult for debugging.

PyTorch

PyTorch [82] is an open-source offering from Facebook that facilitates writing deep learning code in Python. It has two lineages. First, it derives many features and concepts from Torch [83], which was a Lua-based neural network library that dates back to 2002. Its other major parent is Chainer [84], created in Japan in 2015. Chainer was one of the first neural network libraries to offer an eager approach to differentiation instead of defining static graphs, allowing for greater flexibility in the way networks are created, trained, and operated. The combination of the Torch legacy plus the ideas from Chainer has made PyTorch popular over the past couple of years. PyTorch offers a low-level environment for experimentation, giving the user more freedom to write custom layers and look under the hood of numerical optimization tasks.

Comparison between the frameworks

We propose to conduct a comparative study between the early presented deep learning frameworks namely TensorFlow, Keras and PyTorch. Our evaluation is based on [85] and our tests. In the following of this section, we study each of the frameworks using comparison factors that we selected during our investigations.

- Ease of use

Keras has a simple architecture compared to TensorFlow and PyTorch. It is readable, concise and the best choice for prototyping. Tensorflow on the other hand is not easy to use and requires a lot of raw code. As for PyTorch, it has a complex architecture and less

readable than Keras.

- Flexibility

The ease of use comes at the price of flexibility. That is why Keras offers less flexibility than other deep learning frameworks. It might be too high-level and not always easy to customize. On the other hand, PyTorch and TensorFlow offer a comparatively lower-level environment, giving the user more freedom to write custom layers and look under the hood of numerical optimization tasks.

- Debugging

In Keras, there is usually less frequent need to debug simple networks. However, in the case of TensorFlow, it is difficult to perform debugging. PyTorch on the other hand has better debugging capabilities as compared to the other two. Nevertheless, TensorFlow offers better visualization using the TensorBoard toolkit, which allows developers to debug better and track the training process. Pytorch, however, provides only limited visualization.

- Portability

PyTorch saves models in pickles that implement binary protocols for serializing and deserializing a Python object structure, which are Python-based and not portable. As for TensorFlow and Keras, the easiest way of deploying models is by using TensorFlow Serving, a flexible, high-performance serving system for machine learning models, designed for production environments.

- Performance

The performance is comparatively slower in Keras, while TensorFlow and PyTorch provide a high level of performance. Nevertheless, differences in speed benchmarks are not the main criterion for choosing a framework since performance obstacles are caused by failed experiments, unoptimized networks, and data loading; not by the raw framework speed.

- Community

Currently, Keras is the most popular deep learning framework. Followed by TensorFlow and PyTorch, it has gained immense popularity due to its simplicity when compared to the other two. Although of the three main frameworks, PyTorch is the least popular, trends show that this may change soon. The Gradient [86] published a blog showing dramatically the advancement and acceptance of PyTorch in the scientific community (based on the

number of papers implemented at major conferences (CVPR, ICRL, ICML, NIPS, ACL, ICCV, etc.). As you can see from Fig. 4.2, In 2018, PyTorch was obviously a minority, although at major conferences it was disproportionately preferred by researchers in 2019.

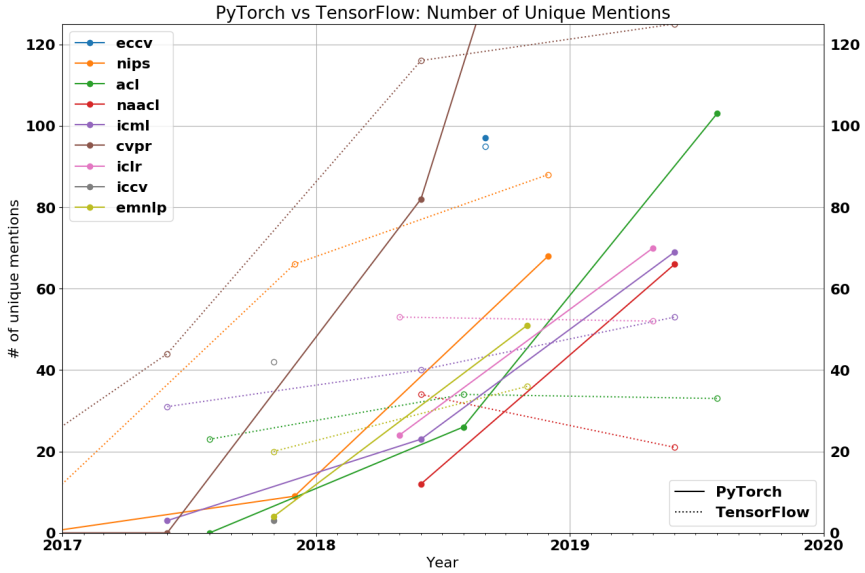


Figure 4.2: Pytorch vs TensorFlow: Number of Unique Mentions

An important point to consider is that since TensorFlow 2.0, Keras is integrated as a sub-module thus can be accessed directly via the TensorFlow library. This allows putting pure TensorFlow code into Keras models and benefiting from the best of both frameworks. Therefore, in our implementation, we leverage the flexibility and performance of TensorFlow and the ease of use of Keras by simply using the TensorFlow library.

4.2.4 Experimentation

Word-level recognition

Once the preprocessed CSI measurements are extracted from the SignFi [60] dataset as an array of multiple values, we change the data shape, so it fits the CNN model. The size of each CSI matrix is (3, 30). There are 200 CSI samples for each sign gesture, so the size of the CSI trace for each sign gesture is (3, 30, 200). First, we extract the amplitude and phase, each with a size of (3, 30, 200) from the CSI values. Therefore, they are combined and reshaped to a tensor of (200, 60, 3). As for the labels, the dataset provides 276 different classes. We use one-hot encoding to represent the categorical variables that return a binary matrix representation of the classes.

a) Model architecture

We implemented the CNN network responsible for recognizing the words as a sequence of multiple layers. In the following, we describe the sequence of layers used in our implementation using Keras by mentioning the parameters used in each of them:

Convolutional layer

The Keras Conv2D class creates a convolution kernel that is convolved with the input layer to produce a tensor of outputs. The first required Conv2D parameter is the number of filters that the convolutional layer learns. After a series of tests, we found that 16 filters are the best fit for our model. The second required parameter is the filter size, a 2-tuple specifying the width and height of the 2D convolution window. In our case, we use filters of size (5×5). Afterward, since it is the first layer in our model, we explicitly specify our input shape so that the CNN architecture has somewhere to start, then all the other shapes are calculated automatically based on the units and particularities of each layer. The input shape follows the shape of our data excluding the sample axis namely (200, 60, 3).

Batch normalization layer

The batch normalization layer normalizes the activations of the previous layer at each batch, i.e., applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. BN can have a dramatic effect on optimization performance, especially for convolutional networks and networks with sigmoidal nonlinearities [19]. We preserve the default parameters' values in our model provided by Keras and add the BN layer before the activation layer.

Rectified Linear Unit layer

ReLU layer provides fast and practical training for deep neural networks since its activation function is easy to compute and optimize. It has been shown more effective than traditional activations, such as logistic sigmoid and hyperbolic tangent, and is widely used in CNNs [19].

Average Pooling layer

An average pooling layer involves calculating the average for each patch of the feature map. In our case, we use a pool size of (3×3). It means that each (3×3) square of the feature map is downsampled to the average value in the square.

Fully connected layer

Fully connected layers are defined using the Dense class in Keras. We can specify the number of units or nodes in the layer as the first argument. The units of each layer will define the output shape (i.e., the shape of the tensor that is produced by the layer, and that will be the input of the next layer). In our case, since it is the last layer, the number of units is the number of gestures to classify, i.e., 276.

Softmax layer

The output layer typically uses either a softmax or sigmoid activation function for classification. The softmax activation function returns the probability distribution over mutually exclusive output classes. Since we deal with a multi-class classification task, we use a softmax function on the top of the neural network. Activations in Keras, can either be used through an activation layer or the activation argument supported by all forward layers. We preferred to dedicate a layer to activation functions to make the model clearer and represented as a piece of blocks, as shown in Fig. 4.3.

```
model = Sequential()
model.add(Conv2D(numKernel, kernelSize,
                 input_shape=csi_tensor.shape[1:]
                 ))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=poolSize))
model.add(Flatten())
model.add(Dense(numClasses))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer= optimizer,
              metrics=['accuracy'])
```

Figure 4.3: The model's Keras implementation

b) Model training

Once the model architecture is defined, the learning process is configured in the compilation step, where we specify the optimizer and loss function that the model should use, as well as the metrics to monitor during training. The loss function is used to find error or deviation in the learning process. Keras provides a set of loss functions in the losses module. The best loss function to use in our case is categorical cross-entropy. It measures the distance between two probability distributions. Here, between the probability distri-

bution output by the network and the exact distribution of the labels. By minimizing the distance between these two distributions, the network is trained to output value as close as possible to the actual labels.

On the other hand, optimization provides a way to minimize the loss function for deep learning. A necessary process to optimize the input weights by comparing the prediction and the loss function. In our model, we use Stochastic Gradient Descent (SGD). The strength of SGD is that it is a popular algorithm for training neural networks due to its robustness in the face of noisy updates [17]. Its implementation is straightforward with Keras using the optimizer sub-module. By default the learning rate is set to 0.01, which is suitable for our model. As for the metrics, since it is a classification task, we use the accuracy. It creates two local variables, total and count, that are used to compute the frequency with which the predictions match the labels.

After compiling the model, we specify the number of epochs, which represents the number of iterations over the training data, and the batch size, which is the number of samples processed before the model is updated. In our case, the network will start to iterate on the training data in a batch size of 10 samples and 10 epochs. At each iteration, the network will compute the gradients of the weights concerning the loss on the batch, and update the weights accordingly. The following figure summarizes the CNN model architecture as well as the parametrization.

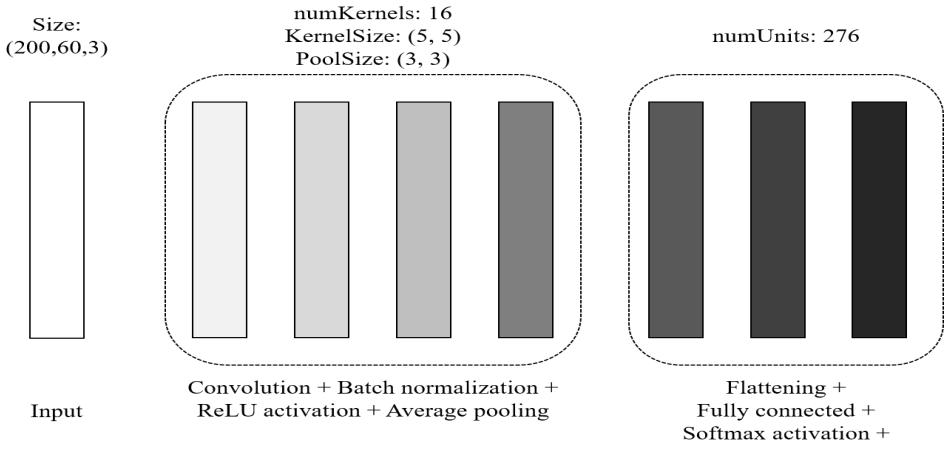


Figure 4.4: Architecture and parameter settings of the CNN model

Sentence-level recognition

As previously mentioned, sentence-level recognition is considered as an NLP task. Giving a set of sentences labeled in different classes (which represent the commands), we need to build a model capable of determining the desired command from them. Before feeding our raw sentences to an ML model, we first go through a preprocessing stage. It consists in making our data edible for any ML model, namely digital data. In our implementation, we pass our sentences through a Tokenizer. Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. We implement tokenization with Keras using the Tokenizer class. Once it fits our sentences, each word is given an index. Therefore, we just have to transform our sentences using these indexes. After that, since the recurrent neural network expects sequences with the same length, we have to map the sentences to the same length. For this, we use a Keras preprocessing method that pads sequences to the same length that is four. Every sequence with a length lower than four is completed with zeros, and the RNN subsequently understands that these values are irrelevant. As for the labels, we use one-hot encoding to represent the categorical variables that return a binary matrix representation of the 2,421 classes.

a) Model architecture

Embedding layer

Keras offers an Embedding layer that can be used for neural networks on text data. It requires that the input data be integer, so each word is represented by a unique integer. This data preparation step has been performed using the Tokenizer. The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. The Embedding layer is defined as the first hidden layer of a network. It must specify three arguments, namely the input and output dimensions and the input length. The input dimension is the size of our vocabulary, which is 74. The output dimension is the size of the vector space in which words will be embedded. After a series of tests, we set the value to 32. As for the input length, it is the length of input sequences that is 4. The output of the Embedding layer is a 2D vector with one embedding for each word in the input sequence.

Recurrent layer

There are three built-in RNN layers in Keras, namely SimpleRNN, GRU, and LSTM. All recurrent layers follow the same procedure and accept the same keyword arguments. Nevertheless, after testing the three layers on our data, the LSTM has lower training time and better accuracy. Thus, we use LSTM cells in our architecture. The only arguments that we are interested in are the number of units and the dropout. We empirically set the number of units to 512 that represents the dimensionality of the output space. As for the dropout rate, it sets input units to 0 with a frequency of rate (a float between 0 and 1; a fraction of the input units to drop) at each step during training time, which helps prevent overfitting. We set it in our case to 0.1. The default activation function for LSTM layers is Tanh and Sigmoid as the gating function. We set these arguments to their default values as well as the rest of the parameters.

Fully connected & Softmax layers

A fully connected and a Softmax layer follow the recurrent layer. We set the number of classes as the number of units i.e. 2421 using the dense class. Note that it is deprecated to use softmax activation function for a high number of classes. In such cases, it is more recommended to use hierarchical softmax or negative sampling [17]. Since Keras does not support it, we will consider it for future work.

b) Model training

Similarly to the CNN network, we use categorical cross-entropy loss function and accuracy metric since we deal with a multi-class classification task. However, we use Adam optimizer with $3e^{-4}$ learning rate based on our tests and because Adam combines features of many optimization algorithms that is why it is currently recommended as the default method to use [20]. On the other hand, we set the batch size and epochs to 10 and 10 respectively for training. Fig. 4.5 summarizes the LSTM model architecture as well as the parametrization.

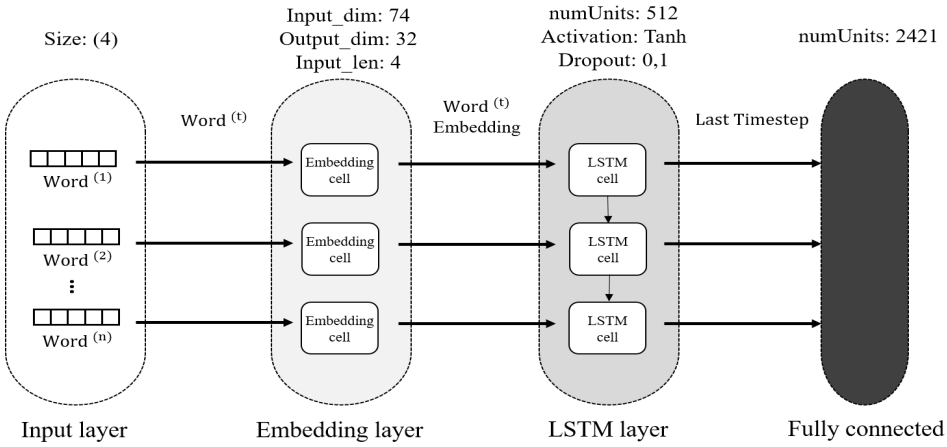


Figure 4.5: Architecture and parameter settings of the LSTM model

Sequence-learning algorithm

In the sequence-based architecture, the system directly uses a sequence of CSI measurements to predict the desired action. These sequences are constructed using the sentences dataset mentioned in Sec. 4.2.2 where each word has been replaced by its own CSI measurements. Once done, our dataset consists of $16,287 \times 20 = 325,740$ CSI sequences ($Numberofsentences \times numberofinstancesperword$) with a length ranging from 1 to 4 representing 2,421 command classes. However, the hardware used in our implementation does not allow us to use all of these sequences. Therefore, only sequences with a length ranging from 1 to 3 are used. This reduces the number of sequences to 17,920, and the number of classes to only 180. Thereafter, to perform the sequence recognition, we rely in our system on a convolutional recurrent neural network model. It expects the sequences to have the same length, thus we use the Keras preprocessing method that pads sequences to the same length that is three.

a) Model architecture

The model input is a CSI sequence with the following format (batch, time, rows, cols, channels). Limiting the sequences by 3 per sample and the CSI tensor format (200, 60, 3), the input final format is (batch, 3, 200, 60, 3). The architecture starts with a Convolutional LSTM (ConvLSTM) layer. It is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional. The ConvLSTM class on Keras contains a wide range of attributes. The ones that interest us most are the number of filters, filter size and the recurrent activation function. The number of filters

and the filter size are the same parameters seen in the convolutional layer, we set their values to 8 and (3×3), respectively. As for the recurrent activation function, by default Keras uses the hard-sigmoid function that is defined as:

$$hardSig(x) = \max(\min(0.25x + 0.5, 1), 0) \tag{4.1}$$

The ConvLSTM layer is followed by a batch normalization layer, a ReLU activation function and an average pooling layer with (3×3) pool size. We used ReLU over Tanh because ReLU is not computationally expensive and converges faster than $Tanh$. Following this block of layers comes the Flatten layer to convert the data shape to 1D vector and the fully connected layer with 180 units.

b) Model training

Once the architecture of our model is in place, we specify the training options, namely the loss function, the metrics and the optimizer. The loss function and metrics are categorical cross entropy and accuracy since it is a multi-classification task. As for the optimizer, according to our tests, the SGD optimizer fits well our model in terms of accuracy and training time. We also set the number of epochs and batch size to 10. The following figure shows the structure of our ConvLSTM model with hyperparameters.

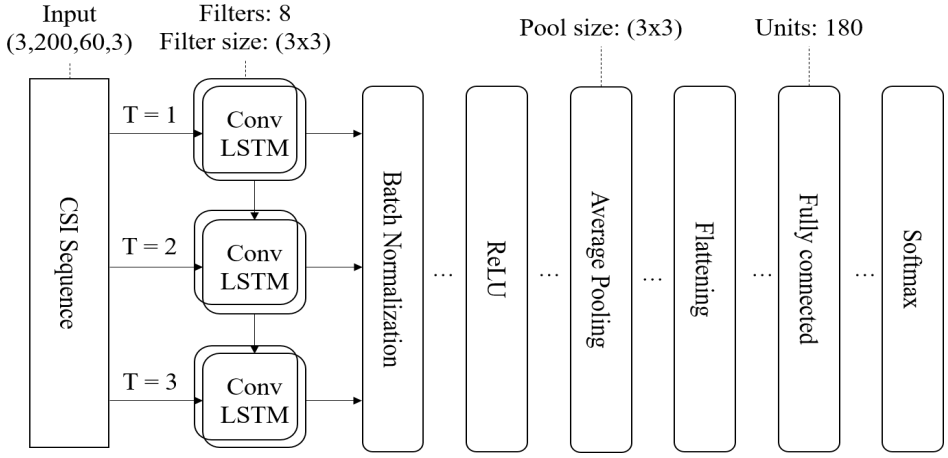


Figure 4.6: ConvLSTM structure with hyperparameters

4.3 Evaluation

In this section, we present the performance evaluation of the proposed system. This through the evaluation of the system on different datasets, the comparison with other systems proposed in the literature and other architectures that suit our system, as well as the parametric study of the two implemented models.

4.3.1 Recognition Accuracy

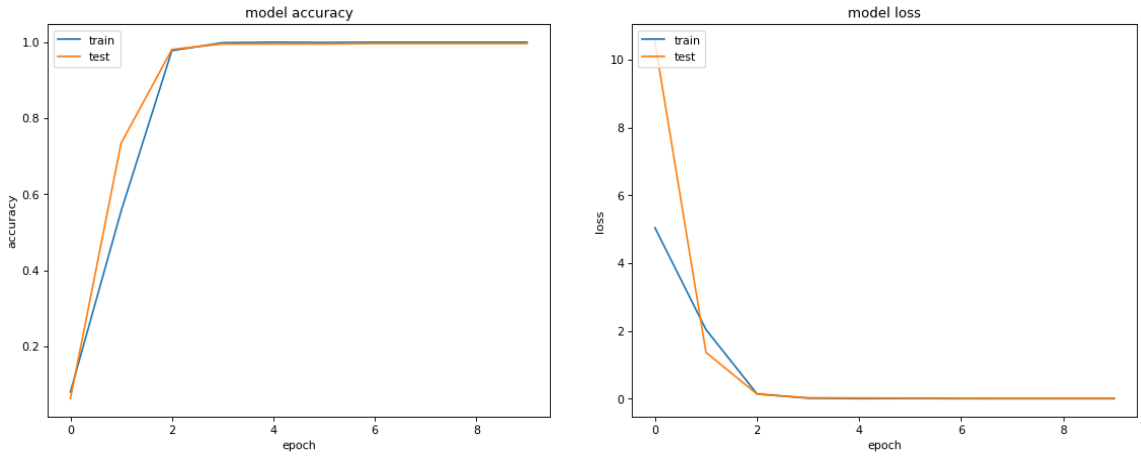
Word-level

In our work, we collect the CSI measurements from the datasets proposed by SignFi [60]. The latter measurements are recorded in different environments and experimentation settings, which led to forming three distinct datasets namely in a laboratory environment, in a home environment, and a dataset recorded by five different users. In this section, we propose to evaluate our system performance on the three datasets as well as on the combination of lab and home environments. The SignFi dataset provides a limited amount of data for the model to learn general representations. One of the strategies to overcome this situation is to use k-Fold cross-validation. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure uses the scoring provided in the given estimator (accuracy in our case) and has a parameter called k that refers to the number of groups that a given data sample is split to. As such, the procedure is called k-fold cross-validation. In our implementation, we used 5-fold cross-validation that splits the dataset into five folds of approximately equal size. The first fold is treated as a validation set while the remaining are training set. The measure reported by cross-validation is the average of the accuracies computed in the loop. This approach can be computationally expensive, but does not waste too much data, which is a major advantage in problems where the number of samples is very small.

a) Laboratory Environment

The laboratory environment dataset includes 5,520 instances of 276 gestures recorded by only one user. The distance between the transmitter and receiver is 230cm and the angle between the transmit antenna array and the direct path is about 40 degrees. Compared to the home environment the lab has more surrounding objects, leading to a more complex multi-path environment. After running a 5-fold cross validation on the 5,520 instances,

the average recognition accuracy resulted is $99,71\%$ for the 276 gestures. As for the training time, using Google Colab hardware, it lasts 29 seconds. The following figure shows the accuracy and loss curves in the function of the epochs. We can notice that the model converges from the 3rd epoch.



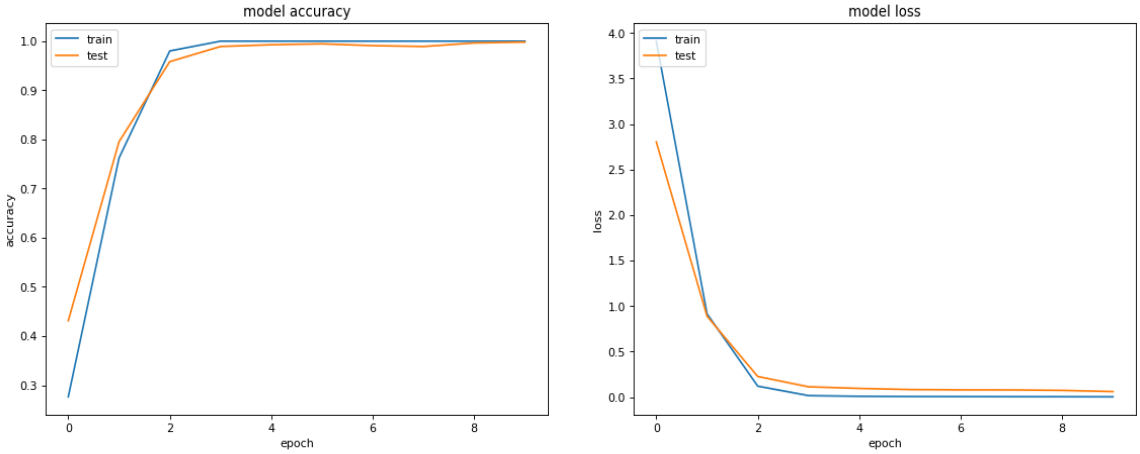
(a) Model accuracy vs. epochs

(b) Model loss vs. epochs

Figure 4.7: Laboratory environment performance evaluation

b) Home environment

The home environment includes 2,760 instances for 276 gestures recorded by the same user as the lab environment. The distance between the transmitter and receiver is 130cm and the transmit antenna array is orthogonal to the direction from the transmitter to the receiver. After running 5-fold cross-validation on the 2,760 instances, the average recognition accuracy resulted is $99,49\%$ for the 276 gestures and training time of 11 seconds. The following figure shows the accuracy and loss curves vs. the number of epochs. We also notice here that the model converges from the 3rd epoch although the number of instances is half that of the laboratory dataset and the experimentation settings are different. The home environment should have shown the same results as the lab environment or better if the number of instances was the same since the experimentation settings are more favorable.



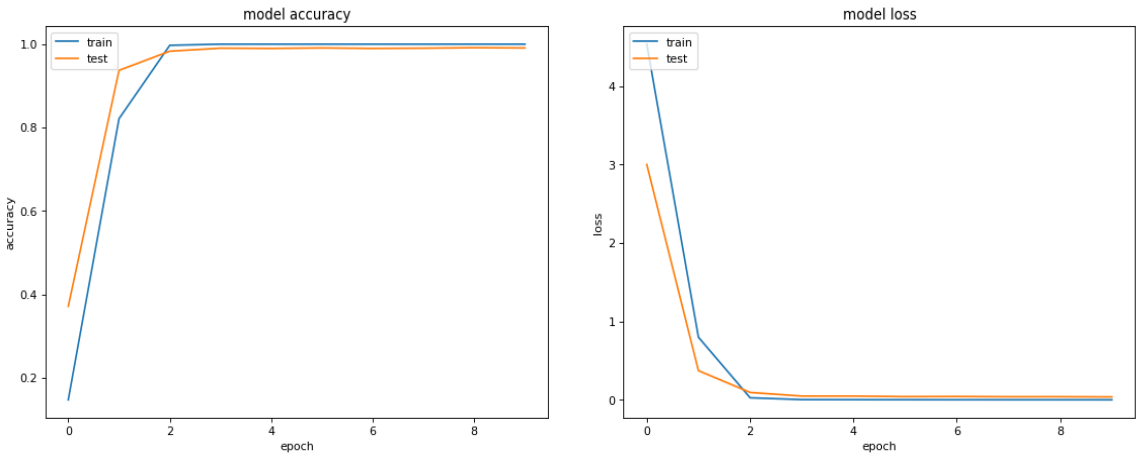
(a) Model accuracy vs. epochs

(b) Model loss vs. epochs

Figure 4.8: Home environment performance evaluation

c) Home and lab combination

The combination of the last two datasets gives 8,280 instances for 276 gestures, all recorded by the same user. The major differences of these two environments are (1) dimension of the room, (2) distance between the transmitter and receiver, (3) angle between the transmit antenna array and the direct path, (4) multi-path environments, and (5) number of instances. After running 5-fold cross-validation on the 8,280 instances, the average recognition accuracy resulted is $99,09\%$ for the 276 gestures and 31 seconds of training. The following figure shows the accuracy and loss curves vs. the epochs. The combination of environments is an important factor to consider in case there are several rooms in a house and each has a different environment. Although we only use two environments in our case, the results of our model show that the system remains robust.



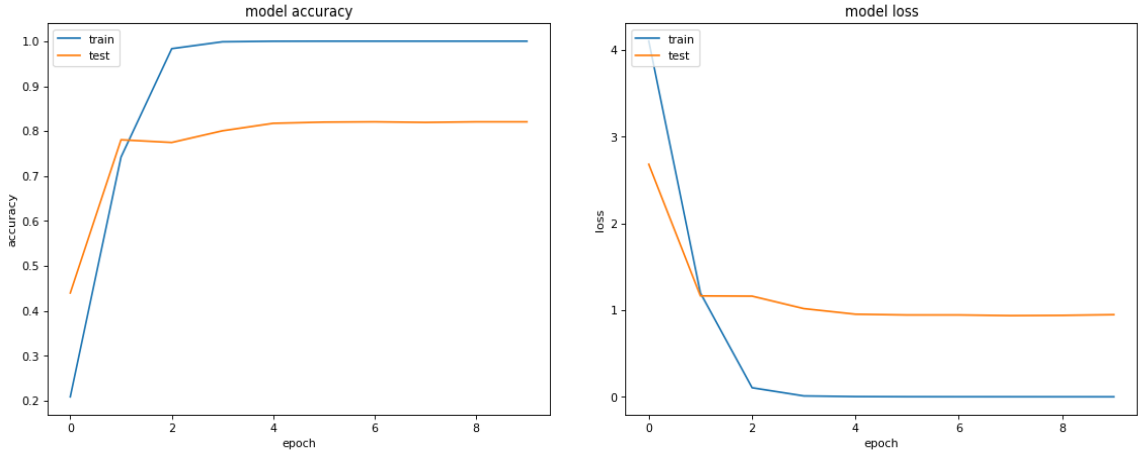
(a) Model accuracy vs. epochs

(b) Model loss vs. epochs

Figure 4.9: Home and lab combination performance evaluation

d) Multiple users

Another dataset in the laboratory environment is recorded, this time with 7,500 instances for 150 gestures performed by five different users. Different users may have different gesture durations and slightly different hand/finger movements for the same sign word. One of the users has different experiment settings, such as laptop displacement, surrounding objects, desk and chair arrangements, etc., even though they are in the same lab environment. During our evaluation, we first tested our model performance on the four users that have the same experiment settings. The recognition accuracy obtained after running 5-fold cross-validation is 92%. However, when adding the user records with different settings, the recognition accuracy collapses to 82% and training time of 26 seconds. This difference shows that experimentation settings considerably influence the performance of the system. Nevertheless, as shown in the following figure, there is a gap between training and testing accuracies. We conclude that the model is overfitting in this situation. More data would help to prevent overfitting and allow the model to generalize better since 1,500 instances/user is not enough.



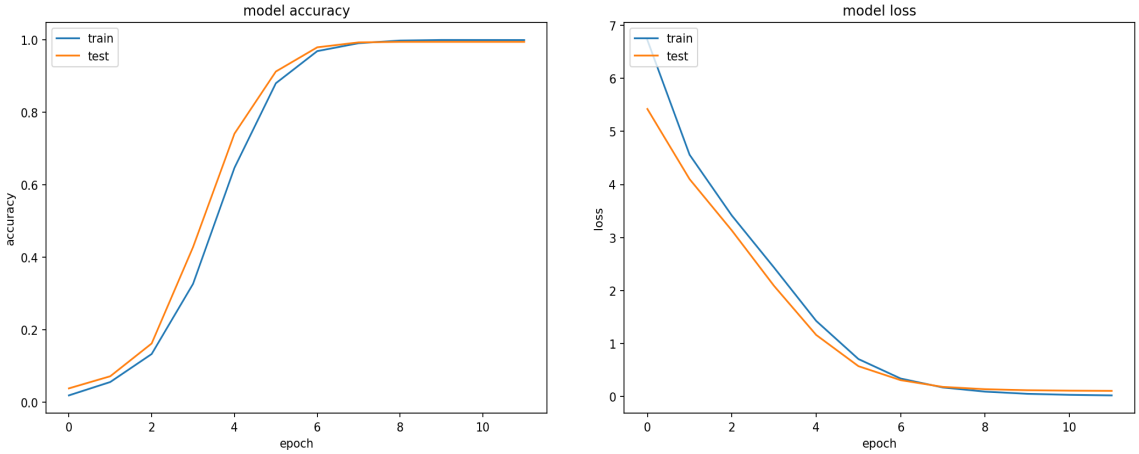
(a) Model accuracy vs. epochs

(b) Model loss vs. epochs

Figure 4.10: Multiple users scenario performance evaluation

Sentence-level

As for the sentence level recognition, the nature of the datasets and the experimental parameters are independent of the performance of the model. Since in our implementation, we use the words present in the datasets to generate sentences. As explained in the previous chapter, we have selected 74 words that can be used in the field of home automation. With these words, we managed to generate a set of 16,287 sentences through grammar. From these sentences, we can distinguish 2421 classes. We evaluated the performance of our LSTM model using 5-fold cross-validation. Despite a large number of classes, the model has an average accuracy of 99.08% and a training time of 114 seconds. Fig. 4.11 illustrates the evolution of the accuracy and loss curves on the training as well as on the testing sets through the epochs. We notice from this figure that the model converges starting from 7th epoch using the LSTM.



(a) Model accuracy vs. epochs

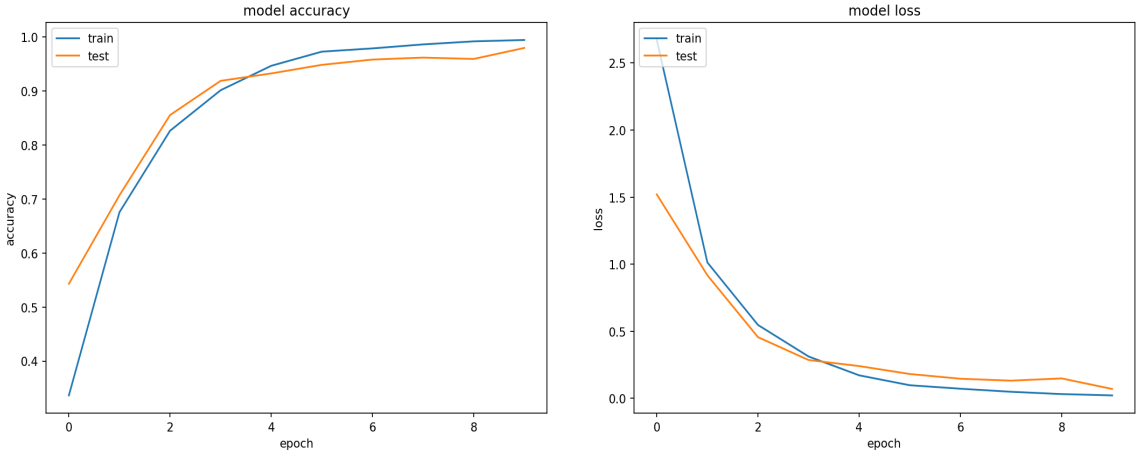
(b) Model loss vs. epochs

Figure 4.11: Sentence-level performance evaluation

Sequence recognition

To build the CSI sequences, we use the measurements from a single dataset, i.e. the laboratory environment dataset. Note that for hardware performance reasons, only sequences with a length ranging from 1 to 3 are used. This reduced the number of sequences to 17,920, and the number of classes to only 180. Therefore, 5-fold cross validation is used to evaluate the performance of our model, and the ConvLSTM achieves an average accuracy of 97.40% in training time of 480 seconds. Fig. 4.12 shows the accuracy and loss curves on the training as well as on the testing sets through the epochs.

We would like to mention that the way we constructed the sequences is not recommended. Because in practice, the measurements are received continuously and not manually segmented word by word as in the dataset that we use. Nevertheless, this method has been proposed to show the efficiency of ConvLSTMs in the case of a CSI sequence, since in future works we are considering building a set of CSI sequences expressing a sentence made with ASL.



(a) Model accuracy vs. epochs

(b) Model loss vs. epochs

Figure 4.12: ConvLSTM model evaluation

4.3.2 Parametric Study

During our work, we developed three distinct neural network models, the CNN interpret CSI measurements and predict the sign made by the user. While the LSTM uses the words generated by the CNN to predict the desired action by the user. As for the ConvLSTM, it takes a CSI sequence to predict directly the desired action. To tune the hyper-parameters that best suit our three models i.e. the best accuracy and training time, we used one of the optimizers offered in the scikit-learn [76] library called Grid Search. Grid search is a process that searches exhaustively through a manually specified subset of the hyper-parameter space of the targeted model. In our case, we specified the number and size of filters in the CNN and ConvLSTM layers, the pool size of the pooling layer as well as the optimizer, the number of epochs and the batch size in the model's hyper-parameter subset. As for the LSTM, we specified the output dimension in the embedding layer, the number of cells in the LSTM layer and the optimizer, the number of epochs, and the batch size. Thus, the hyper-parameters selected in our implementation rely on the result of the grid search, which computes the most optimal values.

On the other hand, the structure of the models and the selection of the layers have a major importance in the optimization of the model. In our implementation, we relied on the work of SignFi [60] as well as on the recommendations of deep learning pioneers [7], [19], [18]. In this section, we discuss the contribution of some implemented layers

in model optimization. However the ConvLSTM model is not covered here since it has almost the same layers as the CNN and they have the same impact.

CNN model

a) Batch normalization layer

Batch normalization can have a dramatic effect on optimization performance, especially for convolutional networks [19]. Fig. 4.13 shows the impact of batch normalization on the recognition accuracy in every used dataset. We can notice that BN has a considerable effect on the Laboratory + Home dataset. This can be explained by the different experimentation settings of the samples that leads to a different range of values.

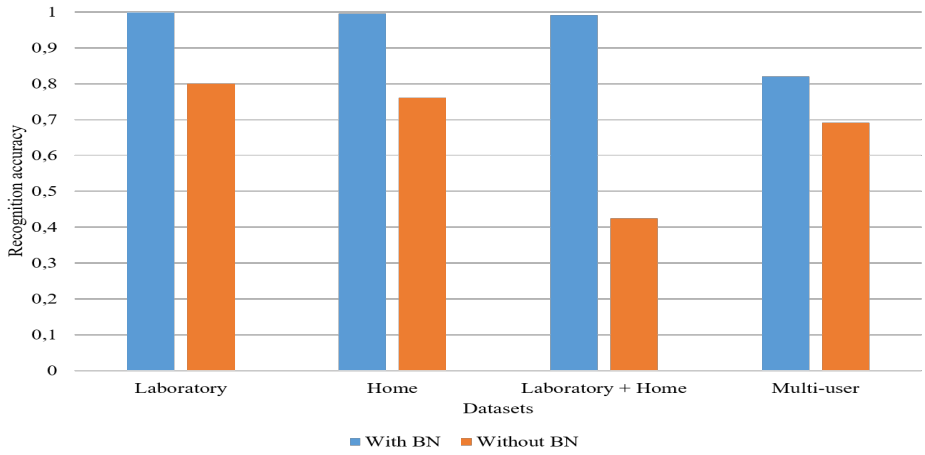


Figure 4.13: Batch normalization impact on the accuracy

b) Dropout layer

As a matter of testing, we tried to implement a dropout layer in our CNN architecture. We placed this layer once between the pooling and flattening layers to apply its effect on the convolutional region, and another time right after the fully connected layer with a dropout rate of 0.25. Dropout mechanism is supposed to prevent overfitting and speed up the training. However, in our case it had no impact on the unseen data accuracy or the training time. In contrast to SignFi [60] that has seen its accuracy considerably increased from 59% to 88% by using a dropout layer on the convolutional unit. Fig. 4.14 shows the dropout layer impact on the recognition accuracy when applied on the Fully Connected layer. We can notice that the model is underfit since there is a huge gap between the testing and training curves.

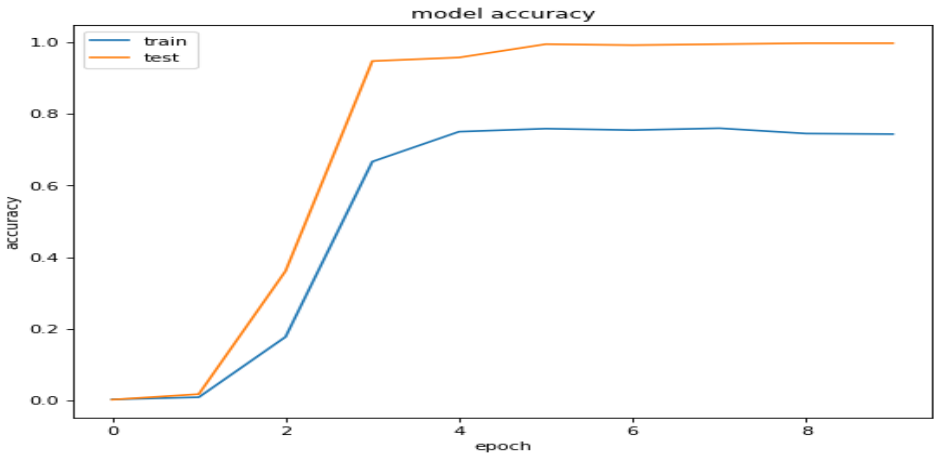


Figure 4.14: Dropout impact on the FC layer

However, when the dropout is applied on the convolution region, we cannot notice an impact on the recognition accuracy as shown in Fig. 4.15. The reason is that the batch normalization reduces generalization error and allows dropout to be omitted, due to the noise in the estimate of the statistics used to normalize each variable [19]. Further, it may not be a good idea to use batch normalization and dropout in the same network.

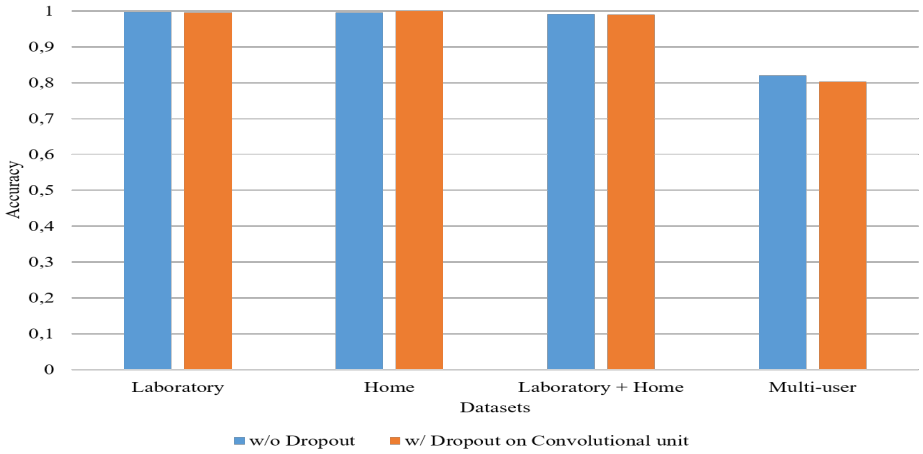


Figure 4.15: Dropout impact on the convolutional unit

c) Average-pooling layer

The pooling layer reduces the number of connections to the following layers by down sampling. It returns the average or the maximum of the inputs within a rectangular region. The pooling size of our system is (3×3). Nevertheless, to know whether the average or the maximum value suits our system the most, we tested both on our model.

The following figure shows the difference between using an average or a max-pooling layer. There is not a difference as big as mentioned in SignFi [60] namely 10% lower when using the max-pooling layer.

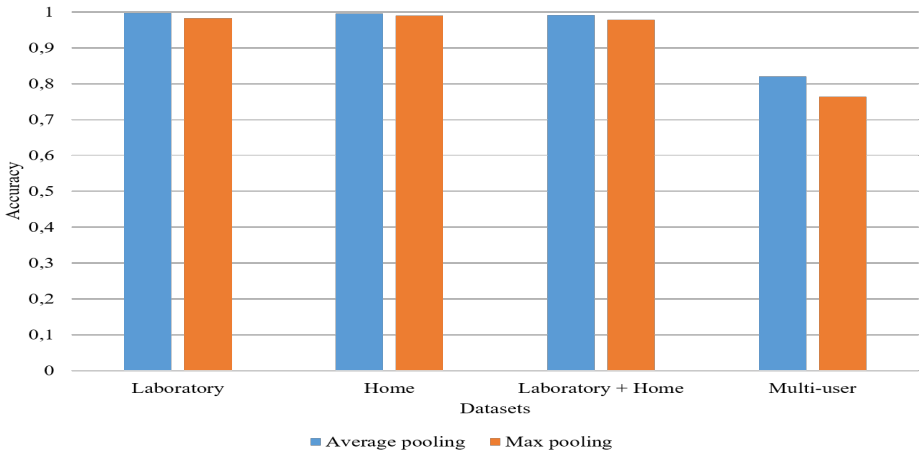
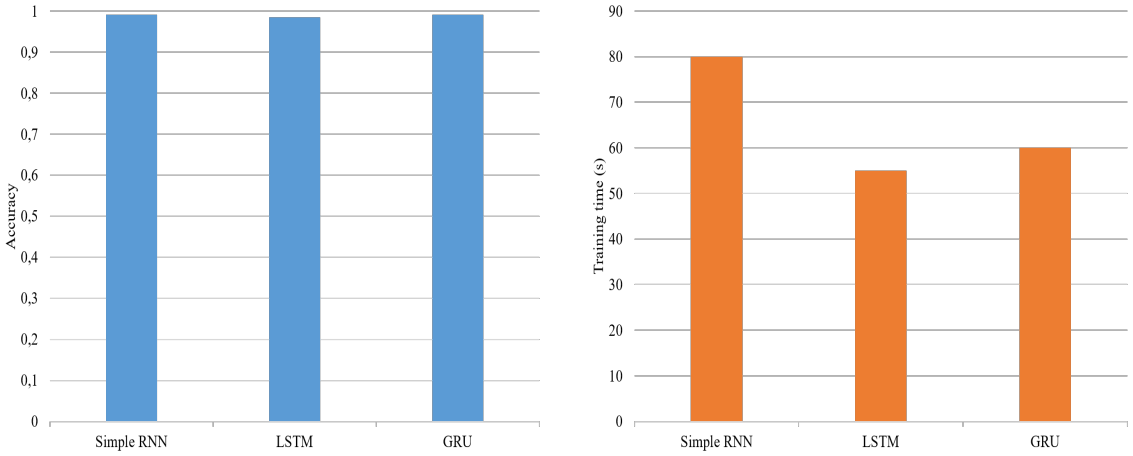


Figure 4.16: Average vs. Max pooling layer

LSTM model

RNN preserves information from inputs that has already passed through it using the hidden state. During the implementation of our system, we tried different recurrent layers to find the one that suits the most our system in terms of accuracy and training time. These layers are RNN, LSTM and GRU. We found out that the three of them have the same average accuracy namely $99,12\%$, $98,52\%$ and $99,08\%$ for the RNN, LSTM and the GRU, respectively. Nevertheless, they do not show the same performance regarding to the training time. As shown in Fig. 4.17, the LSTM layer spend less time in the training phase compared to the RNN and the GRU. The difference is only 25 seconds in our case since we generated a small dataset, but in the case of a larger dataset, the difference in training time could be greater.



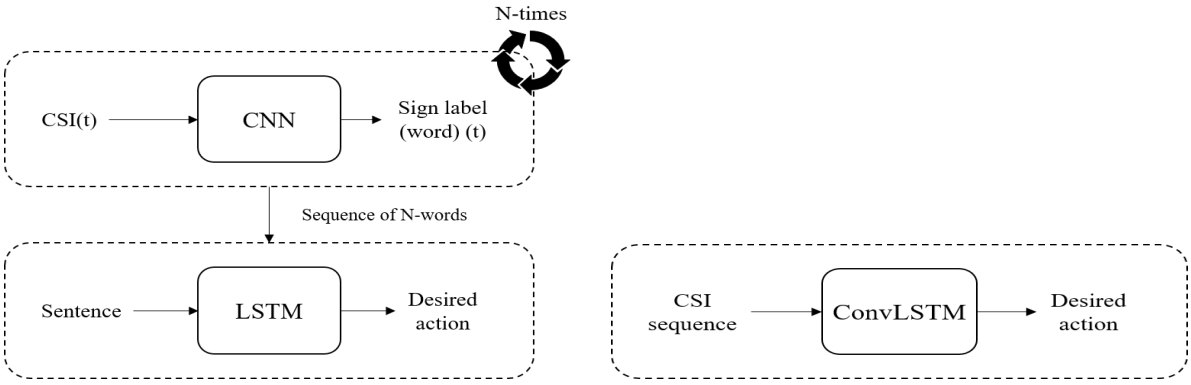
(a) Models' accuracy

(b) Models' Training time

Figure 4.17: Training time and average accuracy RNN vs. LSTM vs. GRU

4.3.3 Comparison Between the Two Methods

The following figures illustrate the process of the two architectures proposed in our solution.



(a) Word-based architecture

(b) Sequence-based architecture

Figure 4.18: Proposed architectures' process

The object of this section is to compare these two proposed architectures. We focus our comparison on two main points that are the structure and the performance.

Structure

In the word-based architecture, the system uses word-for-word segmented CSI measurements that go through two deep learning models to predict the desired action. While the sequence-based architecture relies on CSI sequences representing a sentence in ASL

to predict the desired action. The fact that there are two models in one architecture is a drawback in system deployment, as each model is deployed on a server with a separate port. Therefore, we have to host both models, which can be expensive. On the other hand, we consider in future works to build a dataset with CSI sequences reflecting ASL signs, which would be more appropriate for our context i.e. home automation. In this scenario, we would use an architecture that supports sequences.

Performance

The key performance indicators taken into consideration here are accuracy, training time and the number of classes. The word-based architecture combines a CNN for the word level recognition and an LSTM for the sentence level recognition. The CNN in the laboratory environment has an average accuracy of 99.71% and a training time that lasts 29 seconds. As for the LSTM, It achieves an average accuracy of 99.08% in training time of 114 seconds. Thus, the combination between them is an average accuracy of 99.35% and a training time of 143 seconds to handle an amount of 2,421 classes. On the other hand, the sequence-based architecture has only one model i.e. the ConvLSTM that in the laboratory environment achieves an average accuracy of 97.40% and a training time of 480 seconds for only 180 classes. Therefore, the word-based architecture outperforms the sequence-based.

4.3.4 Comparison With Other Works

In our work, we cited numerous works that aim at the recognition of sign language. Nevertheless, most of them perform only isolated word recognition, not sentence level translation. Thus, we make a comparison between our work and the current state of the art using the CNN model for word level recognition and the combination of CNN LSTM and ConvLSTM for sentence level recognition.

Word-level recognition

In the following table, we cite the word-level sign language recognition systems that we mentioned in the state-of-the-art section.

Ref	Approach	Signal/Device	Num. gestures	Recognition algorithm	Recognition Accuracy
[36]	Vision-based	Webcam	40	SVM, ANN, kNN	92.23%
[37]		Video camera	36	P2-DHMM	98%
[47]	Sensor-based	Accelerometers, Gyroscopes	103	B-LSTM	99.2%
[50]		EMG, Accelerometers, Gyroscopes	70	CNN	92.4%
[60]	Wireless-based	CSI	276	CNN	98.91% (home), 98.01% (lab)
Our design		CSI	276	CNN	99.49% (home), 99.71% (lab)

Table 4.3: Comparison of word-level SLR systems

Signs that vision-based systems rely on are alphanumerical in contrast to the other approaches that use word signs. The reason is that the majority of alphanumerical signs are static, however systems based on wireless and sensors are less sensitive to static gestures but more to motions. However, in our case we are interested in words more than the alphabet to express actions on connected objects. We also mentioned in Tab. 3 the work that inspired us i.e. SignFi [60]. We achieved better results by tuning the CNN model.

Sentence-level recognition

In the following table, we cite the word-level sign language recognition systems that we mentioned in our thesis.

Ref	Approach	Signal/Device	Num. sentences	Max. length	Recognition algorithm	Recognition Accuracy
[47]	Sensor-based	Accelerometers, Gyroscopes	73	2	B-LSTM	98.96%
[50]		EMG, Accelerometers, Gyroscopes	100	8	CNN, B-LSTM	93.1%
Word-based	Wireless-based	CSI	2421	4	CNN, LSTM	99.35%
Sequence-based		CSI	180	3	ConvLSTM	97.40%

Table 4.4: Comparison of Sentence-level SLR systems

In the literature, there is not much work done in recognizing sign language at the sentence level. Especially with the wireless approach. Therefore, we compare our work with two sensor-based references in sentence level SLR namely [47] and [50]. The first thing we can notice is the number of sentences that each system handles. In [47], the authors build their sentences manually using 103 common-used words. The same process is followed in [50], the authors manually construct 100 sentences using 40 ASL words. However, in our case, we automatically construct the sentences using 74 commonly used ASL signs. It leads to having sentences that do not make sense, for instance, "READ BATHROOM WINDOW". Nevertheless, in our dataset even these meaningless sentences are linked to classes that come closest to their meaning. As in the example just cited, the desired action associated with the sentence would be to open the bathroom window. As for the structure of the systems, [50] is similar to our word-based architecture (CNN + LSTM) and [47] proceeds like our sequence-based architecture (ConvLSTM).

4.4 Deployement

To demonstrate our system we proceeded to the deployment of our models. As we do not have the necessary equipment to receive CSI measurements. We designed a web application that simulates a smart home environment, which includes different rooms,

namely a bedroom, a kitchen, a living room, a bathroom and a garage. Each of these rooms has objects that will interact with the system. The web application sends CSI measurements as a request to the models, and in return, the model sends the prediction of these measurements i.e. the desired action. Depending on the desired action received, there will be an interaction in the user interface of the application. The CSI measurements that the web application sends are from the SignFI dataset but unseen to the models. The following figure illustrates the architecture of this web application.

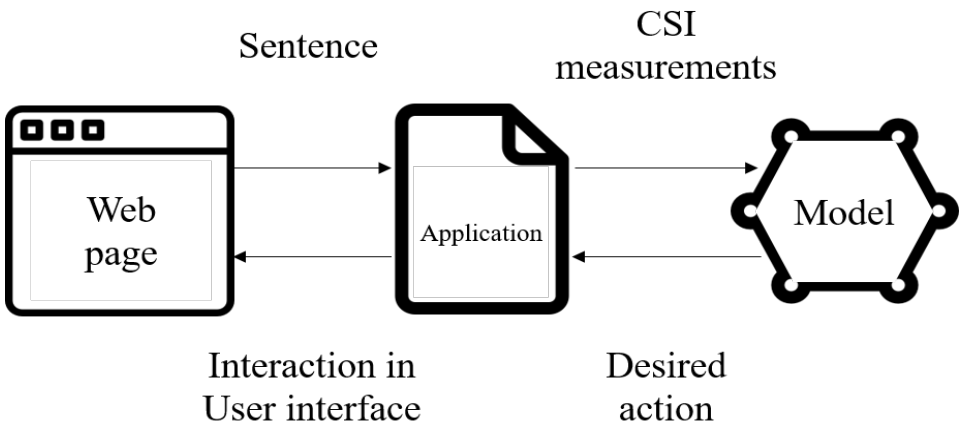


Figure 4.19: Web application architecture

We deploy deep learning models using Tensorflow serving. It is a flexible, high-performance serving system for machine learning models, designed for production environments. Once the models are deployed, we can send requests and receive responses in JSON (JavaScript Object Notation) format. As for the application, we developed it using Flask, which is a web application framework developed with Python. It is designed to make getting started quick and easy. Fig. 4.20 shows screenshots taken from the web application.

(a) Button to enter a sentence

(b) Bathroom

(c) Kitchen

(d) Living room

Figure 4.20: Screenshots from the web application

4.5 Conclusion

In this chapter, we discussed the implementation of the two architectures proposed in this thesis. We started by presenting the datasets and the tools on which we rely in our work. We used the pre-processed and word-by-word segmented CSI measurements present in SignFi public dataset. We justified the choice of the deep learning framework with a comparative study between Keras, TensorFlow and PyTorch. Thereafter, details regarding the experimentation are mentioned, going through the configuration of deep learning models and the discussion of their structure. The second part of this chapter is dedicated to the evaluation where we present the performances of the two methods as well as the impact of certain layers in the structure of models. Following this, we made a comparison between the two architectures as well as the comparison with other works in the literature.

GENERAL CONCLUSION

Context

This thesis was produced at the University of Saad dahleb Blida – Blida 1 – and the Research Centre for Scientific and Technical Information – CERIST – The main goal of this project is to conceive and realize a solution based on machine learning for sign language recognition that allows the control of a smart home environment through gestures.

Solution

We proposed an solution that is based on channel state information measurements present in the Wi-Fi signal, which reflect the gestures of sign language performed in the environment. We made two different architectures that meet the needs of our subject. The word-based architecture processes the input measurements sign-by-sign with a CNN model, and then performs sentence-level recognition with an LSTM model to predict the desired action. As for the sequence-based architecture, it processes the measurements in sequences with a ConvLSTM model to directly predict the desired action. This solution is implemented using the Python programming language and the Keras deep learning framework and is evaluated using the SignFi [60] public dataset, which offers pre-processed and segmented word-by-word CSI measurements.

Contributions

The major contributions of our work are:

- Study of the current state of the art of gesture recognition (Definition of existing approaches, analysis of existing work, establishing comparison criteria and synthesis).
- The proposition of a solution capable of classifying the users' gestures into actions executed by the smart home.
- The design of a solution for the processing of CSI sequences.
- A comparative study of several frameworks and systems implementation tools.
- Implementation and tuning of three deep learning models namely CNN, LSTM and ConvLSTM with Keras.

- Deployment of deep learning models with Tensorflow serving and creation of an application simulating the flow of the system in a Smart home environment with Flask.

Future perspectives

Our work opens a large amount of future perspectives. The most important of them are:

- Conceive our own dataset in a Smart home environment with a larger number of instances, and segmented in a manner to perform sentence level recognition.
- Build a grammar free dataset, which means that the structure of the sentence has not to follow a grammar.
- Conceive a CSI based system to translate Sign language not only for smart home actions but rather to communicate. For example with virtual assistants like Siri or Alexa.

Bibliography

[1] *Deafness and hearing loss.* URL: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss (visited on 02/19/2020).

[2] Mohamed Aktham Ahmed, Bilal Bahaa Zaidan, et al. "A review on systems-based sensory gloves for sign language recognition state of the art between 2007 and 2017". In: *Sensors* 18.7 (2018). Publisher: Multidisciplinary Digital Publishing Institute, p. 2208.

[3] Ming Jin Cheok, Zaid Omar, et al. "A review of hand gesture and sign language recognition techniques". In: *International Journal of Machine Learning and Cybernetics* 10.1 (2019). Publisher: Springer, pp. 131–153.

[4] Gary F Simons and Charles D Fennig. "Ethnologue: Languages of the World". In: *Dallas, Texas: SIL International* (). URL: http://www.ethnologue.com..

[5] Henri Wittmann. "Classification linguistique des langues signées non vocalement". In: *Revue québécoise de linguistique théorique et appliquée* 10.1 (1991). Publisher: Association québécoise de linguistique, pp. 215–288.

[6] Richard Harper. *Inside the smart home.* Springer Science & Business Media, 2006.

[7] F. Chollet. *Deep Learning with Python.* Manning Publications Company, 2017. ISBN: 978-1-61729-443-3. URL: https://books.google.dz/books?id=Yo3CAQAACAAJ.

[8] Djamel Djenouri, Roufaida Laidi, et al. "Machine learning for smart building applications: Review and taxonomy". In: *ACM Computing Surveys (CSUR)* 52.2 (2019). Publisher: ACM New York, NY, USA, pp. 1–36.

[9] Yuji Roh, Geon Heo, et al. "A survey on data collection for machine learning: a big data-ai integration perspective". In: *IEEE Transactions on Knowledge and Data Engineering* (2019). Publisher: IEEE.

[10] Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. 3rd ed. Burlington, MA: Elsevier, 2012. 703 pp. ISBN: 978-0-12-381479-1.

[11] Salvador García, Julián Luengo, et al. *Data preprocessing in data mining*. Springer, 2015.

[12] Ethem Alpaydin. "Introduction to Machine Learning Ethem Alpaydin". In: *Introduction to machine learning* (2014).

[13] Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[14] Leonard E Baum, Ted Petrie, et al. "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains". In: *The annals of mathematical statistics* 41.1 (1970), pp. 164–171.

[15] Arthur P Dempster, Nan M Laird, et al. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[16] Antal van den Bosch. "Hidden Markov Models". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 493–495. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_362. URL: https://doi.org/10.1007/978-0-387-30164-8_362.

[17] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.", 2017.

[18] Yann LeCun, Yoshua Bengio, et al. "Deep learning". In: *nature* 521.7553 (2015). Publisher: Nature Publishing Group, pp. 436–444.

[19] Ian Goodfellow, Yoshua Bengio, et al. *Deep learning*. MIT press, 2016.

[20] Aston Zhang, Zachary C. Lipton, et al. *Dive into Deep Learning*. 2019.

[21] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997). Publisher: MIT Press, pp. 1735–1780.

[22] Yanmin Zhu, Zhibo Yang, et al. "Vision based hand gesture recognition". In: *2013 International Conference on Service Sciences (ICSS)*. IEEE, 2013, pp. 260–265.

[23] Z. Zhang. "Microsoft Kinect Sensor and Its Effect". In: *IEEE MultiMedia* 19.2 (2012), pp. 4–10. ISSN: 1941-0166. DOI: 10.1109/MMUL.2012.24.

[24] Siddharth S Rautaray and Anupam Agrawal. "Vision based hand gesture recognition for human computer interaction: a survey". In: *Artificial intelligence review* 43.1 (2015). Publisher: Springer, pp. 1–54.

[25] Tejashree P Salunke and SD Bharkad. "Power point control using hand gesture recognition based on hog feature extraction and K-NN classification". In: *2017 International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2017, pp. 1151–1155.

[26] Ananyaa Sharrma, Ayush Khandelwal, et al. "Vision based static hand gesture recognition techniques". In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2017, pp. 0705–0709.

[27] Chenyang Li, Xin Zhang, et al. "Lpsnet: A novel log path signature feature based hand gesture recognition framework". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 631–639.

[28] Gongfa Li, Heng Tang, et al. "Hand gesture recognition based on convolution neural network". In: *Cluster Computing* 22.2 (2019). Publisher: Springer, pp. 2719–2729.

[29] Jaya Prakash Sahoo, Samit Ari, et al. "Hand gesture recognition using DWT and F-ratio based feature descriptor". In: *IET Image Processing* 12.10 (2018). Publisher: IET, pp. 1780–1787.

[30] Jayshree R Pansare and Maya Ingle. "Vision-based approach for American sign language recognition using edge orientation histogram". In: *2016 International Conference on Image, Vision and Computing (ICIVC)*. IEEE, 2016, pp. 86–90.

[31] Oyebade K Oyedotun and Adnan Khashman. "Deep learning in vision-based static hand gesture recognition". In: *Neural Computing and Applications* 28.12 (2017). Publisher: Springer, pp. 3941–3951.

[32] Mandeep Kaur Ahuja and Amardeep Singh. "Static vision based Hand Gesture recognition using principal component analysis". In: *2015 IEEE 3rd International Conference on MOOCs, Innovation and Technology in Education (MITE)*. IEEE, 2015, pp. 402–406.

[33] Samta Gupta and Susmita Ghosh Mazumdar. "Sobel edge detection algorithm". In: *International journal of computer science and management Research* 2.2 (2013), pp. 1578–1583.

[34] Nasser H Dardas and Nicolas D Georganas. "Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques". In: *IEEE Transactions on Instrumentation and measurement* 60.11 (2011). Publisher: IEEE, pp. 3592–3607.

[35] Paul Viola and Michael J Jones. "Robust real-time face detection". In: *International journal of computer vision* 57.2 (2004). Publisher: Springer, pp. 137–154.

[36] Joyeeta Singha, Amarjit Roy, et al. "Dynamic hand gesture recognition using vision-based approach for human–computer interaction". In: *Neural Computing and Applications* 29.4 (2018). Publisher: Springer, pp. 1129–1141.

[37] Nguyen Dang Binh, Enokida Shuichi, et al. "Real-time hand tracking and gesture recognition system". In: *Proc. GVIP* (2005). Publisher: Citeseer, pp. 19–21.

[38] Thittaporn Ganokratanaa and Suree Pumrin. "The vision-based hand gesture recognition using blob analysis". In: *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*. IEEE, 2017, pp. 336–341.

[39] Cao Dong, Ming C Leu, et al. "American sign language alphabet recognition using microsoft kinect". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2015, pp. 44–52.

[40] Philipp Koch, Mark Dreier, et al. "A Recurrent Neural Network for Hand Gesture Recognition based on Accelerometer Data". In: *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2019, pp. 5088–5091.

[41] Renqiang Xie and Juncheng Cao. "Accelerometer-based hand gesture recognition by neural network and similarity matching". In: *IEEE Sensors Journal* 16.11 (2016). Publisher: IEEE, pp. 4537–4545.

[42] Jonghwa Kim, Stephan Mastnik, et al. "EMG-based hand gesture recognition for realtime biosignal interfacing". In: *Proceedings of the 13th international conference on Intelligent user interfaces*. 2008, pp. 30–39.

[43] Zhen Zhang, Kuo Yang, et al. "Real-Time Surface EMG Pattern Recognition for Hand Gestures Based on an Artificial Neural Network". In: *Sensors* 19.14 (2019). Publisher: Multidisciplinary Digital Publishing Institute, p. 3170.

[44] Sung-Woo Byun and Seok-Pil Lee. "Implementation of Hand Gesture Recognition Device Applicable to Smart Watch Based on Flexible Epidermal Tactile Sensor Array". In: *Micromachines* 10.10 (2019). Publisher: Multidisciplinary Digital Publishing Institute, p. 692.

[45] Qin Ni, Ana García Hernando, et al. "The Elderly's Independent Living in Smart Homes: A Characterization of Activities and Sensing Infrastructure Survey to Facilitate Services Development". In: *Sensors* 15.5 (2015). Publisher: MDPI AG, pp. 11312–11362. ISSN: 1424-8220. DOI: 10.3390/s150511312. URL: http://dx.doi.org/10.3390/s150511312.

[46] Rami N. Khushaba, Sarath Kodagoda, et al. "Toward improved control of prosthetic fingers using surface electromyogram (EMG) signals". In: *Expert Systems with Applications* 39.12 (2012), pp. 10731–10738. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2012.02.192. URL: http://www.sciencedirect.com/science/article/pii/S0957417412004654.

[47] Jiahui Hou, Xiang-Yang Li, et al. "Signspeaker: A real-time, high-precision smartwatch-based sign language translator". In: *The 25th Annual International Conference on Mobile Computing and Networking*. 2019, pp. 1–15.

[48] Hisatake Sato. *Moving average filter*. Google Patents, Oct. 16, 2001.

[49] WW Kong and Surendra Ranganath. "Signing exact english (SEE): Modeling and recognition". In: *Pattern Recognition* 41.5 (2008). Publisher: Elsevier, pp. 1638–1652.

[50] Qian Zhang, Dong Wang, et al. "MyoSign: enabling end-to-end sign language recognition with wearables". In: *Proceedings of the 24th International Conference on Intelligent User Interfaces*. 2019, pp. 650–660.

[51] Marcus Georgi, Christoph Amma, et al. "Recognizing Hand and Finger Gestures with IMU based Motion and EMG based Muscle Activity Sensing." In: *Biosignals*. 2015, pp. 99–108.

[52] Oscar D Lara and Miguel A Labrador. "A survey on human activity recognition using wearable sensors". In: *IEEE communications surveys & tutorials* 15.3 (2012). Publisher: IEEE, pp. 1192–1209.

[53] Shahin Farahani. "Chapter 7 - Location Estimation Methods". In: *ZigBee Wireless Networks and Transceivers*. Ed. by Shahin Farahani. Burlington: Newnes, 2008, pp. 225–246. ISBN: 978-0-7506-8393-7. DOI: 10.1016/B978-0-7506-8393-7.00007-8. URL: http://www.sciencedirect.com/science/article/pii/B9780750683937000078.

[54] Heba Abdelnasser, Moustafa Youssef, et al. "Wigest: A ubiquitous wifi-based gesture recognition system". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1472–1480.

[55] Mohamed Abudulaziz Ali Haseeb and Ramviyas Parasuraman. "Wisture: Rnn-based learning of wireless signals for gesture recognition in unmodified smartphones". In: *arXiv preprint arXiv:1707.08569* (2017).

[56] Chen Zhao, Ke-Yu Chen, et al. "SideSwipe: detecting in-air gestures around mobile devices using actual GSM signal". In: *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 2014, pp. 527–534.

[57] Saiwen Wang, Jie Song, et al. "Interacting with soli: Exploring fine-grained dynamic gesture recognition in the radio-frequency spectrum". In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 2016, pp. 851–860.

[58] Qifan Pu, Sidhant Gupta, et al. "Whole-home gesture recognition using wireless signals". In: *Proceedings of the 19th annual international conference on Mobile computing & networking*. 2013, pp. 27–38.

[59] Mohammed Abdulaziz Aide Al-qaness and Fangmin Li. "WiGeR: WiFi-based gesture recognition system". In: *ISPRS International Journal of Geo-Information* 5.6 (2016). Publisher: Multidisciplinary Digital Publishing Institute, p. 92.

[60] Yongsen Ma, Gang Zhou, et al. "Signfi: Sign language recognition using wifi". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2.1 (2018). Publisher: ACM New York, NY, USA, pp. 1–21.

[61] Daniel Halperin, Wenjun Hu, et al. "Tool Release: Gathering 802.11n Traces with Channel State Information". In: *ACM SIGCOMM CCR* 41.1 (Jan. 2011), p. 53.

[62] "Noise in Wireless Communications". In: *Advanced Digital Signal Processing and Noise Reduction*. John Wiley & Sons, Ltd, 2006, pp. 433–448. ISBN: 978-0-470-09496-9. DOI: 10.1002/0470094966.ch17. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470094966.ch17.

[63] Hasmath Farhana Thariq Ahmed, Hafisoh Ahmad, et al. "Higher Order Feature Extraction and Selection for Robust Human Gesture Recognition using CSI of COTS Wi-Fi Devices". In: *Sensors* 19.13 (2019). Publisher: Multidisciplinary Digital Publishing Institute, p. 2959.

[64] Wenfeng He, Kaishun Wu, et al. "Wig: Wifi-based gesture recognition system". In: *2015 24th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2015, pp. 1–7.

[65] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011). Publisher: Acm New York, NY, USA, pp. 1–27.

[66] Pedro Melgarejo, Xinyu Zhang, et al. "Leveraging directional antenna capabilities for fine-grained gesture recognition". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 2014, pp. 541–551.

[67] Xianjia Meng, Lin Feng, et al. "Sentence-Level Sign Language Recognition Using RF signals". In: *2019 6th International Conference on Behavioral, Economic and Socio-Cultural Computing (BESC)*. IEEE, 2019, pp. 1–6.

[68] C. Wang, S. Chen, et al. "Literature review on wireless sensing-Wi-Fi signal-based recognition of human activities". In: *Tsinghua Science and Technology* 23.2 (2018), pp. 203–222.

[69] Hongbo Jiang, Chao Cai, et al. "Smart home based on WiFi sensing: A survey". In: *IEEE Access* 6 (2018). Publisher: IEEE, pp. 13317–13325.

[70] Yongsen Ma, Gang Zhou, et al. "WiFi sensing with channel state information: A survey". In: *ACM Computing Surveys (CSUR)* 52.3 (2019). Publisher: ACM New York, NY, USA, pp. 1–36.

[71] Kazuya Ohara, Takuya Maekawa, et al. "Detecting state changes of indoor everyday objects using Wi-Fi channel state information". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.3 (2017). Publisher: ACM New York, NY, USA, pp. 1–28.

[72] SHI Xingjian, Zhourong Chen, et al. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[73] S. van der Walt, S. C. Colbert, et al. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science Engineering* 13.2 (2011), pp. 22–30.

[74] J. D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science Engineering* 9.3 (2007), pp. 90–95.

[75] Pauli Virtanen, Ralf Gommers, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2.

[76] F. Pedregosa, G. Varoquaux, et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[77] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[78] Martín Abadi, Ashish Agarwal, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[79] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[80] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: http://arxiv.org/abs/1605.02688.

[81] Frank Seide and Amit Agarwal. "CNTK: Microsoft's Open-Source Deep-Learning Toolkit". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. event-place: San Francisco, California, USA. New York, NY, USA: Association for Computing Machinery, 2016,

p. 2135. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2945397. URL: https://doi.org/10.1145/2939672.2945397.

[82] Adam Paszke, Sam Gross, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[83] Ronan Collobert, Samy Bengio, et al. *Torch: A Modular Machine Learning Software Library*. 2002.

[84] Takuya Akiba, Keisuke Fukuda, et al. "ChainerMN: Scalable Distributed Deep Learning Framework". In: *Proceedings of Workshop on ML Systems in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017. URL: http://learningsys.org/nips17/assets/papers/paper_25.pdf.

[85] *Deep Learning Frameworks Comparison – Tensorflow, PyTorch, Keras, MXNet, The Microsoft Cognitive Toolkit, Caffe, Deeplearning4j, Chainer*. Netguru Blog on Machine Learning. Publication Title: Netguru Blog on Machine Learning. URL: https://www.netguru.com/blog/deep-learning-frameworks-comparison (visited on 06/27/2020).

[86] Horace He. "The State of Machine Learning Frameworks in 2019". In: *The Gradient* (2019). URL: https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/.