FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Gesture detector with Leap Motion sensor
Student:	Viet Anh Tran
Supervisor:	Ing. Tomáš Nováček
Study program:	Informatics
Branch / specialization:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	until the end of summer semester 2021/2022

Instructions

Goals of the thesis:
1) Analyze the possibilities of user interaction with the virtual environment with the use of hand and finger movement detection, with the emphasis on the Leap Motion sensor.
2) Create a dataset with at least five static gestures (fist, open fist, pinch, pointing and peace sign).
3) Create a Python library for gesture detection that uses a neural network that detects the gestures from point 2, with the library designed so that the gesture set can be easily expanded with other static and dynamic gestures.
4) Create a simple application to visualize the output of the detector.
5) Describe the MultiLeap library and its algorithms for fusing data from multiple Leap Motion sensors and integrate it into your library. Compare the precision of the gesture detector when more than one Leap Motion sensor is used.

Electronically approved by doc. Ing. Jan Janoušek, Ph.D. on 25 November 2020 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Gesture detector with Leap Motion sensor

Viet Anh Tran

Department of Theoretical Computer Science
Supervisor: Ing. Tomáš Nováček

June 27, 2021

Acknowledgements

First, I would like to thank my supervisor Ing. Tomáš Nováček, for his active support and guidance on and off my studies. I also wish to thank my friends, namely Ája, Nikky, Daniel, David, Matěj, for making the world a bit more colorful and Bc. Matouš Kozák for his never-ending help during my time at the faculty. Finally, to thank my mother for putting up with me my entire life.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 27, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tran, Viet Anh. *Gesture detector with Leap Motion sensor.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Zkoumání způsobů pro ovládání virtuálního prostředí je populárním cílem mnoha výzkumných prací v odvětví interakce člověka s počítačem. Jeden ze způsobů je použití Leap Motion optického senzoru, vyvíjeného specificky pro rozpoznávání pohybu ruky a prstů. Tato bakalářská práce se zaměřuje na využití Leap Motion senzorů k rozpoznávání gest v reálném čase za pomocí neuronové sítě. Využili jsme architekturu dvouvrstvé obousměrné LSTM k natrénování statických i dynamických gest. Neuronová síť byla otestovaná na veřejně dostupném ASL datasetu s výsledkem 89.07% za použití 5-fold cross validace s 200 iteracemi. Architektura byla ve finále natrénovaná využitím našeho vlastního datasetu s 3861 vzorky pro rozpoznávání v reálném čase. Demonstrovali jsme, že náš předtrénovaný model je vhodný pro použití v jiných aplikacích a také jsme diskutovali aktuální stav MultiLeap knihovny, vyvíjené pro detekci ruky pomocí více Leap Motion senzorů najednou. Porovnali jsme výsledky více senzorů použitím MultiLeap knihovny s výsledky naměřené jedním senzorem.

Klíčová slova rozpoznávání gest, dvouvrstvé obousměrné LSTM, MultiLeap, strojové učení, rekurentní neuronová síť, rozpoznávání v reálném čase

Abstract

Exploring ways to control the virtual environment is a popular goal of many human-computer interaction researchers. One of the approaches is using Leap Motion optical sensors, developed specifically to track hand and finger movements. The bachelor thesis focuses on utilizing Leap Motion sensors in real-time gesture recognition using neural networks. We used two layered bidirectional LSTM architecture to train static gestures along with dynamic gestures. The neural network was benchmarked on a publicly available ASL dataset acquiring 89.07% using 5-fold cross-validation on 200 epochs. The architecture was ultimately trained using our dataset of 3861 samples for real-time deployment. We demonstrated that the pre-trained model is sufficient to be integrated into other applications, and we also discussed the current state of the MultiLeap library, developed for hand detection using more than one Leap Motion sensor at once. We compared results of using multiple sensors with MultiLeap with results of using one sensor.

Keywords gesture recognition, two-layered bidirectional LSTM, MultiLeap, machine learning, recurrent neural network, real-time recognition

Contents

List of Figures

List of Tables

Introduction

Mouse and keyboard are considered to be default devices for human-computer interaction nowadays. But with the maturity in technology, namely virtual and extended reality, the computer's need to understand human body language is more and more present. Actions such as rotating or grabbing and moving an object in three-dimensional space with a computer mouse are unintuitive. They require a little understanding of the controls to execute the task. The movement is limited to the two-dimensional space of the mouse. Oppose to performing the desired action by hands in our three-dimensional space as we would in real life.

One of the proposed solutions for this issue is gesture recognition, where the general idea is for computers to have the ability to recognize gestures and perform actions based on them. Therefore, several tracking devices were developed to process an image and yield valuable data for gesture recognition.

Our goal is to utilize these tracking devices, specifically Leap Motion controllers, in combination with artificial neural networks, to creating a simple library with a pre-trained model ready to be used and expanded by other applications. We also want to use the pre-trained model to evaluate the performance of the MultiLeap library base on the number of connected Leap Motion sensors.

The structure of the thesis is as follows:

In Chapter Neural Networks, we introduce neural networks, explain basic terminology and several exemplary network architectures.

In Chapter Gesture Recognition, we briefly explain gesture categories, discover hardware image processing devices, and what are some of the proposed methods in the field of gesture recognition using machine learning techniques.

In Chapter MultiLeap, we explore the MultiLeap library developed for unifying the stream of data from multiple Leap Motion sensors.

In Chapter Implementation, we describe used methods and key implementation points of our work.

In Chapter Experiments, we discuss the performance of our work in a real-time environment, explore several setups using multiple Leap Motion sensors, and testing the capabilities of the MultiLeap library.

In Conclusion, we will evaluate the results of the work and suggest possibilities for future research.

Chapter 1

Neural Networks

An artificial neural network (ANN) is a mathematical model mimicking biological neural networks, namely their ability to learn and correct errors from previous experience [1], [2].

The ANN subject was first introduced by Warren McCulloch and Walter Pitts in "A logical calculus of the ideas immanent in nervous activity" published in 1943 [3]. But it was not until recent years when ANN has gained popularity with still increasing advancements in technology and availability of training data. ANN had become one of the default solutions for complex tasks which were previously thought to be unsolvable by computers [4].

This chapter will briefly explore different types of neural units and their activation functions, along with some exemplary network architectures.

1.1 Artificial Neuron

As previously mentioned, artificial neurons are units mimicking behavior of biological neurons. Meaning, it can receive as well as pass information between themselves.

1.1.1 Perceptron

Perceptron is the simplest class of artificial neurons developed by Frank Rosenblatt in 1958 [5].

Perceptron takes several binary inputs, vector $\vec{x} = (x_1, x_2, ..., x_n)$, and outputs a single binary number. To express the importance of respected input edges, perceptron uses real numbers called *weights*, assigned to each edge, vector $\vec{w} = (w_1, w_2, ..., w_n)$.

A *step function* calculates the perceptron's output. The function output is either 0 or 1 determined by whether its weighted sum $\alpha = \sum_i x_i w_i$ is less

or greater than its *threshold* value, a real number, usually represented as an incoming edge with a negative weight -1 [6].

$$output = \begin{cases} 1, & \text{if } \alpha \geq threshold \\ 0, & \text{if } \alpha < threshold \end{cases} \tag{1.1}$$

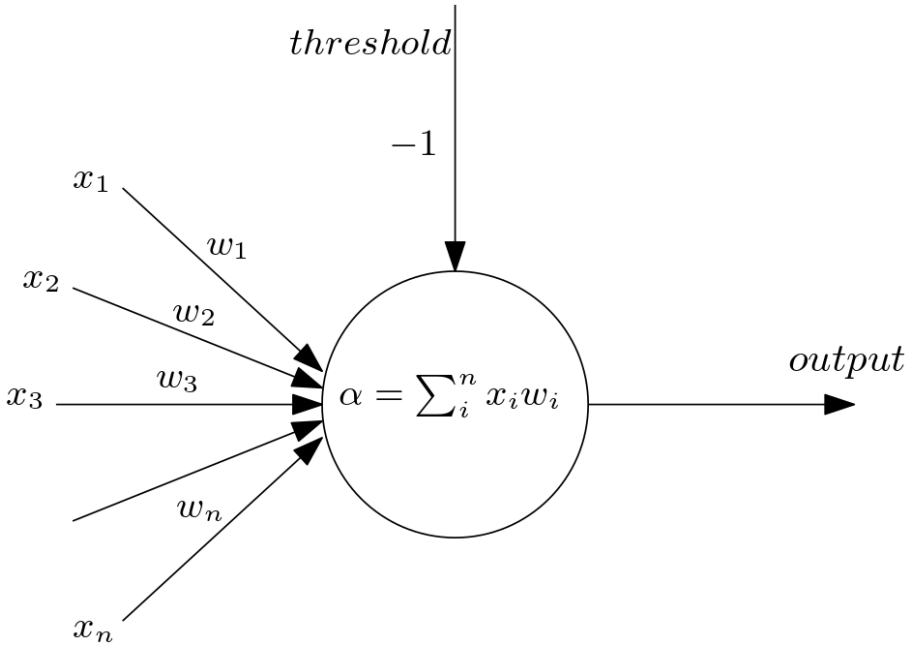


Figure 1.1: Perceptron [6]

1.1.2 Sigmoid Neuron

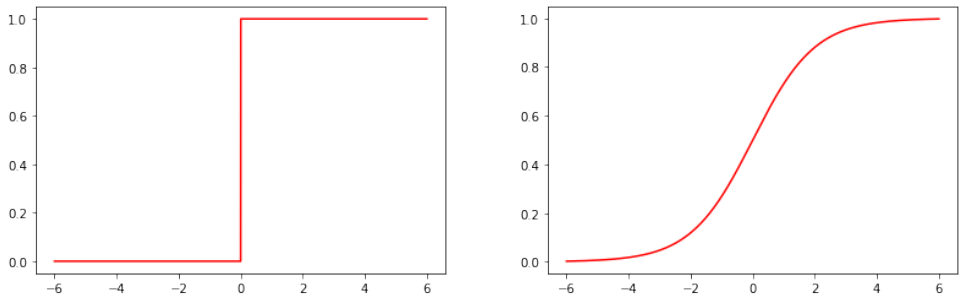
Sigmoid neuron, similarly to perceptron, has inputs $\vec{x}$ and weights. The key difference comes in once we inspect the output value and its calculation. Instead of perceptron's binary output 0 or 1, a sigmoid neuron outputs a real number between 0 and 1 using a *sigmoid function* [7], [8], [6].

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{1.2}$$

As shown in Figure 1.2, the sigmoid function (1.2b) is a smoothed-out version of the step function (1.2a).

1.1.3 Activation Function

An artificial neuron's activation function defines that neuron's output value for given inputs, commonly being $f : \mathbb{R} \to \mathbb{R}$ [9]. A significant trait of many activation functions is their differentiability, which allows them to be used



(a) Step function

(b) Sigmoid function

Figure 1.2: Comparison between step function and sigmoid function

for *Backpropagation*, ANN algorithm for training weights. The activation function needs to have a derivative that does not saturate by heading towards 0 or explode by heading towards inf [6].

For such reasons, the usage of step function or any linear function is unsuitable for ANN.

1.1.3.1 Sigmoid Function

The sigmoid function is commonly used in ANN as an alternative to the step function. A popular choice of the sigmoid function is a *logistic sigmoid*. Its output value is in the range of 0 and 1.

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} = \frac{e^x}{1 + e^x} \tag{1.3}$$

One of the reasons for its popularity is the simplicity of its derivative calculation:

$$\frac{d}{dx}\sigma(\alpha) = \frac{e^x}{(1 + e^x)^2} = \sigma(x)(1 - \sigma(x)) \tag{1.4}$$

On the other hand, one of its disadvantages is the *vanishing gradient*. A problem where for a given very high or very low input values, there would be almost no change in its prediction. Possibly resulting in training complications or performance issues [10], [6].

1.1.3.2 Hyperbolic Tangent

Hyperbolic tangent is similar to logistic sigmoid function with a key difference in its output, ranging between -1 and 1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.5}$$

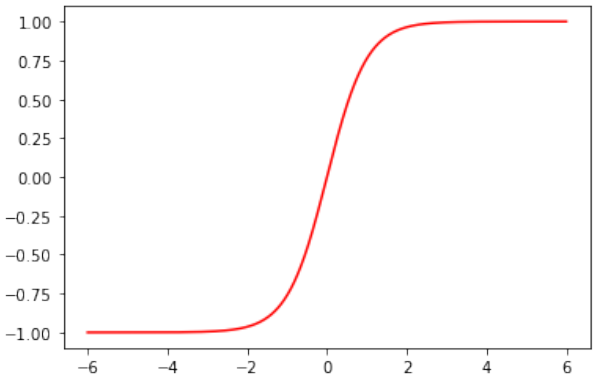


Figure 1.3: Hyperbolic tangent [6]

It shares the sigmoid's simple calculation of its derivative.

$$\frac{d}{dx}\tanh(x) = 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - \tanh^2(x) \tag{1.6}$$

By being only moved and scaled version of the sigmoid function, hyperbolic tangent shares not only sigmoid's advantages but also its disadvantages [9], [6].

1.1.3.3 Rectified Linear Unit

The output of the Rectified Linear Unit (ReLU) is defined as:

$$f(x) = max(0, x) \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{1.7}$$

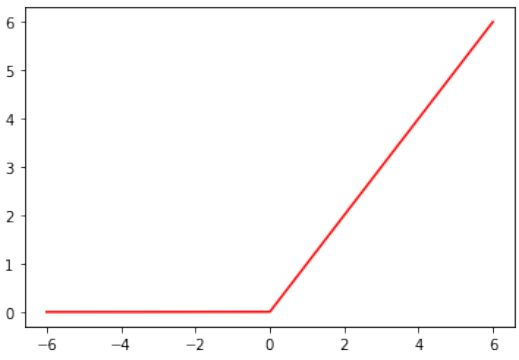


Figure 1.4: Rectified Linear Unit [6]

ReLU's popularity is mainly due to its computational efficiency [10]. Its disadvantages appear when inputs approach zero or to a negative number.

Causing the so-called dying ReLu problem, where the network is unable to learn. There are many variations of ReLu to this date, e.g., Leaky ReLU, Parametric ReLU, ELU, ...

1.1.3.4 Softmax

Softmax separates itself from all the previously mentioned functions by its ability to handle multiple input values in the form of a vector $\vec{x} = (x_1, x_2, ..., x_n)$ and output for each x_i defined as:

$$\sigma(x_i) = \frac{e_i^x}{\sum_{j=1}^{n} e_j^x} \tag{1.8}$$

Because output is being normalized probablty distribution, which ensures $\sum_i \sigma(x_i) = 1$ [11]. It is being used as the last activation function of ANN to normalize the network's output into n probability groups.

1.2 Types of Neural Networks

To this day, there are many types and variations of ANN, each with its structure and use cases. Here we will briefly introduce the most common ones, such as feed-forward networks, convolutional neural networks, or recurrent neural networks.

1.2.1 Feed-forward Networks

Feed-forward network (FFN) was the first ANN to be invented and the simplest form of ANN. Its name comes from the way how information flows through the network. Its data travels in one direction, oriented from the *input layer* to the *output layer*, without cycles. The input layer takes input data, vector $\vec{x}$, producing $\hat{y}$ at the output layer [12].

FFN can contain several hidden layers of various widths but does not have to. By having no back-loops, FFN generally minimizes error, computed by *cost function*, in its prediction by using the *backpropagation* algorithm to update its weight values [13], [11].

1.2.1.1 Cost Function

Cost function $C(\vec{w})$ is used in ANN's training process. It takes all weights and biases of an ANN as its input, in the form of a vector $\vec{w}$ and calculates a single real number expressing ANN's incorrectness [14]. The number is high when the ANN performs poorly and gets lower when the ANN's output gets closer to the correct result. The main goal of training is then to minimize the cost function.

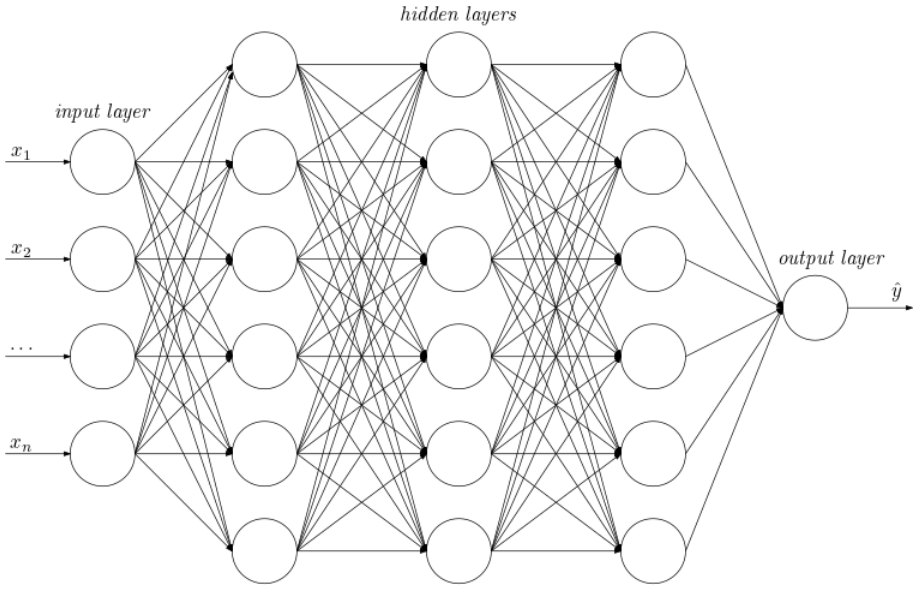


Figure 1.5: Fully connected Feed-forward Neural Network [6]

1.2.1.2 Backpropagation

Backpropagation, short of backward propagation of errors, is a widely used algorithm in training FFN using *gradient descent* to find a local minimum of a cost function and update ANN's weights [15].

A gradient of a function with multiple variables gives us the direction of the steepest gradient ascent, where we should step to rapidly increase the output and find the local maximum. Naturally, its negative will point towards a local minimum.

The usual practice is to divide training samples into small *batches* of size n. We will calculate a gradient descent for each sample in the batch and use their average gradient descent to update the network's weights. The average gradient descent tells us which weights should be adjusted for the ANN to get closer to the correct results [15].

$$-\gamma \nabla C(\vec{w}_i) + \vec{w}_i \rightarrow \vec{w}_{i+1} \tag{1.9}$$

Here, $\vec{w}_i$ is weights of the network at the current state (batch), $\vec{w_{i+1}}$ is updated weights, γ is the learning rate and $-\nabla C(\vec{w}_i)$ is the gradient descent.

1.2.2 Convolutional Neural Networks

Convolutional Neural network's (CNN) main goal is to make a computer recognize images and objects. For such, it is primarily used for image classification or object recognition.

CNN was inspired by the biological processes of the human brain. Its connectivity patterns resemble those of the human visual cortex, but an image is perceived differently by a human brain than by a computer. To a computer, an image is interpreted as an array of numbers. Thus CNN is designed to work with two-dimensional image arrays, although it is possible to work with one-dimensional or three-dimensional arrays too [16].

CNN is a variation of FNN [14]. It usually consists of the input layer followed by multiple hidden layers, typically several *convolutional layers* with standard *pooling layers*, and ending with the output layer.

1.2.2.1 Convolutional Layer

The convolutional layer's objective is to extract key features from the input image by passing a matrix known as a *kernel* over the input image abstracted into a matrix [17].

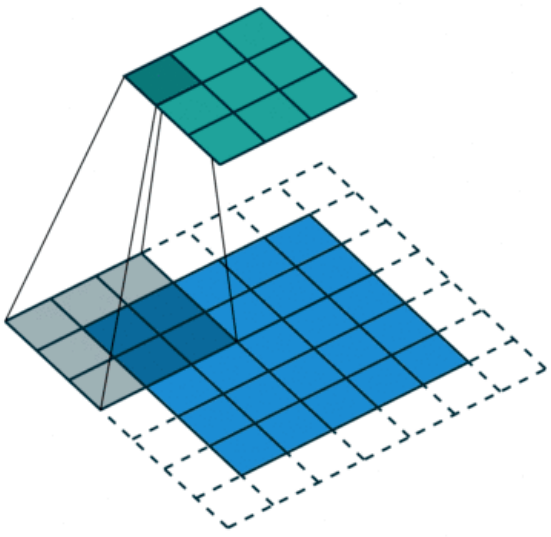


Figure 1.6: Convolution of an 5x5x1 image with 3x3x1 kernel [18]

The convolution result can be of two types depending on their size. One is the convoluted feature, which gets reduced in dimensions compared to the input. It is called *valid padding*. For example, an input image of dimensions 8x8 is reduced to 6x6 after convolution operation, and the other type is where dimensions are either increased or remain the same, which is called *same padding* [18].

1.2.2.2 Pooling Layer

Similar to the previously mentioned convolutional layer, the pooling layer reduces the convolved feature's spatial size to decrease the computational power required for data processing. In addition, the pooling layer is also useful for extracting dominant features, which are rotational and positional invariant, thus effectively training the model [18].

There are two types of pooling: *max pooling* and *average pooling*. Max pooling returns the maximum value from the portion of the image covered by the kernel. It performs as a noise suppressant, discarding the noisy activations altogether and performing de-noising and dimensionality reduction. Where average pooling returns the average of all the values from the same covered portion, performing dimensionality reduction as a noise suppressing mechanism. Hence, it is possible to note that max-pooling performs better [18].

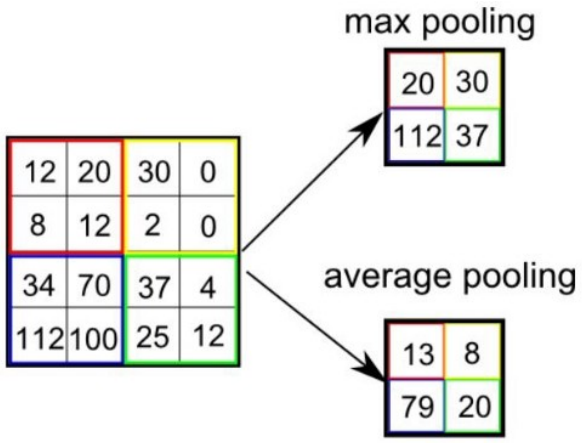


Figure 1.7: Types of pooling [18]

1.2.3 Recurrent Neural Networks

Recurrent Neural Network (RNN) is distinguished by its memory, which takes input sequence with no predetermined size. Its past predictions influence currently generated output. Thus for the same input, RNN could produce different results depending on previous inputs in the sequence [19].

RNNs features make it commonly used in fields such as speech recognition, image captioning, natural language processing, or language translation. Some of the popular being, for example, Siri, Google Translate, or Google Voice search [20].

As previously mentioned, RNN takes into consideration information from previous inputs. Let us look at the idiom "feeling under the weather", where for it to make sense, words have to be in a specific order. RNN needs to

account for each word's positions and use its information to predict the next word in the sequence. Each timestep represents a single word. In our case, the third timestep represents "the". Its hidden state holds information of previous inputs, "feeling" and "under" [20].

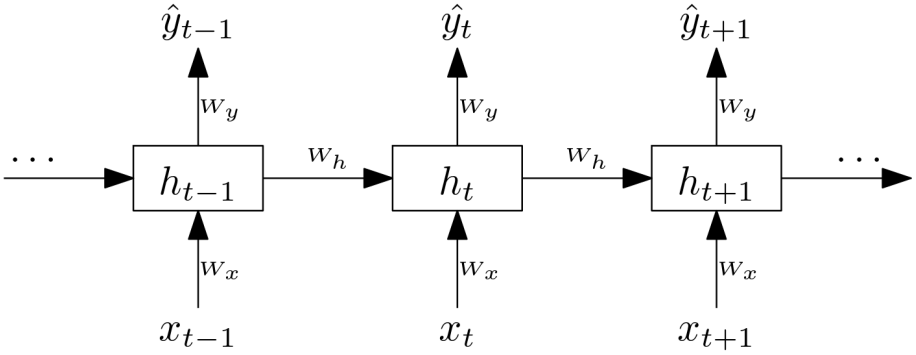


Figure 1.8: Unrolled structure of RNN [6]

Figure 1.8 shows the network for each timestep, i.e., at time t, the input $\vec{x_t}$ goes into the network to produce output $\hat{y}_t$, the next timestep of the input is x_{t+1} with additional input from the previous time step from the hidden state h_t. This way, the neural network looks at the current input and has the context from the previous inputs. With this structure, recurrent units hold the past values, referred to as memory. Making it possible to work with a context in the data [21].

The recurrent unit is calculated as follows:

$$h_t = f(W_x x_t + W_h h_{t-1} + \vec{b_h}) \tag{1.10}$$

f is the activation function, W_x, W_h are weight matrixes, x_t is the input, and $\vec{b_h}$ is the vector of bias parameters. The hiddent stat h_t at time step $t = 0$ is initialized to $(0, 0, ..., 0)$. The output $\hat{y}_t$ is then calculated as:

$$\hat{y}_t = g(W_y h_t + \vec{b_y}) \tag{1.11}$$

g is also an activation function, usually is softmax, to ensure the output is in the desired class range. W_y is the weight matrix, and $\vec{b_y}$ is a vector of biases determined during the learning process.

Training RNNs uses a modified version of the backpropagation algorithm called *backpropagation through time* (BPTT), which works by unrolling the RNN [14], calculating the losses across time steps, then updating the weights with the backpropagation algorithm. More on RNN in [11] by Lipton et al.

1.2.3.1 Bidirection Recurrent Neural Networks

Bidirectional Recurrent Neural Networks (BRNN) allow training the network using all available input information in the past and future of a specific time frame. Oppose to regular RNN, where its hidden state is determined only by the prior states. The idea behind BRNN is to split the hidden state into two. One is responsible for the positive time direction, *forward states*, and the other for the negative time direction, *backward states*.

BRNN's training generally starts with processing forward and backward states before output neurons are passed, *forward pass*. Following with *backward pass*, where output neurons are processed first, and forward and backward states after. Weights are then updated after completing forward pass, and backward pass [22].

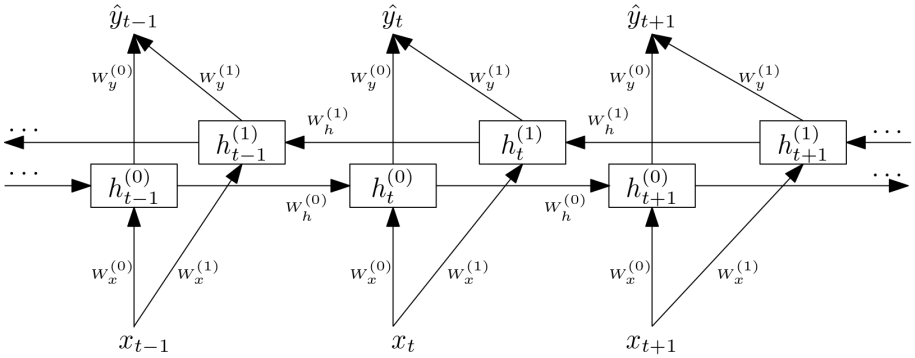


Figure 1.9: Unrolled structure of BRNN [6]

Both hidden states are updated identically as the hidden state in RNN.

$$h_t^{(0)} = f(W_x^{(0)} x_t + W_h^{(0)} h_{t-1} + \vec{b_{h^{(0)}}}) \tag{1.12}$$

$$h_t^{(1)} = f(W_x^{(1)} x_t + W_h^{(1)} h_{t-1} + \vec{b_{h^{(1)}}}) \tag{1.13}$$

The output is then computed in the combination of both hidden states.

$$\hat{y}_t = g(W_y^{(0)} h_t^{(0)} + W_y^{(1)} h_t^{(1)} + \vec{b_y}) \tag{1.14}$$

All the activation functions and parameters remain the same as they were in RNN.

1.2.4 Long Short-Term Memory

Consider a task where we try to predict the last word in "The clouds are in the *sky*". It is fairly obvious the last word is meant to be "*sky*". The gap between the relevant information and the prediction place is small, and RNN can learn to utilize past information and predict the last word. However, if

we consider "I grew up in Spain... I speak fluent *Spanish*", the gap between the relevant information and predicting word can become large. As the gap grows, RNNs are unable to handle the task. Such problem is called *long-term dependencies* [23].

Long Short Term Memory networks (LSTM) are RNN architecture first introduced by Hochreiter S. and Schmidhuber J. [24] with the ability to handle long-term dependencies. Its core idea is to replace RNN's hidden states with so-called **LSTM Cells** and add connections between cells, called *cell states* or c_t. Each LSTM Cell consists of three gates, regulating the input and output of the cell. The calculation in each cell runs as follows:

1. **Forget Gate**: Controls which information should be discarded and which kept. *Sigmoid function* outputs a value between 0 and 1 base on the information from the previous hidden state and from the current input. The value closer to 0 means discard, and closer to 1 means keep.

$$f_t = \sigma(W_{x_f} x_t + W_{h_f} h_{t-1} + \vec{b_f}) \tag{1.15}$$

2. **Input Gate**: Decides which information should be updated. The sigmoid function outputs a value between 0 and 1 base on the previous hidden state and current input state. Closer to 0 means not important, and closer to 1 means important.

$$i_t = \sigma(W_{x_i} x_t + W_{h_i} h_{t-1} + \vec{b_i}) \tag{1.16}$$

The information from the previous hidden state and current input state is also passed into a *tanh* function, getting values between -1 and 1.

$$g_t = \tanh(W_{x_g} x_t + W_{h_g} h_{t-1} + \vec{b_g}) \tag{1.17}$$

The decision on how to update the cell is obtained by multiplying sigmoid output and tanh output. With all the required values available, we can now calculate the *cell state* as follows:

$$c_t = i_t \odot g_t + f_t \odot c_{t-1} \tag{1.18}$$

3. **Output Gate**: Determines what information should the next hidden state contain. The previous hidden state and the current input are passed into a sigmoid function.

$$o_t = \sigma(W_{x_o} x_t + W_{h_o} h_{t-1} + \vec{b_o}) \tag{1.19}$$

Then passing the newly modified cell state into a tanh function and multiplying its output with the sigmoid output, we get the hidden state [25].

$$h_t = o_t \odot tanh(c_t) \tag{1.20}$$

The computation of the output $\hat{y}_t$ proceeds the same way as regular RNN [6].

$$\hat{y}_t = g(W_y h_t + \vec{b_y}) \tag{1.21}$$

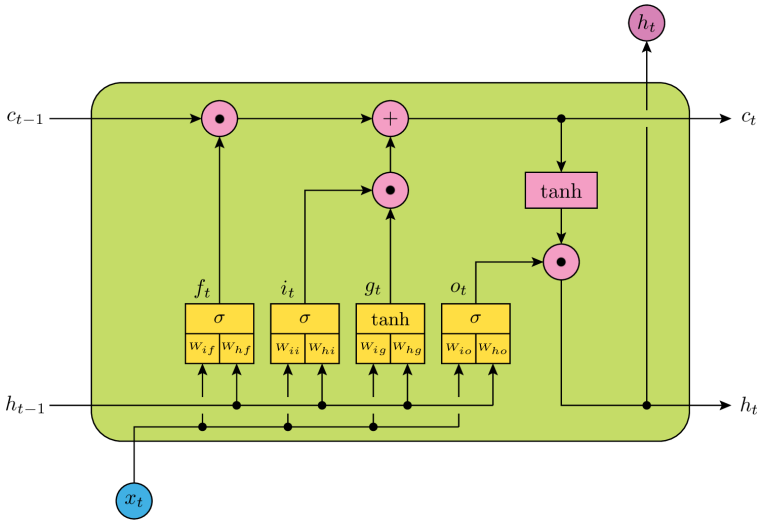


Figure 1.10: LSTM cell [26]

1.2.4.1 Bidirectional Long Short-Term Memory

Similarly, as previously described in BRNN (1.2.3.1), Bidirectional Long Short-Term Memory (BLSTM) has its hidden state split into two, forward states and backward states. Such modification allows the network to gain context from past and future alike. As a result, BLSTM, in comparison with BRNN, handles better the information storage across the timeline with large time gaps from either past or future.

1.2.4.2 Deep Long Short-Term Memory

Deep Long Short-Term Memory (DLSTM), or stacked LSTM, is now considered a stable technique for challenging sequence prediction tasks. It was first introduced by Graves et al. [27], where it was found that the depth of the network has greater importance than the number of memory cells in a given layer. Thus, DLSTM architecture can be described as an LSTM model consisting of multiple LSTM layers.

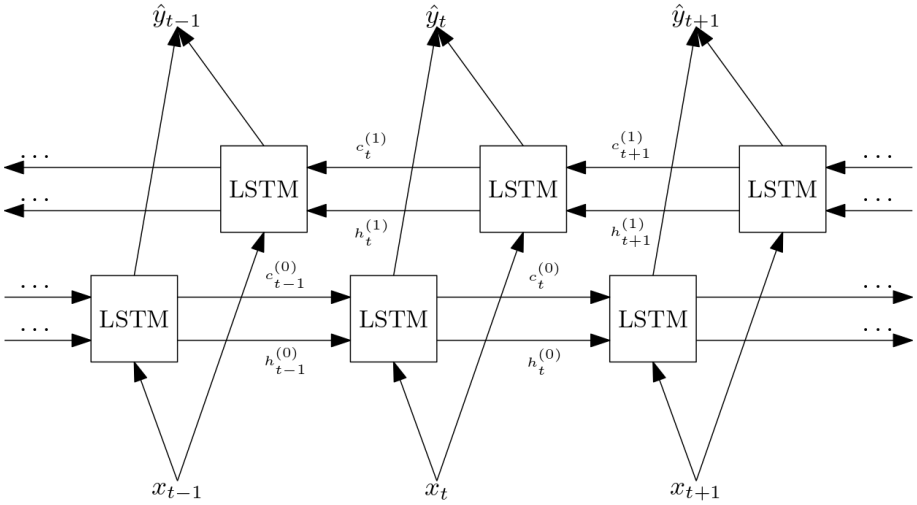


Figure 1.11: Unrolled structure of BLSTM [6]

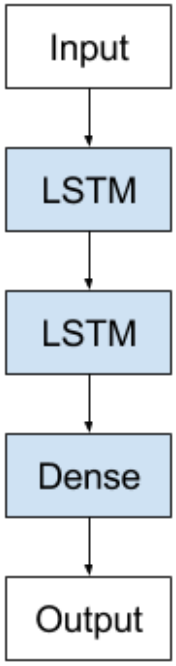


Figure 1.12: Deep Long Short-Term memory architecture [28]

The LSTM layer above outputs a sequence rather than a single value for the LSTM layer below [28].

Chapter 2

Gesture Recognition

2.1 Gesture Categories

Gestures are categorized into *static gestures* and *dynamic gestures*. A group of static gestures consists of fixed gestures, where they are not relative to time. A group of dynamic gestures, on the other hand, are time-varying. These classes can be further subdivided into a set of gestures distinct by their purpose.

- **Deictic gestures** involve pointing to establish the identity or spatial location of an object within the context of the application domain [29].
- **Manipulative gestures** mimic manipulation of a physical object, such as scaling, moving, or rotating.
- **Gesticulation** is commonly used along with the language group. These hand gestures are difficult to analyze.
- **Language group of hand gestures** form a grammatical structure for conversational style interfaces.
- **Semaphoric hand gestures** also may be referred to as communicative gestures, are a group of hand gestures serving as a set of symbols/commands used to interact with machines. The group consists of static hand gestures as well as dynamic hand gestures.

2.2 Tracking devices

Hand and body gesture recognition had followed a conventional scheme of extracting key features via one or multiple preprocessing sensors and applying machine learning techniques on them [30]. The field of gesture recognition gave birth to several image processing devices yielding useful data. We will

only cover optical devices, but there are also controllers in the form of a stick with buttons, like HTC Vive, or others in the form of gloves.

2.2.1 Microsoft Kinect

One of the tracking devices is Microsoft Kinect, a device first released in 2010. Originally developed for gaming but eventually finding more success in academic and commercial applications, such as robotics, medicine, and health care. Microsoft discontinued production of its Xbox version in 2018 and released Azure Kinect in March 2020, incorporating Microsoft Azure cloud computing functionalities.

Figure 2.1: Azure Kinect [31]

Azure Kinect contains a depth sensor, spatial microphone array with a video camera, and orientation sensor as a small all-in-one device with multiple modes, options, and software development kits [32].

With all that said, the primary purpose of the Kinect device overall is to interpret whole-body movement. For such, it lacks in required accuracy for hand gesture recognition, thus making it insufficient for our uses.

2.2.2 Leap Motion Controller

Another option would be using a Leap Motion Controller (LMC), developed specifically to track hand movements and extract its features, such as positions of fingers, hand rotation, and others.

LMC consists of two monochromatic IR cameras and three IR LEDs (emitters).

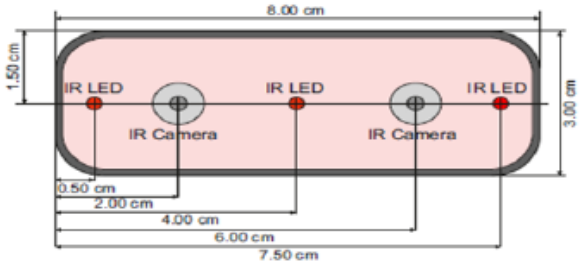


Figure 2.2: Schematic View of Leap Motion Controller [33]

The LMC's current API, Leap Motion Service, yields positions of extracted hand features. All the positional data about the hand and its features are represented in the coordinate system relative to the LMC's center point, positioned at the top of the controller [33]. The x- and z-axes lie in the camera sensors plane, with the x-axis running along the camera baseline. The y-axis is vertical, with positive values increasing upwards (in contrast to the downward orientation of most computer graphics coordinate systems). The z-axis has positive values increasing toward the user [34].

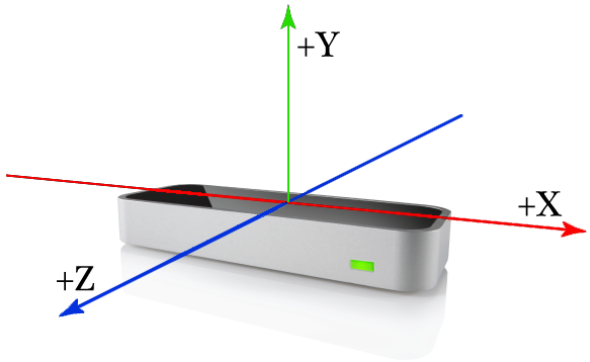


Figure 2.3: Leap Motion Controller Axes [34]

2.2.3 Ultraleap Stereo IR 170

Ultraleap Stereo IR 170, formerly known as the Leap Motion Rigel, is the successor to the Leap Motion controller.

The Stereo IR inherits Leap Motions key features but improves with a wider 170-degree field of view, more powerful LED illuminators providing more extended tracking range, and a higher framerate when used with USB 3.0. The Stereo IR also shares with original LMC its API [35], [36].

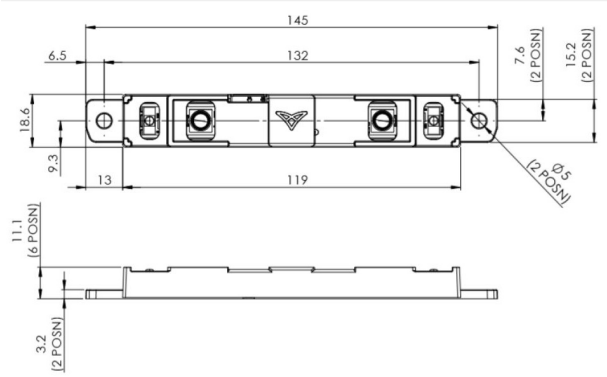


Figure 2.4: Schematic View of Ultraleap Stereo IR 170 [35]

Unfortunately, Leap Motion Controller, as well as Ultraleap Stereo IR 170, has no official library for gesture recognition, limiting developers from utilizing the controller for its key features. Leap Motion provided tracking software built for virtual reality, used to have a gesture detector with its 3.0 version, but the detector is absent with the release of more accurate version 4.0.

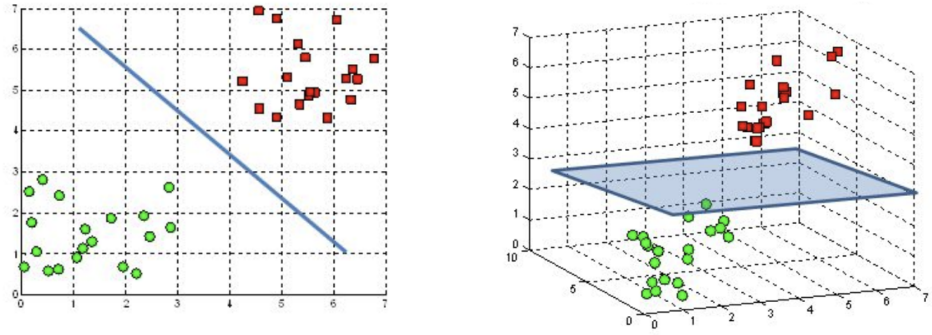
2.3 Gesture Recognition Methods

Gestures group classification should be taken into account when choosing appropriate methods due to their time-varying properties. As previously mentioned, gestures are classified into static and dynamic groups.

2.3.1 Static Gesture Recognition

One of the commonly used methods for static gesture recognition is *Support Vector Machine* (SVM), an algorithm used for both regression and classification tasks. But overall, it is widely used in classifications. SVM's goal is to find a *hyperplane* in N-dimension space, N being the number of features, that distinctively classifies data points [37]. *Hyperplanes* are decision boundaries between data points and *hyperplane* with maximal separation, *margin*, between classes is called *optimal hyperplane* [38].

Chen and Tseng [39] presented an SVM solution for multi-angle hand gesture recognition for rock paper scissors using images from a web camera. The training dataset consisted of 420 images and a testing set of 120 images. Datasets were collected from 5 different people for the right hand only and achieving 95% accuracy. The classifier still managed to recognize left-hand gestures with 90% accuracy.



(a) Hyperplane in $\mathbb{R}^2$ (b) Hyperplane in $\mathbb{R}^3$

Figure 2.5: Hyperplane examples seperating classes in different dimesions [40]

Domino et al. [41] utilized SVM with Microsoft Kinect sensors, extracting hand features, fingertips, and center of the hand, from the depth map and feeding the data into SVM. As a result, achieving 99.5% recognition rate on the dataset provided by Ren et al. [42]. The dataset consists of 10 different gestures performed by ten different people repeatedly, each ten times, a total of 1000 different depth maps.

Mapari and Kharat [43] on the other hand, proposed a method to recognize American Sign Language (ASL) with an *Feed-forward network* using *Multilayer Perceptron (MLP)*, extracting data from LMC and computing 48 features (18 positional values, 15 distance values, and 15 angle values) for 4672 collected signs (146 users for 32 signs). The average classification accuracy is 90%.

2.3.2 Dynamic Gesture Recognition

Katia et al. [44] proposed a method classifying dynamic gestures acquired through LMC with a CNN, adopting a modified version of ResNet-50 architecture, a 50 layers deep CNN, removing the last fully connected layer, and adding a new layer with as many neurons as the considered collection of gesture classes. The acquired gesture information is converted into hand joints color images. The variation of hand joint positions during the gesture is projected on a plane, and temporal information is represented with the color intensity of the projected points. The trained model achieved 91% classification accuracy on the LMDHG dataset [45].

Ameur et al. [46] presented a solution using an SVM classifier used with LMC acquired data, (X, Y, Z) coordinates of fingertips and palm center. The experimental results show an accuracy of 81% on a dataset containing 11 actions performed by ten different subjects, having in total 550 samples.

Yang L., Chen J., and Zhu W. [47] used two-layer Bidirectional RNN in combination with an LMC to classify dynamic hand gestures represented by sets of feature vectors (fingertip distance, angle, height, the angle of adjacent fingertips, and the coordinates of the palm). The proposed method has been tested on modified American Sign Language (ASL) datasets with 360 samples and the Handicraft-Gesture dataset with 480 samples, both containing only dynamic gestures and achieving 90%, 92% accuracy, respectively. The LMC was used only for data acquisition. The architecture was not further tested in a real-time environment, and also the performance on static gestures is unclear since both benchmarked datasets were stripped of any static gesture. [47]

2.3.3 Proposed LSTM solution

Many of the proposed methods focus either on static gesture recognition or dynamic gesture recognition, but very few of them are actually utilized for both types simultaneously.

Avola D., Bernardi M. et al. proposed a method in [30] using LSTM, specifically Deep LSTM (DLSTM), and LMC to recognize sign language and semaphoric hand gestures. It uses a hand skeleton extracted by an LMC and considers angles formed by a specific subset of hand joints. The presented method reached 96% accuracy in its predictions.

The LMC was used only to collect data for training. The method was not tested in a real-time environment, and it is yet to be explored whether it will.

Consider each hand gesture to be represtend as set $X = \{x_0, x1, ..., x_{T-1}\}$ of feature vectors, in predetermined interval Θ size T, T is the number of time instances, in which features are extraxted by LMC. DLSTM is applied to obtain series of output probability vectors $Y = \{y_0, y1, ..., y_{T-1}\}$. At last the gesture classification is performed by a *softmax* layer using $n = |C|$, where C is the set of considered hand gestures [30].

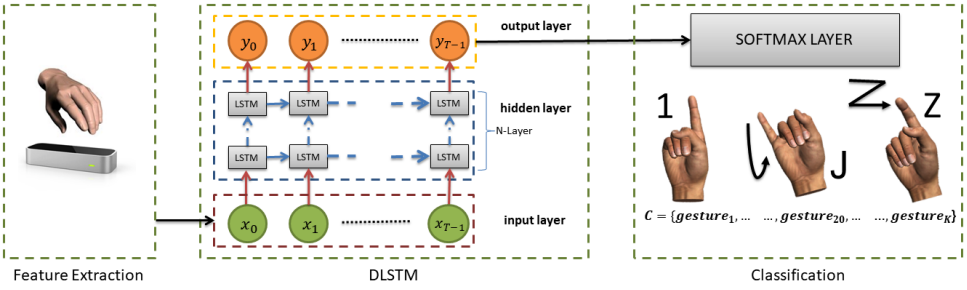


Figure 2.6: Logical structure of the proposed method [30]

2.3.3.1 Feature Extraction

A hand gesture can be considered to be composed of different poses, where particular angles characterize each pose. Each feature vector $x_t \in X$ consists mainly of internal angles, finger segments, palm position, and fingertip positions.

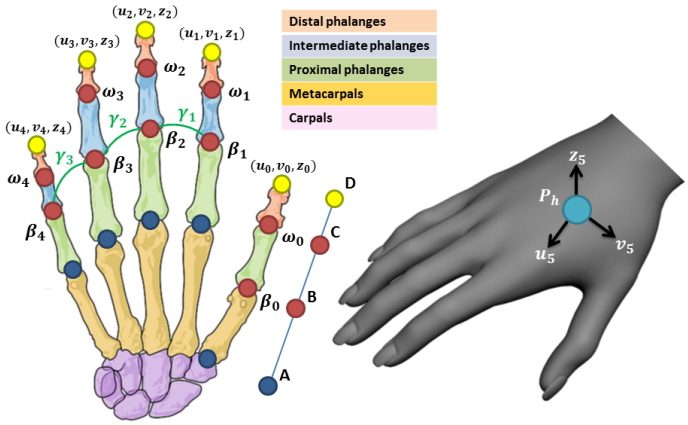


Figure 2.7: Internal angles of hand joints [30]

As seen in Figure 2.7, each finger can be represented as set of segments:

- $\overline{AB}$, proximal phalax, or metacarpal in case of thumb
- $\overline{BC}$, intermediate phalanx, or proximal phalanx in case of thumb
- $\overline{CD}$, distal phalanx

These set of segments are then used to calculate internal angles of the considered finger:

- internal angles $\omega_1, \omega_2, \omega_3, \omega_4$ between distal phalanges and intermediate phalanges. Internal angle ω_0 of the thumb is calculated between distal phalanx and proximal phalanx.

$$\omega_{j\in\{0,\dots,4\}} = \frac{\overline{BC} \cdot \overline{CD}}{|\overline{BC}| \cdot |\overline{CD}|} \tag{2.1}$$

- internal angles $\beta_1, \beta_2, \beta_3, \beta_4$ between intermediate phalanges and proximal phalanges. Internal angle β_0 of the thumb is calculated between proximal phalanx and metacarpal.

$$\beta_{j\in\{0,\dots,4\}} = \frac{\overline{AB} \cdot \overline{BC}}{|\overline{AB}| \cdot |\overline{BC}|} \tag{2.2}$$

- intra-finger angles $\gamma_1, \gamma_2, \gamma_3$ are angles between two neighboring fingers, where considered fingers are: the pointer finger between middle finger, the middle finger and the ring finger, and the ring finger with a pinky finger. The intfra-finger angles are used to handle special static gestures, for example, an open palm and a pop culture "Spock" greeting.

3D displacements of palm and fingertip positions help classify dynamic hand gestures, where the movement is performed in 3D space.

- palm central point coordinates $P_h = (u_5, v_5, z_5)$ help to track the hand transition in the 3D space.
- finger tip positions u_l, v_l, z_l, $l \in 0, ..., 4$ help to track the hand rotation in 3D space.

All above features form the input vector x_t passed to DLSTM at time t.

$$x_t = \{\omega_0, ..., \omega_4, \beta_0, ..., \beta_4, u_0, v_0, z_0, ..., u_5, v_5, z_5, \gamma_1, \gamma_2, \gamma_3\} \tag{2.3}$$

2.3.3.2 Optimal Number of Stacked LSTMs

Several tests were performed to find the optimal number of stacked LSTMs. The results showed that having 4 LSTM layers proved to achieve the best accuracy by using 800 *epochs*, the number of times the learning algorithm goes through the complete training dataset. Although it was possible to get the same results with 5 or 6 stacked LSTM layers, only due to using 1600 and 1800 epochs, thus increasing the training time [30].

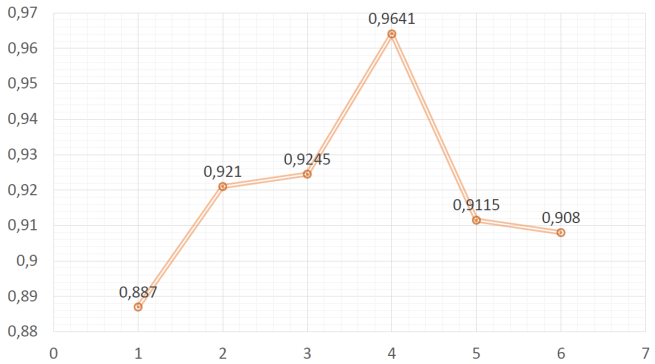


Figure 2.8: Model accuracy by using 800 epochs [30]

The *learning rate* was set to 0.0001 after large empirical tests. The learning rate determines how much the newly acquired information about the weights will influence their updating. If the learning rate is too low, it will require more time to converge towards the local minimum, while if the rate is too large, it may overstep the local minimum [30].

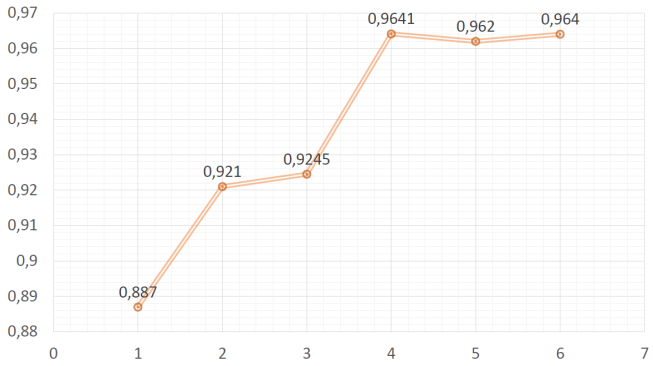


Figure 2.9: Model accuracy by using 1600 epochs for 5 LSTM layers and 1800 epochs for 6 LSTM layers [30]

2.3.3.3 Sampling Process

One gesture can be performed differently by each person, and all collected frame sequences must be composed of the same number of T samples. The proposed solution would collect data only in most significant T time instances, $t \in \Theta$ is considered significant if the joint angle and the central palm point coordinate P_h differs substantially between t and $t+1$.

To explain more specifically, let $f_{\omega_i}(t)$, $f_{\beta_i}(t)$, $f_{\gamma_j}(t)$ be functions representing values of ω_i, β_i, γ_j angles at time t, where $0 \leq i \leq 4$ and $1 \leq j \leq 3$. Coordinates of P_h may be ϕ and coordinates at time t may be represented as $f_\phi(t)$. Then the Savitzky-Golay filter [48] is applied on each of the named functions, $f_g(t)$, $g \in G = \{\omega_i, \beta_i, \gamma_j, \phi\}$. Svaitzky-Golay is a digital filter used to smooth a set of digital data in order to increase the signal-to-noise ratio without distorting the signal itself. Local extremes of each $f_g(t)$ are to be indentified as significant time variations and all time instances t, associated with at least one of these local maximum and minimum of feature g, form a new set Θ^*, representing candidates of possible important time instances to be sampled.

Depending on the cardinality of the newly acquired set Θ^*, the following cases must be considered:

- $|\Theta^*| < T$, the remaining samples $(|\Theta^*| - T)$ are picked randomly from the original set Θ
- $|\Theta^*| > T$, only some of significant time instances for each g feature are picked to be sampled. Let $\Theta_g \subseteq \Theta^*$ be a set of significant time instances for feature g. The number of instances T_g to be sampled is chosen according to the ratio $|\Theta_g| : |\Theta^*| = T_g : T$, where the sum $\sum_{g \in G} T_g = T$ must be preserved [30].

Chapter 3

MultiLeap

In 2018, developers from UltraLeap had released an experimental build for Leap Motion tracking software, which provided data from all connected LMCs at once. Despite having this feature, the provided tracking information for the same hand was different from each sensor due to different points of origin. This problem was solved by the MultiLeap library created by Tomáš Nováček et al. in [34], which merges the information from all sensors and returns a unified stream of data. The library works with the same data structures as Leap Motion's API.

3.1 Alignment of the tracking data

To align tracking data, we must first determine the position of LMCs to place them in the virtual world. This can be achieved by computing the sensor's positions, and rotations in relation to other LMCs [34].

3.1.1 Data sampling

The MultiLeap library allows a user to sample data using a semi-automatic sampling process. Each sample consists of 20 points from the hand — the points represent the center of each finger joint.

The sampling is enabled manually, but data are sampled automatically per every Leap Motion frame, approximately 90 times per second. The general idea of automatic sampling is to calibrate sensors using data from already calibrated devices.

First, one sensor is marked as calibrated. The first marked sensor is either the first connected sensor or one selected by the user. Uncalibrated sensors start acquiring samples if the presented hand is in their field of view and at the same time in the field of view of any calibrated sensor. The pair of samples consists of the uncalibrated sensor's original data and fused data from all calibrated sensors, to which is the hand visible. Once the sensor collects

enough samples, it begins to compute the optimal translation and rotation of the device. The sensor is then marked as calibrated. The process is repeated until all sensors are calibrated [34].

Hands will then align automatically, but it is up to the user, performing the calibration, to cover enough space of the tracking area. Therefore, it is best to have diverse samples for more accurate alignment [34].

Considering the tracking data, where the hand is completely still, it will not have the necessary diversity in its samples. The deviation between collected tracking data is too insignificant. If we were to move the hand across the tracking area, having it rotated in various ways in various positions, the deviation of rotations and positions will be more evident, and the calculation of the alignment more precise [34].

Another option for calibration is a fully manual setting, allowing a user to set the position and rotation of sensors. Values need to be calculated accurately for the alignment to have any use. The main advantage of this approach is having the possibility of tracking different parts of the tracked space with the sensors, for example, LMCs being back to each other [34].

The combined approach is also possible. First, making a rough calibration manually and eventually improved by the semi-automatic.

3.1.2 Kabsch algorithm

Kabsch algorithm [49] also known as Procrustes superimposition, was used to determine the rotation of sensors by calculating optimal rotation matrix minimizing the root mean squared deviation between two paired sets of points. The first set of points consists of merged tracking information from calibrated sensors. The second set of points consists of the tracking information from any other sensor. [34]

The goal of the Kabsch algorithm is to compute the optimal translation rotation of P onto Q, where P and Q are sets of pair points that minimize the distance between the two sets. Both P and Q are represented as $N \times 3$ matrix. Each row consists of coordinates of every point [34].

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{pmatrix} \tag{3.1}$$

Coordinates of the first point are in the first row, the second point in the second row, and the Nth point in the Nth row.

The algorithm has two main steps, computing the optimal translation and computation of the optimal matrix.

The optimal translation can be easily found by being the offset between the averages of two sets of points. As for optimal rotation, we must first calculate

the mean center of the points by subtracting the coordinates of the respective centroid from the point coordinates. The centroid C_P for P is computed as follows:

$$C_P = \frac{\sum_{i=1}^{N} P_i}{N} \tag{3.2}$$

The mean-center calculation of all points in P:

$$P_i = P_i - C_P \tag{3.3}$$

Then, the 3×3 cross-variance matrix between the points must be calculated as follows in matrix notation:

$$H = P^T Q \tag{3.4}$$

At last, we will extract the rotation from the covariance matrix using polar decomposition. The extraction can be done in more iterations, resulting in more accurate rotation calculation but requiring higher computation time in return.

Algorithm 1 Kabsch algorithm

Input:

- **sensors**: List of N collections of samples for N sensors
- **iterations**: The number of iterations of the Kabsch algorithm

1: **for** $sensor = 2, \ldots, N$ **do**
2: optimalTranslation = getAverage(referenceMatrix) - getAverage(sensorMatrix);
3: covarianceMatrix = transpose(sensorMatrix) - referenceMatrix
4: **for** $i = 1, \ldots, iterations$ **do**
5: extractRotation(covarianceMatrix)
6: **end for**
7: Translation and rotation of the sensor in the Unity scene
8: **end for**

3.2 Data fusion

If multiple sensors detect the hand, the fusion algorithm is used. In most cases, not all sensors detect the hands properly. One of the yield information provided by MultiLeap library is a *confidence*, a float value ranging from 0.3 to 1, which denotes the confidence level of the tracking data corresponding Leap Motion frame. The purpose of confidence level is to give more weight to tracking data from the sensor, which detects the hand better, making the

tracking more accurate even if two out of three sensors would send inaccurate tracking data. The confidence level is of value 0.3 when the palm normal is in a 90° and 1 when in 0° or 180° angle to Y-axis. MultiLeap does not use the confidence of 0 because even with the occlusion of fingers and hand, the tracking data still carries some information about the hand. After few experiments, the value 0.3 was determined to be the most suitable confidence level for minimal tracking data when the palm normal is in 90° angle to the Y-axis of the sensor. The mentioned approach resulted in following equation for *confidence* computation:

$$confidence = (0.283699 \times angle^2) - (0.891268 \times angle) + 1 \tag{3.5}$$

The function transfers the angle, in radians, between the palm normal and the sensor's normal to the corresponding confidence level [34].

The confidence level is used to give weight to data from the sensor, which detects the hand better, making the tracking more precise despite faulty data coming from other sensors.

Chapter 4

Implementation

As briefly mentioned in the Introduction chapter, our goal is to utilize Leap Motion controllers combined with the pre-trained ANN model. The following chapter will explore datasets used for our training and the obstacles that came along with them. Then we will discuss the model training itself and its results. At last, we will deploy the trained model for real-time recognition in a C++ environment.

4.1 Dataset Description

Among many publicly available datasets for gesture recognition are only a few containing necessary skeletal information similar to those yield by Leap Motion controllers. We have selected ASL Dataset, and SHREC 2017 Dataset created in conjunction with [30] and [50] respectively, often used as benchmark measurement for trained model accuracy.

4.1.1 SHREC 2017 Dataset

The SHREC dataset contains sequences of 14 dynamic hand gestures. Each gesture was performed between 1 and 10 times by 28 participants in two ways, using one finger and the whole hand. All participants were right-handed. The length of sample gestures varies between 20 to 170 frames. The variation of frames makes it too inconsistent for our usage in real-time deployment and, as such, deemed unsuitable. Some samples can be too short and must be thrown away. Some can be too long, and if we were to shorten it, we might lose the gesture's key features.

4.1.2 ASL Dataset

ASL Dataset consists of 30 hand gestures - 18 static gestures and 12 dynamic gestures. Gestures were collected from 20 different people. 13 were used to

form the training set, while the remaining 7 formed a test set. Each person performed 30 hand gestures twice, once for each hand, and each gesture is composed of fixed 200 frames as oppose to frame varying SHREC dataset [30].

After further inspection of the ASL dataset, we have discovered possible mislabeling of features. Specifically, taking a look at internal angles of gesture for number 1, we can see that 1 requires the ring finger to straighten out instead of the index finger. The same can be said about the gesture of number 2, where it appears to have the ring finger and middle finger straight out instead of the index finger and middle finger. It is unclear whether there is other mislabeling among the features. The mislabeling in itself is not an obstacle for training because the features are independent of each other, and the ANN can still learn on them, but the issue will arise in real-time classification, where raw data must be preprocessed identically as the training data. We decided not to use ASL Dataset for our purposes but only to benchmark model architecture.

4.1.3 Data sampling

By not using ASL Dataset, we have lost a set of static gestures. Also, we want to have the ability to provide the training with our own sets of gestures and not to be bound only to those publicly available. For such purposes, we had created a simple interactive data sampler in the form of a console application.

The sampler saves each sample in .txt format, one line per timestep T, frame yield by LMC. Each line contains a set of features. Features were selected and computed as previously described in section 2.3.3.1. The order of features in a line x_t, at time t is as follows:

$$x_t = \{\omega_0, ..., \omega_4, \beta_0, ..., \beta_4, u_0, v_0, z_0, ..., u_5, v_5, z_5, \gamma_1, \gamma_2, \gamma_3\} \tag{4.1}$$

All samples contain the same number of timesteps, specified at the beginning by the user or using the default value of $T = 60$. The number of timesteps should be further analyzed in order to find the optimal value. The value of timestep mostly affects the delay rate between the presented gesture and its prediction in a real-time environment, higher creates greater delay. Also, if the value is too high, the dynamic gesture may have a minimal role in the sample, and we will not get desired behavior from our ANN. If the number is too low, the dynamic gesture may not be captured completely, and the rate of performed predictions increases, creating greater demand on hardware.

The recording is initiated by key command, but the data collection does not start until the user's hand is in LMC's field of view. Data collection stops once the set of collected frames Θ matches T or if the hand falls out of LMC's view. Features of missing timesteps are then set to zeroes. The sampling can be subdivided into 3 types:

1. **Single recording** records and saves a single sample. The next recording must be initiated by the user.

2. **Open recording** records and saves samples continuously. We recommend using the method only for static gestures. It is best to have full control over recording a dynamic gesture, its beginning, and its end, along with its most significant sequence.

3. **Recording significant frames** records and saves a single sample. The next recording must be initiated by the user. The number of collected frames Θ^* is greater than the required number of timesteps $|\Theta^*| > T$. The last frame is excluded if $|\Theta^*|$ is not even. We will then calculate a *significance* between x_t and x_{t+1}. The *significance* of an interval is calculated as average euclidean distances of palm and finger tip positions P between x_t and x_{t+1}.

$$s_{(t,t+1)} = d(P_t, P_{t+1}) \tag{4.2}$$

$$S = \{s_{(0,1)}, ..., s_{(t,t+1)}\} \tag{4.3}$$

   Frames are then selected into Θ by most significant to least significant till $|\Theta| = T$. The following cases must be considered:

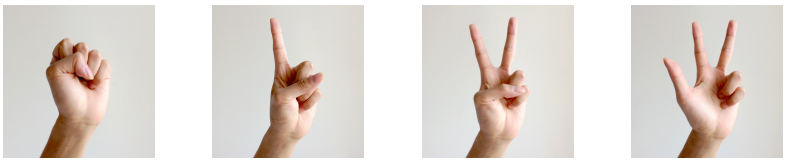
   - $|\Theta| + 2 \leq T \wedge x_t$ and $x_{t+1} \notin \Theta$, both frames x_t and x_{t+1} will be added to Θ.
   - $|\Theta| + 1 = T \wedge x_t$ and $x_{t+1} \notin \Theta$, both x_t and x_{t+1} are possible candidates for Θ but only one can be added due to the size $|\Theta|$ which would reach the limit after addition of one. In order to decide which to pick we will compare the significance $s_{(t-1,t)}$ and $s_{(t+1,t+2)}$, and pick greater of the two.
     It is possible that x_t is the first frame of Θ^*, in which case we will pick x_{t+1} to be included in Θ. On the other hand, if x_{t+1} is the last frame of Θ^*, we will pick x_t.
   - $x_t \notin \Theta \wedge x_{t+1} \in \Theta$, frame x_t is picked
   - $x_t \in \Theta \wedge x_{t+1} \notin \Theta$, frame x_{t+1} is picked

   The method is recommended for sampling dynamic gestures. We attempt to capture the most important part of a dynamic gesture, its movement.

   The most challenging part of creating a dataset is when we do the sampling itself. We want to keep in mind the variety of samples, meaning angles, positions, and additionally regarding dynamic gestures, various

speeds, and starting positions. Also, the factor of overlapping characteristics of gestures must be taken into consideration. For example, an open hand and open-handed swipe right may share some similar sequence of frames. Despite introducing sampling specifically for dynamic gestures, it can still be challenging to create a dataset, which is diverse enough to hold key properties of the gesture and will enforce the model to learn its characteristics.

We created a dataset, which consists of 7 static gestures (fist, number 1-pointing, number 2-peace sign, number 3, number 4, number 5-open fist, pinch) and 2 dynamic gestures (swipe right, swipe left), each of average 429 samples performed by both hands, a total of 3861 samples.



(a) 0-fist (b) 1-pointing (c) 2-peace sign (d) 3

(e) 4 (f) 5-open hand (g) 6-pinch

Figure 4.1: Set of static gestures

Swipe can be performed two different ways, one possibility is using whole hand and other is using only the wrist.

(a) 7-swipe right (using the whole hand) (b) 7-swipe right (using the wrist) (c) 8-swipe left (using the whole hand) (d) 8-swipe left (using the wrist)

Figure 4.2: Set of dynamic gestures

All gestures were sampled by one LMC sensor, which meets its limitation and is unable to create correct hand joint alignments at some specific

angles, as an example *number 1* pointing towards the sensor. Due to this fact, some gestures can be lacking in samples with more intricate angles. We do not recommend using our dataset for any benchmarking as the gesture set is not complex enough for any model performance evaluation. Another factor is that it was sampled only by one user, which should be further expanded in future works. Its sole purpose is to have a working gesture set for real-time recognition.

4.2 Model Training

We selected Python to be our primary language for training the ANN mode along with the web-based interactive development environment Jupyter Notebook. One of the main reasons to pick Python over other available languages was its wide range of libraries and scientific packages supporting machine learning tasks. Most importantly, Keras, a high-level deep learning API integrated with TensorFlow, enables the user to create and train model structures in very few steps.

ASL Dataset was used to benchmark the model and further analyzing optimal parameters. For the purpose of real-time recognition, we trained the model on our original dataset described at the end of section 4.1.3.

ASL dataset was split into 80% for the training set and 20% for the testing set, where 10% of the training set was used for validation. Each feature was then normalized via min-max scaler formula:

$$x' = \frac{x - Min(X)}{Max(X) - Min(X)} \quad x \in X \tag{4.4}$$

4.2.1 DLSTM architecture

At first, we followed the proposed architecture of 4 layers stacked LSTM by Avola D., Bernardi M. et al. [30]. We trained the model using 800 epochs and 0.0001 learning rate, which were proved to be optimal hyperparameters as described in section 2.3.3.2.

Benchmarking the model on ASL Dataset resulted in similar accuracies as in [30]. Our original dataset had also achieved high results.

Despite achieving high accuracies on the ASL dataset and our original dataset, the model itself did not perform well in a real-time environment. More specifically, the 4 layered LSTM architecture struggled with dynamic gestures. The model did not learn the gesture in relation to the movement but rather on its most occurring position in the recorded sequence, which in the case of swipe right was the palm's final position. The model successfully classified test samples because all gestures swipe right contained some frames where the palm position was on the right. Once we presented the model in a real-time environment with an open palm on the right, without the swipe

movement, it classifies such gesture as swipe right, which is an undesired behavior.

It is unclear whether the model would perform better with more stacked LSTM layers or not. Another possibility would be having an insufficient dataset, but later we were able to utilize the same dataset on different architecture in a real-time environment. Hence we concluded that the DLSTM architecture was not suitable for our purposes.

4.2.2 Two-Layered Bidirectional LSTM architecture

After an unsuccessful attempt to utilize DLSTM, we turned over to the two-layered bidirectional LSTM architecture proposed in [47]. The proposed architecture was meant and trained on dynamic gestures only, not knowing how it will perform on static gestures. On the other hand, static gestures can be treated as a special type of dynamic gesture, having one frame stretched out to the desired number of timesteps. Not to mention our dataset was constructed in a way where static gestures have a slight difference in coordinates between t and $t+1$, which makes it possible to look at the static gesture as a very slow type of dynamic gesture.

4.2.2.1 Selection of the Optimal Dropout Rate

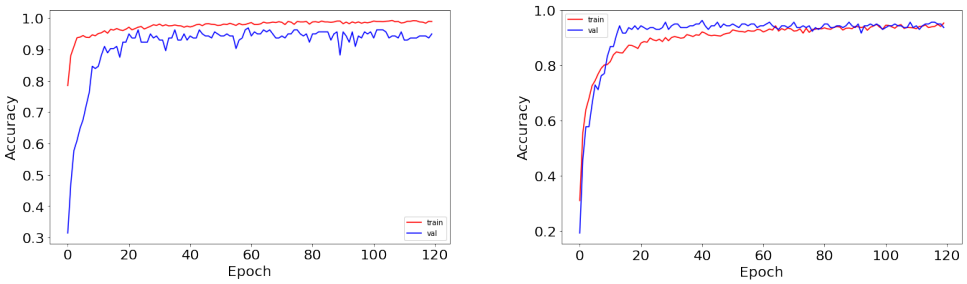
To avoid the problem of overfitting, we used dropout regularization. A technique, that during training, randomly drops out a number of neurons in layers, thus ignoring their connections in the network. This creates a new smaller network and, in essence, simulates model ensembling without creating multiple networks.

We used ASL Dataset for its variety in samples and possible more complexity over our own original dataset. However, despite its mislabeling of features, it still serves well for model benchmarking and validation.

The optimal dropout rate was selected through several experiments using dropout values in a range of 0.0 to 0.9. As seen below, not using dropout regulation caused a visible difference between train and validation accuracies through the course of training. The difference stayed poor, almost the same up to the value of 0.4, which showed improved results but not necessarily optimal. The difference was most promising when using dropout values of 0.5 and 0.6, where 0.6 performed better. However, the performance gets worse from 0.7 and on. The overall results indicate that the optimal dropout value is between 0.5 and 0.6. The value of 0.6 is satisfactory enough for our uses.

4.2.2.2 Optimal number of stacked layers

While searching for optimal dropout rate, we had performed the experiment across different depths of the network, more precisely 1 to 5 layered bidirec-



(a) No dropout regularisation applied

(b) Dropout rate = 0.6

Figure 4.3: Train accuracies compared to validation accuracies through the course of learning

tional LSTM networks. The dropout value appeared to have held the same characteristics for the different number of stacked layers.

The additional testing of the network's depth was to find out whether using a different number of layers than proposed in [47] would make any improvements in overall prediction performance.

To find the optimal depth of the network, we have adopted k-fold Cross-validation. Cross-validation is often used to evaluate the performance of machine learning models on limited datasets. The entire data set is split randomly into k folds, in our case $k = 5$, then train the model using $k - 1$ folds and hold the remaining fold to measure the model's accuracy. We will be repeating this process for each fold and then calculate the average performance across all folds.

Table 4.1: Average recognition accuracies across different depths of bidirectional lstm architectures using 5-fold on 200 epochs

Number of layers	*5-fold (%)*
1	88,14
2	89,07
3	87,48
4	86,31
5	87,98

The two-layered bidirectional LSTM architecture acquired the best performance results compared to other depths. Increasing the number of epochs would improve the results of architectures with more layers, but we would suffer on the side of the training time required. In conclusion, choosing the two-layered architecture is a good compromise between accuracy and training time.

The two-layered bidirectional LSTM architecture successfully learned dy-

namic gestures based on its characteristic movement, and it was also successful in classifying static gestures, both in real-time recognition.

4.3 Real-time recognition

The demo application for real-time recognition is in the form of a simple console application, supporting multiple LMCs using MultiLeap [34] library described in chapter 3 and with key commands for LMC calibration. When a hand gesture is presented, the application prints out the prediction, which is considered a valid prediction if the value passes the threshold of 90% accuracy. The application currently works with only one hand. When more than one hand is presented, the user is notified, and no prediction is made. The demo application served mainly for debugging and experimental purposes.

We chose C++ as our primary language for real-time recognition since C++ and C# are widely used programming languages in graphic engines such as Unity, PhyreEngine, or Unreal, which opens the possibility of integrating our application into graphic engines in future works.

4.3.1 Cppflow 2

Our application uses the trained model from section 4.2 and deploys it in a C++ environment. More specifically, we exported the model in .tf file format and imported it into C++ using CppFlow 2.

Cppflow 2 is an API created by Sergio Izquierdo, allowing the user to run TensorFlow in C++ without the necessity of installing and compiling TensorFlow itself. CppFlow 2 serves as a Tensorflow C API wrapper providing a simple C++ interface similar to TensorFlow callings in Python environment [51].

4.3.2 Sliding window

The data collection, frame collection yield by LMC, starts once a hand is presented in LMC's field of view and stops if the hand falls out of the view. Let us introduce a situation where during the stream of data yield by LMC, we change our hand gesture from a "fist" to a "peace sign". We want to classify both of these gestures, but how do we determine where one gesture ends and the other starts. To tackle the presented scenario, we adopted the concept of *sliding window*.

The basic idea is to have a window of fixed size T, which slides through our data stream and captures a certain portion of it. It is important to remain the same T value as we chose to record our dataset. Otherwise, the shape of the captured data would differ from the input shape of our trained model. It is worth mentioning that using a wider window size creates a noticeable time delay between a presented gesture and its prediction. On the other hand,

using a size too small, there would be a possibility of not capturing a dynamic gesture completely, leading to possible inaccurate predictions. In our case we used $T = 60$. Considering a situation where the collected data is less than T, the missing data are then set to zeroes. If we present more than one hand, the stream is invalidated, collected data are flushed, and the window will begin sliding again once there is only one hand in the controller's view. If we wanted to have an additional feature of recognizing multiple hands, we would need to implement as many sliding windows as there are hands. Not to mention, there are dynamic gestures characterized by two hands. As an example, *clapping* can be considered as one. This characterization would require modifying our dataset structure and explore additional features. We leave this topic for future works.

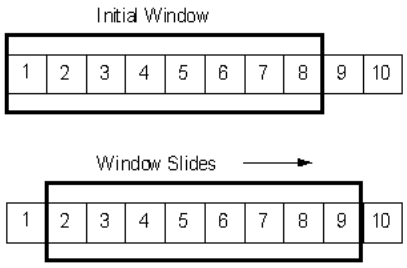


Figure 4.4: General idea of a sliding [52]

For each window, we then calculate features for classification and output the prediction of the captured portion. Features must be preprocessed exactly the same as they were for model training, in our case, same as described in section 2.3.3.1.

The window slides by 10 frame, in other words, throws away the oldest frames and adds in the newest acquired. The sliding rate should be further tuned for optimal value. If we throw away too many frames, we risk leaving some gestures unclassified. On the other hand, sliding by one frame can be demanding on hardware, where weaker computer builds may not keep up, and the prediction may stutter.

Chapter 5

Experiments

One other goal of the thesis is to evaluate the recognition performance based on a number of connected LMC sensors and test the capabilities of MultiLeap library in the real-time environment using various layouts with a different number of sensors. Results from one connected sensor served as the reference, mostly whether having multiple LMC sensors does improve the recognition of difficult angles or not. We also examined how effective was MultiLeap's merging for gestures presented in simple default angles.

We used the demo application and trained model as described in section 4.3. The model was trained on our original dataset from section 4.1.3.

5.1 Testing Method

For each gesture, we performed 1000 classifications. We did not exclude classifications with corrupt sequences, such as when the LMC sensor did not get a correct hand skeletal alignment of the presented hand. We wanted to emulate genuine user interaction with the LMC. Each gesture was held in various positions and angles in a certain span of time until the number of classification was not satisfied.

Results of multiple LMC sensors are an average of 5 different automatic calibrations. There is currently no telling how well sensors were calibrated. Therefore, we want to avoid generalizing MultiLeap capabilities base on experiments conducted on only one calibration. An experimental feature is being currently worked on to recognize the calibration quality, but it was not dependable enough in the time of our experiments.

Dynamic gestures were not tested, as it is hard to evaluate the percentage of correct classifications in a continuous stream of data. If we test dynamic gestures without any mix-up with static gestures, it will defeat the purpose of testing in a real-time environment. We want to evaluate the performance when a dynamic gesture is performed in the middle of the sequence of static ges-

tures, as to whether the trained model is capable of recognizing the difference between the static gesture of *number 5* as oppose to having *number 5* moving quickly to one side, doing a *swipe*. Despite not dedicating any experiments to dynamic gestures, we can still evaluate responsiveness and general correctness when we perform it. We also want to keep track of times when a static gesture gets misclassified for a dynamic gesture, and what is the percentage of valid classification to determine whether the threshold for prediction probability was not set too high.

5.1.1 One Leap Motion Sensor

Experiments for a single connected sensor were performed with LMC's VRVisualizer to understand better how the skeletal structure, which gets classified, looks. It helps us distinguish whether misclassification is caused by our trained model or by LMC's hand joint misalignment.

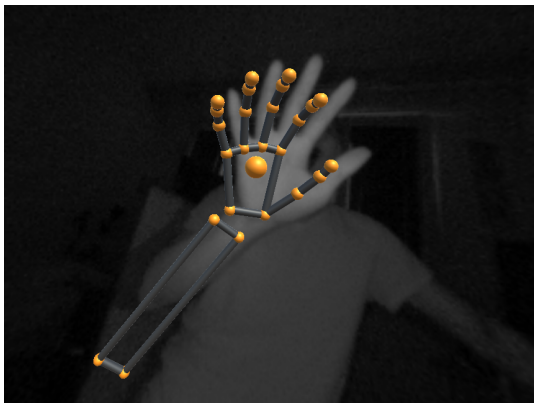


Figure 5.1: VRVisualizer

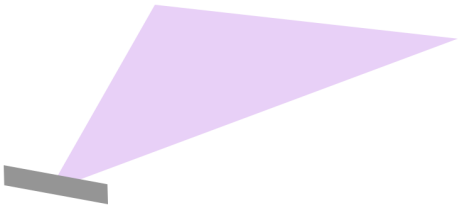


Figure 5.2: Ilustrative field of view of 1 LMC sensor

The classification was responsive with every presented gesture, including dynamic gestures. Still, the limitation of having only one sensor presents itself when we perform a gesture of *number 1* pointing towards the sensor. The sensor struggles to recognize the pointing finger as being straight, and

often times it misclassifies the gesture as a fist, or the prediction does not meet the threshold requirement. It also struggles with the prediction of *number 3* and *number 4*, wherein various angles, the thumb is not recognized by LMC as bent and straighten correctly. Gestures then get confused with *number 2*, and *number 5* respectively. The *pinch* gesture was mostly misclassified due to hand joint misalignment by the LMC sensor. We could be questioning the trained model's performance due to training on a not optimal dataset, but when the hand joint was aligned correctly, the gestures also got classified correctly without any further complications.

The percentage of invalidated gestures was also not too high as most of the discarded predictions were of 0.8 probability and lower. We can assume that our set threshold of 0.9 is not overly strict for the prediction.

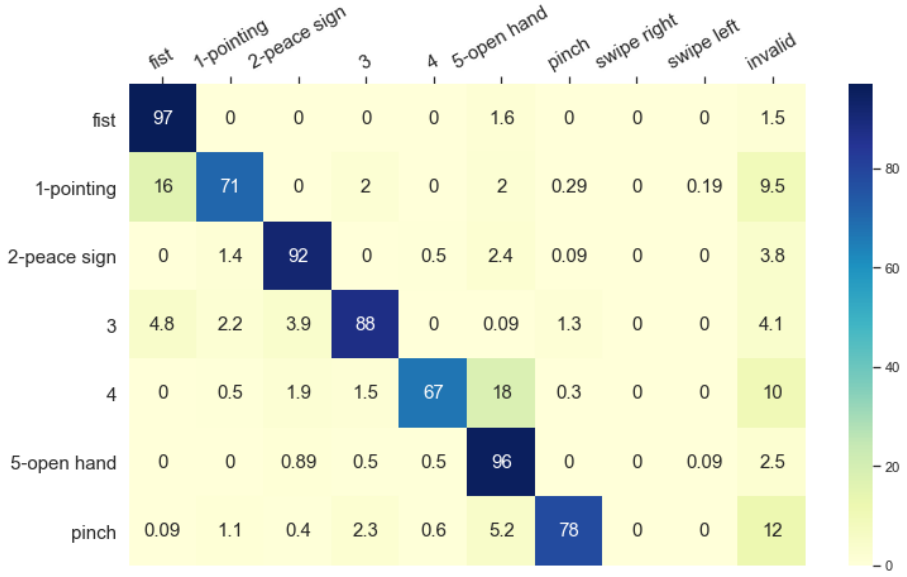


Figure 5.3: Confusion matrix of prediction by using 1 LMC sensor

5.1.2 Two Leap Motion Sensors

For two sensors, we explored several layouts with different calibrations. We could not utilize VRVisualizer as we did with one LMC sensor. The MultiLeap library does not have a feature of visualizing the merged hand or any visualization during calibration at the moment. Therefore, we could not accurately evaluate how the merged hand structure looks like during classification or calibration. Thus, our experiments are limited in correction when evaluating the accuracy of multiple LMC sensors. We recommend conducting experiments again once the visualizing feature is implemented.

5.1.2.1 Parallel Layout

The parallel layout, with sensors facing each other, could not be tested. LMC sensors expect to receive its emitted IR signals to return from a hand. Instead, the emitted IR signal is received by the other sensor and vice versa. The behavior will confuse LMC recognition making the sensors think there is a hand presented even when it is not. Thus we assume this parallel layout is inappropriate to be a part of any setup with multiple LMC sensors.

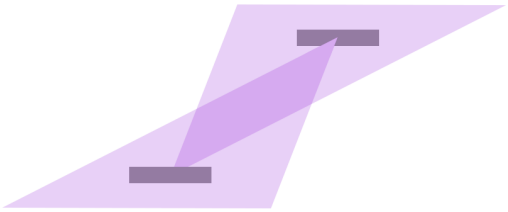


Figure 5.4: Parallel placement layout for 2 LMC sensors

5.1.2.2 Non-parallel Layout

Sensors were placed next to each other at a slight angle facing inwards, avoiding sensors to be disturbed by other's emitted IR signals.

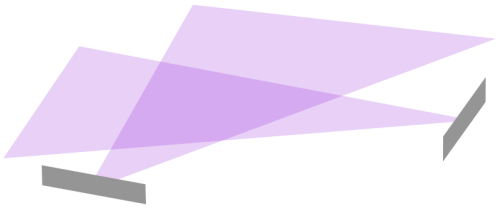


Figure 5.5: Non-parallel placement layout for 2 LMC sensors

Using MultiLeap [34] showed improvements in classifying gestures in difficult angles, which it struggled in previous experiments with one connected sensor. The MultiLeap was able to capitalize on the advantage of having multiple fields of view for capturing a presented hand.

Despite improved performance with various angles, the number of invalidated classifications had increased. The average prediction probability for gesture was 0.697, which is most likely caused by misalignment when merging hands. The number of confusion between gestures had also increased. Both could be caused by poor calibration, which there is currently no way to identify the calibration quality in order to avoid this issue.

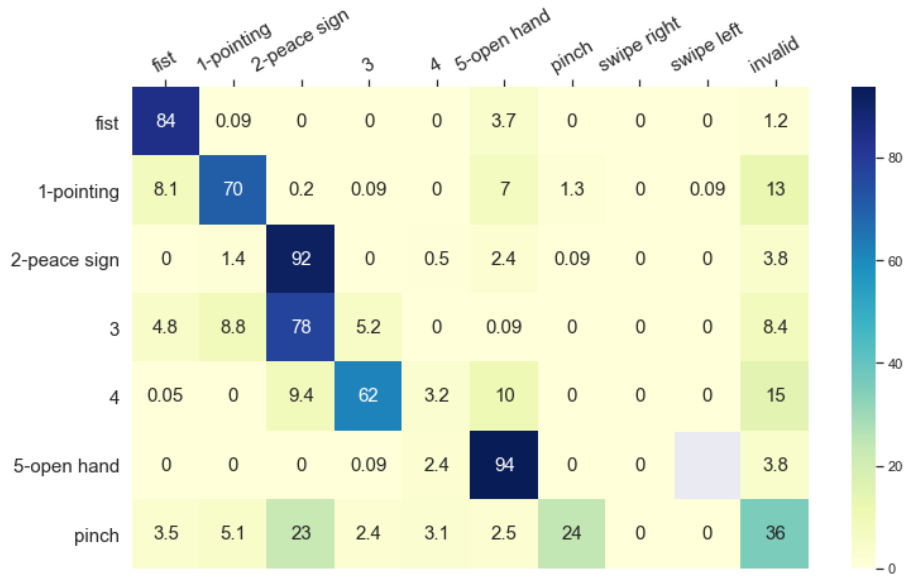


Figure 5.6: Confusion matrix of prediction by using 2 LMC sensors

The calibration quality significantly affects the prediction accuracy. More specifically, when we tested *pinch* gesture, the measured average accuracy was mere 24.4%, but when tested again with a new calibration, the accuracy improved up to 83%, which is already better than referential results of one connected sensor. The calibration could also affect dynamic gestures. While performing experiments, dynamic gestures were not responsive as they were with only one connected sensor. Often times they were not recognized at all, but their responsiveness varied with different calibrations.

The MultiLeap's current most notable issue is an incorrect number of reported hands. Our demo application does not make a prediction when there is more than one hand presented. This feature is conflicting with the current bug of MultiLeap, where it frequently returns two hands instead of one, even though only one is present, which from a user's point of view makes the demo application almost unusable for any accurate consecutive recognition. However, the issue is known, and its fix is currently in development.

5.1.3 Three Leap Motion Sensors

Sensors were carefully placed into a triangular layout so that LMC sensors don't emit IR signals to others, recreating a similar misrecognition issue as in section 5.1.2.1.

Using three sensors shares similar behavior as using two sensors. The improvement in recognition of difficult angles was improved upon additional sensors in fewer cases of calibration. Most of the calibrations made did not

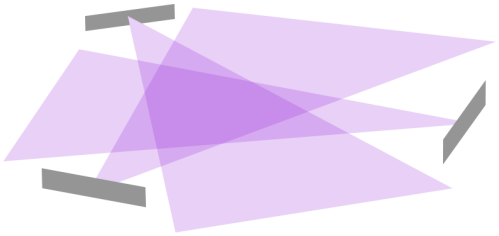


Figure 5.7: Placement layout for 3 LMC sensors

capitalize properly on the advantage of having multiple sensors.

The setup suffered similarly, if not at times more, on the side of increased invalidated classification. The average prediction probability for gesture was 0.6841. The issue with the incorrect number of reported hands still persists in a similar frequency as it did with two connected sensors.

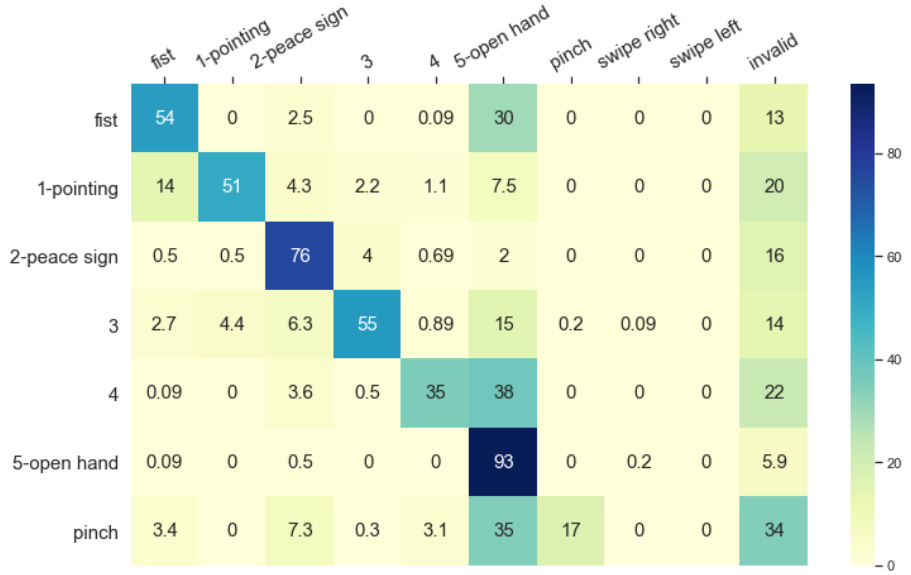


Figure 5.8: Confusion matrix of prediction by using 3 LMC sensors

The *pinch* alongside with *number 4* gesture has low accuracy across all 5 different calibrations. They often got confused with the gesture of *number 5*. Dynamic gestures also suffered where the responsiveness seemingly did not change with various calibrations. We can only assume that having more sensors is harder to calibrate and more demanding on calibration quality.

Chapter 6

Conclusion

The goal of the thesis was to utilize LeapMotion sensors in relation to gesture recognition, create a pre-trained model and use it to evaluate the capabilities of the MultiLeap library.

We explored publicly available ASL and SHREC datasets, discovered possible feature mislabeling in the ASL dataset, and discussed the dataset's suitability for our purposes. In relation to the discussion, we created a simple way to sample our original dataset. The process features a simplified ability to detect moving sequences while sampling dynamic gestures. We created a dataset consisting of 7 static gestures (fist, 1-pointing, 2-peace sign, 3, 4, 5-open fist, pinch) and 2 dynamic gestures (swipe left, swipe right). Our dataset served the purpose but is not optimal for any benchmark evaluation. The dataset lacks complexity as well as the number of users used for sampling. We suggest expanding the dataset in future works with the engagement of more users, increasing the gesture set as well as its complexity.

ASL dataset, despite its mislabeling, was used for benchmarking and performance evaluation in the testing environment. Our dataset was then used for real-time deployment. Both datasets applied on 4-layered LSTM as well as 2-layered bidirectional LSTM. The 4-LSTM showed promising high results in the testing environment but did not have the desired behavior in real-time deployment due to the inability to learn dynamic gestures, while 2-layered bidirectional LSTM performed well on both fronts.

We also explored the optimal number of layers and dropout rates for bidirectional LSTMs, resulting in having 2 layers in combination with the 0.6 dropout rate, which is an optimal compromise between accuracy and required training time. As a result, the 2-layered bidirectional LSTM achieved 89.07% accuracy performing 5-fold cross-validation.

Using the pre-trained model, we created a demo application for debugging and experimental purposes in the form of a simple console application, which supports the connection of multiple Leap Motion sensors. Despite not having an optimal dataset, we achieved to create a responsive classifier for static

gestures and dynamic gestures, suitable to be integrated into other applications. However, the application can only classify one hand. Classification of multiple hands would require a new feature structure of the dataset and an improved sliding window solution for real-time recognition. We leave this for future works.

We have conducted several experiments to evaluate the model's performance in the real-time environment and evaluate the performance of the MultiLeap library by using multiple Leap Motion sensors. We explored several setups and the way they can affect Leap Motion detection. We have pointed out issues with the current MultiLeap library alongside its promising results in classifying hand gestures with challenging angles while using multiple sensors. We did not explore all possible setups there are, but it was enough to have a general idea of the current MultiLeap state. We will revisit our setups and explore more in future works with improved MultiLeap.

Bibliography

[1] Chen, Y.-Y.; Lin, Y.-H.; et al. Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes. *Sensors*, 05 2019, doi:10.3390/s19092047.

[2] Bengio, Y.; Goodfellow, I.; et al. *Deep learning*, volume 1. Citeseer, 2017, ISBN 0262035618, 166–485 pp.

[3] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, volume 5, no. 4, 1943: pp. 115–133, ISSN 0007-4985.

[4] Krishtopa. What Are Neural Networks, Why They Are So Popular And What Problems Can Solve. 2016. Available from: `https://steemit.com/academia/@krishtopa/what-are-neural-networks-why-they-are-so-popular-and-what-problems-can-solve`

[5] Rosenblatt, F. The Perceptron: A Probabilistic ModelFor Information Storage And Organization In The Brain. *Psychological Re-view*, 1958: p. 2047, doi:0.1037/h0042519.

[6] Kozák, M. Static malware detection using recurrent neural networks. [cit. 2020-12-28]. Available from: `https://dspace.cvut.cz/bitstream/handle/10467/88342/F8-BP-2020-Kozak-Matous-thesis.pdf?sequence=-1&isAllowed=y`

[7] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015.

[8] Rojas, R. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013, ISBN 9783642610684, 37–99 pp.

[9] Leskovec, J.; Rajaraman, A.; et al. *Mining of massive data sets*. Cambridge university press, 2020, ISBN 9781108476348, 523–569 pp.

[10] Maladkar, A. I. M., Kishan. 6 Types of Artificial Neural Networks Currently Being Used in ML. [cit. 2020-12-25]. Available from: https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/

[11] Lipton, Z. C.; Berkowitz, J.; et al. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015: pp. 5–25, ISSN 2331-8422. Available from: https://arxiv.org/pdf/1506.00019.pdf

[12] Wiki, B. M. . S. Feedforward Neural Networks. [cit. 2020-12-25]. Available from: https://brilliant.org/wiki/feedforward-neural-networks/

[13] Towards AI, M. Main Types of Neural Networks and its Applications-Tutorial. Aug 2020, [cit. 2020-12-25]. Available from: https://medium.com/towards-artificial-intelligence/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e

[14] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, url-http://www.deeplearningbook.org.

[15] Wiki, B. M. . S. Backpropagation. [cit. 2020-12-25]. Available from: https://brilliant.org/wiki/backpropagation/

[16] How Do Convolutional Layers Work in Deep Learning Neural Networks? April 2020, [cit. 2020-12-25]. Available from: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[17] MathWorks. Convolutional Neural Network. [cit. 2020-12-25]. Available from: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[18] Science, T. D. A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way. Dec 2018, [cit. 2020-12-25]. Available from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[19] Science, T. D. Recurrent Neural Networks. Jun 2019, [cit. 2020-12-25]. Available from: https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[20] IBM. What are Recurrent Neural Networks? [cit. 2020-12-25]. Available from: https://www.ibm.com/cloud/learn/recurrent-neural-networks

[21] Medium. Understanding Recurrent Neural Networks in 6 Minutes. Sep 2019, [cit. 2020-12-25]. Available from: https://medium.com/x8-the-ai-community/understanding-recurrent-neural-networks-in-6-minutes-967ab51b94fe

[22] Schuster, M.; Paliwal, K. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, volume 45, 12 1997: pp. 2673 – 2681, doi:10.1109/78.650093.

[23] Olah, C. Understanding LSTM Networks [online]. [cit. 2020-12-28]. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[24] Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural computation*, volume 9, 12 1997: pp. 1735–80, doi:10.1162/neco.1997.9.8.1735.

[25] Phi, M. Illustrated Guide to LSTM's and GRU's: A step by step explanation [online]. [cit. 2020-12-28]. Available from: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[26] Holzner, A. LSTM cells in PyTorch [online]. Oct 2017, [cit. 2020-12-28]. Available from: https://medium.com/@andre.holzner/lstm-cells-in-pytorch-fab924a78b1c

[27] Graves, A.; Mohamed, A.; et al. Speech Recognition with Deep Recurrent Neural Networks. *CoRR*, volume abs/1303.5778, 2013, 1303.5778. Available from: http://arxiv.org/abs/1303.5778

[28] Brownlee, J. Stacked Long Short-Term Memory Networks [online]. Aug 2019, [cit. 2020-12-28]. Available from: https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

[29] Vafaei, F.; Slator, B.; et al. Taxonomy of Gestures in Human Computer Interaction. 12 2013.

[30] Avola, D.; Bernardi, M.; et al. Exploiting Recurrent Neural Networks and Leap Motion Controller for the Recognition of Sign Language and Semaphoric Hand Gestures. *IEEE Transactions on Multimedia*, volume 21, no. 1, Jan 2019: p. 234–245, ISSN 1941-0077, doi:10.1109/tmm.2018.2856094. Available from: http://dx.doi.org/10.1109/TMM.2018.2856094

[31] Microsoft. Azure Kinect [online]. [cit. 2020-12-25]. Available from: https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RWqOsq?ver=2e37

[32] Microsoft. Azure Kinect DK documentation [online]. [cit. 2020-12-25]. Available from: https://docs.microsoft.com/en-us/azure/Kinect-dk/

[33] Weichert, F.; Bachmann, D.; et al. Analysis of the Accuracy and Robustness of the Leap Motion Controller. *Sensors (Basel, Switzerland)*, volume 13, 05 2013: pp. 6380–6393, doi:10.3390/s130506380.

[34] Novacek, T.; Martin, C.; et al. Project MultiLeap: Fusing Data from Multiple Leap Motion Sensors. In *Proceedings of 2021 IEEE 7th International Conference on Virtual Reality (ICVR 2021)*, ICVR 2021, New York, NY, USA: IEEE, 2021, pp. 19–26.

[35] Ultraleap. Tracking: Ultraleap Stereo IR 170 Evaluation Kit [online]. [cit. 2020-12-26]. Available from: https://www.ultraleap.com/product/stereo-ir-170/

[36] Prototyping, S. Ultraleap Stereo IR 170 [online]. [cit. 2020-12-26]. Available from: https://www.smart-prototyping.com/Ultraleap-Stereo-IR-170

[37] Microsoft. Support Vector Machine - Introduction to Machine Learning Algorithms [online]. 7 2018, [cit. 2020-12-25]. Available from: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[38] Alexandre Savaris, A. v. W. A. Comparative evaluation of static gesture recognition techniques based on nearest neighbor, neural networks and support vector machines. *J Braz Comput Soc*, volume 16, 2010: p. 147–162, doi:10.1007/s13173-010-0009-z.

[39] Chen, Y.; Tseng, K. Developing a Multiple-angle Hand Gesture Recognition System for Human Machine Interactions. 2007: pp. 489–492, doi:10.1109/IECON.2007.4460049.

[40] Dogra, M. Support Vector Machine: Explained and Implemented [online]. [cit. 2021-01-17]. Available from: https://immohann.medium.com/support-vector-machine-explained-and-implemented-bda7f22126a

[41] Dominio, F.; Donadeo, M.; et al. Hand Gesture Recognition with Depth Data. 2013: p. 9–16, doi:10.1145/2510650.2510651. Available from: https://doi.org/10.1145/2510650.2510651

[42] Ren, Z.; Yuan, J.; et al. Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. *MM'11 - Proceedings of the 2011 ACM Multimedia Conference and Co-Located Workshops*, 11 2011: pp. 1093–1096, doi:10.1145/2072298.2071946.

[43] Mapari, R. B.; Kharat, G. American Static Signs Recognition Using Leap Motion Sensor. 2016, doi:10.1145/2905055.2905125. Available from: https://doi.org/10.1145/2905055.2905125

[44] Lupinetti, K.; Ranieri, A.; et al. 3D Dynamic Hand Gestures Recognition Using the Leap Motion Sensor and Convolutional Neural Networks. 2020: pp. 420–439.

[45] Boulahia, S. Y.; Anquetil, E.; et al. Dynamic hand gesture recognition based on 3D pattern assembled trajectories. 2017: pp. 1–6, doi:10.1109/IPTA.2017.8310146.

[46] Ameur, S.; Khalifa, A. B.; et al. A comprehensive leap motion database for hand gesture recognition. 2016: pp. 514–519, doi:10.1109/SETIT.2016.7939924.

[47] Yang, L.; Chen, J.; et al. Dynamic Hand Gesture Recognition Based on a Leap Motion Controller and Two-Layer Bidirectional Recurrent Neural Network. *Sensors*, volume 20, 04 2020: p. 2106, doi:10.3390/s20072106.

[48] Savitzky, A.; Golay, M. J. E. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Anal. Chem.*, volume 36, no. 8, July 1964: pp. 1627–1639, doi:10.1021/ac60214a047. Available from: http://dx.doi.org/10.1021/ac60214a047

[49] Kabsch, W. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, volume 32, no. 5, Sep 1976: pp. 922–923, doi:10.1107/S0567739476001873. Available from: https://doi.org/10.1107/S0567739476001873

[50] De Smedt, Q.; Wannous, H.; et al. SHREC'17 Track: 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset. Apr. 2017: pp. 1–6, doi:10.2312/3dor.20171049. Available from: https://hal.archives-ouvertes.fr/hal-01563505

[51] Izquierdo, S. CppFlow 2 [online]. [cit. 2021-3-18]. Available from: https://github.com/serizba/cppflow

[52] Sliding window protocol [online]. [cit. 2021-06-12]. Available from: https://www.toppr.com/ask/question/sliding-window-protocol-have/

Appendix A

Acronyms

ANN Artificial Neural Network

RNN Recurrent Neural Network

BRNN Bidirectional Recurrent Neural Network

CNN Convolutional Neural Network

LSTM Long Short-Term Memory

BLSTM Bidirectional Long Short-Term Memory

DLSTM Deep Long Short-Tzerm Memory

ReLU Rectified Linear Unit

LMC Leap Motion Controller

LED Light Emitting Diode

SVM Support Vector Machine

MLP Multilayer Perceptron

ASL American Sign Language

SHREC Shape Retrieval Contest

API Application Programming Interface

Appendix B

Contents of enclosed CD

- `README.md` ..... the Markdown file with description
- `executables` ..... the directory with executables
  - `Dataset` ..... the directory of the original dataset
  - `TrainedModel` ..... the directory of trained model
  - `DataSampler.exe` ..... data sampling application
  - `model_trainig.py` ..... model training Python script
  - `GestureApp.exe` ..... gesture recognition demo application
- `src` ..... the directory of source codes
- `text` ..... the directory of LaTeX source codes of the thesis
- `environment.yml` ..... configuration file for conda environment
- `BP_Viet_Anh_Tran_2021.pdf` ..... the thesis text in PDF format