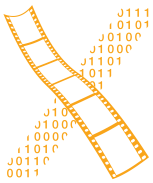

Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

Development of a Hand Detection on a Large-Area Textile Capacitive Pressure Sensor Matrix

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Van Huy Dao

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Thomas Gries

Registration date: 09.09.2021
Submission date: 07.01.2022

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Dao, Van Huy | 352601

Name, Vorname/Last Name, First Name | Matrikelnummer (freiwillige Angabe) Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/ ~~Masterarbeit~~* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present ~~paper~~/Bachelor thesis/~~Master thesis~~* entitled

Development of a Hand Detection on a Large-Area Textile Capacitive Pressure Sensor Matrix

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, 07.01.2022

Ort, Datum/City, Date | Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Aachen, 07.01.2022

Ort, Datum/City, Date | Unterschrift/Signature

Contents

List of Figures

List of Tables

Abstract

The combination of a textile with a sensor, called smart textile, can open up new areas of application and can be a further step towards smart homes and smart devices. Sensors integrated into a textile can be used in most textile applications.

In this work, we deal with a prototype of a capacitive pressure sensor matrix and test its performance for a first hand detection. We present the wiring of the prototype with a microcontroller to receive and collect the data and then process it for training a machine learning model with convolutional neural networks. We explore the capabilities of the prototype with different models. These models were trained with different datasets, from simple datasets with one hand location on the prototype to more complex datasets with multiple hand locations.

Our investigation shows that the prototype is able to provide data for a hand detection for one to two hand locations on the prototype. Further possible improvements to both the prototype and hardware components are sketched out to obtain cleaner and more stable capacitive values in order to increase the prototype's capabilities for future work.

Überblick

Die Kombination eines Textils mit einem Sensor, genannt Smart Textile, kann neue Anwendungsbereiche eröffnen und ein weiterer Schritt in Richtung Smart Home und Smart Devices sein. In ein Textil integrierte Sensoren können in den meisten Textilanwendungen eingesetzt werden.

In der vorliegenden Arbeit befassen wir uns mit einem Prototyp einer kapazitiven Drucksensormatrix und ihre Leistung für eine erste Handerkennung. Die Verkabelung des Prototyps mit einem Mikrocontroller wird vorgestellt, um die Daten zu empfangen, zu sammeln und sie dann für das Training eines maschinellen Lernmodells mit Convolutional Neural Networks zu verarbeiten. Die Fähigkeiten des Prototyps wird mit verschiedenen Modellen untersucht. Diese Modelle wurden mit verschiedenen Datensätzen trainiert, von einfachen Datensätzen mit einer Handposition auf dem Prototyp bis hin zu komplexeren Datensätzen mit mehreren Handpositionen.

Weitere mögliche Verbesserungen sowohl des Prototyps als auch der Hardware-Komponenten werden aufgezeigt, um sauberere und stabilere kapazitive Werte zu erhalten, um die Leistungsfähigkeit des Prototyps für zukünftige Arbeiten zu erhöhen.

Acknowledgements

Firstly, I would like to thank Prof. Dr. Jan Borchers and Prof. Dr. Thomas Gries for examining my bachelor thesis.

Also, I would like to thank my supervisors, René Schäfer and Vadim Tenner, for their feedback, support and time.

Additionally, I would like to thank my family and friends for their support.

Lastly, I would like to thank everyone at the *Digital Capability Center* for their discussions and suggestions.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in American English. We refer to the first and unidentified third person in plural form.

Chapter 1

Introduction

Textiles are ubiquitous, and it is impossible to imagine clothing, the seats of cars and airplanes or the interiors of homes without them. In recent years, they also became the subject of more intensified research, e.g. in smart home applications. These often work with inputs gathered via touchscreens or touchpads. The most common technologies for this are capacitive or resistive. These have the major disadvantage that they are neither flexible or deformable. Smart textiles can be a solution to this obstacle. Smart textiles are textiles combined with sensors to provide intelligent functions and can open up additional and new areas of application. A sensor matrix integrated into a sports mat could monitor and analyze exercises to provide feedback to the user on how the exercises are being performed. In a transport vehicle, a sensor matrix could detect the points of contact between seat and person and thus regulate the seat heating more efficiently.

In the realm of smart home technologies, smart textiles can open up new areas of applications.

In the current project at the *Institut für Textiltechnik der RWTH Aachen (ITA)* we develop sensor techniques for the recognition of pressure patterns on sensor matrices. In this bachelor thesis, as part of this project, we deal with a large-area textile capacitive pressure sensor matrix combined with a sports mat. Many other approaches dealt with the development of a capacitive sensor matrix on textiles (see Chapter 3 “State of the Art”), whereby these approaches usually do not exceed the size of a DINA4 sheet.

This work deals with a large-area textile capacitive pressure sensor matrix prototype.

Furthermore, many of these approaches are based on single sensor plates. The prototype created for this work was developed to test the functionality of a large-area capacitive pressure sensor matrix (2 m x 1 m) made of conductive aluminum strips.

To test the capabilities of the prototype for the first time, the goal of this work is to develop an automatic hand detection. This involves answering two main research questions:
1) How can the data gathered on the sensor matrix be processed to use them as an input for the training of a machine learning model?
2) Is the sensor matrix data suitable for training a model that can subsequently recognize a hand in real time?
For the first part, the sensor matrix is wired to a microcontroller to obtain the capacitive values. A system is then developed to automatically label the data coming from the prototype using a webcam and a depth camera. The collected and labeled data is used to train a model, which is then evaluated with test data and in real time predictions.

The aim is to develop a first hand detection using machine learning techniques to test the capability of the prototype.

1.1 Outline of the Thesis

The bachelor thesis is divided into a theoretical and a practical part. At the beginning, Chapter 2 "Theoretical Background" introduces the basic concepts of machine learning and capacitive sensing that are necessary for understanding this thesis.

Chapter 3 "State of the Art" gives an insight into recent published work dealing with capacitive sensor matrices on textiles. We also take a look at some machine learning concepts used for capacitive sensor matrices.

In Chapter 4 "Methodology", we briefly present the method of operations used in the different work steps for this thesis, which are described in detail in the following chapters.

Chapter 5 "Capacitive Pressure Sensor Matrix Prototype" shows the construction of the capacitive pressure sensor

matrix prototype. We also present the wiring of the prototype with a microcontroller to measure the capacitive values of the sensor matrix.

After presenting the prototype, we describe the system for automatically labeling the data from the sensor matrix in Chapter 6 "Machine Learning Models for Hand Detection". After that, we present the architecture of the model that is used for training. We also show the results of the training and whether the trained models can recognize hand inputs in real-time.

In Chapter 7 "Discussion" we discuss the results from Chapter 6 "Machine Learning Models for Hand Detection" and evaluate the capability of the trained models and the prototype. We also address additional steps that can be taken in the future to eliminate or minimize the identified challenges and problems.

Based on the elaborations, Chapter 8 "Summary and Future Work" summarizes the main findings and outlines steps for the future work.

Chapter 2

Theoretical Background

This chapter introduces basic concepts of machine learning. To understand how machine learning works, we first introduce a basic neural network, followed by a convolutional neural network. Secondly, the structure of a parallel plate capacitor is explained.

2.1 Machine Learning

Machine learning (ML) uses algorithms to analyze data in order to learn from it. The goal of learning is to find out different classifications of the data in order to assign new data to the appropriate label. The labels determine which class the data belongs to. Generally three different types of learning are distinguished: Supervised, unsupervised or reinforced [Kang and Jameson, 2018]. Supervised learning uses a labeled dataset to learn and then new data can be classified based on that knowledge. Unsupervised learning, on the other hand, learns from unlabeled datasets and attempts to find classifications on its own. Lastly, reinforcement learning works with a reward system that provides feedback when an artificial intelligence agent performs an action in a given situation. In this work, supervised learning is used, since we label the data from the prototype before training.

Machine learning algorithms can classify data.

Supervised learning is used for the machine learning model.

Artificial neural networks are used in deep learning.

One method of machine learning is deep learning. Deep learning algorithms are inspired by the architecture and functions of the neural networks of the human brain. Because of these characteristics, the neural networks used in deep learning are called artificial neural networks (ANN), which are also referred to as models [Abrahams et al., 2016].

A basic neural network consists of an input layer, a set of hidden layers and an output layer.

The simplest machine learning structure for supervised learning is the basic neural network (Figure 2.1). It consists of an input layer, a set of hidden layers with a randomly chosen number of neurons (nodes), and finally an output layer. The number of neurons in the input layer depends on the size of the input. The number of neurons in the output layer is identical to the number of output classes. Each neuron of a layer is connected to the neurons of the previous and the next layer. Each connection has a certain weight, which is adjusted during training and represents the strength of the connection between two neurons. The output of the neuron is calculated using the following equation [Alom et al., 2019]:

$$y = \varphi \left(B + \sum_{j=1}^{m} w_j x_j \right) \tag{2.1}$$

where w is the weighting of signal j for the neuron, x is the output of signal j, B is the bias of the neuron, and φ is the activation function. Each neuron has a bias and it determines whether and to what extent a neuron is activated. The activation function converts the neuron's signal into the final output signal and transforms linear inputs into nonlinear outputs. Basically, the activation function decides which neurons should be activated to achieve high accuracy. There are several activation functions such as `linear`, `tanh`, `relu` and `softmax` to name a few [Abrahams et al., 2016]. In this work, the activation functions `relu` and `softmax` are used. For values less or equal to zero, `relu` returns zero. For values bigger than zero that value is returned by `relu`. `softmax` scales the output into probabilities and gives back a vector of probabilities for each possible outcome.

Activation function `relu` and `softmax` are used.

During the training of a model, the model changes its weights and biases to reduce the error generated. This pro-

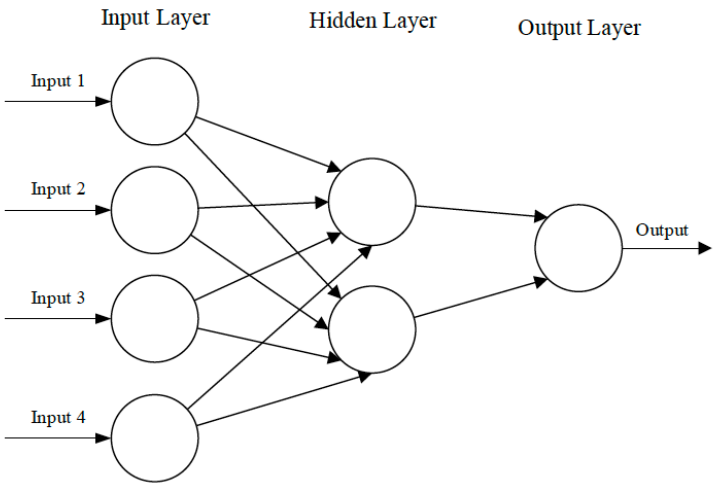

Figure 2.1: Structure of a basic neural network. Consists of an input, a hidden and an output layer. The neurons of each layer are fully connected to the previous and next layer. Image taken from [O'Shea and Nash, 2015].

cess is called learning. The main learning method is backpropagation, which is divided into two parts: forward pass and backward pass. In the forward pass, the equation y (2.1) is used. Its calculated values of the output layer neurons are compared to the correct output values to determine the loss. The loss is a number that indicates how bad a prediction was after an iteration. If the prediction is perfect, the loss is 0, otherwise the loss is greater. The loss is calculated using a function such as the mean square error. During the backward pass, the weights and bias are updated based on the calculated loss using a form of stochastic gradient descent [Hecht-Nielsen, 1989].

Backpropagation adjusts the model to reduce the error.

For each training, the number of epochs can be specified. An epoch is a single run of the entire dataset, and after each epoch the model updates the weights and calculates the loss. It is also possible to select a batch size that specifies the number of data entries per iteration during an epoch. For example, if the batch size is set to 100 and the dataset contains 1000 data entries, an epoch would have 10 iterations.

A number of epochs and a batch size can be defined.

A validation set can be taken from the dataset to check the learning process.

Besides the training dataset, it is also possible to separate a certain amount of the training data into a validation set. This validation set is used to validate the model during training and provides information on how well the updating of the weights was. The validation set can be used directly during training to verify that the model works with data that the model was not trained with. If the accuracy of the validation set is lower than the accuracy of the training set, the model is overfitted. If it is the other way around, the model is underfitted. Reasons for overfitting could be that the model only learns the training data by rote. In this case, the model works well on known data, but not on new test data. Furthermore, the architecture might be too complex. Underfitting can occur when the model has not been trained long enough and therefore has not yet learned all the patterns. Possible solutions against overfitting include adding more data to the training set, reducing the complexity of the model, or using a dropout layer. In a dropout layer, a certain number of randomly selected neurons in a layer are ignored. Underfitting can be remedied, for example, by increasing the complexity of the model [Pothuganti, 2018].

A model can be overfitted or underfitted.

Trained model can be tested with a test set and in real time.

After training the model with the training data and validating the model during training with the validation set, the test set is used. The test set is separated from the training and validation set before training and tests the final capability of the model. Finally, after the model has been trained, validated and tested, we can use the model to generate predictions for new datasets and data inputs in real time.

TensorFlow and Keras are used.

In this work, we used TensorFlow[1] with Keras[2] to implement the machine learning model. TensorFlow is an open source machine learning library, and Keras is an interface for the TensorFlow library.

[1] `https://www.tensorflow.org` (Accessed: 23.11.21)
[2] `https://keras.io/about` (Accessed: 23.11.21)

2.1.1 Convolutional Neural Networks

Besides the basic neural networks, there are several other ANNs. One of the most well-known ANNs is the convolutional neural network (CNN), which is also used in this work. CNNs are mainly used for image and video recognition and can recognize patterns in data [O'Shea and Nash, 2015]. In images, patterns can be edges, curves, shapes or colors, for example. The main difference from a basic neural network is that a CNN works with subsections rather than the entire data. This reduces the number of parameters and connections in the neural network. The architecture of a CNN includes convolutional layers, pooling layers, and fully connected layers.

CNNs can work with subsections of the data and are mainly used for image and video recognition.

In the convolutional layer, filters can detect patterns in the data. These filters are technically small matrices and slide over the entire input, which is called *convolving* and gives the layer its name. The values of these filters are instantiated with random numbers, and the number of filters determines the number of outputs for the next layer. For example, a filter of size 3 x 3, also called kernel size, convolves a 3 x 3 area of the data input into a 1 x 1 output, which is the sum of the element-wise products of the filter and the area of the data input. The CNN learns during training and changes the values of the filters accordingly. Thus, it is possible that the filters for pattern recognition are automatically created and improved during training. Furthermore, as the CNN becomes more complex, the filters can recognize more complex patterns in later layers. In a convolution layer, one can choose a number of filters and the kernel size. In addition, a stride can be specified, which determines how much the filters overlap during convolving over the data input. For a 7 x 7 input, a 3 x 3 filter and a stride of 2, the filters would overlap in one column (Figure 2.2).

In a CNN, the filters (small matrices), try to find patterns in the data.

In a convolution layer, a number of filters, the kernel size and a stride can be defined.

Furthermore, a *padding* parameter can be specified when employing a CNN layer. If padding is set to `same`, the output of the layer is prevented from shrinking in dimension by adding a margin of zeros around the actual data input. This is important if, e.g., the values in the margin

Padding prevents shrinkage of the output.

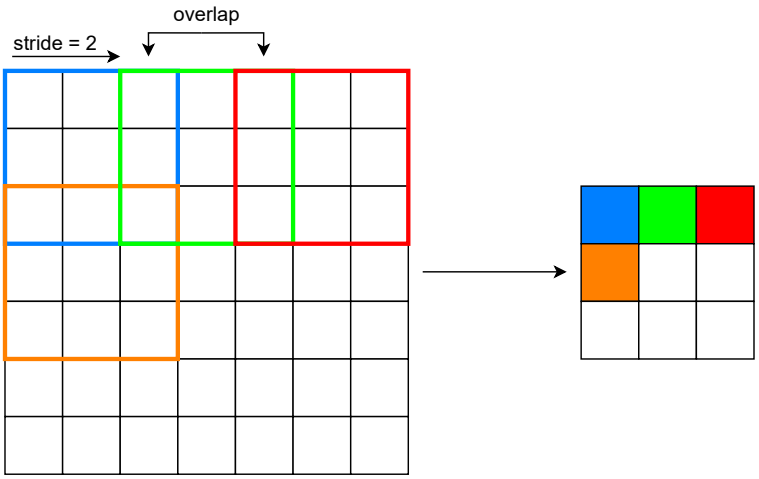

Figure 2.2: Visualization of a step in a convolutional layer. A 3 x 3 filter convolves over a 7 x 7 input. Each filter converts a 3 x 3 field to a 1 x 1 output. With a stride = 2, the filters overlap in one column.

A pooling layer reduces the size of the input to reduce its complexity.

contain important values for classification. Another important layer of a CNN is the pooling layer. *Pooling* is used to reduce the number of parameters and thus the complexity of the input for the next layers. In a pooling layer, a pooling size and a stride can be selected. Known pooling techniques are *max pooling*, *sum pooling* and *average pooling*. Max pooling returns the maximum value of the selected subsection (pooling size), sum pooling returns the sum of the selected subsection, and average pooling returns the average of the selected subsection. In this work max pooling is used. The fully connected layer corresponds to the output layer of a basic neural network.

2.2 Capacitive Sensing

The prototype works on principle of parallel plate capacitors.

Capacitors are generally one of the most commonly used components in electrical engineering and can be used to store electrical charge [Stiny, 2015]. The prototype of this work operates with the principle of the parallel plate capacitor (Figure 2.3). A parallel plate capacitor consists of

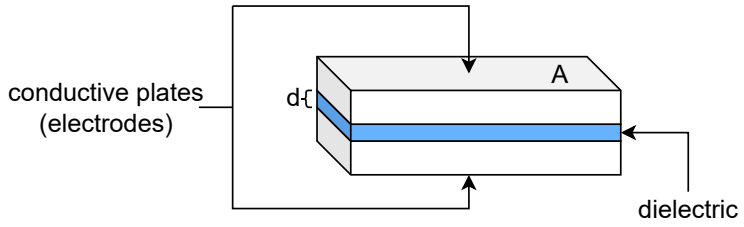

Figure 2.3: Structure of a parallel plate capacitor. It consists of a conductive plates on the top and bottom with a dielectric between them. d is the distance between the two plates and A is the area size of a plate.

two opposing conductive plates (electrodes) separated by a non-conductive layer, the dielectric. The capacitance depends on the size, thickness and material of each layer. The value of the capacitance can be calculated with the following equation [Stiny, 2015]:

$$C = \varepsilon_0 \cdot \varepsilon_r \cdot \frac{A}{d} \tag{2.2}$$

The equation 2.2 specifies the capacitance in units of farads (F). One farad is defined as [Stiny, 2015]:

$$1F = \frac{1C}{1V} = \frac{1As}{1V} = \frac{1s}{1\Omega} \tag{2.3}$$

In equation 2.2, A (in square meter) is the size of the area of a plate, d (in meter) is the distance between the two plates, ε_0 is the value of the absolute dielectric permittivity ($8.854187 \cdot 10^{-12} \frac{As}{Vm}$) and ε_r is the permittivity of the material that is used as a dielectric. When a parallel plate capacitor is connected to a direct current (DC) voltage, electrodes flow from the negative pole of the voltage source to one plate. On the other side, free electrodes flow from the other plate to the positive pole of the voltage source. After some time, both plates are electrically charged.

The capacitive value depends on the size of the plates, the distance between them and the permittivity of the material used for the dielectric.

Chapter 3

State of the Art

In this chapter, we consider the state of the art of capacitive sensor matrices on textiles. For this purpose, different approaches are presented and compared. Furthermore, some machine learning model techniques for capacitive sensors in general and on textiles are discussed.

3.1 Capacitive Sensor Matrices on Textiles

Capacitive sensor matrices on textiles can provide intelligent functions. There are different approaches to integrate a sensor matrix into a textile. These sensor matrices can be printed or woven.

A capacitive sensor matrix on highly elastic fabrics was developed by Vu and Kim [2020]. Their approach includes a capacitive touch sensor layer and a capacitive pressure sensor layer. On the touch sensor layer (Figure 3.1), four horizontal and four vertical conductive diamond-shaped strips out of silver paste are printed on a spandex fabric[1], which works as the dielectric. The diamonds are 14 mm apart and spaced 2 mm apart. To avoid possible short circuits, a laminating film made of polymer polyurethane (PET) with a thickness of 10 µm separates the nodes. The pressure sen-

Vu and Kim use one sensor layer for touch and one for pressure.

The touch sensor has a 4 x 4 format with diamond-shaped stripes.

[1] Elastic synthetic fiber

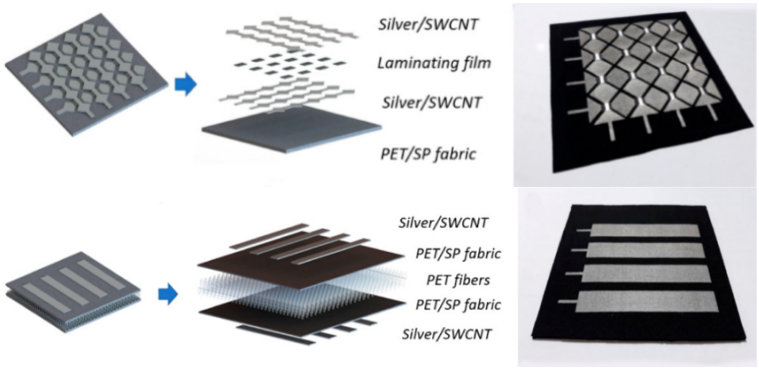

Figure 3.1: Structure of the capacitive touch sensor layer (top) and the capacitive pressure sensor layer (bottom) for simultaneous detection of touch and pressure. Images adapted from [Vu and Kim, 2020].

The pressure sensor has a 4 x 4 format with straight strips.

sor layer (Figure 3.1) also consists of four vertical and four horizontal conductive strips of silver paste, but this time in a rectilinear design. The horizontal strips are printed on the top layer and the vertical strips are printed on the bottom layer of a spandex fabric with a width of 12 mm and a spacing of 5 mm between each strip. Between these two layers is a fabric made of PET fibers as the dielectric. It was found that the recovery time through the dielectric layer was fast, with 6 ms. Additionally, the sensor had a high cycle stability of more than 20000 times. The prototype had high flexibility, was breathable, lightweight and could be easily integrated into textiles. A real-world application testing finger inputs shows that the sensor's architecture could detect and localize touch and pressure from a single finger and even two fingers.

The prototype can simultaneously sense touch and pressure from one and two fingers.

Ferri et al. developed two versions of a diamond-shaped sensor matrix.

The article by Ferri et al. [2017] also deals with a diamond-shaped capacitive sensor matrix on a textile. Two designs were created. Both use screen-printed conductive silver paste strips in a 9 x 6 design. The diamond-shaped strips are spaced 8.3/8 mm apart and have a gap of 0.5/0.4 mm (first design/second design). A ground ring is used around the sensor matrix to reduce electromagnetic interference. In the first design, the horizontal and vertical strips are directly on one layer (Figure 3.2 left). Because of this design, two additional layers are needed to prevent short circuits

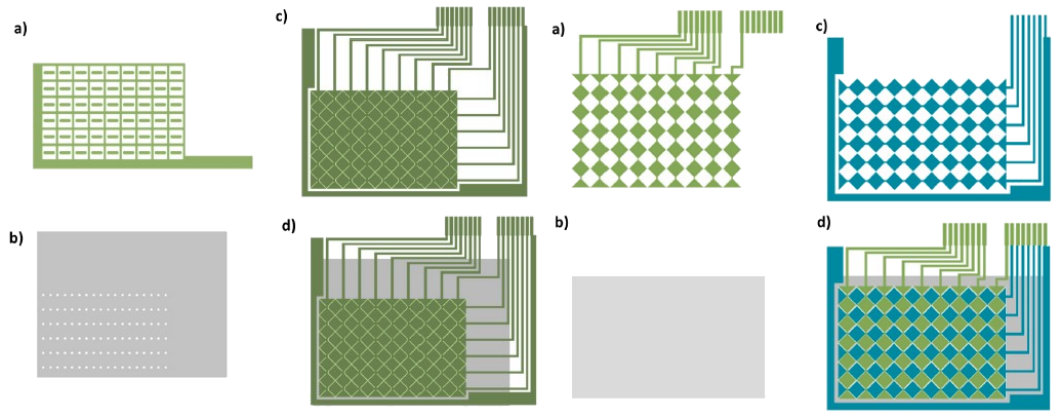

Figure 3.2: One Layers Design (left) and Two Layers Design (right). Left: a) Conductive layer for connection tracks, b) Dielectric with via-holes, c) Vertical and horizontal strips layer, d) Whole sensor (OLD). Right: a) Layer of vertical strips, b) Dielectric, c) Layer of horizontal strips, d) Whole sensor (TLD). Images taken from [Ferri et al., 2017].

between the strips. The first additional layer is a conductive layer for the connection between the strips and the second additional layer is a dielectric layer with via-holes of 1.6 mm diameter. All three layers together result in the *One Layer Design (OLD)*. The second design is the *Two Layers Design (TLD)* (Figure 3.2 right), where one layer contains the vertical strips and the other layer contains the horizontal strips. A dielectric layer is used between these two layers. Both designs were connected to a capacitive touch controller (MTCH6102[2]). The capacitive touch controller has a graphical user interface (GUI) that can directly check gestures such as single click, double click and click and hold. Both designs were able to detect all possible gestures, even when the textile is bent. They found that a dielectric thinner than 10 µm led to insulation problems. Also, the *TLD* had a higher capacitance than the *OLD*, but this observation did not affect the correct functionality of the two sensor designs.

In the first design, all the strips are on one layer. The second design uses two layers.

Both designs were capable of recognizing gestures.

Narakathu et al. [2012] created a fully printed flexible capacitive pressure sensor using screen and gravure printing (Figure 3.3 a) and b)). All components of the sensor are

[2] `https://ww1.microchip.com/downloads/en/DeviceDoc/40001750A.pdf` (Accessed: 23.11.2021)

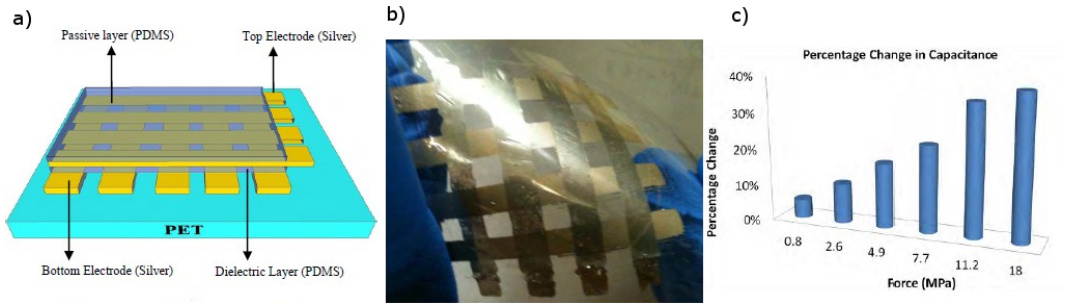

Figure 3.3: a) Schematic design of the sensor, b) Fully printed sensor without PET, c) Percentage change of the capacitance while increasing the force on the sensor. PDMS = Polydimethylsiloxane. Images adapted from [Narakathu et al., 2012].

Narakathu et al. has produced a fully printed sensor in a rectilinear 4 x 4 design.

printed on a flexible PET substrate. First, four strips are printed on the PET substrate using gravure printing, followed by a 4 cm x 4 cm silicone layer as the dielectric using screen printing. Then on a second layer, again four strips are printed at a 90-degree angle to the first strips, creating a 4 x 4 sensor matrix. The strips themselves have a size of 4 cm x 0.5 cm with 0.5 cm spacing between each. A passive layer of silicone completes the sensor. The entire sensor has a thickness of about 200 µm. After fabrication, the fabricated sensor was tested in a force gauge. The sensor was connected to a device for measuring capacitance using wires glued to the strips with silver conductive paste. Different pressure levels were applied to the sensor using the force gauge. The capacitance at different pressure levels is shown in Figure 3.3 c). The sensor was capable of measuring pressure from 800 kPa up to 18 MPa[3]. The durability of the sensor was also tested. The capacitance of the sensor was 26 pF without any pressure. After applying various pressures to the sensor for 5 minutes, the capacitance always returned to the value of 26 pF at rest.

The sensor had different capacities at different pressure values and good durability.

[3]Pa = Pascal. Unit for pressure measurment

3.2 Machine Learning Concepts for Capacitive Sensing

Machine learning algorithms can classify data from capacitive sensors. Various machine learning concepts can be used for this purpose. With the learned classifications, a smart textile can enable interactive applications.

The approach taken by Goertz [2020] was to implement a classification for different types of exercises on a capacitive sensing sports mat. The goal was to enable monitoring of the patient's exercises and to use this information for analysis. The data can be analyzed by a physical therapist to improve the patient's execution of the exercises. The sports mat is equipped with three arrays of four capacitive sensor plates with a size of 5 cm x 28 cm per sensor plate. The sensor plates are connected to a capacitance-to-digital converter (FDC2214[4]) via shielded copper wires of equal length. Multiplexers (TS5A3359[5]) are used to connect all capacitance-to-digital converters to the microcontroller (Adafruit Feather M4 Express[6]). Various techniques are used to avoid and reduce interference, e.g. from other wires, the sensor plates or noise from outside. A conductive sheath covers the wires and a second conductive plate directly under the sensor plates, which is charged with the same voltage as the sensor plates, serves as an active shield against interference. Due to unforeseen circumstances, it was not possible to collect data from the capacitive sensing gym mat described above. For this reason, Goertz [2020] used an existing dataset from the University of California Irvine [Wijekoon et al., 2019] to train the machine learning models. The data were collected from 30 subjects performing seven different physical therapy exercises (Figure 3.4) and contained data from a capacitive sensor, a depth camera and two accelerometers. Two different models were trained using the capacitive sensor data and the depth cam-

Goertz wanted to monitor the exercises on a capacitive sports mat made of 12 sensor plates.

Conductive shielded wires and active shielding protect against interference.

An existing dataset was used.

[4] https://www.ti.com/document-viewer/FDC2214/datasheet (Accessed: 23.11.2021)

[5] https://www.ti.com/document-viewer/TS5A3359/datasheet (Accessed: 23.11.2021)

[6] https://cdn-learn.adafruit.com/downloads/pdf/adafruit-feather-m4-express-atsamd51.pdf (Accessed: 23.11.2021)

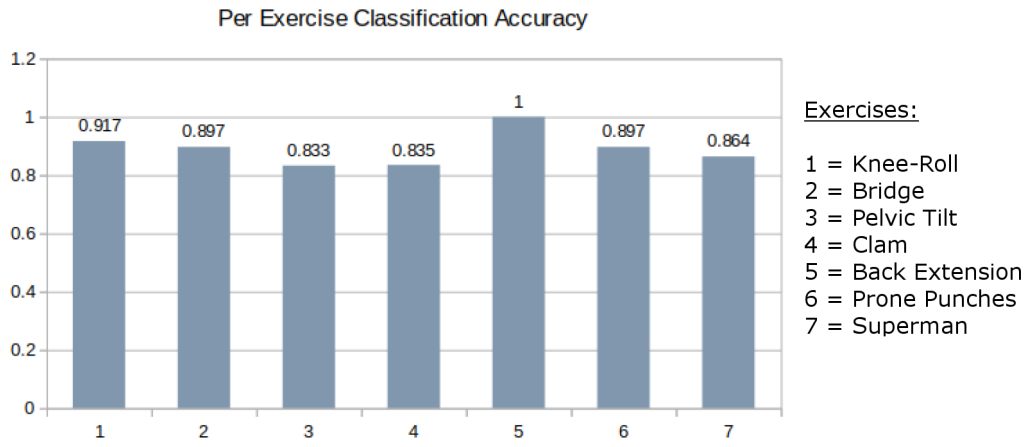

Figure 3.4: Classification accuracy per exercise with the CNN model trained with data from an existing dataset. 1 corresponds to 100 %. Image adapted from [Goertz, 2020].

One model was trained with CNN and one with LSTM. The CNN had a better accuracy.

era data from the existing dataset. Before training, the dataset was split into 80 % training data and 20 % test data. The first model was trained with a CNN. The trained model achieved an overall accuracy of 89.2 %. Figure 3.4 shows the accuracy for each exercise. The classification accuracy ranged from 83.3 % to 100 %. The second model used an *Long Short Term Memory (LSTM)*[7]. The results of the LSTM model were very poor, with an accuracy of 42.5 %, compared to the CNN model. Changes to the model architecture did not improve the results. The work shows that capacitive sensors are capable of providing useful data for differentiating between different exercises with a CNN model.

Capacitivo is an interactive fabric that can distinguish between 20 objects.

In another poject called *Capacitivo*, an interactive fabric using capactive sensors was invented by Wu et al. [2020] (Figure 3.5 top)). The goal was to be able to distinguish between 20 different non-metallic objects used in everyday life (Figure 3.5 bottom). The objects on the interactive fabric can be determined by material and shape, since each material has its own permittivity and therefore affects the measured capacitance differently. Capacitivo has a size of 15.6 cm x 15.6 cm and is made out of conductive fabrics in a 12 x 12 arrangement. The conductive fabrics were cut out in a di-

[7] A type of neural network

amond shape and then glued to a cotton substrate with an iron. The diamonds of the rows are connected with a conductive line and the columns are connected from the back with sewn conductive threads. On the back of the sensor a grounded shielding layer of conductive knit is used to prevent short-circuiting of the column electrodes. Background noise is removed by a filter formed from the average of the last five sensor data. While developing the prototype, Wu et al. tested different diamond sizes and spacing between them and found that a diamond diameter of 7 mm and spacing of 4 mm gave the best signal-to-noise ratio for the prototype. Two capacitive sensing techniques are used: mutual capacitive sensing between the bordering electrodes of a row and column and self-capacitance between the electrodes and the ground layer. The collected dataset consisted of two 12 x 12 arrays, one for each technique. A resonance-based approach is used to measure the capacitance. For this purpose, a resonant circuit[8] is integrated into a custom sensor board, which also contains a microcontroller and eight 4:1 multiplexers (FSUSB74[9]). This has the advantage that the sensors are less sensitive to electromagnetic interference. A random forest[10] was used for the machine learning model. For each object, 50 samples were collected and for the test dataset, 10 participants were invited. After training, Capacitivo achieved an overall accuracy of 94 % for the test data. Of the 20 objects, 18 objects achieved an accuracy greater than 90 %. The model was able to distinguish between similar round objects, but had problems with objects such as credit cards and books (85 % accuracy), due to the low permittivity of these objects. After the main study, Wu et al. conducted two more small studies. The first experiment was for liquid detection, using six different liquids (cold water, hot water, cola, cider, milk and beer) and an empty glass. To reduce complexity, it was decided to train the model with a fixed position on the sensor. The dataset for training and testing contained 20 samples for each type of liquid. 90.71 % of the data was correctly detected after testing, with beer having the lowest

It has a size of 15.6 cm x 15.6 cm in a diamond-shaped 12 x 12 arrangement.

Two resonance-based capacitive sensing techniques are used.

Capacitivo can distinguish between 20 objects with an accuracy of 94 %.

It could also distinguish between six liquids with 96.67 % accuracy after removing one liquid.

[8] Electric circuit consisting of an inductor an a capacitor

[9] https://www.mouser.com/datasheet/2/149/FSUSB74-83766.pdf (Accessed: 23.11.2021)

[10] Machine learning technique used to solve regression and classification problems

Category	Tested Objects
Personal items	AirPods Case (A),Lipstick (B), Discover Credit Card (C), JCPenney Rewards Card (D)
Food items	Kiwis (E), Avocados (F), Grapefruits (G), Cheese Slices in Plastic Wrap (M)
Kitchen items	Table Salt in a Round Package (L), Glass Candle (O), Empty Water Glass (J), Water Glass Filled with Water (K), Empty Bowl, (H) Bowl Filled With Clam Chowder (I)
Office items	Portable External Hard Drive (T), Hand Sanitizer (N), Crouching Figure Statute (S), Book (R), Tabletop Plant (Q), Tabletop Plant After Being Watered (B)

Figure 3.5: Top: The interactive fabric Capacitivo made of conductive fabrics. Bottom: 20 different non-metallic everyday objects were used to test the interactive fabric. Images adapted from [Wu et al., 2020].

Capacitivo was also tested in a bag format and achieved 70 % accuracy.

accuracy (65 %). After removing the beer, the overall accuracy increased to 96.67 %. For the second small study, a bag was created. The model was trained with eight objects. After testing, the bag achieved an overall accuracy of 70 %. Compared to the first design, the accuracy was 25 % lower. According to Wu et al., this could be due to the fact that complete contact with the object was not guaranteed in the bag. The interactive fabric can be used in many real-world applications. For example, the interactive fabric can be integrated into a tablecloth and notifies the user if something important is left on the tabletop when leaving the house.

It can be used as an interactive tablecloth.

Fahr Jr investigated different machine learning models for gesture recognition on a sensor array.

The next work presented does not deal with a textile, but nevertheless capacitive sensors are used. Fahr Jr [2020] reviewed several machine learning models for gesture recognition on a low-cost capacitive sensor array. The capacitive sensor board consists of 64 capacitive sensors (Figure 3.6),

which is connected to a microcontroller (MSP430[11]). Different model architectures and learning strategies were tested using TensorFlow. To train the machine learning models, data from different gestures were collected. Two datasets were created, one containing simple gestures such as finger swipes from left to right, up to down, or in the opposite direction in each case. The second dataset contains more complex gestures such as taps, rubs, diagonal swipes, circles and two-finger V-shapes. The simple gesture dataset contains 400 gesture data and the more complex gesture dataset contains 1200 gesture data. For training, the dataset were divided into frames, with each frame representing the data for one gesture. All frames without a gesture were removed. Thus, the dataset finally contains only data with recorded gestures. The datasets were divided into training, validation, and testing datasets. Different model architectures were implemented, such as a basic neural network, a LSTM, and a *2D convolutional LSTM*[12]. During training, parameters like the number of hidden layers, the number of nodes per hidden layer, the kernel size, and the batch size were changed to compare the loss and the accuracy of the test data. At the beginning each model was tested with the simple gesture dataset to validate the function of the models. The model trained with the basic neural network and the 2D convolutional LSTM used the activation function `relu` and the LSTM model used the activation function `hyperbolic tangent`[13]. The 2D convolutional LSTM used a kernel size of 4 x 4. All models achieved a 100 % test accuracy and a test loss between 0.001 and 0.008. After training with the simple gesture, the three models were trained with the more complex gestures. It was found that the 2D convolutional LSTM model had the highest test accuracy of 99.33 %. Changing the kernel size did not have much effect. A 4 x 4 kernel had an accuracy of 99.67 % compared to 3 x 3 and 2 x 2 kernels with an accuracy of 99.33 %. For the number of nodes in the hidden layers, 25, 50, 75 and 100 nodes were tested. Again, these changes had minimal impact on accuracy. The basic neural network model had the best accuracy with 75 nodes and the LSTM model

One dataset contains simple gestures, the other more complex gestures.

Three different models with different parameters were tested.

Each model was tested with the simple gestures and all models achieved a 100 % test accuracy.

For the more complex gestures, the 2D convolutional LSTM model achieved the highest accuracy of 99.33 %.

[11] `https://www.ti.com/lit/ds/symlink/msp430g2553.pdf` (Accessed: 23.11.21)

[12] LSTM using convolutional layers

[13] Maps the input into a output in a range from -1 to 1

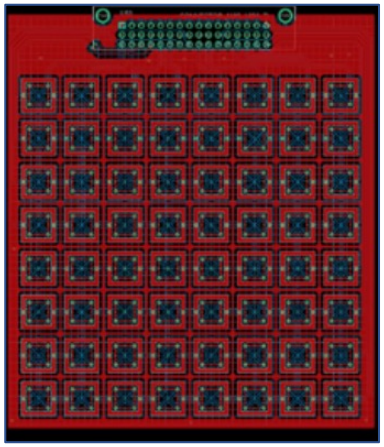

Figure 3.6: Low-cost capacitive sensor pad. 64 capacitive sensors arranged in a 8 x 8 array. Image taken from [Wu et al., 2020].

CNN and LSTM can reliable classify between different gestures.

with 50 to 100 nodes. The 2D convolutional LSTM model achieved the best accuracy with 75 to 100 nodes. Also, five different activation functions were tested: `linear`, `hyperbolic tangent`, `sigmoid`[14] and `relu`. Changing the activation function also had little effect on the training accuracy and the test accuracy. The investigation shows that machine learning can be used for reliable classification between different gestures. The final test accuracy and final test loss results also show that CNN and LSTM were effective neural networks. The functionality was not tested for real-time gesture recognition.

3.3 Comparison

Most of the capacitive sensors presented in Chapter 3 "State of the Art" have a small size. With the prototype of this thesis, we want to find out the capabilities of a larger sensor matrix. In addition, most work focuses more on the fabrication, durability and capacitive behavior of the prototype itself. We want to focus more on working with the data coming from the prototype and how it can be used

[14] Converts the input to an output between 0 and 1

for machine learning and real-time classification. Table 3.1 provides a brief overview of all the work presented.

Literature	**Strips Design**	**Size**	**Used ML?**	**Real-Time Classification?**
Vu and Kim [2020]	- Straight shape - Diamond shape	Approx. 10 cm^2	No	No
Ferri et al. [2017]	- Diamond shape	Approx. 8 cm^2	No	Yes, with MTCH6102 GUI utility
Narakathu et al. [2012]	- Straight shape	Approx. 4 cm^2	No	No
Goertz [2020]	- Single plates	12 plates, each 5 cm x 28 cm	Yes	No
Wu et al. [2020]	- Diamond shape	15.6 cm^2	Yes	Yes
Fahr Jr [2020]	- No Textil - Sensor Board	10.16 cm^2	Yes	No
Thesis requirements	- Straight shape	2 m x 1 m	Yes	Yes

Table 3.1: Brief overview of all work presented in Chapter 3 “State of the Art” and the requirements for this work.

Chapter 4

Methodology

In this chapter we briefly present the work steps of the bachelor thesis. A detailed description of each step follows after this chapter.

In the first step of this thesis, we connect the prototype to a microcontroller to realize the capacitive pressure sensor matrix and to send its values to the computer. Subsequently, the data from the prototype needs to be prepared and processed in order to use the obtained data for training a machine learning model. To prepare and process the data, we use a system with two cameras to automatically label the data with *hand* or *nohand* accordingly. After these steps, we start collecting and labeling the data to create datasets for training a model.

The prototype data is prepared and processed to train some models.

Once a model is trained, it is tested with a test dataset that is split from the entire collected dataset before training. Then the model is tested in real time to see if it is able to detect a hand on the sensor matrix.

A model is tested with a test set and in real time.

We start with a simple dataset (only one hand location) and expand the dataset with another location after the real time recognition worked with the previously simpler trained model. With each extended dataset we test the functions of the prototype. After each extension of the dataset, again a new model is trained with it, tested with the test dataset, subsequently the performance is checked in real time.

Start with a simple dataset and then increase the complexity.

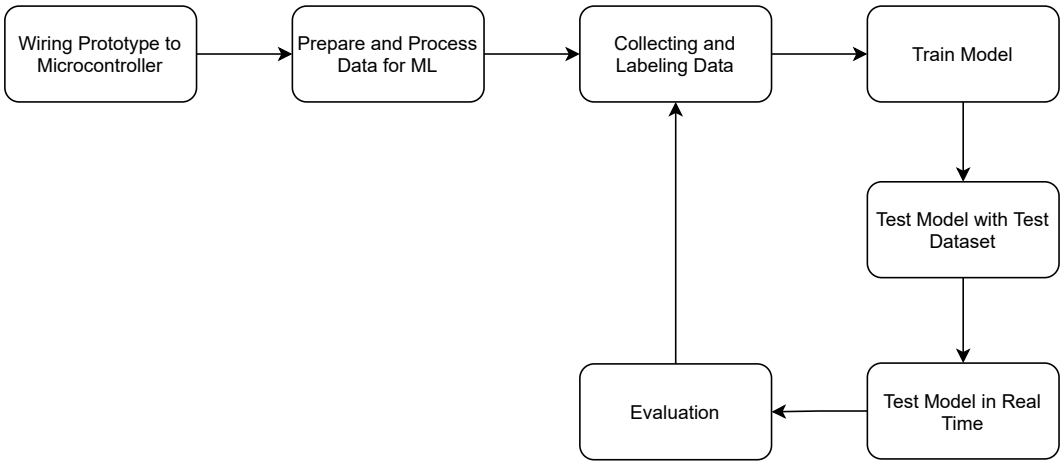

Figure 4.1: Methodology visualization of each step of the process in this thesis.

Chapter 5

Capacitive Pressure Sensor Matrix Prototype

In this chapter, the prototype and its connection to the microcontroller are presented. Chapter 5.1 "Structure of the Prototype" describes the structure of every single layer of the prototype. The prototype is connected to a microcontroller to build up the capacitive pressure sensor matrix. The hardware used, the wiring and the measurement of a capacitive value are shown in Chapter 5.2 "Wiring to the Microcontroller".

5.1 Structure of the Prototype

The prototype for the capacitive pressure sensor matrix is made of three layers of polyvinyl chloride (PVC) with a size of 2 m x 1 m (Figure 5.1). The first two layers are thin PVC foils and the third layer on top is a 4 mm thick sports mat. Self-adhesive aluminum strips are used to realize the capacitive sensor matrix. These strips have a width of 1 cm and a distance of about 2 mm between each strip. Since there was no printer available at the time that could print the strips at this large size, self-adhesive aluminum strips are used for this prototype. In the first version of the prototype, 60 horizontal self-adhesive aluminum strips were

The prototype consists of three layers with a size of 2 m x 1 m. Aluminum strips are used.

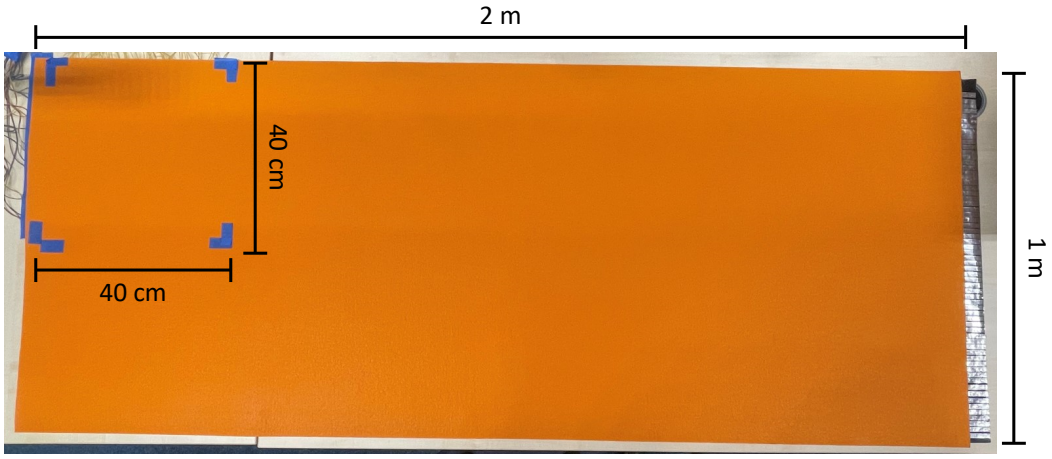

Figure 5.1: Prototype for the capacitive pressure sensor matrix made of three layers of PVC with horizontal and vertical aluminum strips. The prototype has a size of 2 m x 1 m. The blue marked area (40 cm x 40 cm) in the upper left corner shows the area that is used and wired to the microcontroller.

In the first version, aluminum strips were on the first and third layer.

The first version did not work properly, due to damaged strips.

In the second version, the strips are on the first and second PVC foil layer.

glued to the first PVC foil layer. The second layer also was a PVC foil and functioned as the dielectric for the capacitive sensor matrix. On the third layer, 200 aluminum strips were glued vertically on the bottom side of the sports mat (Figure 5.2 a)-d)). These strips were cut out by hand from aluminum foil, which was glued on double-sided adhesive tape. On this layer, the strips had a width of 5 mm and a distance of about 5 mm between them. During development, we noticed that some strips on the top sports mat layer were damaged and therefore not fully conductive. To solve this problem, a new layer was made for the vertical strips using the second PVC foil layer. Due to time constraints and because only a small part of the prototype is needed, 32 self-adhesive aluminum strips are glued vertically on it. These strips have the same dimensions as those of the first layer. Consequently, the second version of the prototype is almost identical to the first version, except that the vertical strips moved from the third layer to the second layer and the number of vertical strips is reduced from 200 to 32. This also means that the sports mat (third layer) in the second version is without any strips (Figure 5.2 e)-g)). The final matrix contains 60 horizontal and 32 vertical lines.

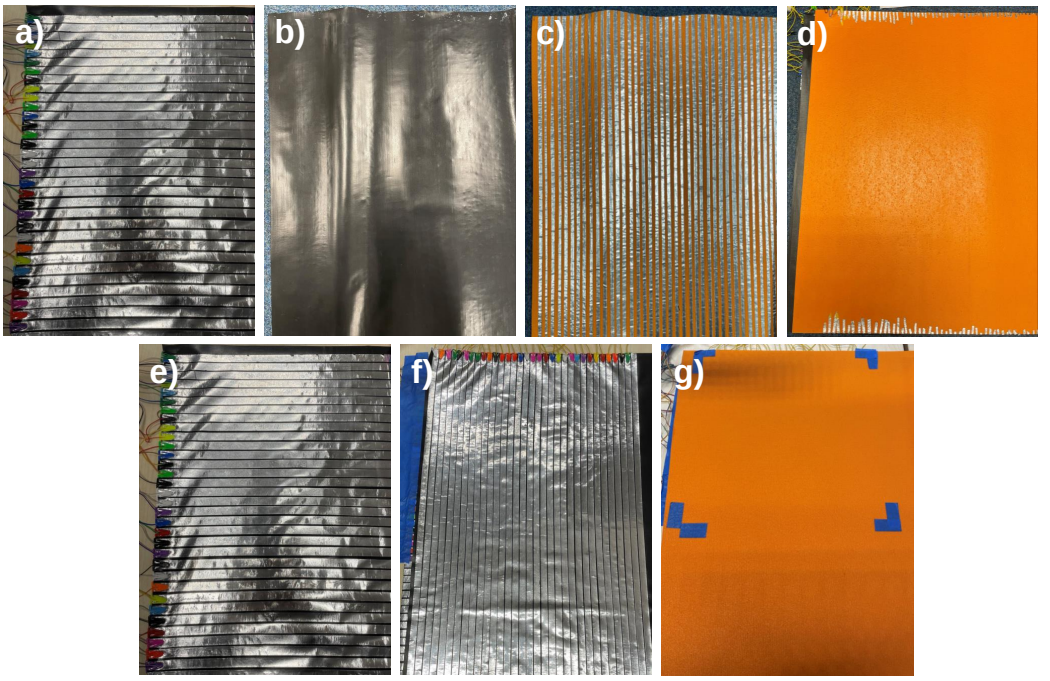

Figure 5.2: Each layer of the first (a)-d)) and second version (e)-g)). a) First layer: Thin PVC foil with 60 horizontal aluminum strips, b) Second layer: Thin PVC foil, works as the dielectric, c) Third layer: Sports mats with 200 vertical aluminum strips on the back, d) Top view of the first version (sports mat), e) First layer: Thin PVC foil with 60 horizontal aluminum strips, f) Second layer: Thin PVC foil with 32 vertical aluminum strips, g) Third layer: Sports mat with blue markings indicating the working area.

A capacitive value can be read at each intersection of a horizontal and vertical strip.

The first layer with the horizontal strips and the second layer with the vertical strips on it forms the capacitive pressure sensor matrix. These enable the measurement of the capacitive value at each intersection of the horizontal and vertical strips. Thus, the prototype has three acting layers: The first PVC foil layer with horizontal electrodes, the second PVC foil layer with vertical electrodes and the PVC foil of the second layer used as the dielectric.

Since we I want to implement a hand detection on this prototype, only a comparably small part of the whole prototype is needed. Therefore, only 32 strips from the first and 32 strips from the second layer are used for this work. This

In this work, only an area of around 40 cm x 40 cm is used.

32 x 32 array provides 1024 capacitive values from an area with a size of around 40 cm x 40 cm. The sensor matrix area is marked with the blue corners in the top left in Figure 5.1.

5.2 Wiring to the Microcontroller

From the prototype, 32 strips of each layer are wired to an Arduino Due.

An Arduino Due[1] microcontroller is used to realize the capacitive pressure sensor matrix and to transmit the capacitive values to a computer. The microcontroller has 12 analog input pins and 54 digital pins. As mentioned in Chapter 5.1 "Structure of the Prototype", we use 32 horizontal and 32 vertical aluminum strips of the prototype. A capacitive value can be read between a digital and an analog pin. The 32 horizontal aluminum strips are wired to the digital pins (D22 - D53) and the 32 vertical aluminum strips are wired to the analog pins of the microcontroller. Since the microcontroller only has 12 analog inputs, two 16:1 multiplexers (CD74HC4067[2]) are used to increase the number of analog inputs. The multiplexer can switch through its 16 channels and pass each incoming measured value to the microcontroller's analog pins (A0 -A1).

We use the digital and analog pins and increased the number of analog pins with multiplexers.

At the beginning, we used the capacitive touch sensor MPR121[3], but found that this sensor does not allow for parallel plate capacitors and is thus unsuitable for implementing the desired sensor matrix.

The capacitance can be read from the intersections of two strips.

The wires are attached to the aluminum strips with plastic paper clips. The complete wiring is shown in Figure 5.3. With this wiring, it is possible to measure the capacitance at the intersections of the horizontal and vertical strips using the stray capacitance of the microcontroller.

[1] https://docs.arduino.cc/hardware/due (Accessed: 24.11.21)

[2] https://www.ti.com/lit/ds/symlink/cd74hc4067.pdf (Accessed: 24.11.21)

[3] https://www.sparkfun.com/datasheets/Components/MPR121.pdf (Accessed: 26.11.21)

For these measurements, we use an Arduino library called CapacitorLite[4]. CapacitorLite allows us to measure capacitive values in the range from 0.2 pF up to 655 pF. At the beginning of the measurement, both strips are discharged and the digital pins are at 0 V. After the digital pins are raised to 5 V, the signal flows through the horizontal strips, and the voltage on the vertical strips connected to the analog pins reach its final value, which can be read off. For example, if 5 V is sent through digital pin 22 and the received voltage is read at pin C0 of the first multiplexer, this value would be the measured capacitive value of the cell in the upper left corner of the sensor matrix. The output is converted to a digital value and is used to calculate the capacitance. The output is an absolute capacitive value in $pF * 100$.

A 5 V signal is sent across the digital pins and the final signal can be read off the analog pins.

The capacitance C is measured with the following equation 5.1:

$$C = \frac{V_{\text{analog output}} \cdot \text{stray capacitance}}{V_{\text{digital input}} - V_{\text{analog output}}} \tag{5.1}$$

where $V_{\text{digital input}}$ is 5 V and $V_{\text{analog output}}$ varies from 0 V to 5 V depending on what is measured at the analog pins. The stray capacitance depends on the microcontroller used.

[4] `https://github.com/codewrite/arduino-capacitor` (Accessed: 26.11.21)

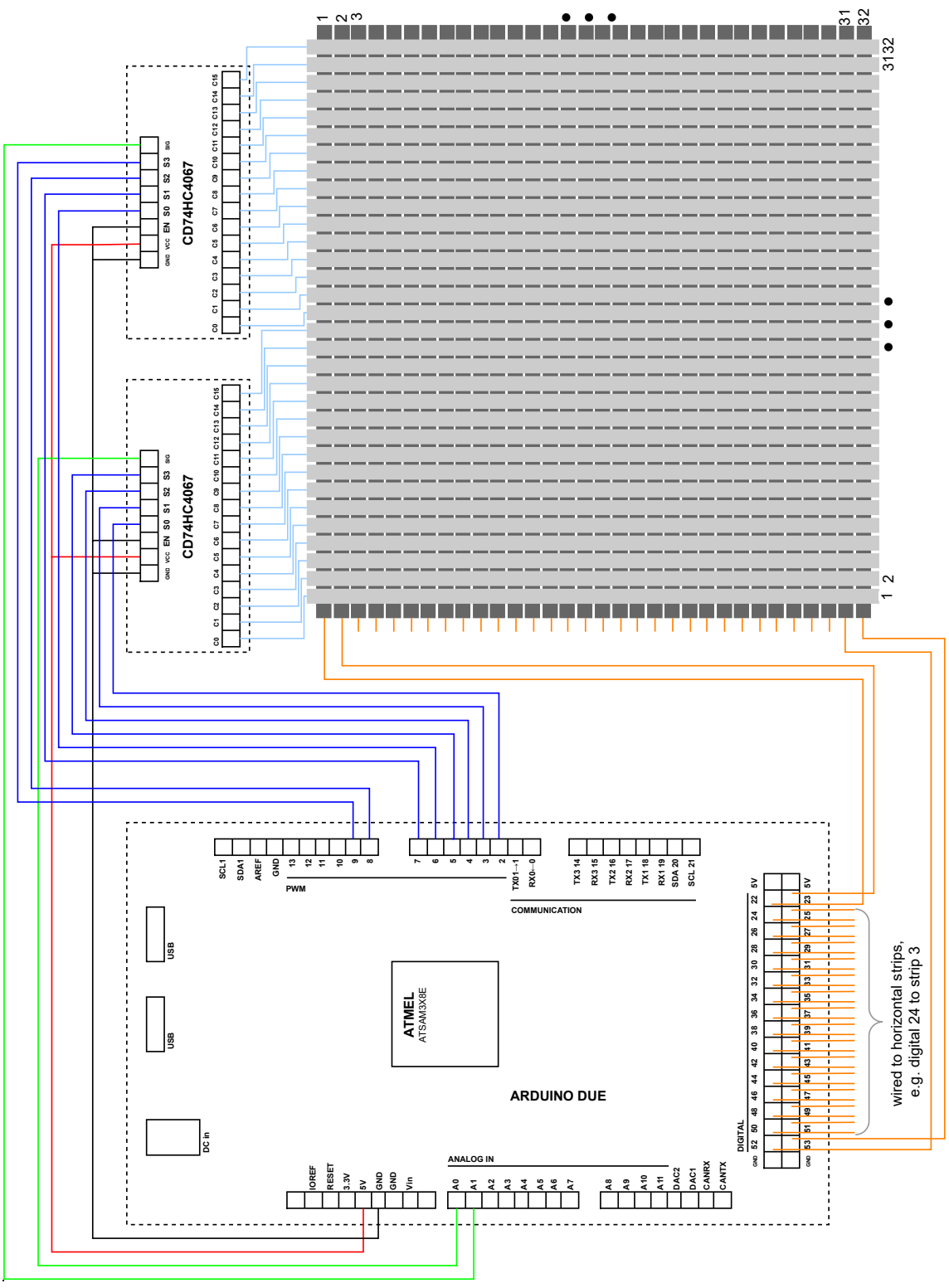

Figure 5.3: Wiring of the prototype with the Arduino Due. The horizontal strips are wired to digital pins 22-53. The vertical strips are wired to two multiplexers (CD74HC4067) which pass the read value to analog pins A0 and A1. A capacitance can be read at each intersection of a horizontal and a vertical strip. A 5 V signal is sent across the digital pins and the final signal can be read at the analog pins.

Chapter 6

Machine Learning Models for Hand Detection

In this chapter, we focus on the machine learning techniques employed. An automatic labeling system using two cameras is described in Chapter 6.1 “Data Labeling and Data Collection”. With their help, labeled datasets for the training of a model can be created. The architecture used for training the model is shown in Chapter 6.2 “Model Architecture”. At last, in Chapter 6.3 “Training of the Models and Results”, we show the datasets used for training the models and their results. We also review the performance of the models for real time hand detection.

6.1 Data Labeling and Data Collection

The Arduino Due sends the data from the prototype to the computer.

The Arduino Due prepares the capacitive values into an array representing the sensor matrix to send them to a computer. The data is in a 32 x 32 array format, where each cell represents the capacitance of one intersection. Before using the Arduino Due, a Arduino Mega 2560[1] was used,

[1] https://docs.arduino.cc/hardware/mega-2560 (Accessed: 03.12.2021)

which was able to send about four data entries per second to the computer. After changing the microcontroller to the Arduino Due, the data stream could be improved to about eight data entries per second.

To label the data for supervised learning, an automatic labeling system with two cameras was developed to collect large datasets.

As mentioned in Chapter 2.1 "Machine Learning", the machine learning model are trained using supervised learning. This implies that the data needs to be labeled to create a usable dataset for training. The labeling process takes into account whether the hand is on the sensor matrix. If a hand is on the sensor matrix surface, the data is labeled with `hand`, everything else is labeled with `nohand`. To label the data, we developed an automatic labeling system with a webcam and a depth camera (Figure 6.1). The system was designed to capture large datasets without having the need to manually press any button to set a label for the data. Another reason is that the used cables are very sensitive to movements and other objects nearby, which cause interference with the capacitive values. Therefore, we also want to use the system to minimize the movements of another hand around the prototype to reduce this source of interference.

The webcam is used with MediaPipe Hands to determine if the hand is in the area of the sensor matrix.

There are 21 landmarks given for the hand.

The task of the webcam is to determine the x and y coordinates of a hand. These coordinates are used to verify whether a hand is within the area of the sensor matrix. A machine learning solution called MediaPipe Hands[2], developed by Zhang et al. [2020], is used for this task. MediaPipe Hands is able to recognize human hands and displays the hand skeleton on the video image, which consists of 21 coordinates. Compared to other current approaches, this solution requires only a simple webcam and no additional special hardware or powerful desktop environment. The solution is a combination of two machine learning models: The BlazePalm Detector[3] and the hand landmark model. First, the BlazePalm Detector localizes the palm with a bounding box around it. Then, the hand landmark model runs on the bounding box area and provides the landmarks of the hand. The topology of the 21 landmarks is the same

[2] `https://google.github.io/mediapipe/solutions/hands` (Accessed: 10.12.2021)

[3] `https://github.com/vidursatija/BlazePalm` (Accessed: 24.11.2021)

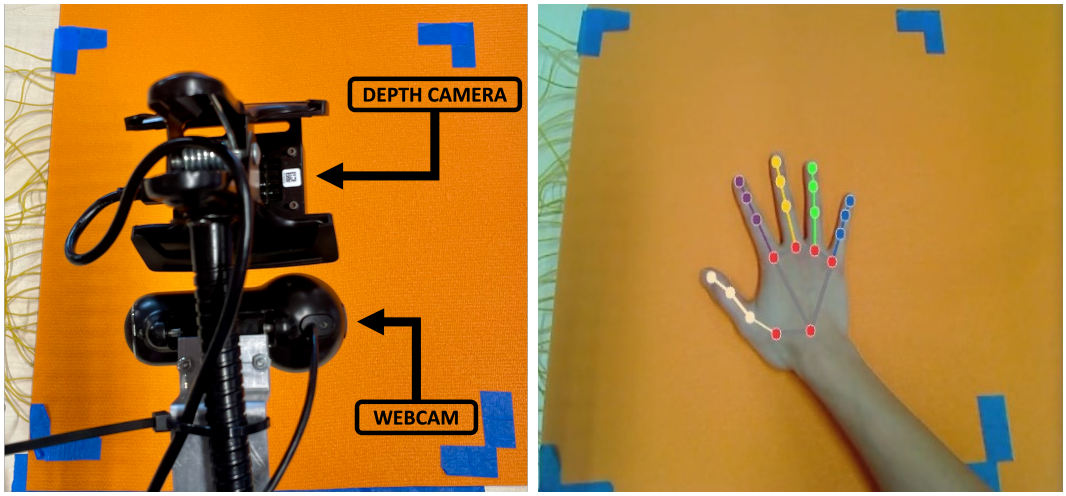

Figure 6.1: Cameras placed above the prototype for the automatic labeling system. A webcam and depth camera is used (left). A hand recognized by the MediaPipe Hands solution. A hand skeleton contains 21 landmarks, where each landmark is a tuple out of the x and y coordinate (right).

as in the work of Simon et al. [2017]. Since the hand landmark model only has to work with the bounding box of the palm detector model, the need for data augmentation is very small. In addition, the palm detection model only needs to run on the first frame or when it can no longer locate a hand. The model outputs not only the landmarks of the hand, but also the probability of the presence of the hand and which hand it is (right or left). The MediaPipe Hands solution achieved a mean square error rate of 13.4%. Our initial goal was to develop a hand detection that could distinguish between `nohand`, `righthand` and `lefthand`, but we found that the resolution of the sensor matrix is not high enough to distinguish between data from a right and left hand. Therefore, we decided to use only the two labels `nohand` and `hand` for this version of the prototype. To label the data, we use the webcam for automatic labeling as follows:

The solution also returns which hand it is.

The resolution of the prototype is too low to use three different labels. Therefore two labels are used.

1. Start the video stream of the webcam.
2. Set the area of the sensor matrix by selecting four corners of the polygon.
3. Check if the 21 hand landmarks are in the set area.

The depth information was still missing.

If it is determined in step 3 that all 21 hand landmarks are in the specified area of step 2, the data is a possible candidate for the label `hand`. At this point, we only had the two-dimensional information about the hand. The next step was to determine whether the hand is on the surface of the sensor matrix.

The TeraBee 3D depth camera is used to obtain the depth information and determine if the hand is on the sensor matrix.

For this purpose, we have to check the depth data of the video image. To obtain the depth data, a depth camera is used in addition to the webcam. We chose the Terabee 3D Cam[4], a 3D Time-of-Flight (ToF) camera that can transmit the depth information in form of an 80 x 60 array to the computer. A ToF camera sends an infrared light to the surface/object, measures the time it takes for the light to travel from the camera to the surface/object and back, and converts the measured time into a distance [Li et al., 2014]. The distance is given in millimeters. Since the depth camera only provides the depth of each pixel, it was not possible to use this camera for the MediaPipe Hands solution as well. The depth camera does not have a actual color image and the resolution of 80 x 60 pixels is too low for MediaPipe Hands to detect a hand. To determine if the hand is on the sensor matrix surface, we use the depth camera as described in the following steps:

1. Start the video stream of the depth camera.
2. Set the area of the sensor matrix by drawing a rectangle with the upper left and lower right corners.
3. Save the depth image of the idle state (no hand in the area of the sensor matrix).

In the specified area, the average of all changes in depth pixels is calculated.

The selection of the area in step two reduces the number of depth pixels that are important for the calculation. After selecting the two corners, the coordinates of the corners have to be converted to the corresponding array entries of the 80 x 60 array coming from the depth camera. For each depth data image, the average of the sum of the changes of each pixel within the selected area is calculated. If the average

[4] `https://terabee.b-cdn.net/wp-content/uploads/2019/03/Terabee_3Dcam80x60_specsheet-2.pdf` (Accessed: 11.12.2021)

is between 17 mm and 30 mm, the data is labeled as `hand`. Some testing showed that the range chosen for the average was best for distinguishing between a hand hovering above and a hand on the prototype. These parameters can be adjusted for later data collection, such as when a dataset is collected for a larger hand.

If the average is within the defined range, the label `hand` is used.

The use of the two cameras makes it possible to distinguish between a hand on a surface and a hand hovering about 5 mm above the surface. A data is labeled as `hand` only if the webcam indicated that all 21 hand landmarks are in the selected area and if the average value calculated with the depth camera is in the specified range. Anything else, such as a hovering hand, no whole hand in the area, or no hand, is labeled as `nohand`.

Only one whole hand on the sensor matrix area is labeled as `hand`.

To process the data for training the machine learning model, we use JSON [Pezoa et al., 2016] to store the labeled data in a dataset. Before the dataset is used for training, the first 30 data entries must be deleted, because the connection to the sensor matrix takes a short time to be established. Each data entry consists of the 32 x 32 array of the sensor matrix and the corresponding label. For example, a data entry is structured as follows:

JSON is used to store the data in a dataset.

```
{"sensormatrix":  32 x 32 array, "label":
"hand"}
```

An example run for labeling and data collection is shown in the Appendix A “Example of a Data Collection Run”.

6.1.1 Digital Low Pass Filter

Observations during development shows that the noise of the capacitive values of the prototype was very strong. To contain this noise, a digital low pass filter is used. This filter is able to attenuate higher frequencies and allows low frequencies to pass through unchanged [Kaiser and Reed, 1977]. A comparison of an unfiltered and filtered capacitive value is shown in Figure 6.2. A capacitive value from the center of the sensor matrix was used as an example for

Since the capacitive values were very noisy, a digital low pass filter is used.

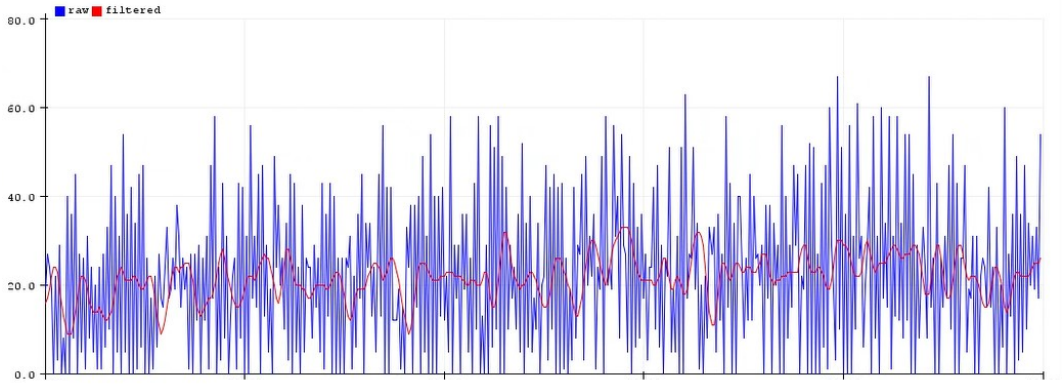

Figure 6.2: Plot of a capacitive value from the center of the sensor matrix in the idle state. The blue graph shows the unfiltered capacitive value and the red graph shows the filtered capacitive value using a digital low pass filter.

The filter can reduce some of the noise of the capacitive values.

the plot. During the plot, the sensor matrix was in the idle state, meaning that no hand was on or near the prototype. It can be seen that without the digital low pass filter, the signal was very noisy and fluctuated between 0 pF and 60 pF (Figure 6.2 blue graph). After applying the digital low pass filter, the noise of the signal was reduced to a range of about 10 pF to 30 pF (Figure 6.2 red graph). The use of the filter allows cleaner capacitive values to be provided for the machine learning model.

6.2 Model Architecture

CNN are used for the model.

A convolutional neural network is used for training the models, as mentioned in Chapter 2.1.1 “Convolutional Neural Networks”. We chose a CNN because the data we receive from the prototype is in a 32 x 32 format, which can be considered as a 32 x 32 pixel image. As mentioned earlier, CNNs are mainly used for image recognition and are therefore suitable for this use case.

The model contains eight layers.

The model consists of a convolutional, a dropout and a max pooling layer. Then the same three layers are added to the model, followed by a flattening layer and a dense layer. The model architecture and the parameters of each layer

are shown here:

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3),
         activation='relu', padding='same',
         input_shape=(32, 32, 1)),
    Dropout(0.2),
    MaxPool2D(pool_size=(2, 2), strides=2),

    Conv2D(filters=64, kernel_size=(3, 3),
         activation='relu', padding='same'),
    Dropout(0.2),
    MaxPool2D(pool_size=(2, 2), strides=2),

    Flatten(),
    Dense(units=2, activation='softmax')])
```

In the first convolutional layer, the input shape is defined. The first two parameters are the shape of the input (32 x 32) and the third parameter is for the number of color channels. Since our data consists of capacitive values and therefore have no color channels, this parameter is set to 1, which is also the standard setting for grayscale images.

The input shape is defined with 32 x 32 and one color channel.

The chosen kernel size of 3 x 3 is generally a very common kernel size [Camgözlü and Kutlu, 2020]. The filter size of 32 and 64 were selected arbitrarily. `Relu` activation is used in both convolutional layers, and zero padding is enabled. A dropout layer is added after each convolutional layer, which ignores randomly selected 20 % of the neurons. Each dropout layer is followed by a max pooling layer to reduce the size of the data for the next layer. The flattening layer converts the data into a one-dimensional input for the dense layer, which is the output layer. This output layer has one neuron for each class, `nohand` and `hand`. In the output layer, the activation function `softmax` is used to determine the probability distribution for the two outputs. Each model is trained for 100 epochs with a batch size of 32. Before training, 20 % of the dataset is split into a validation dataset.

The CNN contains convolutional, dropout, and max pooling layers, followed by a flattening and dense layer.

6.3 Training of the Models and Results

Some models were trained with different hand locations and then tested.

Several models were trained with different datasets. All collected datasets contains data labeled as `hand` or `nohand`. Before training, all datasets were split into 80 % training data and 20 % test data. The model were trained with the architecture presented in Chapter 6.2 "Model Architecture" and the generated training dataset. Then the trained models were tested firstly with the test data and finally in real time. All data was collected using the author's right hand. Each data collection and expansion took about 10 minutes, with half of the time used to collect data from the idle state without any hand input. The other half of the time was spent collecting data of a hand input.

We switched from a complex dataset to a simpler one.

Initially, we collected a dataset that included data from hand inputs that occurred over the entire area of the connected sensor matrix. The model achieved a training and validation accuracy of 100 %. The trained model was able to correctly predict the entire test dataset, but was unable to detect a hand on the sensor matrix in real time. For this reason, we started with a simpler dataset and increased its complexity step by step.

The simple dataset contains data from a hand placed in the middle of the sensor matrix.

For the simple dataset, the hand was always placed in the middle of the sensor matrix during data collection. We decided to train the first model with this simple dataset to check the first functionality of the model and the prototype. The dataset contains 2600 entries (1938 `nohand`, 662 `hand`). Of these, 2079 entries were used for training and 521 entries were used for testing. The training dataset consists of 1549 `nohand` entries and 530 `hand` entries. The test dataset consists of 389 `nohand` entries and 132 `hand` entries. After training, the model achieved a training accuracy of 99.72 % and a validation accuracy of 99.44 %. The training loss (0.0072) and validation loss (0.0188) were also very low. The model was able to correctly predict all test data except for one incorrect prediction where a data labeled with `hand` was predicted as `nohand` (99.80 % accuracy). In real time, the model was able to correctly classify hand inputs in the middle of the sensor matrix most of the time. It rarely got stuck in the `hand` state after the pressure of the hand on the

prototype was removed. Since we tested the prototype in real time without the camera setup, the data could not be labeled during real time prediction. Therefore, it is not possible to provide a real statistical number for the real time prediction performance. Instead, a 4-level evaluation is used to evaluate the real time performance (*not working, sometimes working, mostly working, perfectly working)*.

Test accuracy was 99.80 % and real time prediction worked most of the time.

The second model was trained with data from two hand locations. To do this, we added data from hand inputs in the upper right corner to the dataset of the first model (middle). A total of 804 `nohand` entries and 664 `hand` entries were added. Finally, the second dataset includes a total of 5068 entries. After splitting this dataset into 80 % training and 20 % test data, we had 2992 `nohand` entries and 1061 `hand` entries for training. The test dataset contains 750 `nohand` entries and 265 `hand` entries. After using this dataset for training, the model achieved a high training accuracy of 99.79 % and a high validation accuracy of 99.65 %. Both, the training loss of 0.0048 and the validation loss of 0.0053 were close to zero. The trained model correctly predicted 260 of 265 `hand` entries and 750 of 750 `nohand` entries (99.51 % accuracy). We also tested this model in real time and found that it could recognize both hand inputs, but slightly worse than the first model (middle). When it could not recognize the hand inputs, the real time prediction got stuck in the `hand` state and could not distinguish between the two labels.

We added data from the upper right hand location. The test accuracy was 99.51 % and the real time prediction performed slightly worse.

For the next model, we added another hand location to the second dataset (middle and upper right corner). This time, the data came from hand inputs in the lower right corner. Since the second dataset already contains 3742 `nohand` entries, we added only 629 `hand` entries from the lower right corner to the dataset (5696 entries in total). The training dataset contains 2992 `nohand` entries and 1563 `hand` entries. The test dataset contains 750 `nohand` entries and 391 `hand` entries. The test results of the trained model were only slightly worse than the models before it, with an accuracy of 99.21 % (382/391 `hand`, 750/750 `nohand`). Using this model for real time predictions did not lead to meaningful results. The model predicted a hand the whole time and did not react to any input on the sensor matrix. At this

Data of a third hand location was added (lower right corner). The test accuracy was 99.21 %, but the model did not work in real time.

point, we stopped adding more locations to the data set because we want to concentrate on enhancing the real time performance.

The second dataset was split into two. Model for upper right hand location performed well in real time.

In another experiment, we split the second dataset (middle and upper right) into two datasets for one hand location each and trained a single model for the position in the upper right corner. As before, the dataset was divided into train and test data. This model achieved high accuracy during training (99.98 %) and validation (100 %). The loss on training (0.001) and validation (0.0001) was also low. The test dataset was also predicted 100 % correctly. During real time prediction, the model for the upper right corner was also able to correctly detect a hand most of the time. But as mentioned above, after combining the two datasets, the model with two hand locations was not able to reliably detect different hand locations on the prototype.

The left side of the sensor matrix area was also tested. The trained models for this area did not work in real time.

So far, we have only collected data from the right side of the wired sensor matrix. Therefore, in additional experiments, we collected data from hand inputs on the left side. We collected data from hand inputs in the lower left corner (1817 entries) and in the upper left corner (1946 entries). These were used to train two individual models for each hand location. Similar to the previously trained models, both models achieved a very high training (100 % accuracy) and validation accuracy (100 % accuracy) and low training (both 0.0001) and validation losses (upper left: 0.0001, lower left: 0.00002). They also predicted their test dataset with 100 % accuracy, but in real time, neither model was able to detect a hand. Both models predicted all data as a hand all the time and did not react to any interaction on the prototype.

A summary of all trained models and their results is provided in the Table 6.1. In Appendix B "Model Results Plots", graphs visualize the accuracy and loss of each model during training. TensorBoard[5] was used to display the graphs, which is a visualization toolkit for TensorFlow.

[5] `https://www.tensorflow.org/tensorboard` (Accessed: 18.12.2021)

Hand Locations	Dataset	Accuracy	Loss	Test Results	RT
Everywhere	Training: 1152 NH, 1157 H Test: 289 NH, 288 H Total: 2886	Training: 100 % Val.: 100 %	Training: 0.00003 Val.: 0.000003	289/289 NH 288/288 H 100 % Acc.	- -
Middle	Training: 1549 NH, 530 NH Test: 389 NH, 132 H Total: 2600	Training: 99.72 % Val.: 99.44 %	Training: 0.0072 Val.: 0.0188	389/389 NH 130/131 H 99.8 % Acc.	+
Middle, Upper Right	Training: 2992 NH, 1061 H Test: 750 NH, 265 H Total: 5068	Training: 99.79 % Val.: 99.65 %	Training: 0.0048 Val.: 0.0053	260/265 NH 750/750 H 99.51 % Acc.	-
Middle, Upper Right, Lower Right	Training: 2992 NH, 1563 H Test: 750 NH, 391 H Total: 5696	Training: 99.78 % Val.: 99.23 %	Training: 0.0074 Val.: 0.0227	750/750 NH 382/391 H 99.21 % Acc.	- -
Upper Right	Training: 1443 NH, 515 H Test: 361 NH, 149 H Total: 2468	Training: 99.98 % Val.: 100 %	Training: 0.001 Val.: 0.0001	361/361 NH 149/149 H 100 % Acc.	+
Lower Left	Training: 1016 NH, 437 H Test: 254 NH, 110 H Total: 1817	Training: 100 % Val.: 100 %	Training: 0.00001 Val.: 0.0002	254/254 NH 110/110 H 100 % Acc.	- -
Upper Left	Train: 1135 NH, 421 H Test: 284 NH, 106 H Total: 1946	Train: 100 % Val.: 100 %	Train: 0.00001 Val.: 0.0001	284/284 NH 106/106 H 100 % Acc.	- -

Table 6.1: Comparison of the results of all trained models. *RT* = real time performance, where not working = - -, sometimes working = - , mostly working = + and perfectly working = ++. Acc. = Accuracy, Val. = Validation, *NH* = `nohand` entries and *H* = `hand` entries.

Chapter 7

Discussion

7.1 Challenges

The strip architecture of the prototype results in the capacitive signals not being equal across the entire sensor matrix due to the length of the strips and the difference in length between the horizontal and vertical strips. Training and testing of the models show that the prototype can provide data for a hand detection at one to two locations in real time. For more complex datasets (more than two hand locations), the respective models are no longer able to respond to any interactions on the sensor matrix area. In this case, the model always predicts the data coming from the prototype as a hand.

Models for more than two hand locations do not work and always predict a hand.

One of the main reasons for this is that capacitive sensors are generally very sensitive to all kinds of noises steming from the environment. Because of this noise, the models trained with datasets containing more than two hand locations are not able to distinguish between data from a `hand` and `nohand`. The noise is also the reason that the training and testing accuracy of the models in our case did not match the performance in real time. During the training and testing of the models, the prototype was placed in an open-plan office. In this environment, the sensor matrix was heavily disturbed by people coming and going and by electrical devices being turned on and off. For example,

Capacitors have noisy signals. Because of this noise, it was not possible to get the same values for the idle state every time.

one model worked in real time shortly after training, but stopped working a few hours later. In this setting it was not possible to have the same environmental conditions all the time, and therefore there was not always the same idle state of the capacities.

With the filter, we are able to reduce the noise a bit.

We tried to minimize noise with a digital low pass filter as mentioned in Chapter 6.1.1 “Digital Low Pass Filter”. The digital low pass filter allowed us to train models for one to two hand locations that worked in real time. Without this filter, the models for one and two hand locations did not even work in real time. Generally, we could filter out some of the noise. But it seems that for more than two different hand locations, we reached the natural limit of what can be filtered out with the digital low pass filter.

The prototype does not return to the initial state, due to its material.

One possible reason for the poor real time prediction could be the material of the prototype itself. We noticed that sometimes the layers did not return to its initial state after removing a hand from the prototype. In this case, the layers got stuck together. This problem made it difficult to collect clean data for the model. The material of the aluminum strips was also not ideal, as they were very thin and kinks easily forms in the strips leading to increased interference.

Used cables are not shielded against interferences.

During the training and testing of the models, it was also noticed that models trained with data from a hand on the left side did not work. This is probably due to the fact that half of the cables are attached to the left side of the prototype and every time the sensor matrix is pressed, the cables are also touched or moved, causing too many interfering signals.

Resolution of the sensor matrix is not high enough to differ between left and right hand.

At the beginning of the work, our goal was to develop a hand detection that could distinguish between left and right hand. However, it turned out that the data of the right and left hand on the prototype do not differ enough to distinguish between them. This implies that the resolution of the sensor matrix is not high enough for this requirement.

Interferences in the microcontroller.

We also examined the microcontroller and found that the pins are not 100 % isolated from each other and that there are circuits between the strips within a layer itself. This lead

to interference signals not only in the prototype, but also in the microcontroller.

7.2 Proposals for Solutions

The problems and challenges described above in Chapter 7.1 "Challenges" could be solved or minimized, for example, by improvements to the prototype and the hardware used. Some possible solutions are presented below.

7.2.1 Improvements to the Prototype

The used materials could be upgraded in such a way that the prototype returns more easily to its original state after pressure was applied.

The prototype can be improved for the next production. The material currently used for the layers could be replaced with a material that completely returns to its original state after pressure was applied to the prototype. A foam layer could also be used to provide a fast recovery rate for the prototype. Furthermore, a laminating layer could be applied to the layers themselves to isolate the strips from each other. Another addition could be a grounding shield as in the prototype of Ferri et al.

The strip architecture can be improved and the resolution of the sensor matrix can be increased.

In addition, the architecture and arrangement of the strips can be improved. The strips could be printed instead of glued aluminum strips. Then the strips would not deform as easily under pressure and no small kinks would be created. The design of the strips could be changed to a diamond-shaped design like in Vu and Kim's work. This could result in more stable capacitive signals for better localization of pressure. Another possible way for improvement is to aim for higher resolution, e.g. by reducing the width of the strips while increasing the number of strips in each layer. With a higher resolution more complex data and thus better results should be possible, e.g. to distinguish better between left and right hands.

A further step could also be to divide the sensor matrix into smaller rectangular areas (Figure 7.1). During the development of the wiring, the implementation was tested with a

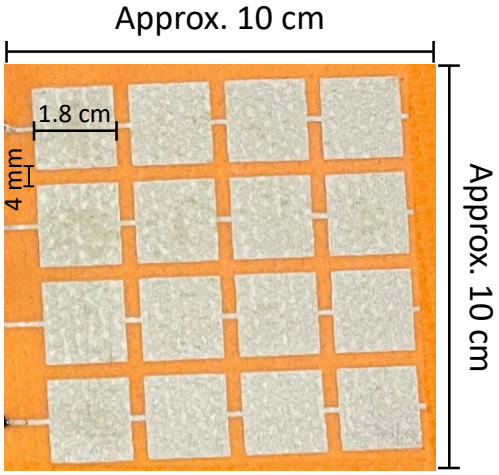

Figure 7.1: Small 4 x 4 sensor matrix (approx. 10 cm x 10 cm) made of printed square-shaped strips. Each square had a diameter of 1.8 cm and a distance of 4 mm between each strip.

The sensor matrix could be divided into small subsections.

smaller sensor matrix, since the larger prototype used was not ready at that time. The smaller sensor matrix had a strip arrangement of 4 x 4 on both sides and a size of about 10 cm x 10 cm. The strips were printed in a square design, with each square having a diameter of 1.8 cm and a spacing of 4 mm between each strip. This small prototype was able to provide stable capacitive values with very low noise. A possible place of future research could be to test up to which matrix size the sensors can actually provide stable values. This could work as input for a modular system in which these small sensor matrices build up to one combined larger one.

7.2.2 Improvements to the Hardware Used

The microcontroller can be changed for better isolation and faster data stream.

Besides the prototype, the hardware around the sensor matrix could be an object of improvement. The Arduino Due could be replaced with a better microcontroller or a custom microcontroller that has only the components needed and better isolated pins to avoid interference within the microcontroller. This change could also improve the speed of the

data stream for collecting the data and testing of the model. As mentioned in Chapter 6.1 "Data Labeling and Data Collection", we have already switched from the Arduino Mega to an Arduino Due to increase the speed of the data stream, but this can still be improved. Also, shielded cables could be used to minimize interference in the cables themselves.

Shielded cables could be used to reduce interference.

In this work, we have realized the capacitive pressure sensor matrix with a technique that works with direct current. In the future, it could be changed to a technique that uses alternating current. The use of alternating current (e.g. a sinus signal) could lead to more stable and cleaner capacitive signals.

We could change from direct current to alternating current.

If more stable capacitive values are obtained, it could be considered to work with the rates of change of the values (relative values) instead of the absolute capacitive values. This could lead to more unique data for training a model.

Relative values could be use.

Table 7.1 provides a brief overview of the proposed solutions presented in this chapter for the challenges/problems discussed in Chapter 7.1 "Challenges".

Possible Solutions	**Goals**
Changing the material of the layers of the prototype	Complete return to the initial state after exposure to pressure
Use of a foam layer	Faster recovery time
Laminating layer on top of the strips	Isolates the strips from each other
Use of a grounding shield	Minimizes interference of the enviroment
Use printed strips	Avoid deformations of the strips, such as kinks
Use diamond-shaped strips	Better localization of pressure
Higher resolution of the sensor matrix	More complex data collection
Divide large sensor matrix in subsections	More stable capacitive values and less interference
Changing the microcontroller	Better isolated pins and faster data stream
Using shielded cables	Minimize interference in the cables
Using alternating current based technique for capacitive sensing	More stable capacitive signals
Using relative values	More unique data for training a model

Table 7.1: Overview of the possible solutions presented in Chapter 7.2 "Proposals for Solutions" and their goals.

Chapter 8

Summary and Future Work

8.1 Summary and Contributions

In this bachelor thesis, we first wired the prototype to a microcontroller to send the capacitive values of the sensor matrix to a computer. Then we developed a labeling system with two cameras to automatically label the data with the correct label. This labeling system can also be used for other pressure sensors that have been developed as part of the project of the ITA or will be developed in the future. After collecting and labeling the data, we constructed a machine learning model architecture using convolutional neural networks. Using this architecture, we trained several models with different datasets and evaluated them with a test set and in real time.

The prototype was wired to the microcontroller. An automatic labeling system was developed. Several models were trained and tested.

Our investigation with the current prototype show that it can provide data for a hand detection of one to two hand locations on the sensor matrix. For more locations, the trained model could not work correctly because it was unable to distinguish between all the data due to noise. The tests also show that for real time prediction to work, the interference must be kept as low as possible. We identify inferferences as the main obstacle that must be overcome.

We achieved a hand detection for one and two locations on the sensor matrix. Interference must be reduced.

Thus, in our use case, the test accuracy did not reflect the performance in real time.

8.2 Future Work

For future work, we plan to improve the prototype and its surrounding hardware to minimize noise. We can use the proposed solutions from Chapter 7.2 “Proposals for Solutions” to obtain more cleaner capacitive values. Then we can collect data again to train a model and to test the capabilities of the new prototype afterwards. These enhancements could allow for increased complexity of the model. The complexity of the model could be increased by a higher number of labels. For example, we could try to distinguish between a right and a left hand or even between the orientation of the hand. However, further investigations could also reveal the limitations of our prototype.

Appendix A

Example of a Data Collection Run

To start the data stream from the prototype to collect and label the data, do the following steps:

1. Start the program.
2. The RGB camera stream starts to run (Figure A.1 top left).
3. Select the four corners of the sensor matrix area. Start with the upper left corner and then select the next corners clockwise. These four corners form a polygon.
4. Press `P` on the keyboard to draw the polygon and `Q` to set the drawn polygon as the sensor matrix area (Figure A.1 top center).
5. The depth camera stream starts to run (Figure A.1 bottom left).
6. Select the upper left and lower right corners of the sensor matrix area to set the area for the depth camera (Figure A.1 bottom center).

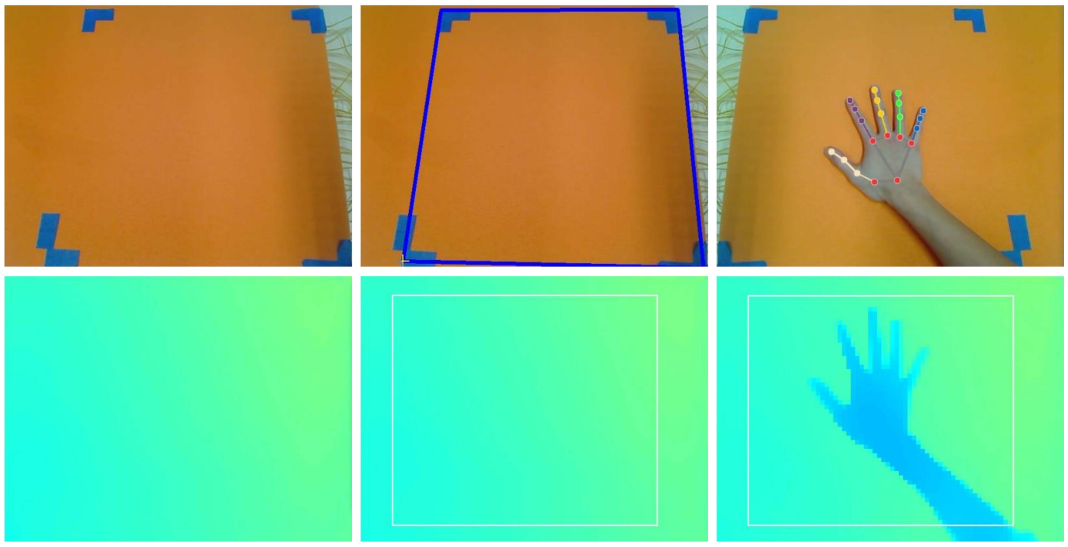

Figure A.1: Top left: Start of RGB camera stream. Top center: Drawn sensor matrix area to determine if hand is in the area. Top right: Recognition of a hand with 21 hand landmarks. Bottom left: Start of depth camera stream. Bottom center: Drawn sensor matrix area to determine if hand is on the sensor matrix. Bottom right: Floating hand above the sensor matrix, seen through the depth camera.

Now the data collection is running. If a hand is on the sensor matrix and is completely in the defined area from step 3, the corresponding data is labeled as `hand`, everything else is labeled as `nohand`. The data collection can be stopped with `ESC` or `CTRL+C`. Afterwards, all data of the sensor matrix (32 x 32 array) and the associated labels are stored in a JSON file. After that, the JSON file can be used as the dataset for training a machine learning model.

Appendix B

Model Results Plots

This chapter shows the graphs of the metrics of each trained model shown in 6.3 “Training of the Models and Results”. In each figure, the top graph shows the training accuracy (orange line) and validation accuracy (blue line). The bottom graph in each figure shows the training loss (orange line) and the validation loss (blue line). All graphs were created using TensorBoard and smoothed at a rate of 0.6.

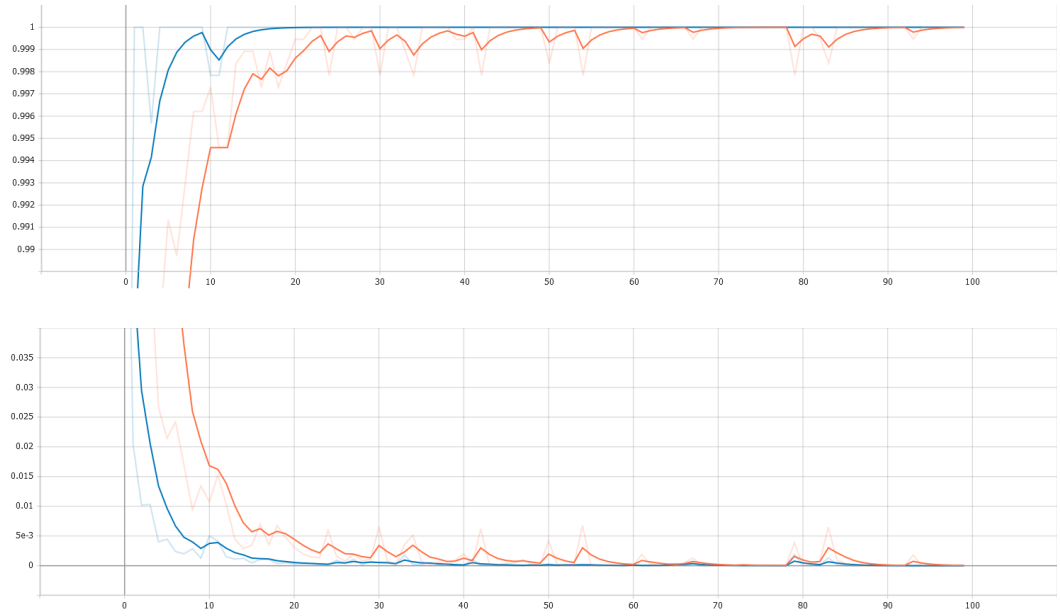

Figure B.1: Plot of model results trained on a dataset containing data from a hand placed everywhere on the sensor matrix. Final training acc.: 100 %, final val. acc.: 100 %, final training loss: 0.00003, final val. loss: 0.00003

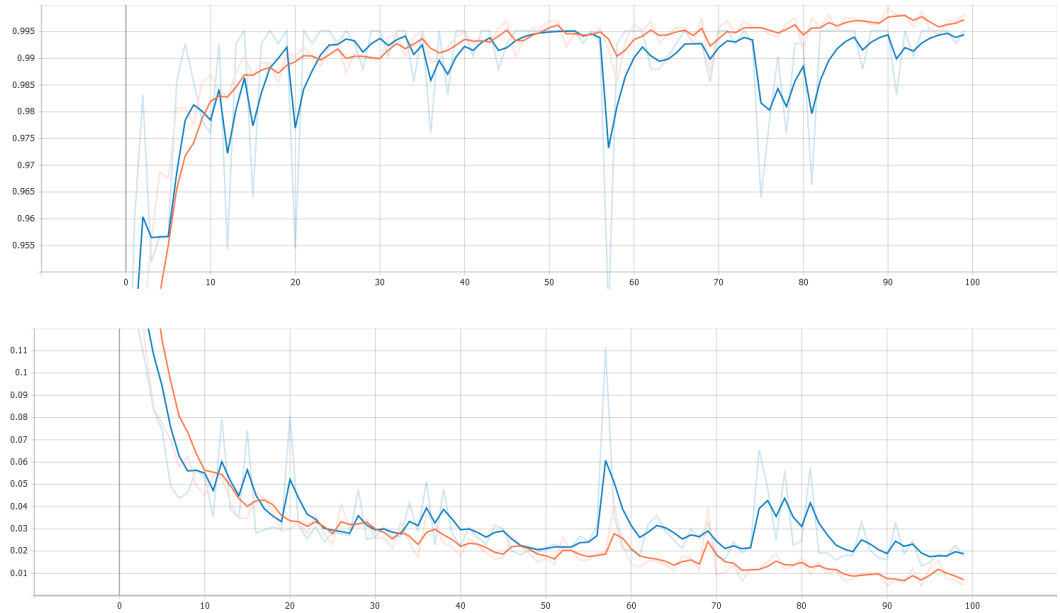

Figure B.2: Plot of model results trained on a dataset containing data from a hand placed in the middle on the sensor matrix. Final training acc.: 99.72 %, final val. acc.: 99.44 %, final training loss: 0.0072, final val. loss: 0.0188

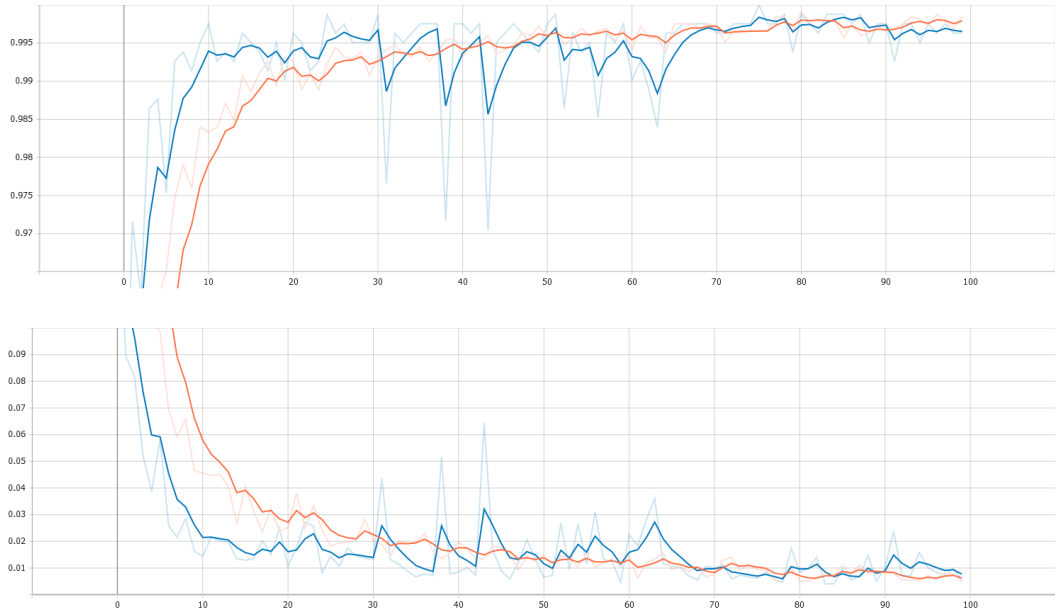

Figure B.3: Plot of model results trained on a dataset containing data from a hand placed in the middle and upper right on the sensor matrix. Final training acc.: 99.79 %, final val. acc.: 99.65 %, final training loss: 0.0048, final val. loss: 0.0053

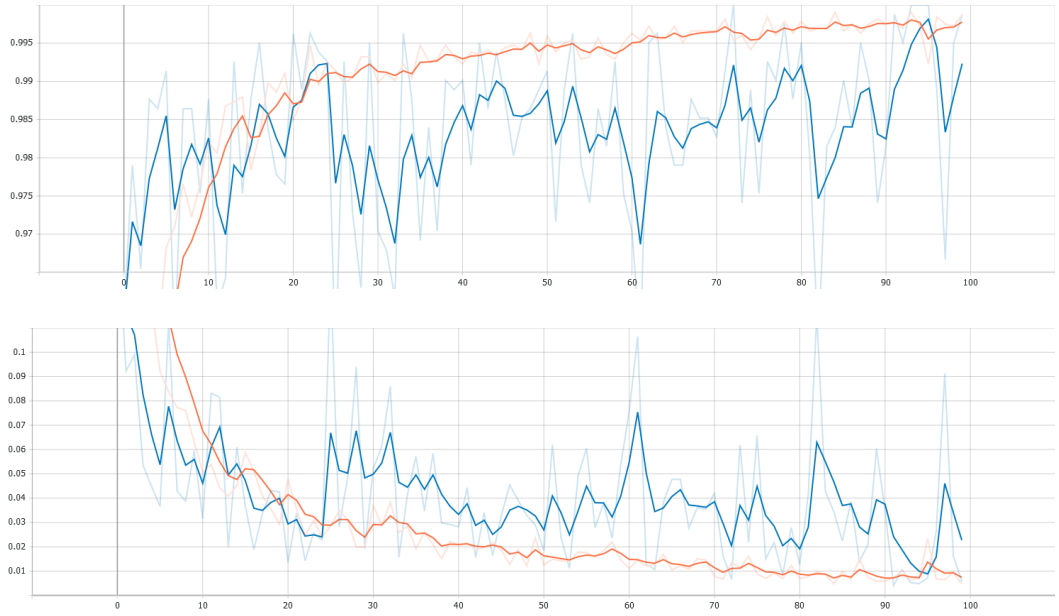

Figure B.4: Plot of model results trained on a dataset containing data from a hand placed in the middle, upper right and lower right on the sensor matrix. Final training acc.: 99.78 %, final val. acc.: 99.23 %, final training loss: 0.0074, final val. loss: 0.0227

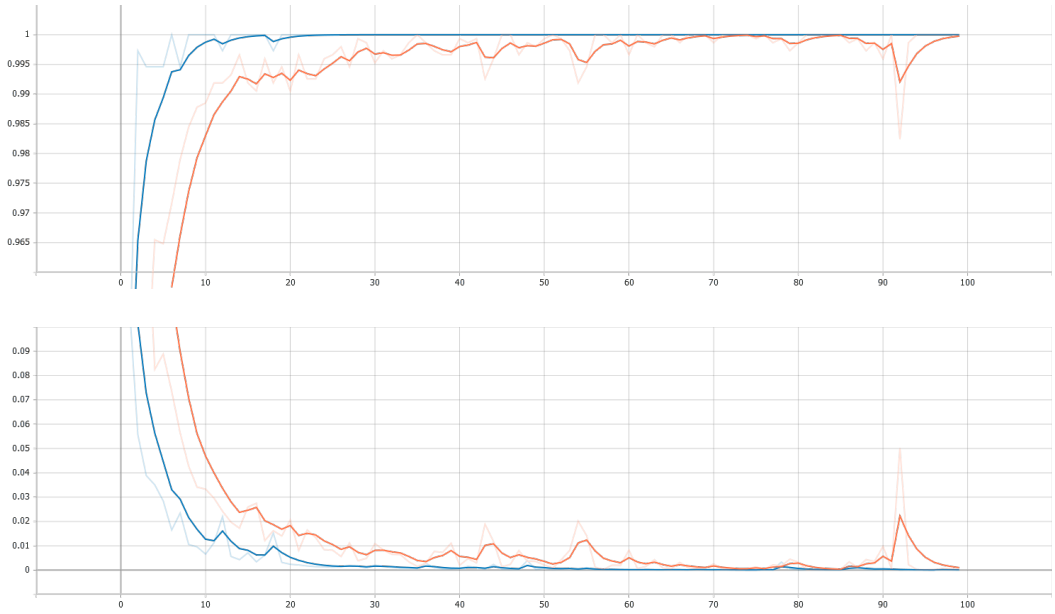

Figure B.5: Plot of model results trained on a dataset containing data from a hand placed in the upper right on the sensor matrix. Final training acc.: 99.98 %, final val. acc.: 100 %, final training loss: 0.001, final val. loss: 0.0001

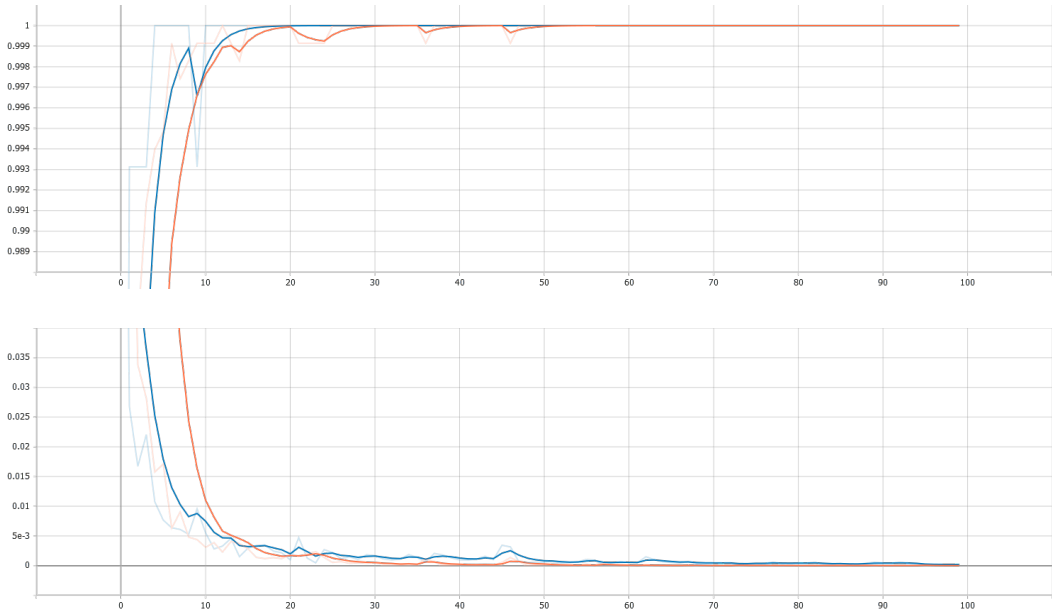

Figure B.6: Plot of model results trained on a dataset containing data from a hand placed in the lower left on the sensor matrix. Final training acc.: 100 %, final val. acc.: 100 %, final training loss: 0.00001, final val. loss: 0.0002

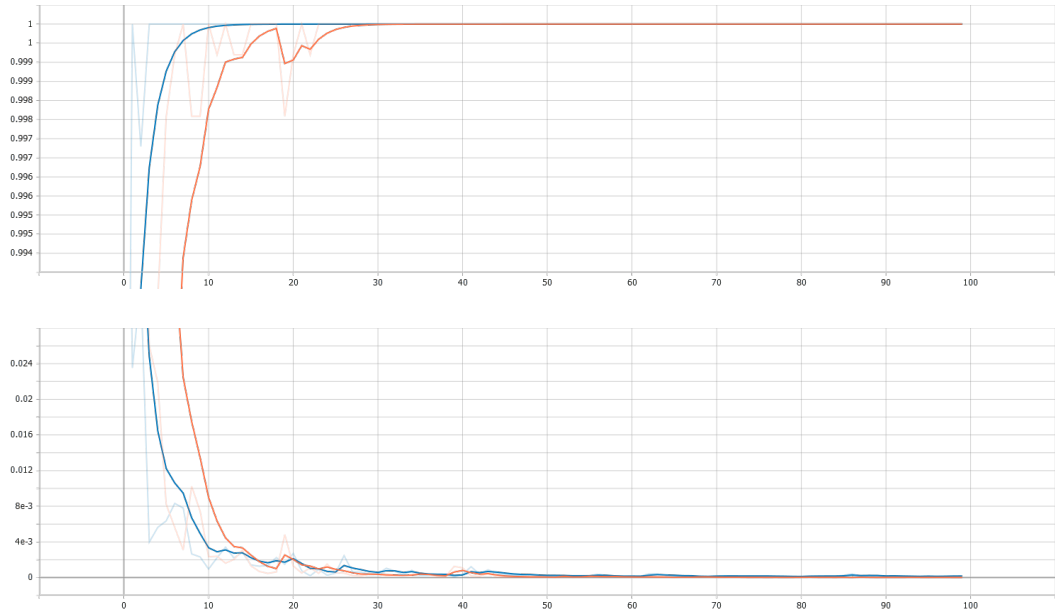

Figure B.7: Plot of model results trained on a dataset containing data from a hand placed in the upper left on the sensor matrix. Final training acc.: 100 %, final val. acc.: 100 %, final training loss: 0.00001, final val. loss: 0.0001

Bibliography

Sam Abrahams, Erik Erwitt, Ariel Scarpinelli, and Danijar Hafner. Tensorflow for machine intelligence: A hands-on introduction to learning algorithms. 2016.

Md Zahangir Alom, Tarek M Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C Van Essen, Abdul AS Awwal, and Vijayan K Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3):292, 2019.

Yunus Camgözlü and Yakup Kutlu. Analysis of filter size effect in deep learning, 2020.

Michael Fahr Jr. Investigating machine learning techniques for gesture recognition with low-cost capacitive sensing arrays. 2020.

Josue Ferri, Jose Vicente Lidón-Roger, Jorge Moreno, Gabriel Martinez, and Eduardo Garcia-Breijo. A wearable textile 2d touchpad sensor based on screen-printing technology. *Materials*, 10(12):1450, 2017.

Adam Goertz. A capacitive sensing gym mat for exercise classification & tracking. 2020.

Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989. doi: 10.1109/IJCNN.1989.118638.

JF Kaiser and WA Reed. Data smoothing using low-pass digital filters. *Review of Scientific Instruments*, 48(11):1447–1457, 1977.

Myeongsu Kang and Noel Jordan Jameson. Machine learning: Fundamentals. *Prognostics and Health Management of Electronics: Fundamentals, Machine Learning, and the Internet of Things*, pages 85–109, 2018.

Larry Li et al. Time-of-flight camera–an introduction. *Technical white paper*, (SLOA190B), 2014.

BB Narakathu, A Eshkeiti, ASG Reddy, M Rebros, E Rebrosova, MK Joyce, BJ Bazuin, and MZ Atashbar. A novel fully printed and flexible capacitive pressure sensor. In *SENSORS, 2012 IEEE*, pages 1–4. IEEE, 2012.

Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273, 2016.

Swathi Pothuganti. Review on over-fitting and under-fitting problems in machine learning and solutions. *International Journal of Advanced Research in Electrical Electronics and Instrumentation Engineering*, 7:3692–3695, 09 2018. doi: 10.15662/IJAREEIE.2018.0709015.

Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1145–1153, 2017.

Leonhard Stiny. *Passive elektronische Bauelemente*. Springer, 2015.

Chi Cuong Vu and Jooyong Kim. Simultaneous sensing of touch and pressure by using highly elastic e-fabrics. *Applied Sciences*, 10(3):989, 2020.

Anjana Wijekoon, Nirmalie Wiratunga, and Kay Cooper. Mex: Multi-modal exercises dataset for human activity recognition. *arXiv preprint arXiv:1908.08992*, 2019.

Te-Yen Wu, Lu Tan, Yuji Zhang, Teddy Seyed, and Xing-Dong Yang. Capacitivo: Contact-based object recognition

on interactive fabrics using capacitive sensing. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 649–661, 2020.

Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.

Index

Typeset January 6, 2022