Touchless Hand Gesture-Based Digit Recognition

using Light-Sensors, Convolutional Neural Networks and a Microcontroller

Winstijn Smit [1]

Supervisor(s): Qing Wang, Mingkun Yang, Ran Zhu

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Winstijn Smit
Final project course: CSE3000 Research Project
Thesis committee: Qing Wang, Mingkun Yang, Ran Zhu, Ranga Rao Venkatesha Prasad

Abstract

Touchless interaction with computers has become more important in recent years, especially in the context of the COVID-19 pandemic. Applications include situations where touch input is not possible or not desirable, e.g. for hygienic purposes in a public setting or a medical setting. Practical examples for touchless interaction include elevators, vending machines, and other public devices that are used by many people. However, most touchless interaction systems are expensive and require significant computational power. This paper proposes a bare-bones low-power and low-cost system for recognizing air-written digits using a microcontroller and light sensitivity sensors. A proof of concept has been created and tested in a fixed lighting scenario with a fixed set of gestures for the digits 0 to 9 to show the feasibility of such a system. The system uses a convolutional neural network to recognize digits and achieves an average accuracy of 58,8% on a validation set of unseen participants. It performs significantly better on new samples from users already seen during training, achieving an average accuracy of 93,5%.

Preface

This paper is part of the CSE3000 Research Project at the Delft University of Technology and with it I conclude my Bachelor of Science in Computer Science and Engineering. Recognizing hand-written digits on paper is seen as the "Hello World" of machine learning, and is often used as a benchmark for new machine learning techniques and models. However, recognizing air-written digits was a much harder problem to solve, especially with low computational power and a low amount of sensor that rely on changing ambient light. Funnily enough, it goes to show how incredible the human brain is. We as humans are able to recognize air-written digits accurately with ease, from great distances, using ambient light, with a wide variety of writing styles and in a wide variety of lighting conditions.

In any case, it has been a great learning experience and pleasure to work on this project and I would like to give appreciation to the following people: Firstly, I want to thank my responsible professor, Dr. Qing Wang and his supervisors Mingkun Yang and Ran Zhu for their kindness, feedback, advice and commitment to this project by making time for weekly meetings. I also want to thank my fellow peer group students for their company, great feedback and ideas during the project: Arne de Beer, Sem van den Broek, Gijs van de Linde and Paco Pronk. Lastly, I want to thank my family and friends for their support during my studies and this project.

1 Introduction

1.1 Motivation

The interest in touchless interaction with devices has risen in the last few years. Since the start of the COVID-19 pandemic, the world was introduced to the concept of social distancing and therefore we were forced to change the way we interact with each other and the world around us. This includes the way we interact with electronic devices, especially the ones in public. Touching the same surfaces as many other people is a potential source of infection and plays an important role in the spread of a virus [1]. The proposed digit recognition system of this paper enables touchless interaction with electronic devices which enables new possibilities.

There are several devices that can benefit of touchless interfaces. A vending machine is one such example application. It is a device that is used in a public space and is touched by many people, making them a potential source of infection. The proposed system enables touchless interaction with vending machines. Another use case is a residential elevator, where the proposed system enables touchless input of the desired floor. Additionally, the proposed system could be used in a situation where physical contact is impossible or undesired. For example, in a hospital where a surgeon needs to interact with a computer during surgery.

Similar systems have been created using cameras and image recognition [2] [3] [4] for a while but require significantly more computational power and memory, subsequently increasing costs. The presented system uses only three light sensors and a small neural network capable of running on a microcontroller. This makes the system more efficient and cheaper which could lead to a wider adoption of such systems in the future.

1.2 Related work

One year previous, in 2022, the basis for this gesture recognition system has been laid out by fellow students at the TU Delft. Dimitar Barantiev, Femi Akadiri, Matthew Lipski, Stijn van de Water and William Narchi worked together to create a prototype of the gesture recognition system. It focused on the recognition of simple hand-gestures, some examples are: "Swipe Down", "Clockwise Rotation", "Zoom in" [5]. Their contributions to this projects are as follows:

- Dimitar Barantiev created several pieces of software to interact and develop the gesture recognition system with the custom hardware. [6]
- Femi Akadiri constructed a dataset to train neural networks to recognize gestures made by changes in ambient light. [7]
- Matthew Lipski made a Recurrent Neural Network (RNN) architecture to recognize the gestures mentioned above from the data of the light sensitivity sensors [8]. The CNN-LSTM model had an accuracy of 43%.
- Stijn van de Water created the custom PCB containing the Arduino Nano 33 BLE and the three OPT101 photodiodes [9]. This hardware is used during this project to collect data and recognize the gestures using the CNN.
- William Narchi worked on a CNN architecture which could run on the Arduino Nano 33 BLE and recognize several gestures using the signals from the photodiodes [5]. His best performing model (Narrow LilConv Padding Pyramid) achieved an accuracy of 86%. His work is used as the basis of this research.

1.3 Research Questions

Building on the work of the research mentioned in section 1.2, the aim is to create a system that could recognize digits (0 to 9) made by gestures. To create such a system, the following challenges need to be overcome:

- **Creation of a dataset of gesture-based digits**
  To train the neural network, a sizable dataset of high quality should be created which represents the various hand-gestures used to represent digits made by various people. Previously created datasets cannot be used for this highly specific task. Furthermore, data needs to be collected in a consistent manner and tools to visualize, compare, analyse and manage the dataset need to be made or modified. Lastly, the hardware test-bed should be set up in such a way to collect data for this purpose.
- **Creation of convolutional neural network architecture**
  The convolutional neural network lays at the heart of this system. It should successfully recognize digits from the signals received from the light sensors. The performance of the neural network will be evaluated and compared to several different architectures and hyperparameters to conclude which model and architecture perform best.
- **Embedding of the Neural Network Model on the Arduino**
  The neural network should run locally on the Arduino Nano 33 BLE. This device has limited computational power and memory, therefore the model should be optimized to run on this device. Additionally, detecting the start of a gestures will be researched to make the system run continuously.

1.4 Contributions

As mentioned in section 1.2, this paper builds on top of the work of the previous mentioned research. However, due to the different type of gesture recognition, digit recognition, this paper makes the following contributions:

- **Creation of a dataset of gesture-based digits**
  A entirely new dataset for gesture-based digit recognition is created. This dataset contains over 6000 samples of several people writing digits from 0 to 9 in the air. Data augmentation methods that were not used before was also created. The dataset is created in a consistent manner and is of high quality.
- **Creation of convolutional neural network architecture to recognize gesture-based digits**
  A new convolutional neural network is created to recognize digits made by hand gestures with three photodiodes. Digit recognition requires more precision than gesture recognition, due to the small nuances in the movement and the low amount of photo diodes. The architecture, pre-processing and methods proposed here may be useful for other type of gesture recognition systems using a convolutional neural network.
- **Experimentation with sampling rate**
  Due to a high sampling rate during data collection it is possible to experiment with different sampling rates by downsampling. Its effect on the performance of the neural network and inference times is researched.

2 Background

In order to understand this research, one must be familiar with the terms and concepts related to a convolution neural networks. This sections aims to provide the reader with the necessary background information to understand the rest of the paper.

2.1 Convolutional Neural Networks

Neural networks are part of the field of machine learning which studies computer systems that can learn from data. Machine learning itself is a subfield of artificial intelligence (AI) in which systems are studied that can perform tasks that normally require human intelligence. In this section, the concepts of a convolutional neural network (CNN) are explained.

2.1.1 Usage of Convolutional Neural Networks

A convolutional neural network (CNN) is a type of neural network, commonly used for image recognition purposes. It is able to recognize patterns in large data fed to it, e.g. images, and is able to extract features from that data, e.g. edges, corners, and shapes. The network is able to learn to recognize these features by itself by training on a suitable dataset.

CNNs can also be used to classify time-series data. [10] Time series data is data that is collected over a period of time, e.g. sensor readings from a photodiode over time. Features and patterns embedded in time-series data are able to be recognized by a CNN, as is shown in [11].

While a CNN has many use cases, it is often used to solve classification problems. A classification problem is a problem where a system is expected to label a given input with a certain class or category. In this research project, the system is expected to label a gesture with a digit between 0 and 9 when a gesture is performed in front of the system.

2.1.2 Layers

A CNN consist of several layers, each with its own purpose. Layers follow each other in a sequential order, and each layer takes the output of the previous layer as input. Passing through a sample to be classified through all layers is called inference. The following layers are used in this research project:

- Convolutional layer: this layer performs a convolution on the input data. A series of filters (also called kernels) of a given size are applied to the input data, each filter extracting a different feature. A convolution is performed by moving the filter over the input data, multiplying data under the filter with weights in the filter, and summing the results. The result of a convolution is called a feature map.
- Pooling Layer: downsamples the input data, reducing parameters and computation in the network. The most common type of pooling layer is the max pooling layer, which takes the maximum value of a given area in the input data.

- Flatten Layer: flattens the input data in a one-dimensional array, necessary for the input of a dense layer.
- Dense Layer: often referred to as a fully connected layer, this layer connects every neuron in the previous layer to every neuron in the current layer. This layer will eventually output the classification result.
- Dropout Layer: used to prevent overfitting. This layer randomly drops a percentage of the neurons in the previous layer, forcing the network to generalize.

2.1.3 Overfitting

Overfitting is a common problem in machine learning, essentially meaning that a model is memorizing the training data instead of learning to generalize. Failing to generalize causes the model to perform poorly on unseen data which is undesirable. Overfitting can be caused my factors such as noise, too little data or a model that is too complex [12]. In this research project, overfitting is prevented by using dropout layers and data augmentation.

3 Methodology

In this section the methodology of the project is described. It will go over the hardware used, the dataset and the collection process, data augmentation and data pre-processing, the model architecture and evaluation process and finally

3.1 Hardware Overview

As mentioned in section 1.2, the hardware in this project has been created by Stijn de Water. It consists out of an Arduino Nano 33 BLE, containing a Cortex MF4 processor running at 64 MHz and 256 KB of SRAM. The micro controller does not run an operating system, only the compiled C++ code. The Arduino is connected to three OPT101 photodiodes that can measure the intensity of ambient light falling on them. The board also contains three feedback circuits that are used to control the sensitivity of the photo diodes by combining certain resistors. This ensures high resolution values in various lighting conditions. Additionally, it uses low pass filters to filter out high frequency noise. An image of the board can be in seen in Figure 1.

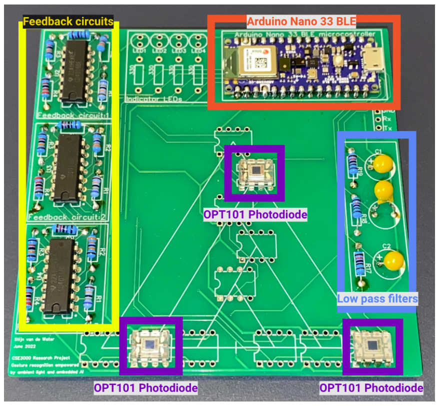


Figure 1: The custom PCB with each component labeled

3.2 Air-written digits

The aim is to recognize air-written digits with the hardware described above. The digits are written in the air by pointing with the index finger towards the light sensors. The finger is then moved as if it is a pen writing the digit on a piece of paper. For this research a map of the digits 0 to 9 has been created that can be used to perform the gestures, which can be seen in Figure 2. This map was used during the data collection process to ensure that the digits written by participants are consistent.

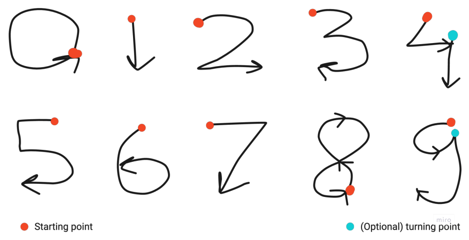


Figure 2: Map visualizing how digits are written with the index finger

3.3 Dataset

The quality of the dataset is of utmost importance for the performance of the neural network. The dataset consists out of samples containing data of gestures representing the digits 0 to 9. Data used for training should be representative of the real world data that the network will be exposed to. The dataset should be large enough to be able to train a network that generalizes well and does not overfit. Furthermore, it should be balanced, meaning that the number of samples for each digits should be roughly the same. The following sections describe the methodology and aspects of the dataset that have been used in this research.

3.3.1 Samples

The dataset consists out thousands of samples, each of which represent a single air-written digit. Along side the samples, various metadata is stored such as the label, the resistance value controlling the light sensitivity of the sensors, the sampling rate, sample duration and edge detection threshold. A single gesture sample is a collection of data points of light sensitivity values of the three sensors over time. Graphs of these samples for each number can be seen in appendix C.

The amount of data points are collected depends on the sampling rate and the duration of the gesture. The sampling rate indicates how many data points are collected per second per sensor. The sample duration indicates how long a gesture is performed. Data in the dataset has a sampling rate of 1000 Hz, allowing for experimentation with lower sample rates by downsampling the data. The duration of a sample is 2 seconds, allowing for enough time to perform digits with a slow pace.

3.3.2 Fixed environment

To prove the feasibility of the project, control of the environment is necessary. If the system is not able to work in a controlled environment, it will definitely not work in a dynamic environment. Therefore, the dataset for this research has been collected in a fixed environment.

This environment consist of a bright LED lamp that is placed 30 to 40 cm above the PCB board sensors and a fixed set of gestures. In appendix B can be seen how the lamp was placed above the board, and how the board was illuminated. Different ways of writing digits was not explored in this research and as such the dataset is limited to only one way of writing digits. Allowing for different ways of writing digits would increase the complexity of the dataset, essentially creating more than 10 classes to classify. The fixed gestures for each digit can be seen in figure 2 and appendix A. Lastly, all gesture were performed 8 to 15 centimeters above the board with the right hand and the index finger to write digits.

3.3.3 Data collection

The dataset can be split into two categories: simulated data and data collected from participants. 50% of samples have been collected from participants, the other 50% is simulated data. Samples are collected in a controlled environment and are saved with the ”Pickle” file format.

During data collection a gesture detection algorithm, as described in section 3.9, is used to detect when a gesture is being performed. This allows for the collection of samples that is consist with the eventual deployment scenario of the system. It also makes data collection easier for participants and collectors as there is no manual synchronization between them needed to collect a sample. Finally, it makes samples more consistent as the start of a gesture is roughly the same for each sample per digit.

3.3.4 Simulated data

Simulated data contains samples that are solely created by the author of this research. The purpose of it is to teach the model how to separate digits by having a set of samples that are clean and consistent. This dataset is never used for validation or testing, only for training purposes.

This simulated dataset can be further split up into the following categories:

- Controlled: samples collected in a controlled environment, with very slow and deliberate gestures. Every gesture has been manually checked for correctness and similarity to the other samples.
- Rough: samples are written in a more rough manner, less deliberate with slow downs and speed ups in the gesture.
- Sequential: samples were collected in a sequential manner in the same sitting, e.g. first 10 times the digit 0 was collected, then 10 times digit 1, etc.
- Random: sample were prompted by the computer using text-to-speech, e.g. ”Please write a 3”. The author then wrote the digit and the sample was recorded. This was done for all digits in a seemingly random order but such that the amount of samples collected per digit stayed roughly the same.

3.3.5 Participant data

Data collected from participants is used for training and validation of the model. Participants are asked to follow the digit map in figure 2. However, they were allowed to vary in the distance from the sensors, angle of the finger and speed of the gesture, as long as their samples were consistently recognizable in their respective graphs. Before collecting data, participants were asked to perform a few samples to ensure that their gestures were consistent and recognizable. Participants were also asked to perform the gestures in a sequential manner, e.g. first 10 times the digit 0, then 10 times digit 1, etc.

3.4 Software Tools

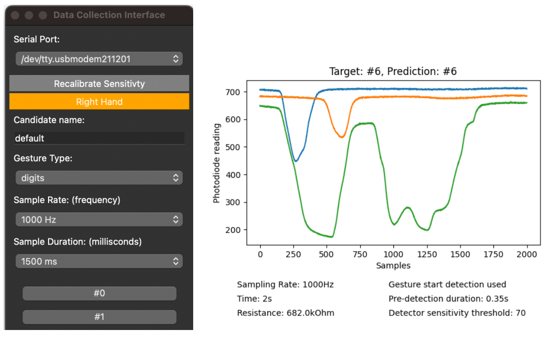
In order to collect the data and validate the model, several small tools have been created in Python. Only roughly 5% of the code of the previous research was reused, with most code being rewritten to suit the needs of this research. The tools are able to communicate with software running on the microcontroller over a serial connection. Below is a list of the tools that have been created:

- Data-collector: A tool that is able to create new samples of gestures for the dataset. It allows for configuring key parameters such as sample rate and duration, continuous data collection by detecting the start of a gesture, saving labeled samples with metadata to the dataset and running inference while collecting new data. Tools can instruct the system to recalibrate and collect data for a certain amount of time at a certain sampling rate.
- Visualizer: used to visualize the data collected in graphs, with their metadata. Visualisation helps with understanding the data and verifying the correctness of data-processing. Barcode style images can also be created with the sensor value over time being represented as a grayscale color, as seen in figure 4.
- Continuous local inference: a tool to test trained machine learning models without running them on the microcontroller. It can perform continuous inference locally on a computer, using the same gesture detection algorithm and pre-processing as on the microcontroller. This allowed for quick demonstrations and early testing with the model.
- Training and validator: tools to train and validate various different models. Using Python, Numpy and Tensorflow, models can be trained and validated on the collected dataset. It also allowed for quantization of the model for deployment on the microcontroller.

Below, in figure 3 the tools can be seen.

3.5 Data Augmentation

Manipulating training data to create new data is called data augmentation and is a common technique to improve the general performance of a model. Appropriate augmentation methods can prevent overfitting of data and improves the neural network ability to generalize [13]. Note that augmented samples are never used for validation or testing, only for enlarging and enhancing the training dataset. To make the resulting model more robust, data is augmented in several ways.


Data Collection Interface

Serial Port: /dev/tty.usbmodem211201

Candidate name: default

Gesture Type: digits

Sample Rate: (frequency) 1000 Hz

Sample Duration: (millisconds) 1500 ms

Target: #6, Prediction: #6

Sampling Rate: 1000Hz
Time: 2s
Resistance: 682.0kOhm
Gesture start detection used
Pre-detection duration: 0.35s
Detector sensitivity threshold: 70

Figure 3: Left: data collection tool - right: local inference with graph visualisation.

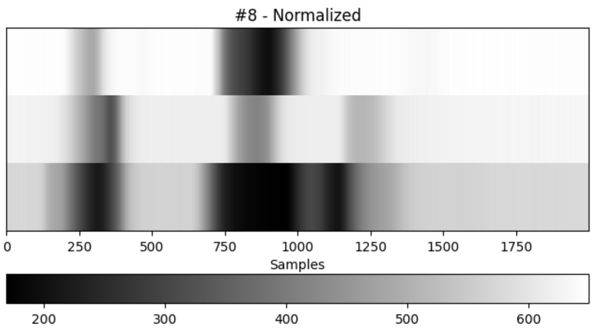


Figure 4: A image visualisation of a sample of the number 8.

- Time Warp: The data is stretched or compressed in time at multiple intervals of the sample. These warps are applied to all channels equally, resulting in no loss of synchronization between the channels. It is used to simulate the effect of a user performing a gesture faster or slower.
- Noise: Random gaussian noise is added to the data, with a probability of 50%. This is done for each sensor individually.
- Drift: Data is drifted randomly and smoothly over time with a certian trend, applied for all sensors equally.

These augmentation methods are from the Python library "tsaug", made available during a study. [10]. A random seed is used to control the randomness of data augmentation. In appendix E a single sample is augmented three times using the above methods.

3.6 Pre-processing of data

As a preparation for insertion into the machine learning model, the data is pre-processed. It has been shown that pre-processing of the data has a significant effect on neural network performance. [14] The following pre-processing steps have been applied to the data:

- Normalization: all data is normalized to a range between 0 and 1. This is done by dividing all values by the maximum value received for a certain sensor. It makes differences in light intensities between the sensors less significant to the model.
- Downsampling: the dataset contains samples with a sampling rate of 1000 Hz. Using a large sampling rate results into slower inference time and a larger model. Therefore, the data is downsampled to 100 Hz, 50 Hz, 25 Hz and 10 Hz, depending on the model. Downsampling to 100 Hz is done by keeping every 10th sample. The effects of downsampling on the data can be seen in appendix D.

3.7 Model Design

The convolutional neural network is trained using (augmented) data from the dataset that is pre-processed. The Tensorflow framework together with Numpy and Keras is used to create and train the network. [15] [16] [17] A trained model is then converted to a Tensorflow Lite model, which can be used on the microcontroller.

Since the data is a one-dimensional time series, a 1D convolutional neural network seemed suitable. Reshaping into a 2D image and using a 2D convolutional neural network is also possible, but this would require proper handling of the edges of the image, which is not trivial. Each sensor has its own channel which is processed separately by the network, only being combined at the end of the network. Several variations of the network were made by manually changing the hyperparameters, such as the number of layers, the number of filters and kernel size. A visualisation of the a model can be seen in figure 5. Most models are similar to this model, but with different amounts of layers.

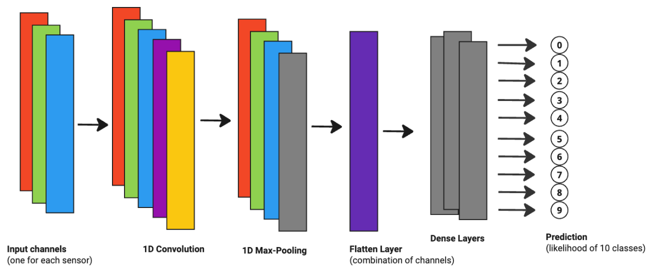


Figure 5: Visualisation of 1D convolutional network, with the input, convolutional, max pooling, flatten and dense layers respectively.

3.7.1 Model quantization

The model without quantization is too large to run on the microcontroller, therefore quantization is used. Quantization is a technique that reduces the model size while trying to maintain the performance of the model. This research uses Full Integer quantization which is applied after the training of the model. The model performance before and after quantization is reported together with the model size.

3.8 Evaluation of the machine learning model

In the following sections the way the model is evaluated is explained, as well as the metrics used to evaluate the model.

3.8.1 Accuracy

The accuracy of a trained model is evaluated using a separate validation set. This validation set is not used for training and has not been seen by the model before. Therefore, it can be used to evaluate the performance of the model on unseen data. Performance is reported as the accuracy of the model, which is the percentage of samples from the validation set that are correctly classified. There are two ways the validation set is created and they indicate different aspects of the performance of the model.

- Within participant validation: 20% of the samples from each participant is removed from the training set and used in the validation set. This validation method verifies if the model is able to recognize gestures from a participant that it has seen before.
- Between participant validation: 20% of participants are removed from the training set and used as validation set. This validation method verifies if the model is able to recognize gestures from a participant that it has not seen before, i.e. a new user of the system.

3.8.2 K-fold cross validation

A dataset split can introduce a bias during evaluation, since some samples and participants might be easier to recognize than others. To prevent this bias, k-fold cross validation is used. A split is made 5 times, each time with a different random seed. For each split a model is trained and validated using the generated training and validation sets for that split. The average accuracy of the 5 models is then reported as the final accuracy of the model. Accuraries can also be reported per class, which is done using a confusion matrix.

3.8.3 Inference Time

The inference time of the model, referring to the time it takes to classify a sample, is measured without running on the microcontroller. Due to time constraints, the model was not run on the microcontroller itself but on a Apple Macbook Pro with an M1 Max processor. The inference time reported is an average of 1000 inferences run for a certain model on the Macbook Pro.

3.9 Detection of gestures

The start of a gesture should be recognized in order to continuously detect the input digits. Since the neural network has a fixed input size, the datastream of the sensors needs to be split into samples of the correct size. However, we do not want to simply cut the datastream into fixed time intervals because it could result in to data being cut in the middle of a gesture.

Therefore, a simple algorithm has been created that detects the start of a gesture. The detection algorithm is written in Python for use during data-collection and ported to C++ to run on the microcontroller for actual inference. The algorithm works as follows:

1. Data is continuously collected from each individual sensor at a determined sampling rate.
2. An edge detector checks each sensor data for a falling or rising edge with a pre-determined threshold.
3. If an edge is detected, a certain amount of samples before detection are saved.
4. The remaining data to complete a sample is collected from the sensors.
5. A sample is created by merging the data pre-detection and post-detection. A representation of a sample can be seen in Figure 6.

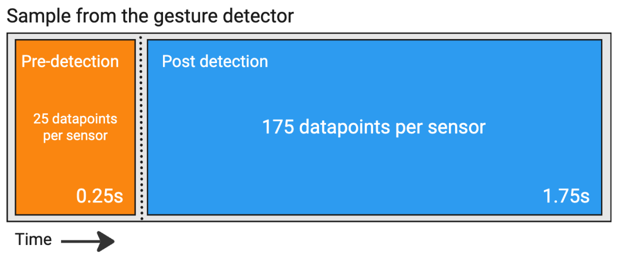


Figure 6: Visualisation of the merging of pre gesture detection and post detection data with 100 Hz sampling rate.

4 Experimental Setup and Results

In this chapter several aspects of the experimental setup and results are elaborated. First the dataset is discussed, then the model design and training process is discussed, followed by the performance of the trained models.

4.1 Dataset

A total of 6482 samples have been collected. The final dataset consists out of samples of air-written digits by the author and 17 other participants, using gesture start detection. Samples in the dataset are recorded in a fixed lighting condition, with the hand movements performed approximately 15 cm above the device. During collection an air-written digit is performed in front of the device, which is then recorded and stored. The setup during data collection can be seen in appendix B.

4.1.1 Simulated Data

The simulated part of the dataset was solely created by the author of this thesis. It consist out of around 3100 samples of air-written digits, with approximately 300 samples for each digit. Samples were collected in multiple sessions, spread over several days, using the methods described in section 3.3.4.

4.1.2 Participants Data

There were 17 participants that have contributed their gestures to the dataset. Ten of which have been collected by the author his peer group member, Gijs van de Linde, working on a similar project. The remaining seven participants have been collected by the author of this thesis. The amount of samples collected from participants can be seen in table 1. This does not include the samples collected by the author himself. Their gestures should match with the ones visualised in the digit map in figure 2 (and appendix A).

4.1.3 Evening out the dataset

The dataset was not evenly distributed over the digits, with some digits having more samples than others. This is caused by some participants recording more samples of a certain digit than others. Another problem was that some digits were discarded after the data collection process due to the sample being of bad quality. To solve this problem, the dataset was evened out using one of the following methods:

Digit	Samples
0	317
1	324
2	320
3	335
4	391
5	339
6	335
7	332
8	337
9	338
Total	**3368**

Table 1: Samples per digit from participants

- Augmentation: some random samples in the dataset until the number of samples for each digit was matching the maximum number of samples for a digit.
- Discard: some random samples in the dataset until the number of samples for each digit was matching the minimum number of samples for a digit.

This randomness was controlled with a random seed.

4.1.4 Caching of augmented data

Due to the large dataset size and the several data augmentation techniques used, it is not feasible to generate a lot of training and data before training. With 5 augmentations per sample it would take 10 minutes for a dataset of 6000 samples to be augmented. Augmentation resulted in a dataset of 30.000 samples. Hence, the augmented data together with the real data, were cached inside a Pickle file in order to speed up the training process.

4.1.5 Resulting dataset

A dataset containing 12.500 samples was created and used to evaluate the performance of the models mentioned in Section 4.3. Augmentation was used to evenly distribute the samples per digit and to increase the size of the dataset with 50%. For training and testing several splits were created, for between candidate and within candidate evaluation. A split consist out of 25% of samples or 25% of participants, used for validation, depending on the evaluation type.

4.2 Model Design Experimentation

During the course of this research, several models have been designed, tested and trained. The following variations of will be discussed in the following sections:

- 1-CNN D1
  Single convolutional layer with a single dense layer.
- 2-CNN D1
  Two convolutional layers with a single dense layer.
- Tiny 2-CNN D1
  Two convolutional layers with a single dense layer, but with a smaller amount of filters and neurons.
- 2-CNN D2 A
  Variation of a two convolutional layers with two dense layers.

Model	**Size**	**Inference Time**
1-CNN D1	55.816 bytes	4,100 microseconds
2-CNN D1	61.616 bytes	5,777 microseconds
Tiny 2-CNN D1	21.280 bytes	3,831 microseconds
2-CNN D2 A	115.472 bytes	6,500 microseconds
2-CNN D2 B	70.128 bytes	7,004 microseconds
2-CNN D2 XL	132.336 bytes	7,804 microseconds

Table 2: Quantized sizes for each model trained on data with a 25 Hz sampling rate in bytes. Inference time in microseconds computed by averaging the classification time of 100 samples with the quantized model running on a Macbook Pro with a M1 Max CPU.

- 2-CNN D2 B
  Variation of two convolutional layers with two dense layers.
- 2-CNN D2 XL
  Variation of two convolutional layers with two dense layers, but with a larger amount of filters and neurons.

The full design for each of these models can be found in appendix G.

4.3 Model Performance

In this section the performance of the models mentioned in section 4.2 are discussed. First the performance between candidates is evaluated, then the performance within candidates. Finally, the performance of the models trained on different sampling rates is evaluated.

Care was taken to ensure a fair comparison between each model. Cross validation was used to reduce the influence of a favourable split influencing the results. The same splits were used for all models, so that the results are comparable. All models were trained for 100 epochs, with a learning rate of 0.001, with data being downsampled to 25 Hz. Since models have to be quantized to run on the microcontroller, the accuracy of the quantized models is also reported. This accuracy is measured by running the quantized model on the same test set as the original model. All confusion matrices can be found in appendix F.

4.3.1 Between candidate performance

Between candidate performance represents the ability to recognize gestures from a participant that was not seen during training. Therefore it can be seen as a measure of how well the model generalizes to unseen users. Samples from some candidates were not used during training, only for verification.

From table 3 it can be seen that the best performing quantized model is 2-CNN D2 A, with an quantized accuracy of 58.779% (±2.637%). The confusion matrix for this model can be found in Figure 7. This is better than randomly guessing, which would result in an accuracy of 10%. However, this is not an optimal result since new users of the system would often have to try multiple times before the correct digit is recognized.

4.3.2 Within candidate performance

Within candidate performance represents the ability to classify new gestures from a participant that was seen before dur-

Model	Accuracy	Quantized Accuracy
1-CNN D1	54,198% (±2,825%)	53,903% (±3,081%)
2-CNN D1	57,826% (±3,144%)	56,607% (±3,215%)
Tiny 2-CNN D1	53,189% (±3,703%)	51,613% (±3,678%)
2-CNN D2 A	59,123% (±2,840%)	**58,779% (±2,637%)**
2-CNN D2 B	58,754% (±4,996%)	58,754% (±5,341%)
2-CNN D2 XL	**59,887% (±2,282%)**	57,843% (±3,294%)

Table 3: 5-fold between candidate accuracy and quantized accuracy for each model trained on 25 Hz data.

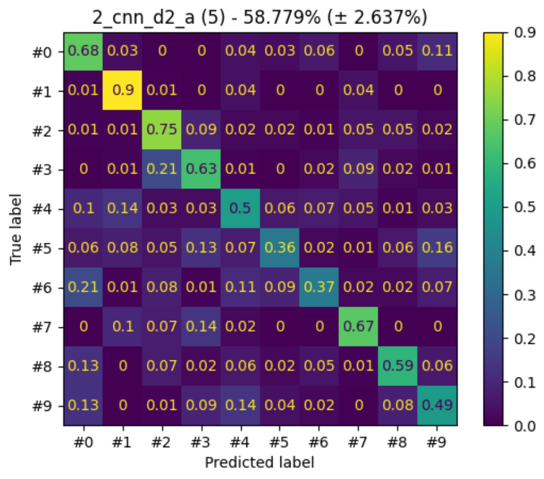


Figure 7: Confusion matrix of the 5-fold within candidate quantized accuracy of 2-CNN D2 A.

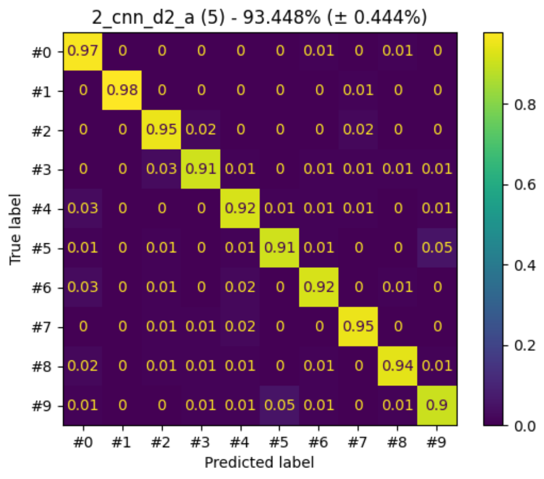


Figure 8: Confusion matrix of the 5-fold within candidate quantized accuracy of 2-CNN D2 A.

ing training. Hence all candidates were used for training, but only a subset of their samples were used for validation.

From table 4 it can be seen that the best performing model is the 2-CNN D2 A model, with a quantized accuracy of 93.448% (±0.444%). The confusion matrix for this model can be found in Figure 8.

Within candidate performance is significantly better than the between candidate performance. The proposed system would therefore work best if the user is already known to the system beforehand.

4.3.3 Downsampling

The dataset contains samples with a sampling rate of 1000 Hz, which allowed experimentation with different sampling rates. The effect of downsampling on data can be seen in appendix D.

The hypothesis was that a lower sampling rate would result in a smaller model and therfore faster inference time. However, it was also expected that a lower sampling rate would result in a lower accuracy of the model. The model 2-CNN D2 A was trained on datasets with sampling rates of 100 Hz, 50 Hz, 25 Hz, 10 Hz and 5 Hz. Moreover, the candidate and sample split were kept the same between the different sampling rates, allowing for a fair comparison. The changes in performance, size and inference time can be found in table 5.

The Arduino has a working memory of 256 KB, therefore it is unlikely any model above 200 KB will be able to actually perform inference. Taking the accuracy numbers into account it seems that the most suitable sampling rate is 25 Hz, as it has the best performance and is still small enough to fit on the Arduino.

5 Responsible Research

5.1 Ethics

With the increasing use of machine learning making impactful decisions, it is important to consider the ethical implications of the use of machine learning. While the proposed system is not intended to be used in a real-world scenario, further research might lead to a system that is used in a real-world scenario. Such scenarios might include the use in an elevators, vending machines, or even high stake scenarios like surgery where touch input is not possible. The following points are important to consider for a real-world system:

- Inclusivity: bias in the data used to train the model might lead to bias in the model itself, which might lead to worse performance for certain groups of people. To prevent this, the dataset should be collected in a way that is representative of the population that the model is intended to be used on. This includes representing different writing styles, hand postures, gesture speed, left and right handedness evenly in the dataset and evaluating performance with respect to these groups.

Model	Accuracy	Quantized Accuracy
1-CNN D1	89,513% (±0,653%)	89,611% (±0,395%)
2-CNN D1	91,696% (±1,556%)	91,339% (±1,121%)
Tiny 2-CNN D1	84,948% (±8,341%)	84,294% (±8,180%)
2-CNN D2 A	**93,399% (±0,392%)**	**93,448% (±0,444%)**
2-CNN D2 B	92,276% (±1,192%)	91,906% (±1,405%)
2-CNN D2 XL	91,795% (±0,948%)	91,610% (±0,894%)

Table 4: 5-fold within candidate accuracy and quantized accuracy for each model trained on 25 Hz data.

Sample rate	Between Accuracy	Within Accuracy	Size (bytes)	Inference Time	Time %	Perf. %
100 Hz	54,543% (±5,243%)	91,314% (±0,583%)	418.576	17,283	100%	97,72%
50 Hz	56,735% (±4,119%)	92,548% (±0,609%)	213.776	9,898	57,27%	99,04%
25 Hz	58,779% (±2,637%)	93,448% (±0,444%)	115.472	6,451	37,33%	100%
10 Hz	56,513% (±3,270%)	89,784% (±0,824%)	49.936	4,365	25,26%	96,08%
5 Hz	45,802% (±4,908%)	76,743% (±3,371%)	33.552	3,657	21,16%	82,12%

Table 5: 5-fold between and within candidate quantized accuracy with model size in bytes and inference time in microseconds for different sampling rates of 2-CNN D2 A.

- Privacy: the device does not have to store any personal information about a user to recognize digits. Additionally, all of the processing on the device itself and no data is sent to a external party. However, the device is probably connected to another system receiving the digits which might store personal information about the user.
- High stakes scenarios: high priority should be given to the safety of the user in such a scenario. The system should be designed in such a way that predictions are correct and reliable. In an event of a failure, such as a wrong prediction, actions should be reversible and non-destructive.

5.2 Reproducibility

Care has been taken to ensure that the results of this research are reproducible. As such, most code written, together with the dataset used for training and validation, is available on Github.

5.3 Participants and consent

During this research, 17 participants have been involved in the data collection process. These participants have been asked to sign a consent form of which an empty example can be found in appendix H. The participants have been informed about the purpose of the research, the data collection process and the use of their data for training a neural network. Furthermore, the privacy of the participants has been protected by anonymizing the data and not storing any personal information about the participants alongside the gesture samples.

6 Discussion

In this section, the process of the research is reflected upon and the results are discussed.

6.1 Collection of the dataset

The collection of the dataset took longer than initially anticipated. Firstly because the dataset collected in the first half of the research was discarded. It was collected in a dynamic environment, meaning that the lighting conditions and digit writing were not fixed. The fixed environment was introduced after noticing that this data was very inconsistent and hard to classify. To be able to still make a minimum working system, the data collection was reproduced in the fixed environment. The reasoning was as follows: if a system with a fixed environment would not work, a system with a dynamic environment would definitely not work either.

Additionally, in week 6, the idea came up to use gesture detection for data collection to remove the need for a button press by the data collector. This would make the data collection more consistent and dataset samples more similar to the real-world use case. The change made classification for the system easier but also resulted in the removal of samples that were not collected with the gesture detection algorithm.

6.2 Lack of deployment on microcontroller

Unfortunately, due to time constraints, the deployment of the model on a microcontroller was not possible. Since deployment of a CNN had been done in previous work on the system, it was not considered a high priority. Time was spent on other aspects of the research, such as the general feasibility, augmentation methods and experiments with the sampling rate that contribute new insights to the system. Furthermore, demonstrations could still be given to validate the model performance, with inference running on a laptop and serial communication to the microcontroller.

6.3 Reflection on performance of the system

The performance of the system seems to be satisfactory due to the difficulty of the task and constraints of the system. The system performs well on new samples from participants that had been seen before, with an accuracy above 90%. Presumably because the model has learned the nuances in writing style of the candidate and can use this to classify the digits of the candidate.

For participants that have not been seen before (between candidate), the performance is significantly worse with an accuracy drop of roughly 40%. This means the system is not able to generalize well to new participants and can not be used in a public real-world scenario with many different users. Due to the low amount of sensors, small variations in writing style of the fixed digits gestures and fixed environment can still cause large variations in the data in the end. When the model has not seen these variations before, it is not able to classify the digits correctly and the accuracy drops. However, the author is confident that the system can be improved by using computer simulated data and some hardware changes as discussed in the section 7.2.

7 Conclusions and Future Work

7.1 Conclusion

This paper has shown a proof of concept of a low-power and low-cost system for recognizing air-written digits using only three light sensitivity sensors. A dataset has been collected in a fixed lighting scenario and with a fixed set of gestures to show the feasibility of such a system. Pre-processed and augmented data has been used to train several convolutional neural networks that are able to recognize the digits 0 to 9 written in the air.

The best architecture ”2-CNN D2 A” achieved an accuracy of 93,48% on unseen samples from participants that have been seen during training. The performance on entirely unseen participants, representative of real world performance, is lower with an accuracy of 58,78%. Experimentation has shown that a sampling rate of 25 Hz is most suited for this system.

Further research is needed to make the system more robust and usable in real world scenarios. In its current state, the system is only reliable for a fixed amount of users, that have input their own samples, in a fixed lighting scenario and with a fixed set of gestures. However, recognizing digits with only three light sensitivity sensors, with many variations in writing style and light environment, is a difficult task.

7.2 Future Work

During this research, several limitations and improvements have been identified that can be used for future research. The following sections describe these limitations and possible improvements to make the system more robust and usable in real world scenarios.

7.2.1 Testing inference times on the microcontroller

Time constraints caused by creating the dataset did not allow for testing the inference times on the microcontroller. The several models trained and quantized could be ported in the future to run on the microcontroller, to see if the system is able to perform inference in real-time. Since running CNN models on the microcontroller of this system has been done before, it is expected to be trivial [5] [6].

7.2.2 Adding dynamic environment in the dataset

A real world application would require the system to be able to recognize digits in a more dynamic environment. However, the dataset used to train the model has been collected in a fixed lighting scenario and with a fixed set of gestures. In turn, the model is only able to recognize digits in this environment and with these predetermined gestures. To improve the model and make it more suitable for real world applications, new samples should be collected in a more dynamic environment.

7.2.3 Computer Simulation of the dataset

To create a system that works in a dynamic environment, with a model that generalizes well and does not overfit, a dataset of high quality is required. Collection of such a dataset is an extremely time consuming task, especially when the dataset needs to be collected in a dynamic environment. A dynamic environment is often hard to control but in a simulation this environment can be precisely controlled.

Using a simulation, a large dataset can be generated in a short amount of time, which can be used to train a model that generalizes well. The simulation needs to simulate the photodiodes and several light sources at various positions and intensities, preferably using ray tracing. It should also simulate the finger position, angle of the hand, size of the hand and a certain writing style. Hand movement might be able to be generated from the popular MNIST dataset, which contains images of handwritten digits. The simulation can also be used to research the optimal amount of photodiodes and the influence of the position of the photodiodes.

7.2.4 Adding more photodiodes

The current system only uses 3 photodiodes, which limits the resolution and in turn the accuracy of the model, especially in a more dynamic environment with a complex task such as recognizing digits. More photodiodes should be added to the system to increase the resolution making it easier for the system to recognize digits and thus increasing the system’s accuracy.

7.2.5 Tracking the finger position using photodiodes

By tracking the finger position over time, a 2D image can be created of the path the finger has traveled, essentially as if using a pen on a piece of paper. The way a digit, letter or word is written would not matter anymore, as long as the ending image is the same. This image can then be given to a neural network to recognize the digits, letters or perhaps even words with possibly higher accuracy. Such a technique has been used to recognize digits in the air using a camera in [4].

To achieve this with photodiodes, training data would require position data of the finger position during data collection and the values of the photodiodes over time. This can be achieved by using a camera on the device during data collection, or by using a simulation as described in section above. The recommendation is to use a simulation and to use a system with a grid of photodiodes.

7.2.6 Modification of the device to emit infrared light

The current system relies on ambient light, which limits the system to a well lit environment. It also makes the system more susceptible to noise and introduces significant more difficulty in creation due to the varying light conditions. By emitting infrared light, that bounces off the finger back to sensors on the board, a theoretical system could be created that is also able to recognize air-written digits. This system would be able to work in a dark environment, would be less susceptible to noise and would be easier to create due the fixed light condition.

Bibliography

[1] N. Zhang, Y. Li, and H. Huang, "Surface touch and its network growth in a graduate student office," *Indoor Air*, vol. 28, no. 6, pp. 963–972, 2018. DOI: https://doi.org/10.1111/ina.12505. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ina.12505. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/ina.12505.

[2] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: A survey," *Artificial intelligence review*, vol. 43, pp. 1–54, 2015.

[3] K. K. Biswas and S. K. Basu, "Gesture recognition using microsoft kinect®," in *The 5th international conference on automation, robotics and applications*, IEEE, 2011, pp. 100–103.

[4] T. Watanabe, M. Maniruzzaman, M. A. M. Hasan, H.-S. Lee, S.-W. Jang, and J. Shin, "2d camera-based air-writing recognition using hand pose estimation and hybrid deep learning model," *Electronics*, vol. 12, no. 4, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12040995. [Online]. Available: https://www.mdpi.com/2079-9292/12/4/995.

[5] W. Narchi, "Recognising gestures using ambient light and convolutional neural networks, Adapting convolutional neural networks for gesture recognition on resource-constrained microcontrollers," Delft University of Technology, 2022.

[6] D. Barantiev, "Designing a software receiver for gesture recognition with ambient light," Delft University of Technology, 2022.

[7] F. Akadiri, "Constructing a dataset for gesture recognition using ambient light," Delft University of Technology, 2022.

[8] M. Lipski, "Hand gesture recognition on arduino using recurrent neural networks and ambient light," Delft University of Technology, 2022.

[9] S. van de Water, "Designing an adaptable and low-cost system for gesture recognition using visible light," Delft University of Technology, 2022.

[10] T. Wen and R. Keyes, "Time series anomaly detection using convolutional neural networks and transfer learning," *CoRR*, vol. abs/1905.13628, 2019. arXiv: 1905.13628. [Online]. Available: http://arxiv.org/abs/1905.13628.

[11] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, 2017. DOI: 10.21629/JSEE.2017.01.18.

[12] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, IOP Publishing, vol. 1168, 2019, p. 022022.

[13] T. T. Um, F. M. Pfister, D. Pichler, *et al.*, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks," in *Proceedings of the 19th ACM international conference on multimodal interaction*, 2017, pp. 216–220.

[14] S. F. Crone, J. Guajardo, and R. Weber, "The impact of preprocessing on support vector regression and neural networks in time series prediction.," in *DMIN*, 2006, pp. 37–44.

[15] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[16] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[17] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

A Digit Legend

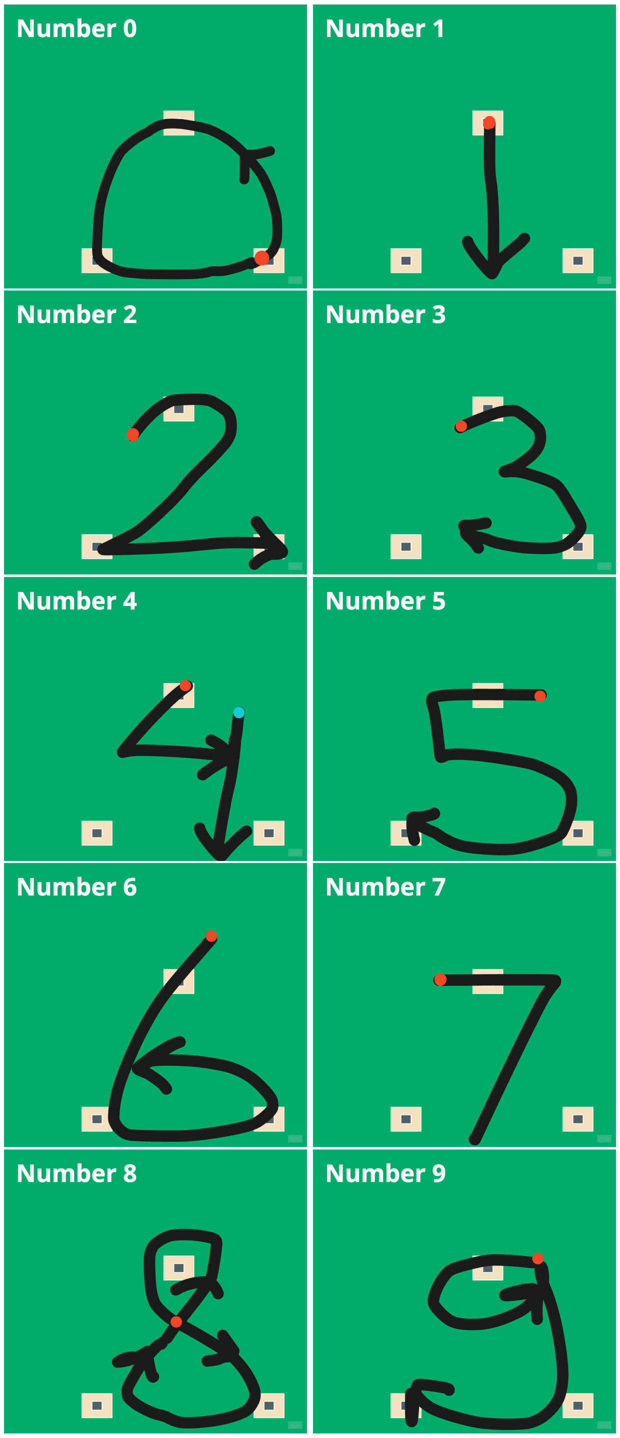


Figure 9: Legend describing how digits are expected to be drawn above the PCB, with dots and arrows indicating starting point and direction.

B Data Collection Setup

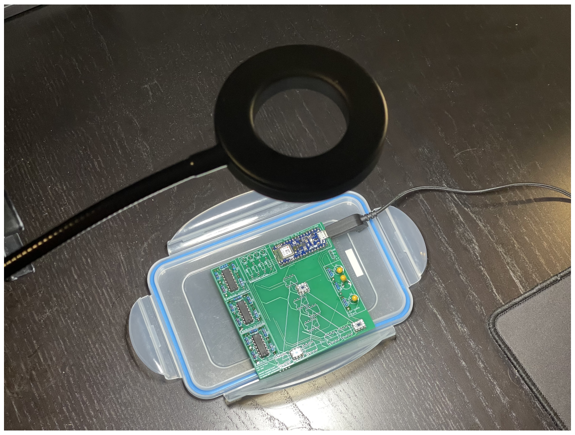


Figure 10: The board illuminated by the lamp from above.

Figure 11: HEMA selfie ring light used for fixing lighting environment.

C Digit graphs

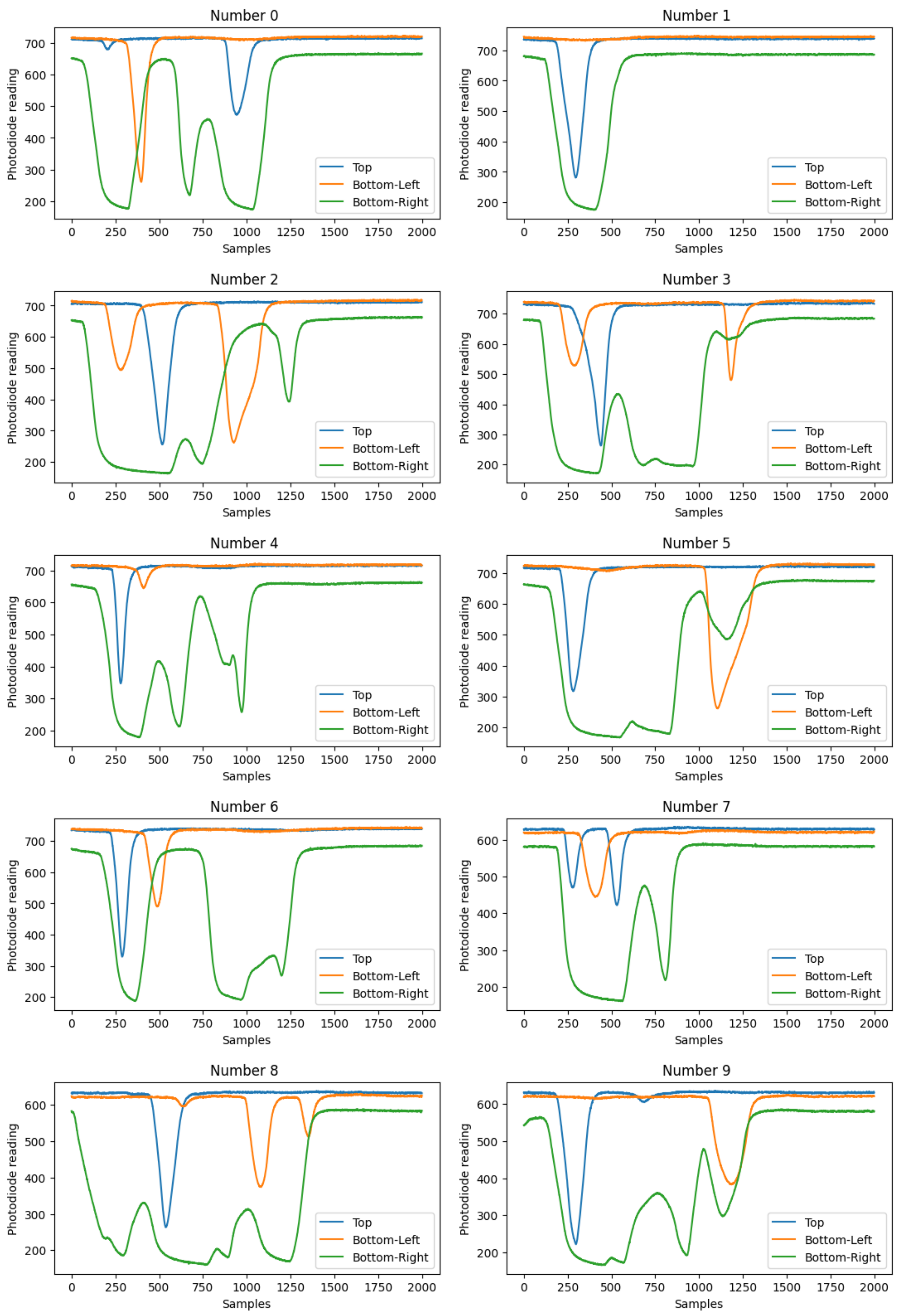


Figure 12: Graphs of the raw sensor data from the dataset, representing each digit at a 1000 Hz sampling rate and 2 second sample duration.

D Data downsampling

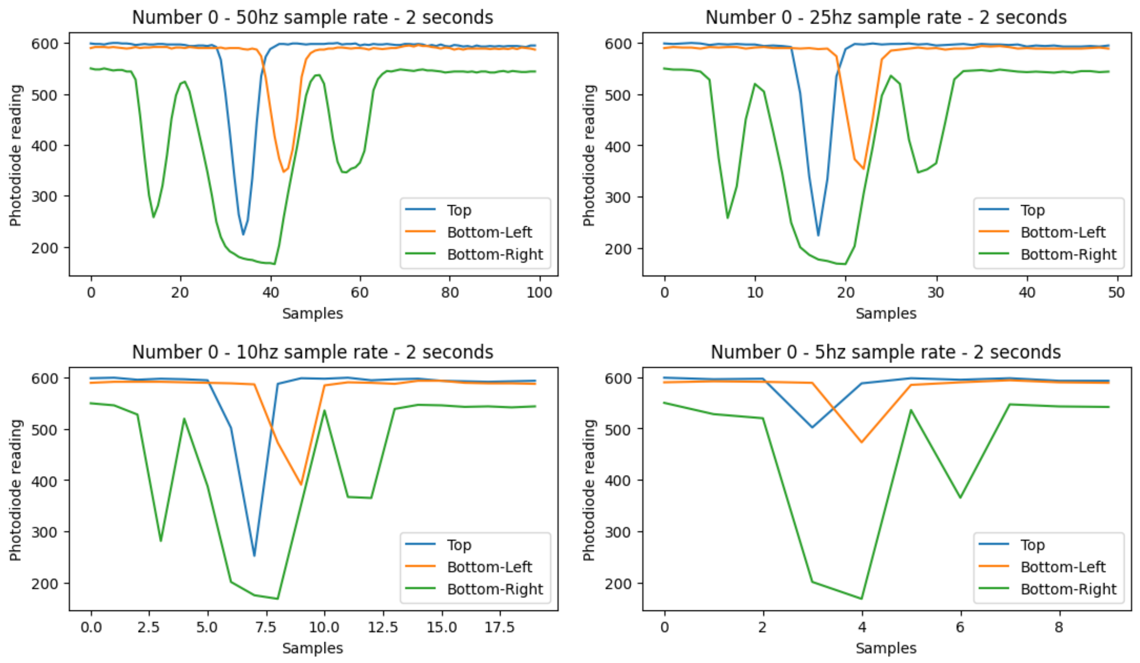


Figure 13: Graphs of the same digit downsampled from 1000 Hz to 50 Hz, 25 Hz, 10 Hz and 5 Hz.

E Augmentation of a sample

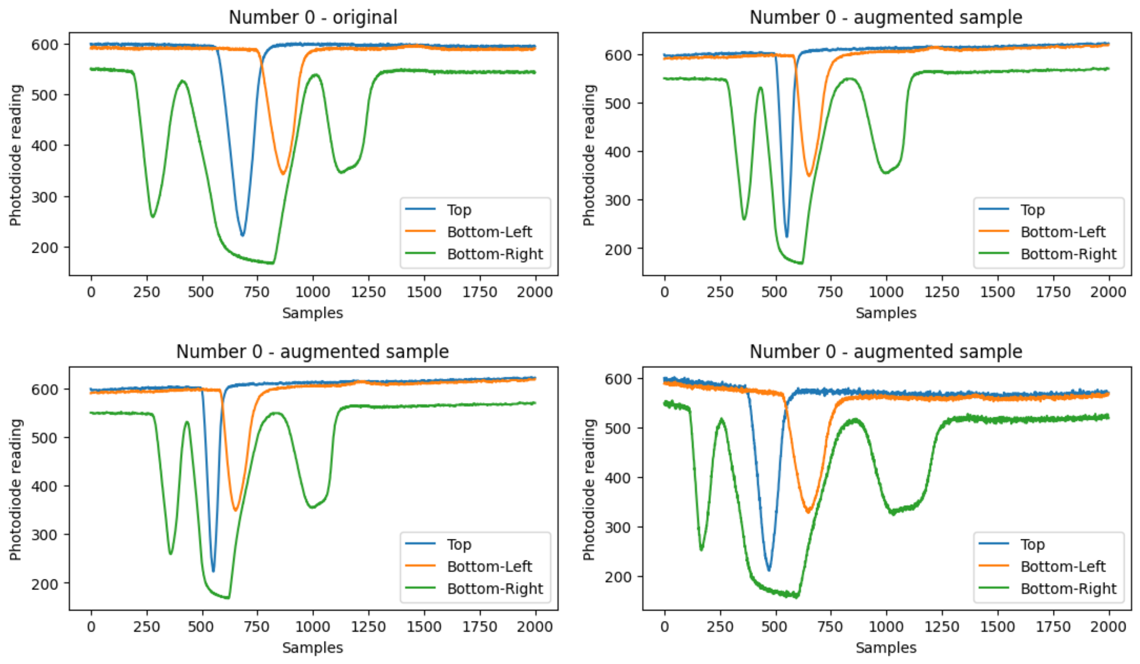


Figure 14: Graphs of one original sample (top left) and three augmented samples based on the original sample.

F Model Performances

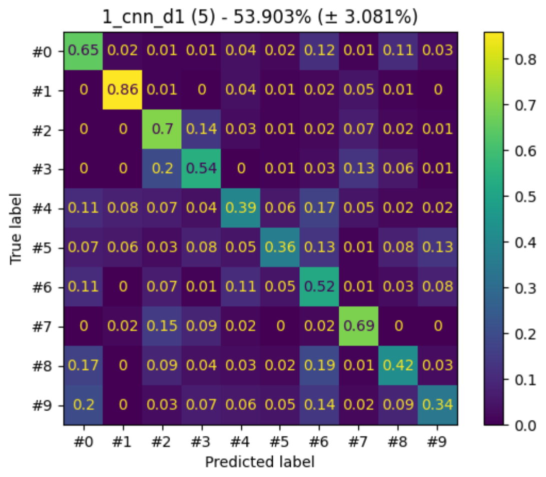


Figure 15: Confusion matrix of the 5-fold between candidate quantized accuracy of the 1-CNN D1 model.

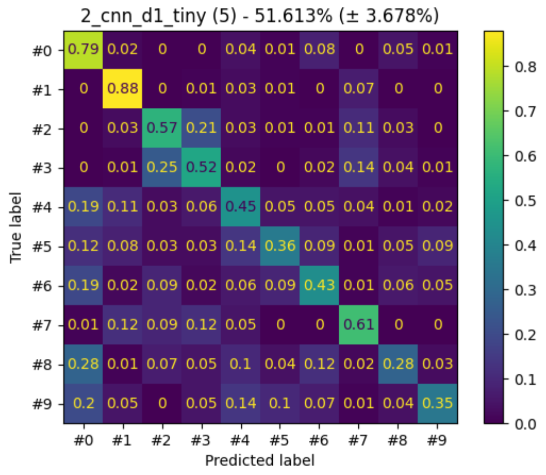


Figure 16: Confusion matrix of the 5-fold between candidate quantized accuracy of the Tiny 2-CNN D1 model.

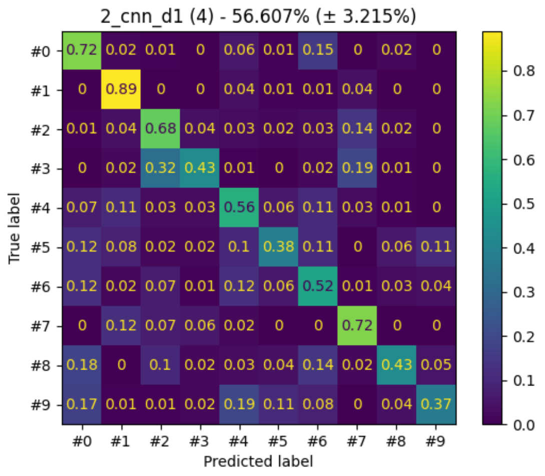


Figure 17: Confusion matrix of the 5-fold between candidate quantized accuracy of the 2-CNN D1 model.

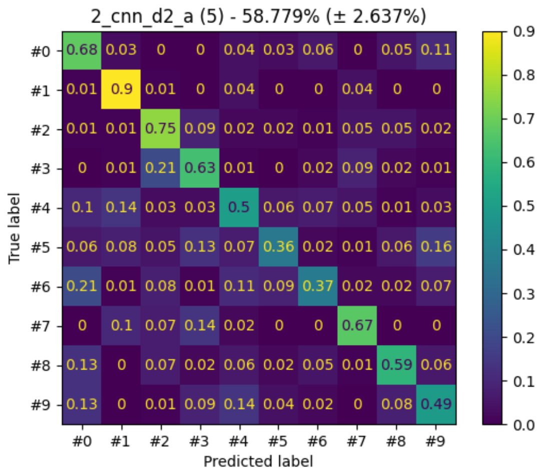


Figure 18: Confusion matrix of the 5-fold between candidate quantized accuracy of the 2-CNN D2 A model.

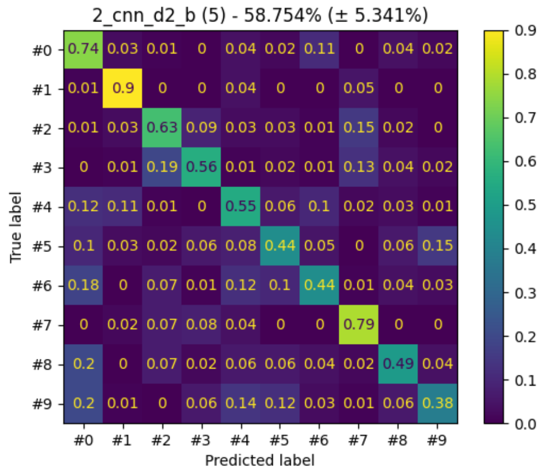


Figure 19: Confusion matrix of the 5-fold between candidate quantized accuracy of the 2-CNN D2 B model.

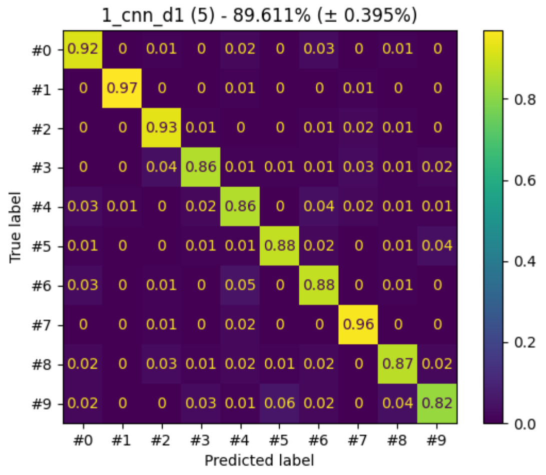


Figure 21: Confusion matrix of the 5-fold within candidate quantized accuracy of the 1-CNN D1 model.

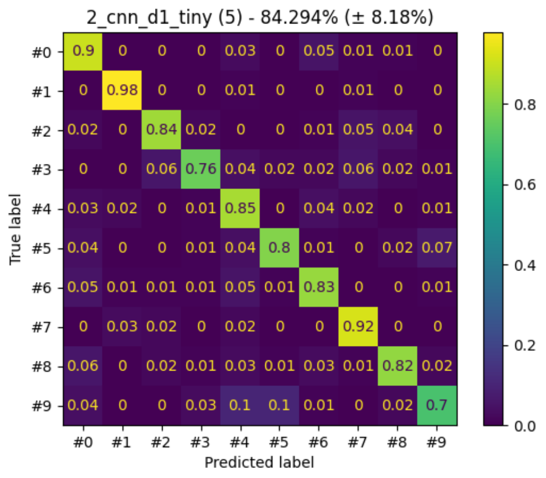


Figure 22: Confusion matrix of the 5-fold within candidate quantized accuracy of the Tiny 2-CNN D1 model.

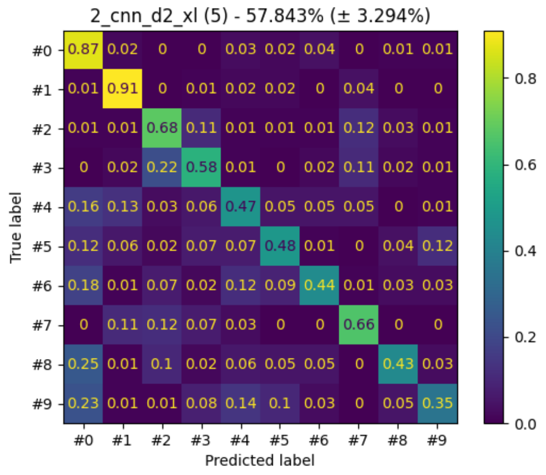


Figure 20: Confusion matrix of the 5-fold between candidate quantized accuracy of the 2-CNN D2 XL model.

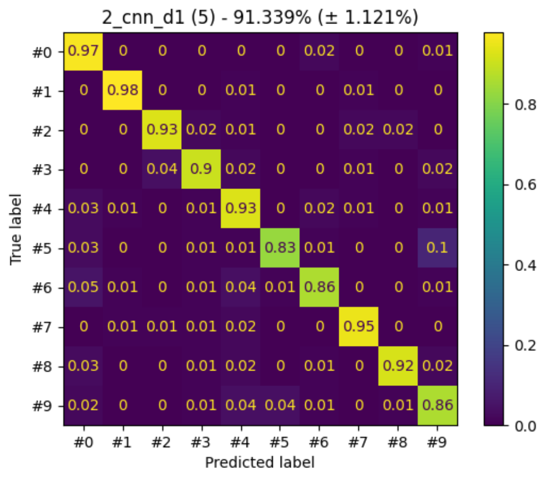


Figure 23: Confusion matrix of the 5-fold within candidate quantized accuracy of the 2-CNN D1 model.

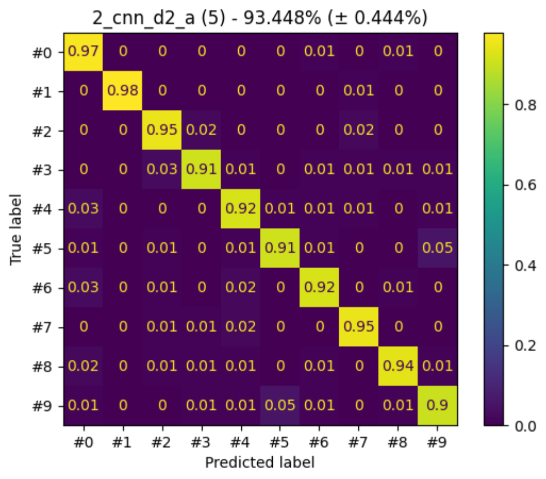


Figure 24: Confusion matrix of the 5-fold within candidate quantized accuracy of the 2-CNN D2 A model.

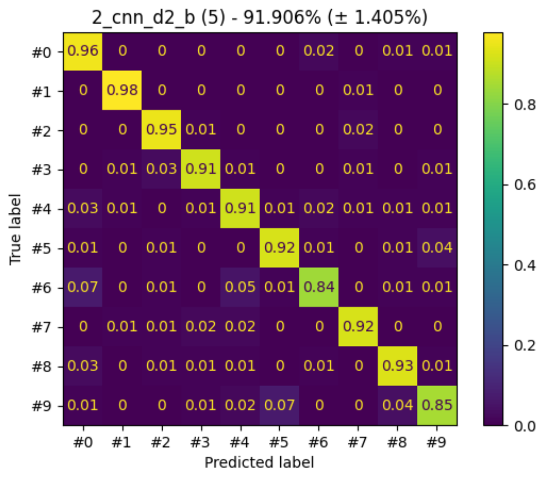


Figure 25: Confusion matrix of the 5-fold within candidate quantized accuracy of the 2-CNN D2 B model.

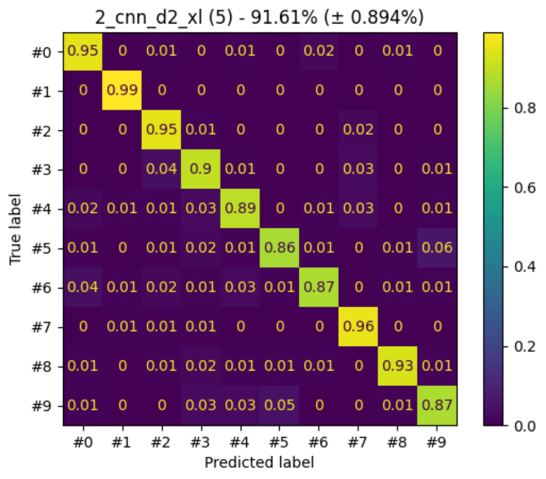
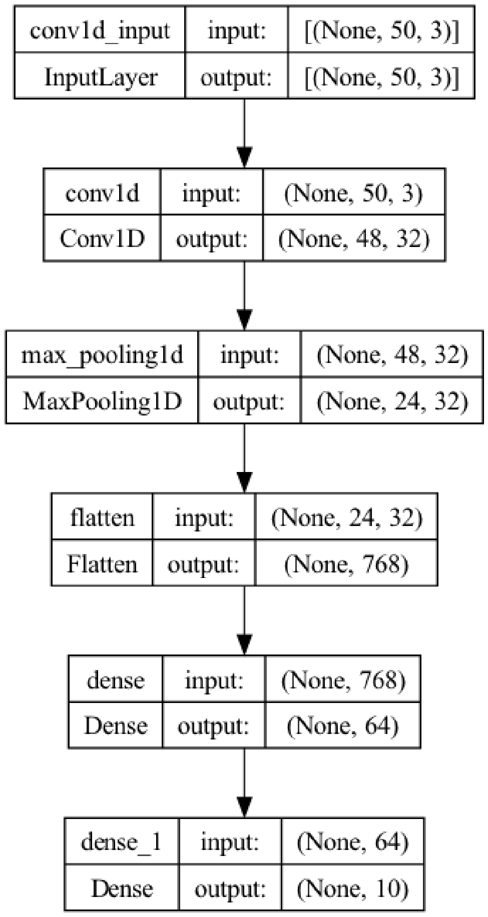


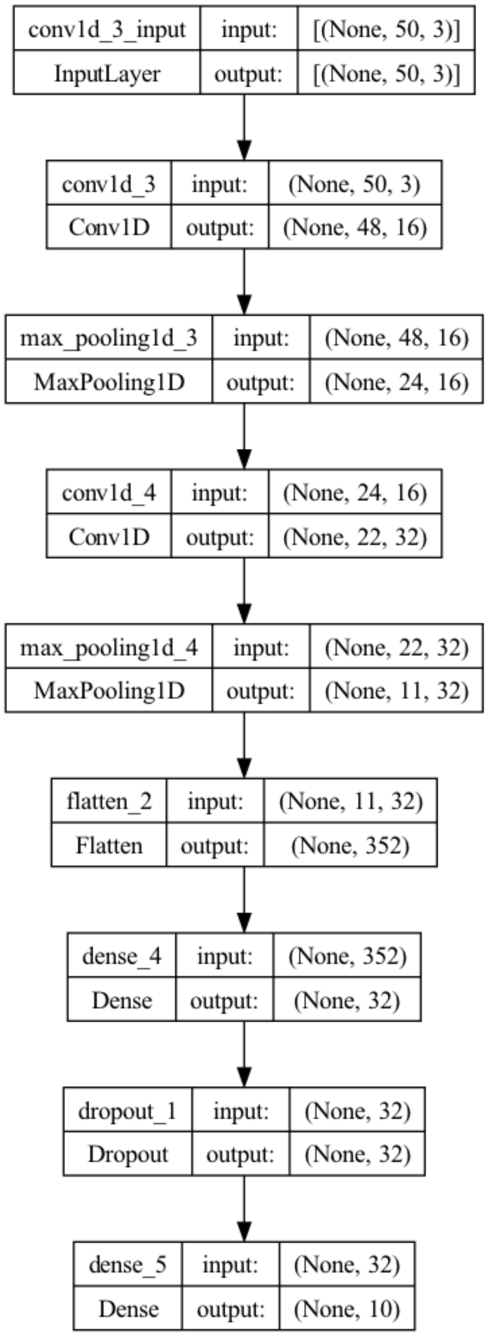
Figure 26: Confusion matrix of the 5-fold within candidate quantized accuracy of the 2-CNN D2 XL model.

G Model Architectures



conv1d_input | InputLayer | input: [(None, 50, 3)] | output: [(None, 50, 3)]
conv1d | Conv1D | input: (None, 50, 3) | output: (None, 48, 32)
max_pooling1d | MaxPooling1D | input: (None, 48, 32) | output: (None, 24, 32)
flatten | Flatten | input: (None, 24, 32) | output: (None, 768)
dense | Dense | input: (None, 768) | output: (None, 64)
dense_1 | Dense | input: (None, 64) | output: (None, 10)

Figure 27: Architecture of 1-CNN D1.



conv1d_3_input | InputLayer | input: [(None, 50, 3)] | output: [(None, 50, 3)]
conv1d_3 | Conv1D | input: (None, 50, 3) | output: (None, 48, 16)
max_pooling1d_3 | MaxPooling1D | input: (None, 48, 16) | output: (None, 24, 16)
conv1d_4 | Conv1D | input: (None, 24, 16) | output: (None, 22, 32)
max_pooling1d_4 | MaxPooling1D | input: (None, 22, 32) | output: (None, 11, 32)
flatten_2 | Flatten | input: (None, 11, 32) | output: (None, 352)
dense_4 | Dense | input: (None, 352) | output: (None, 32)
dropout_1 | Dropout | input: (None, 32) | output: (None, 32)
dense_5 | Dense | input: (None, 32) | output: (None, 10)

Figure 28: Architecture of Tiny 2-CNN D1.

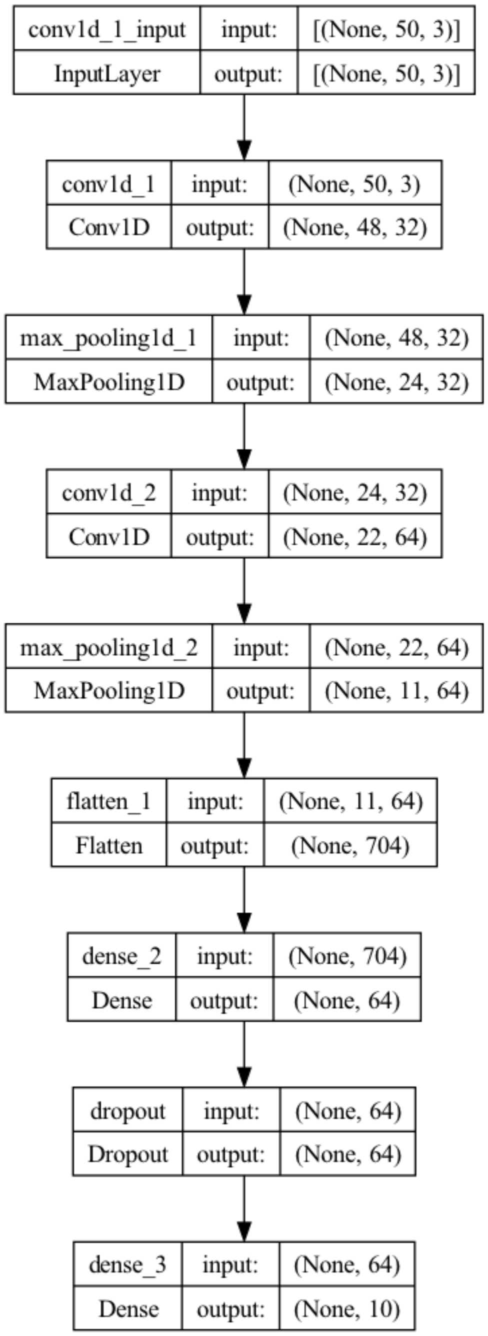

conv1d_1_input | input: | [(None, 50, 3)]
InputLayer | output: | [(None, 50, 3)]
conv1d_1 | input: | (None, 50, 3)
Conv1D | output: | (None, 48, 32)
max_pooling1d_1 | input: | (None, 48, 32)
MaxPooling1D | output: | (None, 24, 32)
conv1d_2 | input: | (None, 24, 32)
Conv1D | output: | (None, 22, 64)
max_pooling1d_2 | input: | (None, 22, 64)
MaxPooling1D | output: | (None, 11, 64)
flatten_1 | input: | (None, 11, 64)
Flatten | output: | (None, 704)
dense_2 | input: | (None, 704)
Dense | output: | (None, 64)
dropout | input: | (None, 64)
Dropout | output: | (None, 64)
dense_3 | input: | (None, 64)
Dense | output: | (None, 10)

Figure 29: Architecture of 2-CNN D1

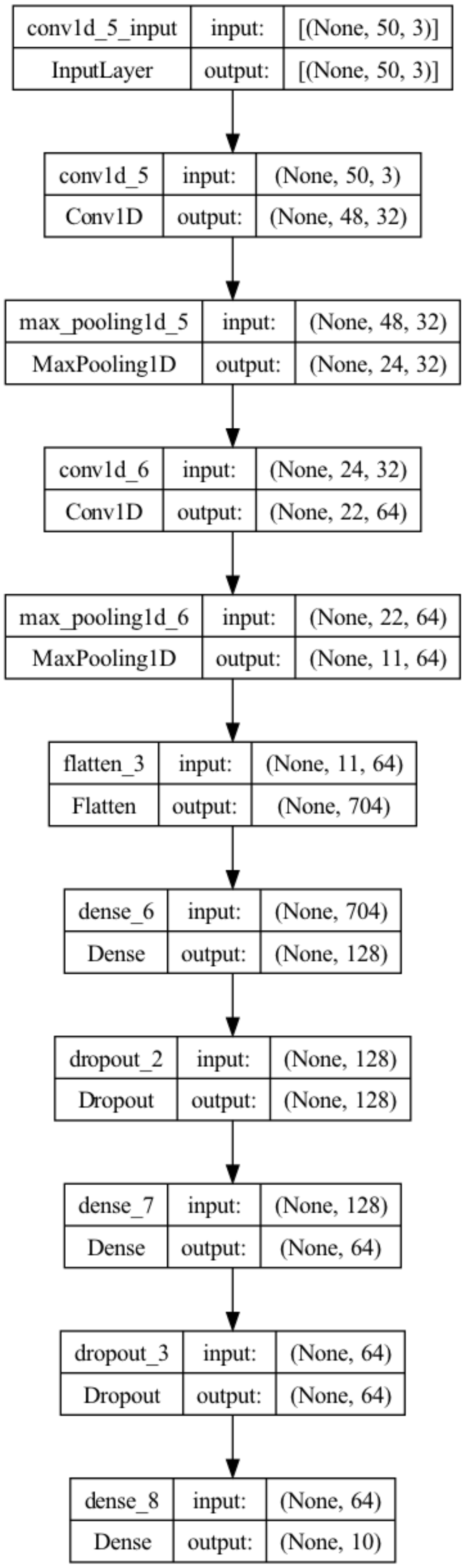

conv1d_5_input | input: | [(None, 50, 3)]
InputLayer | output: | [(None, 50, 3)]
conv1d_5 | input: | (None, 50, 3)
Conv1D | output: | (None, 48, 32)
max_pooling1d_5 | input: | (None, 48, 32)
MaxPooling1D | output: | (None, 24, 32)
conv1d_6 | input: | (None, 24, 32)
Conv1D | output: | (None, 22, 64)
max_pooling1d_6 | input: | (None, 22, 64)
MaxPooling1D | output: | (None, 11, 64)
flatten_3 | input: | (None, 11, 64)
Flatten | output: | (None, 704)
dense_6 | input: | (None, 704)
Dense | output: | (None, 128)
dropout_2 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_7 | input: | (None, 128)
Dense | output: | (None, 64)
dropout_3 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_8 | input: | (None, 64)
Dense | output: | (None, 10)

Figure 30: Architecture of 2-CNN D2 A.

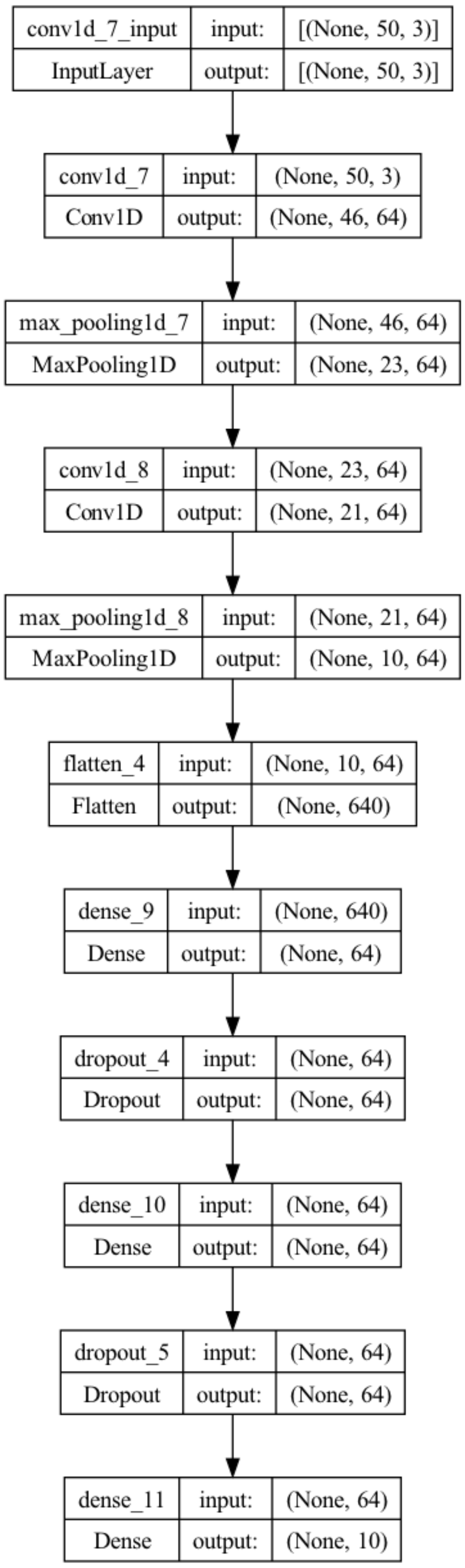

conv1d_7_input | input: | [(None, 50, 3)]
InputLayer | output: | [(None, 50, 3)]
conv1d_7 | input: | (None, 50, 3)
Conv1D | output: | (None, 46, 64)
max_pooling1d_7 | input: | (None, 46, 64)
MaxPooling1D | output: | (None, 23, 64)
conv1d_8 | input: | (None, 23, 64)
Conv1D | output: | (None, 21, 64)
max_pooling1d_8 | input: | (None, 21, 64)
MaxPooling1D | output: | (None, 10, 64)
flatten_4 | input: | (None, 10, 64)
Flatten | output: | (None, 640)
dense_9 | input: | (None, 640)
Dense | output: | (None, 64)
dropout_4 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_10 | input: | (None, 64)
Dense | output: | (None, 64)
dropout_5 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_11 | input: | (None, 64)
Dense | output: | (None, 10)

Figure 31: Architecture of 2-CNN D2 B.

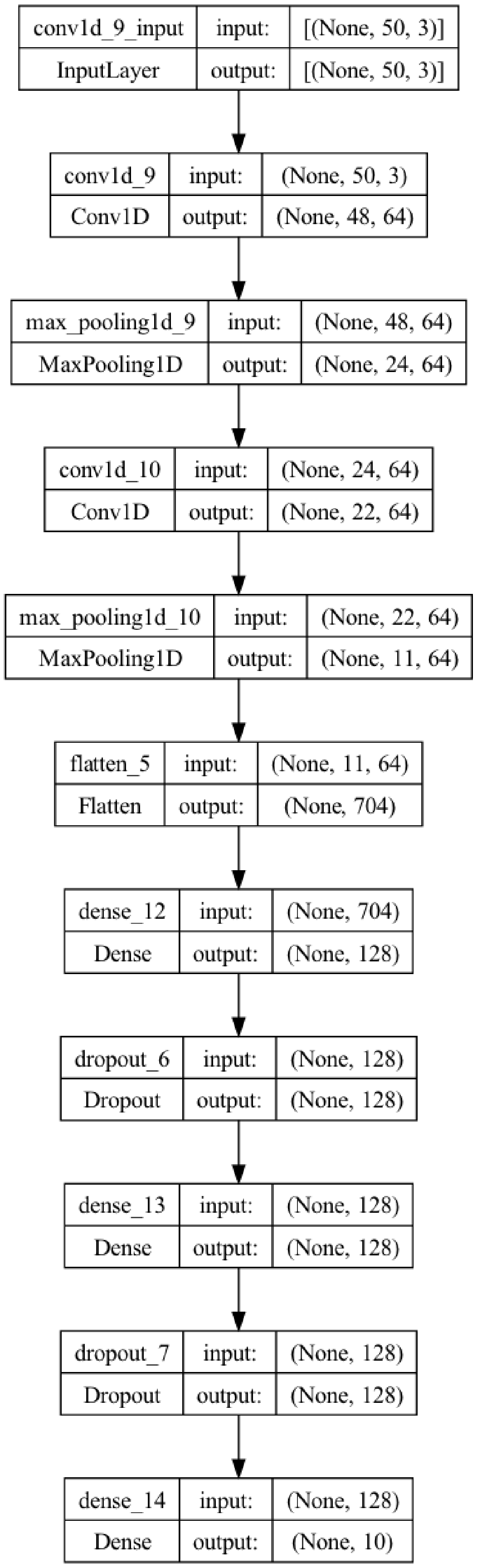
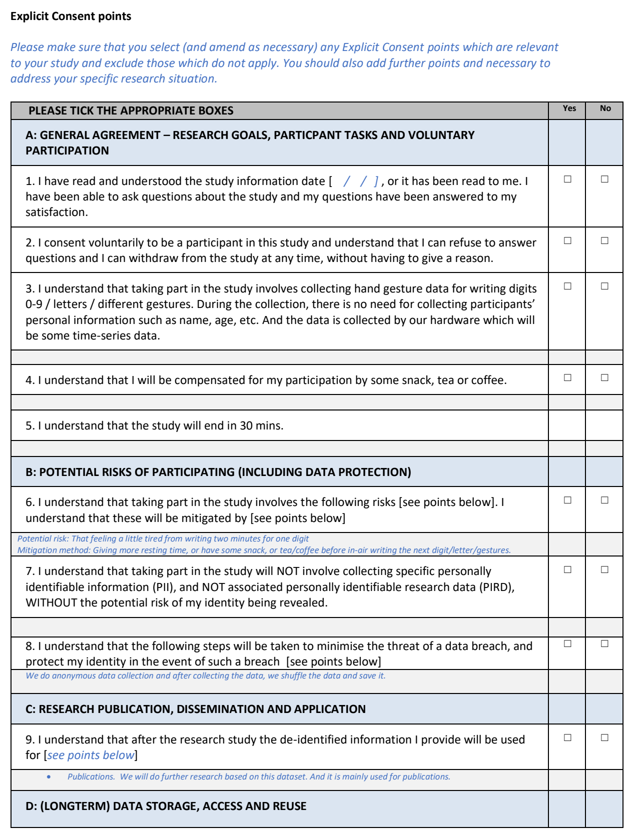

conv1d_9_input | input: | [(None, 50, 3)]
InputLayer | output: | [(None, 50, 3)]
conv1d_9 | input: | (None, 50, 3)
Conv1D | output: | (None, 48, 64)
max_pooling1d_9 | input: | (None, 48, 64)
MaxPooling1D | output: | (None, 24, 64)
conv1d_10 | input: | (None, 24, 64)
Conv1D | output: | (None, 22, 64)
max_pooling1d_10 | input: | (None, 22, 64)
MaxPooling1D | output: | (None, 11, 64)
flatten_5 | input: | (None, 11, 64)
Flatten | output: | (None, 704)
dense_12 | input: | (None, 704)
Dense | output: | (None, 128)
dropout_6 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_13 | input: | (None, 128)
Dense | output: | (None, 128)
dropout_7 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_14 | input: | (None, 128)
Dense | output: | (None, 10)

Figure 32: Architecture of 2 CNN D2 XL.

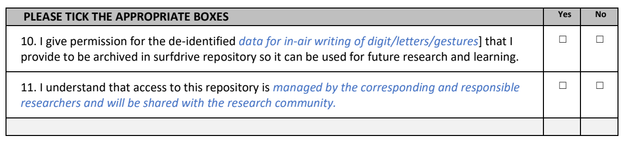
H Consent Form



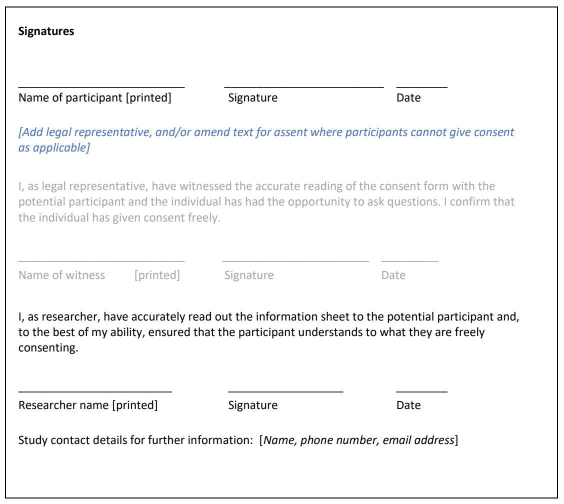
Explicit Consent points

Please make sure that you select (and amend as necessary) any Explicit Consent points which are relevant to your study and exclude those which do not apply. You should also add further points and necessary to address your specific research situation.

PLEASE TICK THE APPROPRIATE BOXES	Yes	No
A: GENERAL AGREEMENT – RESEARCH GOALS, PARTICPANT TASKS AND VOLUNTARY PARTICIPATION		
1. I have read and understood the study information date [*/ /*] , or it has been read to me. I have been able to ask questions about the study and my questions have been answered to my satisfaction.	☐	☐
2. I consent voluntarily to be a participant in this study and understand that I can refuse to answer questions and I can withdraw from the study at any time, without having to give a reason.	☐	☐
3. I understand that taking part in the study involves collecting hand gesture data for writing digits 0-9 / letters / different gestures. During the collection, there is no need for collecting participants' personal information such as name, age, etc. And the data is collected by our hardware which will be some time-series data.	☐	☐
4. I understand that I will be compensated for my participation by some snack, tea or coffee.	☐	☐
5. I understand that the study will end in 30 mins.		
B: POTENTIAL RISKS OF PARTICIPATING (INCLUDING DATA PROTECTION)		
6. I understand that taking part in the study involves the following risks [see points below]. I understand that these will be mitigated by [see points below]	☐	☐
Potential risk: That feeling a little tired from writing two minutes for one digit *Mitigation method: Giving more resting time, or have some snack, or tea/coffee before in-air writing the next digit/letter/gestures.*		
7. I understand that taking part in the study will NOT involve collecting specific personally identifiable information (PII), and NOT associated personally identifiable research data (PIRD), WITHOUT the potential risk of my identity being revealed.	☐	☐
8. I understand that the following steps will be taken to minimise the threat of a data breach, and protect my identity in the event of such a breach [see points below]	☐	☐
We do anonymous data collection and after collecting the data, we shuffle the data and save it.		
C: RESEARCH PUBLICATION, DISSEMINATION AND APPLICATION		
9. I understand that after the research study the de-identified information I provide will be used for [*see points below*]	☐	☐
• *Publications. We will do further research based on this dataset. And it is mainly used for publications.*		
D: (LONGTERM) DATA STORAGE, ACCESS AND REUSE		



PLEASE TICK THE APPROPRIATE BOXES	Yes	No
10. I give permission for the de-identified *data for in-air writing of digit/letters/gestures*] that I provide to be archived in surfdrive repository so it can be used for future research and learning.	☐	☐
11. I understand that access to this repository is *managed by the corresponding and responsible researchers and will be shared with the research community.*	☐	☐



Signatures

__________________________	_________________________	________
Name of participant [printed]	Signature	Date

[Add legal representative, and/or amend text for assent where participants cannot give consent as applicable]

I, as legal representative, have witnessed the accurate reading of the consent form with the potential participant and the individual has had the opportunity to ask questions. I confirm that the individual has given consent freely.

__________________________	_______________________	_________
Name of witness [printed]	Signature	Date

I, as researcher, have accurately read out the information sheet to the potential participant and, to the best of my ability, ensured that the participant understands to what they are freely consenting.

_________________________	__________________	________
Researcher name [printed]	Signature	Date

Study contact details for further information: [*Name, phone number, email address*]