A TinyML system for gesture detection using 3D pre-processed data

Sem van den Broek

Supervisors: Qing Wang, Mingkun Yang, Ran Zhu

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: Sem van den Broek
Final project course: CSE3000 Research Project
Thesis committee: Q. Wang, M. Yang, R. Zhu, R.R. Venkatesha Prasad

Abstract

Visible light sensing is a field of research that creates new possibilities for human-computer interaction. This research shows the viability of designing a system for detecting hand gestures using a cost-effective detection circuit employing 3 light-sensitive photodiodes. The way this research shows viability is by developing a machine-learning model that works on 3D-structured sensor data that is able to distinguish 10 different gestures and deploying the model on a standalone Arduino Nano 33 BLE microcontroller controlling the system. Using a combination of Convolutional Neural Networks and Recurrent Neural Networks it is possible to deploy a model called ConvLSTM-128 that achieves an accuracy of 70% on a dataset of limited size. This research acknowledges that the achieved accuracy is not suitable for real-world use, but concludes by outlining steps that could help future research in increasing the accuracy. Furthermore, an analysis of the 10 gestures shows that in order to improve accuracy, the way some gestures are performed might need alteration. Finally, a model size of around 140Kb and an inference time of 660ms show that this model is compact and fast enough to be deployed in real-world applications.

1 Introduction

Touchless interfaces have the potential to augment or replace the current-day human input devices we encounter. During the COVID-19 pandemic, new demand for touchless human-computer interfaces has risen because they make interaction with public infrastructure more hygienic. Furthermore, they can also make interaction possible in a more natural way [12]. For example, replacing or augmenting the physical buttons that operate a vending machine with a touchless interface, has a hygienic advantage because physical buttons no longer need to be directly pressed.

Direct input of values into a system is usually done with numbers and characters, however, another important way for humans to interact with systems is through the use of gestures. Gestures can range from large hand movements to intricately small finger patterns and often provide a way to interact with systems in a more natural way. For example, navigating to the next page of a slideshow can be done with a swipe-left gesture and controlling the volume of music playback could be done with a rotating hand movement in the desired direction.

In 2022, an effort was made by 5 Computer Science students to create a system for detecting gestures using an array of OPT101 photodiodes on a custom PCB, controlled by an Arduino Nano 33 BLE microcontroller. Machine learning was identified as a possible solution to perform gesture recognition, as it is a suitable option for when lots of varying inputs need to be classified into the correct gesture. Convolutional Neural Networks (CNNs) have been proposed in order to detect gestures based on a 2D representation of an input sample, meaning that the input was transformed into an image on which pattern recognition was performed [10]. Recurrent Neural Networks (RNNs) have also been identified as a viable option by [8], as they are well suited for classifying sequential (time-series) data. However, the constructed models were not yet able to run on resource-constrained microcontrollers.

This research will aim to improve last year's accuracy performance. Furthermore, it will use new developments in the Tensorflow Lite for Microcontrollers library[1] (TFLM) that allow the deployment of RNNs on platforms such as the Arduino Nano 33 BLE. The research question is therefore: *"How to perform gesture detection on a testbed with one Arduino Nano 33 BLE and three OPT101 photodiodes using machine learning based on 3-D pre-processed data?"*

The process of creating the system involves a number of sub-questions:

- How to collect new training data?
- How to shape data in a suitable way for the chosen model run on?
- What type of model can be used?
- What are the hyperparameters of the model?
- What is the accuracy of the model?
- What is the performance of the model on a microcontroller?

Research into other subjects like 2-D pre-processed data, as well as digit and character recognition based on the same platform, is being performed by other Computer Science Bachelor students. This paper will refer to the group as 'the research group' and mention their work whenever research effort has been shared, for example when new data was collected.

This paper will include the following sections: Firstly, background information about the types of machine learning models used in the research is given in section 2, as well as introducing previous work on the detection system created in 2022. In section 3, the paper includes the methodology for the research, describing how data collection and processing will be structured. This paper's main contribution will be discussed in section 4. Research setup and results will be discussed in section 5. Discussion of the research and the potential difficulties encountered during model development will be done in section 6. Section 7 states how this research was performed in a responsible way. Finally, the paper concludes and stipulates any further possible improvements on the system in section 8.

2 Background

Until recently, computer programs were created in a way where each input resulted in a predefined output of the program. Every circumstance that the program would ever encounter, would result in a corresponding action that it should take [16]. Creating this ruleset for when specific actions should be performed, was entirely up to the programmer. This strategy works for programs and systems for which every conceivable input can reasonably be considered during development. However, there are some tasks we like to automate for which human ingenuity is not enough. This is where *Machine Learning*, a field that studies algorithms that can learn from experience, comes in [16].

The photodiodes in the gesture detection system all produce an output signal that reflects the amount of light they currently measure. Collecting the data during a pre-set sample time then produces a graph that looks like Figure 1. There are three separate graphs for each of the photodiodes present on the detection system. The vertical axis displays the normalized light intensity observed, measured as 10-bit values ranging from 0 to 1023 by the Arduino. The horizontal axis displays the amount of samples collected, in this case 100

[1] https://www.tensorflow.org/lite/microcontrollers

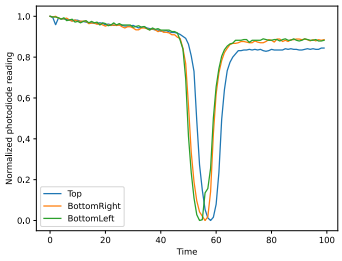


Figure 1: Example of swipe-up measurement

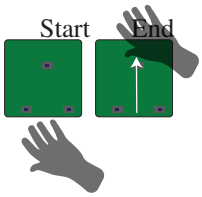


Figure 2: Example of swipe-up gesture

samples in 1 second. More information about sample collection is in section 3.

The gesture that was recorded is the ”swipe-up” gesture, in which the user moves their hand from the bottom of the detection system to the top, as displayed in Figure 2. Due to the analog nature of this signal, some noise will always be present in the collected measurement. Furthermore, humans cannot reproduce the same gesture with exactly the same characteristics in front of the detection system multiple times. A user might perform the gesture faster or slower than normal, or they might move their hand slightly relative to the detection system, resulting in differing measurements. Finally, changing lighting conditions also result in different measurement intensities. These features are the reason we cannot devise a standard ”one-size-fits-all” algorithm for gesture detection, the difference between measurements is simply too large. Therefore, we will employ machine learning to teach the system to recognize gestures based on training data collected for this research.

2.1 Previous work

In 2022, research into touchless gesture recognition has been performed by 5 Computer Science students at the TU Delft for completing their Bachelor research project:

Stijn van de Water conducted research on creating the custom printed circuit board (PCB) in order to house and control three photodiodes with which intensity changes in ambient light can be measured [13]. Furthermore, he created a circuit with which the sensitivity of the photodiodes can be altered so that they can work in varying lighting conditions. Additionally, the output signal from the photodiodes is filtered with a small filtering circuit. The PCB interfaces with one Arduino Nano 33 BLE, on which further processing was performed by other members of the 2022 research group. The custom PCB will be referred to as the testbed in this research paper.

Dimitar Barantiev has performed research into the design of the software for the gesture input receiver [3]. His findings have identified possible ways of detecting gesture starts and processing the signal using a mix of FFT, maximum division and Linear Interpolation.

Femi Akadiri constructed a dataset to aid in the training of models used for gesture recognition [2]. His research explored the impact certain features like ambient light, hand size and sampling rate have on the final dataset. He also constructed tooling with which a dataset can easily be recorded, which will be used for this research. For the results of this paper, the dataset by Akadiri will not be used.

William Narchi researched how to develop a CNN in order to perform gesture detection [10]. His research identified a suitable CNN and a corresponding way to deploy the model on an Arduino Nano 33 BLE microcontroller in a way that the model is sufficiently compact and low-latency for real-time operation.

Matthew Lipski researched the possibilities of recognizing gestures using RNNs [8]. His research has shown that CNN-LSTMs have produced a relatively high validation accuracy but also noted that further work is required for improving the accuracy. This research paper will be mainly focused on improving the models identified by Lipski, as well as deploying them to an actual microcontroller.

2.2 Recurrent or Convolutional

Until recently, most machine learning techniques have used shallow-structured architectures [6]. This means that the data is modelled very simply and structured with a shallow representation. However, these models encounter difficulties when dealing with real-world applications involving natural signals like the detection of gestures [6]. As seen in Figure 1, data from the detection system is inherently sequential. This means that when all data points, or timeframes, are shuffled, the data loses its entire context. For this reason, using a neural network that is able to take the context into consideration is vital. For this reason, numerous research into a field called *Deep Learning* has been performed to explore the complex structures that natural signals contain [6].

Two important developments in the field of Deep Learning are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Both types of neural networks are used for classification of time-series data [16], however, RNNs have proved difficult to train, containing millions of parameters, in the past [9]. Classical RNN models have had troubles with *vanishing* and *exploding* gradients that occur when backpropagating errors across many time steps [9]. Having the gradient ”explode” results in internal model weights oscillating, while gradients ”vanishing” mean that training the model takes a long time, or the model does not train at all [7]. The LSTM (Long Short-Term Memory) model introduced by Hochreiter and Schmidhuber [7] was created in order to overcome the problem of vanishing gradients by implementing LSTM as a memory cell with an efficient algorithm enforcing constant error flow through the internal states of the cells. Model structures using LSTM will become the most important models in this research, as they have been identified as being suitable for classifying sequential data [16].

3 Methodology

This section will explain the methods used to approach the research question. Figure 3 presents an overview of the workflow that is used for this research. Data Collection, Model Development & Training and Model Validation form a cycle as this is a continuous process during the research that is refined each time more training data was available.

3.1 Data collection

Part of training a machine learning model is training it on a varied set of data so that it can work on a representative dataset based on real-world usage. For this reason, data was collected from multiple participants in varying environments. Because this research collected data from human participants, the responsible professor contacted the TU Delft Human Research Ethics Committee[2] (HREC). Based on a risk analysis,

[2]https://www.tudelft.nl/en/about-tu-delft/strategy/integrity-policy/human-research-ethics

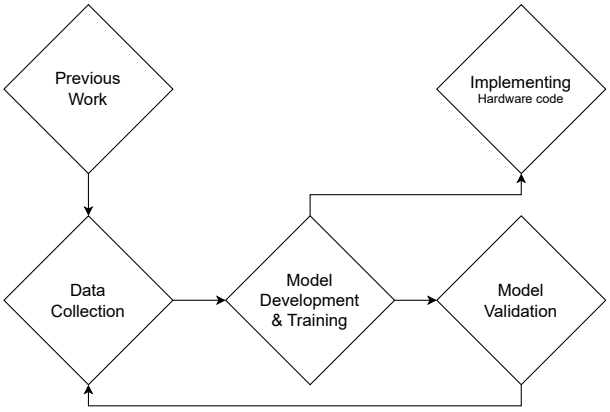


Figure 3: Outline of methodology

the committee has deemed this research as low-risk and provided the research group with approval to conduct research with human participants. An informed-consent form has been created for participants to consent to the research, and the filled forms are stored on GDPR-compliant media and are not part of the research data.

For collecting the data, the same PCB as last year was used. The Arduino on the PCB contains a specialized version of the detection code that outputs the three values of the photodiodes at a set sample rate over serial communication, which was then collected by a custom tool on a personal computer. The programs for this were all used from last year's research, however, the research group quickly found out that we would prefer to have more customizability for the software. Tweaks to the software that have been made can be found in the shared repository.[3] Some notable improvements include:

- The control software supports all types of gestures, digits and characters necessary for the research group.
- The control software is able to change the sampling rate on the microcontroller.
- The control software shows a graph of collected measurements and allows for removing individual measurements when they are incorrect.

Some guidelines for using the board were set for ensuring comparable measurements for all members of the research group:

- Ensure the PCB lays flat on the table.
- Ensure that there are no super bright lights in the environment.
- Ensure that the room is well-lit.
- Recalibrate the photodiode sensitivity when lighting conditions change.
- Ensure a consistent distance between the PCB and the participants' hand, around 6 to 12 centimetres.

Each participant created an anonymized dataset containing the gestures as displayed in Appendix A (as well as digits and characters for other members of the research group). A sample rate of 100 Hz was used and a sample duration of $1s$ was employed. Participants were asked to perform the gesture starting with their hand on the table and also ending with their hand away from the sensors, this was done to ensure a clean start and end period in each gesture. An exception was made for the *Rotate CW & CCW* gestures, for which participants were asked to start performing a circular motion above the board, after which the researcher clicked on the collection button numerous times. Each gesture was measured at least 5 times in succession. At first, the research focused on getting equal amounts of data on left and right-handed usage, however, most of the participants were right-handed so the research switched over to letting candidates pick their preferred hand, more about this in section 7.

[3] https://github.com/arnedebeer/CSE3000-DataCollection.git

3.2 Model development

The development of models involves processing the data to an appropriate form before inputting it in a model. Preprocessing can improve the quality of the signal so that the model can more easily train on important characteristics instead of unwanted signal differences [3]. After preprocessing, the layers and hyperparameters of the model are defined and compiled using Keras [5]. This research analyzes two general types of models, one purely based on an LSTM layer together with some modifications, and variations on another model named ConvLSTM by [8]. Model development and deployment code can be found in the GitHub repository [14].

Preprocessing

The final data processing pipeline utilizes a normalization step. With a sample rate of 100 Hz, each measurement contains 100 measurements over 1 second. The normalization step converts each of the 3 10-bit photodiode readings separately into a normalized 32-bit floating point value ranging from 0 to 1, this is done to account for differences in light intensity readings across measurements. A reshaping step was considered but did not make it into the final model, as the main structure revolves around the shape that is accepted by the primary LSTM layer, which coïncides with the structure the data is already being collected in. Additionally, the ConvLSTM models do internal reshaping that splits the input into multiple "frames" that can be fed to the first convolutional layer so they accept the same input as the plain LSTM models. This means that 3D-structured data can be fed to the model as-is.

Model structure

The model structure is primarily composed of an LSTM layer that accepts input in the following format: `[batch, timesteps, feature]`. The predictions layer — the layer that provides the final output — is a generic densely-connected layer containing 10 nodes, one node for each of the gestures we wish to classify. Multiple additions were conditionally added in the code so that multiple model structures could easily be tested. For example A dense layer could be added before the predictions layer to see if that would aid the LSTM layer in forming predictions. Furthermore, the LSTM layer received the option to be embedded in a Bidirectional layer so that the time-series data will be looked at sequentially beginning from the start and beginning from the end. The layer structure along with most hyperparameters of the ConvLSTM models remained the same as proposed by [8] as that proved to yield the best results. The most important variations in the hyperparameters of the models will be shown in section 5.

3.3 Model validation

The data collected is grouped per candidate. This is done because each candidate also reflects a different environment, in which the system should be able to work. For this reason, we can only validate the model based on data it has never seen

before, ideally from environments that are new. For this reason, the entire dataset is split between groups of candidates, instead of randomly shuffled as shown in Figure 4. The reason this is done, is to have an accuracy rating based on samples that the model has never seen before. This way, when the model is heavily overfit on the training data it cannot perform as well when validating it with the test set, resulting in more accurate performance metrics.

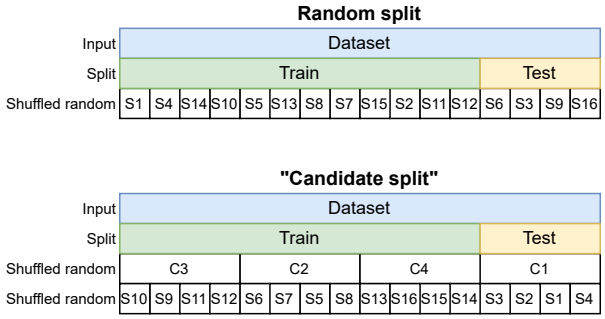


Figure 4: Method to split train and test data

K-Fold Cross-Validation

To estimate reproducible model accuracy statistics, K-Fold Cross-Validation is used [4]. The k unique groups that are chosen will be split between candidates. For each unique group, the model is fitted with a set of data where the group itself is removed from the training set. The model is then evaluated on the group that was removed from the training set. Finally, the accuracy, loss and confusion are stored, the model is discarded, and the cross-validation continues with the next group. For the candidate models, a 5-fold and a 10-fold will be performed.

3.4 Model deployment

Last year, [8] used TFLM version 2.4, which had no support for certain RNN tensor operations. This research deployed the latest version of TFLM built for the Cortex M4 architecture on the Arduino microcontroller, which has support for unidirectional LSTM operators, as well as all previously used Convolutional operators. Necessary libraries are added to the repository as git submodules. The project was statically linked with the manually compiled library, however, in more recent commits this was changed over to a non-statically linked version of the library in order to figure out compatibility issues.

Detection system

Sensor data is collected with a sliding window buffer, this means that we always store the last 100 frames in the buffer. Each ”tick” of the microcontroller shifts the buffer and performs a new measurement, at a rate of 100 Hz. Furthermore, an algorithm for detecting the start of a gesture is in place. This algorithm is adjustable with two parameters, the `activation_threshold` and the `window_size`. Every 200 ticks the microcontroller generates an average of the sensor input it has seen for the last 100 frames, and sets it as the `activation_threshold`. The threshold is thus dynamically updated to changing lighting conditions. For each tick, the microcontroller checks if the last `window_size`, 10 for example, frames are below the threshold. If they are, the controller waits for `sample_size / 2` ticks before starting the preprocessing pipeline, which finally results in the controller running inference on the model. The reason it has to wait 50 more ticks, is to wait for the gesture to complete so that the sliding window buffer contains the gesture roughly in the middle of the measurement. Essentially, it calculates whether it has seen a dip in light intensity over the last few frames.

When the system detects a drop in light intensity, and therefore a possible gesture, the buffer is filled up to 100 measurements. Then, the data is normalized and inference is run. Inference means that the model takes the input sample and predicts an output based on the sample and what it has learned during training. Model prediction output is a list of weights, of which the index of the largest value in the list is chosen to be the actual output. The output of the inference is transmitted to an attached computer over the serial connection.

Quantization

A model with a lot of trained parameters can become quite large, which means that allocating the static memory and the tensor memory on the microcontroller also takes up more space [15]. The parameters and weights of the internal model structures are stored as 32-bit floating point numbers, but they can also be converted to 8-bit integers without losing too much precision [1]. This is called quantization and has the potential to reduce the space and runtime requirements of the model drastically. Quantization is a built-in feature of the TensorFlow-lite library.

Other Microcontrollers

Verifying whether inference and model output are correct on the microcontroller can be done by loading the TensorFlow-Lite model in Python, and running inference with the same sample as the microcontroller. Both the microcontroller and the Python interpreter should then output the same value. During this research, running quantized models on the Arduino Nano 33 BLE did not produce correct outputs for inference. For this reason, this research also performed inference analysis on an ESP32-based development kit, containing the ESP32 microcontroller that has double the memory (512Kb) of the Arduino Nano 33 BLE (256Kb) in order to determine if another platform is able to correctly run the models. Inference times of both microcontrollers have therefore been included in the results, while their output correctness is discussed in section 6.

4 Implementation

This research will analyze the differences between several variants of recurrent LSTM models. In order to analyze the performance, a dataset containing around 1400 right-handed gestures and 900 left-handed gestures was constructed together with fellow students in the research group. Furthermore, data processing, model training and model optimization code in Python has been produced. Finally, model deployment for two different microcontroller architectures has been constructed with code in C and C++, containing preprocessing, a gesture detection algorithm and inference using the TFLM library. Final code used for aforementioned parts of the research can be found in the same repository as the model structure code [14].

5 Results

For baseline measurements, variations on the best performing RNN model by [8] called ConvLSTM are used. Furthermore, some variations of simple LSTM models are analyzed. In summary, this section includes the metrics of the following models:

- ConvLSTM x-unit (with $x \in [16, 32, 64, 128]$)

- LSTM x-unit (with $x \in [16, 32, 64, 128]$)
- LSTM 64-unit + Dense (D.) x-unit (with $x \in [64, 128]$)
- LSTM 64-unit Bidirectional (Bd.) + Dense (D.) 128-unit

The reason we look at the LSTM unit size separately from the extra added features like Dense and Bidirectional is to evaluate whether they are able to actually boost performance for an already "maxed-out" LSTM layer. Furthermore, adding a Dense layer greatly increases the size of the model as lots of parameters have to be stored. The metrics that will be reported for each model are the following:

- 5-fold accuracy & loss
- 10-fold accuracy & loss
- Original & Quantized model size
- Quantized inference time on Arduino Nano 33 BLE & ESP32

ConvLSTM models are trained with 60 epochs and plain LSTM models have been trained with 125 epochs, using the Adam optimizer. The learning rate is left to the default setting, which is 0.001. Models are quantified using the TensorFlow-lite library optimizing with default settings.

Inference time is measured on the microcontroller with timers that output the time in milliseconds the `invoke()` function took to complete, preprocessing time is therefore not measured. Results regarding the accuracy estimation based on K-Fold cross-validation can be found in Table 1, model sizes are shown in Table 2 and inference times are shown in Table 3. Finally, a confusion matrix for the best performing model can be found in Figure 5 to show what gestures are easier to detect with an RNN and which ones are more challenging.

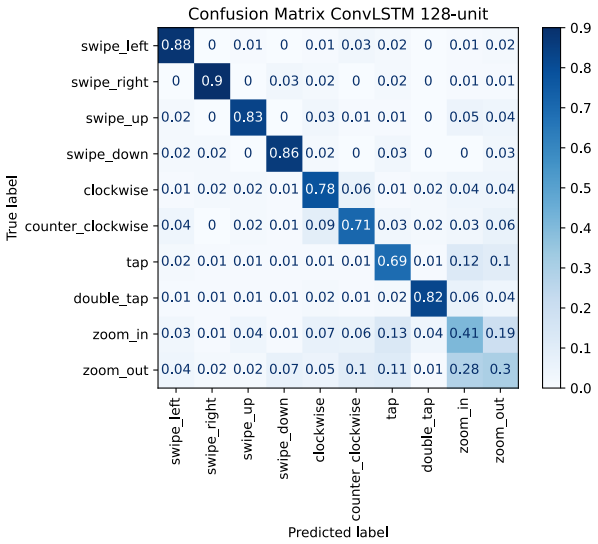


Figure 5: Confusion matrix of ConvLSTM 128-unit model based on 10 folds of cross validation

6 Discussion

From Table 1 we can conclude that variants of the ConvLSTM model outperform all variants of more simple LSTM models. This implies that incorporating some convolutional neural network layers in the model is able to boost accuracy by a small amount for the same data. This can be due to convolutional layers being able to extract the specific features that are important for classification in the samples.

	5-fold		10-fold	
	Acc.	Loss	Acc.	Loss
ConvLSTM 16	65.78%	1.09	67.31%	1.05
ConvLSTM 32	69.36%	1.12	69.66%	1.06
ConvLSTM 64	70.42%	1.29	69.98%	1.32
ConvLSTM 128	70.16%	1.52	70.72%	1.48
LSTM 16	63.72%	1.71	65.71%	1.50
LSTM 32	64.47%	2.40	66.32%	2.19
LSTM 64	65.60%	2.78	67.31%	2.52
LSTM 128	66.68%	2.55	68.12%	2.43
LSTM 64 + D. 64	67.98%	2.36	69.23%	1.97
LSTM 64 + D. 128	69.34%	2.43	68.6%	2.22
LSTM Bd. 64	66.09%	2.51	66.74%	2.42
LSTM Bd. 64 + D. 128	66.92%	2.19	69.29%	2.22

Table 1: K-fold metrics

	Size	
	Original	Quantized
ConvLSTM 16	391.0Kb	112.6Kb
ConvLSTM 32	420.7Kb	120.2Kb
ConvLSTM 64	504.7Kb	141.7Kb
ConvLSTM 128	770.9Kb	209.1Kb
LSTM 16	72.3Kb	22.0Kb
LSTM 32	149.7Kb	41.5Kb
LSTM 64	329.2Kb	86.8Kb
LSTM 128	786.4Kb	201.9Kb
LSTM 64 + D. 64	1714.8Kb	433.8Kb
LSTM 64 + D. 128	3356.0Kb	844.32Kb
LSTM Bd. 64	657.4Kb	172.1Kb
LSTM Bd. 64 + D. 128	6705.1Kb	1684.8Kb

Table 2: Trained model sizes

	Inference speed	
	Arduino	ESP32
ConvLSTM 16	369.6ms	826.8ms
ConvLSTM 32	499.4ms	844.5ms
ConvLSTM 64	658.5ms	919.0ms
ConvLSTM 128	1141.9ms	1269.9ms
LSTM 16	97.1ms	33.2ms
LSTM 32	253.2ms	80.8ms
LSTM 64	777.6ms	237.1ms
LSTM 128	2677.3ms	1670.8ms
LSTM 64 + D. 64	805.2ms	306.6ms
LSTM 64 + D. 128	DNF	398.2ms
LSTM Bd. 64	DNR	DNR
LSTM Bd. 64 + D. 128	DNF	DNR

Table 3: Quantized inference time
DNR: Did not run **DNF:** Did not fit

Furthermore, we can see that having a deeply connected layer of 128 units (LSTM 64 + D. 128) is able to help the model achieve a little higher accuracy, however, this comes at a great model size penalty as can be seen in Table 2 so it is not very feasible for resource-constrained microcontrollers.

Additionally, in Figure 5 we can see the confusion matrix of the ConvLSTM 128-unit model based on 10 folds of cross-validation. In a confusion matrix, the amount of prediction mistakes a model makes during validation is recorded as well as what the model actually predicted. This results in an image in which gestures that are often predicted correctly have a high value close to 1 on the diagonal, but gestures that the model can predict less accurately will have a lower value. It is also possible to see with what other gestures a model is most confused.

A very noticeable example from the confusion matrix is the

confusion between zoom-in and zoom-out gestures. Zoom-in and zoom-out are often confused with one other, but it is important to note that these two gestures also share some confusion with the tap gesture. This can be explained by the motion that is made before and after making a gesture, in which the participant moves their entire hand into and out of the detector's field of view. The actual zoom gesture only makes up a fraction of the total input features that the model sees, making it easily confused with any other gesture that is based on the user moving their hand in and out of the field of view. This can be explained by looking at the similarities in measurements when manually observed in Figure 6, however, it should be mentioned that not all zoom measurements look like tap measurements. Based on the confusion matrix, we can also conclude that the swipe gestures are the most easily recognized of the 10-classification.

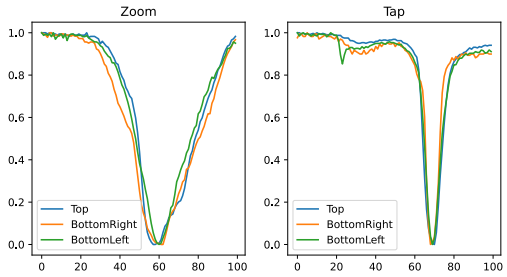


Figure 6: Two hand-picked samples showing the similarities of Tap and Zoom gestures

Deployment

In order to get inference working natively on the microcontroller, the model needs to be converted to a TensorFlow-lite model. Furthermore, the model then also needs to fit inside the limited flash storage on the microcontroller. The largest models in Table 2 will not fit in the standard flash memory in common microcontrollers, however, they might be able to fit on microcontrollers with additional external flash memory. Finally, the microcontroller also needs space in memory to allocate space for all tensors. An edge case in which a model fits but does not run would be a possibility, however, with the models tested this behaviour was not encountered. Models that do not fit are referred to as "Did Not Fit" (DNF) in the table, and models that fit but are unable to run are referred to as "Did Not Run" (DNR). The models containing a bidirectional LSTM layer were not able to run because of a missing operator in TFLM. The LSTM 64 + D. 128 model was able to fit on the ESP32 and not on the Arduino, this is due to the greatly expanded flash storage that was available for that controller.

Furthermore, LSTM models are a recent development in the TFLM library and therefore not everything works as smoothly as planned. For example: Running an LSTM model with only 4 units on both microcontrollers results in a working system, however, when a quantized version is deployed, the output is wrong. Output vectors from the model are always the same, regardless of input. This research has not been able to establish a solution for this behaviour, but the cause is probably found in the way models are quantized, as executing the quantized model interpreters in Python also results in problems. For this reason, the inference speeds in Table 3 on both microcontrollers are purely based on a runtime analysis of the TFLM library's `invoke()` method with the applied model, regardless of its output.

Some notable results in Table 3 are that the microcontrollers both seem to perform differently in scenarios with and without convolution. The ConvLSTM models run slightly faster on the Arduino, while the more simple LSTM-only models run faster on the ESP32. This could be explained by looking at their underlying implementations, as both microcontrollers have incorporated optimized functions for neural networks, implemented for their native architectures. Adding more deeply connected layers has a small performance impact but a large model-size impact. In practice, all inference times lower than 1000ms should be acceptable as this still feels reasonably fast.

7 Responsible Research

This research uses data collected from human participants in a real-world environment. For this reason, any possible bias in the data could impact later system usage in a production environment. For example, the dataset consists of 1400 right-handed gestures and 900 left-handed gestures, which means that the neural network has performed less training on left-handed data. This could result in a system that is less responsive and less accurate for left-hand users which is something this research initially aimed to avoid. To achieve this, right-handed participants were asked to also perform gestures with their left hand in order to gather more data. This results in a less biased dataset, however, it can be argued that data collection with participants using their non-dominant left hand, results in data that does not accurately represent actual left-handed users. Some real left-handed participants are also included in the dataset. The final dataset, however, still contains more right-handed than left-handed measurements. As the ratio between right- and left-handed people in the world is around 90/10 [11], this research deems the achieved bias in the dataset acceptable.

Another possible bias this dataset might include is individual sample bias. Gestures that resulted in hard-to-distinguish features like rotation and zoom gestures have been collected more often than simpler gestures like swiping. While the baseline of gesture collection per participant was 5 measurements per gesture, more were sometimes recorded and not deleted afterwards. As we see in section 6, these gestures are still hard to classify, so having more measurements for these specific classes is justified.

Furthermore, during data collection and analysis, this research selectively removed samples that are not representative of the expected sensor input in a final product. This means that in cases where a participant was too early or too late in performing the gestures, the resulting measurements have been removed and performed again. This was done because this research trains the used models with limited data, meaning that samples that were not up to the specifications could impact final performance.

Finally, an ethical implication of this research can be that the gesture detection system might not be as easy to use for all users and that it could lead to frustration while using the system in a production environment due to low accuracy. For this reason, this research recommends having a backup option for human-computer interaction, physical buttons for example, in real-world scenarios.

8 Conclusions and Future Work

This research proves the feasibility of running a standalone gesture detection system by constructing a workflow for recording, processing and deploying models for gesture de-

tection. With a final accuracy of around 70%, the used models are not yet ready for real-world use. Especially when user input has to result in a reliable action performed by the detection system. Every gesture now has more than 1/4th chance of being predicted wrongly, which would be a major cause for frustration in production deployments of the system. Model accuracy is only marginally improved by employing more complicated models and substantial accuracy improvements can only be made by increasing the size of the dataset.

Recurrent Neural Networks have certainly shown their capabilities being almost on-par with Convolutional Neural Networks in this research, so they are worth exploring more once a larger dataset is available. For improving the dataset, more criteria about lighting conditions could be set as this research was not able to provide a fixed environment in which all measurements were performed. Dataset augmentation can also be done, this means that for each collected sample some variant samples can be automatically generated, possibly aiding the model in becoming more robust.

The 10-classification for gestures is feasible, however, the zoom gestures are hard to distinguish. The easiest solution to this problem could be swapping out zoom-in and zoom-out for other kinds of gestures that are more distinguishable. Other solutions can also be thought of. For example, extra hardware on the board can be added to provide height information of the hand and fingers. Not only can this aid in detecting the start of a gesture, it could also provide more features for gestures in which the height of the hand above the detection system changes, like tap and zoom.

Additionally, more research into deploying models on various microcontrollers is required. As LSTM models are relatively new in TFLM they still require some workarounds to get fully working. Furthermore, model quantization shows unwanted behaviour which can be further analyzed. As inference times are still relatively acceptable for deeply connected layers, experimentation can be done with microcontrollers that offer lots of flash storage in order to store large but simple models.

References

[1] Tensorflow: Model optimization. https://www.tensorflow.org/lite/performance/model_optimization, 10 2021.

[2] Femi Akadiri. Constructing a dataset for gesture recognition using ambient light, 6 2022.

[3] Dimitar Barantiev. Designing a software receiver for gesture recognition with ambient light, 6 2022.

[4] Jason Brownlee. A gentle introduction to k-fold cross-validation. https://machinelearningmastery.com/k-fold-cross-validation/, 5 2018.

[5] François Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

[6] Li Deng. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7:197–387, 2014.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[8] Matthew Lipski. Hand gesture recognition on arduino using recurrent neural networks and ambient light, 6 2022.

[9] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. 5 2015.

[10] William Narchi. Recognising gestures using ambient light and convolutional neural networks: Adapting convolutional neural networks for gesture recognition on resource-constrained microcontrollers, 6 2022.

[11] Marietta Papadatou-Pastou, Eleni Ntolka, Judith Schmitz, Maryanne Martin, Marcus R. Munafò, Sebastian Ocklenburg, and Silvia Paracchini. Human handedness: A meta-analysis. *Psychological Bulletin*, 146:481–524, 6 2020.

[12] Prianka Srinivasan. In a touchless world, how will you embrace technology?, 12 2022.

[13] Stijn van de Water. Designing an adaptable and low-cost system for gesture recognition using visible light, 6 2022.

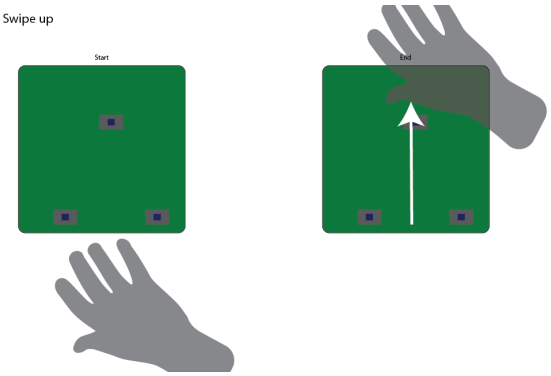
[14] Sem van den Broek. CSE3000 Gesture Detection. https://github.com/SemvandenBroek/CSE3000-gestures, 2023.

[15] Pete Warden and Daniel Situnayake. *TinyML: machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*. O'Reilly Media, Inc., 2019.

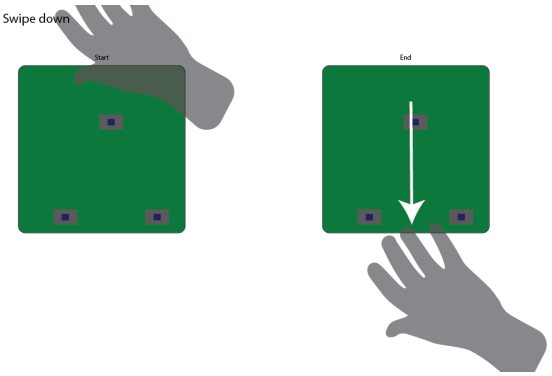
[16] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 6 2021.

A Gesture types

Swipe up

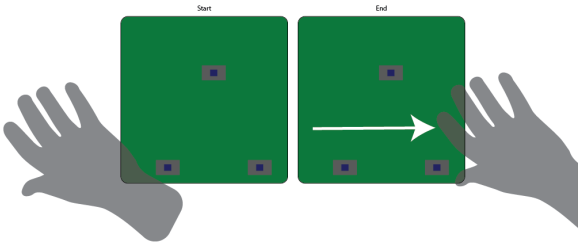


Swipe down

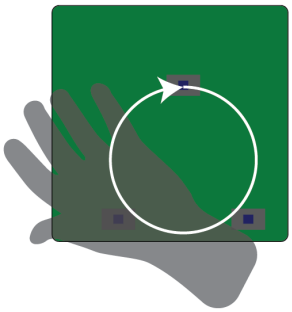


Swipe left

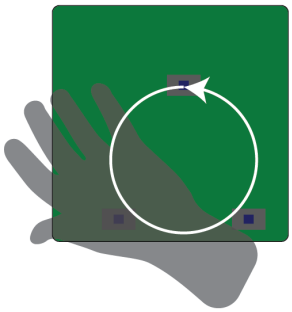
Swipe right



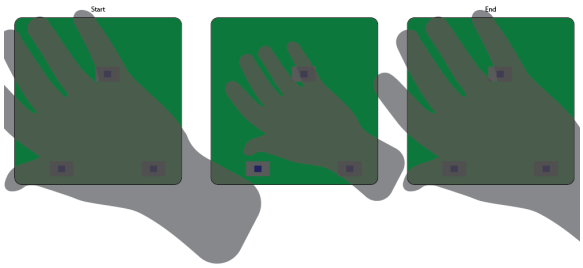
Rotate CW



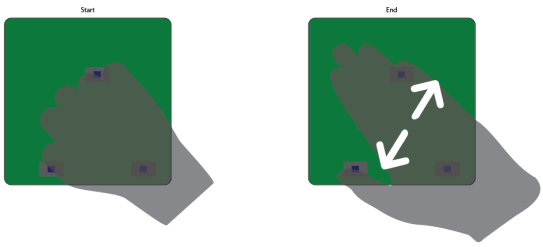
Rotate CCW



Tap (& double tap)



Zoom In



Zoom Out