Digit recognition with visible light using three photodiodes and 3D-preprocessed data

Gijs van de Linde

Supervisor(s): 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: Gijs van de Linde
Final project course: CSE3000 Research Project
Thesis committee: Qing Wang, Mingkun Yang, Ran Zhu, Dr. Ranga Rao Venkatesha Prasad

Abstract

This paper describes the feasibility of digit detection using three photodiodes and an Arduino Nano 33 BLE. This is done using a controlled lighting condition, using a bright lamp. It dives into the process of data collection, pre-processing and model selection for a recurrent neural network to do the classification of gestures into digits. Using a "ConvLSTM double conv. layer model" with 128 units we were able to achieve an average accuracy of 0.500 ± 0.091 on a 5-Fold cross-validation procedure based on data that was collected in a controlled lighting environment. While this provides a foundation for digit detection using time-series data in a controlled light environment, further suggestions are made for future improvements and expansions in this area.

1 Introduction

The COVID pandemic made humans mentally aware of the bacterial and viral traces they leave behind on objects. Though these traces on shared surfaces are rarely the cause of infection [1], they did cause "touch anxiety" among people. This was reported by therapists [2] and found in a survey by London South Bank University [3]. This caused a necessity for public devices allowing for touch-free interaction [4].

There are many solutions to this problem, like using people's smartphones as input devices or having a camera and/or an infrared detection system. These are not always feasible methods, because not everyone necessarily owns a (reliable) smartphone. To illustrate this point a study showed in 2020 in the US 85% of the people owned a smartphone [5]. It would be unacceptable to have elevators that are only operable by 85% of the population. Since user-interaction devices are everywhere it is advantageous for them to use up little energy, in contrast to infrared detection systems and full-color video cameras. This is where photodiodes and the Arduino Nano come in.

A research paper by K. Janczyk, K. Czuszynski and J. Ruminski [6] describes using a single quadrant-photodiode for digit classification. They used a mix of the MNIST data set [7] and their own captured data (with a 20% arbitral rejection of the worst samples) to train their best-performing convolutional neural network. This approach, using their best-performing setup, achieved an average accuracy of 86%. A quadrant photodiode is different from the hardware that we are using, and this paper did the inference of their model on a stationary computer, and not on a micro-controller.

There are multiple scenarios in which the ability to recognize digits from in-air gestures would be useful. An anecdotal example is to enter the designated floor one would like to take an elevator to. Another one is to use our system for inserting a PIN to withdraw money from an ATM.

In 2022 a group of Computer Science Engineering students [8–12] from the Delft University of Technology laid the foundation for an interaction device that uses ambient light to observe hand gestures. This setup is low in cost due to using only three separate photodiodes, a custom-designed PCB, cheap and widely available parts, and an Arduino Nano 33 BLE. This proves prominent due to its energy efficiency.

Last year's team yielded many useful results that can be used to further improve the recognition of certain gestures. First off, access to the hardware from early on in the project allows us to focus mainly on the data-collection and machine-learning model aspect of this project. D. Barantiev proposed potential ways to do pre-processing on the data [9]. The research on data collection, a research project conducted by F. Akadiri [10], and the software tools that came along with it proved a great inspiration source for this research paper. Furthermore, the two papers on neural networks, one on convolutional neural networks conducted by W. Narchi [11], and one on recurrent neural networks conducted by M. Lipski [12], were a valuable asset at the start of this research project, and were used as a guide in the right direction. A summary of these papers can be found in section 2.2

Current research still lacks insights into the feasibility and methods for digit detection using this specific setup. This year we aimed to improve the recognition achieved by the gesture-recognition setup created last year in the application on digits. This paper focuses on the classification of digits drawn in front of the photodiodes with a finger using a 3-dimensional approach. A 3D approach entails that we use a model that uses time as a dimension (using the fact that there is a certain order in this data that is relevant). This is further explained in section 2.1. The problem here consists mainly of collecting the data and training an appropriate model that adheres to the memory constraints of the Arduino Nano 33 BLE[13]. This paper contributes to finding an appropriate model.

This paper contributes the following things:

1. Tools and insights into collecting a representative digit gesture data set.
2. That it is possible to get to a mean accuracy of 50% on digit recognition with 3 photodiodes, though it might require an artificial light source.
3. What kind of model should be used when using 3D-preprocessed data with the setup of this research in mind.

The rest of this paper is structured in the following way: First, there is a background section, section 2, which dives into useful concepts and summarizes last year's research group's papers. Then, the methodology section, section 3 describes the methodology and why it was chosen. Following this, there is an explanation of the experimental setup and results in section 4. This is followed by section 5 about responsible research, tackling the ethical and reproducibility aspects of this paper. Then there follows a discussion in section 6, which is finally followed by a conclusion and future works section, section 7, describing important conclusions from the research performed, as well as giving guidance for future work to progress the field.

2 Background

2.1 Useful concepts

3D formatted data refers to the data being formatted in frames. This is done to make the photodiode data more like a video instead of an image. Figure 1 shows data points of photodiodes over time, divided into frames of size 3. The image approach is shown in figure 3, and described in section 2.2.

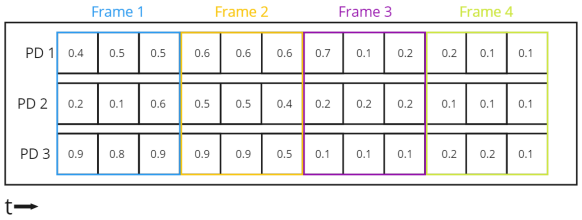

Figure 1: Visual explanation of what data in frames looks like. Data points from the photodiodes are grouped into a frame of a certain size, here frame size n=3.

Inference refers to retrieving a prediction from a fully trained model, usually after providing an unseen data sample.

Convolutional layers are layers used in deep learning models to process image-like data. They work by applying certain filters (or kernels) to a small portion of the (image) data at a time and sliding over the whole image.

A recurrent neural network is a type of artificial neural network that uses sequential data or time series data. Their distinguishing feature is their ”memory”, which keeps track of previous inputs and uses this with a certain weight in the calculations done to make the current input into output. Expansions on recurrent neural networks, RNNs, are GRU and LSTM models. These models use ’gates’ to regulate the flow of information in the network, which allows them to be even better at recognizing patterns.

A ConvLSTM is a model that combines convolutional layers and LSTM layers.

Quantization is a technique used to compress models into a smaller version. It works by lowering the precision of a model’s weights and activations. The TensorFlow Lite documentation on model optimization [14] gives the following description of how quantization works:

> ”Quantization works by reducing the precision of the numbers used to represent a model’s parameters, which by default are 32-bit floating point numbers. This results in a smaller model size and faster computation.”

A guide on how to do this procedure is written in the TensorFlow Lite documentation on post-training quantization [15].

2.2 Summary of last year’s projects

Hardware

The hardware for this project was designed last year by Stijn van de Water [8]. There are two important conclusions that are useful to keep in mind during this research project.

Firstly, for photodiode placement, the paper found experimentally that in a triangular setup, a distance of 5 cm between each photodiode is the best balance between the precision and robustness of the system. This exact hardware setup is also used during this research project and can be seen in figure 2.

Secondly, section 5.1 of this paper describes a software system for the automatic adaption of photodiodes’ sensitivity to a changing light environment. The source code for this can be found in the LightIntensityRegulator-class in the ”diode_calibration” directory of the GitHub page [1]. For data collection we use the code in this class to calibrate the sensors before collecting a sample. During inference, this class would used to do the calibration.

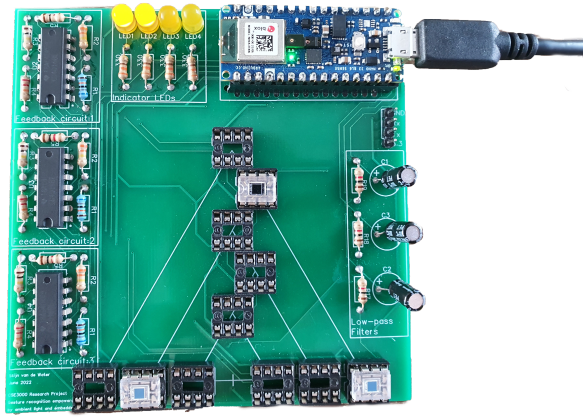

Figure 2: Hardware setup designed by Stijn van de Water

Data Collection

Last year F. Akadiri [10] wrote a paper on data collection for the project. The paper describes the creation of a data set of gestures. The data set that was created consisted of 5 samples for each candidate and 5 candidates for each possible light condition. The paper also touches upon the human interaction aspect of the gestures, but since it is talking about select gestures like tapping, zooming, swiping, etc. they are not relevant to the drawn digit gestures. Therefore nothing can be said about the feasibility of the recognition of digits as researched in this paper.

Pre-processing data

D. Barentiev created a pre-processing pipeline [9], though it was never used in the final training and inference. This

[1] https://github.com/StijnW66/CSE3000-Gesture-Recognition/blob/main/src/diode_calibration/diode_calibration.h

was because there was no time to tweak it such that it had a positive effect on the accuracy of the models. The paper describes the threshold computing algorithm they used for detecting a gesture in section 4.4. The algorithms behind the pre-processing pipeline are described in section 4.5 of the paper.

Neural Networks

As mentioned earlier, there were two papers written on neural network models used for the classification of gestures. The paper written by M. Lipski [12] stands as a basis for this paper, since it's the one that is most similar to this research's research question. It's an application of recurrent neural networks with the use of "3D-formatted data" (as described in section 2.1) to the classification of hand gestures collected on the same setup as we are using. The conclusion from the paper is that a CNN-LSTM approach worked best for the data they had collected. This means they used a model that first put input through some convolutional layers, of which the result was fed into an LSTM layer. The exact configuration can be found in section 7.1 of the paper [12].

The author found that there was no support for RNN or LSTM layers to be included in a model when using the TensorFlow Lite for Microcontrollers library to convert it for deployment on the Arduino. This problem has recently been solved since there now is support for such layers. [16]

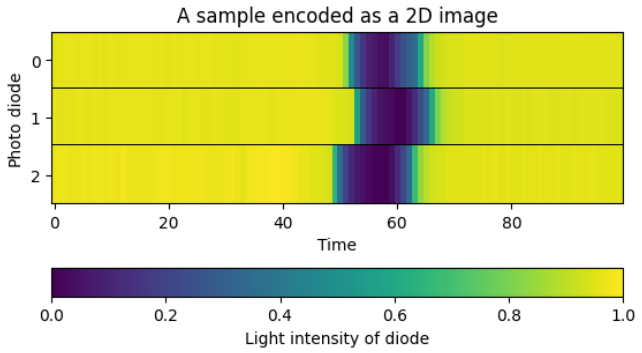

Figure 3: Example gesture, a hand "right-swipe" gesture, encoded as a 2D image. Light intensity data is normalized between 0 and 1.

The paper written by W. Narchi [11] describes the use of convolutional neural networks. This approach encodes every sample as an image, where all the data points of that sample are encoded as a color, as shown in figure 3.

The paper describes why the final model was chosen. The final chosen model (which the author called "Narrow LilConv Padding Pyramid (NLCPP)") consists of a ZeroPadding2D layer, convolutional layers, a 2DMaxPooling layer, another convolutional layer, a flatten layer and finally a 'soft-max' dense layer. This paper included a lot of useful background and explanations of the basics of deep learning.

3 Methodology

In this section, the workflow for this project is described. Figure 4 shows this workflow. To create a model for digit recognition, we first collected a data set. Then we experimented with models and converted them with TensorFlow Lite for Microcontrollers.

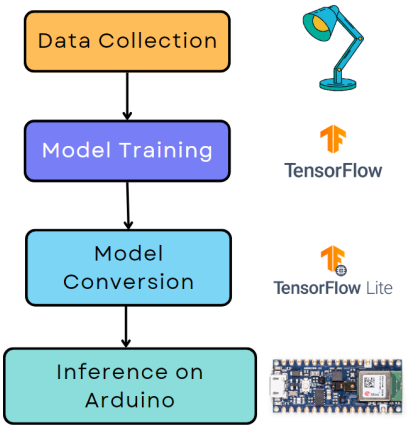

Figure 4: Overview of the process of this research project

3.1 Data collection

There were two data sets created for this project. For both of the data sets the data is strictly anonymous, and the consent forms of each of the participants will not be made public. Figure 5 shows an example plot of what collected gesture data looks like.

For data collection of the first data set, we used a tool that was built last year for this project[2], which was changed into an improved version [17] as a collaborative effort by the Research Project group, group 46, consisting of Arne de Beer, Sem van den Broek, Winstijn Smit, Paco Pronk and the author of this paper, Gijs van de Linde.

The same GitHub repository [17] contains the data that was collected using this tool. This data set was contributed to by Arne de Beer, Sem van den Broek, Winstijn Smit, and the author of this paper, Gijs van de Linde. This data set will be referred to as the **uncontrolled lights dataset**, because the light conditions weren't controlled at all.

Controlled data set

The previously mentioned collaboratively improved tool for data collection was further changed to meet the specific demands of this project, of which the code is available on GitHub [18]. Credit for some concepts used for collecting data based on the threshold detection algorithm needs to be given to Winstijn Smit, who was happy to discuss and explain his ideas.

The second data set that was created was created by Winstijn Smit and Gijs van de Linde. This one did have a controlled light environment, as described in section 4. It will therefore be referred to as the **controlled lights data set**. This data set contains approximately twenty participants. Twenty was chosen because it is a large amount enough amount to cause variations in the exact motion of each

[2]https://github.com/StijnW66/CSE3000-Gesture-Recognition

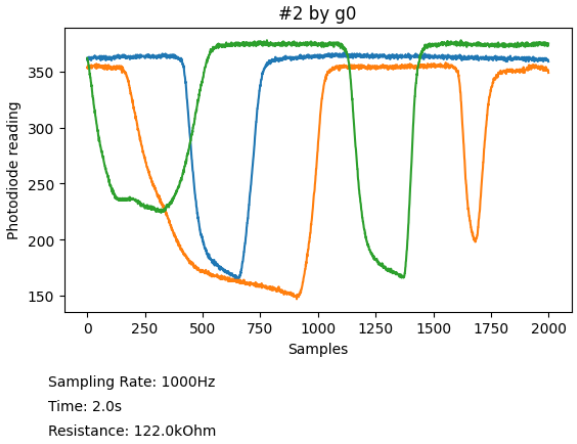

Figure 5: Plot of a recording of the gesture for the digit 2 by candidate "g0" .

gesture, without having to collect an unreasonable amount of different participants. Each candidate did a minimum of 10 recorded gestures of the same digit. Each gesture consisted of a 2-second sample at 1000 Hz, resulting in the final sample containing 2000 data points per photodiode. An example of a gesture is visualized in the plot shown in figure 5.

Figure 6: Setup with lamp used for data collection.

The setup that was used for this data collection procedure is shown in figure 6. This lamp is a 'selfie ring light' from HEMA [19], set to it's 'normal light' mode. The lamp's purpose was to give a consistent light source. This lamp was placed approximately 30 cm above the PCB during the experiment. This caused the photodiode collaboration algorithm described in section 5.1 of the paper on the design of the PCB by S. van de Water [8] to calibrate to 122000 Ohm.

During data collection, we aimed to make a representative data set, considering an approximately equal amount of women and men of different ages. There is a bias in the data set since it is exclusively made up of right-handed people.

To make sure we created a representative data set for the setup it used, we used the threshold-detection algorithm intended for the final device, as described in section 3.3. This makes sure that the sample collected for the data set is similar to the sample used for inference on the final device. It also made collecting data a lot easier, as the participants could start drawing a new digit whenever they felt like it. The changing of the lights on the Arduino board during a 2-second recording made this interaction organic. To remove some complexity from our model, we instructed participants to draw a digit in a certain way. This is shown in figure 7. Another measure we took to make the data less complex, was that for each instruction we had the candidate start under the lower right corner of the Arduino, and after finishing their gesture we had them return to the same position.

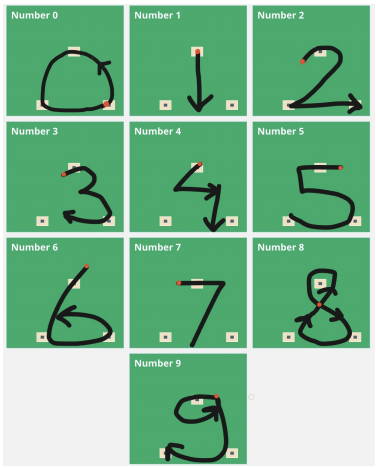

Figure 7: Map that shows how participants had to draw the digits.

3.2 Model choice

For training different models we used Keras[20], the high-level API for TensorFlow. Keras allows for creating and training complex deep learning models, without having to dive into the nitty-gritty of it.

Two main factors influenced our choice of types of neural networks that we considered. First off, the fact that this research is based on 3D pre-processed data, as described in section 2, influenced our choice since this promotes the use of recurrent neural networks. Recurrent neural networks are often used for the classification of time series[21]. Secondly, we needed to convert our model such that it would run on our target device. We, therefore, are required to consider what models are supported by TensorFlow Lite for Microcontrollers. The size restriction of the model comes from the size of the RAM of the microcontroller we use, which is the Arduino Nano 33 BLE Sense. The data sheet of this microcontroller [13] tells us we have 256 KB RAM.

Under these restrictions, there were two evaluation metrics that we used for finding the best deep-learning model.

First off, we did **K-Fold cross-validation** with K=5, which results in an **average accuracy** and **standard deviation**. Ideally, this accuracy would be as high as possible. For each model, a confusion matrix was generated based on the performance of the test set on the model for each fold. These were collected over each fold of the evaluation procedure. Finally, they were compiled into a confusion matrix using the ConfusionMatrixDisplay class[22] from the scikit-learn module. The second metric that is considered is **the size of the model** converted to a TensorFlow Lite version after 200 training epochs. This is representative of how large a model would be on the Arduino, though it's not the theoretical minimum size, since it can potentially be shrunk using quantization. This is further described in the future work section, section 7. The results of these experiments can be found in section 4.2.

A key consideration in data preparation for training models was that it is important to differentiate between two ways of splitting the collected data into testing and training data. The first way to do this split would be fully random, which will be referred to as *within-candidate-split*. The second way is to do a split that groups all data by the candidate that produced the data. This means that a percentage of the candidates is used as test data and a percentage is used for the test data. This will be referred to as *between-candidate-split*. The *Between-candidate-split* yields very different results from a *within-candidate-split*. It was chosen to focus on the *between-candidate-split* during this paper. This choice was made because using this split is closer to a real-world application that requires the recognition of gestures from people that were not included in the process of gathering training data.

To help reduce overfitting a dropout layer was added to certain layers of the models. A paper by N. Srivastava et al.[23] shows that dropout layers can help reduce overfitting of models. The parameters that were chosen for the final experiments are described in section 4.

3.3 Gesture Detection

For detecting the start of a gesture, we use a threshold detection algorithm. This algorithm consists of three circular buffers of size 100. These buffers are updated with a frequency of 1000Hz. Before the buffers are updated there is a check to see if the current photodiode value has a difference certain between the oldest value in the buffer. The **threshold** for this change depends on the frequency used and the lighting conditions. The value that was chosen for this threshold was the integer 10. This was not extensively researched but was chosen since it proved to work.

3.4 Deployment onto Arduino Nano 33 BLE

Though not further researched in this paper, the idea behind these models was to run them on the Arduino Nano 33 BLE itself. This can be done using the TensorFlow Lite for Microcontrollers [24], which is available on GitHub [25]. This is further touched upon in section 7, which contains suggestions for future work.

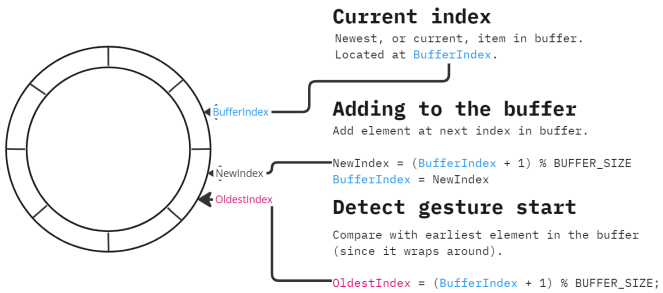

Figure 8: Explanation of circular buffer

4 Experimental Setup and Results

Switch to controlled lighting

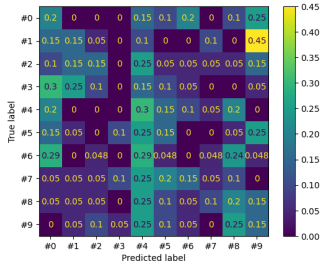

Figure 9: Confusion Matrix for using a ConvLSTM model trained on the uncontrolled data set.

Initially, we set out to attempt data collection without controlling the lighting conditions. The data we collected seemed too varied to create a working model for. This can be seen in figure 9, which shows the result of a 200-epoch training procedure of a ConvLSTM model. This confusion matrix shows that this model performs very badly. It performed with an average accuracy of 0.14 on a 10-class classification problem, which is close to a fully random guessing approach.

To get a foundation for our ambient-light-based digit-classification task we chose to switch to a controlled lighting environment. This narrows down the research since the final results will no longer be applicable to ambient light sources. The data set created for this narrower task is described in section 3.1.

4.1 Experimental Setup

The data was preprocessed before it was used for training. The data was first down-sampled by using interpolation to a 100Hz frequency. This resulted in 200 data points per photodiode per sample. This data was then normalized to be in the range $(0, 1)$. Depending on the model it was then divided into frames, as described in section 2.1. It was chosen to fix the frame size to 5 since that proved to work best in the paper by M. Lipski [12]. It was also chosen to use an Adam optimizer with the learning rate set to 0.0005.

The following deep-learning model configurations were evaluated:

Figure 10: The configurations for each model that were tested

CNN
Parameters: k kernels
1. Time-Distributed Reshape to shape (5, 3, 1) of each time frame.
2. Time-Distributed Conv2D-layer with k kernels of shape (2, 2) followed by a Dropout layer with $p = 0.25$
3. Another Time-Distributed Conv2D-layer with k kernels of shape (2, 2) followed by a Dropout layer with $p = 0.25$
4. A Dropout layer with $p = 0.5$, a Flatten-Layer and a soft-max-activated Dense layer with 10 units, one for each output class.

RNN
Parameters: u units
1. A Time-Distributed Flatten layer to flatten each time frame.
2. A SimpleRNN-layer with u units, followed by a Dropout layer with $p = 0.25$.
3. a Flatten-Layer and a soft-max-activated Dense layer with 10 units, one for each output class.

GRU
Parameters: u units
1. A Time-Distributed Flatten layer to flatten each time frame.
2. A GRU-layer with u units, followed by a Dropout layer with $p = 0.5$.
3. A soft-max-activated Dense layer with 10 units, one for each output class.

LSTM
Parameters: u units
1. A Time-Distributed Flatten layer to flatten each time frame.
2. An LSTM-layer with u units, followed by a Dropout layer with $p = 0.5$.
3. A soft-max-activated Dense layer with 10 units, one for each output class.

ConvLSTM single conv. layer (3, 1) kernel
Parameters: u units
1. A Time-Distributed Reshape to (5, 3, 1) of each time frame.
2. Time-Distributed 'relu'-activated Conv2D-layer with 128 kernels of shape (3, 1) followed by a Dropout layer with $p = 0.25$
3. A Reshape to the shape (40, 3 * 3 * 128).
4. An LSTM layer with u units.
5. A Dropout layer with $p = 0.5$, a Flatten-Layer and a soft-max-activated Dense layer with 10 units, one for each output class.

ConvLSTM single conv. layer (2, 2) kernel
Parameters: u units
1. A Time-Distributed Reshape to (5, 3, 1) of each time frame.
2. Time-Distributed 'relu'-activated Conv2D-layer with 128 kernels of shape (2, 2) followed by a Dropout layer with $p = 0.25$
3. A Reshape to the shape (40, 4 * 2 * 128).
4. An LSTM layer with u units.
5. A Dropout layer with $p = 0.5$, a Flatten-Layer and a soft-max-activated Dense layer with 10 units, one for each output class.

ConvLSTM double conv. layer
Parameters: u units
1. A Time-Distributed Reshape to (5, 3, 1) of each time frame.
2. A Time-Distributed 'relu'-activated Conv2D-layer with 128 kernels of shape (3, 1) followed by a Dropout layer with $p = 0.25$
3. A Time-Distributed 'relu'-activated Conv2D-layer with 128 kernels of shape (2, 2).
4. A Reshape to the shape (40, 2 * 2 * 128).
5. An LSTM layer with u units.
6. A Dropout layer with $p = 0.5$, a Flatten-Layer and a soft-max-activated Dense layer with 10 units, one for each output class.

ConvLSTM max pooling
Parameters: u units
1. A Time-Distributed Reshape to (5, 3, 1) of each time frame.
2. Time-Distributed 'relu'-activated Conv2D-layer with 128 kernels of shape (2, 2) followed by a Dropout layer with $p = 0.25$
3. Time-Distributed MaxPooling2D-layer followed by a Dropout layer with $p = 0.25$
4. A Reshape to the shape (40, 2 * 1 * 128).
5. An LSTM layer with u units.
6. A Dropout layer with $p = 0.5$, a Flatten-Layer and a soft-max-activated Dense layer with 10 units, one for each output class.

- A CNN model
- An RNN model
- A GRU model
- A LSTM model
- Different ConvLSTM configurations

The exact setups used for experimentation are described in depth in figure 10.

4.2 Results

The results from the experiments have been compiled into a single table, table 11.

The results of the 5-split KFold evaluation of the models can be seen in the "accuracy" and the "SD" columns of table 11, which represent the mean accuracy and standard deviation over the 5 splits for each model for the 5-Fold cross-validation evaluation.

Figure 11: Results of 5 split KFold, 200 epochs, rounded to the 3rd decimal. Includes size of model on 200 epochs of training after conversion to a TensorFlow Lite model without quantization.

CNN model				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 kernels	0.383	0.040	3.620	49.016 KB
64 kernels	0.395	0.039	4.545	101.64 KB
128 kernels	0.405	0.038	6.637	231.456 KB
RNN model				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 units	0.3124	0.041	2.218	13.352 KB
64 units	0.3704	0.048	3.115	17.656 KB
128 units	0.369	0.077	3.901	31.8 KB
GRU model				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 units	0.356	0.036	2.414	18.304 KB
64 units	0.416	0.098	3.192	30.072 KB
128 units	0.442	0.121	3.676	71.992 KB
LSTM model				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 units	0.373	0.0794	2.459	19.216 KB
64 units	0.408	0.086	3.247	33.888 KB
128 units	0.413	0.098	3.380	87.776 KB
ConvLSTM single conv. layer (3, 1) kernel				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 units	0.442	0.108	2.758	171.000 KB
64 units	0.423	0.800	3.523	331.192 KB
128 units	0.491	0.073	3.280	676.152 KB
ConvLSTM single conv. layer (2, 2) kernel				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 units	0.424	0.086	2.706	154.752 KB
64 units	0.445	0.070	3.451	298.56 KB
128 units	0.463	0.082	3.49	610.752 KB
ConvLSTM double conv. layer				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 units	0.445	0.077	2.873	159.352 KB
64 units	0.468	0.081	3.127	237.624 KB
128 units	0.500	0.091	3.158	418.744 KB
ConvLSTM max pooling				
Parameters	*Accuracy*	*SD*	*Loss*	*Size*
32 units	0.385	0.091	2.305	57.008 KB
64 units	0.417	0.097	2.952	102.584 KB
128 units	0.429	0.125	3.318	218.168 KB

Generally, most recurrent neural networks perform better with more memory blocks (units) as seen from the GRU, LSTM and ConvLSTM models. With the standard deviation in mind, the models that perform the best are the ConvLSTM with a single convolutional layer that uses a (3,1) shaped kernel and the ConvLSTM model with two convolutional layers of which one uses a (3,1) shaped kernel and one uses

a (2, 2) shape. On the 5-fold cross-validation evaluation, the single-convolution model had an accuracy of 0.491 ± 0.073, whereas the model using two convolutional layers had an accuracy of 0.500 ± 0.091. The confusion matrices of these two best-performing models can be seen in figure 12 and figure 13. These show that both models tend to have a hard time distinguishing the following pairs of digits: 2 and 3, 3 and 7, and finally 5 and 9. This seems like an explainable result since these three digit combinations follow a similar path over the three observing photodiodes when performing their associated gesture. The results of the conversion of the models to TensorFlow Lite models can be seen in the "size" column of table 11.

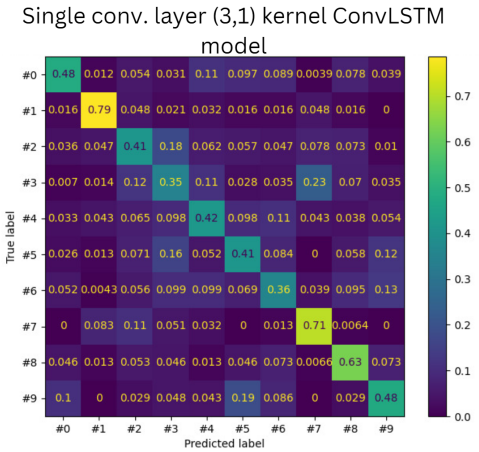

Figure 12: The confusion matrices of the best-performing ConvLSTM model with a single convolutional layer with a (3,1)-shaped kernel.

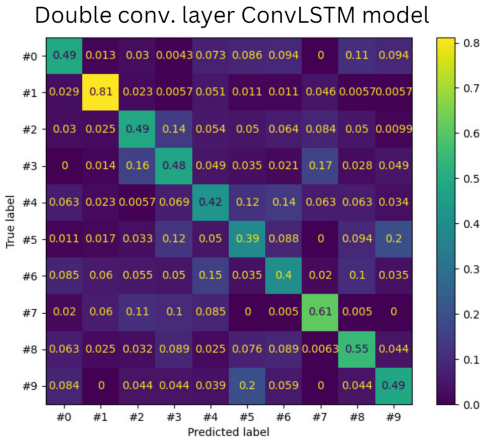

Figure 13: The confusion matrices of the best-performing ConvLSTM model with two convolutional layers, one with a (3,1)-shaped kernel followed by a (2,2)-shaped kernel.

Interestingly, after conversion the single conv. layer models become scaled a lot faster than the double conv. model with the units provided to the LSTM layer of the model. This can be explained due to the fact that for each

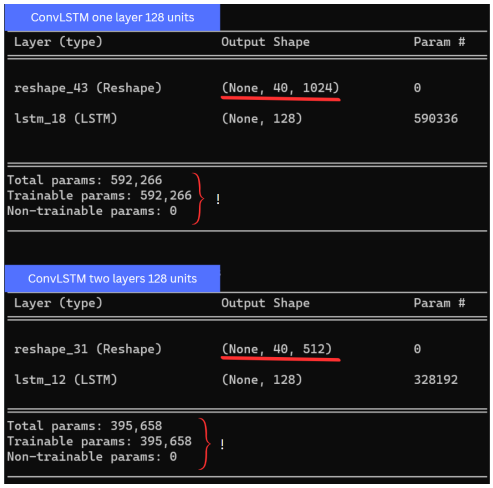

Figure 14: This explains why the ConvLSTM with two conv. layers is smaller after conversion than the one with a single conv. layer.

of these one conv. layer models the LSTM layer takes an input shape that's two times larger than the one for the two conv. layer models. This is visualized in figure 14, where it can be seen that the single conv. model clearly has a larger shape for the data that goes into the LSTM layer. This is also reflected in the fact that the single conv. layer model has a total of $592,266$ trainable parameters, where the two conv. layer model has considerably less, namely only $395,658$.

5 Responsible Research

In this responsible research section, we will discuss the reproducibility of this research. It will also reflect on the ethical aspects of this research.

In the methodology and experimental setup sections, an effort has been made to describe the process of data collection and model training in detail.

For the controlled data set data samples were not thrown out unless a participant signaled that something went wrong. If one gesture on a candidate had more samples than a different gesture on any candidate, then the smallest number was taken, and if there were more samples available a random sample of the samples was taken. This causes an element of randomness, and the validation and test splits are also sources of randomness in these experiments, which means the exact results won't easily be obtained, though similar results are expected. The code for this project is available on GitHub[18], which makes it easier to reproduce the results.

Because this research uses data that is collected from people, there is a responsibility to do this in a manner that is ethically responsible and to inform the participant of what they're participating in. To make sure of this we had them sign a consent form that was passed through the HREC (Human Research Ethics Committee). This form describes all the implications of participation. This form was created by our supervisor, Qing Wang, and is available on Google Drive [26].

6 Discussion

This section discusses the results, and what they mean. It also discusses what could've been better about this paper. An important fact to notice is that the lighting conditions were fixed to bright artificial light. This means the results are not generalizable to "ambient lighting". Using ambient lighting proved really challenging, as described in section 4. This needs to be realized when interpreting the results or using the conclusions.

A factor that should be considered in the uncontrolled data set is that each collector of the data independently made a selection of the samples they collected. The workflow allowed for this with the single press of the 'd' button after looking at a collected sample plot. The fact that this was done independently between four collectors should be highlighted since this does mean that the bias in this data should be representative of normal biases in the data collection of this data set.

It is important to mention that the **controlled lights** data set contains only right-handed data from right-handed participants. This is a bias in the model, but since we are researching how feasible a digit detection algorithm is this is not that much of a problem, especially since the vast majority of people are right-handed. The final product, as well as further research, needs to consider left-handed data from left-handed people.

Something else that needs to be noticed about the data samples is that when doing gesture detection, as described in section 3.3, the values in the circular buffer are not recorded when a gesture is detected. A different approach would be to first capture the values in the circular buffer, and then append all the data collected after that, causing the sample to contain all the data from the exact start of the gesture, maybe even a little bit before it. This is just a choice since the values in the buffer usually consist of the start of a gesture, which means that a participant moves their hand to the desired starting position. It's also worth noting that the amount of time lost for a gesture would be the first 100/1000 = 0.1 seconds.

Something to discuss about the way the participants were instructed to draw a digit is that the digit 8 could be drawn in two ways according to its representation in figure 7. This adds a bit of extra complexity to the model, but such confusion could only make the model perform worse, so it doesn't invalidate any of the results.

Section 3.2 describes that the way of splitting the data into training and test data is very important for the results. It is important when interpreting the results that they were gathered using a *between-candidate-split*. Potentially a higher mean accuracy and a better-looking confusion matrix could have been found using a *within-candidate-split* since this would evaluate the test metrics on data the model had already seen. This fortifies our results though, since the evaluation metrics are based on testing the model with data from completely new, previously unseen candidates.

Furthermore, there was no evaluation done of models after conversion to TensorFlow Lite models. There will most likely be a decrease in performance after conversion since certain optimizations cause trade-offs to be made. This needs to be kept in mind when interpreting the results.

7 Conclusions and Future Work

Conclusion

A conclusion we can draw from this research is that digit detection using ambient light and only three photodiodes is challenging. By adding a constant, bright, artificial light we researched the feasibility of creating a consistent model for recognizing digits. The results from our experiments were decent, and many improvements could be done to increase performance. Suggestions for this are described in the next section, the future work section.

From the model evaluation based on the 5-split KFold cross-validation procedure, the following conclusions can be drawn: According to the results in figure 11 best model is either the "ConvLSTM single conv. layer with a horizontal ((3,1)) kernel" or "ConvLSTM double conv. layer". These performed the best during the experiment, with the single-convolution model getting a mean accuracy of 0.491 ± 0.073 and the model using two convolutional layers getting a mean accuracy of 0.500 ± 0.091.

Of these two, figure 11 tells us that the "ConvLSTM double conv. layer with 128 LSTM units" has a size of 418.744 KB, which is roughly 62% the size of the "ConvLSTM single conv. layer with a horizontal ((3,1)) kernel with LSTM 128 units" model which is 676.152 KB large.

In conclusion, while the current research provides a foundation for digit detection using time-series data in a controlled light environment, there is significant potential for future improvements and expansion in this area.

Future Work

This section describes guidance and suggestions for future work. Something that needs to be further explored in order to deploy models on the Arduino is the conversion to 'TensorFlow Lite for Microcontrollers' models using quantization, which is described in section 2.1, and quantization-aware training [27] of the models. This will make the models smaller in size and likely faster in performance. After that, it would be useful to do inference testing and evaluation of the models using the quantized model either on a PC or when running it directly on the Arduino itself.

The usual workflow for the development of models onto a micro-controller, as described on the website [24], is this:

1. Train a model:
    - Generate a small TensorFlow model that can fit your target device and contains supported operations.
    - Convert to a TensorFlow Lite model using the TensorFlow Lite converter.
    - Convert to a C byte array using standard tools to store it in a read-only program memory on the device.
2. Run inference on the device using the C++ library and process the results.

There are a lot of possible techniques for improving the performance of deep learning models. One that is especially applicable to this project is data augmentation. Because of the limited data and the time-dependent nature of the data, this could help improve the accuracy of our models greatly.

A tool that seems helpful for finding good parameters for the models is Keras Tuner. A small example of this is written out in a Google Colaberatory notebook here: http://bit.ly/keras-tuner-example. Please note that directly running this will not work, and it merely serves as an example.

The learning rate of the models and the frame size for the data were fixed to a single value during this research. It is probable that varying these can grant valuable increases in performance.

The idea behind this research was to do real-time classification. For this, we chose a maximum duration of 200ms. The inference times of our models were not evaluated, and need to be further researched.

For detecting the start of a gesture, we implemented a C++ function that starts inference whenever the values of one of the three photodiodes become less than a certain threshold. This threshold was chosen because it seemed to work, but was not extensively researched. This could still be valuable to research.

References

[1] E. Goldman. "Exaggerated risk of transmission of COVID-19 by fomites". **in***The Lancet Infectious Diseases*: 20.8 (2020), **pages** 892–893. DOI: https://doi.org/10.1016/S1473-3099(20)30561-2.

[2] Ellie Violet Bramley. *Therapists report huge rise in cases of anxiety as England ends Covid rules.* 2021. URL: https://www.theguardian.com/world/2021/jul/11/therapists-report-huge-rise-in-cases-of-anxiety-as-england-ends-covid-rules.

[3] London South Bank University. *The pandemic's mental toll: new survey finds one in five suffer from Covid-19 Anxiety Syndrome.* 2021. URL: https://www.lsbu.ac.uk/about-us/news/the-pandemics-mental-toll-new-survey-finds-one-in-five-suffer-from-covid-19-anxiety-syndrome.

[4] Muhammad Zahid Iqbal **and** Abraham G. Campbell. "From luxury to necessity: Progress of touchless interaction technology". **in***Technology in Society*: 67 (2021), **page** 101796. ISSN: 0160-791X. DOI: https://doi.org/10.1016/j.techsoc.2021.101796. URL: https://www.sciencedirect.com/science/article/pii/S0160791X21002712.

[5] *Mobile Fact Sheet*. URL: https://www.pewresearch.org/internet/fact-sheet/mobile/.

[6] Kamil Janczyk, Krzysztof Czuszynski **and** Jacek Ruminski. "Digits Recognition with Quadrant Photodiode and Convolutional Neural Network". **in***2018 11th International Conference on Human System Interaction (HSI)*: 2018, **pages** 111–117. DOI: 10.1109/HSI.2018.8431246.

[7] Y. LeCun **and** C. Cortes. "The MNIST database of handwritten digits". **in**(1998): URL: http://yann.lecun.com/exdb/mnist/.

[8] S. van de Water. *Designing an adaptable and low-cost system for gesture recognition using visible light*. 2022.

[9] D. Barantiev. *Designing a Software Receiver for Gesture Recognition with Ambient Light*. 2022.

[10] F. Akadiri. *Constructing A Dataset For Gesture Recognition Using Ambient Light*. 2022.

[11] W. Narchi. *Recognising Gestures Using Ambient Light and Convolutional Neural Networks*. 2022.

[12] M. Lipski. *Hand Gesture Recognition on Arduino Using Recurrent Neural Networks and Ambient Light*. 2022.

[13] *Arduino Nano 33 BLE Sense datasheet*. URL: https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf.

[14] *TensorFlow Lite Documentation: Model Optimization*. URL: https://www.tensorflow.org/lite/performance/model_optimization.

[15] *TensorFlow Lite Documentation: Post-training quantization*. URL: https://www.tensorflow.org/lite/performance/post_training_quantization.

[16] Christoph Siegl. *TensorFlow Lite for Microcontrollers adds Support for Efficient LSTM Implementation*. URL: https://medium.com/@christoph-siegl/tensorflow-lite-for-microcontrollers-adds-support-for-efficient-lstm-implementation-25a5f7baa4f6.

[17] A. de Beer **andothers**. *Data Collection*. 2023. URL: https://github.com/charlespwd/project-title.

[18] G. van de Linde. *Photo-diode Time-series Digit Recognition*. 2023. URL: https://github.com/gvandelinde/photo-diode-digit-recognition-research-project.

[19] *Hema webstore: selfie ring licht Ø8.8cm*. URL: https://bit.ly/hema-ring-light.

[20] *Keras*. URL: https://www.tensorflow.org/guide/keras/sequential_model.

[21] Michael Hüsken **and** Peter Stagge. "Recurrent neural networks for time series classification". **in***Neurocomputing*: 50 (2003), **pages** 223–235. ISSN: 0925-2312. DOI: https://doi.org/10.1016/S0925-2312(01)00706-8. URL: https://www.sciencedirect.com/science/article/pii/S0925231201007068.

[22] URL: https://scikit-.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html.

[23] Nitish Srivastava **andothers**. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". **in***Journal of Machine Learning Research*: 15.56 (2014), **pages** 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[24] *TensorFlow Lite for Microcontrollers*. URL: https://www.tensorflow.org/lite/microcontrollers.

[25] *TensorFlow Lite for Microcontrollers*. URL: https://github.com/tensorflow/tflite-micro.

[26] Qing Wang. *HREC approved consent form for this research*. 2023. URL: https://drive.google.com/file/d/1_nrkuTcqVgujbp8T3AHVCoGTEfVJRrOZ/view.

[27] *TensorFlow Lite Documentation: Quantization aware training*. URL: https://www.tensorflow.org/model_optimization/guide/quantization/training.